

New challenges in digital forensics: online storage and anonymous communication

PhD THESIS

submitted in partial fulfillment of the requirements for the degree of

Doctor of Technical Sciences

by

Martin Mulazzani

Registration Number 0225055

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl

The dissertation has been reviewed by:

(Univ.-Prof. Dipl.-Math. Dr.
Stefanie Rinderle-Ma)

(Univ.-Prof. Dipl.-Ing. DDr.
Gerald Quirchmayr)

Vienna, 30.01.2014

(Martin Mulazzani)

Declaration of Authorship

Martin Mulazzani
Kreutzergasse 5, 3400 Klosterneuburg

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

(Vienna, 30.01.2014)

(Martin Mulazzani)

Acknowledgements

I'm very grateful to my advisor Edgar Weippl, who supervised me throughout my studies and always supported me in my own research ideas. Even in the busiest times, feedback and support were always available and the discussion of ideas helped me a lot. I would also like to thank all my colleagues at SBA Research - in particular Markus Huber, Sebastian Schrittwieser and Sebastian Neuner, whom I have the honor to work with on a regular basis. Besides discussing ideas and working together numerous late-nights close to paper deadlines, colleagues like them cannot be taken for granted and are among the reasons why I enjoyed this period in my life as much as I did. I'm also thankful for the all the opportunities I received at SBA Research during these years, in particular teaching skillful students and working with them to develop their own research ideas. I'm also grateful that I did get the opportunity to study and work two semesters at Purdue University, and I would like to thank Prof. Elisa Bertino as well as Prof. Christina Nita-Rotaru for their supervision and mentoring. Furthermore, this work would not have been possible without funding from COMET K1 and project number 825747 (INFORM) by the FFG Austrian Research Agency.

I'm grateful for the support from my family, my parents Eva and Marco Mulazzani as well as my brothers Matthias, Max and Michael, and above all Kathi and our wonderful children Ferdinand and Liselotte for their continuous love and help during the creation of my thesis. Without you I would not be where I am today.

Martin Mulazzani
Vienna, 30.01.2014

Abstract

This thesis is based on seven publications related to the area of digital forensics which were published at conferences or in journals by ACM, IEEE, USENIX and IFIP. Digital forensics as research field has received increasing attention in recent years, as more and more crimes are committed exclusively or with the involvement of computers. At the same time, new challenges emerge constantly, e.g. the prevalent use of encryption, mobile devices of various nature, online cloud storage services and readily available tools that facilitate counter-forensics. In particular, this thesis tries to mitigate current challenges for digital forensics in the areas of online data storage and anonymous communication.

Regarding anonymous communication we analyzed the well-known online anonymity tool Tor, which employs onion routing and is expected to be used by hundreds of thousands of users every day: firstly how it is used, and secondly what can be learnt from the publicly available server information. We were able to show that the majority of users are not employing Tor as recommended by the Tor community, and we found many information leaks that can endanger the users' anonymity. We also studied how the underlying infrastructure, which is run by volunteers, can be monitored to provide useful metrics of interest. We furthermore derived and implemented a new attack on online storage systems abusing client-side data deduplication and analyzed how it can be used to thwart digital forensic investigations which in turn can be used for forensic investigations. We showed its feasibility on Dropbox, one of the largest cloud storage providers with more than 200 million users worldwide at the time of writing this thesis. We quantified slack space on numerous Windows systems, assessed its stability over time regarding system updates and found that up to 100 megabytes of slack space are readily available in files of the operating system. We furthermore implemented a digital alibi framework with a social interaction component which in our opinion can be easily overlooked in forensic analysis as conducted today. Finally we analyzed browser artifacts and how they can be used for browser fingerprinting. We then used browser fingerprinting to enhance HTTP session security by binding the session on the server to specifics of the particular browser used.

Kurzfassung

Diese Dissertation baut auf sieben Arbeiten auf, die auf Konferenzen und in Journalen von ACM, IEEE, USENIX und der IFIP veröffentlicht wurden. Digitale Forensik als Forschungsdisziplin hat sich in den letzten Jahren mehr und mehr etabliert, da kriminelle Handlungen mittlerweile ausschließlich mit oder unter Zuhilfenahme von Computern begangen werden. Gleichzeitig werden durch die Verbreitung von starker Verschlüsselung, einer Vielzahl an neuen mobilen Geräten und das stetig steigende Datenvolumen neue Herausforderungen an die digitale Forensik gestellt. Diese Arbeit beschäftigt sich im Speziellen mit den Problemen der digitalen Forensik hinsichtlich Speicherdienste im Internet und anonymer Kommunikation.

Im Bereich der anonymen Kommunikation untersucht diese Arbeit Tor, ein sehr weit verbreiteter Anonymisierungsdienst im Internet. Es konnte belegt werden, dass Tor meist nicht wie empfohlen verwendet wird und die Gefahr einer kompletten (unbeabsichtigten) Deanonymisierung für die Anwender hoch ist. Wir haben verschiedene Metriken für die dem Tor Netzwerk zugrundeliegende Infrastruktur mit derzeit ca. 5.000 Knoten erstellt, da diese von Freiwilligen betrieben und nicht zentral kontrolliert wird. Im Bereich der digitalen Forensik haben wir eine neue Angriffsmethode auf Internet-Speicherdienste entwickelt und implementiert. Dieser Angriff nützt die Datenduplizierung auf der Anwenderseite aus, um einen Angreifer unberechtigten Zugriff auf Daten zu ermöglichen. Die Anwendbarkeit unseres Angriffs wurde anhand von Dropbox belegt, einem der größten Speicherdienste mit derzeit mehr als 200 Millionen Anwendern. Wir haben weiters die Gesamtspeicherkapazität von Fragmentierungsartefakten ("slack space") von Microsoft Windows vermessen und über einen längeren Zeitraum die Stabilität in Bezug auf Systemupdates ermittelt. Zusätzlich haben wir ein Framework implementiert, das die Erzeugung eines digitalen Alibis ermöglicht. Unser Ansatz beinhaltet eine soziale Kommunikationskomponente, die eine forensische Untersuchung täuschen könnte. Im Bereich der sicheren Online-Kommunikation haben wir Webbrowser untersucht und neuartige Identifizierungsmöglichkeiten entdeckt. Auf diesen Ergebnissen aufbauend erhöhten wir die Sicherheit von Online-Sitzungen, indem die Sitzung Server-seitig an die Charakteristika des Browsers gebunden wird.

Contents

Introduction	1
Background	3
Problem Description	5
Proposed Solutions	9
Goals	9
Methodology	11
Scientific contributions	15
Digital Forensics on Online Storage	16
Insights into Anonymous Communication Methods	18
Contributions to Traditional Forensic Techniques	19
Conclusion	23
Overview of Research Contribution	25
Bibliography	27
Using Cloud Storage as Attack Vector and Online Slack Space	37
Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting	51
SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting	63
How to Monitor the Infrastructure of an Anonymity System	77
Tor HTTP Usage and Information Leakage	95
Quantifying Windows File Slack Size and Stability	109
Towards Fully Automated Digital Alibis with Social Interaction	125

Introduction

The Internet as it is used today has become broadly fragmented regarding how it is used and with a multitude of different software protocols. Browsing the web for information, watching videos and listening to music or communicating using tools like e-mail, instant messaging or social platforms like Twitter or Facebook are nowadays one of the core use-cases for millions of people around the world. The “dark side” of the Internet can be observed in everyday news. Spam, 0day exploits, underground marketplaces, identity theft and many other problems and attack vectors as well as exploitation techniques are used nowadays to conduct mischief on a daily basis. Users are furthermore lured into giving more and more of their private data to companies that use them to generate revenue, although many of these companies struggle to adequately protect their users’ data from a technical point of view. Password breaches with millions of affected users have become mainstream: RockYou lost 32 million passwords in 2009, Adobe lost 150 million passwords in 2013 [74] and up to 40 million customers’ credit card information has been stolen in late 2013 [10]. When such incidents occur, methods of digital forensics are used by law enforcement, forensic investigators and system administrators to reconstruct the timeline of events, identify possible traces left behind and identify the impact of breaches. Digital forensics, as defined by NIST, is the application of science to the law, in particular “*the identification, collection, examination, and analysis of data while preserving the integrity of the information and maintaining a strict chain of custody for the data*” [71]. The tools and methods used are definitely not new and have been employed for decades [117]. However, digital forensics has recently manifested as a subfield of computer science, and curricula are set up all over the world to allow students to study in this field.

Digital forensics have received increasing attention in recent years as more and more crimes are committed exclusively or with the involvement of computers. Digital traces help courts and law enforcement agencies to capture valuable evidence. Existing research as well as applications in the area of digital forensics focus on file systems, log files, network traffic, databases and, more recently, mobile devices like smartphones and tablets [61, 84]. The traces that are left on mobile devices in particular can contain a plethora of information, e.g. pinpoint past communications as well as exact locations due to the use of GPS and wireless networks, regardless of the exact communication channel used (GSM, instant messaging or e-mail, just to name a few). The traditional approach of digital forensics is to acquire data from a device in a forensic lab using special hard- and software. The current approach of analyzing local, residual artifacts has however two major shortcomings: for one, not only the seizure of a suspect’s device is a required, but it has also to be accessible to the investigators. Secondly, and even more importantly,

the emergence of new online services extend the traditional means of data storage, information sharing and communication. These services often do not operate under the same jurisdiction as the user, which can make it difficult to obtain the data. Law enforcement can ask service operators to release certain information, but they are usually not obliged to answer requests from other countries. However, there have been documented cases in Austria where this method was successful [124]. Furthermore, the more prevalent use of encryption can make data acquisition and analysis very hard while counter-forensic tools can be readily obtained from the Internet. As such, digital forensics are often behind the current state of technology.

Background

The forensic process usually starts with the identification and verification of an incident, the acquisition of the devices affected and the analysis in a forensic lab [71]. If the device is found running, the process of data acquisition starts by copying the content of the volatile memory since it contains important dynamic information that is not stored anywhere else. This is specified as the “*order of volatility*”, which dictates that volatile information is a priority during the acquisition process [19]. The content of the RAM is often incredibly useful during investigations as it contains the list of running processes, open network connections, encryption keys etc. Methods that can be used are for example special software or loadable kernel modules, a cold boot attack [58] or modular extension cards that support *direct memory access* (DMA) like Firewire. The acquisition process has to be done carefully, as many approaches modify the content of RAM. If a software-based approach is used (running acquisition software on the machine), this is problematic as it modifies the RAM itself - the process needs to be executed and thus needs space in RAM that can potentially overwrite other information in RAM. The investigator furthermore has to trust the underlying operating system and hardware, as they could be modified to thwart the acquisition process. Hardware-based approaches that rely on DMA can be problematic too, as the Northbridge that executes DMA can be tricked into seeing different memory content as the CPU [113]. This leaves the analyst with the cold boot attack, but it requires an enormous effort compared to the other approaches and as such is expensive and error-prone. While the cold boot attack was originally derived to extract cryptographic keys from memory, it is also suitable to acquire an image of the system memory. Even though it was commonly believed that the content of RAM is deleted once power is shut down - since RAM cells need to be refreshed periodically by the system bus to store information - the cold boot attack showed that RAM content is retrievable even 10 minutes after shutdown (and possibly even more) if the RAM is cooled down [58]. It was shown that this method can even be used to parse the RAM on the fly and reboot the system into a forensically controlled environment without service interruption [25]. Recently, the cold boot attack was extended to be used on mobile devices running Android [95]. The content of RAM is furthermore not reproducible, as the repetition of the exact steps that were taken by a user to arrive in a particular RAM state can lead to a different state due to the operating system, background processes or external influence not under the control of the user, e.g. network connections. A recent paper discusses anti-forensic capabilities for memory acquisition and presents a new acquisition method based on direct page table manipulation and PCI hardware introspection [118].

The system is then abruptly powered down to prevent the execution of shutdown scripts or dead man switches that could possibly modify data. Afterwards the hard drives are copied multiple times using hardware write blockers to prevent unintended data changes and to preserve the integrity of the information [71]. This includes at least a working copy and a backup copy. Every step is fully documented to preserve a full chain of custody [71] and to allow the evidence to be usable in court. Hash values like MD5 [108] or SHA1 [42] are commonly used to guarantee the integrity of the data. Prior to imaging it, a hash value is calculated over the entire hard drive. This hash value is used after imaging to verify that the newly created image contains in fact the same data as the original, but also to prove that the imaging process has not altered any data. Once the working copy has been created, special tools like *EnCase* by Guidance Software¹, *FTK* by AccessData² or the open source tool *The Sleuth Kit*³ are used to assist in the process of analyzing the data, to index the hard drives and process them to extract all possible information of relevance including file system metadata and the data stored in the files itself. However, the fundamental change in the computing environment from PCs and local servers to outsourced services, Web 2.0 applications and cloud computing requires fundamentally new technology regarding digital forensics.

The worst case for a traditional forensic analysis is, at the time of writing, a computer without a hard drive and turned off on arrival. Readily available boot media can be used to start a live Linux distribution, whereas all data is exclusively stored online. Without network traces or a memory dump, there is simply no locally stored information to analyze. Storage providers like Dropbox, Microsoft's SkyDrive or Google Drive offer enough storage capacity to store data online, which can be downloaded to a RAM drive that is in turn deleted upon reboot. This is similar to a turned-off computer with encrypted hard drives - without further knowledge of the key or its copy in RAM, it is infeasible to get the data in clear text. This is for example why the raid on Max Butler (aka Iceman) [103] was accompanied by a team of forensic experts from Carnegie Mellon University to acquire the RAM of the running computers, as it was known to law enforcement that Max Butler was using the software *DriveCrypt* to encrypt his hard drives. It is possible to use encryption without storing the key in RAM, e.g. for RSA [55] or full disk encryption using AES [93]. This has been shown to be feasible for PCs [94] as well as for mobile platforms using ARM [57], the key is stored in a CPU register and never written to RAM. Other artifacts like the list of running processes, open network connections etc. are however unencrypted in RAM.

Network forensics rely (mostly) on the analysis of network captures. Usually this means that the upper network layers are analyzed, in particular the application layer as well as the Transport and Internet layer (according to the TCP layering model [101, 102]). While the application layer contains the actual user communication, including important online protocols like HTTP for web browsing and POP/IMAP for e-mail, the network layer contains the technical information needed for the communication transmission, e.g. IP addresses and port numbers. Both

¹<https://www.encase.com/encase-forensic.htm>

²<http://www.accessdata.com/products/digital-forensics/ftk>

³<http://www.sleuthkit.org/>

are vital in an investigation, given that either can contain relevant information. If encryption at the application layer is used, the network layer still reveals metadata and contains information on who is communicating with whom [30]. Network analyzer software like *wireshark*⁴ can be used to extract information from the network capture on all layers. It can also be used to decrypt encrypted information if the key is known, for example in case of TLS and HTTPS. More advanced (open-source) tools like *Xplico*⁵ can automatically generate statistics and reports of common interest for a multitude of use-cases. *PyFlag* is a tool which is able to reconstruct the target's view in the browser by parsing HTTP traffic [27]. This is especially useful as the entire web session can be vividly presented from the targets point of view to persons not familiar with the technical details of the Internet, e.g. for juries and lawyers, for expert witness presentations in court. Sometimes it is also possible to find data that contains network data structures [14] on hard drives, which can happen if RAM content is swapped to disc and has not yet been overwritten.

Problem Description

The problems of digital forensics are manifold: new devices, operating systems and file systems (e.g. btrfs [109]), in particular currently on mobile devices and smartphones, can make the acquisition of data cumbersome. Cloud computing [11] and the emergence of cloud services render the acquisition of hard drives useless, as vital information is often stored in anonymous data centers around the world - without direct access for law enforcement agencies - and not on local hard drives anymore. Another problem is that it's infeasible to acquire an entire data center, not only logistically but probably also legally. Data formats, operating systems and hardware are often proprietary and custom-built, and existing software solutions would have to be adapted for analysis. Comparably, forensic analysis can already be challenging for large systems like e-mail- or storage servers, as terabytes of data need to be processed and often expensive hardware like special RAID controllers are needed to obtain access to the information. Large service providers like Facebook, Google, Yahoo and others thus comply with law enforcement and extract the data for the requestor, while the agency has to trust the service provider that the methods used are forensically sound and all the relevant data is extracted. The use of encryption can furthermore complicate data acquisition, and legal boundaries are often problematic as the Internet by its very nature is international and knows no boundaries.

Another upcoming problem for digital forensics is the scale of data to analyze and the time required for conducting a forensic analysis: a commodity 4TB hard drive can be bought nowadays for less than US \$200, but it takes many hours to simply create an image for analysis or calculate a hash value over the entire hard drive. The overall amount of information that needs to be processed is increasing exponentially, therefore automated tools with large throughput will become more and more important [52]. *bulk_extractor* [53] by Simson Garfinkel for example is a tool that can search unstructured data and hard drives for e-mail addresses, credit card numbers and more, using recursive scanners. *bulk_extractor* is designed to extensively build upon

⁴<https://www.wireshark.org/>

⁵<http://www.xplico.org/>

multithreading, thus yielding a very high overall performance. An anecdotal story tells that the tool was able to pin a server with 48 CPU cores, thus effectively parallelizing data analysis. Another approach to reduce the manual work for the analyst is the use of white listing hash values of benign files on a hard drive. NIST is quarterly releasing the National Software Reference Library reference data set (NSRL RDS)⁶ which contains more than 110 million hash values for files, executables and software libraries of common operating systems and software products. Another approach that seems promising is the use of sector hashing: instead of hashing files, the sectors of the hard drives are hashed individually. This has been shown to be a promising approach [50, 128], as current use file systems like NTFS or FAT are sector-aligned [21]. To reduce the addressing overhead, NTFS, FAT as well as other file systems logically combine several sectors to clusters which would be another vector for hashing since files are split into clusters on the hard drive. NTFS uses 4 kilobyte clusters as default value for file systems smaller than 16 terabytes, which means that on older hard drives 8 512 byte sectors are combined into a 4 kilobyte cluster. ATA hard drive sectors used to be 512 bytes in size, but newer hard drives are transitioning to 4 kilobytes sectors as the new default value. Even though sector hashing seems promising in terms of accuracy and precision (many files do not share common sectors [128]), one of the limitations of this approach is the size of the hash values and the overhead to query large hash samples. A hard drive with one terabyte capacity has around 250 million 4 kilobyte sectors, resulting in 250 million hash values. Even though the use of GPUs [80] or MapReduce [32] could facilitate large-scale analyses and distributed processing of hash value comparisons, this has not been evaluated regarding performance and accuracy (precision/recall). False-positives and false-negatives could have a severe impact in such large data operations, as they could lead to expensive manual inspections and potential base rate fallacy. Furthermore, multiple different hash window sizes could be beneficial, as different sources of hash values can then be considered as sources: Dropbox uses up to 4 megabyte file chunks for hashing, and many P2P file sharing applications like *Gnutella* or *BitTorrent* use variable hashing windows [77] depending on file size and number of files.

One of the solutions to analyze the ever increasing file numbers and storage capacity are so-called “approximate hash functions”, also known as *fuzzy hashing*. Compared to cryptographic hash functions like MD5 and SHA-1, fuzzy hashing has the benefit that the change of a single bit in the input does not entirely change the resulting hash value. Instead, they are designed to calculate a score value between 0 and 100 on how similar two different files are - 100 if two files are entirely similar, and 0 if no similarity can be found. The benefit is that related files can possibly be identified, as well as previous versions of the same file. It is possible to retrieve files in different versions, for example if the file has been moved across partitions, the file system has been defragmented (in the case of a fragmented file), or if it has been extended and the old clusters have not yet been overwritten. This is also true for copy-on-write file systems like btrfs [109], or in SSD hard drives, as they by design update files at different locations than the original file content. The most important fuzzy hashing tools so far are *ssdeep* [73] and *sd-hash* [110]. While *ssdeep* generates a constant 80-byte output for each file, *sdhash* generates a similarity digest of variable output size. Both can then be used to calculate the similarity score

⁶<http://www.nsrl.nist.gov/>

between files by comparing the fuzzy hash values of files either by using the edit distance (ssdeep) or the Hamming distance (sdhash). Numerous comparisons between those two tools have been published [111, 18], and generally both have their advantages. NIST also releases fuzzy hashes for a subset of the NSRL RDS using sdhash⁷ and ssdeep⁸ as well as certain blocks of files like MD5 on the first 4 kilobytes of the corpus files⁹.

Counter-forensic tools are another front that forensic examiners have to battle. Many different approaches can be used to hinder forensic analysis, e.g. encryption, online network traffic anonymization or file signature analysis. Tools to encrypt hard drives like *LUKS*, *FileVault*, *BitLocker* or *TrueCrypt*, are readily available for all major operating systems and can render an analysis something between hard and close to impossible [22]. Communication content can be encrypted using protocols like TLS (HTTPS, IMAPS, SMTPS) or OTR [17], and online communication can be anonymized using Tor [37], JonDonym or I2P [107]. Tor is a very active field for research [120, 69] with regards to online privacy and network censorship resistance, and so far one of the effective countermeasures against spying according to files revealed by Edward Snowden [13]. Steganographic methods can be furthermore used to hide the existence of information in plain sight [70, 62], for example in pictures or videos, and network traffic can be shaped to look like something completely different [83, 121, 123] if the attacker is using traffic analysis to infer information about encrypted information content [12, 96]. If not done correctly, however, these methods can be defeated: very recent approaches like StegoTorus [123], SkypeMorph [83] and CensorSpoofers [121] have been shown to be vulnerable to detection [63]. Furthermore, cryptography, if implemented incorrectly, can considerably harm these tools, which has been shown recently on CryptoCat¹⁰. While all these problems seem troublesome in regard to digital forensics, they are essentially troublesome (to say the least) in oppressive regimes where the lives of dissidents are in danger. As such, this thesis is not judging on the way digital forensic methods are employed all over the world: just like every coin has two sides, digital forensics can be easily employed for or against certain interests. Thus, the usage of counter-forensic tools is nothing that should be considered to be negative per-se.

The discussion about privacy as the “right to be left alone” [122] is nowadays more prevalent than ever. In particular the revelations by Edward Snowden about the surveillance conducted by the NSA as well as other secret services has lead to public discussions on online security and privacy, as well as mobile phone security and dragnet-surveillance on a global scale in general. This will also affect research in computer security in the near future, as well as digital forensics. The average user is unprotected against such powerful adversaries, and often without a chance to even detect attacks until it is too late to mitigate or reduce the possible impact. Not only is the NSA actively exploiting software weaknesses on target’s computers (codename *Quantum*), they are also collecting unencrypted as well as encrypted communication content of users on a large scale (codenames *Upstream*, *Tempora* and *Prism*) and between data centers of large Internet

⁷http://www.nsrll.nist.gov/morealgs/sdhash_3.3/sdhash.html

⁸<http://www.nsrll.nist.gov/ssdeep.htm>

⁹<http://www.nsrll.nist.gov/morealgs.htm>

¹⁰<http://tobtu.com/decryptocat.php>

companies like Google and Yahoo (codename *Muscular*). The NSA is also accused of weakening the standardization process of a cryptographic pseudorandom number generator published by NIST (codename *Bullrun*) DUAL_EC_DRBG [116], which was the default PRNG in many products including RSA's [56]. The full extent of the NSA's surveillance programs is still unclear, and new revelations are constantly released.

The naive approach to conduct network forensics has several limitations, as simply encrypting the data in transit is insufficient. Anonymizing networks like Tor, as well as SSL MITM attack can be used to inspect data, and secret service agencies can be expected to do that on a regular basis. HTTPS is one of the most important protection mechanisms for data in transit. It uses the TLS (and SSL) protocol to secure online communication, with TLS 1.2 [34] being currently the most recent version of TLS. Client and server can authenticate themselves to each other, use the TLS protocol to derive an encryption key as well as agree on a symmetric encryption algorithm like RC4 or AES [115] by using public key cryptography [35]. However, it is still not commonly used for a large portion of websites, with just a few notable exceptions: Gmail uses it by default for all users since January 2010 [114], whereas Twitter followed in February 2012 [119]. Facebook announced in July 2013 that they enabled HTTPS by default for all users [106], while Yahoo announced to follow sometime early 2014 [100]. HTTPS also has been subject to a plethora of attacks in recent time [26]: RC4 has been found to be weak and insecure [6], CBC as the preferred mode of operation has its problems with attacks named BEAST [39] and Lucky13 [7]. Numerous attacks targeted compressed plaintexts prior to encryption, e.g. CRIME [40] and BREACH [104]. TLS errors are hard to understand for users [3, 4], the trust chain when using commercial certificate authorities can be troublesome [41, 8] and implementations in mobile apps like Android are often insecure [47, 46]. As such, HTTPS is not suitable to defend against powerful adversaries. Especially with targeted attacks as well as broad, Orwellian surveillance manifesting itself as a global, passive adversary, defense in depth is the only option. Encryption alone is often insufficient, and protection mechanisms are needed on multiple levels.

To counter these developments, this thesis aims at providing insights into online storage providers as well as extend the current state of the art in digital forensics for this kind of scenarios. The traditional approach to conduct a forensic analysis of online storage systems has several limitations: for one, the operator is involved in retrieving the information, which is problematic as it needs to put trust in the operator who is also often not obliged to help due to cross-country jurisdictions. Furthermore, it can take weeks or even months to obtain information. The ultimate goal for this thesis is to develop novel information extraction techniques as well as to critically assess existing processes and methods, in particular for distributed environments. Counter-forensic methods are evolving, and the increasing prevalence of mobile computers and online connectivity will further challenge forensic investigations. The ever more pervasive usage of encryption will furthermore advance the difficulties when analyzing computer systems.

Proposed Solutions

This chapter describes the goals and methods used to obtain the results of my evaluations.

Goals

The goal of this thesis is to tackle current problems of digital forensics in connection to online services, like anonymizer- and online storage systems, and work towards their solution in a broader picture. The goals for this dissertation are to enhance the current research on digital forensics, with special focus on the following areas:

- Information extraction from online- and **cloud services** as well as the corresponding area of **browser forensics** in a sound way, so that the information is usable for examinations and in court.
- Analysis of **anonymization networks**, in particular Tor: how they are used as counter-forensic tools, and what information still leaks from their usage.
- Examining **online storage** services like Dropbox and how traditional file forensic methods can be applied.
- Assess the feasibility of **fully automated digital alibis** with regards to counter-forensics.
- Enhance **browser fingerprinting methods** to reliably detect a given browser based on its characteristics.

So far, cloud forensics is conducted mostly in a passive way - a request is sent to the service provider, and trust has to be put in the answer to meet forensics requirements, in particular regarding data completeness, presentation and preservation. In a recent paper, an API-based approach has been proposed to conduct forensic analysis on social networks [64]. This has the benefit that no direct involvement of the service operator is required, and data acquisition can be considered repeatable. The API often allows access to additional information compared to webpages, like exact timestamps and additional fields of information. Regarding online storage services, we would like to assess whether a given file is stored at the service or not, and, additionally (if possible), by which user. This allows us to test a set of possibly critical files, without the need for dragnet surveillance or passive deep-packet inspection that has to be already set up

prior to an incident. This approach is successfully used for example by PhotoDNA¹¹ (on Twitter, Facebook and Bing) which uses a form of proprietary image hash to test image similarity to a known set of images in connection with sexual exploitation of children.

With regards to browser artifacts, the reliable identification of a browser is still non-trivial. Even though the browser identifies itself via a string (the UserAgent string), this is not a security feature - it can be easily spoofed or modified by the user. As such, web servers that log the UserAgent of users cannot be certain that the browser used was indeed the one proclaimed. We would like to identify new methods that allow us and others to draw conclusions on the specific browser of a user by looking at its characteristics. These methods can be either active or passive, e.g. for the web server or for captured network traffic. In a forensic context this would be of importance for example in cases where a web server gets hacked, as the access- and error logs usually contain not only the IP address of the attacker but also the UserAgent. On the other hand we would like to use our findings to improve the overall security of session management, as broken session- and authentication management is currently among the most prevalent threats online (“A2 – Broken Authentication and Session Management” in the OWASP Top 10 from 2013) [99].

Regarding online anonymization services like Tor, many parameters of its usage are in the dark. It is unknown why people use it and why they donate bandwidth to run Tor relays, even though many hundreds of thousands of people use it every day and there are currently more than 5,000 Tor relays. Related work found that it is mostly used for browsing the web and downloading files using P2P protocol [81], but this work dates back to 2008 when the Tor network was much smaller. Furthermore, P2P network usage has been found to be possibly dangerous on Tor as many torrent clients leak identifiable information like the client IP address [16]. We would like to understand how Tor is used nowadays, and especially if it is used in a secure way as recommended by the Tor project itself. Furthermore, as Tor could be used as a counter-forensic tool, we would like to understand the implications of its usage for digital forensics, in particular network forensics, and if the data transmitted still leaks information that could be used to reduce online anonymity.

Finally, we would like to draw attention to the fact that traces left on hard drives are not necessarily coming from a user or background processes, but can merely come from a purportedly fully automated program to thwart analysis. To show this, we will implement a digital alibi engine that incorporates social interactions like chatting, writing e-mails and using local programs just like a normal user. Traditional forensic processes that rely on file system metadata and network traffic analysis are expected to be easily tricked by such a program into detecting patterns of user interaction in cases where no user was present. Another goal of our work is to study slack space, a well known artifact of digital forensics. While current analysis methods target slack space implicitly instead of explicitly (keyword search on entire hard drives by ignoring file system metadata), free tools are available to store data in slack space. These tools also allow to encrypt data prior to storing it in slack space, thus defeating the implicit analysis with keywords.

¹¹ <http://www.microsoft.com/en-us/news/presskits/photodna/>

There is no existing prior work towards assessing the amount of slack space created by modern operating systems, with tens of thousands of system files. Given that many of these files are static and do not change over time, the slack space can be considered persistent and protected from getting accidentally overwritten.

If possible, our methods and obtained results should be quantified and have to be compared to previous research in this field regarding effectiveness and applicability. If flaws and vulnerabilities in implementations or protocols are discovered, they have to be communicated to the creators and developers as soon as possible to improve overall security of the affected products. Research results in general (papers, data and tools) will be published as openly as possible. Ethical guidelines and codes of conduct from, e.g. ACM [9] and IEEE [68] are honored as well as ethical principles that are agreed upon by the majority of the scientific community. Many ideas discussed in the literature as well as published in the papers presented here have limitations and restrictive conditions that hinder the general applicability. As such, the goal has always been to find the most permissive set of preconditions and requirements while making sure that the prospect and implications are still generally applicable and not restricted to particular technologies. An example from one of the papers presented here would be the data deduplication attack, which was demonstrated on the cloud service Dropbox: even though it was specific to the implementation and particular protocol used by Dropbox, the attack itself is based on the naive use of client-side hashing for data deduplication [60], compared to more complicated data possession proofs [130, 59]. We showed that trusting the client to do this faithfully and correctly is not to be taken for granted and a possible attack vector for unauthorized data access [91].

Methodology

The following methodology was used:

- **Extensive literature review** on the current state of forensic research and the identification of urgent challenges that need to be solved.
- **Prototype implementation** for proof of concepts like new data extraction techniques or analyzation methods.
- **Empirical analysis** of feature predominance and assessment of expected information increase in forensic examinations.
- **Prototype dissemination** with an **open source license like the GPL** so that the forensic community can use the outcomes and enhance their functionality as needed.

The majority of the results are based on quantitative research; a-priori research questions [29] were typically formulated in this way. This is in particular true for novel attack vectors, as a proof-of-concept is often needed to not only show the general feasibility of the attack, but also to assess the possible impact on a larger scale. Most data collections and evaluations are based

on or built around proof-of-concept implementations. The designs of the experiments are described in the corresponding papers in a way that they are reproducible in general; available data and used tools were openly published to make the results reproducible as well. However, responsible disclosure was employed: tools that exploit security weaknesses or could potentially harm users or service operators in any way will not be released until the underlying issues are communicated to the vendor.

We started our work on the online storage analysis by evaluating the protocol of the most popular software at the time: Dropbox. Our initial observation was that once a file has been stored on the Dropbox servers there is in general no need to re-upload it. Even if it was deleted and re-added to the Dropbox folder, somehow the client software tracked which files were already stored online. This was also true for cross-account uploads: another client (without any relation to the first user account but) with the exactly same file was not asked to upload the files to the servers. It was deduplicated, and the same exact file had to be uploaded only once. Upon looking at the obfuscated executable we were able to see that files are split into chunks of 4 megabytes of which each was hashed using the SHA-256 cryptographic hash function. The hash was then sent to the server, and if the chunk was already stored on the servers, retransmission was omitted. As the hash calculation was done locally, we were able to manipulate it and to obtain unauthorized access to data. Knowing the hash value of a file of interest, it could be obtained from Dropbox without prior possession of the file. This was of interest for digital forensics regarding two particular methods: for one, it was possible to assess if a given file was stored on Dropbox, even though we did not find a way to identify the account that uploaded it; secondly, data could be hidden online without leaving any local traces, as long as the hash values are remembered. At a later point in time, this weakness could be used to download it again. Thus up to 4 megabytes were retrievable by one 256 bit hash sum. To evaluate the file retrieval attack, we downloaded the most popular files (without any obvious copyright on them) from The Pirate Bay torrent tracker¹² and downloaded the corresponding files from Dropbox. We also evaluated to what extent files could be hidden using another flaw in the protocol: once a file is uploaded, it was linked to the user account by a final request. By omitting this final linking request we were able to upload the files successfully to Dropbox, but without linking them to our account. Even weeks and months later the data was still retrievable. All in all, these issues were not only worrying from a security point of view, but also for digital forensics as a chance and a valid method for counter-forensics. We proposed countermeasures, in particular a data possession protocol to prevent our attacks while at the same time allowing deduplication. Dropbox fixed it, however, by preventing deduplication altogether, and every file since is uploaded to their servers. We speculate that this was an easy fix for them, given that Amazon, who is running the underlying infrastructure, is not charging customers for inbound traffic.

Based on browser fingerprinting artifacts we found that it is possible to identify a given browser by actively probing it with JavaScript. While browser fingerprinting is currently a very active area of research [43, 127, 86], we derived novel fingerprinting vectors that are three orders of magnitude faster than related work [85]. In our work we use fingerprinting of the underly-

¹²www.thepiratebay.se

ing JavaScript engine as well as upcoming browser features and standards based on HTML5 and CSS3, as these are not yet uniformly implemented in all major browsers and thus very suitable for identifying the browser currently used by the client. We started by analyzing the JavaScript engines for different browsers (Firefox, Internet Explorer, Opera, Chrome and Safari) on multiple operating systems (Windows 7, Windows XP and Mac OS X) and how they differ in standard accordance to the ECMAScript standard [44] for JavaScript. We then collected, for each browser/operating system combination (more than 150), the outcome of the official TC39 test suite, *test262*¹³ and stored the results in a database. From that data we derived two different methods to minimize the computational overhead and runtime on the client. Furthermore we were able to show that UserAgent string modifications are easily detectable by our method. Based on these results we extended the scope of our browser fingerprinting to include upcoming web standards, HTML5 and CSS3. While some features are already uniformly agreed upon and implemented by the browser vendors, some are not even yet standardized, but already implemented in some browsers. We used that additional information to increase the precision of our active tests and built a framework to increase HTTP session security, by tying a HTTP session to the specific browser. Session hijacking becomes thus harder, as the attacker has to run the same exact browser, or needs additional information despite traditional session management information like session cookies. The framework, which was named SHPF, can thus detect session hijacking on the server side and implement proper countermeasures like requesting the user to re-authenticate or simply terminating the session for that user. We used this framework to also include additional encryption, as proposed in [2]. The framework is released under an open source license and can be easily incorporated in existing websites and frameworks with just a few lines of code.

Regarding anonymous communication and their (counter-)forensic effects we evaluated the Tor network in depth. We ran a Tor relay (and in fact are currently still running a non-exit Tor relay) and analyzed publicly available information like the Tor consensus information¹⁴. The consensus information contains all necessary information for the clients to build their paths through the Tor network and is essential to prevent intersection attacks where clients only know a partial state of the entire network [37]. While the Tor network's infrastructure is run by volunteers, we found that it is very stable in total numbers and bandwidth. We incorporated our data collection process into the tor-status website which used a Perl script to connect to a local Tor relay to periodically extract all information about the Tor network. tor-status is now superseded by Tor Metrics¹⁵ and Atlas¹⁶. We furthermore designed analysis methods that allow to present the most relevant information on a first sight, like total number of relay, total number of exit relays or country-specific numbers. To analyze possible artifacts of user data we ran an on-the-fly analysis script on a Tor exit relay. We did this to find out if Tor is used in a safe way and as recommended by the Tor Project, and also to see if users can be deanonymized by merely looking at the unencrypted network traffic. Our data collection was built to be as least invasive to

¹³<http://test262.ecmascript.org>

¹⁴<https://metrics.torproject.org/data.html>

¹⁵<https://metrics.torproject.org/network.html>

¹⁶<https://atlas.torproject.org/>

user privacy as possible, and we analyzed only the HTTP requests issued over the Tor network without permanently storing them. Since Tor is used in many countries to circumvent online censorship, we found many connections to, e.g. Facebook or other social media websites that directly contain user information and were unencrypted by default at the time of the analysis. We also found unencrypted file downloads for software with known vulnerabilities (PDF, office files or executables) which could be piggybacked or exchanged with malware on the fly by an adversary running the Tor exit node.

To analyze the quantity and persistency of slack space, we installed 18 different versions of the Windows operating system ranging from Windows XP and Server 2003 R2 to the most recent Windows 8 RC and Server 2012 RC. Our goal was to analyze how many files are changed during system updates and how this affects the corresponding slack space. In total, more than 2500 system updates were installed, including 10 service packs. We used virtual machines to run that many versions of Windows and exported the file system to a raw image prior to analysis. Slack space is an artifact of clustering file systems like NTFS or FAT [21] and can be used to hide data in the last sectors of files, as these sectors are allocated to a file, but often unused since files hardly align exactly in size with the allocated space. Many tools are available to hide information in slack space, e.g. *bmap* or *slacker.exe*. We then collected the file system metadata using *fiwalk* [51] and estimated total capacity and stability. For all Windows versions we analyzed, 44 megabytes were available on average across all steps in our evaluation. From the initial file slack 78% were still available at the end of the evaluation process. Creating digital alibis with social interaction were another research problem we wanted to encounter: currently, in particular in court or in companies, digital forensic methods are used to find out what happened in what order of events. However, the traditional analysis processes can be thwarted with automated processes that manipulate timestamps on disks as well as generate network traffic. We built a proof-of-concept framework in Python that runs fully automated and is able to interact online with multiple means of communication (e-mail, chat). Our framework is capable of interacting with the computer like an actual user by sending key strokes and mouse actions as simulated input. As such, it can also be used to generate test data for new and upcoming software, not only hard drive images, but also network traffic of various kinds.

Scientific contributions

The scientific contributions of this thesis can be roughly categorized according to for parameters - they are either active or passive, and work online or offline. The passive contributions work with data or knowledge that is already there but has not yet been used in this particular fashion. Instead of modifying, e.g. source code or websites, the tools can work with log data or information that are already collected and processed. Active on the other side means that either on the client- or the server side additional code has to be executed and information collected for them to work properly, and existing frameworks and source codes need to be extended. Online solutions refer to contributions that require Internet connection, or work only in online environments, whereas offline means that they work without network connection of any kind. A graphical representation of the contributions according to this scheme can be seen in Figure 1.

The passive online contributions contain the research on Tor - both the paper on infrastructure monitoring [88] as well as the paper on Tor usage and information leakage [67]. While the Tor infrastructure monitoring works with existing data, the network consensus, it can be used offline. However, since the Tor network consensus is updated once every hour, it can be beneficial to analyze the data as soon as it is published, and for that some form of network connection is required. For the historical data, which is available online from the beginning of 2006¹⁷, the data can be analyzed offline. That is why Tor infrastructure monitoring is located in the graph close to the line between online and offline. The Tor information leakage analysis works only in online environments, but is completely passive in nature. As such it is located in the upper left corner. The lower left quarter contains the work on Windows file slack analysis and persistency [89], as it neither requires network connectivity nor an active component besides the collection tool. Measuring slack space can be even done on the fly in the running operating system (for most of the files), which means that it can be calculated passively and offline. The rest of the papers are located in the upper right quarter, as they are active in nature and require network connectivity. Automated alibis [15] can make use of the fact that a computer is online by generating social interaction with online communication protocols like e-mail, browsing the web or instant messaging. Without Internet connection, only local information is modified and programs that work on local data are run. The online framework regarding JavaScript engine fingerprinting [90], online storage deduplication and slack space [91], as well as the SHPF framework [92] work purely online and need modifications of existing code and frameworks. Further distinctions for these papers would be whether the modifications have to be done on the server- or the client side

¹⁷<https://metrics.torproject.org/data.html>

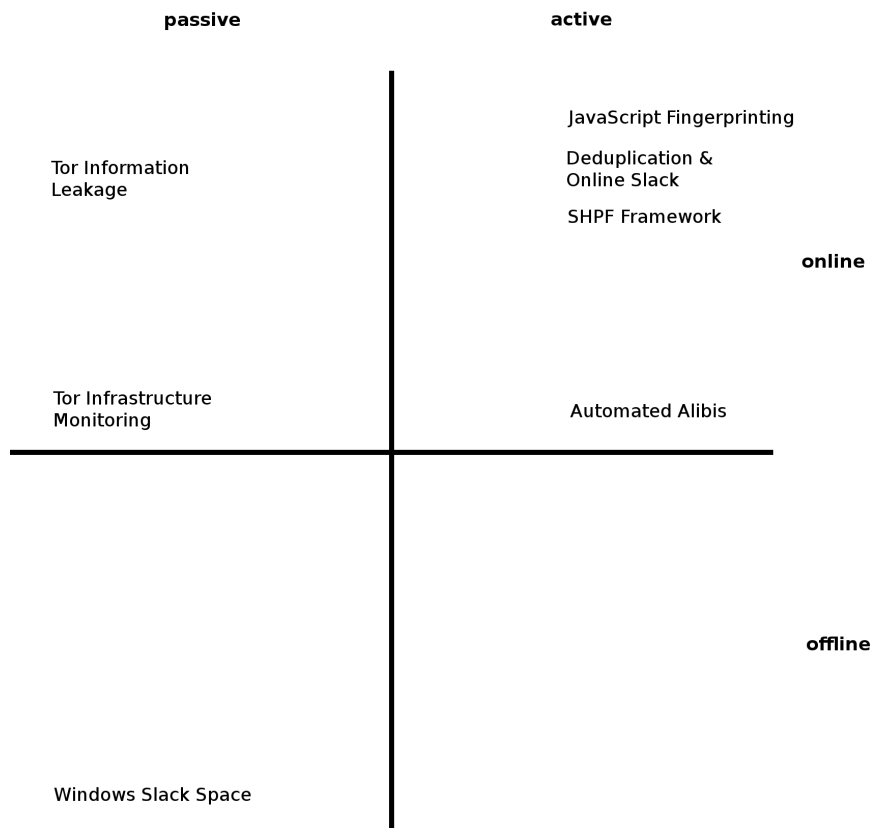


Figure 1: Classification of Contributions

and if the additional code needs to be executed at the client, at the server or at both, but was omitted for reasons of brevity. SHPF can be added to existing webpages with a few lines of code to invoke the SHPF framework. JavaScript engine fingerprinting needs to add the required tests to the webpage’s source code, and online data deduplication with the possibility of online slack space requires local modifications to the source code, in our case Dropbox.

The papers and in particular their context are discussed in detail in the following, categorized accordingly to the previously described research fields: online storage, anonymous communication and digital forensics.

Digital Forensics on Online Storage

The paper **Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space**, which was published at the *USENIX Security Symposium* [91] in 2011, describes

three different attacks against Dropbox¹⁸, a popular cloud storage service. The first attack abuses the local hash computation on the client side to get unauthorized access to remotely stored files - if the hash value(s) of the file are known to the attacker. While this scenario can pose as a hen-and-egg problem, it could be used for file sharing and covert communication as (independently) implemented by Dan DeFelippi [33, 82]. This attack could further be used for stealth data exfiltration, as only a hash value has to be secretly transmitted to the attacker instead of the entire file. While the underlying problem for this attack has been independently found by Harnik et al. [60], our work had a proof-of-concept implementation and showed that the attack is applicable on the (at the time) largest cloud storage service provider with more than 10 million users. Dropbox changed their protocol soon after our notification. As of July 2013, Dropbox has 175 million users [28] and is still subject of active research [72, 38, 105]. The second attack is based on the observation that Dropbox did not validate or tie the so-called hostID to the specifics of a system once it is set - if this somehow becomes known to the adversaries, they can access all files of the victim. This attack has also been independently discovered by Derek Newton [97]. Both attacks happen completely transparent to the victim, who cannot detect these attacks as they are targeting the protocol and the servers only. The third attack abuses the transmission protocol by up-/downloading files without linking them to a specific account, which usually happens after uploading files. This can be used to secretly hide files at Dropbox and thus inside the Amazon cloud [38]. This attack can furthermore be used to upload files to other peoples' Dropbox if the victim's hostID is known. None of these attacks are specific to Dropbox, but apply to other cloud storage services with vulnerable implementations or protocols as well. Dropbox fixed these issues after notification by disabling client-side data deduplication entirely and encrypting the hostID at rest. Our paper finally proposed to use interactive challenge-based data possession proofs instead of relying on cryptographic hash functions as sole method to check if a client is really in possession of a new file which is possibly already stored on the server. Numerous other methods have been recently proposed in the literature [59, 130, 129], but our countermeasure specifically targets the use-case in which two parties (willingly or unwillingly) collaborate to facilitate file sharing and cryptography alone is not suitable as a sole countermeasure, given that cryptographic keys can be exchanged between collaborating parties.

The paper **Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting**, which was published at the *Web 2.0 Workshop on Security and Privacy (W2SP)* [90] accompanying the IEEE Symposium on Security & Privacy in 2013, improves previous research in the area of browser fingerprinting based on the Javascript engine by three orders of magnitude. While previous research used performance and timing patterns [85] of various benchmarks, our method uses Javascript conformance tests which are available online, in particular *test262*¹⁹. These tests assess to what extent the Javascript engine of the browser conforms to the official ECMAScript standard, and the failed test cases are specific for browsers and particular browser versions. Recent browser versions failed at something between four and 35 cases, whereas older browsers were having problems with more than 3,500 cases out of the approximately 11,500 tests. While a full run on all these 11,500 tests only takes about 10 minutes on a modern PC

¹⁸<https://dropbox.com>

¹⁹<http://test262.ecmascript.org>

and up to 45 minutes on a smartphone, we also discussed techniques to make our tests as efficient as possible. Our tests to fingerprint a particular browser required only a few hundred lines of code to be executed on the client side, which reduces the runtime to a fraction of a second. We evaluated our approach using a plethora of browser versions and operating system combinations, resulting in more than 150 configurations. We derived techniques to build a decision tree to find the browser of a user without any a-priori knowledge like the UserAgent string. We were also able to show that the underlying Firefox version used in the Tor browser bundle [37] can be identified as it employs a modified UserAgent string to increase the size of the anonymity set [36]. From the browser bundles that were released between May 2011 and February 2013, our technique was able to identify 6 out of 9 browser versions correctly and find the modified UserAgent strings. This is however not an attack on Tor or the Tor browser, but can be used to decrease the size of anonymity sets. Still, it could be used to conduct forensic analysis on clients connecting to a webserver using Tor, but also to identify modified UserAgent strings during web sessions. Unfortunately the dataset we collected was lost due to a hardware failure.

The paper **SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting**, which was published at the *International Conference on Availability, Reliability and Security (ARES)* [92] in 2013, builds upon previous work on browser fingerprinting and presents a framework that allows HTTP session security to be enhanced using browser fingerprinting. Traditional HTTP session security, as it is used today, has many shortcomings. This was also demonstrated with the release of numerous tools that allow automatic session hijacking, for example in unencrypted wireless networks, e.g. FaceNiff²⁰, Firesheep²¹ or Droidsheep²². SHPF ties the session of a user to a characteristic set of properties of the underlying browser using browser fingerprinting and is implemented in a modular way. As such it can be extended using upcoming fingerprinting methods like HTML5 rendering differences across browsers [86] quite well. Part of the implementation were HTML5- and CSS feature fingerprinting, as well as binding a session to a particular UserAgent string or IP address. Browser fingerprinting has been recently discussed to be already in use by advertising networks [98, 1], and as such we believe that the public release of the SHPF source code can help to secure online communications. This is an extension of the previously discussed idea to identify browsers in forensic investigations and to prevent session hijacking. While session hijacking can have legitimate use-cases, e.g. to transfer a web session to an untrusted terminal without entering a password [20], it is usually an indicator for hacking attempts.

Insights into Anonymous Communication Methods

The paper **Anonymity and Monitoring: How to Monitor the Infrastructure of an Anonymity System**, which was published in the journal *IEEE Transactions Systems, Man, and Cybernetics, Part C: Applications and Reviews* [88] in 2010 describes data collection of public information

²⁰<http://faceniff.ponury.net/>

²¹<http://codebutler.github.io/firesheep/>

²²<http://droidsheep.de/>

and numerous evaluations on the Tor network [37]. Tor is an anonymizing overlay network, and the underlying infrastructure (the Tor relays) is run by volunteers. The paper clearly showed that not only the number of servers is volatile, but also that there are significant country-specific usage patterns in the type and total number of Tor relays. For example, the number of Tor relays can be considered rather static in the US during an one week period, while the total number of relays in Germany had a daily pattern: +/- 10% (out of approximately 350) of the servers were running only during the evening hours. The same was surprisingly true for exit relays. Even though we could not find any indication that these patterns are correlated, they clearly showed that Tor relays and how they are operated should be further investigated. The overall number of Tor relays has tripled since the paper was published, while the number of exit relays has doubled. The paper has been an extension of my master thesis conducted at the Vienna University of Technology in 2009 [87]. The Tor project incorporated parts of the results and evaluations in their official Tor metrics portal (online at <https://metrics.torproject.org/>) and independently published results and related methods [78, 79] similar to ours.

The paper **Tor HTTP Usage and Information Leakage**, which was published at the *IFIP International Conference on Communications and Multimedia Security* [67] in 2010, showed that Tor users are unwillingly leaking (possibly sensitive) information by browsing the web. It furthermore showed that Tor users are often vulnerable to MITM-like file replacement attacks, as a malicious Tor exit relay can easily replace certain requested files. Furthermore, plenty of social networking information was observable in the clear, which makes user deanonymization often trivial. Related work from 2008 analyzed [81] how Tor clients use the network in general, and a paper from 2011 showed that using Tor for P2P file sharing often leaks identifiable information that allows user deanonymization [16]. Today, Tor users are often protected by the Firefox extension *HTTPS everywhere*, which uses a whitelist of about 10.000 preconfigured websites²³ to encrypt communication content with TLS by default. Websites that use TLS for all connections, like Gmail (since 2010), Twitter (since 2012) and Facebook (since 2013), are still a minority, even though anecdotal evidence suggests that the computational overhead is small [75]. This leaves room for novel attacks, e.g. our friend-in-the-middle attack on social networks [65, 66]. With the release of the NSA files by Edward Snowden, numerous website operators and e-mail service providers have started to switch to https-by-default, including Yahoo, Microsoft and LinkedIn.

Contributions to Traditional Forensic Techniques

The paper **Quantifying Windows File Slack in Size and Stability**, which was published at the *IFIP WG 11.9 International Conference on Digital Forensics* [89] in 2013, analyzed the quantity and persistence of file slack space in Windows environments, with special regard to Windows system updates. The intuition behind this paper was that once Windows is installed, it is likely that many system files are never touched again, updated or rewritten with regular usage, e.g. font

²³<https://gitweb.torproject.org/https-everywhere.git/tree/HEAD:/src/chrome/content/rules>

files, program libraries, help files or pictures, as well as many other file types. If a file is kept for a long time without modifications, the slack space behind the file is static. Therefore we analyzed 18 different versions of Windows, ranging from Windows XP to Windows 8 RC, respectively Server 2003 R2 to Server 2012 RC, and measured slack space capacity and stability. While the detailed results can be seen in the paper, we observed that tens of megabytes, sometimes even more than a hundred megabytes of file slack space are available just by inspecting the files of the operating system itself. This is due to the fact that Windows is a complex system, using tens of thousands of files, but the results are applicable to any other operating system that users sector clustering for file system efficiency. Slack space is a well known phenomenon in digital forensics, where fragments of files can be possibly recovered once the cluster was marked deleted, reused and the old data was not fully overwritten [54]. Identifying file fragments regarding their file type or what file they originally belong to is a closely related and very active area of research at the time of writing [112, 128].

The paper **Towards Fully Automated Digital Alibis With Social Interaction**, which was published at the *IFIP WG 11.9 International Conference on Digital Forensics* [15] in 2014, raises awareness regarding the problem that forensic investigations can be foiled with fully automated, digital alibis. Digital evidence is often regarded as per-se authentic and tamperproof, at least to my general impression, in particular in court cases in which the analysis is often done by expert witnesses. We proposed and released a framework as a proof-of-concept to show that digital alibis cannot be blindly trusted. Our framework is, among other things, built upon social interactions, whereas chatting in Skype or interacting on Facebook is simulated. A similar technique to our approach has been previously discussed as an attack vector for fully automated social engineering [76]. Captured network traffic, as well as hard drive analysis [19], cannot easily tell the presence of our framework if certain (non-trivial) precautions are met. We suspect however that advanced analysis techniques as well as statistical analysis are able to detect the usage of the framework. Compared to previous work in this area, our framework is not dependent on particular operating systems like Android [5], OS X [23] or Windows [31, 24], as it is written in Python. However, it was recently announced that the Skype desktop API will be retired by the end of 2013 [45]. Since our implementation uses this API, we will have to change that usage towards GUI automation. Another potential use-case for this framework is the fully automated generation of disk images and network captures for educational purposes. While there exist numerous standardized forensic corpora [49] that are used to develop new methods and evaluations, all of them have some form of limitation [125]. For one, the real world corpus holds currently in total approximately 70 TB of disk images [48], but access to them is tricky - for one, sending and processing that vast amount of information is non-trivial. Secondly, as the owner's institution is located in the US and the hard drives could contain potentially sensitive and personal information, a US-based institutional review board (IRB) approval is required. However, many other countries in the world (and Austria in particular) do not have these form of approval process, which can as such be considered an obstacle, or at least time-consuming. Our framework allows the realistic generation of organic disk images, since the operating system and the applications on top of it are really executed, and as such contain every piece of data and metadata that an actual user would leave behind. This is different to related work, which creates synthetic hard

drive images by simulating activities of users using Markov chains [126]. While the synthetic approach is faster and has more parameters that can be configured, the organic hard drive image has the benefit that instead of the creation process itself the user is simulated. It could also be used for the creation of realistic network captures, since an actual browser is used to surf the web, and real applications are used to communicate. It is extensible and can be easily adapted to include custom applications, additional features or scripting actions. We released our framework as open source²⁴.

²⁴<https://github.com/mmulazzani/alibiFramework>

Conclusion

This thesis demonstrates new techniques for forensic investigations and highlights current challenges, in particular for online environments. With data being ubiquitously stored in the cloud and web interfaces being used for data access across numerous different devices, the forensic process as it is conducted today can be easily overwhelmed. Depending on the threat model and the capabilities of the person to be investigated, counter-forensic methods, its tools and the increasingly present use of encryption can hinder and even prevent forensic analysis. Nevertheless, this field of research is currently very active, and new methods and data extraction techniques are direly needed.

The results in particular demonstrate findings with forensic context on the Tor network and how it is used, how browsers can be fingerprinted and how this fingerprinting can be used to enhance HTTP session security as it is implemented today. This thesis also adds to traditional forensics by analyzing the size and stability of file slack space and presenting a framework that was used to contribute towards automated alibi generation by adding social interactions using numerous communication channels.

Overview of Research Contribution

Publications

List of published papers, in chronological order:

- Anonymity and Monitoring: How to Monitor the Infrastructure of an Anonymity System, published in the *IEEE Journal on Transactions Systems, Man, and Cybernetics, Part C: Applications and Reviews* in 2010 [88]
- Tor HTTP Usage and Information Leakage, published at the *IFIP International Conference on Communications and Multimedia Security* in 2010 [67]
- Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space, published at the *USENIX Security Symposium* in 2011 [91]
- Quantifying Windows File Slack in Size and Stability, published at the *IFIP WG 11.9 International Conference on Digital Forensics* in 2013 [89]
- Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting, published at the *Web 2.0 Workshop on Security and Privacy (W2SP)* in 2013 [90]
- SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting, published at the *International Conference on Availability, Reliability and Security (ARES)* in 2013 [92]
- Towards Fully Automated Digital Alibis With Social Interaction, to be published at the *IFIP WG 11.9 International Conference on Digital Forensics* in 2014 [15]

Released Source Code and Tools

- Source code of our framework for enhancing HTTP(S) session security with browser fingerprinting at <https://github.com/mmulazzani/SHPF>
- Source code of CSS and HTML5 fingerprinting, as part of the SHPF source code
- Source code of our digital alibi framework using social interaction at <https://github.com/mmulazzani/alibiFramework>

Released Data Sets

- Data set of our NTFS slack space evaluation at <http://sba-research.org/wp-content/uploads/publications/slackspaceDataset.7z>
- Data set for JavaScript engine fingerprinting has unfortunately been lost due to a hardware failure

(Co-)Instructed Master Theses

- Stefanie Beyer, master thesis at the Technical University of Vienna: “Towards automated digital alibis”
- Ioannis Kapsalis, master thesis at Aalto University: “Security of QR Codes”
- Robert Koch, master thesis at the Technical University of Vienna, funded by Google Summer of Code: “On WebSockets in Penetration Testing”
- Christian Kadluba, master thesis at the University of Applied Sciences Technikum Wien: “Windows Installer Security”
- Reinhard Kugler, master thesis at the University of Applied Sciences Campus Wien: “Analysis of Android Apps”
- Philipp Reschl, master thesis at the University of Applied Sciences Campus Wien: “Identifizierung der Webbrowser Version anhand des Verhaltens der JavaScript Engine”
- Thomas Unger, master thesis at the University of Applied Sciences Campus Wien: “HTTP Session Hijacking Prevention”
- Herbert Brunner, master thesis at the Technical University of Vienna (in progress): “Detecting Privacy Leaks in the Private Browsing Mode of Modern Web Browsers through Process Monitoring”
- Andreas Juch, master thesis at the Technical University of Vienna (in progress): “Btrfs Filesystem Forensics”
- Richard Köwer, master thesis at the University of Applied Sciences Wien Campus (in progress): “HoneyConnector - Detecting Eavesdropping Nodes in the Tor Network”
- Robert Annessi, master thesis at the Technical University of Vienna (in progress), funded by Google Summer of Code: “Improvements on path selection in the Tor network”

Bibliography

- [1] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1129–1140. ACM, 2013.
- [2] Ben Adida. Sessionlock: securing web sessions against eavesdropping. In *Proceedings of the 17th international conference on World Wide Web*, pages 517–524. ACM, 2008.
- [3] Devdatta Akhawe, Bernhard Amann, Matthias Vallentin, and Robin Sommer. Here’s my cert, so trust me, maybe?: understanding tls errors on the web. In *Proceedings of the 22nd international conference on World Wide Web (WWW)*, pages 59–70. International World Wide Web Conferences Steering Committee, 2013.
- [4] Devdatta Akhawe and Adrienne Porter Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *Proceedings of the 22th USENIX Security Symposium*, 2013.
- [5] Pietro Albano, Aniello Castiglione, Giuseppe Cattaneo, Giancarlo De Maio, and Alfredo De Santis. On the construction of a false digital alibi on the android os. In *Intelligent Networking and Collaborative Systems (INCoS), 2011 Third International Conference on*, pages 685–690. IEEE, 2011.
- [6] Nadhem AlFardan, Daniel Bernstein, Kenneth Paterson, Bertram Poettering, and Jacob Schuldt. On the security of rc4 in tls. In *USENIX Security Symposium*, 2013.
- [7] Nadhem J AlFardan and Kenneth G Paterson. Lucky thirteen: Breaking the tls and dtls record protocols. In *IEEE Symposium on Security and Privacy*, 2013.
- [8] Bernhard Amann, Robin Sommer, Matthias Vallentin, and Seth Hall. No Attack Necessary: The Surprising Dynamics of SSL Trust Relationships. 2013.
- [9] Ronald E Anderson. Acm code of ethics and professional conduct. *Communications of the ACM*, 35(5):94–99, 1992.
- [10] Target Press Announcement. Target confirms unauthorized access to payment card data in u.s. stores, December 2013. Online at <http://pressroom.target.com/news/target-confirms-unauthorized-access-to-payment-card-data-in-u-s-stores>.

- [11] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.
- [12] Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Information Hiding*, pages 245–257. Springer, 2001.
- [13] James Ball, Bruce Schneier, and Glenn Greenwald. Nsa and gchq target tor network that protects anonymity of web users, October 2013. Online at <http://www.theguardian.com/world/2013/oct/04/nsa-gchq-attack-tor-network-encryption>.
- [14] Robert Beverly, Simson Garfinkel, and Greg Cardwell. Forensic carving of network packets and associated data structures. *digital investigation*, 8:S78–S89, 2011.
- [15] Stefanie Beyer, Martin Mulazzani, Sebastian Schrittwieser, Markus Huber, and Edgar Weippl. Towards fully automated digital alibis with social interaction. In *Tenth Annual IFIP WG 11.9 International Conference on Digital Forensics*, 1 2014.
- [16] Stevens Le Blond, Pere Manils, Chaabane Abdelberi, Mohamed Ali Dali Kaafar, Claude Castelluccia, Arnaud Legout, and Walid Dabbous. One bad apple spoils the bunch: exploiting p2p applications to trace and profile tor users. *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*, 2011.
- [17] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84. ACM, 2004.
- [18] Frank Breitingner, Georgios Stivaktakis, and Harald Baier. Frash: A framework to test algorithms of similarity hashing. *Digital Investigation*, 10:S50–S58, 2013.
- [19] D Brezinski and Tom Killalea. Guidelines for evidence collection and archiving. *Request For Comments*, 3227, 2002.
- [20] Elie Bursztein, Chinmay Soman, Dan Boneh, and John C Mitchell. Sessionjuggler: secure web login from an untrusted terminal using session hijacking. In *Proceedings of the 21st international conference on World Wide Web*, pages 321–330. ACM, 2012.
- [21] Brian Carrier. *File system forensic analysis*, volume 3. Addison-Wesley Boston, 2005.
- [22] Eoghan Casey and Gerasimos J Stellatos. The impact of full disk encryption on digital forensics. *ACM SIGOPS Operating Systems Review*, 42(3):93–98, 2008.
- [23] A. Castiglione, G. Cattaneo, R. De Prisco, A. De Santis, and K. Yim. How to forge a digital alibi on Mac OS X. *Multidisciplinary Research and Practice for Information Systems*, pages 430–444, 2012.

- [24] Aniello Castiglione, Giuseppe Cattaneo, Giancarlo De Maio, Alfredo De Santis, Gerardo Costabile, and Mattia Epifani. The forensic analysis of a false digital alibi. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 114–121. IEEE, 2012.
- [25] Ellick Chan, Shivaram Venkataraman, Francis David, Amey Chaugule, and Roy Campbell. Forenscope: A framework for live forensics. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 307–316. ACM, 2010.
- [26] Jeremy Clark and Paul C. van Oorschot. Sok: Ssl and https: Revisiting past challenges and evaluating certificate trust model enhancements. 2013.
- [27] MI Cohen. Pyflag—an advanced network forensic framework. *digital investigation*, 5:S112–S120, 2008.
- [28] Josh Constine. Dropbox now has 175 million users, up from 100m in november 2012, July 2013. Online at <http://techcrunch.com/2013/07/09/dropbox-dbx-conference/>.
- [29] John W Creswell. Educational research: Planning, conducting, and evaluating quantitative and qualitative research. 2002.
- [30] George Danezis and Richard Clayton. Introducing traffic analysis, 2007.
- [31] A. De Santis, A. Castiglione, G. Cattaneo, G. De Maio, and M. Ianulardo. Automated construction of a false digital alibi. *Availability, Reliability and Security for Business, Enterprise and Health Information Systems*, pages 359–373, 2011.
- [32] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- [33] Dan DeFelippi. Dropship on github, April 2011. Online at <https://github.com/driverdan/dropship>.
- [34] Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.
- [35] Whitfield Diffie and Martin Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.
- [36] Roger Dingledine and Nick Mathewson. Anonymity Loves Company: Usability and the Network Effect. In *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, June 2006.
- [37] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

- [38] Idilio Drago, Marco Mellia, Maurizio M Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside dropbox: understanding personal cloud storage services. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 481–494. ACM, 2012.
- [39] Thai Duong and Juliano Rizzo. Here come the XOR Ninjas. *Ekoparty*, 2011.
- [40] Thai Duong and Juliano Rizzo. The crime attack. *Ekoparty*, 2012.
- [41] Zakir Durumeric, James Kasten, Michael Bailey, and Alex Halderman. Analysis of the HTTPS Certificate Ecosystem. In *Internet Measurement Conference (IMC)*, 2013.
- [42] Donald Eastlake and Paul Jones. Secure hash algorithm 1 (sha1), 2001.
- [43] Peter Eckersley. How Unique is Your Web Browser? In *Privacy Enhancing Technologies*, pages 1–18. Springer, 2010.
- [44] E. ECMAScript, European Computer Manufacturers Association, et al. ECMAScript Language Specification. Online at <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.
- [45] Noah Edelstein. Feature evolution and support for the skype desktop api, Nov 2013. Online at <http://blogs.skype.com/2013/11/06/feature-evolution-and-support-for-the-skype-desktop-api/>.
- [46] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 73–84. ACM, 2013.
- [47] Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith, Lars Baumgärtner, and Bernd Freisleben. Why eve and mallory love android: An analysis of android ssl (in) security. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 50–61. ACM, 2012.
- [48] Simson Garfinkel. Lessons learned writing digital forensics tools and managing a 30tb digital evidence corpus. *Digital Investigation*, 9:S80–S89, 2012.
- [49] Simson Garfinkel, Paul Farrell, Vassil Roussev, and George Dinolt. Bringing science to digital forensics with standardized forensic corpora. *digital investigation*, 6:S2–S11, 2009.
- [50] Simson Garfinkel, Alex Nelson, Douglas White, and Vassil Roussev. Using purpose-built functions and block hashes to enable small block and sub-file forensics. *digital investigation*, 7:S13–S23, 2010.
- [51] Simson L Garfinkel. Automating disk forensic processing with sleuthkit, xml and python. In *Systematic Approaches to Digital Forensic Engineering, 2009. SADFE'09. Fourth International IEEE Workshop on*, pages 73–84. IEEE, 2009.

- [52] Simson L Garfinkel. Digital forensics research: The next 10 years. *Digital Investigation*, 7:S64–S73, 2010.
- [53] Simson L Garfinkel. Digital media triage with bulk data analysis and `bulk_extractor`. *Computers & Security*, 2012.
- [54] Simson L Garfinkel and Abhi Shelat. Remembrance of data passed: A study of disk sanitization practices. *Security & Privacy, IEEE*, 1(1):17–27, 2003.
- [55] Behrad Garmany and Tilo Müller. Prime: private rsa infrastructure for memory-less encryption. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 149–158. ACM, 2013.
- [56] Dan Goodin. Stop using nsa-influenced code in our products, rsa tells customers, Sep 2013. Online at <http://arstechnica.com/security/2013/09/stop-using-nsa-influence-code-in-our-product-rsa-tells-customers/>.
- [57] Johannes Gotzfried and Tilo Muller. Armored: Cpu-bound encryption for android-driven arm devices. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 161–168. IEEE, 2013.
- [58] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Proceedings of the USENIX Security Symposium*. USENIX Association, 2008.
- [59] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 491–500. ACM, 2011.
- [60] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *Security & Privacy, IEEE*, 8(6):40–47, 2010.
- [61] Andrew Hoog. *Android forensics: investigation, analysis and mobile security for Google Android*. Access Online via Elsevier, 2011.
- [62] Nicholas Hopper, Luis von Ahn, and John Langford. Provably secure steganography. *Computers, IEEE Transactions on*, 58(5):662–676, 2009.
- [63] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The parrot is dead: Observing unobservable network communications. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, 2013.
- [64] Markus Huber, Martin Mulazzani, Manuel Leithner, Sebastian Schrittwieser, Gilbert Wondracek, and Edgar Weippl. Social snapshots: Digital forensics for online social networks. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 113–122. ACM, 2011.

- [65] Markus Huber, Martin Mulazzani, and Edgar Weippl. Who on earth is “mr. cypher”: Automated friend injection attacks on social networking sites. In *Security and Privacy—Silver Linings in the Cloud*, pages 80–89. Springer, 2010.
- [66] Markus Huber, Martin Mulazzani, Edgar Weippl, Gerhard Kitzler, and Sigrun Goluch. Friend-in-the-middle attacks: Exploiting social networking sites for spam. *Internet Computing, IEEE*, 15(3):28–34, 2011.
- [67] Markus Huber, Martin Mulazzani, and Edgar R. Weippl. Tor http usage and information leakage. In *Proceedings of IFIP CMS 2010*, pages 245–255. Springer, 2010.
- [68] IEEE. Code of ethics. Online at <http://www.ieee.org/about/corporate/governance/p7-8.html>.
- [69] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: traffic correlation on tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 337–348. ACM, 2013.
- [70] Stefan Katzenbeisser, Fabien AP Petitcolas, et al. *Information hiding techniques for steganography and digital watermarking*, volume 316. Artech house Norwood, 2000.
- [71] Karen Kent, Suzanne Chevalier, Tim Grance, and Hung Dang. Guide to integrating forensic techniques into incident response. *NIST Special Publication 800-86*, 2006.
- [72] Dhiru Kholia and Przemysław Wegrzyn. Looking inside the (drop) box. In *Proceedings of the 7th USENIX conference on Offensive Technologies (WOOT)*. USENIX Association, 2013.
- [73] Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. *digital investigation*, 3:91–97, 2006.
- [74] Brian Krebs. Adobe breach impacted at least 38 million users. Online at <http://krebsonsecurity.com/2013/10/adobe-breach-impacted-at-least-38-million-users/>.
- [75] Adam Langley. Overclocking ssl. imperial violet blog, june 25, 2010. Online at <https://www.imperialviolet.org/2010/06/25/overclocking-ssl.html>.
- [76] Tobias Lauinger, Veikko Pankakoski, Davide Balzarotti, and Engin Kirda. Honeybot, your man in the middle for automated social engineering. In *LEET’10, 3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats, San Jose*, 2010.
- [77] Marc Liberatore, Robert Erdely, Thomas Kerle, Brian Neil Levine, and Clay Shields. Forensic investigation of peer-to-peer file sharing networks. *digital investigation*, 7:S95–S103, 2010.

- [78] Karsten Loesing. Measuring the tor network from public directory information. *Proc. HotPETS, Seattle, WA*, 2009.
- [79] Karsten Loesing, Steven J Murdoch, and Roger Dingledine. A case study on measuring statistical data in the tor anonymity network. In *Financial Cryptography and Data Security*, pages 203–215. Springer, 2010.
- [80] Lodovico Marziale, Golden G Richard III, and Vassil Roussev. Massive threading: Using gpus to increase the performance of digital forensics tools. *digital investigation*, 4:73–81, 2007.
- [81] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining light in dark places: Understanding the tor network. In *Privacy Enhancing Technologies*, pages 63–76. Springer, 2008.
- [82] Cade Metz. Dropbox snuffs open code that bypassed file-sharing controls, April 2011. Online at http://www.theregister.co.uk/2011/04/26/dropbox_moves_to_squash_open_source_dropship_project/.
- [83] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 97–108. ACM, 2012.
- [84] Sean Morrissey. *iOS Forensic Analysis: for iPhone, iPad, and iPod touch*. Apress, 2010.
- [85] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. Fingerprinting information in javascript implementations. In *Proceedings of Web*, volume 2, 2011.
- [86] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in html5. *Proceedings of W2SP*, 2012.
- [87] Martin Mulazzani. Anonymity & monitoring. *Master thesis, Vienna University of Technology*, 2009.
- [88] Martin Mulazzani, Markus Huber, and Edgar R. Weippl. Anonymity and monitoring: How to monitor the infrastructure of an anonymity system. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(5):539–546, 9 2010.
- [89] Martin Mulazzani, Sebastian Neuner, Peter Kieseberg, Markus Huber, Sebastian Schrittwieser, and Edgar R. Weippl. Quantifying windows file slack in size and stability. In *Ninth Annual IFIP WG 11.9 International Conference on Digital Forensics*, 1 2013.
- [90] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, and Edgar R. Weippl. Fast and reliable browser identification with javascript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)*, 5 2013.

- [91] Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar R. Weippl. Dark clouds on the horizon: Using cloud storage as attack vector and online slack space. In *Proceedings of the USENIX Security Symposium*. USENIX Association, 2011.
- [92] Martin Mulazzani, Thomas Unger, Edgar R. Weippl, Sebastian Schrittwieser, Markus Huber, and Dominik Frühwirt. Shpf: Enhancing http(s) session security with browser fingerprinting. In *Proceedings of the Eighth International Conference on Availability, Reliability and Security (ARES)*, 9 2013.
- [93] Tilo Müller, Andreas Dewald, and Felix C Freiling. Aesse: a cold-boot resistant implementation of aes. In *Proceedings of the Third European Workshop on System Security*, pages 42–47. ACM, 2010.
- [94] Tilo Müller, Felix C Freiling, and Andreas Dewald. Tresor runs encryption securely outside ram. In *USENIX Security Symposium*, 2011.
- [95] Tilo Müller and Michael Spreitzenbarth. Frost – forensic recovery of scrambled telephones. In *Applied Cryptography and Network Security*, pages 373–388. Springer, 2013.
- [96] Steven J Murdoch and George Danezis. Low-cost traffic analysis of tor. In *Security and Privacy, 2005 IEEE Symposium on*, pages 183–195. IEEE, 2005.
- [97] Derek Newton. Dropbox authentication: insecure by design, April 2011. Online at <http://dereknewton.com/2011/04/dropbox-authentication-static-host-ids/>.
- [98] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *IEEE Symposium on Security and Privacy*, 2013.
- [99] The open web application security Project. Owasp top 10. 2013.
- [100] Andrea Peterson, Barton Gellman, and Ashkan Soltani. Yahoo to make ssl encryption the default for webmail users. finally., Oct. 2013. Online at <http://www.washingtonpost.com/blogs/the-switch/wp/2013/10/14/yahoo-to-make-ssl-encryption-the-default-for-webmail-users-finally/>.
- [101] Jon Postel. Internet protocol. *Request For Comments*, 791, 1981.
- [102] Jon Postel. Transmission control protocol. *Request For Comments*, 793, 1981.
- [103] Kevin Poulsen. *Kingpin: How one hacker took over the billion-dollar cybercrime underground*. Random House Digital, Inc., 2012.
- [104] Angelo Prado, Neal Harris, and Yoel Gluck. SSL, gone in 30 seconds - a BREACH beyond CRIME. 2013.

- [105] Darren Quick and Kim-Kwang Raymond Choo. Dropbox analysis: Data remnants on user machines. *Digital Investigation*, 2013.
- [106] Scott Renfro. Secure browsing by default, Jul. 2013. Online at <https://www.facebook.com/notes/facebook-engineering/secure-browsing-by-default/10151590414803920>.
- [107] Thorsten Ries, Andriy Panchenko, Thomas Engel, et al. Comparison of low-latency anonymous communication systems: practical usage and performance. In *Proceedings of the Ninth Australasian Information Security Conference-Volume 116*, pages 77–86. Australian Computer Society, Inc., 2011.
- [108] Ronald Rivest. The md5 message-digest algorithm. *Request For Comments*, 1321, 1992.
- [109] Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs: The linux b-tree filesystem. *Trans. Storage*, 9(3), August 2013.
- [110] Vassil Roussev. Data fingerprinting with similarity digests. In *Advances in Digital Forensics VI*, pages 207–226. Springer, 2010.
- [111] Vassil Roussev. An evaluation of forensic similarity hashes. *digital investigation*, 8:S34–S41, 2011.
- [112] Vassil Roussev and Candice Quates. File fragment encoding classification—an empirical approach. *Digital Investigation*, 10:S69–S77, 2013.
- [113] Joanna Rutkowska. Beyond the cpu: Defeating hardware based ram acquisition. *Proceedings of BlackHat DC 2007*, 2007.
- [114] Sam Schillace. Default https access for gmail. Online at <http://gmailblog.blogspot.co.at/2010/01/default-https-access-for-gmail.html>.
- [115] Bruce Schneier. Applied cryptography, 1996.
- [116] Dan Shumow and Niels Ferguson. On the possibility of a back door in the nist sp800-90 dual_ec_prng. *27th Annual International Cryptology Conference (CRYPTO)*, 2007. Informal presentation.
- [117] Cliord P Stoll. The cuckoo’s egg: Tracing a spy through the maze of computer espionage, 1989.
- [118] Johannes Stüttgen and Michael Cohen. Anti-forensic resilient memory acquisition. *Digital Investigation*, 10:S105–S115, 2013.
- [119] Twitter. Securing your twitter experience with https, Feb. 2012. Online at <https://blog.twitter.com/2012/securing-your-twitter-experience-with-https>.

- [120] Chris Wacek, Henry Tan, Kevin Bauer, and Micah Sherr. An empirical evaluation of relay selection in tor. In *Proceedings of the Network and Distributed Security Symposium (NDSS)(February 2013)*, 2013.
- [121] Qiyang Wang, Xun Gong, Giang TK Nguyen, Amir Houmansadr, and Nikita Borisov. Censorspoof: asymmetric communication using ip spoofing for censorship-resistant web browsing. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 121–132. ACM, 2012.
- [122] Samuel D Warren and Louis D Brandeis. The right to privacy. *Harvard law review*, 4(5):193–220, 1890.
- [123] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. Stegotorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 109–120. ACM, 2012.
- [124] Kleine Zeitung Wolfsberg. Facebook rückte daten über neonazi heraus, December 2011. Online at <http://www.kleinezeitung.at/kaernten/wolfsberg/2911087/facebook-rueckte-daten-ueber-neonazi-heraus.story>.
- [125] York Yannikos, Lukas Graner, Martin Steinebach, and Christian Winter. On the availability of data corpora for digital forensic education and research. In *Tenth Annual IFIP WG 11.9 International Conference on Digital Forensics*, 1 2014.
- [126] York Yannikos, Christian Winter, and Markus Schneider. Synthetic data creation for forensic tool testing: Improving performance of the 3lspg framework. In *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*, pages 613–619. IEEE, 2012.
- [127] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martin Abadi. Host Fingerprinting and Tracking on the Web: Privacy and Security Implications. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS 2012)*, February 2012.
- [128] Joel Young, Simson Garfinkel, Kristina Foster, and Kevin Fairbanks. Distinct sector hashes for target file detection. 2012.
- [129] Yihua Zhang and Marina Blanton. Efficient dynamic provable possession of remote data via balanced update trees. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 183–194. ACM, 2013.
- [130] Qingji Zheng and Shouhuai Xu. Secure and efficient proof of storage with deduplication. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 1–12. ACM, 2012.

Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space

The paper *Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space* was published at the USENIX Security Symposium 2011 in San Francisco²⁵.

You can find the paper online at http://static.usenix.org/events/sec11/tech/full_papers/Mulazzani6-24-11.pdf. USENIX also offers video recordings²⁶ as well as the slides of the presentation²⁷ online.

²⁵<https://www.usenix.org/conference/usenixsecurity11>

²⁶<https://www.usenix.org/conference/usenix-security-11/dark-clouds-horizon-using-cloud-storage-attack-vector-and-online-slack>

²⁷<http://static.usenix.org/events/sec11/tech/slides/mulazzani.pdf>

Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space

Martin Mulazzani
SBA Research

Sebastian Schrittwieser
SBA Research

Manuel Leithner
SBA Research

Markus Huber
SBA Research

Edgar Weippl
SBA Research

Abstract

During the past few years, a vast number of online file storage services have been introduced. While several of these services provide basic functionality such as uploading and retrieving files by a specific user, more advanced services offer features such as shared folders, real-time collaboration, minimization of data transfers or unlimited storage space. Within this paper we give an overview of existing file storage services and examine Dropbox, an advanced file storage solution, in depth. We analyze the Dropbox client software as well as its transmission protocol, show weaknesses and outline possible attack vectors against users. Based on our results we show that Dropbox is used to store copyright-protected files from a popular filesharing network. Furthermore Dropbox can be exploited to hide files in the cloud with unlimited storage capacity. We define this as *online slack space*. We conclude by discussing security improvements for modern online storage services in general, and Dropbox in particular. To prevent our attacks cloud storage operators should employ data possession proofs on clients, a technique which has been recently discussed only in the context of assessing trust in cloud storage operators.

1 Introduction

Hosting files on the Internet to make them retrievable from all over the world was one of the goals when the Internet was designed. Many new services have been introduced in recent years to host various type of files on centralized servers or distributed on client machines. Most of today's online storage services follow a very simple design and offer very basic features to their users. From the technical point of view, most of these services are based on existing protocols such as the well known FTP [28], proprietary protocols or WebDAV [22], an extension to the HTTP protocol.

With the advent of cloud computing and the shared

usage of resources, these centralized storage services have gained momentum in their usage, and the number of users has increased heavily. In the special case of online cloud storage the shared resource can be disc space on the provider's side, as well as network bandwidth on both the client's and the provider's side. An online storage operator can safely assume that, besides private files as well as encrypted files that are specific and different for every user, a lot of files such as setup files or common media data are stored and used by more than one user. The operator can thus avoid storing multiple physical copies of the same file (apart from redundancy and backups, of course). To the best of our knowledge, Dropbox is the biggest online storage service so far that implements such methods for avoiding unnecessary traffic and storage, with millions of users and billions of files [24]. From a security perspective, however, the shared usage of the user's data raises new challenges. The clear separation of user data cannot be maintained to the same extent as with classic file hosting, and other methods have to be implemented to ensure that within the pool of shared data only authorized access is possible. We consider this to be the most important challenge for efficient and secure "cloud-based" storage services. However, not much work has been previously done in this area to prevent unauthorized data access or information leakage.

We focus our work on Dropbox because it is the biggest cloud storage provider that implements shared file storage on a large scale. New services will offer similar features with cost and time savings on both the client and the operators side, which means that our findings are of importance for all upcoming cloud storage services as well. Our proposed measurements to prevent unauthorized data access and information leakage, exemplarily demonstrated with Dropbox, are not specific to Dropbox and should be used for other online storage services as well. We believe that the number of cloud-based storage

operators will increase heavily in the near future.

Our contribution in this paper is to:

- Document the functionality of an advanced cloud storage service with server-side data deduplication such as Dropbox.
- Show under what circumstances unauthorized access to files stored within Dropbox is possible.
- Assess if Dropbox is used to store copyright-protected material.
- Define online slack space and the unique problems it creates for the process of a forensic examination.
- Explain countermeasures, both on the client and the server side, to mitigate the resulting risks from our attacks for user data.

The remainder of this paper is organized as follows. Related work and the technical details of Dropbox are presented in Section 2. In Section 3 we introduce an attack on files stored at Dropbox, leading to information leakage and unauthorized file access. Section 4 discusses how Dropbox can be exploited by an adversary in various other ways while Section 5 evaluates the feasibility of these attacks. We conclude by proposing various techniques to reduce the attack surface for online storage providers in Section 6.

2 Background

This section describes the technical details and implemented security controls of Dropbox, a popular cloud storage service. Most of the functionality is attributed to the new cloud-paradigm, and not specific to Dropbox. In this paper we use the notion of cloud computing as defined in [9], meaning applications that are accessed over the Internet with the hardware running in a data center not necessarily under the control of the user:

“Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services.” ... “The datacenter hardware and software is what we will call a Cloud.”

In the following we describe Dropbox and related literature on cloud storage.

2.1 Dropbox

Since its initial release in September 2008 Dropbox has become one of the most popular cloud storage provider on the Internet. It has 10 million users and

stores more than 100 billion files as of May 2011 [2] and saves 1 million files every 5 minutes [3]. Dropbox is mainly an online storage service that can be used to create online backups of files, and one has access to files from any computer or similar device that is connected to the Internet. A desktop client software available for different operating systems keeps all the data in a specified directory in sync with the servers, and synchronizes changes automatically among different client computers by the same user. Subfolders can be shared with other Dropbox users, and changes in shared folders are synced and pushed to every Dropbox account that has been given access to that shared folder. Large parts of the Dropbox client are written in Python.

Internally, Dropbox does not use the concept of files, but every file is split up into chunks of up to 4 megabytes in size. When a user adds a file to his local Dropbox folder, the Dropbox client application calculates the hash values of all the chunks of the file using the SHA-256 algorithm [19]. The hash values are then sent to the server and compared to the hashes already stored on the Dropbox servers. If a file does not exist in their database, the client is requested to upload the chunks. Otherwise the corresponding chunk is not sent to the server because a copy is already stored. The existing file on the server is instead linked to the Dropbox account. This approach allows Dropbox to save traffic and storage costs, and users benefit from a faster syncing process if files are already stored on the Dropbox servers. The software uses numerous techniques to further enhance efficiency e.g., delta encoding, to only transfer those parts of the files that have been modified since the last synchronization with the server. If by any chance two distinct files should have the same hash value, the user would be able to access other users content since the file stored on the servers is simply linked to the users Dropbox account. However, the probability of a coincidental collision in SHA-256 is negligibly small.

The connections between the clients and the Dropbox servers are secured with SSL. Uploaded data is encrypted with AES-256 and stored on Amazons S3 storage service that is part of the Amazon Web Services (AWS) [1]. The AES key is user independent and only secures the data during storage at Amazon S3, while transfer security relies on SSL. Our research on the transmission protocol showed that data is directly sent to Amazon EC2 servers. Therefore, encryption has to be done by EC2 services. We do not know where the keys are stored and if different keys are used for each file chunk. However, the fact that encryption and storage is done at the same place seems questionable to us, as

Amazon is most likely able to access decryption keys ¹.

After uploading the chunks that were not yet in the Dropbox storage system, Dropbox calculates the hash values on their servers to validate the correct transmission of the file, and compares the values with the hash values sent by the client. If the hash values do not match, the upload process of the corresponding chunk is repeated. The drawback of this approach is that the server can only calculate the hash values of actually uploaded chunks; it is not able to validate the hash values of files that were already on Dropbox and that were provided by the client. Instead, it *trusts the client software* and links the chunk on the server to the Dropbox account. Therefore, spoofing the hash value of a chunk added to the local Dropbox folder allows a malicious user to access files of other Dropbox users, given that the SHA-256 hash values of the file's chunks are known to the attacker.

Due to the recent buzz in cloud computing many companies compete in the area of cloud storage. Major operating system companies have introduced their services with integration into their system, while small startups can compete by offering cross-OS functionality or more advanced security features. Table 1 compares a selection of popular file storage providers without any claim for completeness. Note that “encrypted storage” means that the file is encrypted locally before it is sent to the cloud storage provider and shared storage means that it is possible to share files and folders between users.

2.2 Related Work

Related work on secure cloud storage focuses mainly on determining if the cloud storage operator is still in possession of the client's file, and if it has been modified. An interesting survey on the security issues of cloud computing in general can be found in [30]. A summary of attacks and new security problems that arise with the usage of cloud computing has been discussed in [17]. In a paper by Shacham et al. [11] it was demonstrated that it is rather easy to map the internal infrastructure of a cloud storage operator. Furthermore they introduced co-location attacks where they have been able to place a virtual machine under their control on the same hardware as a target system, resulting in information leakage and possible side-channel attacks on a virtual machine.

¹Independently found and confirmed by Christopher Soghoian [5] and Ben Adida [4]

Early publications on file retrievability [25, 14] check if a file can be retrieved from an untrusted third party without retransmitting the whole file. Various papers propose more advanced protocols [11, 12, 20] to ensure that an untrusted server has the original file without retrieving the entire file, while maintaining an overall overhead of $O(1)$. Extensions have been published that allow checking of dynamic data, for example Wang et al. [32] use a Merkle hash tree which allows a third party auditor to audit for malicious providers while allowing public verifiability as well as dynamic data operations. The use of algebraic signatures was proposed in [29], while a similar approach based on homomorphic tokens has been proposed in [31]. Another cryptographic tree structure is named “Cryptree” [23] and is part of the Wuala online storage system. It allows strong authentication by using encryption and can be used for P2P networks as well as untrusted cloud storage. The HAIL system proposed in [13] can be seen as an implementation of a service-oriented version of RAID across multiple cloud storage operators.

Harnik et al. describe similar attacks in a recent paper [24] on cloud storage services which use server-side data deduplication. They recommend using encryption to stop server-side data deduplication, and propose a randomized threshold in environments where encryption is undesirable. However, they do not employ client-side data possession proofs to prevent hash manipulation attacks, and have no practical evaluation for their attacks.

3 Unauthorized File Access

In this section we introduce three different attacks on Dropbox that enable access to arbitrary files given that the hash values of the file, respectively the file chunks, are known. If an arbitrary cloud storage service relies on the client for hash calculation in server-side data deduplication implementations, these attacks are applicable as well.

3.1 Hash Value Manipulation Attack

For the calculation of SHA-256 hash values, Dropbox does not use the `hashlib` library which is part of Python. Instead it delegates the calculation to OpenSSL [18] by including a wrapper library called `NCrypto` [6]. The Dropbox clients for Linux and Mac OS X dynamically link to libraries such as `NCrypto` and do not verify their integrity before using them. We modified the publicly available source code of `NCrypto` so that it replaces the hash value that was calculated by OpenSSL with our own value (see Figure 1), built it

Name	Protocol	Encrypted transmission	Encrypted storage	Shared storage
Dropbox	proprietary	yes	no	yes
Box.net	proprietary	yes	yes (enterprise only)	yes
Wuala	Cryptree	yes	yes	yes
TeamDrive	many	yes	yes	yes
SpiderOak	proprietary	yes	yes	yes
Windows Live Skydrive	WebDAV	yes	no	yes
Apple iDisk	WebDAV	no	no	no
Ubuntu One	u1storage	yes	no	yes

Table 1: Online Storage Providers

and replaced the library that was shipped with Dropbox. The Dropbox client does not detect this modification and transmits for any new file in the local Dropbox the modified hash value to the server. If the transmitted hash value does not exist in the server's database, the server requests the file from the client and tries to verify the hash value after the transmission. Because of our manipulation on the client side, the hash values will not match and the server would detect that. The server would then re-request the file to overcome an apparent transmission error.

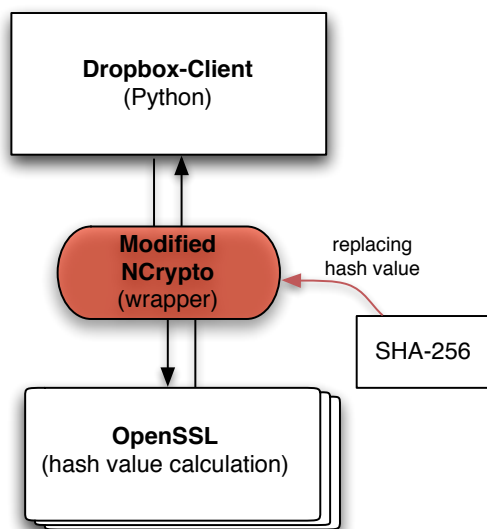


Figure 1: Hash Value Manipulation Attack

However, if the hash value is already in the server's databases the server trusts the hash value calculation of the client and does not request the file from the client. Instead it links the corresponding file/chunk to the Dropbox account. Due to the manipulation of the hash value we thus got unauthorized access to arbitrary files.

This attack is completely undetectable to the user. If

the attacker already knows the hash values, he can download files directly from the Dropbox server and no interaction with the client is needed which could be logged or detected on the client side. The victim is unable to notice this in any way, as no access to his computer is required. Even for the Dropbox servers this unauthorized access to arbitrary files is not detectable because they believe the attacker already owns the files, and simply added them to their local Dropbox folder.

3.2 Stolen Host ID Attack

During setup of the Dropbox client application on a computer or smartphone, a unique **host ID** is created which links that specific device to the owner's Dropbox account. The client software does not store username and password. Instead, the host ID is used for client and user authentication. It is a random looking 128-bit key that is calculated by the Dropbox server from several seeding values provided by the client (e.g. username, exact date and time). The algorithm is not publicly known. This linking requires the user's account credentials. When the client on that host is successfully linked, no further authentication is required for that host as long as the Dropbox software is not removed.

If the host ID is stolen by an attacker, extracted by malware or by social engineering, all the files on that users accounts can be downloaded by the attacker. He simply replaces his own host ID with the stolen one, re-syncs Dropbox and consequently downloads every file.

3.3 Direct Download Attack

Dropbox's transmission protocol between the client software and the server is built on HTTPS. The client software can request file chunks from `https://dl-clientXX.dropbox.com/retrieve` (where XX is replaced by consecutive numbers) by submitting the SHA-256 hash value of the file chunk and a valid host ID as HTTPS POST data. Surprisingly, the host ID doesn't even need to be linked to a Dropbox account that owns

the corresponding file. Any valid host ID can be used to request a file chunk as long as the hash value of the chunk is known and the file is stored at Dropbox. As we will see later, Dropbox hardly deletes any data. It is even possible to just create an HTTPS request with any valid host ID, and the hash value of the chunk to be downloaded. This approach could be easily detected by Dropbox because a host ID that was not used to upload a chunk or is known to be in possession of the chunk would try to download it. By contrast the hash manipulation attack described above is undetectable for the Dropbox server, and (minor) changes to the core communication protocol would be needed to detect it.

3.4 Attack Detection

To sum up, when an attacker is able to get access to the content of the client database, he is able to download all the files of the corresponding Dropbox account directly from the Dropbox servers. No further access to the victim's system is needed, and in the simplest case only the host ID needs to be sent to the attacker. An alternative approach for the attacker is to access only specific files, by obtaining only the hash values of the file. The owner of the files is unable to detect that the attacker accessed the files, for all three attacks. From the cloud storage service operators point of view, the stolen host-ID attack as well as the direct download attack are detectable to some extent. We discuss some countermeasures in section 6. However, by using the hash manipulation attack the attacker can avoid detection completely, as this form of unauthorized access looks like the attacker already owns the file to Dropbox. Table 2 gives an overview of all of the different attacks that can lead to unauthorized file access and information leakage ².

4 Attack Vectors and Online Slack Space

This section discusses known attack techniques to exploit cloud storage and Dropbox on a large scale. It outlines already known attack vectors, and how they could be used with the help of Dropbox, or any other cloud storage service with weak security. Most of them can have a severe impact and should be considered in the threat model of such services.

²We communicated with Dropbox and reported our findings prior to publishing this paper. They implemented a temporary fix to prevent these types of attacks and will include a permanent solution in future versions.

4.1 Hidden Channel, Data Leakage

The attacks discussed above can be used in numerous ways to attack clients, for example by using Dropbox as a drop zone for important and possibly sensitive data. If the victim is using Dropbox (or any other cloud storage services which is vulnerable to our discovered attack) these services might be used to exfiltrate data a lot stealthier and faster with a covert channel than using regular covert channels [16]. The amount of data that needs to be sent over the covert channel would be reduced to a single host ID or the hash values of specific files instead of the full file. Furthermore the attacker could copy important files to the Dropbox folder, wait until they are stored on the cloud service and delete them again. Afterwards he transmits the hash values to the attacker and the attacker then downloads these files directly from Dropbox. This attack requires that the attacker is able to execute code and has access to the victim's file system e.g. by using malware. One might argue that these are tough preconditions for this scenario to work. However, as in example, in the case of corporate firewalls this kind of data leakage is much harder to detect as all traffic with Dropbox is encrypted with SSL and the transfers would blend in perfectly with regular Dropbox activity, since Dropbox itself is used for transmitting the data. Currently the client has no control measures to decide upon which data might get stored in the Dropbox folder. The scheme for leaking information and transmitting data to an attacker is depicted in Figure 2.

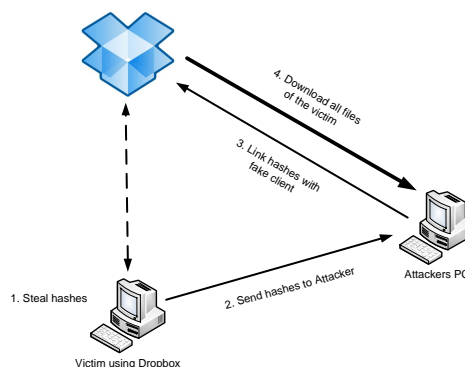


Figure 2: Covert Channel with Dropbox

4.2 Online Slack Space

Uploading a file works very similarly to downloading with HTTPS (as described above, see section 3.3). The client software uploads a chunk to Dropbox by calling `https://dl-clientXX.dropbox.com/store` with the hash value and the host ID as HTTPS POST data along with the actual data. After the upload is finished, the client

Method	Detectability	Consequences
Hash Value Manipulation Attack	Undetectable	Unauthorized file access
Direct Download Attack	Dropbox only	Unauthorized file access
Stolen Host ID Attack	Dropbox only	Get all user files

Table 2: Variants of the Attack

software links the uploaded files to the host ID with another HTTPS request. The updated or newly added files are now pushed to all computers of the user, and to all other user accounts if the folder is a shared folder.

A modified client software can upload files without limitation, if the linking step is omitted. Dropbox can thus be used to store data without decreasing the available amount of data. We define this as *online slack space* as it is similar to regular slack space [21] from the perspective of a forensic examiner where information is hidden in the last block of files on the filesystem that are not using the entire block. Instead of hiding information in the last block of a file, data is hidden in Dropbox chunks that are not linked to the attackers account. If used in combination with a live CD operating system, no traces are left on the computer that could be used in the forensic process to infer the existence of that data once the computer is powered down. We believe that there is no limitation on how much information could be hidden, as the exploited mechanisms are the same as those which are used by the Dropbox application.

4.3 Attack Vector

If the host ID is known to an attacker, he can upload and link arbitrary files to the victim's Dropbox account. Instead of linking the file to his account with the second HTTPS request, he can use an arbitrary host ID with which to link the file. In combination with an exploit of the operating system file preview functions, e.g. on one of the recent vulnerabilities in Windows³, Linux⁴, or MacOS⁵, this becomes a powerful exploitation technique. An attacker could use any 0-day weakness in the file preview of supported operating systems to execute code on the victim's computer, by pushing a manipulated file into his Dropbox folder and waiting for the user to open that directory. Social engineering could additionally be used to trick the victim into executing a file with a promising filename.

To get access to the host ID in the first place is tricky, and in any case access to the filesystem is needed in the first place. This however does not reduce the conse-

quences, as it is possible to store files remotely in other peoples Dropbox. A large scale infection using Dropbox is however very unlikely, and if an attacker is able to retrieve the host ID he already owns the system.

5 Evaluation

This section studies some of the attacks introduced. We evaluate whether Dropbox is used to store popular files from the filesharing network thepiratebay.org⁶ as well as how long data is stored in the previously defined online slack space.

5.1 Stored files on Dropbox

With the hash manipulation attack and the direct download attack described above it becomes possible to test if a given file is already stored on Dropbox. We used that to evaluate if Dropbox is used for storing filesharing files, as filesharing protocols like BitTorrent rely heavily on hashing for file identification. We downloaded the top 100 torrents from thepiratebay.org [7] as of the middle of September 2010. Unfortunately, BitTorrent uses SHA-1 hashes to identify files and their chunks, so the information in the .torrent file itself is not sufficient and we had to download parts of the content. As most of the files on BitTorrent are protected by copyright, we decided to download every file from the .torrent that lacks copyright protection to protect us from legal complaints, but are still sufficient to prove that Dropbox is used to store these kind of files. To further protect us against complaints based on our IP address, our BitTorrent client was modified to prevent upload of any data, as described similarly in [27]. We downloaded only the first 4 megabytes of any file that exceeds this size, as the first chunk is already sufficient to tell if a given file is stored on Dropbox or not using the hash manipulation attack.

We observed the following different types of files that were identified by the .torrent files:

- Copyright protected content such as movies, songs or episodes of popular series.
- "Identifying files" that are specific to the copyright protected material, such as sample files, screen captures or checksum files, but without copyright.

³Windows Explorer: CVE-2010-2568 or CVE-2010-3970

⁴Evince in Nautilus: CVE-2010-2640

⁵Finder: CVE-2006-2277

⁶Online at <http://thepiratebay.org>

- Static files that are part of many torrents, such as release group information files or links to websites.

Those “identifying files” we observed had the following extensions and information:

- *.nfo*: Contains information from the release group that created the .torrent e.g., list of files, installation instructions or detailed information and ratings for movies.
- *.srt*: Contains subtitles for video files.
- *.sfv*: Contains CRC32 checksums for every file within the .torrent.
- *.jpg*: Contains screenshots of movies or album covers.
- *.torrent*: The torrent itself contains the hash values of all the files, chunks as well as necessary tracker information for the clients.

In total from those top 100 torrent archives, 98 contained identifying files. We removed the two .torrents from our test set that did not contain such identifying files. 24 hours later we downloaded the newest entries from the top 100 list, to check how long it takes from the publication of a torrent until it is stored on Dropbox. 9 new torrents, mostly series, were added to the test set. In Table 3 we show in which categories they were categorized by thepiratebay.org.

Category	Quantity
Application	3
Game	5
Movie	64
Music	6
Series	29
Sum	107

Table 3: Distribution of tested .torrents

When we downloaded the “identifying files” from these 107 .torrent, they had in total approximately 460k seeders and 360k leechers connected (not necessarily disjoint), with the total number of complete downloads possibly much higher. For every .torrent file and every identifying file from the .torrent’s content we generated the sha256 hash value and checked if the files were stored on Dropbox, in total 368 hashes. If the file was bigger than 4 megabytes, we only generated the hash of the first chunk. Our script did not use the completely stealthy approach described above, but the less stealthy approach by creating an HTTPS request with a valid host ID as the overall stealthiness was in our case not an issue.

From those 368 hashes, 356 files were retrievable, only 12 hashes were unknown to Dropbox and the corresponding files were not stored on Dropbox. Those 12 files were linked to 8 .torrent files. The details:

- In one case the identifying file of the .torrent was not on Dropbox, but the .torrent file was.
- In three cases the .torrent file was not on Dropbox, but the identifying files were.
- In four cases the .nfo file was not on Dropbox, but other iIn fact, it might be the case that only one person uses Dropbox to store these files. identifying files from the same .torrent were.

This means that for every .torrent either the .torrent file, the content or both are easily retrievable from Dropbox once the hashes are known. Table 4 shows the numbers in details, where hit rate describes how many of them were retrievable from Dropbox.

File	Quantity	Hitrate	Hitrate rel.
.torrent:	107	106	99%
.nfo:	53	49	92%
others:	208	201	97%
In total:	368	356	97%

Table 4: Hit rate for filesharing

Furthermore we analyzed the age of the .torrents to see how quick Dropbox users are to download the .torrents and the corresponding content, and to upload everything to Dropbox. Most of the .torrent files were relatively young, as approximately 20 % of the top 100 .torrent files were less than 24 hours on piratebay before we were able to retrieve them from Dropbox. Figure 3 shows the distribution of age from all the .torrents:

5.2 Online Slack Space Evaluation

To assess if Dropbox could be used to hide files by uploading without linking them to any user account, we generated a set of 30 files with random data and uploaded them with the HTTPS request method. Furthermore we uploaded 55 files with a regular Dropbox account and deleted them right afterwards, to assess if Dropbox ever deletes old user data. We furthermore evaluated if there is some kind of garbage collection that removes files after a given threshold of time since the upload. The files were then downloaded every 24 hours and checked for consistency by calculating multiple hash functions and comparing the hashvalues. By using multiple files with various sizes and random content we minimized the likelihood of an unintended hash collision and avoided testing for a file that is stored by another user and thus

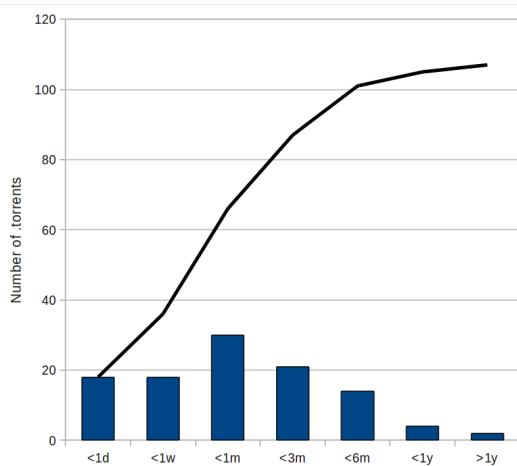


Figure 3: Age of .torrents

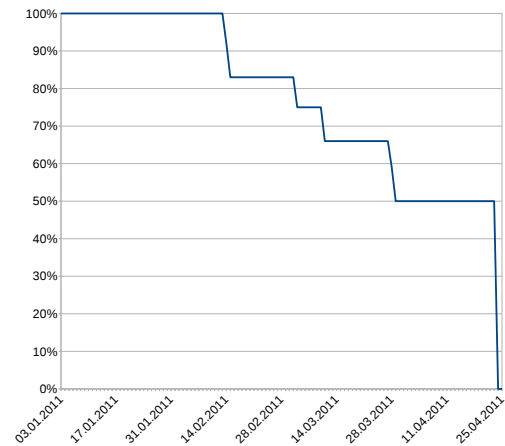


Figure 4: Online slack without linking over time

always retrievable. Table 5 summarizes the setup.

Method of upload	#	Testduration	Hirate
Regular folder	25	6 months	100%
Shared folder	30	6 months	100%
HTTPS request	30	>3 months	50%
In total:	85	—	100%

Table 5: Online slack experiments

Long term undelete: With the free account users can undo file modifications or undelete files through the webinterface from the last 30 days. With a so called “Pro” account (where the users pay for additional storage space and other features) undelete is available for all files and all times. We uploaded 55 files in total on October 7th 2010, 30 files in a shared folder with another Dropbox account and 25 files in an unshared folder. Until Dropbox fixed the HTTPS download attack at the end of April 2011, 100% have been constantly available. More then 6 months after uploading, all files were still retrievable, without exception.

Online slack: We uploaded 30 files of various sizes without linking them to any account with the HTTPS method at the beginning of January 2011. More then 4 weeks later, all files were still retrievable. When Dropbox fixed the HTTPS download attack in late April 2011, 50% of the files were still available. See Figure 4 for details.

5.3 Discussion

It surprised us that from every .torrent file, either the .torrent, the content or both could be retrieved from

Dropbox, especially considering that some of the .torrent files were only a few hours created before we retrieved them. 97% means that Dropbox is heavily used for storing files from filesharing networks. It is also interesting to note that some of the .torrent files contained more content regarding storage space than the free Dropbox account currently offers (2 gigabytes at the time of writing). 11 out of the set of tested 107 .torrents contained more then 2 gigabytes as they were DVD images, the biggest with 7.2 gigabytes in total size. This means that whoever stored those files on Dropbox has either a Dropbox Pro account (for which he or she pays a monthly fee), or that he invited a lot of friends to get additional storage space from the Dropbox referral program.

However, we could only infer the existence of these files. With the approach we used it is not possible to quantify to what extent Dropbox is used for filesharing among multiple users. Our results only show that within the last three to six months at least one Bittorrent user saved his downloads in Dropbox, respectively that since the .torrent has been created. No conclusions can be drawn as to whether they are saved in shared folders, or if only one person or possibly thousands of people uses Dropbox in that way. In fact, it is equally likely that a single person uses Dropbox to store these files.

With our experiments regarding online slack space we showed that it is very easy to hide data on Dropbox with low accountability. It becomes rather trivial to get some of the advanced features of Dropbox like unlimited undelete and versioning, without costs. Furthermore a malicious user can upload files without linking them to his account, resulting in possibly unlimited storage space

while at the same time possibly causing problems in a standard forensic examination. In an advanced setup, the examiner might be confronted with a computer that has no harddrive, booting from read only media such as a Linux live CD and saving all files in online slack space. No traces or local evidence would be extractable from the computer [15], which will be an issue in future forensic examinations. This is similar to using the private mode in modern browsers which do not save information locally [8].

6 Keeping the cloud white

To ensure trust in cloud storage operators it is vital to not only make sure that the untrusted cloud storage operator keeps the files secure with regards to availability [25], but also to ensure that the client cannot get attacked with these services. We provide generic security recommendations for all storage providers to prevent our attacks, and propose changes to the communication protocol of Dropbox to include data possession proofs that can be precalculated on the cloud storage operators side and implemented efficiently as database lookups.

6.1 Basic security primitives

Our attacks are not only applicable to Dropbox, but to all cloud storage services where a server-side data deduplication scheme is used to prevent retransmission of files that are already stored at the provider. Current implementations are based on simple hashing. However, the client software cannot be trusted to calculate the hash value correctly and a stronger proof of ownership is needed. This is a new security aspect of cloud computing, as up till now mostly trust in the service operator was an issue, and not the client.

To ensure that the client is in possession of a file, a strong protocol for provable data possession is needed, based on either cryptography or probabilistic proofs or both. This can be done by using a recent provable data possession algorithm such as [11], where the cloud storage operator selects which challenges the client has to answer to get access to the file on the server and thus omit the retransmission which is costly for both the client and the operator. Recent publications proposed different approaches with varying storage and computational overhead [12, 20, 10]. Furthermore every service should use SSL for all communication and data transfers, something which we observed was not the case with every service.

6.2 Secure Dropbox

To fix the discovered security issues in Dropbox we propose several steps to mitigate the risk of abuse. First of all, a secure **data possession protocol** should be used to prevent the clients to get access to files only by knowing the hash value of a file. Eventually every cloud storage operator should employ such a protocol if the client is not part of a trusted environment. We therefore propose the implementation of a simple challenge-response mechanism as outlined in Fig. 5. In essence: If the client transmits a hash value already known to the storage operator, the server has to verify if the client is in possession of the entire file or only the hash value. The server could do so by requesting randomly chosen bytes from the data during the upload process. Let H be a cryptographic hash function which maps data D of arbitrary length to fixed length hash value.

$Push_{init}(U, p(U), H(D))$ is a function that initiates the upload of data D from the client to the server. The user U and an authentication token $p(U)$ are sent along with the hash value $H(D)$ of data D . $Push(U, p(U), D)$ is the actual uploading process of data D to the server. $Req(U, p(U), H(D))$ is a function that requests data D from the server.

$Ver(Ver_{off}, H(D))$ is a function that requests randomly chosen bytes from data D by specifying their offsets in the array Ver_{off} .

Uploading **chunks without linking** them to a users

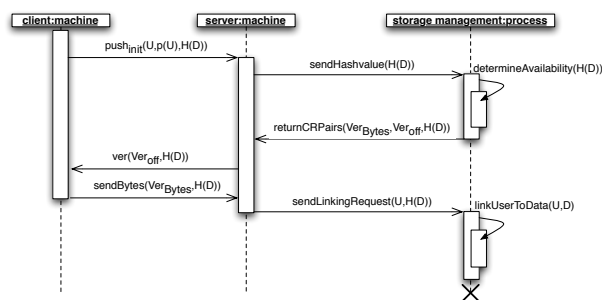


Figure 5: Data verification during upload

Dropbox should not be allowed, on the one hand to prevent clients to have unlimited storage capacity, on the other hand to make online slack space on Dropbox infeasible. In many scenarios it is still cheaper to just add storage capacity instead of finding a reliable metric on what data to delete - however, to prevent misuse of historic data and online slackspace, all chunks that are not linked to a file that is retrievable by a client should be deleted.

To further enhance security several behavioral aspects

Security Measure	Consequences
1. Data possession protocol	Prevent hash manipulation attacks
2. No chunks without linking	Defy online slack space
3. Check for host ID activity	Prevent access if host is not online
4. Dynamic host ID	Smaller window of opportunity
5. Enforcement of data ownership	No unauthorized data access

Table 6: Security Improvements for Dropbox

can be leveraged, for example to **check for host ID activity** - if a client turns on his computer he connects to Dropbox to see if any file has been updated or new files were added. Afterwards, only that IP address should be allowed to download files from that host IDs Dropbox. If the user changes IP e.g., by using a VPN or changing location, Dropbox needs to rebuild the connection anyway and could use that to link that host ID to that specific IP. In fact, the host ID should be used like a cookie [26] if used for authentication, dynamic in nature and changeable. A **dynamic host ID** would reduce the window of opportunity that an attacker could use to clone a victim's Dropbox by stealing the host ID. Most importantly, Dropbox should keep track of which files are in which Dropboxes (**enforcement of data ownership**). If a client downloads a chunk that has not been in his or her Dropbox, this is easily detectable for Dropbox.

Unfortunately we are unable to assess the performance impact and communication overhead of our mitigation strategies, but we believe that most of them can be implemented as simple database lookups. Different data possession algorithms have already been studied for their overhead, for example S-PDP and E-PDP from [11] are bounded by $O(1)$. Table 6 summarizes all needed mitigation steps to prevent our attacks.

7 Conclusion

In this paper we presented specific attacks on cloud storage operators where the attacker can download arbitrary files under certain conditions. We proved the feasibility on the online storage provider Dropbox and showed that Dropbox is used heavily to store data from thepiratebay.org, a popular BitTorrent website. Furthermore we defined and evaluated online slack space and demonstrated that it can be used to hide files. We believe that these vulnerabilities are not specific to Dropbox, as the underlying communication protocol is straightforward and very likely to be adopted by other cloud storage operators to save bandwidth and storage overhead. The discussed countermeasures, especially the data possession proof on the client side, should be included by all cloud storage operators.

Acknowledgements

We would like to thank Arash Ferdowsi and Lorcan Morgan for their helpful comments. Furthermore we would like to thank the reviewers for their feedback. This work has been supported by the Austrian Research Promotion Agency under grant 825747 and 820854.

References

- [1] Amazon.com, Amazon Web Services (AWS). Online at <http://aws.amazon.com>.
- [2] At Dropbox, Over 100 Billion Files Served—And Counting, retrieved May 23rd, 2011. Online at <http://gigaom.com/2011/05/23/at-dropbox-over-100-billion-files-served-and-counting/>.
- [3] Dropbox Users Save 1 Million Files Every 5 Minutes, retrieved May 24rd, 2011. Online at <http://mashable.com/2011/05/23/dropbox-stats/>.
- [4] Grab the pitchforks!... again, retrieved April 19th, 2011. Online at <http://benlog.com/articles/2011/04/19/grab-the-pitchforks-again/>.
- [5] How Dropbox sacrifices user privacy for cost savings, retrieved April 12th, 2011. Online at <http://paranoia.dubfire.net/2011/04/how-dropbox-sacrifices-user-privacy-for.html>.
- [6] NCrypto Homepage, retrieved June 1st, 2011. Online at <http://ncrypto.sourceforge.net/>.
- [7] Piratebay top 100. Online at <http://thepiratebay.org/top/all>.
- [8] AGGARWAL, G., BURSZEIN, E., JACKSON, C., AND BONEH, D. An analysis of private browsing modes in modern browsers. In *Proceedings of the 19th USENIX conference on Security (2010)*, USENIX Security'10.
- [9] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., AND ZAHARIA, M. A view of cloud computing. *Communications of the ACM* 53, 4 (2010), 50–58.
- [10] ATENIESE, G., BURNS, R., CURTMOLA, R., HERRING, J., KHAN, O., KISSNER, L., PETERSON, Z., AND SONG, D. Remote data checking using provable data possession. *ACM Transactions on Information and System Security (TISSEC)* 14, 1 (2011), 12.
- [11] ATENIESE, G., BURNS, R., CURTMOLA, R., HERRING, J., KISSNER, L., PETERSON, Z., AND SONG, D. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security (2007)*, CCS '07, ACM, pp. 598–609.
- [12] ATENIESE, G., DI PIETRO, R., MANCINI, L., AND TSUDIK, G. Scalable and Efficient Provable Data Possession. In *Proceedings of the 4th international conference on Security and privacy in communication networks (2008)*, ACM, pp. 1–10.

- [13] BOWERS, K., JUELS, A., AND OPREA, A. HAIL: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security* (2009), ACM, pp. 187–198.
- [14] BOWERS, K., JUELS, A., AND OPREA, A. Proofs of retrievability: Theory and implementation. In *Proceedings of the 2009 ACM workshop on Cloud computing security* (2009), ACM, pp. 43–54.
- [15] BREZINSKI, D., AND KILLALEA, T. Guidelines for Evidence Collection and Archiving (RFC 3227). *Network Working Group, The Internet Engineering Task Force* (2002).
- [16] CABUK, S., BRODLEY, C. E., AND SHIELDS, C. Ip covert timing channels: design and detection. In *Proceedings of the 11th ACM conference on Computer and communications security* (2004), CCS '04, pp. 178–187.
- [17] CHOW, R., GOLLE, P., JAKOBSSON, M., SHI, E., STADDON, J., MASUOKA, R., AND MOLINA, J. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security* (2009), ACM, pp. 85–90.
- [18] COX, M., ENGELSCHALL, R., HENSON, S., LAURIE, B., YOUNG, E., AND HUDSON, T. *Openssl*, 2001.
- [19] EASTLAKE, D., AND HANSEN, T. US Secure Hash Algorithms (SHA and HMAC-SHA). Tech. rep., RFC 4634, July 2006.
- [20] ERWAY, C., KÜPCÜ, A., PAPAMANTHOU, C., AND TAMASSIA, R. Dynamic Provable Data Possession. In *Proceedings of the 16th ACM conference on Computer and communications security* (2009), ACM, pp. 213–222.
- [21] GARFINKEL, S., AND SHELAT, A. Remembrance of data passed: A study of disk sanitization practices. *Security & Privacy, IEEE 1*, 1 (2003), 17–27.
- [22] GOLAND, Y., WHITEHEAD, E., FAIZI, A., CARTER, S., AND JENSEN, D. HTTP Extensions for Distributed Authoring–WEBDAV. *Microsoft, UC Irvine, Netscape, Novell. Internet Proposed Standard Request for Comments (RFC) 2518* (1999).
- [23] GROLIMUND, D., MEISSER, L., SCHMID, S., AND WATTENHOFER, R. Cryptree: A folder tree structure for cryptographic file systems. In *Reliable Distributed Systems, 2006. SRDS'06. 25th IEEE Symposium on* (2006), IEEE, pp. 189–198.
- [24] HARNIK, D., PINKAS, B., AND SHULMAN-PELEG, A. Side channels in cloud services: Deduplication in cloud storage. *Security & Privacy, IEEE 8*, 6 (2010), 40–47.
- [25] JUELS, A., AND KALISKI JR, B. PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security* (2007), ACM, pp. 584–597.
- [26] KRISTOL, D. HTTP Cookies: Standards, privacy, and politics. *ACM Transactions on Internet Technology (TOIT) 1*, 2 (2001), 151–198.
- [27] PIATEK, M., KOHNO, T., AND KRISHNAMURTHY, A. Challenges and directions for monitoring P2P file sharing networks: why my printer received a DMCA takedown notice. In *Proceedings of the 3rd conference on Hot topics in security* (2008), USENIX Association, p. 12.
- [28] POSTEL, J., AND REYNOLDS, J. RFC 959: File transfer protocol. *Network Working Group* (1985).
- [29] SCHWARZ, T., AND MILLER, E. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on* (2006), IEEE, p. 12.
- [30] SUBASHINI, S., AND KAVITHA, V. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications* (2010).
- [31] WANG, C., WANG, Q., REN, K., AND LOU, W. Ensuring data storage security in cloud computing. In *Quality of Service, 2009. IWQoS. 17th International Workshop on* (2009), Ieee, pp. 1–9.
- [32] WANG, Q., WANG, C., LI, J., REN, K., AND LOU, W. Enabling public verifiability and data dynamics for storage security in cloud computing. *Computer Security–ESORICS 2009* (2010), 355–370.

Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting

The paper *Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting* was published at the Web 2.0 Workshop on Security and Privacy (W2SP) 2013 co-located with the 2013 IEEE Symposium on Security and Privacy in San Francisco²⁸.

You can find the paper online at <http://www.w2spconf.com/2013/papers/s2p1.pdf>. The slides of the presentation are available online as well²⁹.

²⁸<http://www.w2spconf.com/2013/>

²⁹<http://www.w2spconf.com/2013/slides/s2p1-slides.pdf>

Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting

Martin Mulazzani*, Philipp Reschl†, Markus Huber*,
Manuel Leithner*, Sebastian Schrittwieser* and Edgar Weippl*

*SBA Research

Vienna, Austria

†FH Campus Wien

Vienna, Austria

Abstract—Web browsers are crucial software components in today’s usage of the Internet, but the reliable detection of whether a client is using a specific browser can still be considered a nontrivial problem. Reliable browser identification is crucial for online security and privacy e.g., regarding drive-by downloads and user tracking, and can be used to enhance the user’s security. So far the UserAgent string is often used to identify a given browser, but it is a self-reported string provided by the client and can be changed arbitrarily.

In this paper we propose a new method for identifying web browsers based on the underlying Javascript engine, which can be executed on the client side within a fraction of a second. Our method is three orders of magnitude faster than previous work on Javascript engine fingerprinting, and can be implemented with well below a few hundred lines of code. We show the feasibility of our method with a survey and discuss the consequences for user privacy and browser security. Furthermore, we collected data for more than 150 browser and operating system combinations, and present algorithms to make browser identification as fast as possible. UserAgent string modifications become easily detectable with JavaScript engine fingerprinting, which is shown exemplarily on the Tor browser bundle as it uses a uniform UserAgent string across different browser versions. Finally, we propose to use our results for enhancing state-of-the-art session management (with or without SSL), as reliable browser identification can be used to increase the complexity of session hijacking attacks considerably.

Keywords—Browser Fingerprinting, Privacy, Security

I. INTRODUCTION

With the rise of ever more sophisticated Web applications that nowadays even compete with native software, the web browser became the dominant interface connecting the user to a computing system. Platforms such as Gmail or Zoho.com were designed from the ground up to be primarily accessed via web browser, replacing their native counterparts (email client and office suite). Due to the immense importance of the web browser for the interaction with the user, it became a central component of almost every modern operating system: Microsoft has Internet Explorer, Apple has Safari, and Google is building ChromeOS, an operating system based entirely on its web browser Chrome. Furthermore, system-independent browsers such as Opera also contribute to the highly competitive and diverse browser market.

While today’s browsers interpret a website’s code in similar ways (based on standards), the actual implementations of these standards differ. This diversity of browsers has always caused headaches for Web developers, as the same website can vary across different browsers with respect to functionality or appearance, requiring additional testing and debugging of a website’s code in order to ensure correct functionality in relevant browsers. However, this can also have severe implications on privacy and security. In this paper, we propose a novel concept for browser identification, which exploits exactly these imperfect implementations of standards in the different browsers. Our work was originally motivated by the security scanner *nmap*, which uses TCP/IP stack fingerprinting to determine the operating system of a remote host. In a very similar way, we use the browser’s underlying JavaScript engine for browser identification. While implementation differences in HTML5 or CSS could also be used for fingerprinting, we decided to base our approach on JavaScript as it is well established, supported by all major browsers, and works on mobile devices such as smartphones and tablets. JavaScript is furthermore used by a very high percentage of websites, and enabled by default on all major browsers. While other methods for server-side browser identification exist (in particular and by design, the User-Agent string), our approach can be considered more robust. While the User-Agent string can be set to an arbitrary value by the user, the JavaScript fingerprint is authentic for each browser and cannot be easily ported to a different browser. It is not easily possible to use a modified version of e.g., Internet Explorer with SpiderMonkey, the JavaScript engine of Mozilla’s Firefox in order to obfuscate the actual browser in use.

In particular, the contributions of this paper are as follows:

- We propose a new method to reliably identify a browser based on the underlying JavaScript engine. Our method is more than three orders of magnitude faster than previous work.
- We show the feasibility and reliability of our method with a survey.

- We show how this can be used to detect modified UserAgent strings, used, for example, by the Tor browser bundle to increase the size of the anonymity set of its users.
- We propose an iterative protocol for server-side detection of session hijacking using browser fingerprinting.
- Raise awareness for such advanced fingerprinting methods, and discuss measures to protect users.

The rest of the paper is organized as follows: Section II gives the technical background. Our method for browser identification based on fingerprinting the JavaScript engine is introduced in Section III. We show the feasibility of browser identification with JavaScript engine fingerprinting in Section IV and discuss our results as well as possible countermeasures in Section V. Related work is presented in Section VI before we conclude in Section VII.

II. BACKGROUND

Today's browser market is highly competitive. Browser vendors publish new versions in ever-shorter time spans and regularly add new features, with especially mobile browsers for smartphones and tablets on the rise. Many of these updates increase the overall performance of the browser in order to enhance the user experience and reduce loading times: just-in-time compilation (JIT) of Javascript, for example, allows dynamic compilation of Javascript and became part of the Javascript engines in Firefox and Google Chrome's V8 quite recently, among others. Using the GPU for rendering in Internet Explorer's Chakra engine is yet another feature that was introduced recently and increased browser performance considerably. Sandboxing the browser or specific parts, like the Flash plugin, was introduced to increase the overall browser security and to combat the widespread use of Flash-based security exploits.

Javascript has been standardized as ECMAScript [8], and all major browsers implement it in order to allow client-side scripting and dynamic websites. Traditionally, Web developers use the *UserAgent* string or the navigator object (i.e., *navigator.UserAgent*) to identify the client's Web browser, and load corresponding features or CSS files. The UserAgent string is defined in RFC2616 [11] as a sequence of product tokens and identifies the software as well as significant subparts. Tokens are listed in order of their significance by convention. The navigator object contains the same string as the UserAgent string. However, both are by no means security features, and can be set arbitrarily by the user.

Mowery et al. [22] recently implemented and evaluated browser identification with Javascript fingerprinting based on timing and performance patterns. In their paper, the authors used a combination of 39 different well-established Javascript benchmarks, like the SunSpider Suite 0.9 and the V8 Benchmark Suite v5, and generated a normalized fingerprint from runtime patterns. Even though these artificial Javascript benchmarks, such as SunSpider, do not necessarily

reflect real-world Web applications [28], using their patterns for fingerprint generation is a convenient approach. In total, the runtime for fingerprinting was relatively high, with 190 seconds per user on average (caused partly by an intentional delay of 800ms between tests). Our approach is superior in multiple ways: (1) It's runtime is **more than three orders of magnitude faster** (less than 200ms on average compared to 190s), while having a comparable overhead for creating and collecting fingerprint samples. (2) It can be implemented in just a few hundred lines of Javascript and is undetectable for the user, as the CPU is not stalled noticeably. (3) Many recent browser versions stall the execution of Javascript from tabs and browser windows that are currently not visible to the user to increase the responsiveness of the currently active windows. This, however, could severely distort the timing patterns generated from [22] and was not addressed in the paper.

For the rest of the paper we will use the following terminology due to sometimes ambiguous usage of the term browser fingerprinting in the literature: the fingerprinting in our approach refers to Javascript fingerprinting, not the browser. We use Javascript engine fingerprinting to reliably identify a given browser, and for identifying the browser itself as well as the major version number. Related work (like Panopticlick [7]) uses the term browser fingerprinting for identifying a particular browser instance.

III. DESIGN

For our fingerprinting method, we compared test results from openly available Javascript conformance tests and collected results from different browsers and browser versions for fingerprint generation. These tests cover the ECMAScript standard in version 5.1 and assess to what extent the browser complies with the standard, what features are supported and specifically which parts of the standard are implemented incorrectly or not at all. In essence, our initial observation was that the test cases that fail in, e.g., Firefox, are completely different from the test cases that fail in Safari. We started working with Google's *Sputnik* test cases, but later switched to *test262*¹. *test262* is the official TC39 test suite for ECMAScript, it is still improved regularly and is a superset of the *Sputnik* test cases. For ensuring comparability within our results from the experiments in Section IV we used *test262* from mid-January 2012, which includes 11,148 unique test cases for desktop browsers, while for the mobile browsers we used an updated version of *test262* with 11,570 test cases. However, the ECMAScript standard as well as the test suite are constantly updated, leaving enough future testing capabilities for Javascript engine fingerprinting. Running the full *test262* suite takes approximately 10 minutes on a desktop PC, while on smartphones and tablets it takes between 45 minutes and an hour, depending on the underlying hardware.

¹<http://test262.ecmascript.org>

A. Efficient Javascript Fingerprinting

While Javascript conformance tests like *Sputnik* or *test262* consist of thousands of independent test cases, not all of them are necessary for browser identification. In fact, a single test case may be sufficient, e.g., to distinguish two specific browsers - if one of the browsers fails in a particular test case, while the other one does not, and assuming a priori that only these two browsers are within the set of browsers to test, this single test case is already enough to distinguish them. An example: Opera 11.64 only fails in 4 out of more than 10,000 tests cases from mid-January, while the most recent version of Internet Explorer 9 at that time failed in almost 400 test cases. If the test set contains only those two browsers, and the goal is to distinguish whether the client is using Opera 11.61 or Internet Explorer 9, a single test from the 400 failed test cases of Internet Explorer 9 (that are not within the set of 4 failed test cases from Opera) is sufficient to reliably distinguish those two browsers, and can be executed within a fraction of a second.

To formalize this approach: the **test set** of browsers is the set of browsers and browser versions that a given entity wants to make reliably distinguishable, in our case with Javascript engine fingerprinting. First, each browser is tested with *test262*. The results are then compared, and a **minimal fingerprint** is calculated for each browser (in relation to the other browsers in the test set). The use case for the minimal fingerprint is a web server that wants to assess whether a UserAgent string from the client is forged with respect to the other browsers in the test set. The web server can **verify** the browser. For efficiency, one of the requirements is that the fingerprint for each browser is as small as possible. The details of our implementation and the algorithm for generating the minimal fingerprints can be found in Section III-B.

Another use case of our method is to calculate a **decision tree**: instead of fingerprinting a particular browser with respect to the test set, we propose to build a binary decision tree to iteratively **identify** the browser in multiple rounds. The use case for this method is that the web server wants to identify the browser used under the assumption that the UserAgent might be forged. This method allows a larger test set than using minimal fingerprints while reducing the execution time on the client side. The pseudocode for calculating a minimal decision tree can be found in Section III-C.

B. Minimal Fingerprint

We use a greedy algorithm to find a (possibly minimal) fingerprint for a given test set: We start by running *test262* for each of the browsers in the test set, and calculate the number of browsers that fail for each test case. As the JavaScript engine is a static part of the browser, this needs to be done only once per browser. We then compare the results of *test262* for the browsers within the test set and calculate for each failed test case the number of browsers

that fail. We call the total number of browsers that fail a particular test case the *uniqueness* u (with respect to the test set). We then select a test case with $u = 1$ at random and remove the browser from the test set, as this test uniquely identifies this browser. The uniqueness u is then recalculated for the remaining test cases. The process is repeated until either a unique fingerprint has been found for every browser, or no test case with $u = 1$ is found. In the latter case, we change our selection and choose a different test case until either a minimal test set is found or no solution can be found. An alternative approach would be to use some form of machine learning to find minimal fingerprints, but our results indicate that this is not (yet) necessary and our simplistic, greedy algorithm works well in practice. With the resulting set of fingerprints it becomes possible to assess whether a browser is what it claims to be: if all the tests of the minimal fingerprint for that browser fail, and no minimal fingerprints for the other browsers from the test set do, the browser is uniquely identifiable with respect to the test set. To make the algorithm and, in consequence, browser fingerprinting more resilient against errors, multiple tests could be used per browser (in case the user's browser is not part of the test set and the UserAgent string is not used to check this beforehand).

However, a basic assumption here is that the browser is included in the test set during fingerprint calculation in the first place. If the browser is not in the test set, false positives could occur if the engine is similar to one of the fingerprints (with respect to the minimal fingerprint). It is also possible to dynamically extend the test set: If a new UserAgent string is encountered that was not part of the test set, fingerprints could be recalculated on the fly to determine whether the UserAgent correctly identifies the browser: Instead of using the precalculated fingerprints, the browser is added to the test set, fingerprints are recalculated, and the identification process starts again with new minimal fingerprints for all browsers in the test set. This would allow *relative fingerprinting* over time and could be used to verify only the, e.g., five most popular browser versions for the previous month or day.

The minimal set of failed test cases for the four common browsers from 2012 shown in Table I to illustrate minimal fingerprints. The browsers in the test set are Firefox 12, Opera 11.64, Internet Explorer 9 and Chrome 20, with a resulting minimal fingerprint consisting of only 4 tests. With the algorithm explained above, we calculate the minimal fingerprints as follows: For every test case, the uniqueness in the test set is calculated. If a test fails for a specific browser, it receives a check mark in the table, and if the browser does not fail that test, it is crossed out. While this seems counter-intuitive, the check mark highlights the potential to use this particular test case for fingerprinting, as the number of failed test cases is much smaller than the number of tests passed. One of the test cases with $u = 1$ is selected at random, in the example this is 13.0-13-s. This test then becomes the minimal fingerprint for Internet Explorer 9, and

Internet Explorer is removed from the set of browsers that do not yet have a fingerprint. The uniqueness is recalculated, and another test case is selected at random with $u = 1$, e.g., 10.6-7-1, which becomes the minimal fingerprint for Firefox 12. Next, Opera gets 15.4.4.4-5-c-i-1 as fingerprint, and Chrome S15.8.2.16_A7. If a web server now tries to verify a given UserAgent, all 4 tests are sent for execution to the client, and the web server can verify the UserAgent with respect to the test set if only one test fails (in this example).

C. Building a Decision Tree

To identify a user's browser without relying a priori on the UserAgent, we build a binary decision tree for a given test set and assess if the browser is included in it by running multiple test rounds. For every test, we step down one level of the decision tree until we finally reach a leaf node. Inner nodes in this decision tree are test cases, while the edges show whether the browser fails that test or not. Instead of calculating a unique fingerprint for each browser in the test set, we need to identify the test cases that can be used to split the number of browsers that fail (respectively pass) equally. Multiple rounds of discriminating test cases can thus be used instead of calculating the minimal fingerprints for large test sets. The decision tree can reduce the total number of executed test cases considerably for such large test sets, making browser identification much faster. The decision tree is especially useful if the test set and the total number of test cases for the minimal fingerprints are rather large.

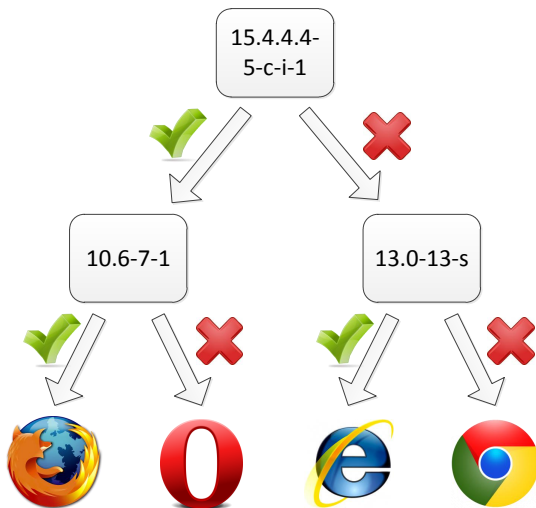


Fig. 1. Decision tree for Table I

To calculate a decision tree, we adapt the algorithm above slightly. We start again by calculating the *uniqueness* u for each *test262* test case that fails, sort the list and pick the test that splits the set into halves as the first test in our tree. If there is no such test, we select the statistical mode. We then

continue to split the array of browsers into two parts, and recursively repeat this until we have built a complete decision tree with all the browsers from the test set. No assumptions can be made for the distribution of failed test cases, which means that in the worst case the tree can become a linear list instead of a tree if all failed tests have uniqueness $u = 1$. Again, if no tree can be found using the statistical mode, we can slightly vary the choice of test cases for the inner nodes and rerun the algorithm. In the ideal case, every inner node in the tree splits the subset of browsers in the test set in half, and the total number of tests that need to be executed at the client is only $O(\log n)$ compared to $O(n)$ for executing the minimal fingerprints. Referring to the example from Section III-B, we can construct a decision tree as follows (cf. Table I): We start again by calculating the uniqueness for every test case of every browser that fails. We sort the results, and pick test 15.4.4.4-5-c-i-1 as our root node, because it splits the test set perfectly into halves. We then select the tests 10.6-7-1 and 13.0-13-s as the child nodes, and can identify the browser by running only two test cases, instead of four with the minimal fingerprinting approach. The resulting decision tree is shown in Figure 1. As with the algorithm for calculating minimal fingerprints, the algorithm is straightforward and fast to implement and execute on the client as well as on the server - it works well across different browsers and versions, thus negating the need for a more complex algorithm.

D. Implications on Security and Privacy

While the UserAgent string is traditionally used to report the web browser and version to a server, this is often not sufficient as the user can change it arbitrarily. In the context of browser **security**, current malware often relies on vulnerabilities in browsers (besides plugins like Flash) for launching exploits. Especially exploit kits like Blackhole [16] have been shown to use the UserAgent String to exploit client-side vulnerabilities. It is furthermore well known in the security community that Javascript and drive-by-download attacks can be used to endanger client security and privacy [3], [32], [4]. For the implications to privacy we use the security model of Tor [6] and the definition of an anonymity set [27], which could be decreased by a malicious website using JavaScript engine fingerprinting. Section VI discusses numerous recent papers that have been using browser fingerprinting to endanger user's online privacy.

In our threat model we assume that an attacker has the capabilities to host a website and direct users to it. The victim then fetches and executes Javascript code on the client side. This can be done e.g., by renting advertisement space, or with social engineering attacks where the user is tricked into opening a malicious website. This is already happening with malware on a large scale, and everything necessary to conduct such attacks can be purchased relatively easily. This is of relevance for our work, as malware authors could use browser fingerprinting to use it for increasing reliability of

Web browser	15.4.4.4-5-c-i-1	13.0-13-s	S15.8.2.16_A7	10.6-7-1	15.2.3.6-4-410
Opera 11.64	✓	✗	✗	✗	✗
Firefox 12.0	✓	✗	✗	✓	✗
Internet Explorer 9	✗	✓	✗	✗	✗
Chrome 20	✗	✓	✓	✗	✓
Uniqueness u	2	1	1	1	1

TABLE I
TESTS FROM *test262* AND THEIR USABILITY FOR BROWSER IDENTIFICATION

their exploits, thwart sandboxed environments like Wepawet² and to increase the stealthiness of their malware: instead of relying on the UserAgent string to find out if a victim is exploitable, Javascript fingerprinting could be used. The bar to incorporate this is low, and could be of significance for the arms race between malware authors and security research in the future. Detection of such malicious code would be considerably harder, and we aim to increase awareness for security researchers of such sophisticated browser fingerprinting methods. More work is needed to assess if this or similar fingerprinting is already used by malware in the wild.

E. Benign Uses of Fingerprinting

Here we discuss some benign use cases in addition to the sections discussing the framework and our results, respectively. To protect against session hijacking, web servers could use JavaScript engine fingerprinting to verify or refute validity of HTTP sessions, as session hijackers usually clone all possibly identifying plaintext information like session cookies (e.g., Firesheep³ or FaceNiff⁴ do) or the complete HTTP header. With JavaScript engine fingerprinting such attacks become detectable at the server side, as modified UserAgents can be detected. Another way to secure HTTP sessions would be to constantly challenge the browser and add Javascript engine fingerprinting as an additional security layer: At the beginning of a session the browser is identified with minimal fingerprinting. For every latter request the webserver chooses a subset of random test cases and includes them in the JavaScript code, thus challenging the client. The overhead would be minimal and not noticeable to the client. If the responses do not correlate with the expected outcome, the session is terminated. While the attacker is able to see the challenges, he might not know the correct responses - the attacker is forced to (1) either use the same browser down to the very same version (which may be not possible, e.g., to run an Internet Explorer on Android), or (2) collect the fingerprints for his victim beforehand to fake the replies, which would be very time consuming. Thus Javascript fingerprinting can be used to raise the bar for session hijacking in the arms race against attackers. This method could also be used for connections that are secured with HTTPS to prevent HTTPS MITM

attacks. Recently hacked CAs like DigiNotar or Comodo and “Operation Black Tulip”⁵ have shown that HTTPS alone is simply not enough to secure online communication anymore. However, it cannot completely defy session hijacking as the attacker might for example simply relay the challenges to the actual client. We believe though that this would be a valid countermeasure against session hijacking as this can be added easily to existing web applications.

IV. RESULTS AND EVALUATION

To evaluate the possibility and power of Javascript fingerprinting, we implemented the methods outlined above. We collected different browser version and operating system combinations for desktop browsers as well as for mobile browser versions on smartphones and tablets for fingerprint generation in a database. An excerpt of the data can be seen in Table II⁶. To evaluate our method with respect to the security and privacy implications discussed in Section III-D, we first evaluate if it is possible to determine the actual browser behind a modified UserAgent as used by the Tor Browser Bundle on a large scale. We also conducted a survey and measure the performance impact of our method on the client side.

A. Desktop and Mobile Browsers

In total, we stored the *test262* results for more than 150 different browser version and operating system combinations, ignoring minor version upgrades of browsers that contained no changes in the underlying Javascript engine. While this may not sound like much, it includes all major browser releases from the last three years, which accumulates to approximately 98% of browser market share since 2008⁷. For desktop browsers we collected test results for fingerprint generation from the five most popular browsers on three different operating systems: Windows, OS X and Linux. Different mobile browser versions and their test results can be seen in Table III. Results for mobile browsers are focused on Android and iOS devices. While the setup files for desktop browsers are often freely available and were easy to collect, it was much more difficult for us to get access to a broad spectrum of older mobile browsers as it is not possible to revert the running operating system of a smartphone or a tablet to an older software version, among other reasons as

²<https://wepawet.iseclab.org>

³<http://codebutler.com/firesheep>

⁴<http://faceniff.ponury.net>

⁵<http://www.enisa.europa.eu/media/news-items/operation-black-tulip/>

⁶Please see the author’s homepage for the full data set

⁷http://www.w3schools.com/browsers/browsers_stats.asp

Browser	Win 7	WinXP	Mac OS X	Browser	Win 7	WinXP	Mac OS X
Firefox 3.6.26	3955	3955	3955	Chrome 8	1022	1022	1022
Firefox 4	290	290	290	Chrome 10	715	715	715
Firefox 5	264	264	264	Chrome 11	489	489	489
Firefox 6	214	214	214	Chrome 12	449	449	—
Firefox 7	190	190	190	Chrome 13	427	427	—
Firefox 12	165	165	165	Chrome 14	430	430	430
Firefox 15	161	161	161	Chrome 16	420	420	420
Firefox 17	171	171	171	Chrome 17	210	210	210
Firefox 19	191	191	191	Chrome 18	35	35	35
IE 6 (Sputnik)	—	468	—	Chrome 19	18	18	18
IE 8 (Sputnik)	—	473	—	Chrome 21	9	9	9
IE 9	611	—	—	Chrome 23	10	10	10
IE 10	7	—	—	Chrome 25	17	17	17
Opera 11.52	3827	3827	3827	Safari 5.0.5	777	1585	1513
Opera 11.64	4	4	4	Safari 5.1	777	853	—
Opera 12.02	4	4	4	Safari 5.1.2	777	777	776
Opera 12.14	9	9	9	Safari 5.1.7	548	548	547

TABLE II
SELECTION OF BROWSERS AND THEIR FAILED TEST CASES FROM *test262* (AND *Sputnik*)

Browser	OS	Device	# of fails
Safari	iOS 5.1.1	iPhone 4S	988
Safari	iOS 6.1.2	iPhone 4	28
Browser	Android 2.2	GalaxyTab	2130
Browser	Android 2.3.7	HTC Desire	1328
Browser	Android 4.0.3	GalaxyTab2	588
Browser	Android 4.0.4	Nexus S	591
Browser	Android 4.1.2	Nexus S	23
Chrome 18	Android 4.0.3	GalaxyTab2	46
Firefox 19	Android 4.0.3	GalaxyTab2	191

TABLE III
NUMBER OF FAILED TEST CASES FOR MOBILE BROWSERS

protection against jailbreaking.

As the Javascript engines are not as dynamic as the numbering scheme of browser vendors, equal results are obtained with consecutive browser versions if the underlying Javascript engine has not been changed. For example, the major Firefox version numbers often indicate changes in the underlying principles, while the minor numbers are used for, e.g., security updates: Updates 6.0.1 and 6.0.2, for example, were used to solely remove the Dutch certificate authority Diginotar, which got hacked and was used for issuing rogue certificates [25]. The results are discussed in detail in Section V.

B. Tor Browser Bundle

While modifying the UserAgent can be used to hide a user's browser and version, JavaScript engine fingerprinting can be used to reliably identify the web browser of a user. The Tor Browser Bundle is using modified UserAgent strings on a large scale, and we will show how these modified UserAgents can be detected by web servers. The Tor network [6] is an overlay network that provides online anonymity to its users by hiding the user's IP address. At the time of writing it is estimated to be used by more than 500,000 users every

day⁸. It has been previously shown that the majority of Tor users do not browse the Web securely [15], and Javascript engine fingerprinting can be used to further increase the attack surface for sophisticated de-anonymization attacks. The *Tor Browser Bundle* (TBB) is a convenient way to use the Tor anonymization network with a known, secure configuration and without the need for the user to install any software. It is available for Windows, Linux and OS X and has many important privacy-enhancing features enabled by default, e.g., TorButton or HTTPS Everywhere, prepackaged and preconfigured, making it the recommended way to use the Tor network securely at the time of writing. By default, the Tor Browser Bundle changes the UserAgent string to increase the size of the anonymity set [5]. In the Tor Browser Bundle the UserAgent is uniformly changed to Firefox 5.0 while the shipped browser often uses a more recent version. Mozilla releases new versions of Firefox every six weeks. The numbers of the actual and the expected results from *test262* running on Windows 7 can be seen in Table IV. A decision tree similar to the example in Section III-C can be constructed to minimize the number of tests needed to

⁸<https://metrics.torproject.org/users.html>

accurately identify the browser used with the Tor Browser Bundle. In the most recent versions of TBB the browser was changed to Firefox 17 with long-term support, and the UserAgent is correctly identifying the browser as Firefox 17.

Care has to be taken when interpreting the implications of Javascript engine fingerprinting on the Tor network: Even though Javascript is not disabled by default in the Tor Browser Bundle [1], the only information the malicious website operator obtains is that the user is, in fact, using a different version of Firefox than indicated. The web server can already easily determine that the user is using the Tor network by comparing the client's IP address to the public list of Tor exit relays. However, Javascript fingerprinting can reduce the size of the anonymity set of all Tor users, and can harm anonymity to a yet unknown extent.

C. Experimental Survey

To evaluate the performance and practicability of our fingerprinting method, we conducted a survey among colleagues, students and friends for a duration of several weeks in 2011 to find out (1) whether our method was working reliably, and (2) to measure the time and bandwidth needed for fingerprinting. The test set consisted of Firefox 4, Chrome 10 and Internet Explorer 8 & 9, which were the top browsers at that time and had a cumulative worldwide market share of approx. 66% at that time. For each of the browsers in our test set, we manually selected 10 failed test cases from the Sputnik test suite to be run on the client instead of the minimal fingerprint, to increase accuracy and decrease the possibility of false positives. As a result, every client executed 40 test cases in total, and the failed test cases were then used to determine the user's Javascript engine. Due to the automatic update function of Google Chrome, the version number changed from 10 to 12 during the testing period, but the 10 test cases we had selected for Chrome did not change, so the updates did not skew our results and Chrome was still correctly identified even though the Javascript engine changed with the updates (see Table II). Users were directed to a webpage where they were asked to identify their web browser manually using a dropdown menu and to start the test. As a ground truth to evaluate our fingerprinting, we relied on the UserAgent string in combination with the manual browser classification by the users. The Javascript file containing the 40 tests as well as the testing framework had a size of 24 kilobytes, while each of the 10 tests per browser were only between 2,500 and 3,000 bytes in size. The results were written to a database. We used a cookie to prevent multiple test runs by the same browser, and also blocked submissions with the same UserAgent string and IP address that originated in close temporal vicinity in case the browser was configured to ignore cookies.

In total, we were able to collect **189** completed tests. From those 189 submissions, 175 were submitted by one of the four browsers covered by the test set, resulting in an overall relative

coverage of more than 90%. 14 submissions were made with browsers not in the test set, mainly smartphone web browsers. We compared the results of Javascript fingerprinting with the UserAgent string as well as the user choice from the dropdown menu, and Javascript fingerprinting had the correct result for all browsers in the test set. In one case our method identified a UserAgent string manipulation, as it was set to a nonexistent UserAgent. In 15 cases, the users made an error identifying their browser manually from the dropdown menu, but the UserAgent and the results from fingerprinting matched. There were no false positives for the browsers within the test set; the algorithm for fingerprinting identified browsers if and only if all the test cases for that browser failed and all tests for the other browsers did not fail. The runtime for the entire test was short, with 90ms on average for PCs and 200ms on average for smartphones (even though smartphones were not part of the test set).

V. DISCUSSION

The results above show that JavaScript engine fingerprinting is a feasible approach to identify or verify a given browser, even for mobile devices like smartphones, with only small overhead regarding execution time on the client and bandwidth. On the server side the impact is negligible, as it can be implemented as a small number of database lookups. The "best" browser regarding Javascript standard conformance in our set of tested browsers was Opera, with only 4 failed tests in its most recent versions. Firefox and Chrome improved the engine constantly between releases, which happen at a much higher pace. Internet Explorer used a different XMLHttpRequest method before version 8 and thus did not work with *test262*, so we relied on the *Sputnik* tests and test numbers for fingerprint generation in Section IV-C. Please note that it is not the total number of failed test cases that is of importance, but if there is a difference between the browsers in the test set. For browser identification and with respect to the chosen test set, a single test case per browser is often sufficient to distinguish between two or more browsers. Also, these results and the number of failed tests are not static in nature: browsers, ECMAScript and the test suites are under active development and are constantly improved, with ECMAScript currently preparing version 6 of the standard (codename "Harmony").

While it is possible to detect a specific browser version with the algorithms discussed above, our method cannot be used to detect the underlying operating system (compared to the approach used in [22]). Other means are necessary to identify it as well as the underlying computing architecture (x86, x64, ...). Due to their complexity, JavaScript engines reuse their engine across different operating systems, as it nowadays takes multiple man-years to develop a modern JavaScript engine. All the latest browser versions at the time of writing that run on different operating systems and platforms seem to use the same Javascript engine. The only exception we

Version TBB	Browser	UserAgent	test262	exp. test262	Detectable
2.3.25-4	Firefox 17esr	Firefox 17	171	171	✗
2.3.25-2	Firefox 10esr	Firefox 10	172	172	✗
2.2.35-9	Firefox 12.0	Firefox 5.0	165	264	✓
2.2.35-8	Firefox 11.0	Firefox 5.0	164	264	✓
2.2.35-3	Firefox 9.0.1	Firefox 5.0	167	264	✓
2.2.33-2	Firefox 7.0.1	Firefox 5.0	190	264	✓
2.2.32-3	Firefox 6.0.2	Firefox 5.0	214	264	✓
2.2.30-2	Firefox 5.0.1	Firefox 5.0	264	264	✗
2.2.24-1	Firefox 4.0	Firefox 3.6.3	290	3956	✓

TABLE IV
DETECTABILITY OF USERAGENT STRING MANIPULATIONS IN TBB

could find were (mostly historic) versions of Safari, where the same version number on different operating systems used different versions of the Javascript engine (see Table II). For all the other browsers we tested, the version number convention across operating systems seems to correlate with the Javascript engine. We could show on the other hand that operating system detection for smartphones and tablet PCs is possible, and that we can easily distinguish between e.g., Android or iOS with our method. Due to the larger update cycles compared to desktop browsers, and due to the fact that there are still a lot of old Android versions in use, JavaScript engine fingerprinting is thus especially dangerous for mobile devices. It is furthermore possible to distinguish between a mobile browser and one running on a regular computer easily, if both are included in a test set. However, our sample size for mobile devices is much smaller compared to our desktop browser dataset - more work is needed in this area.

A. Countermeasures

It is naive to believe that Javascript engines across different browsers will conform uniformly with the standard in the future due to the complexity of the Javascript engines. As this is unlikely to happen in the near future, we propose preventing or detecting fingerprinting on the client side. Client-side protection could be done either by the browser itself [4], a browser extensions looking for fingerprinting of any kind, or by using a proxy server that can detect and block fingerprinting patterns similar to TCP/IP stack fingerprinting prevention methods [30]. We are currently working on a browser extension that can detect Javascript fingerprinting, and hope to work on a proxy solution in the near future as well.

B. Future Work

Future work towards browser fingerprinting includes other core features of browsers that are not yet uniformly implemented, such as HTML5 or CSS3. We plan to add these to the fingerprint generation process, to decrease overall runtime and the computational overhead even further, and to make our approach work with browsers that have Javascript disabled. We also plan to assess whether current advertising networks [29] are already using Javascript fingerprinting,

just as they were recently found to already use tricks to spawn almost undeletable cookies like *evercookie* [18], Flash cookies [31] or ETag respawning [2]. We are also working on a framework that can detect and prevent session hijacking on insecure connections (with or without SSL alike), as proposed in Section III-E.

VI. RELATED WORK

Javascript has recently received a lot of attention with the rise of AJAX as a new programming paradigm and especially with respect to client-side security [3], [13] and privacy [17]. Cross-site scripting (XSS) as one of the most prevalent online security vulnerability in fact only works when a browser has Javascript enabled.

Fingerprinting in general has been applied to a broad and diverse set of software, protocols and hardware over the years. Many implementations try to attack either the security or the privacy aspect of the test subject, mostly by accurately identifying the exact software version in use. One of the oldest security-related fingerprinting software is *nmap* [12], which is still used today and uses slight differences in the implementation of network stacks to identify the underlying operating systems and services. OS fingerprinting is often a crucial stepping stone for an attacker, as remote exploits are not uniformly applicable to all versions of a given operating system or software. Another passive fingerprinting tool, *p0f*⁹, uses fingerprinting to identify communicating hosts from recorded traffic. Physical fingerprinting, on the other hand, allows an attacker to identify (or track) a given device, e.g., using specific clock skew patterns, which has been shown to be feasible in practice to measure the number of hosts behind a NAT [19], [24]. History stealing [26], [33] has been shown to be another, effective attack vector to de-anonymize users and could be used for browser fingerprinting as well. User tracking is yet another threat to the privacy of users, which is, e.g., used heavily by advertising networks [29], [21].

⁹<http://lcamtuf.coredump.cx/p0f3>

In recent years, the focus shifted from operating system fingerprinting towards browser and HTTP traffic fingerprinting in the area of security and privacy research. On the one hand, this was caused by the widespread use of firewalls as well as normalized network stacks and increased awareness of administrators to close unused ports. On the other hand, the browser has become the most prevalent attack vector for malware by far. This trend has been further boosted by the advent of cloud computing (where the browser has to mimic or control operating system functionality), online banking and e-commerce, which use a web browser as the user interface. Recent malware relies on fingerprinting to detect if the victim's browser is vulnerable to a set of drive-by-download attacks [9], [3]. For encrypted data, Web-based fingerprinting methods rely on timing patterns [10], [14], but at higher expenses in terms of accuracy, performance, bandwidth and time. The EFF's Panopticlick project¹⁰ does browser fingerprinting by calculating the combined entropy of various browser features, such as screen size, screen resolution, UserAgent string, and supported plugins and system fonts [7]. Mayer was among the first to discuss technical features that can be used for browser fingerprinting [20]. In recent work, browser fingerprinting with the aim of harming the user's privacy has been used effectively solely by using the UserAgent string [34]. Another recent paper uses novel HTML5 features and WebGL to accurately fingerprint browsers [23] and the underlying hardware (GPU).

VII. CONCLUSION

In this paper, we introduced a method for reliable browser identification based on the underlying Javascript engine, and evaluated its feasibility in multiple ways. In a survey with 189 participants, our method identified all browsers within the test set correctly. We also evaluated the impact on systems like the Tor Browser Bundle that use a modified UserAgent string on purpose to increase the anonymity of users, and collected data for generating fingerprints for more than 150 browser and operating system combinations. We showed that this method can be used efficiently in terms of bandwidth and computational overhead, takes less than a second to run on the client, and can reliably identify a web browser without relying on the UserAgent string provided by the client.

REFERENCES

- [1] T. Abbott, K. Lai, M. Lieberman, and E. Price. Browser-based attacks on tor. In *Privacy Enhancing Technologies*, pages 184–199. Springer, 2007.
- [2] M. D. Ayenson, D. J. Wambach, and A. Soltani. Flash Cookies and Privacy II: Now with HTML5 and ETag Respawning. 2011.
- [3] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *Proceedings of the 19th international conference on World Wide Web*, pages 281–290. ACM, 2010.
- [4] C.urtsinger, B. Livshits, B. Zorn, and C. Seifert. ZOZZLE: fast and precise in-browser JavaScript malware detection. In *USENIX Security Symposium*, 2011.
- [5] R. Dingleline and N. Mathewson. Anonymity loves company: Usability and the network effect. In *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, Cambridge, UK, June 2006.
- [6] R. Dingleline, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium-Volume 13*, pages 21–21. USENIX Association, 2004.
- [7] P. Eckersley. How Unique is Your Web Browser? In *Privacy Enhancing Technologies*, pages 1–18. Springer, 2010.
- [8] E. ECMA Script, E. C. M. Association, et al. ECMA Script Language Specification. Online at <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.
- [9] M. Egele, E. Kirda, and C. Kruegel. Mitigating drive-by download attacks: Challenges and open problems. *iNetSec 2009–Open Research Problems in Network Security*, pages 52–62, 2009.
- [10] E. W. Felten and M. A. Schneider. Timing Attacks on Web Privacy. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 25–32. ACM, 2000.
- [11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext transfer protocol–HTTP/1.1, June 1999. Status: Standards Track, 1999.
- [12] F. Gordon Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.
- [13] O. Hallaraker and G. Vigna. Detecting Malicious Javascript Code in Mozilla. In *Proceedings 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2005)*, pages 85–94. Ieee, 2005.
- [14] A. Hintz. Fingerprinting websites using traffic analysis. In *Proceedings of the 2nd international conference on Privacy enhancing technologies*, pages 171–178. Springer-Verlag, 2002.
- [15] M. Huber, M. Mulazzani, and E. Weippl. Tor HTTP usage and information leakage. In *Communications and Multimedia Security*, pages 245–255. Springer, 2010.
- [16] Imperva. Imperva Data Security Blog–Deconstructing the Black Hole Exploit Kit, 2011. Online at <http://blog.imperva.com/2011/12/deconstructing-the-black-hole-exploit-kit.html>.
- [17] D. Jang, R. Jhala, S. Lerner, and H. Shacham. An empirical study of privacy-violating information flows in JavaScript web applications. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 270–283. ACM, 2010.
- [18] S. Kamkar. evercookie–never forget. *New York Times*, 2010.
- [19] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. *Dependable and Secure Computing, IEEE Transactions on*, 2(2):93–108, 2005.
- [20] J. R. Mayer. Any person... a pamphleteer: Internet anonymity in the age of web 2.0. *Undergraduate Senior Thesis, Princeton University*, 2009.
- [21] J. R. Mayer and J. C. Mitchell. Third-party Web tracking: Policy and technology. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2012.
- [22] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting Information in JavaScript Implementations. In *Proceedings of Web 2.0 Security and Privacy 2011 (W2SP)*, San Francisco, May 2011.
- [23] K. Mowery and H. Shacham. Pixel Perfect: Fingerprinting Canvas in HTML5. *Proceedings of Web 2.0 Security and Privacy (W2SP) 2012*, 2012.
- [24] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *Security and Privacy, 2005 IEEE Symposium on*, pages 183–195. IEEE, 2005.
- [25] E. E. Network and I. S. Agency. Operation Black Tulip: Certificate authorities lose authority. *Press Release*, 2011.
- [26] L. Olejnik, C. Castelluccia, and A. Janc. Why Johnny Can't Browse in Peace: On the Uniqueness of Web Browsing History Patterns. *5th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETS 2012)*, 2012.
- [27] A. Pfiztmann and M. Köhntopp. Anonymity, unobservability, and pseudonymity—a proposal for terminology. In *Designing privacy enhancing technologies*, pages 1–9. Springer, 2001.
- [28] P. Ratanaworabhan, B. Livshits, and B. G. Zorn. JSMeter: Comparing the behavior of JavaScript benchmarks with real Web applications. In *Proceedings of the 2010 USENIX conference on Web application development*, pages 3–3. USENIX Association, 2010.
- [29] F. Roesner, T. Kohno, and D. Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. In *Proceedings of the 9th USENIX Conference on Networked systems design and implementation (NSDI 2012)*. USENIX Association, 2012.

¹⁰<https://panopticlick.eff.org/>

- [30] M. Smart, G. R. Malan, and F. Jahanian. Defeating TCP/IP stack fingerprinting. In *Proceedings of the 9th USENIX Security Symposium*, volume 24, 2000.
- [31] A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle. Flash Cookies and Privacy. *SSRN preprint (August 2009) <http://papers.ssrn.com/sol3/papers.cfm>*, 2009.
- [32] Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson. I Still Know What You Visited Last Summer: Leaking Browsing History via User Interaction and Side Channel Attacks. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 147–161. IEEE, 2011.
- [33] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A practical attack to de-anonymize social network users. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 223–238. IEEE, 2010.
- [34] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi. Host Fingerprinting and Tracking on the Web: Privacy and Security Implications. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS 2012)*, February 2012.

SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting

The paper *SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting* was published at the Eighth International Conference on Availability, Reliability and Security (ARES) 2013 in Regensburg³⁰.

You can find the paper online at <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6657249>. An extended preprint is available on the author's homepage at http://www.sba-research.org/wp-content/uploads/publications/shpf_extendedPreprint.pdf.

³⁰<http://www.ares-conference.eu/ares2013/www.ares-conference.eu/conf/index.html>

SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting (extended preprint)

Thomas Unger
FH Campus Wien
Vienna, Austria

Martin Mulazzani, Dominik Frühwirt,
Markus Huber, Sebastian Schrittwieser, Edgar Weippl
SBA Research
Vienna, Austria
Email: (1stletterfirstname)(lastname)@sba-research.org

Abstract—Session hijacking has become a major problem in today’s Web services, especially with the availability of free off-the-shelf tools. As major websites like Facebook, Youtube and Yahoo still do not use HTTPS for all users by default, new methods are needed to protect the users’ sessions if session tokens are transmitted in the clear.

In this paper we propose the use of browser fingerprinting for enhancing current state-of-the-art HTTP(S) session management. Monitoring a wide set of features of the user’s current browser makes session hijacking detectable at the server and raises the bar for attackers considerably. This paper furthermore identifies HTML5 and CSS features that can be used for browser fingerprinting and to identify or verify a browser without the need to rely on the UserAgent string. We implemented our approach in a framework that is highly configurable and can be added to existing Web applications and server-side session management with ease. To enhance Web session security, we use baseline monitoring of basic HTTP primitives such as the IP address and UserAgent string, as well as complex fingerprinting methods like CSS or HTML5 fingerprinting. Our framework can be used with HTTP and HTTPS alike, with low configurational and computational overhead. In addition to our contributions regarding browser fingerprinting, we extended and implemented previous work regarding session-based shared secrets between client and server in our framework.

Keywords-Session Hijacking, Browser Fingerprinting, Security

I. INTRODUCTION

Social networks and personalized online services have an enormous daily user base. However, Internet users are constantly at risk. Popular websites like Facebook or Yahoo, along with many others, use HTTPS-secured communication only for user authentication, while the rest of the session is usually transmitted in the clear. This allows an attacker to steal or copy the session cookies, identifiers or tokens, and to take over the session of the victim. Unencrypted Wi-Fi and nation-wide interceptors have used this as an attack vector multiple times recently, proving that session hijacking is indeed a problem for today’s Internet security. A recent prominent example includes the hacked Twitter account of Ashton Kutcher [20], who used an unencrypted Wi-Fi and got hacked— at that time, he had more than six million followers. Tunisia on the other hand was accused of malicious JavaScript injection on websites like Facebook, Gmail and Yahoo, to harvest login credentials and sabotage dissidents online activities [11].

To protect the session of a user, we implemented a framework that ties the session to the current browser by fingerprinting and monitoring the underlying browser, its capabilities, and detecting browser changes at the server side. Our framework, the *Session Hijacking Prevention Framework (SHPF)*, offers a set of multiple detection mechanisms which can be used independently of each other. SHPF protects especially against session hijacking of local adversaries, as well as against cross-site scripting (XSS). The underlying idea of our novel framework: If the user’s browser suddenly changes from, e.g., Firefox on Windows 7 64 bit to an Android 4-based Webkit browser in a totally different IP range, we assume that some form of mischief is happening.

Our framework uses a diverse set of inputs and allows the website administrator to add SHPF with just a few additional lines of code in existing applications. There is no need to change the underlying Web application, and we can use the initial authentication process which is already part of many applications to build further security measurements on top. As part of the authentication process at the beginning of a session, the server asks the browser for an exact set of features and then monitors constantly whether the browser still behaves as expected over the entire session. While an attacker can easily steal unencrypted session information, e.g., on unencrypted Wi-Fi, it is hard to identify the exact responses needed to satisfy the server without access to the same exact browser version. Furthermore, we use a shared secret that is negotiated during the authentication, which is used to sign requests with an HMAC and a timestamp, building and improving on previous work in this direction. Recent attacks against HTTPS in general and the certificate authorities Diginotar and Comodo [22] in particular have shown that even the widespread use of SSL and HTTPS are not sufficient to protect against active adversaries and session hijacking. Previous work in the area of server-side session hijacking prevention relied, e.g., on a shared secret that is only known to the client’s browser [1] and never transmitted in the clear. While this is a feasible approach and can protect a session even for unencrypted connections, our system extends this method by employing browser fingerprinting for session security, thus allowing us to incorporate and build upon existing security mechanisms like HTTPS. Furthermore, it offers

protection against both passive and active adversaries.

Our contributions in this paper are the following:

- We present a framework to enhance HTTP(S) session management, based on browser fingerprinting.
- We propose new browser fingerprinting methods for reliable browser identification based on CSS3 and HTML5.
- We extend and improve upon existing work on using a shared secret between client and server per session.
- We have implemented the framework and will release the code and our test data under an open source license¹.

The rest of the paper is organized as follows: Section II gives a brief technical background. The new browser fingerprinting methods are presented in Section III. Our SHPF framework and its general architecture is described in Section IV. We evaluate our framework in Section V. The results of our evaluation are discussed in Section VI, before we conclude in Section VII.

II. BACKGROUND

Web browsers are very complex software systems requiring multiple person-years of development time. Different international standards like HTML, JavaScript, DOM, XML or CSS specified by the W3C² try to make the browsing experience across different browsers as uniform as possible, but browsers still have their own "touch" in interpreting these standards - a problem Web developers have been struggling with since the infancy of the Web. Due to the complexity of the many standards involved, there are differences in the implementations across browsers. New and upcoming standards further complicate the landscape - HTML5 and CSS3 for example, which are not yet fully standardized but already partly implemented in browsers. These imperfect implementations of standards with different depth are perfectly suited for fingerprinting. *Nmap* [24], for example, uses this exact methodology to identify the operating system used on a remote host based on TCP/IP stack fingerprinting.

Authentication on websites works as follows: A HTML form is presented to the user, allowing them to enter username and password, which are then transmitted to the server. If the login succeeds, the server typically returns a token (often referred to as a session ID), which is subsequently sent along with further client requests to identify the user due to the stateless internals of the HTTP protocol. In an unencrypted HTTP environment, this presents multiple challenges to the user's confidentiality: If login credentials are transmitted in an unencrypted state, an eavesdropping attacker can learn them without any further effort. Even if the document containing the login form as well as the subsequent request containing

the credentials are transmitted over HTTPS, attackers may later learn the session ID from unencrypted requests to the server or by using client-side attacks such as XSS. Thus, it is imperative to enforce SSL throughout the entire site (or at least on those domains that are privileged to receive the session token).

Many administrators regard introducing SSL by default as too cost-intensive. Anecdotal evidence suggests that naively enabling SSL without further configuration may incur significant performance degradation up to an order of magnitude. Gmail, however, switched to HTTPS by default in January 2010 [21]. Remarkably, Google reported that they did not deploy any additional machines and no special hardware (such as hardware SSL accelerators), but employed a number of SSL optimization methods. Only about 10 kB memory per connection, 1% of the CPU load and less than 2% network overhead were incurred for SSL in this configuration. Many other problems with HTTPS have been discussed in the literature, ranging from problems with the CA system [36], [8] to the fact that a large number of keys have been created with weak overall security [12], [23], [44]. While HTTPS can be found on a large number of popular websites that require some form of authentication [13], only a minority of these binds the sessions to a user's device or IP address to protect the user against session hijacking [3].

Multiple tools have been released that allow automated session hijacking: FaceNiff [34], DroidSheep³, Firesheep, cookiemonster [32] and sslstrip, just to name a few. Firesheep was among the first to receive widespread public attention when it was released as open source in 2010 [4]. Firesheep works as follows: Upon startup, Firesheep tries to start sniffing on an IEEE 802.11 or Ethernet device. Whenever HTTP packets are captured and can be parsed as such, they are matched against domain-specific handlers (as of writing, the current git repository includes handlers for sites such as Facebook, Google, and LinkedIn). These handlers store a list of cookie values that comprise a valid session. When a match is found, these values are extracted and an entry for this hijackable session is added to the attacker's sidebar in Firefox. When the attacker selects one of these entries, Firesheep writes the stored cookie values into Firefox' cookie manager and opens the site in a new tab, thereby presenting the attacker with a hijacked and fully operational session of the victim.

III. BROWSER FINGERPRINTING

This section introduces our new browser fingerprinting methods, namely CSS and HTML5 fingerprinting. While browser fingerprinting has ambiguous meanings in the literature i.e., identifying the web browser down to the browser family and version number [7] vs. (re-)identifying a given user [26], we use the former. Our framework relies on fingerprinting to reliably identify a given browser, and CSS fingerprinting is one of the fingerprinting techniques

¹Note to the reviewer: we will include the link here once the paper is accepted for publication

²<http://www.w3.org/TR/>

³<http://droidsheep.de/>

Browser	Layout Engine	Prefix
Firefox	Gecko	-moz-
Konqueror	KHTML	-khtml-
Opera	Presto	-o-
Internet Explorer	Trident	-ms-
Safari	Webkit	-webkit-
Chrome	Webkit	-webkit-

Table I
BROWSER LAYOUT ENGINES AND CSS PREFIXES

implemented in our framework. Furthermore, we present details on how we monitor HTTP headers in SHPF, which allows website administrators to configure advanced policies such as preventing an HTTP session from roaming between a tightly secured internal network and a public network beyond the control of the administrators.

A. CSS Fingerprinting

CSS as a standard is under ongoing development and standardization. CSS 2.1 was published as a W3C Recommendation in June 2011, while the upcoming CSS3 is not yet finished. The CSS3 modules vary in stability and status and while some of them already have recommendation status, others are still candidate recommendations or working drafts. Browser vendors usually start implementing properties early, even long before they become recommendations. We identify three CSS-based methods of browser fingerprinting: CSS properties, CSS selectors and CSS filters.

Depending on the layout engine, progress in implementation varies for new and upcoming CSS properties, which allows us to identify a given browser by the CSS properties it supports. Table I shows which browser uses which layout engine. When properties are not yet on "Recommendation" or "Candidate Recommendation" status, browsers prepend a vendor-specific prefix indicating that the property is supported for this browser type only. Table I also shows the vendor prefixes for the most popular browsers. Once a property moves to Recommendation status, prefixes are dropped by browser vendors and only the property name remains. For example, in Firefox 3.6 the property *border-radius* had a Firefox prefix resulting in *-moz-border-radius*, while in Chrome 4.0 and Safari 4.0 it was *-webkit-border-radius* (as they both use the Webkit layout engine). Since Firefox 4 as well as Safari 5.0 and Chrome 5.0 this feature is uniformly implemented as *border-radius*. The website <https://www.caniuse.com> shows a very good overview on how CSS properties are supported in the different browsers and their layout engine.

Apart from CSS properties, browsers may differ in supported CSS selectors as well. Selectors are a way of selecting specific elements in an HTML tree. For example, CSS3 introduced new selectors for old properties, and they too are not yet uniformly implemented and can be used for browser fingerprinting.

The third method of distinguishing browsers by their behavior is based on CSS filters. CSS filters are used to modify the rendering of e.g., a basic DOM element, image, or video by exploiting bugs or quirks in CSS handling for specific browsers, which again is very suitable for browser fingerprinting. Centricle⁴ provides a good comparison of CSS filters across different browsers.

How to test: As CSS is used for styling websites it is difficult to compare rendered websites at the server side. Instead of conducting image comparison (as used recently by Mowery et al. [29] to fingerprint browsers based on WebGL-rendering), we use JavaScript in our implementation to test for CSS properties in style objects: in DOM, each element can have a style child object that contains properties for each possible CSS property and its value (if defined). These properties in the style object have the same name as the CSS property, with a few differences, for example dashes (-) are removed, the following letter becomes upper case. Vendor-specific prefixes however are preserved if the CSS property has a vendor prefix. An Example: *-moz-border-radius* becomes *MozBorderRadius*.

There are now two ways to test CSS support of a property in the style object: the first way is to simply test whether the browser supports a specific property by using the *in* keyword on an arbitrary style object. The returning Boolean value indicates whether the property is supported. Browser-specific prefixes need to be considered when testing properties with this method. An example: *'borderRadius' in document.body.style*. The second way to test whether a given CSS property is supported is to look at the value of a property once it has been set. We can set an arbitrary CSS property on an element and query the JavaScript *style* object afterwards. Interpreting the return values shows whether the CSS property is supported by the browser: *undefined (null)* as return value indicates that the property is not supported. If a not-*null* value is returned this means the property is supported and has been parsed successfully by the browser.

Care has to be taken when interpreting the return values for fingerprinting: A returning value may deviate from the CSS definition if some parts were not understood by the browser. This can happen, e.g., with composite properties, which allow several sub-properties to be defined in just one long definition. For example, the *background* definition can be used to define *background-color*, *background-repeat*, *background-image*, *background-position* and *background-attachment* all at once. Interestingly, the value string returned upon querying the style object also differs between browsers, and can be used as yet another test for fingerprinting based on CSS properties. For example, consider the following CSS3 background

⁴<http://centricle.com/ref/css/filters/>

definition:

```
background:hsla(56, 100%, 50%, 0.3)
```

Upon testing the background property on the style object as described above, Firefox returns the following:

```
none repeat scroll 0% 0% rgba(255, 238, 0, 0.3)
```

Internet Explorer, on the other hand, returns this:

```
hsla(56, 100%, 50%, 0.3)
```

As this shows, Firefox returns all possible values of the composite background property explained above (repeat, color, image, position) and additionally converts the *hsla* definition to *rgba* values. In contrast, Internet Explorer only returns exactly what was stated in the CSS definition, no more and no less, and does not convert the values into another format. The order of elements within the return string for composite values may also deviate between browsers, for example with the *box-shadow* property with distance values as well as color definitions.

B. HTML5 Fingerprinting

HTML5, like CSS3, is still under development, but there are already working drafts which have been implemented to a large extent by different browsers. This new standard introduces some new tags, but also a wide range of new attributes. Furthermore HTML5 specifies new APIs (application programming interfaces), enabling the Web designer to use functionalities like drag and drop within websites. Since browser vendors have differing implementation states of the new HTML5 features, support for the various improvements can be tested and used for fingerprinting purposes as well. For identifying the new features and to what extent they are supported by modern browsers, we used the methodology described in [33]. The W3C furthermore has a working draft on differences between HTML5 and HTML4 that was used as input [38].

In total we identified a set of 242 new tags, attributes and features in HTML5 that were suitable for browser identification. While 30 of these are attributed to new HTML tags that are introduced with HTML5 [41], the rest of the new features consist of new attributes for existing tags as well as new features. We then created a website using the Modernizr [2] library to test for each of these tags and attributes and whether they are supported by a given browser. We collected the output from close to 60 different browser versions on different operating systems. An excerpt of the data and how the tags and attributes are supported by different browsers can be seen in Table II. One of our findings from the fingerprint collection was that the operating system apparently has no influence on HTML5 support. We were unable to find any differences between operating systems while using the same browser version, even with different architectures. An example: Firefox 11 on Windows XP (32 bit) and on Windows 7 (64 bit) share the same fingerprint.

C. Basic HTTP Header Monitoring

For each HTTP request, a number of HTTP headers is included and transmitted to the Web server. RFC 2616 defines the HTTP protocol [10] and specifies several HTTP headers that can or should be sent as part of each HTTP request. The number of headers, the contents and especially the order of the header fields, however, are chosen by the browser and are sufficient for identifying a browser. Using this method for browser identification has already been discussed in previous work [7], [43] and is already used to some extent by major websites [3], we will thus only briefly cover the parts which are of particular interest for SHPF. In our implementation we use the following header fields for HTTP session monitoring:

- **UserAgent string** contains browser version and platform information.
- **Accept** specifies which data types the browser supports. It is also used to announce a preference for a certain data type.
- **Accept-Language** specifies, similar to Accept, which language is preferred by the browser.
- **Accept-Encoding** specifies which encodings are supported and which encoding is preferred by the browser.
- **IP-Address** of the client is not part of the HTTP header. However, the client IP address can be processed by the server easily.

The UserAgent contains information about the browser - often the exact browser version and the underlying operating system. It is, however, not a security feature, and can be changed arbitrarily by the user. SHPF is not depending on the UserAgent, and works with any string value provided by the browser. If the UserAgent changes during a session this is a strong indication for session hijacking, especially across different web browsers. Depending on the configuration of SHPF and the particular security policy in place, it might however be acceptable to allow changes in the browser version e.g., with background updates of the browser while using a persistent session if the browser is restarted.

The UserAgent as well as the other headers and data usually remain consistent during a session. If any values or a subset of these values change during a session, the session has been hijacked (in the simplest case). For example, if during a session multiple UserAgents from different IPs use the same session cookie, this implies in our framework that the session has been hijacked (session identifiers ought to be unique). The session would be terminated immediately and the user would need to reauthenticate. In order to bypass this part of our framework, the attacker would need to replicate all the original headers and use the same IP address as the client in order to send valid requests to the server. While cloning the HTTP headers is rather easy, binding a session to a given IP address considerably raises the bar for adversaries, even if they can obtain a valid session cookie and the HTTP header with e.g., one of various

Tag	Attribute	FF12	FF13	C18	C19	IE8	IE9	O11	O12	S4	S5
<audio>	—	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗
<fieldset>	name	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗
<textarea>	maxlength	✓	✓	✓	✓	✗	✗	✓	✓	✗	✓
<nav>	—	✓	✓	✓	✓	✗	✓	✗	✓	✗	✓
<meter>	—	✗	✗	✓	✓	✗	✗	✓	✓	✗	✗
<input>	type="url"	✓	✓	✓	✓	✗	✗	✓	✓	✗	✓
<canvas>	—	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓

Table II
EXCERPT OF HTML5 TAGS AND ATTRIBUTES FOR BROWSER FINGERPRINTING

kinds of cross-site scripting (XSS) attack [37].

Apart from the HTTP header values themselves, there is also a significant difference in how the browsers order the HTTP header fields. While Internet Explorer 9 for example sends the UserAgent before the Proxy-Connection and Host header fields, Chrome sends them in the exact opposite order. The content of the header fields is not important in this case, all header fields are included for this check in our implementation. HTTP header ordering is especially useful against session hijacking tools like that clone only the UserAgent or copy the session cookie, but not the rest of the header information.

IV. SHPF FRAMEWORK

This section describes the implementation of our framework and its architecture, the *Session Hijacking Prevention Framework (SHPF)*. The source code is released under an open source license and can be found on github⁵. Despite the new fingerprinting methods presented in the previous section, we also implemented and improved SessionLock [1] for environments that do not use HTTPS by default for all connections.

A. General Architecture

SHPF is a server-side framework which is written in PHP5 and consists of multiple classes that can be loaded independently. Its general architecture and basic functionality is shown in Figure 1. We designed it to be easily configurable (depending on the context and the security needs of the website), portable and able to handle a possibly large number of concurrent sessions. Our implementation can be easily extended with existing and future fingerprinting methods, e.g., textfont rendering [29] or JavaScript engine fingerprinting [28], [35].

The main parts of the framework are the so-called *features*. A feature is a combination of different checks for detecting and mitigating session hijacking. In our prototype we implemented the following features: HTTP header monitoring, CSS fingerprinting and SecureSession (which implements and extends the SessionLock protocol by Ben Adida). Features are also the means to extending the framework, and we provide base classes for fast feature

development. A feature consists of one or more *checkers*, which are used to run certain tests. There are two different types (or classes) of checkers:

- *Synchronous checkers* can be used if the tests included in the checker can be run solely from existing data, such as HTTP requests or other website-specific data that is already available.
- *Asynchronous checkers* are used if the tests have to actively request some additional data from the client and the framework has to process the response.

While synchronous checkers are passive in nature, active checkers can challenge the client to send some information for a specific checker, allowing the server to verify that the browser is behaving as expected. Client responses are sent via asynchronous calls (AJAX) as part of SHPF, thus not blocking the session or requiring to rewrite any existing code. Appendix A shows the basic code needed to incorporate SHPF into a website.

The distinction between features and checkers gives the website control over which checks to run. Features can be disabled or enabled according to the website's security needs, and can be assigned to different security levels. Different security levels within a webpage are useful, for example, in privacy-heterogeneous sessions - basic checks are performed constantly, while additional checks can be run only when necessary, e.g., when changing sensitive information in a user's profile (much like Amazon does for its custom session management). In order to communicate with a Web application, callbacks can be defined both in PHP and JavaScript. These callbacks are called if a checker fails and thus allow the application to react in an appropriate way, e.g., terminate the session and notify the user. An example configuration for different security levels with SHPF can be seen in Table III. The details for each checker in this example are explained in detail below. Consider a website, e.g., a web store, which uses three different security levels for every session:

- Level 1 is for customers who are logged in and browsing the web store.
- Level 2 is for customers who are in a sensitive part of their session, e.g., ordering something or changing their profile.
- Level 3 is for administrators who are logged into the administrative interface.

Level 1 is a very basic security level. In this example it

⁵<https://github.com/mmulazzani/SHPF>

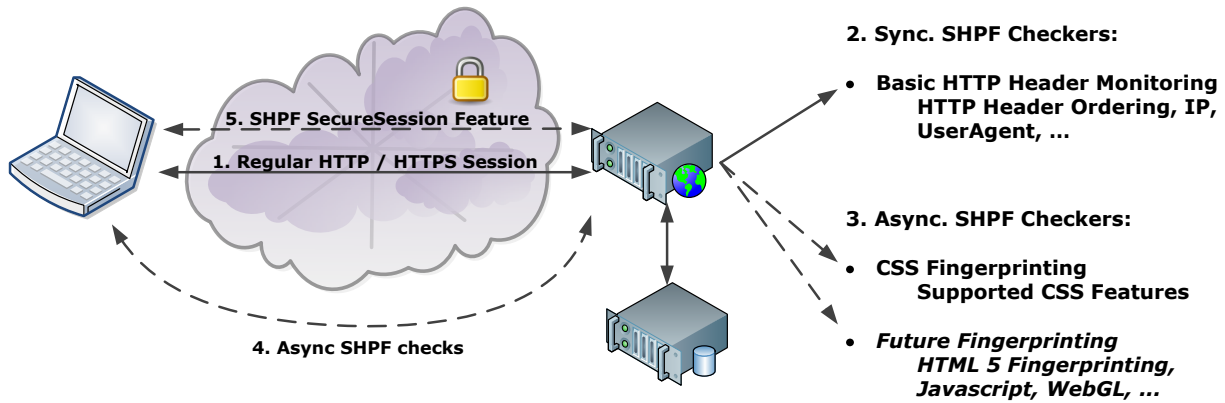


Figure 1. SHPF Architecture

prevents session hijacking by monitoring the UserAgent string of the user for modifications. As a sole security measure it only protects the user against attacks that can be considered a nuisance, and can possibly be bypassed by an attacker (by cloning the UserAgent string). The Web application is designed in such a way that an attacker cannot actively do any harm to the user, for example browsing only specific products to manipulate the web store's recommendation fields. If the customer decides to buy something, level 2 is entered, which uses two additional security measures: the current session is locked to the user's IP address and the order of the HTTP headers is monitored to detect if a different browser uses the same UserAgent string. Once the transaction is complete, the customer returns to level 1. For an administrator, even more checkers are enabled at the start of the session: SecureSession protects the session cryptographically with a shared secret between the particular browsers that started the sessions, and the CSS properties supported by the browser are monitored. Please note that this configuration is given merely by way of an example and must be matched to existing security policies when implemented. Furthermore, note that HTTPS is not mentioned in the example - even though it is strongly advised to use HTTPS during a session (besides SHPF), it is not a requirement. SHPF can prevent session hijacking even if session tokens are transmitted in the clear.

Checks	Security Levels		
	Level 1	Level 2	Level 3
UserAgent monitoring	✓	✓	✓
IP binding	✗	✓	✓
HTTP Header ordering	✗	✓	✓
CSS fingerprinting	✗	✗	✓
SecureSession	✗	✗	✓

Table III

EXAMPLE - DIFFERENT SECURITY LEVELS FOR A WEBSITE

Additional components in our framework are used for keeping track of the session state in a database, for output

and logging, and there is a special class for integrating the framework into existing websites and a crypto feature *CryptoProvider*. The crypto feature defines methods of encrypting and decrypting data. If a crypto provider is set in SHPF and SecureSession is used, all asynchronous messages exchanged between the browser and the framework are automatically encrypted by the crypto provider (see Section IV-D).

B. Basic HTTP Header Monitoring

The HTTP header monitoring feature does the following:

- 1) On the first request, the framework stores the contents and the order of the HTTP headers as described above.
- 2) For each subsequent request, the feature compares the headers sent by the client with the stored ones and checks whether their content and/or order match.

Depending on the particular use case, different configurations are possible, e.g., binding a session to a given IP, a certain IP range or a UserAgent string. Another example would be to allow IP address roaming while enforcing that the operating system as claimed by the UserAgent string as well as the browser has to stay the same, allowing the browser version to change, e.g., through background updates in Chrome or Firefox. HTTP header monitoring is implemented as a synchronous checker, as the data needed for processing is sent with every request.

C. CSS Fingerprinting

Using the CSS fingerprinting methods explained above, a SHPF feature has been implemented that does the following:

- 1) Check whether the client's browser supports JavaScript.
- 2) On the first request of the client: Run the complete fingerprinting suite on the client (using 23 CSS properties at the time of writing) and save the values.
- 3) For each subsequent request of the client: choose a subset of CSS properties and test them on the client.

- 4) Receive the data and check if it was requested by the framework (anti-replay protection).
- 5) Compare the values with the saved values.

As this feature needs data from the client, this checker has been implemented as an asynchronous checker. The client is challenged to answer a subset of the previously gathered properties either for each HTTP request or within a configurable interval between CSS checks (say, every 10 or 20 requests). By default, the framework tests three CSS properties and compares the results with the previously collected fingerprint of that browser. The data received asynchronously must match the requested properties and must arrive within a configurable time span. If the received data do not match the expected data, arrive too late or are not requested by the feature, the session is terminated immediately. If no response is received within a configurable time span, the session is terminated as well. SHPF may be also configured to terminate the session if no JavaScript is enabled, thus making CSS fingerprinting mandatory by policy.

In order to reliably identify a given browser, we selected a mix of CSS3 properties that are not uniformly supported by current browsers. In total, 23 properties were identified as suitable for fingerprinting. The website <http://www.canuse.com> was used to identify CSS properties that are not uniformly compatible across browsers, as well as properties that still have vendor prefixes. The 23 identified properties, possible testing values, and their status in the standardization and implementation process are shown in Table IV. Please note that in some cases merely the additional values of an already existing property are new, while the CSS property itself is not a novel CSS feature.

For each CSS property, an empty HTML `<div>` element is inserted into the page, which contains an inline CSS definition. The element is assigned an ID so that it can be later accessed with JavaScript. Such an element might, e.g., look like this:

```
<div id="cssCheck1" style="min-width:35px;"></div>
```

JavaScript is then used to check whether the properties set previously exist in the style object, and also to query the property's value. The answers from the client are collected in an array, which is then converted into JSON and sent to the server secured by HTTPS or the SecureSession feature against eavesdroppers.

```
[ "minWidth" in
  $("cssCheck1").style,$("cssCheck1").style.minWidth ]
```

For our implementation of CSS fingerprinting in SHPF we chose to use CSS properties only - CSS selectors were not used because CSS properties are sufficient to reliably identify a given browser. Nonetheless, the framework could be extended by supporting CSS selector and CSS filter fingerprinting in the future.

D. SecureSession

The SecureSession feature implements the SessionLock protocol by Ben Adida [1], but extends and modifies it in

certain aspects:

- SessionLock utilizes HTTPS to transfer the session secret to the client. In our SecureSession feature we use a Diffie-Hellman Key Exchange [5] as discussed by Ben Adida in his paper because of the recent attacks against the trust foundation of HTTPS (Digiprotar, Comodo) to do the same. We also looked at performance and security implications of that choice.
- We use the new WebStorage [40] features implemented by modern browsers by using JavaScript and the *localStorage* object to store the session secret.
- We improved patching of URLs in JavaScript compared to the original protocol.

SessionLock used the URL Fragment Identifier to keep the session secret around but hidden in the network traffic. For that, each URL needs to be patched so that the fragment gets appended. Using WebStorage is superior in multiple ways. If WebStorage is not supported by the browser, SecureSession falls back to using the fragment identifier. SessionLock furthermore hooks into the XMLHttpRequest object to intercept and modify asynchronous messages. We determined that there are cross-browser issues in using this method. In order to improve compatibility across browsers, we used the modified XMLHttpRequest object by Sergey Ilinsky [16] to make message interception compatible across different browsers.

In order to implement the above features we used two checkers for the framework feature:

- The **SecureSessionSecretNegotiation-Checker** is an asynchronous checker. The server has to run a Diffie Hellman Key Exchange only if no valid session secret is present. The server first calculates its private and public parts, sends them to the client as JavaScript code, and receives the client parts asynchronously in response when the client is done calculating. Both sides can then calculate the shared session secret.
- The **SecureSessionSecretChecker-Checker** is a synchronous checker that validates all incoming requests regarding HMAC and timestamp.

The SecureSessionSecretNegotiation initiates the key exchange by sending JavaScript to the client containing the calculations as well as the prime, generator and public number. The client sends its public number back via an asynchronous call. The server assumes that JavaScript is disabled if it receives no answer from the client within a (configurable) period of time. Again, SHPF can be configured to make this feature mandatory. If an answer is received, all further requests need to be appended with a valid HMAC and timestamp (configurable). This is done by the SecureSessionSecretChecker. While the method is the same as in SessionLock, we ported it to PHP. However, there is an exception to the rule: As Ben Adida discussed in his paper, there may be cases where a URL is not valid, such as when a page is opened from a bookmark. In such a case, the feature allows a configurable amount of consecutive requests that may

CSS Status - Recommendation		CSS Status - Working Draft	
Feature	Value	Feature	Value
display	inline-block	transform	rotate(30deg)
min-width	35px	font-size	2rem
position	fixed	text-shadow	4px 4px 14px #969696
display	table-row	background	linear-gradient (left, red, blue 30%, green)
opacity	0.5	transition	background-color 2s linear 0.5s
background	hsla(56, 100%, 50%, 0.3)	animation	name 4s linear 1.5s infinite alternate none
		resize	both
		box-orient	horizontal
CSS Status - Cand. Recommendation			
Feature	Value		
box-sizing	border-box	transform-style	preserve-3d
border-radius	9px	font-feature-setting	dlig=1,ss01=1
box-shadow	inset 4px 4px 16px 10px #000	width	calc(25% - 1em)
column-count	4	hyphens	auto
		object-fit	contain

Table IV
23 CSS PROPERTIES AND VALUES IDENTIFIED FOR CSS FINGERPRINTING

fail. If a valid request is received before that amount is exceeded, no action is taken. To make the key exchange secure against MITM attacks, this feature should only be used on top of HTTPS or a secure, offline communication channel for exchanging the parameters and the JavaScript code.

For implementation we used the *Crypt_DiffieHellman* library from the PEAR Framework⁶ on the server side. On the client, we used the *Big Integer Library* of Leemon Baird⁷. The SecureSession feature also implements a CryptoProvider. The CryptoProvider offers AES-CBC encryption by using the SHA-1 hash of the session secret negotiated between the framework and the client as the key. For PHP, the PHP extension *mcrypt*⁸ is used, for JavaScript we use the library *crypto-js*⁹. The CryptoProvider then encrypts all asynchronous messages from the client to the framework. Furthermore, it prepends a timestamp to the plaintext before encryption, thus preventing replay attacks if the age of the timestamp exceeds a configurable time span.

E. Further Fingerprinting Methods

Our framework is especially designed to allow new and possibly more sophisticated fingerprinting methods to be added at a later point in time by implementing them as additional checkers. The presented results on HTML5 fingerprinting above, e.g., have not yet been implemented at the time of writing. We are planning to implement HTML5 fingerprinting as an asynchronous checker in the near future. Other fingerprinting methods e.g., EFF's Panopticlick, can be added at ease adding 18.1 bits of entropy on average [7]. See Section VI-A for related work and other fingerprinting methods which could be added to SHPF.

V. EVALUATION

There are multiple possible attack vectors that enable an attacker to obtain session tokens of any kind and take

over the victim's session. We will discuss for each attack vector how SHPF can detect session hijacking and how it prevents it.

A. Threat Model

An attacker in our threat model can be local or remote from the victim's point of view, as well as either active or passive. While a passive attacker just listens without interacting with the client, an active attacker sends, modifies or actively drops communication content. Different requirements have to be met for each of the outlined attacks, however, these are beyond the scope of this paper.

Figure 2 shows an overview of the different points of attack that were considered while designing SHPF. They are based on the OWASPs Top 10 from 2010¹⁰, which has multiple categories that either directly allow session hijacking, or facilitate it. The most notable categories are "A2 Cross-Site Scripting", "A3 Broken User Authentication and Session Management" and "A9 Insufficient Transport Layer Protection". We particularly considered threats that are actively exploited in the wild, with tools available for free.

The following points of attack allow an attacker to hijack a session:

- 1) Different attacks where the attacker has access to the victim's network connection.
- 2) The target website is vulnerable to code injection attacks (XSS), pushing malicious code to the client.
- 3) Local code execution within the victim's browser's sandbox, e.g., by tricking the victim into executing Javascript (besides XSS).
- 4) Attacker has access to 3rd party server with access to the session token, e.g., a proxy, Tor sniffing or via HTTP referrer string.

The detailed attack descriptions for each of these attacks are as follows: 1) If the attacker is on the same network as the victim, e.g., on unencrypted Wi-Fi, searching for unencrypted session tokens is trivial - these are the methods used, for example, by Firesheep and

⁶http://pear.php.net/package/Crypt_DiffieHellman

⁷<http://leemon.com/crypto/BigInt.html>

⁸<http://php.net/manual/en/book.mcrypt.php>

⁹<https://code.google.com/p/crypto-js/>

¹⁰https://owasp.org/index.php/Top_10

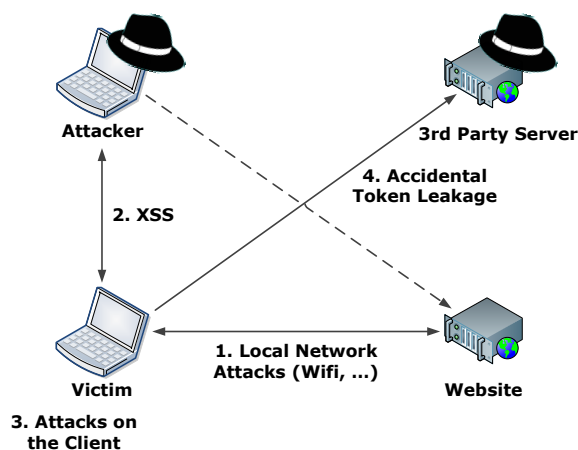


Figure 2. Attack Points for Session Hijacking

FaceNiff. In case the connection between victim and website is encrypted with HTTPS, the attacker might use `sslstrip` [25] or `cookiemonster` [32], as HTTPS as a sole countermeasure against session hijacking has been shown to often be insufficient. The token could also be obtained by an active attacker on the same network by means of ARP or DNS spoofing, redirecting the victim's communication in a man-in-the-middle attack, or DNS cache poisoning [18]. 2) If an attacker is able to inject Javascript into the website which is then executed at the victim side (XSS), he can transmit all necessary session tokens to himself and take over the session. 3) An attacker could access session tokens by attacking the browser directly using social engineering or a malicious browser extension, e.g., by tricking a victim into copy-pasting some Javascript into the URI bar of the browser. 4) In case of a poorly implemented Web application (HTTP referrer string), insecure transport (HTTP only) or network design (logging proxy server), an attacker might be able to access accidentally leaked tokens. This class of attacks would also include shoulder surfing (if the token is part of the URI) and improper usage of the Tor [6] anonymization network [27], [14].

B. Discussion

To counter the attacks listed above, SHPF relies on a combination of its features: the shared secret between the server and client using the `SecureSession` feature, and session hijacking detection using browser fingerprinting. An attacker would thus have to find out the secret, share the same IP and copy the behavior of the victim's browser - either by running the same browser version on the same operating system or by collecting the behavior of the browser over time.

The basic monitoring of HTTP information gives a baseline of protection. Binding a session to, e.g., an IP address makes it considerably harder for a remote attacker to attack, and a local attacker needs to be on

the same local area network if the victim is behind NAT. Changes in the `UserAgent` or the HTTP header ordering are easily detectable, especially if careless attackers use sloppy methods for cloning header information, or only use some parts of the header for their user impersonation: `Firesheep` and `FaceNiff`, for example, both parse the header for session tokens instead of cloning the entire header. A recent manual analysis of the Alexa Top100 pages showed that only 8% of these very popular websites use basic monitoring in any form - notably eBay, Amazon and Apple [3]. Even though asynchronous challenges for fingerprinting on the attacker's machine could also simply be forwarded to the victim for the correct responses, the additional delay is detectable by the server.

We shall now discuss for each of the attacks outlined above how SHPF protects the session through active attack detection and prevention. Even though SHPF could work without HTTPS in certain configurations and environments, it should be used for starting the session, as without HTTPS bootstrapping the session becomes complicated, e.g., with respect to possible MITM attacks. As HTTPS is already used widely for user authentication, we assume that it is available at least for bootstrapping the `SecureSession` feature. SHPF has the following impact on the attack vectors: 1) Snooping or redirecting local network traffic can be detected at the server with either browser fingerprinting or using the shared secret, which is never transmitted in clear from `SecureSession` - both methods are equally suitable. 2) Cross-site scripting prevention relies on browser fingerprinting only, as the attacker could obtain session tokens by executing Javascript code in the victim's browser. The shared secret is not protected against such attacks. 3) Local attacks are also detected by browser fingerprinting only - the session secret is not safe, thus the attacker has to either run the same browser, or answer the asynchronous checks from the framework correctly. 4) Accidental token leakage is again prevented by both aspects, so even if the session is not encrypted by HTTPS the content is encrypted by the `SecureSession` feature and fingerprinting is used to detect changes in the used browser. Please see the original paper about `SessionLock` [1] for a detailed security analysis of the protocol.

SHPF does not intend to replace traditional security features for web sessions. While our approach cannot prevent session hijacking entirely it makes it considerably harder for the attacker. For sensitive websites with a high need for security, additional measures like 2-factor authentication or client-side certificates should be employed.

C. Limitations

Even though SHPF makes session hijacking harder, it has limitations: the HTTP headers and their ordering, as well as the `UserAgent`, are by no means security measures and can be set arbitrarily. However, if enough information specific to a browser is used in combination with ever

shorter update intervals for browsers, we believe that fingerprinting is suitable for preventing session hijacking. Secondly, SHPF does not protect against CSRF: An attacker who is able to execute code outside of the browser's sandbox, or has access to the hardware, can bypass our framework. Thus session hijacking is made harder in the arms race with the adversary, but not entirely prevented. Another limitation is the vulnerability to man-in-the-middle attacks: Diffie-Hellman in Javascript for shared secret negotiation is vulnerable to MITM, and either a secure bootstrapping process for session establishment or offline multifactor authentication is needed to protect the session against such adversaries.

D. Future Work

We plan to extend CSS fingerprinting with CSS selectors and CSS filters and intend to implement HTML5 fingerprinting as an additional SHPF feature. We furthermore plan to assess the tradeoff between the numbers of asynchronous challenges sent by the server to the total pool size of challenges for recent browser versions, as well as to measure the entropy of each fingerprinting method in practice. Even though the SHPF features for CSS (and soon HTML5 fingerprinting) are not designed to be used as single-use challenges within a session, we believe that measuring the entropy on a large set of users would be beneficial for the area of browser fingerprinting.

VI. RESULTS

In general, the performance impact of running SHPF on the server is negligible as most of the processing is implemented as simple database lookups. Only a few kilobytes of RAM are needed per session and client for all features combined, while the overhead on the network is around 100 kilobytes (mostly for the libraries used by our framework - they need to be transferred only once due to browser caching). A mere 15 lines of code are needed to include SHPF in existing websites (see Appendix A), while the features each consist of a few hundred lines of code on average, with SecureSession being by far the biggest feature (about 4000 lines). Existing Web applications implement far more complicated logic flows and information processing capabilities than SHPF.¹¹

The biggest impact on performance is caused by the generation of the primes for the Diffie-Hellman key exchange. We used a small but diverse set of devices to assess the clients' performance for creating the shared secret: a notebook (i7 CPU with 2 Ghz), a netbook (AMD Sempron with 1.5 Ghz) and two different smartphones (iPhone 4S and Nexus S). On the notebook, the latest versions of Chrome and Firefox at the time of writing (Chrome 18 and Firefox 12) were the fastest browsers for this operation, while Internet Explorer 9 was up to four times slower. As smartphones are limited with regard to CPU performance, they were even slower. A comparison

¹¹We will release our datasets along with the source code once the paper is accepted.

of runtime needed for generating primes of different length can be seen in Figure 3. Depending on the security need of the website this overhead should be considered, as well as the amount of expected mobile users. The overhead caused by CSS fingerprinting on the client side is negligible compared to regular website rendering.

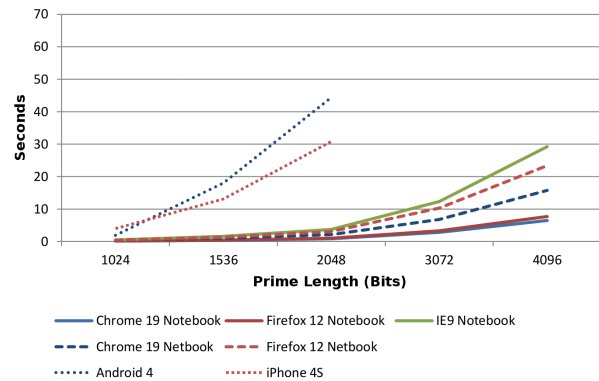


Figure 3. Performance of Prime Number Generation

A. Related Work

In the area of browser fingerprinting, different approaches have been used to identify a given browser. Panoptlick¹² relies on the feature combination of UserAgent string, screen resolution, installed plugins and more to generate a unique fingerprint [7] that allows the tracking of a given browser even if cookies are disabled. Even though the features of Panoptlick, such as screen resolution or installed browser plugins, are not yet fully incorporated in our framework, we are planning to do this in the near future. Other recent work in the area of browser fingerprinting identifies a client's browser and its version as well as the underlying operating system by fingerprinting the JavaScript engine [9]. While the approach in [28] uses various well-known JavaScript benchmarks to generate a unique fingerprint based on timing patterns, [35] employs a JavaScript conformance test to identify subtle differences in the conformance of the underlying JavaScript engine. Another recent method uses website rendering differences as presented in [29]. Like SHPF, these methods allow the detection of a modified or spoofed UserAgent string, as it is not possible to change the behavior of core browser components like the rendering or the JavaScript engine within a browser.

With regards to privacy, cookies and browser fingerprinting can be employed to track a user and their online activity. A survey on tracking methods in general can be found in [26]. Other work has recently shown that the UserAgent is often sufficient for tracking a user across multiple websites or sessions [43]. Intersection attacks on browsing history [31] or social networking

¹²<https://panoptlick.eff.org>

sites [42] can be used to identify users. Session hijacking has been shown to allow access to sensitive information on social networking sites [15]. Finally, session hijacking is often conducted using cross-site scripting (XSS) attacks that are used to send the session information to an attacker. While this can be employed to protect a user from entering the password at an insecure terminal [3], it is often used maliciously, e.g., to impersonate the victim. Different approaches have been implemented to protect users from XSS on the client side [19], [39], [30] as well as on the server side [17].

The OWASP AppSensor project¹³ is a framework that offers similar features as SHPF for Web applications: It can detect anomalies within a session and terminate it if necessary. However, it only uses a very limited set of checks compared to SHPF, namely the IP and the UserAgent string.

VII. CONCLUSION

In this paper, we presented our framework SHPF, which is able to raise the bar for session hijacking significantly. It detects and prevents attacks and hijacking attempts of various kinds, such as XSS or passive sniffing on the same network (Wi-Fi). We furthermore proposed two new browser fingerprinting methods based on HTML5 and CSS, which can identify a given browser. SHPF uses browser fingerprinting to detect session hijacking by constantly checking (e.g., with every request) if the browser is still behaving as it did when the session was started. SHPF can be configured to run with different security levels, allowing additional security checks for sensitive sessions or session parts. Future and upcoming fingerprinting methods can be incorporated easily.

APPENDIX

APPENDIX A - SHPF EXAMPLE

```
include ('../SHPF/SHPF.php');

$shpf = new SHPF($SHPF ());
$shpf->setCheckFailedHandler ('failedHandler');
$shpf->getOutput ()->includeJSLibrary = false;

$serverEnvFeature = new SHPF\Features\HttpHeader\HttpHeaderFeature ($shpf);
$serverEnvFeature->setCheckAll (true);
$shpf->addFeature ($serverEnvFeature);

$shpf->addFeature (new SHPF\Features\SecureSession\SecureSessionFeature ($shpf));
$shpf->addFeature (new SHPF\Features\CSSFingerprint\CSSFingerprintFeature ($shpf));

$ret = $shpf->run ();

$output = Registry::get ('smarty');

$output->append ($shpf->getOutput ()->flushHead(true), 'head');
$output->append ($shpf->getOutput ()->flushHTML(true));
$output->append ($shpf->getOutput ()->flushJS(true));
```

REFERENCES

- [1] B. Adida. Sessionlock: Securing web sessions against eavesdropping. In *Proceeding of the 17th International Conference on World Wide Web (WWW)*, pages 517–524. ACM, 2008.
- [2] F. Ates and P. Irish. Modernizr - front-end development done right. Online at <http://modernizr.com/>, 2006.
- [3] E. Bursztein, C. Soman, D. Boneh, and J. Mitchell. Session-juggler: secure web login from an untrusted terminal using session hijacking. In *Proceedings of the 21st international conference on World Wide Web*, pages 321–330. ACM, 2012.
- [4] E. Butler and I. Gallagher. Hey web 2.0: Start protecting user privacy instead of pretending to. *ToorCon 2010*, 2010.
- [5] W. Diffie and M. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.
- [6] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.
- [7] P. Eckersley. How unique is your web browser? In *Proceedings of Privacy Enhancing Technologies (PETS)*, pages 1–18. Springer, 2010.
- [8] P. Eckersley and J. Burns. Is the ssliverse a safe place? *Talk at 27C3. Slides from https://www.eff.org/files/ccc2010.pdf/online*, 2011.
- [9] E. ECMAScript, E. C. M. Association, et al. EcmaScript language specification. Online at <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.
- [10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616: Hypertext transfer protocol-http/1.1, 1999. Online at <http://www.rfc.net/rfc2616.html>, 1999.
- [11] D. Goodin. Tunisia plants country-wide keystroke logger on facebook. Online at http://www.theregister.co.uk/2011/01/25/tunisia_facebook_password_slurping/, retrieved 2012-11-19.
- [12] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Proceedings of the 21st USENIX Security Symposium*, Aug. 2012.
- [13] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The ssl landscape: a thorough analysis of the x. 509 pki using active and passive measurements. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 427–444. ACM, 2011.
- [14] M. Huber, M. Mulazzani, and E. Weippl. Tor http usage and information leakage. In *Communications and Multimedia Security*, pages 245–255. Springer, 2010.
- [15] M. Huber, M. Mulazzani, and E. Weippl. Who On Earth Is Mr. Cypher? Automated Friend Injection Attacks on Social Networking Sites. In *Proceedings of the IFIP International Information Security Conference 2010: Security and Privacy (SEC)*, 2010.
- [16] S. Ilinsky. Xmlhttprequest.js - cross-browser xmlhttprequest 1.0 object implementation, 2007. Online at <http://www.ilinsky.com/articles/XMLHttpRequest/>.
- [17] M. Johns. Sessionsafe: Implementing xss immune session handling. *Computer Security-ESORICS 2006*, pages 444–460, 2006.

¹³https://www.owasp.org/index.php/OWASP_AppSensor_Project

- [18] D. Kaminsky. Black ops 2008—it's the end of the cache as we know it. *Black Hat USA*, 2008.
- [19] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 330–337. ACM, 2006.
- [20] M. Kirkpatrick. Ashton kutcher's twitter account hacked at ted. Online at http://www.readwriteweb.com/archives/ashton_kutchers_twitter_account_hacked_at_ted.php, retrieved 2012-07-19.
- [21] A. Langley, N. Modadugu, and W.-T. Chang. Overclocking ssl. In *Velocity 2010*, 2010.
- [22] N. Leavitt. Internet security under attack: The undermining of digital certificates. *Computer*, 44(12):17–20, 2011.
- [23] A. Lenstra, J. Hughes, M. Augier, J. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, whit is right. *IACR eprint archive*, 64, 2012.
- [24] G. F. Lyon. *Nmap Network Scanning: The Official Nmap Project Guide To Network Discovery And Security Scanning*. Nmap Project, 2009.
- [25] M. Marlinspike. sslstrip. Online at <http://www.thoughtcrime.org/software/sslstrip/>, retrieved 2012-07-26.
- [26] J. Mayer and J. Mitchell. Third-party web tracking: Policy and technology. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 413–427. IEEE, 2012.
- [27] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. Shining light in dark places: Understanding the tor network. In *Privacy Enhancing Technologies*, pages 63–76. Springer, 2008.
- [28] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting information in javascript implementations. In *Proceedings of Web 2.0 Security & Privacy Workshop (W2SP)*, 2011.
- [29] K. Mowery and H. Shacham. Pixel perfect: Fingerprinting canvas in html5. In *Proceedings of Web 2.0 Security & Privacy Workshop (W2SP)*, 2012.
- [30] N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen. Sessionshield: lightweight protection against session hijacking. *Engineering Secure Software and Systems (ESSOS)*, pages 87–100, 2011.
- [31] Ł. Olejnik, C. Castelluccia, and A. Janc. Why johnny can't browse in peace: On the uniqueness of web browsing history patterns. In *5th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs)*, 2012.
- [32] M. Perry. Cookiemonster: Cookie hijacking. Online at <http://fscked.org/projects/cookiemonster>, 2008.
- [33] M. Pilgrim. Dive into html5, 2010.
- [34] B. Ponurkiewicz. Faceniff. Online at <http://faceniff.ponury.net>, 2011.
- [35] P. Reschl, M. Mulazzani, M. Huber, and E. Weippl. Poster abstract: Efficient browser identification with javascript engine fingerprinting. *Annual Computer Security Applications Conference (ACSAC)*, 12 2011.
- [36] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. Osvik, and B. de Weger. Md5 considered harmful today. *Creating a rogue CA certificate*, 2008.
- [37] D. Stuttard and M. Pinto. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. Wiley, 2011.
- [38] A. van Kesteren. Html 5 differences from html 4. *Working Draft, W3C*, 2008.
- [39] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross-site scripting prevention with dynamic data tainting and static analysis. In *Proceeding of the Network and Distributed System Security Symposium (NDSS)*, volume 42, 2007.
- [40] W3C. Webstorage, 2011. Online at <http://www.w3.org/TR/webstorage/>.
- [41] w3schools. Html5 tag reference. Online at http://www.w3schools.com/html5/html5_reference.asp, 2012.
- [42] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A practical attack to de-anonymize social network users. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 223–238. IEEE, 2010.
- [43] T. Yen, Y. Xie, F. Yu, R. Yu, and M. Abadi. Host fingerprinting and tracking on the web: Privacy and security implications. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS)*. NDSS, 2012.
- [44] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public: results from the 2008 debian openssl vulnerability. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 15–27. ACM, 2009.

Anonymity and Monitoring: How to Monitor the Infrastructure of an Anonymity System

The paper *Anonymity and Monitoring: How to Monitor the Infrastructure of an Anonymity System* was published in the IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Volume 40, Issue 5 in 2010³¹.

You can find the paper online at <http://dx.doi.org/10.1109/TSMCC.2010.2045372>. A preprint is available on the author's homepage at http://www.sba-research.org/wp-content/uploads/publications/IEEE_SMC_Tor_finalPreprint.pdf

³¹<http://www.ieeesmc.org/publications/>

Anonymity & Monitoring: How to Monitor the Infrastructure of an Anonymity System

Martin Mulazzani¹, Markus Huber², and Edgar Weippl³

¹ Security Research, Vienna, Austria mmulazzani@securityresearch.at

² Security Research, Vienna, Austria mhuber@securityresearch.at

³ Vienna University of Technology, Austria weippl@ifs.tuwien.ac.at

Abstract. The Tor network is a widely deployed anonymity system on the Internet used by thousands of users every day. A basic monitoring system has been designed and implemented to allow long term statistics, provide feedback to the interested user and to detect certain attacks on the network. The implementation has been added to TorStatus, a project to display the current state of the Tor network. During a period of six months this monitoring system collected data, where information and patterns have been extracted and analyzed. Interestingly, the Tor network is very stable with more than half of all the servers located in Germany and the US. The data also shows a sinusoidal pattern every 24 hours in the total number of servers.

1 Introduction

This work is about monitoring the infrastructure of the Tor network. Tor, “The Onion Router”, is an anonymity network that is widely used in the world to hide the user’s IP address [12]. The required infrastructure is run by volunteers. Anyone can host a Tor server to improve the overall performance of the Tor network. Because servers can join and leave the current set of servers anytime, the network is highly dynamic and the overall performance depends on the time of usage. However, the number of servers is surprisingly constant over time.

To provide some form of monitoring of this widely distributed network is difficult as it is an anonymity network. Especially since the user’s anonymity must not be compromised. We have implemented a form of monitoring to the infrastructure with a special focus on performance statistics, without introducing new or improving known attacks on the user’s anonymity.

This work is organized as follows: The second section discusses anonymity on the Internet and the Tor network. Section 3 presents the approach chosen for the monitoring and the selected values of interest. Section 4 explains briefly the implementation, while Section 5 interprets the data from the monitoring over a period of six months. It shows interesting regular patterns and properties of the Tor network as well as unusual patterns. The last section is dedicated to the conclusion.

2 Anonymity on the Internet

The word anonymity comes from the Greek language, meaning “without a name” or “namelessness”. When talking about anonymity, the usual setting refers to the transportation of a message from a sender to one or more recipients [29]. These messages can be of any kind (like electronic messages, votes in general elections, emails, etc.) and can be transported using any media (a piece of paper, radio waves, the Internet, natural language, etc.).

Anonymous electronic communication was first introduced in 1981 by David Chaum [5]. Instead of sending the message directly from the sender to the recipient it passes several additional relay servers on its way (so called mixes). These mixes collect the messages and forward them in batches. With the use of public key cryptography [10] it is possible that only the intended receiver can read the message and the mixes can not. Since the first introduction of mixes the development has evolved into two main directions: low latency and high latency anonymity systems [7].

High latency anonymity systems started in the early nineties and provided anonymity by mixing the messages over time. It is not important when exactly the message reaches the destination, but it has to reach it eventually with all possible identifying information removed. One of the first heavily used systems which started in 1992 was the pseudonymous remailer by Johan Helsingius (anon.penet.fi) [28]. By sending an email to this server including the intended target, it was possible to send *pseudonymous* emails and postings to the Usenet. However, in 1995 Helsingius was forced by the authorities to reveal a users real email address because of a complaint filed by the Church of Scientology. Thus the provided anonymity was weak, no encryption was used and the server was a single point of failure. If the server was down, the service was unusable. A deployed example for an anonymous remailer at the time of writing is Mixminion [8]. It was first published in December 2002 and is the successor of Mixmaster [21] and the Cypherpunk remailer [28]. It uses a set of trusted and synchronized directory servers which know the current network and server states. On the current Mixminion network there are between 80,000 and 100,000 messages transmitted daily [20]. The differences between the systems are that every succeeding remailer uses a stronger threat model and becomes more resistant against attacks like traffic analysis [6]. One of the most recent developments in this field is the publication of the Sphinx cryptographic message format [9].

Low latency anonymity systems on the other hand are not able to use time to achieve strong anonymity. They are built to be usable for interactive communication like SSH or web browsing, which means that long delays like in anonymous remailers are unacceptable. Bidirectional communication is used where a client can request something from another entity like a server and gets the response with only minor additional delay. Various systems exist with different threat models and different compromises between flexibility and anonymity. One of the

first systems designed was “ISDN-mixes” [30]. Today, Tor is probably the most heavily used with an estimated 100,000 plus users every day. An alternative to Tor is “JonDo”, formerly known as the Java Anon Proxy [4].

2.1 The Tor Network

Tor [12] is the second generation of onion routing and is based on the original onion router [15] from 1999. It is a low latency anonymity system and used all over the world. The public network was deployed in October 2003 [13] and has grown from only a few servers to more than 2000 distinct servers with around 1200 servers running any time at the time of writing. It is an open source project and offers a lot of features compared to earlier implementations of onion routing for users seeking anonymous communication.

The goal of the Tor network is to provide communication anonymity and to prevent traffic analysis [6], thus making it hard for an attacker to link communication partners or link multiple conversations to or from a single user. It was developed for a high degree of deployability and usability, resulting in an increased anonymity set [11]. Tor also makes it easy for anyone to extend the underlying infrastructure by running a Tor node at very low cost. Tor nodes run on a variety of platforms and the system is very flexible regarding what applications can be used, and incorporates many good designs from previous anonymity systems.

The threat model of Tor is weaker than other anonymity systems. The most commonly assumed threat regarding anonymity systems is a global passive adversary who can monitor all traffic going into and out of an anonymity systems as well as the traffic inside of the system. Additionally, it is often assumed that the adversary may has the ability to inject, delete or modify messages in transit and a subset of the nodes can be under this persons control; it is however unrealistic that he controls all of the nodes. Anonymity systems that protect against such attackers can assume to be secure for most of the possible users [12]. By design Tor does not protect against such a strong attacker. The assumed adversary in Tor is able to run or control a subset of network nodes, is able to observe some fraction of the network traffic and is able to inject, delete or modify messages [12].

Tor protects against traffic analysis attacks which would allow an attacker to find out which nodes of importance to attack only by watching traffic patterns. In favor of a simple and deployable design, some commonly used techniques in anonymity like message reordering were intentionally not used when Tor was designed. The network is not solely peer-to-peer based and is not using steganography; thus it is not hiding the fact that somebody is using Tor. Another design choice was to keep Tor completely separated from protocol normalization: many protocols like HTTP are very complex and variable, which would require complex protocol normalization to make all clients look the same. For HTTP, Tor relies on TorButton, a firefox extension to prevent IP and cookie leakage, and

Privoxy, a filtering web proxy that enhances privacy. TorButton on the other hand filters many types of active web content which could possibly leak identity. For other protocols this has to be considered separately before relying on the anonymity provided by Tor.

The Tor network consists of the following basic entities - the details of the basic architecture and the entities can be found in the original paper [12] as well as on the Tor website [1]:

- Directory Server: The core of the Tor network, a small set of trusted authorities (7 at the time of writing). They know the current set of valid Tor servers, distribute this knowledge to the clients and sign it with their private keys. The directory servers vote on the current set of Tor servers with simple majority, so that control over one directory server has no value. For an adversary control over more than half of these servers would be required to successfully attack Tor at the directory server level.
- Server: The Tor servers are used to actively hide the user's IP address. They are operated by volunteers, anyone can download the server software and run a Tor server. As soon as the server program is started it publishes its keys and descriptor to the directory servers, and will then become part of the directory. Once the server is listed in the directory it can be used by clients. It will continue to publish signed statements to the directory servers constantly. To be as flexible as possible, every server operator can decide to limit the bandwidth or transferred data Tor might consume, and whether the server will be an exit server and allow connections to hosts outside the Tor network. There are many other configurable options and many properties a server can have, for example if it is flagged as a guard node. Guard Nodes [34] were not part of the original design but were added later because of a certain attacks [25] against hidden services and predecessor attacks in general [37].
- Clients: The client software runs on the user's computer. Right now there are two software clients available, the original client written in C and available for the majority of popular operating systems (Windows, Mac OS X and various Linux/Unix flavors) and OnionCoffee, which was written in Java for the PRIME project [31] as a proof of concept and is no longer maintained. Additionally the client is available in different packages, like preconfigured USB sticks and VMware images.

Communication in Tor works on the transport layer, anonymizing TCP streams. Many TCP streams can share a circuit, and by default a circuit consists of three Tor servers: the *entrance node*, the *middleman node* and the *exit node*. The entrance node is aware of the real IP address of the users, but not the communication content whereas the exit node is able to see the content the user was originally transmitting and the final destination but not the originating IP address. The circuit is constructed incrementally by the client software. Everything is transported in fixed size cells, which get encrypted once for every relay in the circuit by the client. Packet anonymization instead of stream anonymization was proposed [17] and has the drawback of causing a lot of resend requests as the

packets take different paths with arbitrary delays to the destination.

As Tor became popular in recent years, many attacks were designed to defeat or degrade anonymity [3, 16, 22] and their according countermeasures [25]. Among the most notable attacks is the Sybil attack [14], which is applicable on almost any system that relies on distributed trust. It is based on the idea that a small number of entities can impersonate multiple identities. Traffic analysis [2] on the other hand uses metadata from network communications instead of content to attack anonymity systems, as content is often encrypted with strong ciphers. This metadata includes volume and timing information as well as source and destination of messages. Tor does not protect against a global passive adversary that can monitor all network connections. Traffic analysis makes it possible to link communication partners at random, just by controlling and monitoring some part of the network. This has been shown in [23] by estimating the traffic load on specific Tor nodes. Other publications wanted to improve the overall Tor performance and the security inside Tor [24, 26, 27, 32, 33].

TorStatus, also known as Tor Network Status, is one of many open source projects around the Tor Network [35]. Its goal is to present the current status of the Tor network to the interested user. In essence it consists of a Perl script which connects to a local Tor server and writes the necessary information into a database, the frontend is a PHP application which takes care of the appropriate presentation. It was originally developed by Joseph Kowalski in 2006. The current lead developer is Kasimir Gabert.

3 Monitoring Tor

Monitoring the Tor network has been done before, but with a different focus. Researchers tried to find out more information about Tor users and to understand what they use Tor for [19] in the beginning of 2008. This is somehow contradicting with the purpose of an anonymity service. Usage pattern on the other hand are important for future development of Tor and the perception of Tor in the public. The result was that most connections are used for HTTP traffic, while most of the traffic is caused by BitTorrent, a common file sharing protocol. For Tor hidden services, statistics have been collected as well [18], but without long term public presentation.

3.1 Values of Interest

Before monitoring a system it is important to identify the values of interest, i.e. not all values are bearing useful information. As with the Tor network it is important to find values that have a direct influence on the user's anonymity. Care has to be taken not to introduce possibilities for new attacks on the network, or to decreasing the degree of anonymity and making existing attacks easier. The Tor infrastructure is a dynamic number of Tor servers operated by volunteers,

so the most basic values of interest consists of the total number of Tor servers.

Monitoring the infrastructure of the Tor network has been done before but was suspended again. There is a website that collected the number of Tor nodes and the total traffic of the Tor network over time, [36], but stopped publishing those values as of August 2007. Instead, it refers to the current TorStatus websites, which only shows the current state of the network and does not monitor anything over time (yet). One of the design papers by the creators of Tor [13] published some statistics about the number of running routers and the total relayed traffic in the early stages of the Tor network, from August 2004 till January 2005. Apart from that no official statistics over time are publicly available.

Inspired by those early data collections the following important values worth monitoring have been identified:

- Total number of running servers
- Total number of running exit servers
- Total number of running fast servers
- Total number of running guard servers
- Country distribution of running servers
- Country distribution of running exit servers

These values have been chosen with two goals in mind: to measure the effects of the infrastructure on the possible degree of anonymity and to measure the effects on the quality of service.

For the overall anonymity and security, the number of Tor servers at the time of usage is crucial. A handful of servers would be easy to attack, either directly with the goal of defeating anonymity or with a denial of service attack. Thus the greater the number of Tor servers, the better. In the network consensus all servers are flagged according to their capabilities and qualities. The important flags for the degree of anonymity are “running” and “exit server”. Those are the servers a client uses for his circuits, as apparently a server that is not reachable is of no value in a Tor circuit. Thus the number of running servers and running exit servers is the smallest practical metric with influence on the path selection and anonymity. The client’s actual path selection algorithm is more complex and depends on further server attributes like advertised bandwidth and server IP address, which are not part of the monitoring. For the quality of the Tor connections, two other flags become important, namely the “fast” and “guard” nodes: the more fast Tor nodes there are, the better the overall performance for the client. The more guard nodes in the network, the better the performance (as the proportion guard nodes to clients decreases) and the number of clients a given guard handles decreases.

The distribution of servers over different countries and therefore different jurisdictions is the second value that influences the degree of anonymity. To keep the demands on the performance of the monitoring system low and to prevent

the user from an uncontrollable information flood, only selected countries are getting monitored. The countries with the highest number and the vast majority of Tor servers are those of interest. Over a period of two months the country distribution of Tor servers has been monitored, resulting in the implementation of the following set of countries to monitor: United States of America, Germany, China, France, Sweden, Russia, the Netherlands, Canada, Great Britain, Italy and Austria. Additionally the sum of all other countries and Tor nodes that are not mappable to any country get accumulated and stored as well. Based on this information, the top 11 countries have been selected for monitoring. The mapping of IP addresses to countries was done by using TorStatus, more precisely by using the GeoIP library together with TorStatus.

4 Implementation

The gathering of data was implemented by expanding the TorStatus update script, *tns_update.pl*, together with a graphical user presentation in the frontend, *network_history.php*. Whenever the script is updating the TorStatus database, it is also updating the Tor History monitoring. The values are stored in a RRD, a so called round robin database, which is part of the software “RRDtool” and was already used for generating and updating the other graphs of the TorStatus website. The changes can already be found in the SVN trunk of TorStatus, and will be included in the next release.

When working on an anonymity network it has to be assured that by adding features the additional features should never possibly harm the network or anonymity. This means that the new features should not introduce new attacks or make existing attacks easier. In our case, all the used information is part of the network consensus, which means that all the information are already known to every Tor server as well as every Tor client, they are kind of public knowledge. The only difference is that some coarse part of it is now stored over time. From all the recent published attacks on the Tor network none becomes easier. And even future attacks should not benefit from it, as e.g. the number of servers per country or the total number of exit servers holds no valuable information when trying to attack the user’s anonymity. The main objective is to provide the interested user an overview over the current status of the underlying Tor network and to make changes in the Tor network visible.

4.1 Further improvements

The collected data set might be expanded to collect information of other countries in detail, and collect additional data on the overall network. One useful example would be the total transmitted traffic (as in [13]), but as this information is provided by the servers it is easy to manipulate and to bias the statistics. Another feature worth implementing in the future would be the port distribution of the exit servers in general and the distribution of specific important ports like

HTTP by country. As soon as IPv6 is supported by Tor the monitoring software should be extended to support IPv6 as well.

The focus of this work was on basic network monitoring, so these improvements are left open for the future. However, any extensions should be done very carefully and only after evaluating the impact on anonymity.

5 Data Interpretation

The data collection started in the middle of October 2008 and is still going on. The time interval of the collected data presented here refers to around six months. Note that the time zone in the graphics is GMT and that gaps in the graphics were caused by failures in the data collection process.

5.1 Overall Network Size

Since deployment, the overall network size and the number of running Tor servers has been steadily increasing. Since public deployment in October 2003 with only a handful of servers, the infrastructure has grown to a total of 50 nodes by November 2004 [23]. This doubled to around one hundred nodes in January 2005 [13]. In August 2007 there were around 1000 running servers as of [36]. Up to today, everyday between 1300 and 1500 servers form the basic infrastructure of the Tor network, whereof 600 to 700 of them are exit servers.

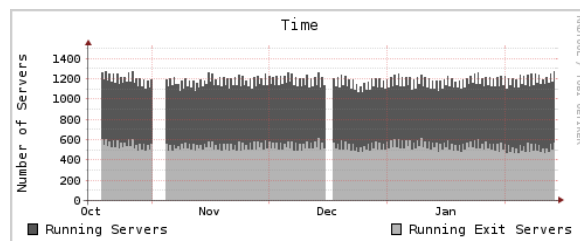


Fig. 1. Running Tor Servers in the first 3 months

Figure 1 shows the overall number of servers and the number of exit servers during the first three months of the data collection period. On average, 1163 servers were online during that period, 532 of them exit servers. The highest number of servers were 1289, the lowest 1040 (626/454 exit servers). It is clearly visible that the number of servers stays roughly the same and the network is quite constant. This does not mean that these are the same servers all the time: more than half of the servers that report their uptime have an uptime less than one week. Less than five percent have an uptime which is longer than ten weeks.

About eighty percent of the servers have a reported uptime in TorStatus. During the last 3 months the overall network size increased, which is shown in figure 2.

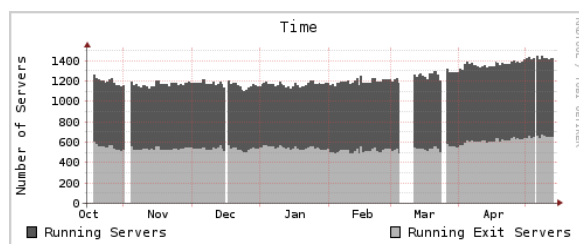


Fig. 2. Running Tor Servers during 6 months

On country level the most active countries are by far Germany and the United States. Both countries together host in total more than half of all the servers, which is shown in figure 3. The reasons for that are not easy to identify. Among

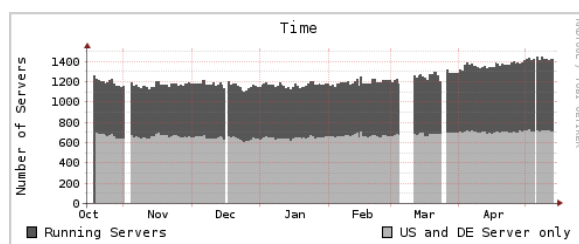


Fig. 3. Running Servers in DE and US, in total

other reasons, data privacy seems to have a high significance in Germany, probably based on the country's history and because of many popular organizations that want to enforce civil rights and electronic privacy like the Chaos Computer Club. In the US there is a different approach on data privacy compared with Germany or the European Union. Both countries are among the top five countries regarding the Internet density in the world according to internetworldstats.com. If the Internet density is high, Internet access and bandwidth for the users becomes cheap. This of course increases the likelihood that a user is willing to donate bandwidth to the Tor project by running a Tor node.

5.2 Total Number Patterns

The most obvious pattern is that the number of overall servers is dependent on the current time of day. This is shown in figure 4. It looks like a sine function (or

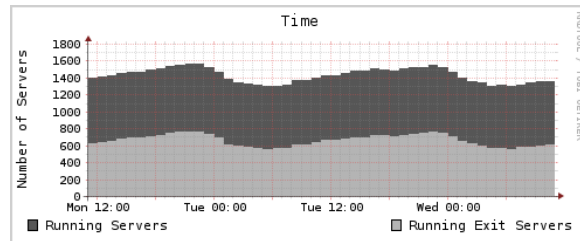


Fig. 4. Running Tor servers within 48 hours

cosine) as the number of servers reaches a maximum and a minimum within a 24 period each day. The time in the figure is GMT, which means that most of the servers are online in the “GMT evening” which is insofar interesting as this data represent the worldwide Tor network. This is mainly influenced (among other things) by the daily variation of servers in Germany, where most of the servers are online in the evening.

This daily pattern is also found in the data set of the “other” servers. These are the servers where the country is unknown, summed up with the servers that are in a different country than the countries monitored separately. The sum of the other servers and how it changes over time is shown in figure 5. By looking

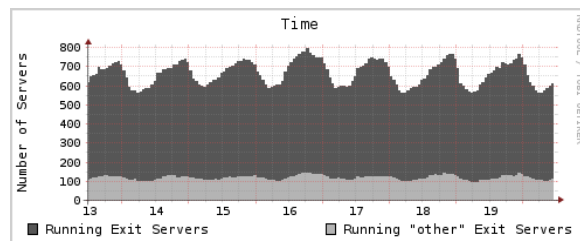


Fig. 5. Daily pattern in “other servers”

at it, it is visible that this category of servers as well has a daily pattern. The amplitude is much lower than compared with the number of exit servers, but it is still visible. This means that firstly Germany seems to be the reason that the number of servers changes over the day, but also the servers that are not in the top countries or where the country is not identifiable by GeoIP within TorStatus. Secondly, it seems that the majority of servers have to be geographically close if the server administrators operate their servers similar, which is needed that an overall pattern becomes visible.

Finally, regarding the overall network pattern, the proportion of exit servers is of importance. The Tor network is designed that 33 % of the servers need to be exit servers. Figure 6 shows that between 45 % and 50 % of the servers are exit

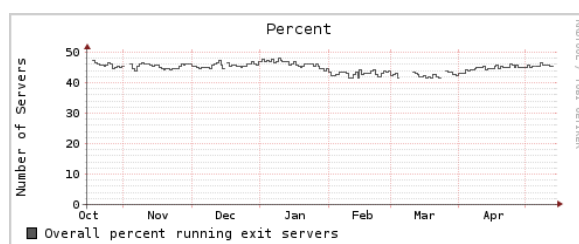


Fig. 6. Overall percent exit servers

servers, which is very good. Just because a server is flagged as an exit server does not mean that the client software has to choose this server as the exit server. Interestingly, the ratio of normal to exit servers stays quite constant compared to the rest of the Tor network. Sudden changes in the percentage of exit servers would be a good indicator in case of any attacks. For example if an attacker tries to redirect as many users as possible to his exit servers by attacking the others with a denial of service attack. However, a sophisticated attacker might combine the denial of service attack with a Sybil attack and start an exit server for every attacked exit server. This would keep the ratio unchanged. The percentage of exit servers does not take into account the various exit server policies as they allow services based on the port number, but gives a good overview of the overall network.

In our monitoring, around 80 % of the servers are monitored, which means that at least 80 % of all Tor servers are hosted in only 11 countries. In the future it would make sense to take the various exit policies into account as well.

5.3 Country Pattern

When watching the data on country level, other interesting patterns can be identified. The patterns discussed in this section refer to the top countries that are monitored separately. The biggest part of the Tor network is made by servers from Germany and the United States. By comparing those two countries, subtle differences are detectable. This is shown in figure 7. While the number of servers in the US stays quite constant, the number of servers in Germany is changing with a 24 hour pattern. Around 50 servers or 15 % of the German servers are turned off and on again during the night. This might be because of Internet access pricing models from Internet service providers or because the servers are operated at workplaces. The exact reasons are hard to tell. The total number of

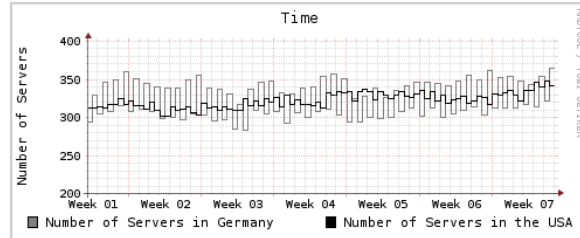


Fig. 7. Differences Germany and the USA

servers in Germany per day is between 300 and 400.

It is also interesting how many servers in a country are exit servers. This is shown for a few countries in figure 8. Only the top 5 countries have been

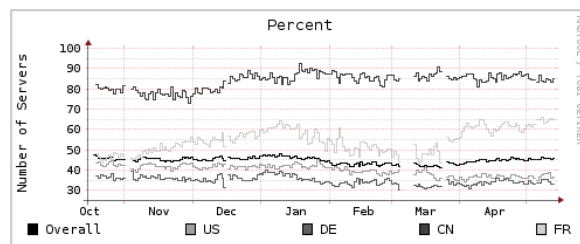


Fig. 8. Percent of Exit Servers

compared for simplicity of the graph. It is very interesting to see which countries have more than average in exit servers. The average of exit servers was already shown in figure 6. Germany and the United States have a less than average percentage of exit servers, while the countries hosting not so many Tor servers like China and France have a higher percentage. This is most likely because of the much lower number of servers. Overall, China has constantly between 80 and 90 % exit servers.

5.4 Strange Pattern found

During the collection of data, some strange patterns have been found. One of them seems to be inside of Tor itself: When looking at the number of guard servers, the values change quite a lot. It seems as if the Tor consensus is adding and removing a large proportion of guard servers. This is shown in figure 9. Although the number of guard servers seems to change a lot, there is still a daily visible pattern. The sine pattern again can be attributed to Germany, the country with the highest number of guard servers, and some other countries.

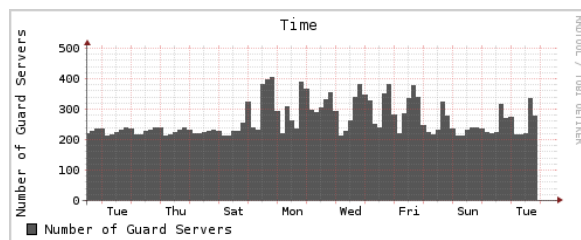


Fig. 9. Problem with number of Guard Servers

The large jumps with more than 100 guard nodes and more joining and leaving the set of guard nodes is not explicable by daily pattern. This is likely due to some topics in the Tor implementation, or the network consensus protocol. This issue has been reported to the Tor developer team and not yet solved.

6 Summary and Conclusion

Monitoring the Tor network greatly increases the overall benefits of the Tor community. First, the most obvious benefit, is that the development and growth of the Tor network is now visible to the Tor developers, the server operators, the users and everyone interested. The information is presented conveniently and self-explanatory, no deep knowledge or understanding of the Tor network is needed. This feedback can be used by the advanced user to decide about the current degree of anonymity and whether the network is providing sufficient anonymity or not. The collected statistics about the history of the Tor network might one day allow decisions as to which country is to be used as an entry to Tor or other improvements to the path selection. An example: all of the sudden the number of servers in Italy drops significantly. Is it then still ok to use the remaining servers in Italy? It probably is ok for a user relatively close to Italy who is seeking high data throughput, if the Italian servers are used as a middleman or entry node. For a user seeking high anonymity it might be better to avoid Italy in all circuits at all. Recently the complete network consensus information since the beginning of the Tor network has been made available, which will further improve the findings of this paper in the future. By comparing the present with the past it becomes possible to observe trends and this might be a triggering point for an analysis or discussion.

This brings us to a very important benefit, the possibility of attack detection. Right now, only the server operator is able to detect attacks on servers under his control, like denial of service attacks. Sometimes there follows a discussion on the Tor mailing list (or-talk@freehaven.net), mainly to check if other operators have or had similar problems. If an attacker would launch a more significant attack like a distributed denial of service attack on half of the Tor servers, the real impact on the Tor infrastructure would be hard to detect fast. The same holds if

an attacker launches a Sybil attack by adding numerous potentially manipulated servers. With the monitoring of the Tor network this becomes easier to detect, as the change in the number of servers is reflected immediately respectively as soon as the change becomes part of the consensus.

In the far future, the collected information could be used to compare the current state of the network with the expected state. By calculating a conservative set of rules about the “normal” behavior of the Tor network, changes that could probably have an influence on the security can be detected. Incorporating these rules into the path selection algorithm would allow finely tuned privacy constraints on the selected nodes. An example would be an adaptive path length: in the normal case, three nodes could get selected as it is currently implemented. If then all of the sudden the number of servers drops or significantly increases the path length could be automatically set to six Tor nodes to increase security. It is not clear if an increase in path length would increase the security, but it is sufficient as an example where the collected data of the Tor network could get used in the future. Another example would be the integration of the data into TorFlow, a project to detect malicious nodes. Instead of testing all the servers all the time, which could eventually be costly in terms of time and resources, only selected Tor nodes or all the nodes in a specific country could be tested. Based on the collected data a set of Tor nodes responsible for the most suspicious changes in the infrastructure could be identified for testing.

Acknowledgements

We would like to thank the anonymous reviewers. This work has been supported by the Austrian Research Promotion Agency under grant 820854.

References

1. Tor: anonymity online. <https://www.torproject.org/>.
2. Adam Back, Ulf Möller, and Anton Stiglic. Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems. In *Proceedings of Information Hiding Workshop (IH 2001)*, pages 245–257, April 2001.
3. Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-Resource Routing Attacks Against Tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007)*, October 2007.
4. Oliver Berthold and Hannes Federrath. Project “anonymity and unobservability in the internet“. *Workshop on Freedom and Privacy by Design / CFP2000*, 2000.
5. David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 4(2), February 1981.
6. George Danezis. Introducing Traffic Analysis: Attacks, Defences and Public Policy Issues. Technical report, University of Cambridge Computer Lab, 2005.
7. George Danezis and Claudia Diaz. A Survey of Anonymous Communication Channels. Technical report, Microsoft Research, January 2008.

8. George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15, May 2003.
9. George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. *Security and Privacy, IEEE Symposium on*, 2009.
10. Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
11. Roger Dingledine and Nick Mathewson. Anonymity Loves Company: Usability and the Network Effect. In *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, June 2006.
12. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
13. Roger Dingledine, Nick Mathewson, and Paul Syverson. Challenges in deploying low-latency anonymity (DRAFT), February 2005. <https://www.torproject.org/svn/trunk/doc/design-paper/challenges.pdf>.
14. John R. Douceur. The Sybil Attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, 2002.
15. David Goldschlag, Michael Reed, and Paul Syverson. Onion Routing for Anonymous and Private Internet Connections. *Communications of the ACM*, 42:39–41, 1999.
16. Andrew Hintz. Fingerprinting Websites Using Traffic Analysis. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)*, April 2002.
17. Olaf Landsiedel, Alexis Pimenidis, Klaus Wehrle, Heiko Niedermayer, and Georg Carle. Dynamic Multipath Onion Routing in Anonymous Peer-To-Peer Overlay Networks. In *Proceedings of the GLOBECOM 2007*, 2007.
18. Karsten Loesing, Werner Sandmann, Christian Wilms, and Guido Wirtz. Performance Measurements and Statistics of Tor Hidden Services. In *Proceedings of the 2008 International Symposium on Applications and the Internet (SAINT)*, July 2008.
19. Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas C. Sicker. Shining Light in Dark Places: Understanding the Tor Network. In *Privacy Enhancing Technologies*, pages 63–76, 2008.
20. Mixminion network load. <http://www.noreply.org/load/>.
21. Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2. Draft, July 2003. <http://www.abditum.com/mixmaster-spec.txt>.
22. Steven J. Murdoch. Hot or Not: Revealing Hidden Services by their Clock Skew. In *Proceedings of CCS 2006*, October 2006.
23. Steven J. Murdoch and George Danezis. Low-Cost Traffic Analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, May 2005.
24. Steven J. Murdoch and Robert N. M. Watson. Metrics for Security and Performance in Low-Latency Anonymity Networks. In *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, July 2008.
25. Lasse Overlier and Paul Syverson. Locating Hidden Servers. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 100–114. IEEE Computer Society, 2006.
26. Lasse Overlier and Paul Syverson. Improving Efficiency and Simplicity of Tor Circuit Establishment and Hidden Services. In *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, June 2007.

27. Andriy Panchenko, Lexi Pimenidis, and Johannes Renner. Performance Analysis of Anonymous Communication Channels Provided by Tor. In *ARES '08: Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*, pages 221–228. IEEE Computer Society, 2008.
28. Sameer Parekh. Prospects for Remailers. *First Monday*, 1(2), August 1996. <http://www.firstmonday.dk/issues/issue2/remailers/>.
29. Andreas Pfitzmann and Marit Hansen. Anonymity, Unobservability, and Pseudonymity: A Consolidated Proposal for Terminology. Draft, February 2008. http://dud.inf.tu-dresden.de/Anon_Terminology.shtml.
30. Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. ISDN-mixes: Untraceable communication with very small bandwidth overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems*, pages 451–463, February 1991.
31. PRIME - Privacy and Identity Management for Europe. <https://www.prime-project.eu>.
32. Stephen Rollyson. Improving Tor Onion Routing Client Latency. Technical report, Georgia Tech College of Computing, 2006.
33. Robin Snader and Nikita Borisov. A Tune-up for Tor: Improving Security and Performance in the Tor Network. In *Proceedings of the Network and Distributed Security Symposium - NDSS '08*, February 2008.
34. TOR Path Specification. <https://svn.torproject.org/svn/tor/trunk/doc/spec/path-spec.txt>.
35. TOR Network Status Project. <http://project.torstatus.kgprog.com/>.
36. TOR Running Routers, outdated as of August 2007. <http://www.noreply.org/tor-running-routers>.
37. Matthew Wright, Micah Adler, Brian Levine, and Clay Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Trans. Inf. Syst. Secur.*, 7(4), 2004.

Tor HTTP Usage and Information Leakage

The paper *Tor HTTP Usage and Information Leakage* was published at the 11th Joint IFIP TC6 and TC11 Conference on Communications and Multimedia Security (CMS) 2010 in Linz³².

You can find the paper online at http://dx.doi.org/10.1007/978-3-642-13241-4_22. A preprint is available on the author's homepage at <http://www.sba-research.org/wp-content/uploads/publications/2010-Huber-TorHTTPUsage.pdf>.

³²<http://www.cms2010.net/>

Tor HTTP usage and Information Leakage

Markus Huber¹ and Martin Mulazzani² and Edgar Weippl³

¹ Vienna University of Technology, Austria mhuber@sba-research.org

² Secure Business Austria, Austria mmulazzani@sba-research.org

³ Vienna University of Technology, Austria weippl@ifs.tuwien.ac.at

Abstract. This paper analyzes the web browsing behaviour of Tor users. By collecting HTTP requests we show which websites are of interest to Tor users and we determined an upper bound on how vulnerable Tor users are to sophisticated de-anonymization attacks: up to 78 % of the Tor users *do not* use Tor as suggested by the Tor community, namely to browse the web with TorButton. They could thus fall victim to de-anonymization attacks by merely browsing the web. Around 1% of the requests could be used by an adversary for exploit piggybacking on vulnerable file formats. Another 7 % of all requests were generated by social networking sites which leak plenty of sensitive and identifying information. Due to the design of HTTP and Tor, we argue that HTTPS is currently the only effective countermeasure against de-anonymization and information leakage for HTTP over Tor.

Keywords: Tor, information leakage, privacy

1 Introduction

The Tor network [1] is a widely deployed anonymization network which hides the user's IP address on the Internet. It is expected to be used by hundreds of thousands of users every day and is the most heavily used open anonymization network today [2]. The main contribution of this paper is an in-depth analysis on how the Tor network is used to browse the web.

At the time of writing little is known about the traffic that leaves the Tor anonymization network. McCoy et al. [3] published a first investigation into how Tor is being used by its end-users. They found that the majority of connections leaving the Tor network are caused by the hypertext transfer protocol (HTTP). According to statistics from the Tor project [4], their anonymization network played a crucial role in the aftermaths of the latest Iranian elections whereas HTTP-based services such as Facebook had been blocked on a national-level. The ability of Tor to bypass Internet-censorship techniques got recently even the attention of mainstream media (e.g. Forbes [5]). A deeper analysis of HTTP traffic that is tunnelled through the Tor network is however lacking. Hence we aim to provide a first investigation into how Tor is used to access the world wide web.

By running a Tor exit server and logging the HTTP requests, we collected in total 9×10^6 HTTP requests over a period of several weeks. The captured HTTP requests form the basis of our investigation into the web browsing behaviour of Tor users. The main contributions of this paper are:

- An in-depth analysis of HTTP traffic tunnelled through the Tor network (Section 4).
- HTTP-based attack scenarios and mitigation strategies (Section 5).
- Potential risks for complete HTTP user de-anonymization, which can not be prevented by Tor (Section 5.1).

We additionally show dangers to which users are exposed by using Tor incorrectly and which information users leak by browsing the web with Tor.

The rest of the paper is organized as follows: Section 2 discusses Tor and previous work about Tor usage analysis. Section 3 shows how we collected our data while we interpret the results in Section 4. Section 5 describes new attack scenarios based on HTTP as well as countermeasures. The last section is dedicated to the conclusion.

2 Tor Security and Threat Model

The Tor network hides the user's IP address by sending its packets through a number of Tor servers. Tor itself does not hide nor encrypt communication content leaving the Tor network: the user has to take care that it is used correctly. Sensitive information should only be sent over an encrypted protocol such as HTTP secure (HTTPS). A passive adversary running an exit server would need to break the end-to-end encrypted transmission in order to capture sensitive information. We will show later to what extent sensitive information is transmitted unencrypted over HTTP.

The basic infrastructure of Tor is run by volunteers, and anyone can set up a relay at relatively low cost. It provides reliable, bi-directional communication that can be used for low latency communication such as interactive conversations, shell access or web browsing. Tor can be used by any program that is able to use a SOCKS proxy and is freely available for almost any operating system as well as in the form of prepared live CDs and virtual machines. Tor uses three servers per path by default to hide its users real IP address, all servers chosen at the client-side. The user's communication content is protected from eavesdropping within the Tor network by using multiple layers of encryption. Before forwarding a message, every Tor server removes his layer of encryption.

At the time of writing, the Tor network consists of approximately 1600 running servers, distributed all over the world. The first server in a path is chosen from an ordered set of so called "entry nodes"; these servers are able to see the users real IP address. Without the ordered set, every entry node would eventually

see every user's real IP address. The last Tor server in the path is the so called "exit server" and is chosen based on the communication target's port number and self-proclaimed available bandwidth. The so called "exit policy" at every server specifies which port numbers are allowed for communication and whether the server is allowing connections only within Tor or leaving Tor to the regular Internet. The exit policy is defined by the server operator. Finally, a small set of trusted "directory servers" collect information about the current state of the network and vote on the current set of reachable servers. This information is publicly available to all clients. The security of the Tor network relies on the fact that instead of a single point or entity a user has to trust (for example by using an open proxy server or a dubious VPN service); the trust is distributed among the three Tor relays in a path.

Previous research about the usage of Tor has been conducted in the beginning of 2008 [3]: by running a high bandwidth Tor relay and inspecting the communication content it was found that the majority of connections leaving Tor was created by HTTP traffic, in total more than 90 %. However, a disproportional part of the transferred data was caused by BitTorrent, a common file sharing protocol. Yet a detailed analysis of the HTTP usage has not been conducted. Another analysis has been conducted by Dan Egerstad in 2007 [6] who published a list of 100 sensitive email accounts including passwords from embassies that apparently used Tor incorrectly. Other published attacks on Tor aimed at decreasing or defeating the users anonymity by means of traffic analysis [7,8,9,10] as well as attacks on unique aspects such as path selection [11] or the "hidden services" of the Tor network [12,13].

3 Tor HTTP Sniffing - Ethics and Methodology

In our experiment we ran a Tor exit node and collected all HTTP requests by running *urlsnarf* from the *dsniff* toolkit [14]. *Urlsnarf* sniffs HTTP traffic and is able to format it in CLF (Common Log Format), which is a format commonly used by webservers. Compared to other experiments [3] our server was advertising less bandwidth to represent an average node and not to bias the client's Tor path selection algorithm towards our node; only HTTP traffic was allowed in our exit policy. The collection period was from December 2009 till January 2010 with 9×10^6 HTTP requests in total resulting in a logfile of 2.5 gigabytes. We took special care that the users identities were not endangered or revealed during the collection process: we did not store any packet dumps, user credentials, authentication cookies or any other possibly compromising data except the HTTP request. We did not try to become a "guard server" which clients use as their entry to the Tor network and which are able to see the users real IP address, and we did not monitor incoming connections. The data was stored in an encrypted filecontainer and moved to another computer after the collection process to protect against data disclosure in case of a server compromise.

A Python script using the “apachelog” library [15] deconstructed the requests into three parts, according to our evaluation criteria:

- *Target domain*: the domain name of the request, without any deeper analysis of the specific requested file or session context.
- *User agent*: the string that the browser sends to identify itself. This gives a hint if the users are using TorButton, the recommended software by the Tor developers to prevent user identification by an adversary.
- *File type*: the extension of the requested file. Executables pose a direct danger to the user as an adversary might replace or modify them to defeat anonymity or even worse. Indirect danger comes from file formats where there exist known vulnerabilities in the corresponding software.

Subsequently the requests were sanitized and normalized for evaluation:

- *File formats*: various file extensions are treated the same on the client side, notably image file formats such as `jpg` and `jpeg` or websites within the browser, such as `html`, `htm`, `php` or `cgi`.
- *Domain affiliation*: some websites use different domains for content distribution, such as for example `fbcdn.net` and `facebook.com` belong to the same website.

4 Results & Data Interpretation

Our goal was to analyze what Tor is used for, which domains are most popular among Tor users and to discover potential threats to users.

4.1 Domains

In a first analysis based on the collected HTTP traffic we looked into depth into the different websites that were visited through our Tor exit server.

Table 1a shows the top 10 visited domains, based on the percentage a certain domain accounts to the total requests. Facebook.com and google.com were amongst the most visited sites. In fact, the majority of the requests belonged to one of the following website categories:

- *Social networking* sites (e.g. facebook.com, blackplanet.com)
- *Search engines* (e.g. google.com, yellowpages.ca)
- *File sharing* (e.g. btmon.com, torrentbox.com)

Social networking sites (SNSs), which today account to the most popular web sites, account in total to 7.33 per cent of all analysed HTTP traffic. These web-services are interesting for two reasons: SNSs leak plenty of personal information and secondly these services do not support HTTPS at the time of writing, except for user authentication. Table 1b shows the Top SNSs as well as how much of the SNSs traffic accounts to which service.

Domain (total %)		Social Networking Site (relative %)	
URL	Per cent (%)	Name	Per cent (%)
'facebook.com'	4.33	'facebook.com'	59.06
'www.google.com'	2.79	'blackplanet.com'	21.94
'blackplanet.com'	1.61	'vkontakte.ru'	5.61
'yandex.ru'	1.57	'tagged.com'	4.95
'btmon.com'	1.47	'orkut.com'	3.13
'photobucket.com'	0.98	'myspace.com'	2.36
'craigslist.org'	0.90	'mixi.jp'	1.54
'torrentbox.com'	0.88	'hi5.com'	0.48
'ameba.jp'	0.87	'adultfriendfinder.com'	0.47
'maps.google.com'	0.70	'badoo.com'	0.46

(a) Overall services

(b) Social Networking Sites

Table 1: Analysis of most accessed domains

4.2 Fileformats

Among all the collected HTTP GET requests, `.html` was predominant with almost 54 % of all the requests. Around 32 % were caused by image formats, followed by JavaScript with 4.25 % of all GET requests. The details of the top 10 requested file extensions are shown in table 2.

Fileformat		
Extension	Description	Per cent (%)
'html'	HyperText Markup Language	53.83
'jpg'	JPEG image	18.15
'gif'	Graphics Interchange Format (GIF)	11.43
'js'	JavaScript	4.25
'css'	Cascading Style Sheets (CSS)	3.03
'png'	Portable Network Graphics	2.81
'xml'	Extensible Markup Language	2.62
'ico'	ICO image	0.77
'swf'	Shockwave Flash file	0.48
'rar'	RAR archive file format	0.20

Table 2: Top 10 file formats

4.3 Web browser types

Web browsers submit a text string known as “user agent string” to servers with details on the client version and the operating system they are running on. When

looking at the browser user agent string, various browser and operating systems combinations were seen. The browser used to access websites through the Tor network plays a crucial role for the anonymity of the end-user. TorButton [16] is a plugin for the Mozilla Firefox browser developed by Mike Perry and Scott Squires, which makes it possible for the user to switch between using the Tor network and browsing the web directly. Even more important, it disables many types of active content which could be used to bypass Tor and thus defeat Tor's anonymity protecting methods. To begin with, we inspected which were the ten most common user agent strings. Table 3 shows the top 10 browser user agent strings within our experimental data.

Full User Agent String	Per cent (%)
'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.7) Gecko/2009021910 Firefox/3.0.7'	18.86
'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.7) Gecko/2009021910 Firefox/3.0.7'	4.48
'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9) Gecko/2008052906 Firefox/3.0'	2.71
'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.16) Gecko/20080702 Firefox/2.0.0.16'	1.81
'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)'	1.66
'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)'	1.64
'Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9) Gecko/2008052906 Firefox/3.0'	1.59
'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)'	1.50
'Mozilla/5.0'	1.34
'Opera/9.63 (Windows NT 5.1; U; en) Presto/2.1.1'	1.31

Table 3: Top10 Browser (Raw user agent string)

TorButton uses a more constant user agent string in contrast to Firefox since the Firefox user agent string changes with every update of Firefox. This increases the anonymity set, as long as the used string is plausible and the used version is still in widespread use. By an analysis of the TorButton source code we identified nine distinct user agent strings that have been used by the different versions of TorButton. We found that solely 22 % of all the user agent strings matched one of the nine user agent strings used by the TorButton Firefox extension. Hence we argue that at least 78 % per cent of all traffic originated from a browser without the TorButton extension. It remains unclear if the users take additional preventive measure when using other browsers, however it seems unlikely that the majority of the non-TorButton users enforced all required browserside anonymity protection measures. Table 4 shows the Top 10 user agent strings, whether the user agent strings were used by TorButton and the revision of the TorButton source code. Furthermore the table shows the user agent strings and operating systems versions in a more readable format.

5 Further Tor Exit Server Attack Scenarios

A malicious exit server has many different options to gain knowledge of the Tor users, active as well as passive. Many HTTP requests leak information of

Browser		
Version	OS	Per cent (%)
TorButton > svn:r18954 (Firefox/3.0.7)	-	18.86
-	-	4.48
Firefox/3.0 (Gecko/2008052906), en-US	Windows XP	2.71
TorButton > svn:r16066 (Firefox/2.0.0.16)	-	1.81
Internet Explorer 6.0 SV 1.0	Windows XP	1.66
Internet Explorer 6.0	Windows XP	1.64
Firefox/3.0 (Gecko/2008052906)	Windows Vista	1.59
Internet Explorer 6.0	Windows 2000	1.50
Mozilla/5.0	-	1.34
Opera 9.63 (Presto/2.1.1), en	Windows XP	1.31

Table 4: Top 10 Browser (interpretation)

various kinds. Additional information can be gained by active content insertions and modifications.

5.1 Information leakage of sensitive information

Many HTTP requests leak information, sometimes even sensitive information. Even if the malicious exit server is unable to identify the originator of the request, it is able to gain knowledge only by watching the requests passively.

- *Search query strings*: Search queries are often submitted to a search engine via a HTTP GET request. If a Tor user searches information about e.g. a special disease, location information about a hotel, how to get to a certain place, or a recent political event, this gives hints about the identity, location or nationality of the user. Additional information can be deduced by language, browser and operating system to further reduce the anonymity set. This theoretical risk has also become evident in practice by the incident with AOL’s release of search queries [17].
- *Social networks*: As described above social networking sites accounted for more than 7 per cent of the all the HTTP traffic captured by our Tor exit server. In the case of Facebook as well as other popular SNSs, HTTP requests include the user-id in plaintext. Because users often represent their realworld persona, SNSs users can easily be identified. The social graph of a user could furthermore reveal the identity of a SNS user. Thus an analysis of a user’s id and corresponding social graph could completely deanonymize the Tor user. This is especially dangerous as many Tor users apparently use social networks.
- *Localization information* sent by a misconfigured browser reduces the anonymity set considerably. TorButton normalizes all user to use “en-us” within the browser user agent. This increases the size of the anonymity set for users

from the US, however decreases the size for the rest of the world. Other localization information can be retrieved from toplevel domains as we suspect that many users e.g. browse to their localized version of the Google search engine (google.ca, google.dk, ...) instead of the normalized google.com.

- *Other* sensitive and possibly incriminating content will be transmitted without proper usage of Tor, e.g. authentication cookies. However, as these are not transmitted in the HTTP GET request, we did not include them in our analysis. A motivated attacker surely will harvest those information as well, thereby possibly increasing the chance of defeating anonymity of a certain Tor users.

As an example we will use a search query on google maps, asking for driving instructions from Times Square to Central Park in New York. As it can be seen, plenty of parameters are part of the request, and of course the exact address coordinates as well:

```
http://maps.google.com/maps?f=d&source=s_d&saddr=times+square ->
&daddr=central+park&hl=en&geocode=&mra=ls ->
&sll=40.771133,-73.974187&sspn=0.053106,0.111494 ->
&g=central+park&ie=UTF8&t=h&z=16
```

5.2 Script injection

Dynamic website content such as e.g. AJAX or Flash is hard to filter: either all dynamic content is blocked which results in poor usability, or dynamic content is allowed which opens the door for exit servers injecting malicious scripts. It has already been shown that the insertion of invisible iframes and Flash can defeat anonymity [18]. By injecting JavaScript within an invisible iframe it was possible to further reduce the requirements on the client to become vulnerable to this attack [19]. The authors even showed that their attack is feasibly by modifying the HTML meta refresh tag of a website, so without JavaScript.

By manipulating HTTP communication content, phishing attacks become possible if no communication encryption or verification is used. Instead of sending the data to the destination, a malicious exit server might save the data and present an error page.

5.3 File replacement

There exist many file formats commonly used on the Internet which could be used by an adversary to execute malicious code as the corresponding software handling the files has well known weaknesses. Among all the HTTP requests we have seen, up to 97993 requests, or 1% of the total requests were for files with known vulnerabilities and publicly available exploits. As we did not inspect the communication content and the transferred files itself, it remains unclear if the files could have been used for infecting the client or not. Instead, it can be seen as an upper bound for a new possible and yet not documented infection vector.

- executable (5590 requests): `.exe` files could get replaced or malicious code could be appended to the executable. This could result in remote code execution on the client side and could among other things be used to reveal the users identity.
- PDF (1619 requests): vulnerabilities in Adobe Reader and Acrobat as well as alternative PDF viewers could be used to execute malicious code in `.pdf` files.
- Office (400 requests): Microsoft Office has many documented vulnerabilities. The requests we monitored were for `.doc`, `.xls` and `.ppt` files.
- Mediafiles (23821 requests): Several Mediaplayer like e.g. Apples Quicktime, the Windows Media Player or VLC have documented flaws that are exploitable by manipulated media files. We encountered various requests that could be used for exploit piggybacking: `.avi` (10105 requests), `.wmv` (6616 requests), `.mp3` (4939 requests) or `.mpg` (1138 requests).
- other file formats (66563 requests): compressed file archives like `.zip` (9937 requests) and `.rar` (16512 requests) might get used by an attacker as they can be used to exploit vulnerabilities in the software itself; but also to add, manipulate or replace files within those archives. Shockwave flash files `.swf` (40114 requests) account for a major proportion of vulnerable file formats and could be used for client infection as well.

These vulnerabilities could be used to massively compromise anonymity on a large scale by exploiting vulnerable software, using Tor for cheap “man in the middle” attacks and even creating a botnet of Tor clients. However, it has to be noted that this is not a flaw of Tor but of rather an inappropriate use of it. Clients should verify the integrity of their files when using unencrypted communication.

5.4 Countermeasures and discussion

It is hard if not even impossible to prevent information leakage in HTTP requests, as the requests are often transmitting information of a certain user’s context. The collection and aggregation of webservice specific information is non-trivial, but we showed that a great amount of information can be gathered by a passive adversary. In the following we briefly outline three methods that can help to mitigate security and privacy risks caused by a malicious Tor exit server:

Detection of malicious exit servers The most straightforward solution would be to detect bad exit servers and ban them from the Tor exit server list. McCoy et al. [3] proposed to use reverse DNS lookups in order to detect exit servers that run packet analyzer software with a host resolution feature. A complete passively adversary is almost undetectable. TorFlow is an application developed by Mike Perry [20] which supports exit server integrity checks. Hereby the basic principle is that: a reference file is used as a sample and downloaded through different Tor exit servers, cryptographic checksums are used afterwards

to check if a manipulation occurred. The basic idea works fine on files like binaries or static websites, dynamic content however is much harder to verify. For this reason, dynamic content is blocked by TorButton instead of analyzed for maliciousness.

User education The Tor network offers reliable anonymity in case if properly used. Awareness-campaigns as well as end-user education can help to ensure that people use Tor always in combination with TorButton as well as take special care of which services they use through Tor. The incident with the embassy mail accounts [6] has shown what might happen if Tor is used incorrectly, even seriously harming privacy instead of preventing de-anonymization and obfuscating user activity. Active content (e.g. Macromedia Flash) of all kind should be avoided if possible, and using trusted exit nodes could further reduce the risk of data leakage.

HTTPS The only solid protection of user data would be the use of strong encryption such as secure HTTP (HTTPS). The usage of the HTTPS protocol is unfortunately not always possible as many website operators do not support it, e.g. <https://www.google.com> redirects the user to <http://www.google.com>. At the time of writing the great majority of social networking providers fail to support HTTPS.

6 Summary and Conclusion

By collecting 9×10^6 HTTP requests we observed that up to 78 % of the Tor users browse the Internet without TorButton, the recommended software by the Tor community. The majority of users is therefore possibly vulnerable to sophisticated deanonymization attacks by an exit server, e.g. by injecting invisible iframes or scripts. 1 % of the requests were for vulnerable file formats, which could be used for exploit piggybacking. Even if the adversary running an exit server is completely passive, without modifying or redirecting communication content, an uncountable amount of sensitive information like search queries or authentication cookies are leaked. Furthermore, 7 % of all analysed HTTP connections were created by social networking services which leak plenty of personal information. To protect the Tor users, various tools like TorButton or TorFlow have been proposed and implemented. However, the only effective countermeasure at the time of writing is the use of end-to-end cryptography, namely HTTPS.

Future research in this area can be done to quantify the amount of sensitive information observable by a passive adversary. It has to take into account the different requirements for anonymization and/or censorship circumvention by the users. Additional research is needed on protection measures.

References

1. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *13th USENIX Security Symposium*, San Diego, CA, USA, August 2004.
2. Tor: anonymity online. <https://www.torproject.org/>.
3. Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining Light in Dark Places: Understanding the Tor Network. In *Proceedings of the 8th Privacy Enhancing Technologies Symposium (PETS 2008)*, Leuven, Belgium, July 2008.
4. Tor Project. Measuring Tor and Iran, 2009. <https://blog.torproject.org/blog/measuring-tor-and-iran>.
5. Forbes. PluggedIn: Web tools help protect human rights activists, 2009. <http://www.forbes.com/feeds/afx/2009/08/19/afx6794971.html>.
6. Dan Egerstad. DERanged gives you 100 passwords to Governments & Embassies. online, 2007. [Retrieved 2009-12-20].
7. J.F. Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. *Lecture Notes in Computer Science*, pages 10–29, 2001.
8. N. Mathewson and R. Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. *Lecture Notes in Computer Science*, 3424:17–34, 2005.
9. N. Hopper, E.Y. Vasserman, and E. Chan-Tin. How much anonymity does network latency leak? In *Proceedings of the 14th ACM conference on Computer and communications security*, page 91. ACM, 2007.
10. Steven J. Murdoch and George Danezis. Low-Cost Traffic Analysis of Tor. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 183–195, Washington, DC, USA, 2005. IEEE Computer Society.
11. Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against Tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007)*, Washington, DC, USA, October 2007.
12. S.J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of the 13th ACM conference on Computer and communications security*, page 36. ACM, 2006.
13. Lasse Øverlier and Paul Syverson. Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE CS, May 2006.
14. dsniff. <http://www.monkey.org/~dugsong/dsniff/>.
15. apachelog. <http://code.google.com/p/apachelog/>.
16. M. Perry and S. Squires. Torbutton. <https://www.torproject.org/torbutton/>.
17. M. Barbaro and T. Zeller. A face is exposed for AOL searcher no. 4417749. *New York Times*, 9:2008, 2006.
18. Andrew Christensen. Practical Onion Hacking: finding the real address of Tor clients, 2006. http://www.fortconsult.net/images/pdf/Practical_Onion_Hacking.pdf.
19. T.G. Abbott, K.J. Lai, M.R. Lieberman, and E.C. Price. Browser-Based Attacks on Tor. *Lecture Notes in Computer Science*, 4776:184, 2007.
20. Mike Perry. Torflow: Tor network analysis. In *HotPETS 2009: 9th Privacy Enhancing Technologies Symposium*, page 14, Aug 2009.

Quantifying Windows File Slack Size and Stability

The paper *Quantifying Windows File Slack Size and Stability* was published at the Ninth Annual IFIP WG 11.9 International Conference on Digital Forensics 2013 in Orlando, Florida³³.

You can find the paper online at http://dx.doi.org/10.1007/978-3-642-41148-9_13. A preprint is available on the author's homepage at http://www.sba-research.org/wp-content/uploads/publications/ifipSlack_2013_preprint.pdf

³³<http://www.ifip119.org/Conferences/>

Quantifying Windows File Slack in Size and Stability*

Martin Mulazzani, Sebastian Neuner, Peter Kieseberg,
Markus Huber, Sebastian Schrittwieser, Edgar Weippl
SBA Research, Vienna
[1stletterfirstname][lastname]@sba-research.org

Abstract

In digital forensics, different forms of slack space can be used to hide information from either the operating system or other users, or both. While some forms are easily detectable others are very subtle, and require an experienced forensic investigator to discover the hidden information. The exact amount of information that can be hidden varies with the form of slack space used, as well as environmental parameters like file system block size or partition alignment. While some methods for slack space can be used to hide arbitrary amounts of information, file slack has tighter constraints and was thought to be rather limited in space.

In this paper we evaluate how much file slack space modern operating systems offer by default and how stable it is over time with special regards to system updates. In particular we measure the file slack for 18 different versions of Microsoft Windows using NTFS. We show that many files of the operating systems are rather static and do not change much on disk during updates, and are thus highly suitable for hiding information. We furthermore introduce a model for investigators to estimate the total amount of data that can be hidden in file slack for file systems of arbitrary size.

Keywords: Digital Forensics, Slack Space

*This is the author's preprint for the Ninth Annual IFIP WG 11.9 International Conference on Digital Forensics. The original publication is available at www.springerlink.com

1 Introduction

With ever increasing hard drive and storage capacities, slack space is now more than ever an interesting topic in digital forensics. Hard drives with 3 terabytes and more are commodity hardware now, which leaves plenty of space to hide information and making manual forensic analysis time-consuming and cumbersome. While encryption can be used and is used to make certain or all information on a hard drive inaccessible during forensic investigations [5] (e.g., Truecrypt ¹, dm-crypt, FileVault or Bitlocker [13]), slack space and steganography hide information more or less in plain sight.

While special areas for hiding data like the *Host Protected Area* (HPA) and *Device Configuration Overlay* (DCO) on the hard drive can be easily detected with current tools e.g., *The Sleuth Kit (TSK)* [4] or *Encase* [3], it is much harder for an investigator to find data that was hidden in file slack space on purpose. During the natural life-cycle of files on hard drives, file slack is constantly created and overwritten, thus not necessarily stable. Multiple tools have been published that allow to hide information in file slack, even encrypted. This paper is the first work towards quantifying file slack space of common operating systems, and how slack space is affected by system updates. We focus our work on different versions of Microsoft Windows, as it is in widespread use on clients and servers. We used Windows versions from the last 10 years that are still supported by Microsoft and receive operating system updates, starting with Windows XP and Server 2003. We also included the most recent versions at the time of writing, Windows 8 and Windows Server 2012 RC. We furthermore propose a model that allows an investigator to estimate the amount of file slack capacity across all files on a hard drive, thus including files created by the user. In particular, the contributions of this paper are as follows:

- We quantify file slack space in a multitude of Microsoft operating systems.
- We evaluate the durability of OS-specific file slack space regarding system updates.
- We present a simple model to estimate the total possible file slack for investigators.

¹<http://www.truecrypt.org/>

The rest of the paper is organized as follows: Section 2 briefly explains different forms of slack space and their background in file systems, and why file slack is particularly interesting for forensic investigations. Our method for measuring file slack space is presented in Section 3. Section 4 evaluates how much information can be hidden in commodity operating systems, and how stable OS-specific file slack is over time. Results are discussed in Section 5, as well as our model for estimating file slack space. Related work is presented in Section 6 before we conclude in Section 7.

2 Background

File slack is defined as the space between the end of the file, and the end of the allocated cluster [4]. Slack space is a byproduct of operating systems: to reduce addressing overhead they cluster multiple sectors of a hard drive together, and implicitly create file slack space for every file that does not align in size with the cluster size. The size of usable slack space is in general depending on the cluster size of the filesystem, the sector size of the underlying hard drive, and the padding strategy used by the operating system. Hard drives used to have a sector size of 512 bytes, with modern hard drives having a sector size of 4k. FAT32 usually has a cluster size between 4k and 32k (depending on the actual partition size), while NTFS usually has a cluster size of 4k for drives < 16 TB [14]. Windows uses padding only for the last sector at the end of each file, while the other sectors in the cluster are left untouched [4], possibly leaving up to $n - 1$ sectors untouched if the portion of the file in the last cluster (with n sectors) is smaller than the sector size.

Numerous other forms of slack space (like volume slack or partition slack) or places to hide data on a hard drive (e.g., HPA and DCO) can be encountered in a forensic investigation [2, 10, 6]. The benefit of file slack, however, is that it exists in any case on every hard drive, and can be extended in size by storing a large number of files and/or using large cluster sizes (NTFS for example supports a cluster size of up to 64 KB). File slack is also the reason why bitwise copies have to be created during image acquisition [11], as otherwise the content of file slack would be lost for the investigation.

For a forensic investigation the slack space is of interest in two cases: if the suspect used a secure deletion software like *shred* or *WipeFile* for wiping

files, slack space can contain fragments of previous data. In the second case, the suspect might have hidden data in slack space on purpose, using freely available tools. Several tools have been developed to facilitate easy storage of information in slack space, among them *slacker.exe* for NTFS from the Metasploit anti-forensic toolkit, or *bmap* for Linux.

3 Quantifying OS File Slack Space

To analyze file slack in NTFS for different operating systems we used *fiwalk* [7] by Simson Garfinkel which is now part of *SleuthKit*, and which is based on the DFXML language for digital forensics [8]. For our evaluation we used 18 different versions of Microsoft Windows, as it is still by far the most predominantly operating system in use. We evaluated client and server versions alike, and used different service pack levels as starting point to measure slack file persistency. Each system used the NTFS default cluster size of 4KB with underlying 512 bytes sector size and was installed using the default settings. We then patched each installed system with the available system updates and available service packs, including optional updates like the .NET framework and additional language packs, and ran *fiwalk* on the acquired disk images. The full list of operating systems and the total number of security updates and service packs that have been installed during our evaluation can be seen in Table 1.

Our scripts parsed the XML outputs from *fiwalk*², and calculated the slack space for each disk image. During calculating the usable file slack capacity we omitted NTFS file system metadata objects like the \$MFT or \$BitClus, and files that are less than a full disc cluster in size to avoid files that are possibly resident in the NTFS master file table (\$MFT) and thus not directly usable for file slack. For calculating file slack persistency we checked the SHA-1 hash values as well as the file's timestamp for last write access before and after the updates. Again, we only included files that use more than one disc cluster in size.

²Note to the reviewers: we will release our datasets and scripts online, and will include the link here once the paper is accepted for publication

Operating System	Upd.	SPs	Operating System	Upd.	SPs
Windows XP Pro.	+189	+2	Windows 7 Pro. SP1	+106	—
Windows XP Pro. SP2	+164	+1	Windows 7 Ent.	+212	+1
Windows XP Pro. SP3	+177	—	Windows 7 Ent. SP1	+167	—
Vista Business	+246	+2	Windows 8 RC	+11	—
Vista Business SP1	+72	+1	Server 2003 R2 Std. SP2	+163	—
Vista Business SP2	+143	—	Server 2003 R2 Ent. SP2	+167	—
Vista Ent. SP1	+207	+1	Server 2008 R2 Std.	+148	+1
Vista Ent. SP2	+143	—	Server 2008 R2 Std. SP1	+103	—
Windows 7 Pro.	+156	+1	Server 2012 RC	+6	—

Table 1: List of evaluated operating systems and their updates

4 Evaluation

For our evaluation we first quantify file slack in Windows before we analyze file slack stability during system updates.

4.1 Quantification of OS Slack Space

The amount of available file slack space depends on the number of files and thus on the complexity of the underlying operating system. The base installation of Windows XP (the oldest system in our evaluation) can be used to hide about 22 megabytes of data in file slack on average, the newest versions of Windows at the time of writing (i.e., Windows 8 and Server 2012 RC) can be used to hide 86 and 70 megabytes respectively. Windows Vista in its different versions (Business vs. Enterprise) allows approximately 62 megabytes in file slack, and Windows 7 (Professional and Enterprise) a little more than 69 megabytes. A similar trend is observable for Windows server: Windows Server 2003 R2 has approximately 20 megabytes in file slack capacity, while Server 2008 R2 has about 73 megabytes.

The amount of file slack increases tremendously with system updates, especially with service packs, as the old system files are usually kept referenced in the file system in case something goes wrong during the update process. Table 1 shows which version of Windows has received a service pack since its release, and the number of system updates that have been installed for our evaluation. Many system updates affect only a small number of files, while service packs are major updates. An increase in file slack capacity of more than 100% during the lifetime of an operating system is not uncommon, on

average the slack space doubles in total. At the end of the evaluation process Windows XP had more than 30 megabytes on average, Vista 105 megabytes and Windows 7 Professional 100 megabytes on average. Windows 7 Enterprise showed an exceptional increase of more than 500%, from around 72 megabytes to more than 400 megabytes. Windows 8 and Server 2012 RC were stable, as there are not many updates available yet. The detailed results can be seen in Table 4.2.

The details of the cumulative slack space are visualized in Figure 1: we use a standard box-plot over all collected states in the evaluation process, grouped by the product lines. This graph can be used by forensic examiners in assessing the amount of possible file slack for a given operating system. Depending on the install date of the underlying operating system, one hundred to two hundred megabytes of file slack space is possible for newer operating systems, even with the default cluster size of 4k in NTFS. We will discuss the amount of file slack for larger cluster sizes in Section 5. Note that the number of samples (respectively update steps) for the different product lines was as follows: Windows XP had 15, Vista had 32, Windows 7 had 19, 8 RC had 4. For the Windows Server: Server 2003 R2 had 10, Server 2008 R2 had 9 and Server 2012 RC had 4.

4.2 Stability of OS Slack Space

To assess the stability of file slack in the files of the operating system we compared SHA-1 hash values as well as the timestamps for each file in the images. While the number of files increased heavily (especially due to service packs), the stability of the initial file slack was high: in some cases more than 90% of the slack areas were not modified, and the amount of stable file slack was 50 megabytes and more. While the older versions of Windows i.e., XP and Server 2003 had 20 megabytes and less that persisted all system updates, Vista, Windows 7 and Server 2008 had 50 megabytes and more. On average and across all different Windows versions, 44 megabytes and 78% of initial file slack were still available at the end of the evaluation process. This means that up to two third of the file slack is still available even today e.g., for Windows XP SP2, which was already released in 2004. For Server 2003 R2 we measured that up to 80% would be still available today. Vista SP2 from 2009 as well as Windows 7 SP1 from 2011 kept more than 90% of their files

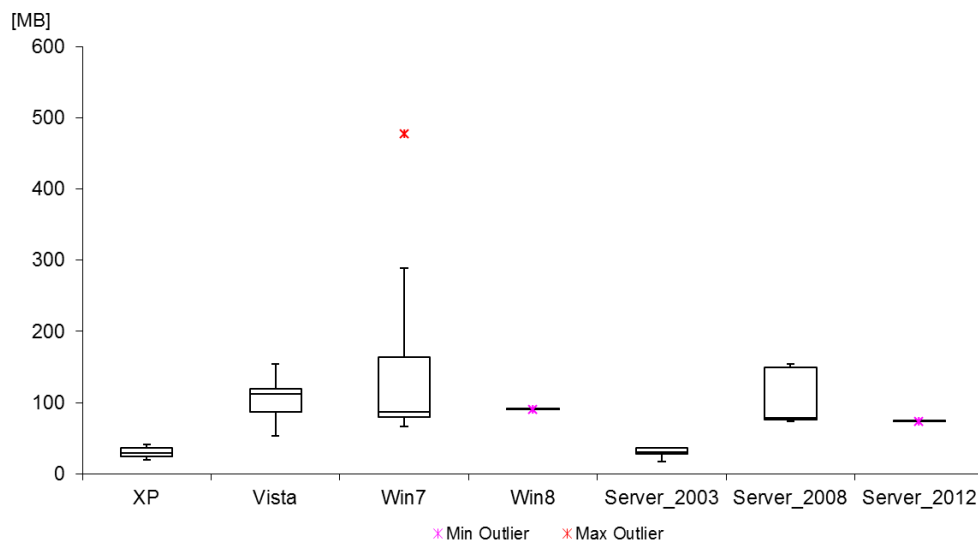


Figure 1: Box plot of quantified file slack space

intact.

File slack stability for a selection of different Windows versions and the corresponding number of update steps is shown in Figure 2. Again, the details of our data can be found in Table 4.2. Please note that even though the total file slack capacity increases with the system updates and update steps, the stability of file slack from the initial installation is represented as a monotonically decreasing function as the update can change existing files, but not add files to the previous.

5 Discussion

All tested operating systems rely on a large number of files for storage: while the earliest versions of Windows XP had only a little more than 10.000 files that are bigger than the default cluster size of 4k, Windows 7 has already more than 40.000 and Windows 8 more than 55.000. The number of files increases heavily with service packs: Windows Vista, which started with 35.000

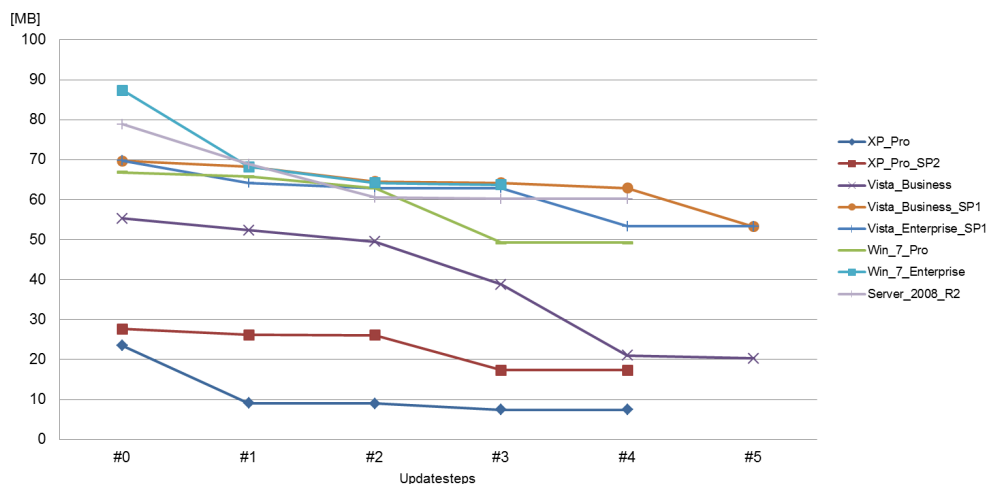


Figure 2: Stability of file slack across updates

files had more than 90.000 after installing two service packs. For forensic investigators this is particularly interesting as on one hand, the amount of information that could possibly be extracted from these files can be large. On the other hand these vast number of files offer a lot of slack space that can be used with readily available tools to hide information.

An adversary that actively uses file slack could furthermore increase the cluster size of NTFS - up to 64kb are supported by NTFS. A quick estimation by running our scripts on a xml file chosen at random showed that increasing the cluster size to 32kb on the same Windows Vista with more than 90.000 files could increase file slack to 1.25 gigabytes. This is 8.11 times larger compared to the 150 megabytes in our results with the default cluster size of 4kb, just using the files from the operating system itself. Forensic investigators should thus pay attention to artificially large cluster sizes, as they can be instrumented to hide possibly very large amount of data in the slack space.

Operating System	Initial Slack	Final Slack	Stability
Windows XP Pro.	22.36 MB	36.97 MB (165%)	7.09 MB/31.7%
Windows XP Pro. SP2	26.31 MB	29.49 MB (112%)	16.49 MB/62.7%
Windows XP Pro. SP3	18.72 MB	23.15 MB (124%)	14.21 MB/75.9%
Vista Business	52.70 MB	147.13 MB (279%)	19.34 MB/36.7%
Vista Business SP1	66.49 MB	119.89 MB (180%)	50.77 MB/76.4%
Vista Business SP2	50.89 MB	82.99 MB (163%)	48.13 MB/94.7%
Vista Ent. SP1	66.51 MB	140.35 MB (211%)	50.82 MB/76.4%
Vista Ent. SP2	71.73 MB	113.76 MB (159%)	67.36 MB/93.9%
Windows 7 Pro.	63.71 MB	115.16 MB (181%)	46.96 MB/73.7%
Windows 7 Pro. SP1	65.03 MB	77.80 MB (120%)	60.73 MB/93.4%
Windows 7 Ent.	83.33 MB	454.62 MB (546%)	60.74 MB/72.9%
Windows 7 Ent. SP1	65.10 MB	381.56 MB (586%)	60.77 MB/93.3%
Windows 8 RC	86.40 MB	87.06 MB (101%)	65.10 MB/75.3%
Server 2003 R2 Std. SP2	24.42 MB	33.90 MB (140%)	20.13 MB/82.4%
Server 2003 R2 Ent. SP2	16.55 MB	35.13 MB (212%)	15.20 MB/91.8%
Server 2008 R2 Std.	75.16 MB	146.80 MB (195%)	57.43 MB/76.4%
Server 2008 R2 Std. SP1	69.82 MB	73.03 MB (105%)	69.19 MB/99.1%
Server 2012 RC	70.16 MB	70.58 MB (101%)	70.01 MB/99.8%

Table 2: Detailed results of our file slack quantification

5.1 A Model for File Slack Space Estimation

Following we will give an estimation on the expected size of the slack space. Let n be the number of files and k be the cluster size (i.e. the number of sectors inside a cluster). Furthermore, s denotes the number of bytes inside a sector. Since only the last cluster of a file may not be filled completely (i.e. all clusters of a file except for one do not provide slack space), the slack space that can be expected from one file is completely independent from the actual size of the file. Thus we arrive at the following expectation S_E for the average slack space of a file: $S_E = s \cdot \mathbb{E}(X)$, where $\mathbb{E}(X)$ is the expectancy with respect to the underlying statistical distribution of the fill-grade of the last cluster.

In general it can be expected that the fill-grade of the last cluster is equally distributed among all possible values. Since the first sector inside a cluster is always filled in the case of the NTFS, the number of free sectors

that can be used lies in $\{1, \dots, k - 1\}$, thus yielding

$$S_{(n,s,k)} = n \cdot s \cdot \mathbb{E}(X) = n \cdot s \cdot \sum_{x=1}^{k-1} x \cdot \frac{1}{k} = n \cdot s \cdot \frac{1}{k} \cdot \frac{(k-1)k}{2} = n \cdot s \cdot \frac{k-1}{2}$$

Using a typical sector size of 512 bytes (s) and 8 sectors per cluster (k), we can reduce this formula to $S_n = 1792 \cdot n$.

5.2 Limitations

Our approach for measuring file slack has some limitations: for one, we deliberately ignored the fact that modern hard drives have a default sector size of 4 kb. This was necessary to be able to include operating systems still in widespread use (like Windows XP) that are yet completely unaware of the fact that hard drives can have larger sectors [15]. We also ignored user activity and did not simulate it, as we came to the conclusion that there is no sufficient method to simulate user activity in a realistic fashion. We also ignored additional files that may be found on a PC, either by the user or by software installed by the user. In particular we did not install any software that might be commonly found on Windows PCs or additional services for the Windows Servers to mimic real world systems, in particular Microsoft Office, alternative browsers like Chrome or Firefox, or services like IIS or a SQL server. Thus our results cannot be considered to be set in stone, but should be considered a conservative upper bound for OS-specific file system slack, and a conservative lower bound when including user data.

6 Related Work

In recent work regarding slack space, file fragmentation was used to hide data in FAT partitions [12]. FragFS on the other hand used slack space in \$MFT entries to hide data, as \$MFT entries have a fixed length of usually 1024 bytes per entry. In their estimation, a total of about 36 megabytes of data could be hidden in \$MFT slack [17] in systems running Windows XP, as most \$MFT entries use less than 450 bytes on average. Slack space in cloud environments has been recently discussed, as it can be used to retrieve possibly sensitive data like deleted SSH keys [1] or to hide data without leaving traces at the client or in the log files of the service operator [16].

Slack space was also found to be not securely deleted on the second hand market of hard drives [9].

6.1 Future Work

For future work we want to address the limitations discussed above, and plan to conduct a large-scale survey to measure slack space in real-world systems. So far we have not evaluated user files and how stable they are over time. We also plan to extend our evaluations to other modern operating systems and file systems that use clustering of sectors, in particular HFS+ on OS X and ext4 on Linux.

7 Conclusion

In this paper we have evaluated the amount and stability of file slack in NTFS-based operating systems. We especially considered system updates, and to what extent the file slack persists and evaluated it with 18 different versions of Microsoft Windows. All operating systems that we tested offered initial slack space in the tens of megabytes, which was largely immune to system updates: on average 44 megabytes, respectively 78%, of initial file slack was available after installing all available system updates, even full service packs. Especially with newer versions of Windows like Windows 7 or Server 2008, between 100 and 200 megabytes were available with the default cluster size of 4 kb.

Acknowledgements

This research was funded by the Austrian Research Promotion Agency under COMET K1, as well as Grant No. 825747.

References

- [1] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, and S. Loureiro. A security analysis of amazon’s elastic compute cloud service. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 1427–1434. ACM, 2012.

- [2] H. Berghel. Hiding data, forensics, and anti-forensics. *Commun. ACM*, 50(4):15–20, Apr. 2007.
- [3] S. Bunting and W. Wei. *EnCase Computer Forensics: The Official EnCE: EnCaseCertified Examiner Study Guide*. Sybex, 2006.
- [4] B. Carrier. *File system forensic analysis*. Addison-Wesley Professional, 2005.
- [5] E. Casey and G. Stellatos. The impact of full disk encryption on digital forensics. *ACM SIGOPS Operating Systems Review*, 42(3):93–98, 2008.
- [6] S. Garfinkel. Anti-forensics: Techniques, detection and countermeasures. In *The 2nd International Conference on i-Warfare and Security (ICIW)*, pages 77–84, 2007.
- [7] S. Garfinkel. Automating disk forensic processing with sleuthkit, xml and python. In *Systematic Approaches to Digital Forensic Engineering, 2009. SADFE'09. Fourth International IEEE Workshop on*, pages 73–84. IEEE, 2009.
- [8] S. Garfinkel. Digital forensics XML and the DFXML toolset. *Digital Investigation*, 2011.
- [9] S. Garfinkel and A. Shelat. Remembrance of data passed: A study of disk sanitization practices. *Security & Privacy, IEEE*, 1(1):17–27, 2003.
- [10] E. Huebner, D. Bem, and C. Wee. Data hiding in the ntfs file system. *Digital Investigation*, 3(4):211–226, 2006.
- [11] K. Kent, S. Chevalier, T. Grance, and H. Dang. Guide to integrating forensic techniques into incident response. *NIST Special Publication*, pages 800–86, 2006.
- [12] H. Khan, M. Javed, S. Khayam, and F. Mirza. Designing a cluster-based covert channel to evade disk investigation and forensics. *Computers & Security*, 30(1):35–49, 2011.
- [13] J. D. Kornblum. Implementing bitlocker drive encryption for forensic analysis. *Digital Investigation*, 5(3):75–84, March 2009.

- [14] Microsoft. Default cluster size for NTFS, FAT, and exFAT. Online at <http://support.microsoft.com/kb/140365>, retrieved Sept. 2012.
- [15] Microsoft. Information about Microsoft support policy for large-sector drives in Windows. Online at <http://support.microsoft.com/kb/2510009>, retrieved Okt. 2012.
- [16] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl. Dark clouds on the horizon: Using cloud storage as attack vector and online slack space. In *USENIX Security*, volume 8, 2011.
- [17] I. Thompson and M. Monroe. Fragfs: An advanced data hiding technique. *BlackHat Federal, January*.^[online] <http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Thompson/BH-Fed-06-Thompson-up.pdf>, 2006.

Towards Fully Automated Digital Alibis with Social Interaction

The paper *Towards Fully Automated Digital Alibis with Social Interaction* was published at the Tenth Annual IFIP WG 11.9 International Conference on Digital Forensics 2014 in Vienna, Austria³⁴.

A preprint is available on the author's homepage at http://www.sba-research.org/wp-content/uploads/publications/alibigenerator_preprint.pdf.

³⁴<http://www.ifip119.org/Conferences/>

Towards Fully Automated Digital Alibis with Social Interaction

Stefanie Beyer*, Martin Mulazzani*

Sebastian Schrittwieser§, Markus Huber*, Edgar Weippl*

* SBA Research, Vienna

[1stletterfirstname][lastname]@sba-research.org

§ University of Applied Sciences St. Pölten

sebastian.schrittwieser@fhstp.ac.at

Abstract

Digital traces found on local hard drives and in online activity have become the most important data source for the reconstruction of events in digital forensics. As such, digital alibis have already been used in court decisions and during investigations. In this paper we want to show that forged alibis can be created, which include online activity and social interactions. We are the first to use social interactions in digital alibis, and implemented a proof of concept that can be automated to create false digital alibis. Our framework simulates user activity, is fully automated and able to communicate using email as well as instant messaging using a chatbot. We evaluate our framework by extracting forensic artifacts and comparing them with results of real users in a user study.

Keywords: Digital Forensics, Automated Alibis

1 Introduction

Digital forensics techniques are nowadays applied to more and more criminal investigations due to the ever increasing prevalence of computers, smartphones and the involvement of modern technology in crimes. Traces like MAC-timestamps and OS-specific log files left on hard drives and information transmitted over network connections often are combined to produce

a holistic reconstruction of events and for specific times of interest [3, 16]. Digital alibis as such are commonly used in reliable expert witness opinions to charge and discharge suspects in court, as well as during the investigation itself.

In this paper we present a framework that can fully simulate user interaction and implements the automated creation of digital alibis with special focus on online social interactions like writing emails and chatting with friends. Social interactions have been so far neglected in previous work. The framework is highly configurable, and we are planning to release it under an open source license¹. Our framework is able to counter hard drive and network forensics as it is conducted today. We evaluate our framework by comparing it with the usage of real world users, and show that digital forensic analysis methods are not reliable if they are specifically targeted. The goal of this paper is to raise awareness that digital alibis can be forged. We want to question the reliability of digital alibis, intend to show that digital alibis can be forged.

The rest of the paper is organized as follows: Section 2 gives an overview of the relevant research areas of digital forensics for this paper, as well as related work. Section 3 presents our framework which was implemented as proof-of-concept to show the feasibility to forge digital alibis. In Section 4, we compare and evaluate our framework compared with real-world users. We discuss the results in Section 5, and conclude in Section 6.

2 Background

Several cases can be found where evidence for digital alibis played an important role. In one case, Rodney Bradford was charged of armed robbery but was released because of digital evidence that confirms that he was performing activities on his Facebook account during the time of the crime [17]. This digital evidence was later called an "unbeatable alibi" [8] by his attorney. In another case, a suspected murderer was acquitted because of digital evidence [9, 7]. During the time of the crime, working activity was found on the suspect's laptop.

¹Note to the reviewers: we will include the link here once the paper is accepted for publication

Usually a forensic analyst is confronted with multiple hard drives that have been imaged using hardware write blockers [2], and is asked specific questions about user actions that have or have not been conducted on the corresponding computers [4]. Modern communication technologies like online social networks [15] or smartphones [14, 13] can additionally increase the broad spectrum of digital traces. In the future, due to ever increasing storage capacity of consumer devices and overall case complexity, automated analysis will be crucial [12] to extract information of interest [11] in a reasonable amount of time.

2.1 Related Work

In the literature, several concepts for the analysis as well as the automatic construction of digital alibis can be found [10, 6]. However, existing alibi generators often use proprietary languages like AutoIt (Windows) or Applescript (OS X) and thus are specific to the underlying operating system that they run on e.g., Android [1], OS X [7], or Windows [10, 6]. Our framework does not rely on any os-specific components or software, and can thus run (with minor adaptations) on any Desktop operating system that runs Python. We implemented our prototype for Linux as there weren't any related frameworks available on Linux so far. We furthermore compared numerous configuration parameters with real world users, making our approach statistically harder to detect (with randomized values) and the persistently stored evidence more realistic compared to related work.

The previously described approaches try to hide the program, for instance with the help of harmless file names or on a separate storage device. Instead of using a file wiper for post processing to remove suspicious traces, our framework is constructed in such a way that there are no obvious traces left behind (despite the framework itself). For a forensic analyst, it should be not decidable if the artifacts on disc originate from the framework or a human user as we instrument (among other things) keyboard signals and mouse click events.

3 Alibi Generator Framework

The goal of our framework can be put simply: to simulate user activity as realistic and thorough as possible. To such extend, we want to simulate standard user activity: browsing the web, chatting with instant messaging software, writing emails and creating & editing documents of various kind. The concrete actions run by the framework should not be scripted or predictable, but randomized yet still realistic and convincing for a forensic investigator. Different word lists and online sources like Google Trends are used as inputs to capture a snapshot of current online interests with the need to incorporating specific user preferences at the same time. Many factors of computer usage are highly user dependent, and for the alibi framework to be as realistic as possible, factors like common online social interaction partners for chatting and email, language of communication as well as time delays between actions and usual concurrent actions need to be configured beforehand. Social interaction in particular is vulnerable to traffic analysis, as not only the content of the messages is of interest, but who is communicating with whom. The response time is also dependent on the length of the messages, which needs to be considered when simulating social interactions.

3.1 Implementation

Our proof-of-concept was implemented on Ubuntu 12.04 LTS using Python. The core features of the framework were chosen similar to the approaches in [7, 6]. In essence our implementation is split into three main components: the *scheduler*, the *program manager* and the *social interaction component*. Once the framework is started, the *scheduler* is in charge of overall management, it controls startup and triggers shutdown of all programs involved. It has the purpose of deciding which actions to take, either local or online, and when. The *program manager* runs and manages all applications, including the browser, the email- and chat software. The *social interaction component* consists of a chatbot for instant messaging and the email manager, responsible for email communications.

The framework can start and use local applications by sending key strokes and mouse clicks. Our framework comes pre-configured to use and handle the following applications in a meaningful manner: Firefox, gedit, LibreOffice, Thunderbird, Skype, and VLC. For automation the Python libraries *xau-*

*tomation*², *skype4py*³ and the chatbot implementation *pyAIML*⁴ are used. Furthermore, we use *splinter*⁵ for the automation of Firefox, which is based on *Selenium*⁶. Thus our implementation can browse the web, send and receive emails, chat in Skype, open and edit documents (LibreOffice and gedit) and start programs like music or video players. Figure 1 shows the main features of our proof-of-concept. Related features which are not yet implemented are marked in grey. Frequency and content of alibi events were derived from the analysis of several typical office workstations at our university.

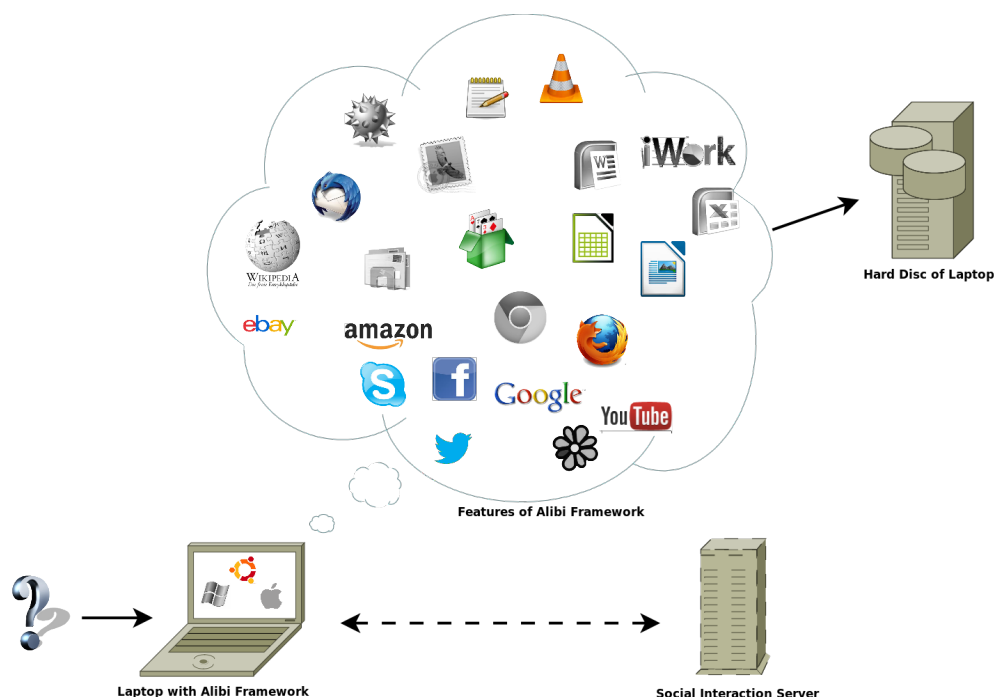


Figure 1: Conceptual Alibi Framework

More specifically, our framework queries Google and follows suggested links, tweets on Twitter and logs into Facebook, can search for Youtube

²<http://hoopajoo.net/projects/xautomation.html>

³<http://sourceforge.net/projects/skype4py>

⁴<http://pyaiml.sourceforge.net>

⁵<http://splinter.cobrateam.info/>

⁶<http://www.seleniumhq.org>

videos and browses websites with random mouse clicks and following links. New emails are drafted, received emails are forwarded, and answers to emails are sent with a reasonable delay. Additionally it is possible to mark new emails as read and delete emails. The action to be performed is chosen at random, not every email that is read will be replied to. The subject and content of emails can be predefined and stored in lists. Regarding instant messaging answering to incoming messages of buddies is supported, with the help of a chatbot. Reasonable time delays depending on the size of the response or on a random delay are implemented. Chat templates can be easily adapted due to the use of AIML [18]. If the timer of the scheduler expires, the chatbot says goodbye to his buddies and shuts down. Editing of local documents is implemented either by deleting a random amount of content, inserting predefined texts which fits the content of document at a random position. We implemented the use of LibreOffice by simulating key strokes and mouse clicks, as there are no Python bindings for LibreOffice available.

However, one post-processing step is necessary: Splinter has to use a separate firefox profile, and can not work on the user's profile directly. On startup splinter copies the user's profile to /tmp, and we overwrite the user's profile on shutdown by moving it back. Depending on the user's threat model, additional steps might be appropriate.

4 Evaluation

To evaluate our implementation we compare the framework with real world users. Since the particular usage is highly dependent on the person using it, this evaluation is intended to show that the default configuration is reasonable. We asked nine volunteers to use a virtual machine for 30 minutes just like they usually use their computer, and compared it with a single run of our framework. They were asked to browse the web, send emails, chat, edit documents or do anything they usually would do in their own environment. Forensic analysis was applied afterwards on the image of the virtual machine using *Sleuth Kit* and *Autopsy*, to extract data in a forensic manner. For that, the timestamps of the entire file system are extracted, as well as files that contain user data of interest are manually inspected. Manual analysis was conducted on the browser history *places.sqlite* of Firefox, the local mailbox files by Thunderbird, and the chat history *main.db* of Skype

to extract timestamps, message content, conversation partner and the time interval between messages, emails and visited websites. Network forensics would have been an alternative approach for evaluation, by inspecting the network traffic. However, hard drive analysis allows to extract unencrypted as well as additional information like local timestamps, and has thus been the evaluation method of choice. We then used different metrics to compare our framework with real world users during the half hour period e.g., the number of visited websites, duration of visit on websites and the number of chat messages sent and received.

Figure 2 shows the exemplary timeline of 20 minutes framework run-time. The social interaction can be clearly observed, in total 12 messages are sent from the social interaction component to various recipients, either as responses in ongoing conversations or as new messages to initialise conversations. The browser is directed to different websites and follows links to simulate further browsing (not shown in the figure). This includes news sites like nytimes.com and prominent news stories on the front page, youtube.com and videos from the "most popular" section, as well as the top videos from random search queries. Local timestamps as well as history logs are written to disk in all cases. Furthermore VLC is started and a local video from the hard drive is opened, which is also reflected in the timestamps on disk.

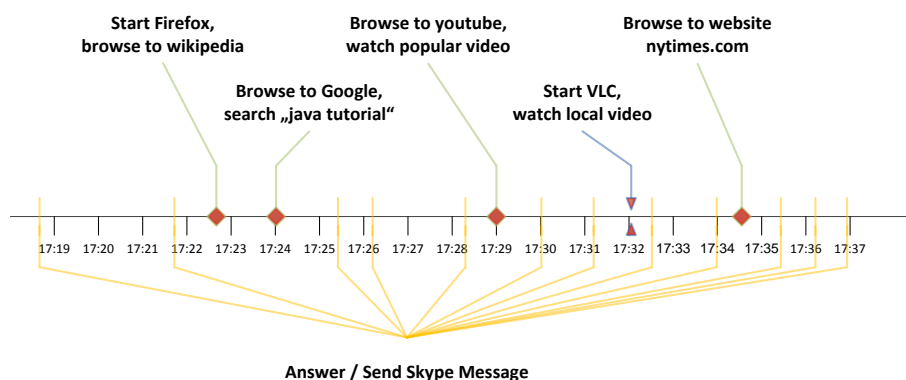


Figure 2: Exemplary Timeline of 20 Minutes Framework Activity

4.1 User Survey

Regarding the browsing behavior of the user group, target websites and time patterns have been extracted. The most popular websites have been google.com, facebook.com as well as some Austrian news sites. The framework on the other hand used a preconfigured list of websites as well as randomized google queries. The five most visited websites however matched in 4 out of 5 cases between the framework and the users. Some time patterns that were extracted can be seen in Table 1. The test users did not receive any emails due to our setup, but were asked to send some. On average, one email was sent per user. The maximum number of sent emails by the users was 3, the minimum 0. The majority of test persons did not write an email. The words in an email vary in number between 6 and 51 words, the average number of words is 21. The time between sending emails varies between 1 minute and 15 minutes.

<i>Browsing parameters</i>	<i>Framework</i>	<i>Test Persons</i>
# Visited Websites	11	min: 1 max: 12 avg: 9
Time on Website (min)	8 sec	5 sec
Time on Website (max)	2 min 16 sec	13 min 05 sec
Time on Website (avg)	1 min 52 sec	2 min 50 sec

Table 1: Comparison of browsing patterns

Nearly all test persons did chat during their session. There are between 7 and 46 outgoing messages and between 15 and 35 incoming messages. The shortest message is 1 word per message for each person. The highest number of words in a chat message is 23 words. The topics in chat messages depend strongly on the test person, there were topics such as health and small talk as well as football or movies. The response time of chat messages was at least 2 seconds and at most 8 minutes. The average response time is about 1 minute and 4 seconds. See Table 2 for details. The users furthermore edited or opened one document (.ods or .pdf) during the 30 minute timeframe, which is also consistent with the actions of the framework.

<i>Chatting Parameters</i>	<i>Framework</i>	<i>Test Persons</i>
Outgoing chat messages	22	(7/46/19)
Incoming chat messages	23	(15/35/21)
Length of chat messages	(1/18/8)	(1/23/4)
Response time	(2s/175s/45s)	(2s/480s/64s)

Table 2: Observed chat behavior (min/max/avg)

5 Discussion

Regarding the browsing habits of the framework we could show that the number of visited websites is reasonable compared to the results from the test persons. The time on each website is on average shorter than the time of the test, but this is a parameter which can be easily changed (just like the number of websites to visit). Some test persons stayed more than 10 minutes on one site, but in general the simulator fits into the timely patterns of the test persons. 4 of 5 of the most visited websites of the simulator are equal to the most visited websites of the test persons, which is a very good result as the websites were a-priori configured. However, the sites visited actually per person, depend strongly on user’s preferences and have to be adapted. In summary we can say that the surfing feature is properly simulated by framework, but needs a user-specific configuration. Table 3 shows the overall comparison between the alibi framework and the values of the test persons.

Regarding chat and using the social interaction server, we were able to observe that the response time to answer chat messages fits the expected time frame. The framework does not response to every message, and the time it waits for sending a response is within the observations from the real users.

5.1 Limitations and Future Work

One limitation of our prototype is the lack of sophisticated contextual analysis of instant messages. While AIML can be used to generate genuine-looking conversations for short periods of time, a forensic analysis of extended conversations probably would reveal the chatbot. While this is definitely a problem in scenarios where the disk is analyzed forensically, generated alibis would most likely withstand network forensic analysis because most protocols such





	Feature	Simulator	Test persons		
	visited homepages	11	min: 1	max: 12	✓
	time on homepage	1m 52s	min: 5s	max: 2m 50s	✓
	most visited homepages		matching in 4/5 sites		✓
	emails (out)	1	min: 0	max: 3	✓
	emails (in)	2	min: 0	max: 0	✗
	length of emails (words)	6	min: 6	max: 51	🌀
	content of emails		matching in 1/4 topics		✓
	chat messages (out)	22	min: 7	max: 46	✓
	chat messages (in)	23	min: 15	max: 35	✓
	length in words	avg: 8	min: 1	max: 23	✓
	response time	avg: 45s	min: 2s	max: 1m 4s	✓
	content of conversation		matching in 2/5 topics		✓
	opened documents	1	min: 0	max: 2	✓
	document type	.ods	.ods, .odt, .pdf		✓

Table 3: Overall comparison framework vs. survey users

as Skype implicitly encrypt messages anyway. For unencrypted email conversations, this limitation is more important. In future work, we thus aim at identifying methods for more content-dependent responses.

Another limitation is adaptivity. To forge a digital alibi reliably it is necessary to adapt the framework to the user's preferences. For comparison of efficiency of the framework, the values, ranges and preferences of results of test systems were taken for reference values. To fit the typical system usage of an individual, there are several possibilities to adapt the framework. Currently, most parameters of the framework are configured manually and have to be adapted for each user. In the future the framework should be able to adapt those parameters automatically. This may be realized by either collecting the user's specific information from user data or by collecting it over a longer period as a learning phase. We would also like to compare long-term runs of the framework with real user data, as 30 minutes is not particularly long enough for all use cases where a digital alibi might be needed. Testing

and supporting other operating systems as well as other browsers is also a goal for the future.

If the user has insufficient knowledge of the tools or the system it may happen that there is unwanted evidence left behind. It is essential to continuously upgrade and improve the framework, because operating systems and techniques are constantly changing as well. We did not implement any obfuscation or methods to hide the execution of the framework. Running it from external media as suggested in [5] should be for example just one item in the list of additional steps, as these techniques may strengthen the validity of a forged alibi.

6 Conclusion

In conclusion, we were able to show that it is possible to forge digital alibis with social interaction. We implemented a proof of concept and showed in the evaluation that the results of forensic analysis of our framework meet the range of values observed from real users. The framework has the ability - if it is configured correctly - to simulate browsing, write and respond to emails, chat with buddies, and opening and/or editing documents. The exact execution of the simulation depends on the configurations that should be adapted to user's preferences. In order to show that the behavior simulated by the framework differs from the actual user's behavior, a very intimate knowledge of the user's habits and usage patterns is necessary. In the future, we want to add a component for automatic configuration based on either existing log files, or use an additional learning phase to obtain an user-specific configuration that is even more indistinguishable with respect to a forensic investigator.

References

- [1] P. Albano, A. Castiglione, G. Cattaneo, G. De Maio, and A. De Santis. On the construction of a false digital alibi on the android os. In *Intelligent Networking and Collaborative Systems (INCoS), 2011 Third International Conference on*, pages 685–690. IEEE, 2011.

- [2] D. Brezinski and T. Killalea. Guidelines for evidence collection and archiving. *Request For Comments*, 3227, 2002.
- [3] F. P. Buchholz and C. Falk. Design and implementation of zeitline: a forensic timeline editor. In *DFRWS*, 2005.
- [4] B. Carrier. *File system forensic analysis*, volume 3. Addison-Wesley Boston, 2005.
- [5] A. Castiglione, G. Cattaneo, G. De Maio, and A. De Santis. Automatic, selective and secure deletion of digital evidence. In *Broadband and Wireless Computing, Communication and Applications (BWCCA), 2011 International Conference on*, pages 392–398. IEEE, 2011.
- [6] A. Castiglione, G. Cattaneo, G. De Maio, A. De Santis, G. Costabile, and M. Epifani. The forensic analysis of a false digital alibi. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 114–121. IEEE, 2012.
- [7] A. Castiglione, G. Cattaneo, R. De Prisco, A. De Santis, and K. Yim. How to forge a digital alibi on Mac OS X. *Multidisciplinary Research and Practice for Information Systems*, pages 430–444, 2012.
- [8] CNN. Facebook status update provides alibi. (november 12, 2009), <http://edition.cnn.com/2009/crime/11/12/facebook.alibi/index.html>, June 2009.
- [9] X. A. W. Community. Garlasco, alberto stasi acquitted (december 2009), http://richardseifer.xomba.com/garlasco_alberto_stasi_acquitted, June 2013.
- [10] A. De Santis, A. Castiglione, G. Cattaneo, G. De Maio, and M. Ianulardo. Automated construction of a false digital alibi. *Availability, Reliability and Security for Business, Enterprise and Health Information Systems*, pages 359–373, 2011.
- [11] S. L. Garfinkel. Digital forensics research: The next 10 years. *Digital Investigation*, 7:S64–S73, 2010.
- [12] S. L. Garfinkel. Digital media triage with bulk data analysis and bulk_extractor. *Computers & Security*, 2012.

- [13] A. Hoog. *Android forensics: investigation, analysis and mobile security for Google Android*. Access Online via Elsevier, 2011.
- [14] A. Hoog and K. Strzempka. *iPhone and iOS Forensics: Investigation, Analysis and Mobile Security for Apple iPhone, iPad and iOS Devices*. Elsevier, 2011.
- [15] M. Huber, M. Mulazzani, M. Leithner, S. Schrittwieser, G. Wondracek, and E. Weippl. Social snapshots: Digital forensics for online social networks. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 113–122. ACM, 2011.
- [16] J. Olsson and M. Boldt. Computer forensic timeline visualization tool. *digital investigation*, 6:S78–S87, 2009.
- [17] T. N. Y. Times. I’m innocent. Just check my status on facebook. (november 12, 2009), http://www.nytimes.com/2009/11/12/nyregion/12facebook.html?_r=2&, June 2013.
- [18] R. Wallace. The elements of aiml style. *Alice AI Foundation*, 2003.