

The Art of WWWar

Web Browsers as Universal Platforms for Attacks on Privacy, Network Security and Arbitrary Targets

Florian König

floki@vorsicht-bissig.de, 0255220 / 921

Abstract. Browsers are now one of the most widely used pieces of software and are therefore prime targets for attackers. They contain a plethora of private data (history, passwords), can be misused for hijacking sessions in web applications and for directly probing and attacking other systems. In this paper I'm going to describe state-of-the-art attacks on privacy and security and some approaches on thwarting them. The goal is to give an overview on attack vectors, concrete techniques and possible security measures as well as provide ample reference to further literature.

1 Introduction

While the Internet and the World Wide Web have existed for quite some time already it is only in the last 15 years that those world-wide electronic networks and their resources have become accessible and used by the general public. As a consequence web browsers such as Microsoft Internet Explorer, Mozilla Firefox, Apple Safari and Opera are now the most widely used tools to access digital content. With the technical advances in web content standards and scripting languages like JavaScript web browsers have become even more powerful. They today allow the development of interactive web applications that come close to traditional desktop applications both in functionality and usability. Browsers have become a universal platform for running applications.

With this development comes a corresponding rise in threats to the security and privacy of browser users. There are many reasons why the Internet is nowadays increasingly used for malicious activities: More people have access to the Internet, whereas previously only the military and educational institutions had connections. The amount of information that can be gathered online has increased exponentially. This not only means publically available information but also confidential data in online databases and personal information on the users. The Internet has not only become a major market place for legitimate trade but there's also a lot of money to make by engaging in illegal activities like spamming, phishing and click-fraud.

One of the main tools to perform attacks on security and privacy over the Internet is the browser. This paper will look at some current possibilities to bypass security measures in place in order to get information on users, hijack

authentication information, breach firewalls and perform attacks on other computers. In section 2 the threats are classified, detailing the kinds of data that can be extracted from browsers as well as attacks that can be performed. Important terms are defined in section 3. Section 4 shows how current attacks can be performed and provides some hints on how they can be thwarted. Finally some of the findings are discussed in section 5.

2 Threat Classification

During the literature research a certain pattern of threats to web browsers emerged. Three different types of threats could be identified, and in each of those the browser plays a different role. The roles are outlined below.

2.1 The Browser as Source of User Data

Browser environment information. Quite some information like the browser name and version, operating system, computing architecture, preferred natural language, IP address, local time, names and versions of installed plugins (e.g. Flash, Java, Quicktime) as well as screen size can be retrieved from the browser via legitimate ways. In Internet Explorer even more information can be gathered such as the version numbers of system components (e.g. Media Player, Outlook Express, DirectX), installed fonts, if a soundcard is installed, whether you have up-to-date security certificates installed and much more. For a comprehensive list and a possibility to test your browser just go to <http://gemal.dk/browserspy>.

One of the reasons why this data is so valuable is because it allows attackers to better target subsequent actions. Browsers and plugins alike have a history of security holes and by knowing the exact version an attacker can apply just the right exploit. Another way to use this information is when performing a phishing attack. As shown by Ye [1] it is possible to create nearly perfect simulations of browser interfaces in pop-up windows with no browser chrome (address bar, toolbar, statusbar). This can be used to conceal the true identity of a phishing site. Browser environment information enables the attacker to prepare the fake interface elements and simulate the user's browser.

Personal information. In the default installation configuration all major browsers store a range of data that accumulates during their use. Most of this can be categorized as private to the user and should not be divulged. Browsers keep a history of visited sites, cached copies of accessed content and cookies with configuration and authentication information. As we will see in section 4 all of this information can in some way be accessed by determined attackers. The contents of the history file and the cache play an important role in a number of attacks. Both can be used to target phishing attempts at sites the user has visited, a practice known as *spear phishing* [2].

All security measures fail if the user can be convinced to install a malicious browser plugin or such a plugin is preinstalled (e.g. at a public terminal). Louw

[3] describes *BrowserSpy*, a Firefox plugin that was created in less than three weeks and can take complete control of the browser. It was possible to covertly harvest and transmit all form data, even if sent over encrypted connections, complete history data and all passwords in Firefox's built-in password manager. BrowserSpy can remain undetected because it hides itself in the Google Toolbar plugin. According to Louw there is currently no provision in Firefox to provide protection against malicious extensions and a comparable plugin called FormSpy has already been spotted on the Internet.

User tracking. An often overlooked risk where browsers readily play into the hands of the adversary are tracking cookies. Bennett [4] speaks of the *data trail* we all leave as we unwittingly engage in everyday and innocent activities. He sees this as an international problem where the private sector is as much (if not more) involved than the traditional 'Big Brother' government. According to Conti [5] it's only a matter of time before web sites and search engine collect enough information on a person to identify him/her. He reports that even high-profile incidents like the disclosure of the search queries by over 600,000 AOL users have not made the public aware of the dangers [6].

Most tracking cookies are set by third-party domains such as banner providers and visitor counters. As services like Doubleclick.net and Google Ads are used by a lot of sites users can be tracked all over the Internet. Also privacy-related information such as the geographic location, organization and through the **Referer** field search strings and request parameters like account numbers can be collected [7]. Google Analytics, which is included in the target site as JavaScript, can also register user interaction such as scrolling which according to Claypool has a strong positive relationship with explicit interest [8]. In the case where cookies are disabled browsers can still be tracked through the use of so called *cache cookies* [9].

2.2 The Browser as Trusted Party

The second role a browser can play in an attack is the trusted party. Trusted in this case means that the real target the adversary wants to attack in some sense trusts the browser. The trust may be established in a number of ways, two of which are possession and location. In the first case the browser contains a cookie with authentication information for a certain service. If an attacker can get the browser to access this service the connection is automatically authenticated and the attacker can use the service with the permissions of the user. It's therefore not even necessary to explicitly steal the cookie because the browser sends it to the service anyway. This attack is called *session hijacking* and can be accomplished via a cross-site scripting attack (see the definition in next section) [10].

Trust by location can be used to perform *IP hijacking*. In this case the attack is based on the fact that the IP address of the computer the compromised browser runs on is trusted by a certain services. Jackson [11] gives the example of the ACM digital library that can be freely accessed from certain educational

IP ranges. Other possible examples are: sending spam through the provider's SMTP server, performing click fraud by accessing advertisement banners and even performing illegal activities in order to frame the browser's user. The trust (or actually lack of mistrust) in the case of click fraud comes from the fact that it's not only one IP from which the banners are accessed but a lot of different, legitimate looking addresses if the attack is performed on a host of browsers.

2.3 The Browser as Attack Platform

In the third role the browser itself is used as a platform for attacks on third parties. Its programming capabilities with JavaScript and the network connection allow attackers to perform distributed denial of service (DDoS) attacks on arbitrary hosts. With so-called *puppetnets*, browsers that can be remotely instructed to attack, traffic volumes comparable to current botnets can be generated. It's also possible to use those puppetnets for worm propagation. With careful planning and planting of the necessary code on a high-profile site 90% of vulnerable computers can be infected in less than two minutes [12].

Because people often browse from behind firewalls their browsers can be used to reach into internal networks. Once inside the attacker can perform reconnaissance on future targets, use and spy on internal services and compromise machines. Especially home users with routers that still have the default factory password activated are vulnerable [11]. Johns also mentions that internal computers and intranet applications often remain unpatched and home-grown applications are rarely audited for security problems thoroughly [13].

3 Definitions

Some fundamental principles and techniques need to be defined in advance in order to be able to refer to them later.

3.1 Same Origin Policy (SOP)

The Same Origin Policy [14] is one of the main security features that seek to prevent access to private data. It states that documents or scripts loaded from one origin (i.e. website or domain respectively) cannot access properties (form data, cookies, JavaScript variables and functions) of a document from a different origin. In Mozilla the protocol (HTTP, HTTPS, FTP ...), the port (if specified) and the host have to match to be able to access another page's properties.

Therefore if a page is loaded from `http://www.a.com/b/c.html` it can access (and change) properties of the pages `http://www.a.com/b/m/n.html` and `http://www.a.com/r/s.jsp`. Access is blocked to `https://www.a.com/x.html`, `http://www.a.com:8080/y.html` and `https://store.a.com/z.html`.

In order to allow interaction among sites of the same company it's possible to set the value of the DOM element `document.domain` to a suffix of the current domain (e.g. `a.com`). Access to `https://store.a.com/z.html` is then

granted. In the case of Google this allows for a *single sign-on* to all its services (Mail, Calendar, Docs, ...) because authentication information can be shared. If, however, one service was compromised (for example through an XSS attack, see next section) all of the user's data in the other services could be accessed by the attacker.

3.2 Cross-site scripting (XSS)

Cross-site scripting attacks try to sneak malicious data into the browser's sandbox. They rely on a vulnerable, trusted site, that the user is accessing, to deliver this data. Because the malicious data (mostly JavaScript code) comes from the trusted site's web server the SOP doesn't restrict the access to form and authentication data used by this web application. It's therefore easily possible to steal cookies and private data or misuse the user's authenticated session [15]. Ways to perform cross-site scripting attacks are discussed in section 4.

3.3 Cross Site Request Forgery (CSRF/XSRF)

Cross Site Request Forgery (also known as session riding) is similar to cross-site scripting. It also depends on the attacker's ability to use a trusted site for delivering attack code. With XSRF, however, it's not necessary to send JavaScript code. An attacker might for example simply include an image that has its `src` attribute set to `http://www.bank.com/transfer?amount=1000&to=attacker`. The browser sends an HTTP request and existing authentication information (e.g. cookies) to this URL, thus accessing it with the privileges of the person that is currently using the attacked browser [13]. Online fora often allow linking to external images in posts, making it hard to prevent this type of attack.

4 Threats and Attacks

4.1 Browser History Sniffing

The browser's history data is one of the main assets to be guarded from attacks. It is, however, possible to glean information about whether the user has visited certain websites in the past. Both attacks described in this section don't even have to circumvent security measures in browsers because they take advantage of core web technologies making them hard detect and prevent.

CSS attribute attack. This attack takes advantage of the CSS `:visited` pseudo-class. As a suffix to CSS selectors resolving to links it can be used to apply an alternate styling when the user has already visited the page this link points to. Clover [16] describes one possibility to relay this information back to the attacker's website with code like in listing 1. In this case the link contains no text and is therefore not even visible to the user.

```
<a id="ebaylink" href="http://www.ebay.com"></a>

#ebaylink:visited {
    background: url(/visited.cgi?site=ebay);
}
```

Listing 1: HTML and CSS code for checking whether eBay was visited.

Clover also lists ways to distinguish between visited and non-visited links in JavaScript by reading the `currentStyle` property in Internet Explorer and the `offsetTop` property (combined with a `top` offset in CSS) in Mozilla. He mentions that this attack doesn't take advantage of a browser bug and therefore cannot be easily fixed. This view is reflected by Bug #147777 recorded in Mozilla's bug tracking system¹. Entered in 2002 and detailing this attack vector the discussion is still ongoing and the bug is not marked as fixed yet.

Jakobsson [2] sees the value of history information in the possibility to target phishing attempts at sites the user has visited. This targeted phishing is known as *spear phishing*. On his site² he has two demonstrations: one shows a list of links (banks, e-commerce, web-mail) and whether you have visited them; the other one shows the message "If I were a phisher, I would be glad to know you bank with:" and the logo of one of the e-banking sites you have visited. This technique could be used to create very convincing phishing sites that mimic your bank's page.

Timing attacks. In his seminal paper Felten [17] describes how to use the timing behaviour of browser requests to detect whether a user has visited a specific site. Depending on whether parts of the site (e.g. images) are in the *browser cache* or not the time to load them is different. A malicious Java applet or JavaScript can measure this difference with high accuracy (above 90%) and can further improve it by including known misses and hits as well as by combining multiple measurements.

Disabling Java and JavaScript doesn't prevent this attack. It's possible to load three files (e.g. images) in sequence: the first from the attacker's site, the second from the target and the third again from the attacker's site. Subtracting the time of the first and the third request his own site the attacker can once again measure the timing of the second request with sufficient accuracy.

There's also no use in disabling caching because the time needed to do DNS lookups can be measured in Java and JavaScript. As most machines keep a *DNS cache*, the timing is once again different for recently accessed sites. By distinguishing between local DNS caches and caching DNS servers access patterns in departments and companies can be derived. The same applies to caching web proxies for a group of people.

¹ https://bugzilla.mozilla.org/show_bug.cgi?id=147777

² <http://www.browser-recon.info>

The problem with timing attacks is (like with CSS attack) that they don't exploit a browser bug but use basic properties of web browsers. They can be performed without the victim's knowledge, hidden in web pages, forum posts, advertising banners and even HTML email messages.

Preventive measures. Jakobsson [18] proposed a way to guard against most CSS and timing history attacks. He distinguishes two kinds of URLs and presents measures to protect them from detection by attackers:

- *Entrance URLs*: links that a user types, has bookmarked or gets from a search engine to start accessing a site (e.g. <http://www.ebay.com>). To hide these URLs the cache is being *polluted* with a large number of similar pages. This makes it difficult for an attacker to know which of the sites was really accessed. This doesn't, however, work for stigmatized content because nobody would like to have lots of pornography and warez sites in his history.
- *Internal URLs*: those are to ones accessed when following links on entrance pages, logging in or searching (e.g. <http://www.ebay.com/profile.cgi>). They are much more useful to attackers because they show that the victim has indeed used the site. Those URLs are made impossible to guess by extending them with temporary (i.e. for the session) or long-term *pseudonyms*.

The system works by putting a filter in front of the web server whose URLs should be protected. On the first access of the browser to an external URL the filter generates a pseudorandom pseudonym and on sending back the page translates all URLs by appending this pseudonym. If an URL with a pseudonym is requested the filter either checks the HTTP-Referer if the user came from one of the domain's pages and the pseudonym is unchanged or it checks for the pseudonym value in a cookie. This is done to prevent attackers from linking to pages with fake pseudonyms which could again allow them to be identified in a browser cache. Search engines accesses are detected and for them temporary pseudonyms that get replaced immediately are used.

4.2 Cross-site scripting attacks

As already defined in section 3 a cross-site scripting attack uses a vulnerable web site to deliver malicious code to a browser. The difficulty in detecting and preventing XSS attacks according to Kirida [19] is that although these scripts are confined by the sand-boxing mechanisms and conform to the same-origin policy they still violate the security. He distinguishes two main classes of XSS attacks:

- *Stored attacks*: the malicious code is permanently stored on the target server (e.g. in its database, a message forum or a guestbook).
- *Reflected attack*: the code is embedded in a link to the vulnerable site and gets 'reflected' off the web server which incorporates this code in its response. The link may be delivered to the victims via email messages or embedded on other web pages.

Kirda [19] also gives an example for how a reflected attack could work. The user has to be tricked into clicking a link like `<a href="http://www.trusted.com/<script>document.location="http://www.evil.com/stealcookie?" + document.cookie;</script>">`. The web server on `www.trusted.com` won't find the requested page and will return an error message. If it includes the requested file name in the return message then the script will be sent to the user's browser and will be executed in the context of the `trusted.com` origin. A (authenticating) cookie set by `trusted.com` will be sent to the malicious web site as a parameter to the invocation of the `stealcookie` server-side script.

XSS attacks are the result of insufficient checking of user input in web applications and web servers. As more and better input checks are developed attackers create ever more sophisticated ways of hiding the script in the input. The web page [20] details over 100 possibilities, ranging from simple approaches like hiding the script in an `img` tag, through CSS `background` URLs containing JavaScript to sophisticated attacks like hiding the script in an embedded SVG literal or by using different encodings (e.g. UTF-8) that are missed by filters but rendered correctly by browsers.

4.3 Externally targeted attacks

According to Lam [12] current security models like the same-origin policy are focused mostly on protecting the browser and its host from malicious web servers and measures against attacks like cross-site scripting focus on protecting web servers from malicious browsers. He misses approaches which prevent attacks against third parties. In this section we will look at how browsers can be misused for such attacks.

Puppetnets. Puppetnets are comprised of browsers (*puppets*) that perform (malicious) activities and are doing so according to code sent to them by a web site. The code will be mostly JavaScript but it's also possible to perform attacks just by sending plain HTML. Lam [12] describes three different attacks:

- *Distributed Denial of Service:* The simplest way to flood a target site with requests is to include one of its images in a popular page. Through a JavaScript loop the load can be increased and attaching a unique parameter prevents caching. Browser restrictions on the number of simultaneous connections can be circumvented by using aliases of the server, stripping the `www` part or using its IP address. The amount of traffic generated is comparable to conventional botnets. A compromised web site of the top 100 most could command more than 100,000 browsers at any time.
- *Worm propagation:* Worms that infect web sites through URL-encoded exploits can be spread by puppetnets. After infecting a site the worm could add propagation code to its pages in order to make visiting browsers to new carriers of the worm. Because the browsers initiate the connection protective systems like firewalls or NAT can be penetrated as well.

- *Reconnaissance probes*: With puppetnets it's possible to scan for targets in stealth and even behind NAT and firewalls. By determining the response time of HTTP requests the puppetnet code can check for live web servers. Some browsers (e.g. Safari) even allow connections to other ports like FTP, SMTP or SSH, allowing to map services pretty accurately.

Puppetnets are hard to track and defend against because browsers all over the world join and leave as users browse the web. Filtering out DDoS traffic and attacks therefore becomes very hard. One problem (especially with reconnaissance probes) is that no data except liveness information can be relayed back to the attacking site. The other problem is that web sites need to be compromised in order to install puppetnet code. Both problems are solved by DNS rebinding, which will be described next.

DNS rebinding. DNS rebinding (as described by Jackson [11]) circumvents the same origin policy (SOP) by letting the browser think that two resources (the attacker's site and the target) are from the same origin. Once this succeeds data between the attacker and the target can be tunneled through the browser. DNS rebinding works by registering a domain name (e.g. `attacker.com`) and attracting web traffic, for example by running an advertisement.

For the first DNS query trying to resolve `attacker.com` the attacker sends the IP address of his own server and serves malicious JavaScript or Flash code. This code rebinds the domain name to the target's IP address, so that it can send data to the target as well as the attacker and doesn't violate the SOP. This can be done by having multiple `A` records in DNS or by giving the attacker's IP a very short `TTL` and dynamically answering with the target's IP on the second request.

Browsers try to prevent this attack with a technique called *DNS pinning*, which caches the IP of a host for some time and uses it regardless of `TTL`. This, however, can be circumvented in all current browsers by convincing the browser that the current IP is unreachable, thus forcing it to resolve to the second one. Often plugins like Flash or Java maintain separate pinning caches and can be used instead for the actual attack. Their additional advantage is the possibility to connect to arbitrary ports on the target's IP address.

Attacks have already been described in section 2. What remains to be discussed are some possible defenses: firewalls could forbid external host names from resolving to internal IP addresses, thus preventing attacks in the internal network. Pinning is difficult to fix because browser vendors don't want to break fail-over schemes of sites that have multiple servers. It's, however, possible to restrict the range of IPs, a domain is allowed to resolve to, to a small subnet or have a restrictive rebinding policy (in the reverse DNS record or on the web server) which the browser has to check before failing over. In his comprehensive paper Jackson [11]) describes a large number of possible solutions and their implications.

5 Discussion

In this paper we have seen the possibilities for spying, hijacking and attacking that browser inadvertently offer. One of the main problem is, that a lot of these techniques work because of basic functionalities and not because of some bug. It may therefore be advisable to rethink some of the core technologies like DNS, HTTP, HTML and JavaScript. Encryption, authentication and trust measures could go a long way to thwart current attacks.

Another problem is, that most of these techniques can be applied in stealth without the user noticing. It is therefore the job of browser vendors to better secure their products and provide visible and understandable warnings. Users also need to be told about the dangers on the Internet and how to avoid them. In the real world we know exactly which parts of the city not to go to and not to trust strangers. This instinctive feeling for danger is at the moment not very pronounced when surfing the web. People will need to enhance their knowledge and hone their instincts in this respect.

Finally, as is mostly the case with security-related topics, we have an ongoing arms race between researchers and attackers. In order to gain an advantage over the ‘bad guys’ we need to put more effort into educating developers and administrators about the dangers and pitfalls to watch out for when developing for the web. The overview on state-of-the-art attacks given in this should contribute to this goal.

References

1. Ye, Z.E., Smith, S., Anthony, D.: Trusted Paths for Browsers. *ACM Transactions on Information and System Security* **8**(2) (2005) 153–186
2. Jakobsson, M., Stamm, S.: Invasive Browser Sniffing and Countermeasures. In: *WWW '06: Proceedings of the 15th international conference on World Wide Web*, New York, NY, USA, ACM (2006) 523–532
3. Louw, M.T., Lim, J.S., Venkatakrishnan, V.: Extensible Web Browser Security. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Volume 4579 of *Lecture Notes in Computer Science*. Springer, Berlin / Heidelberg (September 2007) 1–19
4. Bennett, C.J.: Cookies, web bugs, webcams and cue cats: Patterns of surveillance on the World Wide Web. *Ethics and Information Technology* **3**(3) (2001) 195–208
5. Conti, G.: Googling considered harmful. In: *NSPW '06: Proceedings of the 2006 workshop on New security paradigms*, New York, NY, USA, ACM (2006) 67–76
6. Conti, G.: Could Googling take down a President? *Communications of the ACM* **51**(1) (2008) 71–73
7. Krishnamurthy, B., Malandrino, D., Wills, C.E.: Measuring Privacy Loss and the Impact of Privacy Protection in Web Browsing. In: *SOUPS '07: Proceedings of the 3rd symposium on Usable privacy and security*, New York, NY, USA, ACM (2007) 52–63
8. Claypool, M., Brown, D., Le, P., Waseda, M.: Inferring User Interest. *IEEE Internet Computing* **5**(6) (2001) 32–39

9. Juels, A., Jakobsson, M., Jagatic, T.N.: Cache Cookies for Browser Authentication. In: SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (2006) 301–305
10. Johns, M.: SessionSafe: Implementing XSS Immune Session Handling. In Gollmann, D., Meier, J., Sabelfeld, A., eds.: Computer Security – ESORICS 2006, 11th European Symposium on Research in Computer Security in Lecture Notes in Computer Science. Volume 4189., Springer (2006) 444–460
11. Jackson, C., Barth, A., Bortz, A., Shao, W., Boneh, D.: Protecting Browsers from DNS Rebinding Attacks. In: CCS '07: Proceedings of the 14th ACM conference on Computer and Communications Security, New York, NY, USA, ACM (2007) 421–431
12. Lam, V.T., Antonatos, S., Akritidis, P., Anagnostakis, K.G.: Puppetnets: misusing web browsers as a distributed attack infrastructure. In: CCS '06: Proceedings of the 13th ACM conference on Computer and Communications Security, New York, NY, USA, ACM (2006) 221–234
13. Johns, M.: A first approach to counter JavaScript malware. In: Proceedings of the 23rd Chaos Communication Congress, Berlin, Deutschland, Chaos Computer Club (2006)
14. Ruderman, J.: The Same Origin Policy. <http://www.mozilla.org/projects/security/components/same-origin.html> (August 2001) accessed on 25th of May 2008.
15. Endler, D.: The Evolution of Cross-Site Scripting Attacks. White paper, iDEFENSE Labs, Chantilly, VA, USA (May 2002)
16. Clover, A.: Timing Attacks on Web Privacy (Paper and Specific Issue). <http://www.securiteam.com/securityreviews/5GP020A6LG.html> (February 2002) accessed on May 25th 2008.
17. Felten, E.W., Schneider, M.A.: Timing Attacks on Web Privacy. In: CCS '00: Proceedings of the 7th ACM conference on Computer and Communications Security, New York, NY, USA, ACM (2000) 25–32
18. Jakobsson, M., Stamm, S.: Web Camouflage: Protecting your clients from browser-sniffing attacks. *IEEE Security and Privacy* **5**(6) (2007) 16–24
19. Kirda, E., Kruegel, C., Vigna, G., Jovanovic, N.: Noxes: a client-side solution for mitigating cross-site scripting attacks. In: SAC '06: Proceedings of the 2006 ACM symposium on Applied computing, New York, NY, USA, ACM (2006) 330–337
20. Hansen, R.: XSS (Cross Site Scripting) Cheat Sheet. <http://ha.ckers.org/xss.html> accessed on 25th of May 2008.