FireDrill: Interactive DNS Rebinding

Yunxing Dai, Ryan Resig
Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48109
{yunxing, rresig}@umich.edu

ABSTRACT

By using traditional DNS rebinding attacks, an attacker is able to circumvent firewalls in order to access internal network servers. Although many of the variations of this attack are well-known and sufficiently defended against, we show that by exploiting browsers' DNS cache table, it is possible to launch a DNS rebinding attack on modern browsers. Furthermore, we implement FireDrill, a tool that uses this DNS cache flooding technique to initialize an interactive session between the attacker and victim's web server. This interactive session opens up a number of malicious possibilities for the attacker on top of existing DNS rebinding uses. Some of the new potential uses include authentication, modification of website state, framing of the victim, and more.

Categories and Subject Descriptors

C.2.0 [**COMPUTER-COMMUNICATION NETWORKS**]: Security and Protection

General Terms

Security, Design, Experimentation

Keywords

DNS, DNS rebinding, Firewall, Network security, Same-origin policy

1. INTRODUCTION

DNS rebinding attacks circumvent the same-origin policy[1, 2] of web browsers. The attack confuses the victim's browser, causing it to pool two distinct entities into one origin. This allows the attacker to circumvent firewalls, scan internal networks, access and infiltrate private nodes on the network, uncover sensitive information, and even convert victim browsers into open network proxies.

A DNS rebinding attack is particularly powerful because it is easy to initiate and has a high impact once open access is established. In order to initiate the attack, an attacker merely needs to drive traffic to his page. This could be through advertisements, spam emails, or social engineering. Once the victim begins connecting to the attacker's web server, the browser is quickly compromised and the attacker has open access to the victim's internal network using the victim's IP.

In a traditional DNS rebinding attack, the attacker would set up a DNS server which answers queries to his own website. The query responses would have a short time-to-live (TTL). The attacker's web server would send malicious JavaScript to the user, which would then attempt to send a request back to the server after the TTL has expired. The subsequent DNS lookup would rebind the host name to the target server's IP address, thus placing both the victim's web server and the attacker's web server under the same origin. In its simplest form, this attack will then gather as much data from the webserver as it can via HTTP requests and then exfiltrate that data back to the attacker's web server, as shown in Figure 1.

A common defense against the traditional attack is DNS pinning[3]. With DNS pinning, the browser will cache the result of the DNS lookup for a relatively long period of time regardless of the response's TTL. This defense is not entirely effective though, as browser plug-ins generally maintain separate DNS entry databases. Such *multi-pin* vulnerabilities are the result of each plug-in mapping to a different IP address, and then communicating with one another in order to execute the attack[4]. However, many multi-pin vulnerabilities

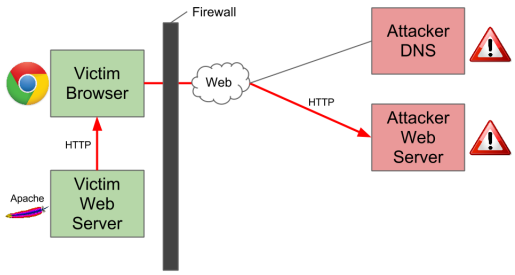


Figure 1: **Traditional DNS Rebinding attack.** Once the victim's browser has established connection to both servers, it can relay data from the internal server back to the attacker's server. The attacker can use this to gain access to private information stored on the victim's intranet.

have been closed as well by the developers of the plug-ins. Moreover, the web is developing towards way that controls the permissions of plug-ins.

Our main work focuses on executing a DNS rebinding attack by flooding the DNS cache on the victim's browser. We flood the cache with invalid entries in order to force the browser to do the vital second DNS lookup. In order to demonstrate this exploit, we implement FireDrill, a tool that uses this vulnerability to initialize a fully interactive session between the attacker and the victim's web server.

The rest of the paper is organized as follows: Section 2 discusses related work on DNS rebinding. Section 3 introduces the cache flooding exploit and outlines our implementation of FireDrill. Section 4 evaluates this technique against alternative approaches. Section 5 discusses defenses and future work. Section 6 concludes.

2. RELATED WORK

Jackson et al. [5] surveyed a number of previously undiscovered DNS rebinding attacks that exploit interactions between browsers and their plug-ins. Many of the attack vectors described in this paper have been closed since its publication. Their work outlines the possibility of using DNS rebinding not only for connecting to otherwise inaccessible services, but also for accessing public services using the victim's IP address. Once the attacker has hijacked the victim's IP address, he can execute a number of attacks including committing click fraud, sending spam, defeating IP-based authentication, and framing the victim. Each of these has important ramifications, but are all outside the scope of our work.

Other tools have been created which take different approaches to DNS rebinding, and have different intended uses. One tool, called Rebind [6], implements the multiple A record DNS rebinding attack. However, since the multiple A record attack is only possible when all the records are public IP addresses, this kind of attack cannot be used on local addresses. The author worked within this limitation, and made the target of the attack the victim's router's public IP address. This attack vector relies on exploiting default passwords on the router hardware, and the frequency with which the default credentials are left unchanged. Our approach does not require using only public IP addresses because at its root, our approach is not a multiple A record attack, it is a time-varying attack. We are able to gain access to the entire intranet via binding to local IP addresses.

Byrne also demonstrated how to turn a victim's browser into a web proxy using a standard time-varying and plug-in attack [7]. However, those attacks have their limitations: standard time-varying attacks potentially require several minutes to complete due to DNS pinning. Our approach accomplishes a similar result, while requiring only a fraction of the time. The vulnerabilities that enable a plug-in attack have been mostly fixed, and thus require the user to have an old version of a browser plug-in installed, such as Java or Flash Player. Such vulnerabilities have been patched out of most if not all modern versions of the plug-ins.

Finding web servers on the victim's intranet is a well-solved problem. It has been demonstrated by scanning IP addresses in JavaScript and monitoring responses[8], and various host-name-guessing techniques[5]. Thus, it is not a focus of this work.

3. IMPLEMENTATION

Our approach to the DNS rebinding attack is derived from a standard time-varying attack, which can potentially take several minutes based on browser implementation of DNS Pinning. We discovered a previously undocumented variation which takes on the scale of tens of seconds. Instead of waiting for the pinned entries to expire, we flood the DNS cache with enough invalid entries to remove valid entries from the list. We built on this idea and provided the attacker with a seamless browsing experience on the victim's internal server, as shown in Figure 2. The next step is to retrieve the data from the victim's server (similar to existing scraping methods). Then we will allow the attacker to click links, take actions, and submit forms by sending the data to the victim's browser,which is acting as a proxy. The JavaScript on the browser then forwards the appropriate request to the server.

3.1 Malicious DNS server

Our attack scenario consists of a custom DNS server authoritative for an attacker controlled domain name: attacker.com. The DNS server keeps track of DNS requests and their source IP address. When the DNS server sees a request, it checks: 1) If it is the first time

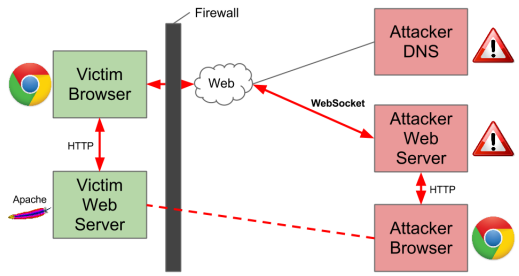


Figure 2: **FireDrill Attack Overview.** When a victim accesses the attacker's web server, a malicious Javascript payload is delivered and runs on the victim's browser. It then issues a large batch of DNS request to flood victim's DNS table and rebind the original domain name to the IP address of victim's web server. The victim's browser then becomes a proxy between the internal websites and the attacker's browser. The attacker can navigate it as he would in any other website.

the server sees the request, it returns a IP address denotes as **IP(attacker.com)**, which is the address of the attacker's server that provide web content including html page and Javascript payload. 2) If the DNS server has seen the DNS requests from the same IP address twice, it knows that this DNS request is initiated by the rebinding Javascript that runs on victim's browser. At this time, the the malicious DNS server will return an attacker-specified IP address, typically an internal IP, denotes as **IP(victim.com)**.

3.2 Rebinding based on DNS table flooding

Modern browsers will pin a DNS entry (*www.attacker.com to IP(www.attacker.com)*) it sees for a period of time, during this time, no other DNS entry with the same domain name will get accepted[3]. To remove a pinned entry from the DNS entry table, we use a DNS flooding technique. In current implementations of Chrome, all the domain names in the same level have the same priority. For this reason, we set our malicious URL (which the victim must request) to **www.attacker.com**.

Our malicious Javascript code then flood the DNS table by sending out 120 DNS resolving requests, from **n1.takenoteswith.us** to **n120.takenoteswith.us**. In this case, we assume the browser's DNS table size is 100, which is the default size in Chrome 25(Chrome 26 increases the cache size to 1000, which we will discuss later). The number of invalid DNS requests are slightly more than 100 here because we want to speed up the process by eliminating the tail effects that some DNS resolutions can take a long time. After the DNS entry has been evicted by the DNS flooding, the malicious Javascript code will ask the content of **www.attacker.com/index.html**. Since the browser can't find the DNS entry in its cache(it has already been evicted), it will ask the malicious DNS server to resolve IP of *www.attacker.com* again, which will then return a different IP address, **IP(www.victim.com)**.

Our Malicious DNS server is written in Python2.7. It keeps a big cache of the IP of DNS requests' initiators. When a request comes to our DNS server, if the IP of initiator is in the cache, the server knows the record is initiated by malicious Javascript and return the malicious IP. Otherwise, it will just insert the entry into the cache. Each cache entry has a expiration time set to 5 minutes to be able to relaunch the attack when the victim connect to the server again.

The careful readers may notice that the DNS requests **n*.takenotes.us** will not complete since they disobeys the same-origin policy. However, we found that Chrome will still insert the invalid entries into the DNS cache table and treat them with same priority.

3.3 Malicious Javascript proxy

A malicious Javascript proxy will be running on the victim's browser. It maintains a WebSocket connection to the attacker's webserver and receives proxy commands from it in the format of JSON. The commands from the webserver have three fields, the **method** field is used to specify whether an HTTP *post* or *get* request should be forwarded. The **url** field specify the target of the request. The **args** field contains the arguments of the request.

The response from the JavaScript proxy to the server must take additional step to maintain data integrity. Apart from plain HTML, sometimes the attacker wants to access binary data such as image and audio files. In this case, the proxy has to put the HTTP headers('content-type', specifically) into the response so that the attacker's browser knows how to parse and encode it. If the response is compressed, the JavaScript will be responsible to decompress it. In this case, the HTTP header field 'content-length' should be changed accordingly. Lastly, if the response contains binary data, the JavaScript must encode it using base64 so that it can be transmitted in a JSON object.

For instance, if the attacker is asking to submit a *post* request to */form/login* with arguments {*name=alice*}, it will send a JSON command to the victim's browser {*method:'post', url:/form/login', args:name='alice'*}. The proxy will then contrust a XMLHTTPRequest object based on it, fetch the content of response, and pass it back to be displayed on attacker's browser. Note that since we are using relative paths, we don't need to translate the links and forms here.

3.4 Attacker's interface

The attacker's interface is developed to give the attacker the ability to get notified when a new victim

clicks the malicious link and to switch between multiple victims.

When a victim's browser is connected to attacker's website, it immediately creates a WebSocket connection to the session server, which is responsible for creating a new session object to handle all the interactions between the victim's browser and the attacker. The attacker will then be notified via both web interface and e-mail that a new victim is connected. After the attacker selects an interactive session, he can then browse it using his browser as we would with regular websites.

When the attacker's browser requests a web object from the victim's intranet, it sends a request to his web server which is connected to the victim's browser via the persistent WebSocket connection. The web server will redirect the request to corresponding session object, which will spawn a thread to handle the request. The thread will then forward the request again to the JavaScript proxy running on victim's browser, sleep and wait for the response and wake up again when the response from the proxy is available. The thread then decodes the response, rebuilds the HTTP header, and forwards it back to the attacker's browser, thus completing the request.

4. EVALUATION

We measured our DNS rebinding attack by two primary factors. We analyzed the *time-to-launch* and the *impact* of the attack. We then compared it to other existing DNS rebinding techniques. The results are shown in Table 1.

4.1 Configuration

We tested DNS rebinding experimentally by registering a malicious domain name *takenoteswith.us* and running our framework on an Amazon EC2 instance that runs on Ubuntu 12.04. The client side experiment is runnig on OS X 10.8.3 and Chrome 26.0.1410.65. For victim's server, we set up an internal wiki using Tiki Wiki that hosted on victim's machine. We also configured firewalls and Apache filtering rules so that it can only be accessed through local connection. We will discuss other major browsers later.

4.2 Time-to-Launch

In order to protect from a time-varying attack, most modern browers have implemented DNS pinning technologies [3] that locks a domain name to a IP address in the first DNS response. At this time, a time-varying attack would take 160 seconds to launch according to our experiment on the latest Chrome browser. However, by flooding the DNS cache, we found that in the current Chrome implementation, if a DNS record is evicted from the cache, the pinning time would be nullified as the entry no longer exists. We found that in our attack, only 10 seconds are needed to launch the attack on a browser that has a cache size of 100 entries.

In early 2013, the Chromium community has increased the size of DNS cache from 100 to 1000. This was not the result of security concerns, but rather a performance related patch[9, 10]. We then ran our experiments on the staging version Chrome, and found that it would only take 10 more seconds to flood the DNS table and launch the attack.

Two other different approaches of DNS rebinding are multiple A-records attack and multi-pin attack. These attacks need only a small amount of time due to the small number of packets transmitted. However they all have certain limitations, which we will discuss it in the next subsection.

4.3 Impact

We now evaluate the impact of our attack against other DNS rebinding approaches. As mentioned in the last section, the multiple A record approach has the advantage of requiring less time to launch. However, it also has several limitations on its impact. First, the rebound IP address cannot be an internal IP address. Otherwise, the browser will prioritize it and select it in the first place which results in a failure to execute DNS rebinding. Second, the attacker cannot change the rebound IP address on the fly, which makes it unable to scan the subnet.

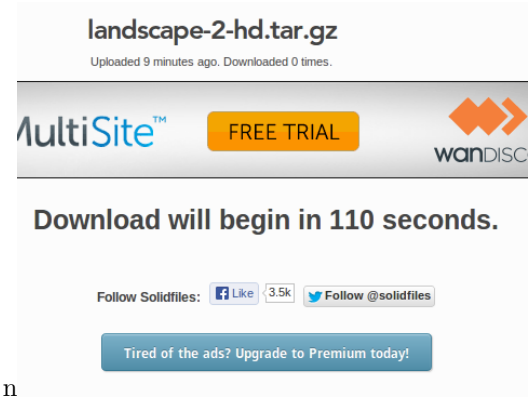
Multi-pin attack can be both fast and able to access IP address from intranet. However, it is actually based on browser's plug-in support such as Abode Flash[11]. Most of the vulnerabilities have been fixed years ago by developers and browser plug-ins are getting more restrictions on which permissions they could have.

For time-varying attack, although it is possible to bind to an internal IP address, it is also hard to change the rebound IP address on the fly due to the extremly long launching time.

In our experiement, we are able to use FireDrill to rebind the domain name to an internal IP address to build a interactive session. Also, we are able to dynamically change the IP address during an attack. The attacker has the ability to navigate through the entire intranet instead of just one single IP address.

4.4 Making The Victim Stay

Our attack establishes a fully interactive session, and as a result, requires the victim's browser to act as a proxy. Thus, it requires the victim to stay on the page for the attacker to have access to it. In order to do that, we designed a "pending download" page (shown in Figure 3) that attempts to convince the victim to stay on the page for two minutes to download a file. While the victim is waiting for the download countdown, the JavaScript proxy is actually running in the background



n

Figure 3: **Attack page.** The page that a victim first connects to. The user believes he is waiting for a file download to start on a free file-hosting website. While waiting for the file to be downloaded, the JavaScript is running in the background as a proxy.

for the attacker to navigate through the internal websites. While it is often challenging to convince a user to stay on a website for such a long duration, we show that some scenarios facilitate such a requirement.

4.5 Changing the Content of Internal Wiki

Now we demonstrate how to use FireDrill to access a victim's internal wiki. Many organizations have internal wikis that contain extremly sensitive information and is only accessible through the company's local network or VPN. By building a session using the employee's browser as a proxy, the attacker could not only gain full access to the company's wiki, but also the ability to change the contents of it. Moreover, the modifications are done using the victim's IP address, adding to the anonymity of the attack, as shown in Figure 4. In a real-world scenario, the attacker could potentially add malicious links into the wiki in order to launch subsequent attacks.

4.6 Other Browsers

We also tested our attack on Firefox 20 and Internet Explore 9 in order to measure the impact of our attack.

- *Firefox.* We found that Firefox is also vulnerable to this attack. Firefox doesn't have its own DNS resolution, thus it depends on Operating System to manage the DNS cache. In this case, flooding Firefox's DNS table actually floods the DNS cache in the OS. Since the OS has no knowledge of which DNS request a DNS cache entry is from, it makes the defense even harder.

- *Internet Explore 9.* Internet Explore is not vulnerable to our attack. We found that after a request disobeys same origin policy, Internet Explorer will not add it into the DNS cache. Although we managed to bypass the same origin policy by using a X-Domain request object from Internet Explorer, we still failed to evict the DNS entry we wanted to rebind. Due to the lack of development documentation and a close-source enviroment, we don't exactly know what defenses Internet Explorer has adopted. Some defenses from Internet Explore that we inferred and recommend other browsers to adopt are: 1) Building the browser's own DNS resolution that is independent to OS. 2) Using smart eviction policy to prevent an important entry to be evicted. 3) Pin a DNS entry for a relatively long time. We will discuss other defenses in the next section.

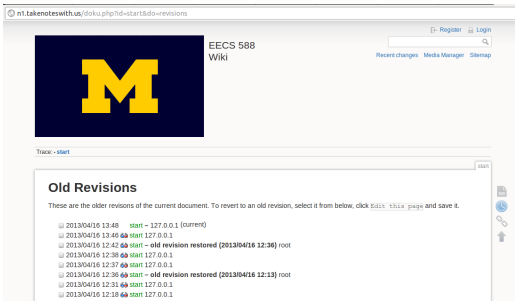


Figure 4: **Victim's web site's revision history.** The attacker can change the content of the company wiki anonymously. The revision history shows that the author of change was the victim's local IP address.

5. DISCUSSION

Many in the security community consider DNS rebinding attacks to be dead. However, we aimed to show in this work that there are ongoing developments in the area, and that DNS rebinding attacks are still possible on modern hardware and software configurations. Along with motivating further work on DNS rebinding, we hope to introduce some preliminary defenses against the particular techniques we proposed in this paper.

5.1 Defense Against DNS Rebinding

A significant amount of work has been done in the area to defend against DNS rebinding attacks at each stage of the process. Browsers, plug-ins, DNS resolvers, firewalls, and servers can all be augmented to help defend against the attack [5]. Many of the most promising defenses have been implemented, such as DNS pinning and patching many of the plug-in vulnerabilities.

5.2 Defense Against DNS Cache Flooding

DNS cache flooding is a new method of forcing the second DNS lookup which is crucial to the success of a DNS rebinding attack. We demonstrated that it is

	Time-to-Launch	Impact
Time-varying	$5 \sim 60$ minutes	Need javascript support
Multiple A-records	Instant	Unable to bind to IP addresses from intranet
Multi-pin	Instant	Need (old) plug-in support; Rebound IP can be changed on the fly
DNS-flooding	$10 \sim 20$ seconds	Need javascript support; Rebound IP can be changed on the fly

Table 1: A comparison between different DNS rebinding techniques. The DNS-flooding technology that we use is a trade-off between launch speed and impact.

possible to use this technique on modern browsers, but we believe a few simple provisions will be able to successfully defend against it.

- *Host Head Checking.* An server-side defense against DNS rebinding is to reject incoming HTTP requests with unmatched *Host* headers[12]. However, while most of the browsers have implemented the client part that containing hostname in the request header, a lot of servers don't implement it or don't turn it on by default. We deem that a more reliable defense would be on client side that is patched from the source of the attack: Browsers.
- *Increasing Cache Size.* Making the browser cache large enough that cache flooding takes prohibitively long is a very sensible approach. However, it is not clear whether this will prevent this attack in its entirety, but rather make it impractical. There are also performance concerns involved with scaling up the DNS cache that should be taken into account before adopting this approach.
- *Smarter Cache Eviction.* Cache flooding is made possible by the fact that invalid entries are being inserted into the table, which evict valid entries. The entries are invalid because the requests disobey same-origin policy, so an attempt resolve them should not occur. If the browser insists on inserting these invalid entries to the DNS cache, they should at least be the first to be evicted when the cache is full.

5.3 Future Work

FireDrill brings together many existing and novel ideas in order to demonstrate a very powerful DNS rebinding attack. Ensuring that the malicious DNS, web server, attacker interface, and other pieces are working in unison is a complex task. Automating the process of launching these utilities and monitoring for potential victims could reveal some opportunities to improve the efficiency and impact of the attack as a whole, which is important given the rigid time requirement of the attack.

Many of the original attack vectors of DNS rebinding achieved nearly instantaneous execution, but have since been closed and patched. At this point, we are able to achieve a DNS rebinding attack on modern browsers in about ten seconds. While this is an improvement over alternative approaches, it can still be a prohibitively long duration to wait. The defense strategy for DNS rebinding focuses on preventing the browser from doing a second DNS lookup. Exploring new ways of circumventing defenses could lead to a new, faster form of the attack.

We have outlined three promising defenses to the DNS cache flooding approach. Both fixes rely on browser developers to change DNS cache behavior. While we believe the best approach to be defending at the source, the cache, it is possible that a proper defense to this technique could be employed elsewhere in the configuration.

6. CONCLUSIONS

An attacker can implement a DNS rebinding attack to circumvent firewalls and confuse the browser into breaking the same-origin policy. While many existing approaches towards exploiting DNS rebinding vulnerabilities have been fixed, many new vulnerabilities are still being discovered. Existing defenses attempt protect against specific attack vectors, but do not prevent DNS rebinding attacks as a whole. These attacks are highly cost effective, relatively quick to execute, and are capable of doing severe damage to both the victim and the intranet to which he is connected. The ability to interactively communicate with the otherwise inaccessible server gives the attacker even more power. The attacker can hole punching into firewall, scan internal networks, access and infiltrate private nodes on the network, uncover sensitive information, modify the state of web pages under the IP address of the victim, login and authenticate as another user, and hijack the victim's IP address for use in a botnet.

DNS rebinding attacks have been around for more than 15 years[13], many defenses have been presented in previous work for preventing traditional DNS rebinding attacks but the threat hasn't been completely removed. We present possible defenses against the DNS cache flooding technique we introduced in this paper. Increasing the cache size can help make the attack prohibitively impractical to execute, while smarter cache eviction could potentially eliminate this particular form

of DNS rebinding altogether. We believe that DNS rebinding is still a very important and dangerous exploit, and hope that future work in this area will explore new vulnerabilities.

7. ACKNOWLEDGMENTS

This paper is derived from our course project of Advanced Computer Security in University of Michigan. We would like to thank the lecturer, Professor J. Alex Halderman for his introduction to DNS rebinding that motivated our work and for his encouragement to publish our results. We would also like to thank Eric Wustrow and Zakir Durumeric for comments on earlier versions of this paper and all the classmates for the discussions and insightful suggestions.

8. REFERENCES

[1] J Ruderman. Same-origin policy. `http://www.mozilla.org/projects/security/components/same-origin.html`, 2013.

[2] C Jackson, A Bortz, D Boneh, and J Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th international conference on World Wide Web*, pages 737–744. ACM, 2006.

[3] C Matthies. Dns pinning explained. `http://christ1an.blogspot.com/2007/07/dns-pinning-explained.html`, 2007.

[4] K Anvil. Anti-dns pinning + socket in flash. `http://www.jumperz.net`, 2007.

[5] C Jackson, A Barth, A Bortz, W Shao, and D Boneh. Protecting browsers from dns rebinding attacks. `http:///crypto.stanford.edu/dns/dns-rebinding.pdf`, 2007.

[6] C Heffner. Remote attacks against soho routers. `http://media.blackhat.com/bh-us-10/whitepapers/Heffner/BlackHat-USA-2010-Heffner-How-to-Hack-Millions-of-Routers-wp.pdf`, 2010.

[7] D Byrne. Intranet invasion through anti-dns pinning. `https://www.blackhat.com/presentations/bh-usa-07/Byrne/Presentation/bh-usa-07-byrne.pdf`, 2007. Invited talk.

[8] J Grossman and T Niedzialkowski. Hacking intranet websites from the outside: Javascript malware just got a lot more dangerous. *Blackhat USA*, 2006.

[9] Issue 114277:hostcache of size 100 fills up very quickly with dnstransaction. `https://code.google.com/p/chromium/issues/detail?id=114277`.

[10] `http://src.chromium.org/viewvc/chrome/trunk/src/net/dns/host_cache.cc`.

[11] Adobe. Adobe flash player 9 security. `http://www.adobe.com/devnet/flashplayer/articles/flash_player_9_security.pdf`, 2006.

[12] D Ross. Notes on dns pinning. `http://blogs.msdn.com/dross/archive/2007/07/09/notes-on-dns-pinning.aspx`, 2007.

[13] D Dean, E Felten, and D Wallach. Java security: From hotjava to netscape and beyond. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 190–200. IEEE, 1996.