Cross-Site Framing Attacks

Nethanel Gelernter
Department of Computer Science
Bar Ilan University
nethanel.gelernter@gmail.com

Yoel Grinstein
Department of Computer Science
Bar Ilan University
yoelgri@gmail.com

Amir Herzberg
Department of Computer Science
Bar Ilan University
amir.herzberg@gmail.com

ABSTRACT

We identify the threat of *cross-site framing attacks*, which involves planting false evidence that incriminates computer users, without requiring access to their computer. We further show that a variety of framing-evidence can be planted using only modest framing-attacker capabilities. The attacker can plant evidence in both the logs of popular reputable sites and in the computer of the victim, without requiring client-side malware and without leaving traces.

To infect the records of several of the most popular sites, we identified operations that are often considered benign and hence not protected from cross-site request forgery (CSRF) attacks. We demonstrate the attacks on the largest search engines: Google, Bing, and Yahoo!, on Youtube and Facebook, and on the e-commerce sites: Amazon, eBay, and Craigslist.

To plant pieces of framing evidence on the computer, we abused the vulnerabilities of browsers and weaknesses in the examination procedure done by forensic software. Specifically, we show that it is possible to manipulate the common NTFS file system and to plant files on the hard disk of the victim, without leaving any traces indicating that these files were created via the browser.

We validated the effectiveness of the framing evidence with the assistance of law authorities, in addition to using prominent forensic software. This work also discusses tactics for defense against cross-site framing and its applicability to web-services, browsers, and forensic software.

Categories and Subject Descriptors

J.0 [**Computer Applications**]: General

Keywords

Web attacks; Security; Forensic; Framing

1. INTRODUCTION

ACSAC '15, December 07-11, 2015, Los Angeles, CA, USA

DOI: http://dx.doi.org/10.1145/2818000.2818029.

Computers offer high reliability for data retention, and indeed, computer records are considered reliable and trustworthy. In many countries, computer records are often used in criminal investigations and admitted as legal evidence. *Digital, computer, and network forensics*, the science of collecting forensic evidence related to the use of computers and networks and to crimes involving them, is an important and well-established discipline. It has many practitioners, methodologies, tools, and publications (e.g., [8, 18]). It is not surprising that law-enforcement authorities spend a considerable amount of effort collecting computer-forensic records for investigation and prosecution purposes.

Usually, computer records accurately reflect the actions of the user, even when these actions are illegal or violate social, business, or ethical codes. These records serve as confirmation even when the user denies any involvement in these actions when confronted with these records. However, there are several known incidents in which computer records were manipulated intentionally to cause a false impression of wrongdoing, i.e., to *frame* the user. For example, Spencer [27] presents case-studies of sophisticated, high-profile forgeries, with very significant repercussions, which were exposed only using advanced forensic techniques. Nevertheless, only limited attention has been given by the research community to the risk of forged digital evidence and its potential use in framing users. In fact, this threat is rarely even mentioned. This is in contrast to the related area of digital image and video, where there is substantial effort to develop techniques that detect forgery. See survey [25].

While such computer-framing incidents are hopefully rare, the damage can be significant. For example, consider the case of Michael Fiola [6]. In 2007, a technician (accidentally) found child pornography in the browser-cache of Fiola's computer. Fiola was fired and charged with possession of child pornography, which carries up to five years in prison. He endured death threats, his car tires were slashed, and he was shunned by friends. The charge was dropped only a year after the case was filed, when an inspection for his defense revealed that the laptop was severely infected. Fiola suffered tremendous amount of financial, emotional, and even physical damages.

A major argument for Fiola's vindication was the identification of viruses on Fiola's computer. Furthermore, the evidence against Fiola consisted mostly of files containing illegal content, and the web-history did not support search and access to these files. Experts concluded that the illegal files were downloaded by malware, which was controlled remotely by an unknown agent for his own purposes. What

would have happened if Fiola was *intentionally* framed and the investigation would not have identified any malware on his computer? Suppose further that an investigation would have found seemingly-supportive evidence, such as a web-history full of visits to pedophile sites, purchases and sales of suspect content on eBay, search history of pedophile-related terms in sites including Google, Facebook, Yahoo!, Bing, Youtube, and Craigslist. What would have been the outcome?

Computer-forensics has become an integral part of criminal investigations and the resulting evidence is used in many trials. Defendants often argue that they were not responsible for the illegal content, which was collected by a virus or otherwise without their awareness. This argument is often ridiculed and referred to as 'the dog ate my homework' excuse. The common view among experts is that these claims are mostly false. However, it is conceivable that *some* pieces of evidence are due to intentional framing. They may also be a result of 'unintentional framing', e.g., to hide traces of the real criminal. Alternatively, the evidence may simply be due to the operation of malware on the computer (for other purposes). Unfortunately, in most courts and jurisdictions, the burden of proof in such cases shifts to the defendant, who is expected to show that a virus or other malware exists in the system, and that the incriminating evidence is likely to have resulted from its operation [9]. Providing such vindicating proof can be challenging or infeasible, especially if the malware was designed to hide traces. And what if there simply is no malware?

In this paper, we present the threat of *cross-site framing attacks*, in which a computer user is *intentionally framed* by a malicious adversary, using only cross-site communication with the victim's computer, and in particular, *without requiring the adversary to control the computer via malware or otherwise*. Such attacks can be deployed using limited capabilities and resources. We believe our work can help motivate further study of such attacks, the adoption of appropriate defensive measures, and increased caution by forensic analysts. We discuss defenses (as well as attacks), but significant challenges remain. Improved defenses against framing attacks are vital for preventing wrongful convictions, as well as preventing the real culprits from casting doubt over the computer-forensics evidence against them.

Cross-site framing attacks only require that the victim visits a malicious website. This is a relatively modest requirement that the attacker can often ensure. We show how such framing attacks allow attackers to 'plant' a wide variety of incriminating evidence involving alleged-activity in many different and popular sites. The fact that the different pieces of evidence are of different forms, and involve multiple popular sites, makes the overall set of (framed) evidence a formidable argument for incrimination.

We separate the discussion between framing evidence that is planted in the logs and history-records of websites (framing *web-services evidence*), and evidence that is planted on a device (framing *computer evidence*).

When investigating web-services evidence, we exclude attacks that exploit 'regular' site or browser vulnerabilities. This includes vulnerabilities that allow the attacker to take over the victim's account, or allow the attacker to run a malicious script that the victim's browser thinks is coming from the third-party web-service (i.e., XSS). These are known risks and there is no dispute about the need to block them.

Table 1: Framing evidence in web-services

	Rank [4]	Search history	Items history
Google	1	Search and links followed	Videos, news, ads
Facebook	2	Search	-
Youtube	3	Search	Watched videos
Amazon	4	-	Watched items
Yahoo	5	Search	-
eBay	8	-	Watched items
Craigslist	10	Saved searches	-
Bing	18	Search and links followed	-

Table 2: Legal cases and digital evidence used

Type	Cases	Search history	Files
Pedophilia	[6]		√
Hit-and-run	[10]	√	
Hacking	[23]	√	
Online piracy	[24]		√
Murder	[3] [22]	√	
Murder	[12]	√	√

In spite of this limitation, we found that it is possible to 'plant' fake 'evidence' of different types, in most popular sites, as shown in Table 1. Some examples of web-service evidence are as follows:

Search history. The terms a user searched for reflect her interests and can be vulnerable to manipulation.

Relevant items history. Watched videos, watched items, and clicked advertisements are examples of data that the attacker can easily manipulate to forge the interests of her victim.

We also show several types of evidence that an attacker may be able to 'plant' in the victim's computer, specifically:

Browser cache. Files can be planted in the browser cache without leaving traces.

File download and browser history. Exploiting the browser's features, it is possible to automatically download files to the computer of the victim and to add entries in the browser's history.

File system manipulations. We show how to manipulate the common NTFS file system, which is used in all the latest Windows operating systems, to plant files on the hard disk of the victim's computer. Our technique plants the files such that they are not linked to the web. Namely, the forensic software indicates a file found on the hard disk, without linking it to browser-related folders.

The types of digital evidence we planted were used in legal cases. Examples can be seen in Table 2. In addition to the use of framing in the legal context, an adversary may also use framing to discredit a victim in the social, workplace, business, or political context. In particular, the adversary can plant evidence to cause false beliefs about an individual, which may harm that individual and potentially benefit the adversary. For example, an adversary may plant false-evidence about sexual orientation, infidelity, or other issues.

Evaluation by Government Forensic Experts

We approached the National Cyber Unit of the National Crime Unit (Lahav 433) in the Israel Police, and the Computer Forensics Lab within the Department of Investigations

in the Israeli Law, Information and Technology Authority (ILITA) of the Israel Ministry of Justice. We asked for their help in evaluating how the (fake) evidence produced by our attacks affected their forensic investigation process. Both organizations agreed to cooperate under their limitations.

We created two virtual machines (VMs) containing the results of our framing attack. We asked them to run their standard forensic procedure and let us know the results. Specifically, we wanted to know whether their procedure detected the framing evidence and whether there was any indication or warning that the evidence may be fake.

The first VM was framed with the following evidence: (1) visiting a terrorist's website, (2) search history and followed links in Google, (3) search history in YouTube, and (4) automatically downloaded file. We used our 'trace covering' methods as described in Section 6.1. On the second VM machine we planted an image on the hard disk, as described in Section 5.2.

Both departments could not disclose the details of the forensic investigation procedure, but gave us important feedback. The forensic experts in ILITA evaluated the computer based framing attacks in the first VM, and reported that all the planted pieces of evidence were found by their advanced forensic software. However, they mentioned that in one of the examined attacks (they could not specify), their experts, following their extensive forensic procedure, identified an anomaly, which would have resulted in careful further investigation and evaluation of additional evidence.

The police helped us evaluate the attacks using two advanced and expensive forensic software tools: Encase and IEF (unavailable to us). The police experts also told us that, following our work and the tests they ran on our VMs, they updated their forensic investigation procedures.

1.1 Contributions

The basic conceptual contribution of this paper lies in identifying and calling attention to the threat of framing, especially via cross-site attacks.

The 'classic' computer-framing attack requires physical access to the device or remote control over the devices, as with malware. We identify and demonstrate the more insidious threat of *cross-site framing attacks*, which do not require physical access or control over the computer by malware or otherwise. Such attacks are easier and less-risky to launch and may be harder to defend against.

Additional contributions of this paper are in the identification and the evaluation of risks that have not yet been studied in popular web-services, browsers, and operating systems. These include:

- Planting search history is possible in popular and reliable sites (Section 3; see Table 1).
- Exploiting and evaluating automatic file download in Google Chrome and Safari for Mac OS as well as other risky browser features (Section 4).
- Manipulations of file systems to unlink framing files from the web (Section 5).
- Covering the traces of cross-site attacks, both in the victim's computer and in logs of the web-services (Section 6).
- Discussing mitigation techniques and offering a design for effective defense (Section 7).
- Evaluation of the attacks by forensic software and with the collaboration of legal authorities.

Following our work, the Israel Police updated their forensic investigation procedures. This is a strong indication regarding the impact of our results and the importance in publishing them. We hope legal authorities in other countries will also test and improve their forensic procedures.

Demos of the attacks are available in [15].

1.2 Related Work

There is extensive research on different attacks by rogue websites on their visitors, including many cross-site attacks exploiting weaknesses in popular websites, e.g., [2, 28], and off-path attacks exploiting network-protocol weaknesses [16]. However, to the best of our knowledge, this is the first paper that raises the risk of cross-site framing attacks.

In this work, we sent forged cross-site requests to manipulate popular websites. Xing et al. [28] used similar manipulations but only to pollute user personalization algorithms in Google, Amazon, and YouTube.

2. ADVERSARY MODEL & ROADMAP

We consider an adversary that is running a malicious website, without eavesdropping or MitM abilities. We assume the adversary is able to 'lure' the victim into visiting the website; we justify this assumption below.

While the victim visits the attacker's website, we assume the browser will run scripts on that page using typical 'sandbox' mechanisms. For example, these mechanisms let scripts instruct the browser to display objects from arbitrary domains (e.g., images) and load other pages (embedded in frame using <iframe> or in separate window/tab).

The malicious script is often referred to as a *puppet* [5], since it is running within sandbox limitations.

We now discuss the roadmap of our framing attacks.

Luring the victim to the attacker's website. Cross-site framing and other attacks by a malicious website need to cause the user to visit the malicious site. There are several ways the attacker can cause a random user, or even a specific user, to visit his website. These range from legitimate site-promotion techniques, to the use of (targeted) phishing emails and social-engineering [13, 19, 20], or even the take-over of a benign (but not well protected) site.

Attacks on a specific site (Section 3) require that the user is authenticated to that site. Many users are authenticated to several sites most of the time, and since our attacks include some of the most popular sites, this assumption is generally true. In other cases, the attacker may use social engineering to coerce the user into connecting to the desired website.

Planting evidence. Once the victim loads the adversary's website, the adversary can plant incriminating evidence using the techniques described in the following sections.

Covering traces. The adversary can use several techniques to hide the attack from the victim and eliminate the attack's traces from both the victim's computer and from the logs of the web-services.

3. FRAMING WEB-SERVICES EVIDENCE

Records kept by service providers such as banks, utility companies, telecommunication providers, and others are of-

ten considered reliable evidence and are generally admissible in court. With the growing familiarity and usage of many different popular web-services, their records are also increasingly viewed as legitimate, reliable evidence, and have been applied in several court cases (see Table 2).

As with other evidence, these have a cumulative value. Namely, a collection of a large amount of web-service evidence can have considerable weight, especially if it includes different forms of evidence from multiple web-services. We next argue that such evidence should be used with care. We show that popular, reliable, and widely-trusted web-services, may often allow such records and evidence to be easily planted. See summary in Table 1.

The vulnerabilities we present may not pose an obvious business risk to the providers, beyond their potential abuse for creating fake evidence. Consequently, the web-service providers may not have significant business motivation to fix them. This may be an important difference between the public perception of record keeping by a reliable, trustworthy organization, and the reality of web-services. This also raises interesting ethical, legal, and social dilemmas. Should society demand that web-services do more to protect their records and prevent framing? Is there an ethical obligation on web developers and security experts to consider and fix such vulnerabilities? Are such records subject to privacy laws and regulations?

In the adversary model discussed here, an attacker can launch CSRF attacks [2] to perform framing operations in the name of the victim. We show how, by performing innocuous operations in the name of the victim, it is possible to create new framing records in the logs kept by the web-services. We do not discuss framing operations such as sending or posting messages in the name of the victim, as it is clear that sites should prevent attackers from performing such activities. Instead, we focus on simple operations that do not appear to be suspicious; some of the popular websites we tested do not protect these operations from CSRF attacks. We categorize these operations into two categories: search history and relevant items history.

3.1 Search History Evidence

Search is a basic operation performed by search engines and many other websites. Websites save the search history of their users, to give them personalized services, e.g., provide more relevant content. However, the search history is private; the terms that a user searches for, may expose information about the user and her needs and interests. We found that by sending a cross-site search request, an attacker can add a record of this search to the logs kept by websites. In some sites it is also possible to manipulate the clicked search results or to add saved searches. We now elaborate on the different attacks we found.

Search history. Search engines, and other sites providing search services, often maintain the history (records) of users' search queries. We checked the search history in the three most popular search engines [4] Google, Yahoo!, and Bing, and for YouTube and Facebook. We found that all of them save a history of the user's search queries by default, even if the queries are sent from other sites. Furthermore, in all engines there does not appear to be any filtering of 'problematic' terms.

All of these sites collect users' search history, even when the user surfs privately (e.g., in Chrome *incognito mode*) and/or chooses privacy-options such as *Send "Do not track" request with your browsing traffic* in Google Chrome browser [17]. In particular, this implies that search-history framing is also possible from a site visited while in 'incognito' mode.

Followed-links history. Two of the three search engines we tested, Google and Bing, maintain and provide an interface showing the history of links followed ('clicked') by the user from among the search results; this is used for different (legitimate) purposes. However, these records can also provide additional (framing) evidence about user activity, possibly even more incriminating than the search history. We found vulnerabilities allowing the insertion of fake records into the followed-links history of both Google and Bing, each of these was done using a completely different technique, as we now explain.

Followed-links framing - in Google. When Google presents search results, the links provided with each result are not direct links to the corresponding pages. Instead, the links are all GET requests to Google, with a parameter that indicates the destination URL. Clicking on a link redirects the browser to the destination URL and adds that URL to the followed-links history. It turns out that sending the same GET requests when other clients are authenticated from other websites, has the same effect; see demo in [15].

Followed-links framing - Bing. In Bing, we found a different vulnerability that offers the same result, i.e., injecting entries to the followed-links history. Specifically, Bing allows its search page to be used within a frame (<iframe> tag), embedded within another website, permitting *clickjacking* [26]. To inject a URL into the followed-links history of Bing, the attacker embeds this link inside an iframe, and overlays this with a layer that causes the user to click on that link (without being aware of this being a link).

Saved searches history. Some web-services allow users to save selected searches in their profile. In particular, Craigslist offers such a mechanism, using a simple fixed-request format that can be used from an arbitrary website. Attackers can therefore inject fake 'saved search' records.

3.2 Evidence of Relevant Items History

E-commerce and content websites save the items in which users express interest. This is done to personalize the pages presented to the user and to offer her more relevant items. It also allows the website to learn about trends and offer information for other users. Attackers can easily manipulate these records to plant fake indications about the interests of the users.

We now bring examples from the popular websites we tested:

Clicked videos, news, and advertisements in Google. Similar to the followed-links framing in Google, it is possible to take links to videos, news, or advertisements that appear in the search, and send them from the attacker's website. The items that are related to the links will be added to the history of the victim.

Watched video history in YouTube. In addition to search history, YouTube also maintains and displays the 'watched videos'. YouTube's mechanism is similar to Google's 'followed links', and has the same vulnerability. Specifically, it adds videos to the user's history upon an HTTP GET request, normally sent by the script running in the browser of a user visiting YouTube. However, the same

request may be sent from the browser upon visiting a rogue website, adding a video to the user's history.

Amazon watched items. Amazon saves the items watched (clicked) by the user. An attacker can copy the GET requests linking to specific items from the search results returned by the Amazon e-commerce service. These links can then be invoked on the attacker's website when it is visited by an Amazon client. This will cause Amazon to list these items as viewed by the client.

Watched items in eBay. eBay allows its users to add products to both the shopping cart and the watch-list. While this is not a purchase of product, it indicates the user's interests.

4. COMPUTER FRAMING

The framing attacks discussed in Section 3 exploit vulnerabilities of the web-service. Here we present framing attacks that exploit *browser features and vulnerabilities*, instead of web-server vulnerabilities as in previous sections. The framing attacks we describe in this section include 'classical' pieces of evidence that are found on digital devices, such as files and browsing history.

4.1 Framing via Files in the Browser Cache

The screening of a suspect's computer to search for incriminating files is a standard forensic procedure [8, 18]. Specifically, it is recommended to check files in the browser cache; indeed, cached-files were reported as evidence in several of the cases we surveyed in Table 2.

Web users often visit the same pages several times. Hence, browsers automatically save received pages and other objects in a cache. The browser used the cached objects when the user visits the same page again, if the contents are still valid. For each object (file), browsers save the content as well as meta-data such as the URL, download time, and expiration time.

Browsers normally allow any website to request arbitrary objects and web pages. Furthermore, the cache does not maintain a record of the site originating the request. Hence, a framing attacker can cause the browser to load incriminating content (e.g., in iframes). The content can be taken from different websites on the Internet or from the Deep Web via services like Tor2web [1]. This would allow the attacker to provide customized content from a site controlled by the attacker, without leaving traces. In short, it is easy for a framing attacker to cause the caching of arbitrary incriminating files and objects. While users are technically able to inspect their browser cache, remove specific items or simply clean the cache, most users rarely do it, if ever. Therefore, attackers can assume that incriminating, false-evidence files that are stored in the cache, will remain there for long periods, without the user noticing. Moreover, traces of cached files may remain on the disk even after deletion.

4.2 Framing via File Download

Browsers allow users to save or download web-objects (e.g., complete web pages, images, movies, and documents) usually to a default directory. The download is generally initiated by the user. Web pages can also initiate the download process, however, this is less common, and the user is asked and/or allowed to cancel the download. Consequently, incriminating files in the downloads folder may be more damning than files stored (automatically) in the browser cache, as discussed in the previous subsection.

In at least two popular browsers, Google Chrome and Safari for MacOS, files are downloaded *automatically* by default, without asking users for explicit consent. Once the file is downloaded, the user has to delete it via the regular file system. Note that forensic software can often find such incriminating evidence even after it was deleted (usually, until it is eventually overwritten by new data). The relevant questions are, therefore: how effective is incrimination by exploiting the automatic file download feature? Would users notice and abort the download, or later delete the file? We tested these questions in the experiment described in Section 4.2.1.

4.2.1 Experiment: Automatic Download Framing

Goal: Determine for how many users automated download of files will work and how many would abort the download or remove the file.

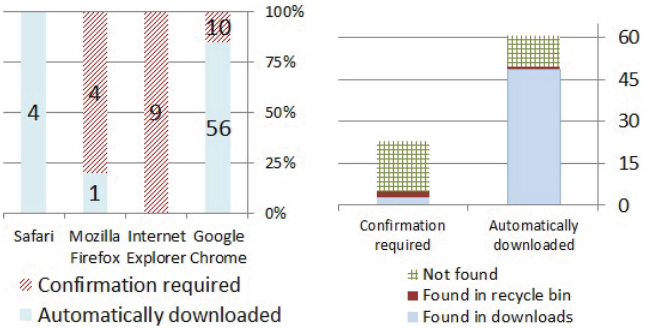
Methodology and ethics: We did not expect users to react similarly to automated download attempts on a computer we provided to them for the experiment. Such reactions are likely to be biased, possibly even more so if we explain to the users that our goal is to measure their reaction to automated file download. Hence, we could not conduct the experiment in our lab or on an experiment-computer. We had to create a natural environment in which users would use their personal computers for a typical purpose, and then test their responses to an automated download attempt.

We created a web page containing an online practice-exam for students in the Data Structures course (first year undergraduate computer science course). We then published a link to this page to a group of 165 students in the course. Students who solved our exam were asked to add their email to get the answers for the exam and receive updates about additional exams that will be published. This web page containing the first exam tried to load a file in a new iframe, which initiates the downloading file procedure. The file was a 1MB zip file with an image protected by a password, downloaded from an anonymous file storage site.

Two days after the deadline for this first online exam, we sent an email with a new exam to the 108 students who solved the first exam. At the end of this second exam, we asked the students who solved both exams on the same computer to help us in the experiment by checking whether the file was in their downloads folder or in their recycle bin. We also asked them about their browser and OS, and referred them to a web page where they could check whether their browser prompts them with a message before downloading files. Participation was voluntary; 84 students participated using the same computer in both exams and replying to the questions.

Experimental results Out of the 84 participants, 61 participants (i.e., 73%) had their browsers configured to 'automatically download' files. Most participants, and also most users with 'automatically download' set, used the Chrome browser, see Figure 1(a). A large majority among these users (79%) also reported finding the file in the downloads folder. In total, 60% of the participants found the file in the downloads folder, and few more found it in the recycle bin. See Figure 1(b).

Our results indicate that the 'automated download' feature significantly increases the risk of framing. Notice, how-



(a) Participants' download features, by browser

(b) Files found, by download feature

Figure 1: Results for file download experiment

ever, that the few users who did not have 'automated download', still had the file. They apparently downloaded the file manually when prompted by the browser.

We recommend that vendors reconsider the use of fully automated download, and also consider adding metadata to identify the origin site and the type of download, such as user-initiated versus site initiated, and automated or approved.

4.3 Framing by Browser History

By default, browsers maintain *history* records of the requests sent to different web pages. This browser history is routinely mentioned in computer forensic literature and guidelines [8,18] as an important source of forensic evidence. The browser history also includes the exact search strings used in requests to most search engines. This search history is mentioned as important forensic evidence in many of the court cases we viewed (Table 2). However, this evidence does not appear with an indication of its source, whether browser, provider, or history record provided by the site to the user.

Consequently, framing attackers may try to inject forged entries into the browser history, to create another source of framing evidence, complementary to the website-history and cache evidence. Note that the requests used for the website history framing would not appear in the browser history, since these are injected from embedded objects (such as IMG and IFRAME tags).

To inject browser history, all the adversary has to do is open a web page briefly, in a small new window. Although all the browsers we tested block JavaScript from prompting windows arbitrarily, they do allow opening a new window when the user clicks a button or a link and closing it immediately. Using clickjacking techniques [26], this behavior can be abused to inject browser history. See demo [15].

5. FILE SYSTEM MANIPULATIONS

In Section 4 we began the discussion about planting pieces of evidence in the victim's computer. All the pieces of framing evidence we discussed there were caused by an interaction with the web, and could be identified as such. However, there is other digital evidence that is not related to the web, such as files on the hard disk. In this section we present the *partial overriding attack (POA)*, which allows a rogue website to manipulate the file system and plant a framing file on the hard-disk, with no indication at all that this file was received from the web.

In the POA attack, the malicious website embeds an object (e.g., image) that is a composition of a legitimate file and the malicious framing file. Then the malicious website causes the browser to partially *override* the legitimate part of the file, leaving the framing file floating in the hard disk.

We found that prominent forensic tools detect the injected file as a deleted file, without any warning, e.g., that this file may have been received from the web rather than created locally.

We first applied the POA attack on the old and simple FAT32 file system, which is still used by removable media. We then applied the attack on the common NTFS file system. In this Section we focus on the NTFS file system, and begin with a brief background in Subsection 5.1. In Subsection 5.2 we describe the framing attack. We evaluate the attack using forensic software in Subsection 5.3.

5.1 Background on NTFS

NTFS is the file system of modern Windows operating systems. All files, directories, and their metafile data (i.e., file name, creation date, access permissions and size) are stored as metadata in the *Master File Table (MFT)*.

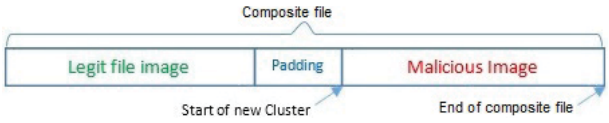
The smallest logical amount of disk space that can be allocated to hold a file in the NTFS is called a *cluster*. Usually, the default cluster size is 4 KB; we used this size during our experiments. The cluster is a logical limit, as compared to the sector size, which is the physical limit set by the manufacturer for the drive (512 bytes for old hard-disks, 4 KB for newer ones). Each file is split into one or more clusters, depending on its size. This means that every file has two different sizes: the exact size of the file in bytes, and the total space the file actually takes up on the disk, which is divisible by the cluster size. The one exception is tiny files, typically less than 900 bytes, which can be stored directly in the MFT without allocating any clusters.

A new feature of NTFS (cf. to FAT) is the journal file. The journal file logs every action that is committed to file system's driver. Hence, the journal file can provide indications of the POA. However, none of the forensic programs applied in this research issued any warning, e.g., about missing journal records; this includes programs we used and programs available only to the law enforcement units.

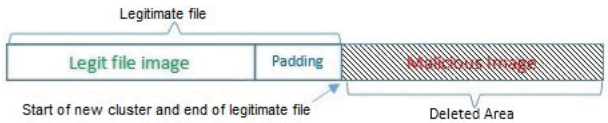
5.2 Partial Overriding Attack (POA)

The POA attack has two steps: (1) The attacker identifies the victim surfing in her rogue website. That attacker then creates a *composite file* that is a concatenation of a legitimate file and the framing file (see Figure 2(a)) and causes the victim's browser to load it (e.g., as a legitimate image). (2) The rogue site initiates an additional request for the same file; this time, the attacker replies with a shorter (legitimate) file that overrides the prefix of the original file. This procedure leaves the framing file (the suffix of the composite file) floating in the disk; namely, the framing file is not connected to any file entry in the journal (see Figure 2(b)).

In the composite file, the legitimate part might be an image that the victim can see in the website without anything appearing to be suspicious. This legitimate part is placed together with some padding to reach a multiplication of the cluster size. The framing part can contain arbitrary content, such as a pedophilic photo. While surfing, the victim sees



(a) File system after downloading the composite file.



(b) File system after overwriting the composite file with a legitimate file of the same name. The malicious part of the file is marked as deleted, while the other parts belong to the legitimate file.

Figure 2: The composite file in memory during the POA attack

only the legitimate image, because the rest of the file does not match the image format.

This attack is simple to implement and the framing file does not leave any traces pointing to the attacker or any indication that it was received from the web.

5.3 Evaluation

We successfully tested the POA attack on the latest versions of three of the most popular browsers: IE 11.0.96, Firefox 38.0.1, and Chrome 43.0.2357. On Safari 5.1.7 the attack failed because Safari stores its cache in a SQLite file, which contains different offsets for padding.

To implement the attack, we built a web page that embeds an IMG HTML tag with the composite image as the source. We added a JavaScript code that initiates a second request for the same image once the browser has completed loading the image for the first time. We replied to the second request with a shorter legitimate image that overwrites the prefix of the composite image. At the end of the process, the framing file is floating in the hard disk.

Evaluation with forensic software. We used two known forensic software tools to evaluate the attack: Autopsy and OSForensics. The National Cyber Unit in the Israel Police assisted us in the examination of the attack using two additional tools, Encase and IEF; both are expensive, sophisticated forensic software tools. All of these tools detected the framing file and did not give any warning of anything unusual. In particular, they did not indicate a framing attempt or even that the file originated via the web.

As expected, the forensic software tools ignored the fact that there is no journal entry for the framing file. This is most likely due to three reasons: (1) Searching for files as they appear in the hard disk is a general task that can be done on several file systems. (2) The main goal of forensic tools is to find incriminating files; they do not consider the framing threat. (3) Journal entries might be deleted, either automatically, due to lack of space, or intentionally, using dedicated software.

Size of the files. The size of the composite image was 206 KB, of which 55 KB was the framing image. We tried several other sizes, which produced similar results. A study of the optimal sizes requires further experimentation.

Framing evidence lifetime. An important factor of the attack is the lifetime of the evidence. For the evaluation, we used Windows 7 machines in the Google Cloud. To keep the machines active after the attack, we ran a script that randomly loads 1 of the 100 most popular websites, sleeps for a random time of up to 5 minutes, and then repeats the process infinitely.

We ran the attack on the latest versions of the IE, Chrome, and Firefox browsers, and noticed that the file remained on the hard disk for one or several hours. To improve the results, we built a new web page that repeats the attack 100 times to plant 100 framing images. We found that this repetition improved the results significantly and that the files remained on the hard disk for one or several days. We are planning to perform a more conclusive experiment that will run for longer periods on the computers of volunteers.

6. COVERING TRACES

Detection of a framing attack could result in serious repercussions, often including criminal charges. Therefore, risk of exposure can be a major deterrent to potential framing attackers. Naturally, the attackers are likely to consider this risk and take steps to minimize it. Since the attacks involve the use of a script received from the attacker's site, one obvious way to detect the attack on the malicious web page is by identifying this script and/or other content that indicates the intent of framing the user. We now evaluate the ability of the framing attacker to 'eliminate traces' and prevent detection of the script or other suspicious signs such as iframe tags, on the attacker's web page.

6.1 Covering Browser Traces

Normally, the framing web page and the script (within it or as a separate object), would be cached by the browser, similar to other objects. Consequently, it might be possible to find the attacking page in the cache and detect that it actually created the framing evidence.

To prevent a web page from being saved in the cache, the attacker can use the Cache-Control HTTP response header [14]. However, in some cases, although the content of the malicious web page will not appear in the cache, evidence of the visit might remain in the browser history or even in network logs. The mere fact that this page's objects are not cached could then cause them to become suspect, since web objects are almost always cached (except when containing sensitive information).

There is an alternative method that would not raise suspicions and can be used to prevent the caching of the framing web page. Namely, the attacker simply *reloads* a new, benign version of the page. The cache only keeps the latest version of each object, hence, it would simply overwrite the previous version. See Figure 3 and the demonstration in [15].

This process can be done while the original framing page and script continue to operate. Specifically, loading the 'benign' versions of the page and script into a new hidden iframe is sufficient for the browser to overwrite the framing versions in the cache with the benign versions now received.

6.2 Covering Web-Service Traces

We do not know which information is saved by web-services and whether the history information they give to the law authorities contains anything beyond the history available to the user. However, as the recipients of cross-site requests, the web-services could potentially save information that allows exposure to the framing. We first show how

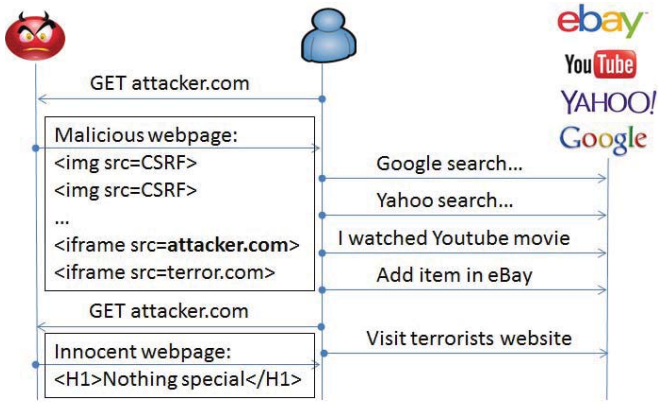


Figure 3: Covering traces. The victim sends a request to the attacker's website. The attacker replies with a web page that sends cross-site requests to different websites and loads web pages in iframes. Then, the malicious page sends an additional request to reload itself in a hidden iframe, returning benign page and script. The browser overwrites the original cached page and script with the newly received (benign) versions.

to avoid these headers and then show that sometimes it is possible to send the requests from other websites.

6.2.1 Manipulating HTTP Headers

In spite of the fact that the attacker controls the content of the HTTP requests, the browser is the one that controls their headers. Some of these headers contain indications of the cross-site requests that the attacker may want to avoid.

Referer header. The Referer header is attached by default to requests and serves to indicate the URL where the request was initiated. However, the Referer header is often omitted. For example, many companies strip this header in the gateway of their network to avoid the information leaked by this header. An attacker can easily test whether the Referer header is stripped by sending a cross-site request to a server she owns. Furthermore, the attacker can cause the browser not to send a Referer, using standard techniques. For example, if the third-party site is insecure (using *http* rather than *https*), sending the request to the third-party site from a protected (*https*) framing-site, would not include the Referer header.

Origin header [7]. The fact that the Referer header is often omitted, alongside its privacy problems, was a major motivation for introducing the Origin header; this has similar functionality but only identifies the domain.

Browsers that support the Origin header are expected to identify the origin domain and attach it to HTTP requests. However, we found that, at least currently, some browsers such as FireFox (as of version 37.0.2) and IE (version 11.0.18), do not attach the Origin header to POST requests that are sent via forms and are targeted to hidden iframe. It is also possible to send GET requests from the IMG tag, such that only the Accept header (see below) might be seen as suspicious.

It is also possible to load the web-service's URL into a new window (and immediately close it) as described in Section 4.3 without generating suspicious header values. See examples in [15].

Accept request header. Modern browsers use this header to specify certain types of media that are acceptable for the response. In all of the attacks noted in this paper, all HTTP GET requests were sent from the SRC attribute of the IMG tag. Hence, the Accept header indicated an image; the servers ignored this and handled the request as a valid form. From our findings, web applications do not pay attention to the Accept header, even though it can be used to easily detect some CSRF attacks. It is possible to avoid a suspicious Accept header by sending the request into a new window.

6.2.2 Sending Requests from Other Sites

The need to manipulate the HTTP headers derives from the fact that the attacker sends cross-site framing requests. However, in some cases, it is possible to cause other websites to send the framing request, in which case the HTTP headers will indicate a legitimate request. We give two examples of such techniques and demonstrate them on the Google search engine.

Exploiting the hash sign (#) in URL. The hash sign separates a URL into two parts. The first part is sent as an HTTP request by the browser and the second is added by the browser once the response arrives. If the addition of the second part initiates another framing request, it will be sent from the page returned by the first request. We found that Google is vulnerable to this technique. In particular, it is possible to initiate a search using the hash sign. From the perspective of Google, the victim was referred to Google from the attacking site, and then (from the Google page) searched for the framing term. Similarly, it is possible to initiate two search requests via one request. We demonstrate the procedure in [15].

Exploiting redirection by JavaScript. Many websites use JavaScript to redirect the users to other pages. Such redirection has a similar effect to the use of the hash sign. The site first loads the page and then the JavaScript code loads the other page. Unlike HTTP redirection (response code 302), where the browser initiates the loading of the target page with the original Referer header, in redirection by JavaScript the page with the JavaScript is the origin of the request. Hence, the browser names it in all the relevant headers.

Redirection by JavaScript also occurs in Google search results. Therefore, an attacker who wants to install a visit from the victim's IP address in the log of some website needs to load a link to the website that was taken from Google search results (see Section 3.1) instead of loading it directly. By doing so, the HTTP headers will indicate that the request came from a normal Google search.

7. DEFENSES

In Section 7.1 we discuss *existing, known* defenses that can prevent the planting of evidence by web-services (as discussed in Section 3). We then briefly summarize the risky browser features that allow the attacker to hide the planting of evidence in Section 7.2. In Section 7.3, we discuss the main challenge of identifying framing files and present countermeasures.

7.1 Web-Service Defenses

The framing attacks in Section 3 all exploited the fact that websites allow cross-site requests for seemingly harmless op-

erations such as search, which do not change the state in the server in a 'meaningful way'. An obvious solution would be to prevent *all* cross-site requests, using existing, well-known cross-site request forgery (CSRF) countermeasures, see [11, 21, 29].

One popular defense is to identify the 'calling' third-party site, using the Referer or Origin HTTP request headers. Another defense uses an (unpredictable) anti-CSRF tokens, sent with the request from the webpage, which is then validated by the server. All the websites we tested use such tokens to protect against CSRF attacks for 'sensitive operations'. Websites which *intentionally* allow some ('harmless') cross-site requests, may, at least, maintain records of the fact that a request was received from a specific third-party domain.

7.2 Dealing with Risky Browser Features

In Section 4 we discussed several browser features that allow hidden file download and browser history injection.

The results of the experiment described in Section 4.2.1 show that automatic download can be used for effective framing attacks because users generally do not bother to delete downloaded files. Two countermeasures can improve the current situation: (1) removing the automatic download from being the default option, and (2) adding a deletion option to the downloads bar so the users can easily (permanently) delete files without having to open the downloads folder.

To inject history into the browser, we suggested opening a website and immediately closing it. While it seems reasonable to allow opening a single window per click event, the ability to close windows might not be that obvious. We have no data about the extent of use for closing windows, so it is difficult to claim that it should be completely blocked. However, it seems reasonable to block one window from closing a window that loads a page with a different origin. Similar to the X-Frame-Options header, which limits loading of web pages in an iframe, it is possible to set a new HTTP response header that will block or restrict pages from closing windows that arrive with a new header.

7.3 Blocking File Manipulations

In Section 4 we discussed framing using files saved by the browser in the cache and in the 'download' folder, and framing via the browser history. Later, in Section 5, we showed that other manipulations can be done to unlink the downloaded files from the browser. We believe that guarding against these threats may require a new defense mechanism. We propose such a mechanism below.

Protection against framing evidence on the computer should meet two challenges: (1) Overcoming cache browser poisoning that is done without leaving traces. (2) Preventing framing files from being left on the hard disk.

Preventing cross-site requests or the loading of web pages in iframes seems impractical. Changing the cache mechanism to also save old requests and avoid covering traces would change the cache to an advanced history feature. This is also a bad idea and might not prevent false-evidence from being placed on the disk.

The crux of the framing by files is the lack of records showing details about the requests that initiated their creation. It seems that a simple solution, with negligible overhead, would be to save for each cache entry the details on how the request was generated. Specifically, saving the values of the *Referer* and *Accept* headers with each request, seems sufficient to avoid such framing attacks, based on what we observed. This idea can also be used to protect against framing via files stored by the browser. By adding an indication of the relevant *Referer* and *Accept* headers, it is possible to distinguish between files downloaded intentionally by the user and files downloaded automatically by some site.

For cached files, this information should be kept together with the file itself. This can be done by creating a new special cache entry file format that will wrap the current format together with the origin data. Attaching the origin data to each cached file makes it highly unlikely that traces of several cache entries in the disk would all have their origin fields missing. If an incriminating file is found on the disk, there would be a good chance that the information about how it was requested appears there as well.

However, for the attack described in Section 5.2, saving additional information about the way the file was created might not be sufficient, because that data might be overwritten by the attacker. A solution that overcomes this attack must ensure that the information about the source is linked to the framing file or, alternatively, destroys the framing file so it cannot be recovered by forensic investigators.

Solutions to the problem can be implemented either at the file system level or in the browser. We concentrate on browser-level solutions, as these are simpler and easier to deploy. The solution we offer uses wiping techniques.

Browser-level wiping. Wiping is a known technique for cleaning information from memory. Wiping is usually done by overwriting the data with zeros or random data. By completely wiping every overwritten file, it is impossible to plant floating files. To implement the wiping at the browser level, one could proceed as follows. A browser about to overwrite a file A with a smaller file B, first overwrites A with a temporary file of the same length that contains only random data. Then, the temporary file is overwritten by B. Upon deleting a file from the cache, the browser should similarly wipe it from the memory.

The overhead caused by wiping does not appear to be significant. This is mainly because (1) most of the files are small, (2) the wiping is done only on a relatively small fraction of file creation operations, when a smaller file overwrites a larger file.

8. CONCLUSIONS

We discussed and presented the threat of remote framing attacks. We showed that it is easy to plant false pieces of evidence in the victim's computer, as well as in 'history' records kept by third-party websites, including many popular reputable websites. We also presented defenses that can be applied to browsers, websites, and forensic software.

We confirmed that the attacks are effective by testing 'framed computers' using popular forensic software and with the cooperation of forensic experts from ILITA and the Israel Police. In particular, the National Cyber Unit in the Israel Police informed us that they updated their forensic procedures following our findings and their experiments on machines that were 'framed' by our attacks.

Although this amount of evaluation and feedback is insufficient to draw conclusions, we consider this an indication that the cross-site-planted, fake evidence could mislead forensic experts.

Framing is an interdisciplinary challenge, and it is our hope that this paper will help stimulate discussion and cooperation among experts in security, forensics, and legal, to understand this challenge and how it can best be met.

9. ACKNOWLEDGMENTS

We would like to thank Yaniv Azani, Koby Furlaiter and the National Cyber Unit of the National Crime Unit (Lahav 433) in the Israel Police, and Pini Cohen, Oren Butchmits and the Computer Forensics Lab within the Department of Investigations in the Israeli Law, Information and Technology Authority (ILITA) for their huge help in the evaluation of our findings. We also thank Hezi Moriel for his useful feedback. This research was supported by grants from the Ministry of Science and Technology, Israel, and from the Israeli Science Foundation.

10. REFERENCES

[1] Tor2web: browse the anonymous internet. http://tor2web.org.

[2] Gmail CSRF Security Flaw. http://ajaxian.com/archives/gmail-csrf-security-flaw, 2007.

[3] M. Aguilar. If You Kill Someone, Don't Google How to Do It First. http://gizmodo.com/5916184/if-you-kill-someone-dont-google-how-to-do-it-first, June 2012.

[4] Alexa Web Information Company. Top Sites in United States (April 2015). http://www.alexa.com/topsites/countries/US.

[5] S. Antonatos, P. Akritidis, V. the Lam, and K. G. Anagnostakis. Puppetnets: Misusing Web Browsers as a Distributed Attack Infrastructure. *ACM Transactions on Information and System Security*, 12(2), 2008.

[6] AP. Framed for child porn - by a pc virus. Online. http://www.nbcnews.com/id/33778733#.U2Ana_l_tLV.

[7] A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 75–88. ACM, 2008.

[8] E. Casey. *Digital evidence and computer crime: forensic science, computers and the internet*. Academic press, 2011.

[9] F. Cohen. *Challenges to digital forensic evidence*. Fred Cohen and Associates, 2008.

[10] D. . C. Court of Appeal, First District. The PEOPLE, Plaintiff and Respondent, v. Lee David HARBERT, Defendant and Appellant. http://caselaw.findlaw.com/ca-court-of-appeal/1089011.html, 2009.

[11] A. Czeskis, A. Moshchuk, T. Kohno, and H. J. Wang. Lightweight server support for browser-based csrf protection. In *Proceedings of the 22nd international conference on World Wide Web*, pages 273–284. International World Wide Web Conferences Steering Committee, 2013.

[12] F. D. District Court of Appeal of Florida. Justin Mertis BARBER, Appellant, v. STATE of Florida, Appellee. http://caselaw.findlaw.com/fl-district-court-of-appeal/1164299.html, 2006.

[13] A. J. Ferguson. Fostering e-mail security awareness: The west point carronade. *EDUCASE Quarterly*, 2005.

[14] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999.

[15] N. Gelernter, Y. Grinstein, and A. Herzberg. Cross-Site Framing Attacks. Demos site. https://sites.google.com/site/framingattacks/.

[16] Y. Gilad, A. Herzberg, and H. Shulman. Off-path hacking: The illusion of challenge-response authentication. *IEEE Security & Privacy*, 12(5):68–77, 2014.

[17] Google. Incognito Mode (browse in private). https://support.google.com/chrome/answer/95464?hl=en.

[18] S. V. Hart, J. Ashcroft, and D. J. Daniels. Forensic examination of digital evidence: a guide for law enforcement. *National Institute of Justice NIJ-US, Washington DC, USA, Tech. Rep. NCJ*, 199408, 2004.

[19] D. Irani, M. Balduzzi, D. Balzarotti, E. Kirda, and C. Pu. Reverse social engineering attacks in online social networks. In *Detection of intrusions and malware, and vulnerability assessment*, pages 55–74. Springer, 2011.

[20] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Communications of the ACM*, 50(10):94–100, 2007.

[21] N. Jovanovic, E. Kirda, and C. Kruegel. Preventing cross site request forgery attacks. In *Securecomm and Workshops, 2006*, pages 1–10. IEEE, 2006.

[22] S. Morris. Vincent Tabak 'researched killings and sentences after Joanna Yeates's death'. http://www.theguardian.com/uk/2011/oct/19/vincent-tabak-joanna-yeates-death, October 2011.

[23] U. S. C. of Appeals. UNITED STATES of America, Plaintiff-Appellee, v. Matthew R. SCHUSTER, Defendant-Appellant. http://caselaw.findlaw.com/us-7th-circuit/1203561.html, October 2006.

[24] RIAA. Piracy Online - The Law. http://www.riaa.com/physicalpiracy.php?content_selector=piracy_online_the_law.

[25] A. Rocha, W. J. Scheirer, T. E. Boult, and S. Goldenstein. Vision of the unseen: Current trends and challenges in digital image and video forensics. *ACM Comput. Surv*, 43(4):26, 2011.

[26] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. *IEEE Oakland Web*, 2:6, 2010.

[27] M. Spencer. Sledgehammer and ergenekon: Case studies in sophisticated digital forgery. In *The United States Cyber Crime Conference*, 2014.

[28] X. Xing, W. Meng, D. Doozan, A. C. Snoeren, N. Feamster, and W. Lee. Take this personally: attacks on personalized services. In *Proceedings of the 22nd USENIX conference on Security*, pages 671–686. USENIX Association, 2013.

[29] M. Zhou, P. Bisht, and V. Venkatakrishnan. Strengthening xsrf defenses for legacy web applications using whitebox analysis and transformation. In *Information Systems Security*, pages 96–110. Springer, 2011.