NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

DNS REBINDING ATTACKS

by

Georgios Kokkinopoulos

September 2009

Thesis Co-Advisors: Geoffrey G. Xie
John H. Gibson

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2009	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: DNS Rebinding Attacks	**5. FUNDING NUMBERS**
6. AUTHOR(S) Georgios Kokkinopoulos	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

A Domain Name System (DNS) Rebinding attack compromises the integrity of name resolution in DNS with the goal of controlling the IP address of the host to which the victim ultimately connects. The same origin policy and DNS Pinning techniques were introduced to protect Web browsers from DNS rebinding attacks, but their effectiveness has been undermined by vulnerabilities introduced by plug-ins such as JavaScript and Adobe Flash Player. The new attacks fall into two broad categories: firewall circumvention and IP hijacking, depending on the consequences of each attack.

Using a realistic network testbed, this research has enacted two firewall circumvention attack scenarios, with JavaScript and Adobe Flash Player respectively. Also confirmed is the effectiveness of several published countermeasures, including configuration options for DNS and Web servers, and security updates released by plug-in vendors. Finally, the research analyzes the defense-readiness of the DNS server and client configuration guidelines used by the U.S. Department of Defense (DoD), including the Defense Information Systems Agency (DISA) DNS Security Technical Implementation Guidance (STIG), the Windows Vista Client Specialized Security Limited Functionality (SSLF) Guidance, and the split-DNS architecture.

14. SUBJECT TERMS network security, DNS rebinding, DNS pinning, same origin policy, anti DNS pinning, adobe security updates, DNS STIG, Windows Vista security, split DNS	**15. NUMBER OF PAGES** 129
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

DNS REBINDING ATTACKS

Georgios Kokkinopoulos
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2009

Author:	Georgios Kokkinopoulos
Approved by:	Geoffrey G. Xie, PhD Thesis Co-Advisor
	John H. Gibson Thesis Co-Advisor
	Peter J. Denning, PhD Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A Domain Name System (DNS) Rebinding attack compromises the integrity of name resolution in DNS with the goal of controlling the IP address of the host to which the victim ultimately connects. The same origin policy and DNS Pinning techniques were introduced to protect Web browsers from DNS rebinding attacks, but their effectiveness has been undermined by vulnerabilities introduced by plug-ins such as JavaScript and Adobe Flash Player. The new attacks fall into two broad categories: firewall circumvention and IP hijacking, depending on the consequences of each attack.

Using a realistic network testbed, this research has enacted two firewall circumvention attack scenarios, with JavaScript and Adobe Flash Player respectively. Also confirmed is the effectiveness of several published countermeasures, including configuration options for DNS and Web servers, and security updates released by plug-in vendors. Finally, the research analyzes the defense-readiness of the DNS server and client configuration guidelines used by the U.S. Department of Defense (DoD), including the Defense Information Systems Agency (DISA) DNS Security Technical Implementation Guidance (STIG), the Windows Vista Client Specialized Security Limited Functionality (SSLF) Guidance, and the split-DNS architecture.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

It is a pleasure to thank those who made this Thesis possible.

I would like to thank my advisor, Professor Geoffrey G. Xie, who gave me the opportunity to work with such an interesting subject in Network Security. His support, guidance, and encouragement helped me significantly to complete my research and made this Thesis process a lifetime learning experience. Thanks also to my co-advisor, John H. Gibson, who was always willing to discuss and advise me during the whole process. They both provided me with all necessary means, but most of all they taught me how to study in depth and discover solutions.

I thank the Hellenic Navy for giving me the opportunity to study at the Naval Postgraduate School.

I thank, from the bottom of my heart, my precious children, my daughter Marousa and my son Nicolas, for their understanding, support, and love, during my absence from our home for such a long period.

I save the most important thanks to my wonderful wife, Tota, for her uncomplaining tremendous efforts to keep our family on track while I was, once more, far away from home. Her dedication, love, and patience never stop reminding me that I am blessed to have her by my side.

This Thesis is dedicated with endless love and respect to you, my beloved wife.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Modern Web browsers use the Same Origin Policy (SOP) to separate contents of trusted Web sites from other, potentially malicious ones. The origin of a Web object is uniquely identified by the combination of the protocol, host IP address, and port number parts of the object's Uniform Resource Locator (URL). Web browsers enforce the SOP by preventing a document or script loaded from one origin from interacting with objects of a different origin. Without this protection, an attacker can easily inject a malicious Web link to violate the confidentiality or integrity of another Web page. Domain Name System (DNS) Rebinding attacks aim to violate the SOP of Web browsers, with the objective of either impersonating the hosts or using them as proxy servers to circumvent the firewall and launch various attacks against internal servers.

DNS rebinding attacks have been known, at least conceptually, since 1996. As a countermeasure, DNS Pinning for Web browsers was introduced to avoid any kind of DNS spoofing attacks facilitated by client-side code execution. DNS pinning forces a browser to use a single IP address for any given host. The Web browser "pins" the DNS response in its memory (cache) and as longs as the browser runs, it uses that cached IP address.

However, DNS pinning is not always effective because of vulnerabilities introduced by plug-ins such as JavaScript, Adobe Flash Player, and Java. These plug-ins provide additional functionalities to Web pages, but at the same time may permit an attacker to evade DNS pinning. The new DNS rebinding attacks fall into two broad categories: firewall circumvention and IP hijacking, depending on the consequences of each attack.

This research starts with an overview of the known DNS rebinding attack techniques and their consequences. Then, using a real network testbed, the research implements two firewall-circumvention attack scenarios against all major Web browsers.

The first attack scenario exploits a vulnerability of the round-robin DNS feature using JavaScript. The SOP of a victim Web browser is violated and the malicious JavaScript program is able to connect to a different host behind the firewall (e.g., an internal Web server). The same set of experiments also verifies that Host Header Checking is an effective defense method if the attack is directed toward an internal Web server. When host header checking is configured for the internal Web server, the server will reject the malicious requests because the "Host:" part of the HTTP header for these requests does not match the server's own host name.

The second scenario uses an Adobe Flash application embedded in a HTML page, with the networking part coded using the `Socket` class included in the Adobe Flash package. The results confirm the DNS rebinding vulnerabilities of earlier versions of the Adobe Flash Player plug-in and the effectiveness of a series of security updates released by the plug-in vendor.

Based on the insights gained from the experiments, the research analyzes the defenses that have been proposed in the open literature. The analysis is then applied to some of the guidelines currently used by the U.S. Department of Defense (DoD), including the Defense Information Systems Agency (DISA) DNS Security Technical Implementation Guidance (STIG), the Microsoft Windows Vista Client Specialized Security Limited Functionality (SSLF) Guidance, and the split-DNS architecture.

The rest of the thesis is organized as follows. Chapter II first presents a detailed explanation of the same origin policy and the DNS pinning technique, and then describes the DNS rebinding attack methods and their consequences. Chapter III describes the testbed network configuration and the software tools used for this research, the experiments that were carried out, and the analysis of the results. Chapter IV presents an analysis of the defenses that have been developed and applies the analysis to several U.S. DoD guidelines. Chapter V presents the conclusions and suggestions for future work. Finally, the appendices include the source files of the HTML, Java, and Adobe Flash application programs created for this research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. CHAPTER OVERVIEW

The purpose of this chapter is to provide the background for this research. It answers the questions of what DNS rebinding attacks are and what vulnerabilities they exploit. Since these attacks have been known for twelve years, the chapter describes their first implementations as well as how they have mutated and become more difficult to mitigate. It also explains the same origin policy and the concept of DNS pinning, which are strongly related to how such attacks are mitigated today. The chapter concludes with a brief enumeration of the potential damages inflicted by DNS rebinding attacks.

B. SAME ORIGIN POLICY

Browsers enforce the same origin policy in order to prevent a document or script loaded from one origin from interacting with a document loaded from a different origin, as shown in Figure 1.

The Uniform Resource Locator (URL) for a Web object contains four basic components [1]:

<protocol>: // <server-host> : <port-number> / <path>

The origin of a Web object is defined as the combination of the first three components of its URL [1]:

Origin = {<protocol>, <server-host>, <port-number>}

Same origin policy is implemented by Web browsers. Two objects are considered to belong to the same origin if and only if their URL contains the same protocol*;* host and port-number so that the above sets are identical.

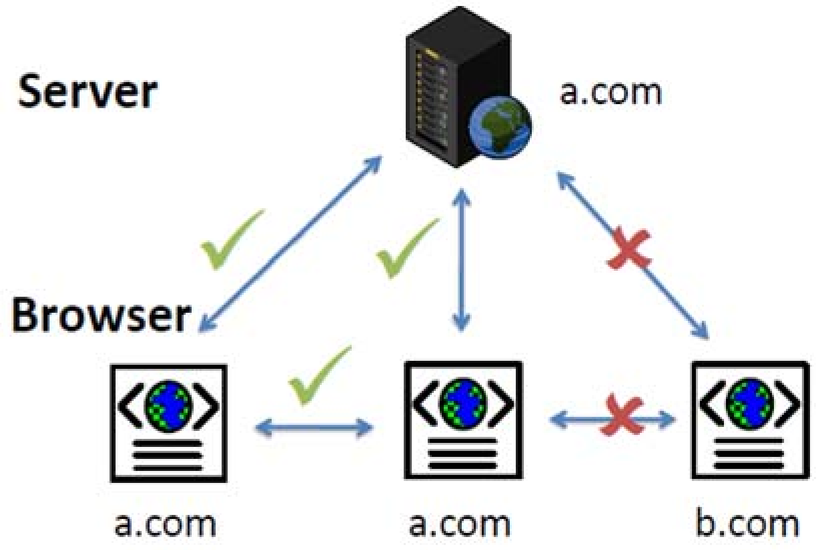


Figure 1. Same origin policy in action [From 2]

The same origin policy is also called Single Origin or Same Site Policy. It was originally released with Netscape Navigator 2.0 and is integrated into every major Web browser [1]. As an example, consider the following URL:

http:// networks.nps.edu / lab / index.html

Table 1 compares the above URL with several URLs in terms of origins, explaining the result:

URL	RESULT	REASON
http:// networks.nps.edu / lab / index.html	success	Same origin
https:// networks.nps.edu / lab / index.html	failure	Different protocol
http:// **cs4550.**networks.nps.edu / lab / index.html	failure	Different host
http:// networks.nps.edu**:21** / lab	failure	Different port-number

Table 1. Origin comparison example

However, this security policy is affected by various vulnerabilities because of design restrictions and because of trust established with other insecure services, such as the DNS. For example, Web browsers must translate the host name of the desired URL into an IP address, and then open a socket to that IP address. If the DNS responds with multiple IP addresses in one host name query, the browser will accept all of them as if they belong to the same origin, even if they are owned by different entities. DNS rebinding attacks exploit this vulnerability [3].

C. BASIC REBINDING ATTACKS

The network access security policy in Web browsers is based on host names, which are bound by the DNS to IP addresses. This policy can be undermined by a DNS rebinding attack, which binds the host name to two IP addresses. The one IP address belongs to an attacker and the other belongs to a target server. In this way, DNS rebinding attacks sabotage the same origin rule by confusing the browser into mixing content controlled by different entities into a particular security origin [3].

1. Princeton's Attack using Java Virtual Machine (JVM 1)

The first DNS rebinding attack was carried out by Princeton's Computer Science Department in 1996 [4]. It exploited the DNS server's capability of resolving multiple IP addresses for a single host name in conjunction with JVM's security policy of that time, which was vulnerable to the following scenario:

- Suppose that a user visits an attacker's Web page, http:// attacker.com. The attacker, who controls his Web and DNS servers, binds attacker.com to two IP addresses: his own Web server and the target's Web server. In addition, his Web page contains a Java applet.
- The victim's browser uses the first IP address to get the content of the page and runs the embedded applet, which requests that JVM open a socket to the second IP address, which corresponds to the target

server’s IP address. JVM permitted this connection; because the target’s IP address is contained in the DNS record for attacker.com.

The key to a successful attack is that even if the user (victim) and server (target) sit behind a firewall, that firewall is powerless to stop the attack, as illustrated in Figure 2. The firewall is supposed to guard the victim by preventing machines outside the firewall from opening random network connections to the LAN it protects. However, in this attack the dangerous network connections come from one of the LAN’s trusted internal machines, so the firewall is useless. To summarize, the attacker uses the victim's Web browser to attack the victim's cooperating machines [5].

JVM is no longer vulnerable to this attack because the Java security policy has been improved so that applets are limited to connecting only to the IP address from which they were loaded [3].

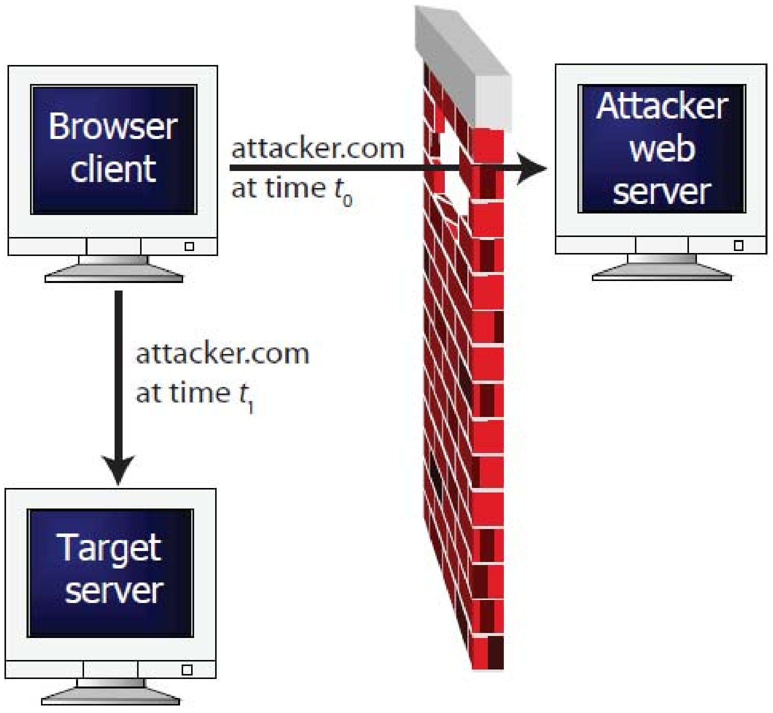


Figure 2. Basic DNS rebinding attack [From 3]

2. Low TTL Version Attack

In 2001, research came up with another attack scenario that was based on the original idea of the attack described above [6], [7]. This time the attack used the Time-To-Live (TTL) value of DNS records, which can be configured from a DNS server according to an attacker's wish. Additionally, the method used the `XmlHttpRequest` object, which can be used by scripts to connect to their originating server via HTTP. It can be used inside a Web browser scripting language, such as JavaScript, to send an HTTP request directly to a Web server and load the server response data directly back into the scripting language.

The technique was characterized as Time-Varying DNS and the process contained the following steps: [3]

- A user visits the malicious Web site. An attacker's DNS server responds with the normal IP address of the attacker's Web page, but the TTL is set to be zero or very low.
- The user retrieves a Web page containing a malicious script that contains an `XmlHttpRequest` pointing back to the attacker's page.
- Due to low the TTL, the DNS entry eventually expires and the victim's browser issues a new query for the attacker's site, which now resolves to the IP address of the target's server.
- The victim's browser is cheated now, and connects to the target server, which it thinks is the attacker's. The victim does not recognize that the `XmlHttpRequest` is receiving data from the target Web server instead of the attacker's, and the data leads to the attacker's server.

The connection in the last step has the same host name as the original malicious script, so the victim's browser thinks that the connection is same origin policy authenticated, while in reality the authentication is confounded and permits the attacker to read the response from the target server. As in the previous attack scenario, the attacker uses the victim's Web browser to attack the victim's cooperating machines. Another key point about this attack is the exploitation of `XmlHttpRequest`, which, although it can be used only with the origin server, uses DNS rebinding to cause the victim's server to be considered as having the same origin, breaking the same origin policy [8].

The above attack scenarios have the common purpose of confusing the browser, so that by breaking the same origin policy the attacker gets access to an internal server using the victim's browser as an insider, without the victim knows it. The DNS has not been disrupted with these attacks, as all the DSN responses are authoritative and normally provided by an attacker who owns the attacker.com domain name. Thus, DNSSEC is not useful against these types of attacks [3].

D. DNS PINNING

The first DNS rebinding attacks proved that same origin policy was not enough to defend against them. DNS pinning was discovered and applied to all modern Web browsers in order to provide protection against the exploitation of same origin policy vulnerabilities and the DNS's lack of security.

With DNS pinning, the browser keeps in memory (cache) the mapping of the IP address to a host name that comes after a DNS response, regardless the TTL value of the record, until the browser is closed. More specifically, DNS pinning forces the browser to put in cache the first DNS response for a host name and it does not allow additional queries. In other words, browsers keep a context database in their cache, "pinning" host names to IP addresses to prevent the association of one host name to many IP addresses, which is an action that the DNS permits.

Consequently, DNS pinning protects browsers against the latter attack scenario with a low TTL, because if the browser issues a second DNS query for a host name that already has been resolved—hence stored in its pinned database—the attacker's response with the different IP address for the same host name will be rejected from the browser and the scenario will fail [8], [9].

Unfortunately, researchers have observed significant DNS pinning vulnerabilities. In 2006, it was documented that Web browsers do not fully apply DNS pinning [10]. Further, it was observed that DNS pinning only works in the case that the Web server in the query is actually online and available. This makes sense because if the server appears somehow to be offline, a new DNS query is necessary to find out whether it has changed or moved in some way. Thus, an attacker can shut down his server whenever he wants to circumvent the user browser's DNS pinning [9].

Moreover, researchers from the Web Security Team of Stanford University identified specific DNS pinning weaknesses for several Web browsers [3]. According to that research:

- *Internet Explorer 7* pins DNS bindings for 30 minutes. Unfortunately, if the attacker's domain has multiple "A" records and the current server becomes unavailable, the browser will try a different IP address within one second.

- *Internet Explorer 6* also pins DNS bindings for 30 minutes, but an attacker can cause the browser to release its pin after one second by forcing a connection to the current IP address to fail, for example by including the element <*img src*= *"http: //attacker.com:81/"*>.

- *Firefox 1.5 and 2* cache DNS entries for between 60 and 120 seconds. DNS entries expire when the value of the current minute increments twice. Using JavaScript, the attacker can read the user's clock and compute when the pin will expire. Using multiple "A" records, an attacker can further reduce this time to one second.

- *Opera 9* behaves similarly to Internet Explorer 6. In [the Stanford security team's] experiments, [it was] found that it pins for approximately 12 minutes but can be tricked into releasing its pin after 4 seconds by connecting to a closed port.
- *Safari 2* pins DNS bindings for one second. Because the pinning time is so low, the attacker may need to send a *Connection: close* HTTP header to ensure that the browser does not re-use the existing TCP connection to the attacker [3].

For these reasons, although DNS pinning is a security measure that can defend against basic DNS rebinding attacks, it appears that it was not fully implemented by Web browsers. This eventually led to the implementation of second-generation DNS rebinding attacks, which are known as Anti-DNS pinning attacks.

E. ANTI-DNS PINNING REBINDING ATTACKS

The combination of DNS rebinding and anti-DNS pinning was first employed in 2006. These attacks are known as anti-DNS pinning attacks, and the distinction from the basic attacks is important because, as was discovered during this research, there is a lot of confusion about terminology related to DNS rebinding attacks. So, the first attacks, as described above, should be considered basic DNS rebinding attacks, whereas the attacks that were carried out after the implementation of DNS pinning and which act against it, should be considered anti-DNS pinning rebinding attacks. This author believes that this is a necessary distinction to make, in order to make the concept clearer.

1. Anti-DNS Pinning Rebinding Attack using JavaScript

In 2006, an anti-DNS pinning rebinding attack using JavaScript was presented by researchers [10], which was a variant of the basic attack. However, this time the challenge was DNS pinning, which had been applied in Web browsers as an extra security measure against DNS rebinding. Despite this, the attack succeeded in defeating DNS pinning by taking advantage of its vulnerabilities.

The scenario published by M. Johns was as follows:

- The victim visits the malicious Web site attacker.com and loads the script it contains.
- The attacker then changes the DNS entry of attacker.com in order to resolve to the internal server's IP address, which is the target. Moreover, the attacker disconnects the Web server that was running on the original IP address.
- The script uses a timed event (`setIntervall` or `setTimeout`) to load a Web page from attacker.com.
- The victim's Web browser executes the script and tries to connect back to attacker.com using the IP address, which is bound to it due to DNS pinning. But, as the Web server is no longer available, the connection is rejected and DNS pinning is dropped, due to the weakness described in the previous section.
- The browser then drops the DNS pinning and does a new DNS lookup request for attacker.com. This time, the response results in a different IP address; the browser has removed from its cache the previous mapping of the server hostname (attacker.com) to an IP address, so cannot be protected from the misdirection.
- As the new IP address points to the internal server, the attacker's script is now able to access the internal server's content and reveal it [10].

The key point to this attack is that the attacker undermines the browser's DNS pinning by rejecting the connection back to him and then using DNS rebinding to succeed in getting access to an internal server that otherwise would not be permitted. It is interesting to note that there are several ways to accomplish that rejection. Some of these can be summarized as follows:

- Web server can simply be disconnected or stopped [10]
- Web server can apply a firewall rule to its own IP address. So, when the victim tries to connect using its IP address, the connection will be rejected [9].
- The attacker can terminate the TCP connection using the RST (reset) control bit. This idea comes from the TCP Reset Denial of Service attacks [11] but it can also be used for defeating DNS pinning [3].
- The attacker may include in the HTML code the element:

  `<img src= “http://attacker.com:81”>`,

  This is an instruction to find a Web page’s image in a closed port, so the connection will fail and cause the victim’s browser to initiate a new DNS query [12].

2. Anti-DNS Pinning DNS Rebinding Attack Using Adobe Flash Player 9.0.48.0

DNS rebinding attacks affected Adobe Flash Player, which is a very popular software platform. A brief but comprehensive description of Adobe Flash Player is provided as follows from Wikipedia:

> The *Adobe Flash Player* is software for viewing animations and movies using computer programs such as a Web browser; in common usage, *Flash* lets you put animation and movies on a Web site. *Flash player* is a widely distributed proprietary multimedia and application player created by *Macromedia* and now developed and distributed by *Adobe* after its acquisition. *Flash Player* runs *SWF* files that can be created by the *Adobe Flash* authoring tool, by *Adobe Flex* or by a number of other Macromedia and third party tools.
>
> *Adobe Flash*, or simply *Flash*, refers to both a multimedia authoring program and the *Adobe Flash Player*, written and distributed by *Adobe*, that uses vector and raster graphics, a native scripting language called *ActionScript* and bidirectional streaming of video and audio. Strictly speaking, *Adobe Flash* is the authoring environment and *Flash Player* is the virtual machine used to run the *Flash* files, but in colloquial language these have become mixed:

> "Flash" can mean the authoring environment, the player, or the application files. [13]

Researchers from Stanford security team in 2006 published an attack scenario against Adobe Flash Player 9.0.48.0 [3], which also affected the previous Flash versions. The attack used DNS rebinding, exploiting vulnerabilities in the interaction between Flash files and Web browsers. These vulnerabilities were characterized as critical, according to Adobe official site as was stated in a security bulletin of 2007:

> Critical vulnerabilities have been identified in Adobe Flash Player that could allow an attacker who successfully exploits these potential vulnerabilities to take control of the affected system. A malicious SWF must be loaded in Flash Player by the user for an attacker to exploit these potential vulnerabilities [14].

First, the Stanford security team discovered that Adobe Flash Player 9 does not perform any DNS pinning, nor does it use the browser's pinning, which means that Flash is vulnerable to DNS rebinding attacks. The vulnerability is registered and described in Common Vulnerabilities and Exposures database (CVE-2007-5275):

> The Adobe Macromedia Flash 9 plug-in allows remote attackers to cause a victim machine to establish TCP sessions with arbitrary hosts via a Flash (SWF) movie, related to lack of pinning of a hostname to a single IP address after receiving an allow-access-from element in a cross-domain-policy XML document, and the availability of a Flash Socket class that does not use the browser's DNS pins, aka DNS rebinding attacks, a different issue than CVE-2002-1467 and CVE-2007-4324 [16].

Second, the reference to the cross-domain-policy XML document is about the policy file request that Adobe Flash Player sends back to the server as well as the response it receives. The attack exploits the effects of an unauthorized policy file. In order to make this better understood, author provides a description of how policy files work in Flash, using the material provided from the official Adobe site. As shown in Figure 3, when a user visits a Web page that embeds a

Small Web Format (SWF) file, the policy file request and reply is as follows (the steps map to numbers shown in Figure 3):

- User visits a.com and browses the SWF file, which is embedded.
- The SWF file from a.com takes permission to load data from b.com using the policy file on b.com. The b.com site hosts a policy file that gives permission for a.com domain, which means that a SWF file from a.com can load data directly from b.com site.
- Now the SWF file can load data from b.com, manipulating the privileges the user has for connecting with b.com.
- Since the SWF file has the permission it can send any data it finds on b.com back to its own server at a.com [15].

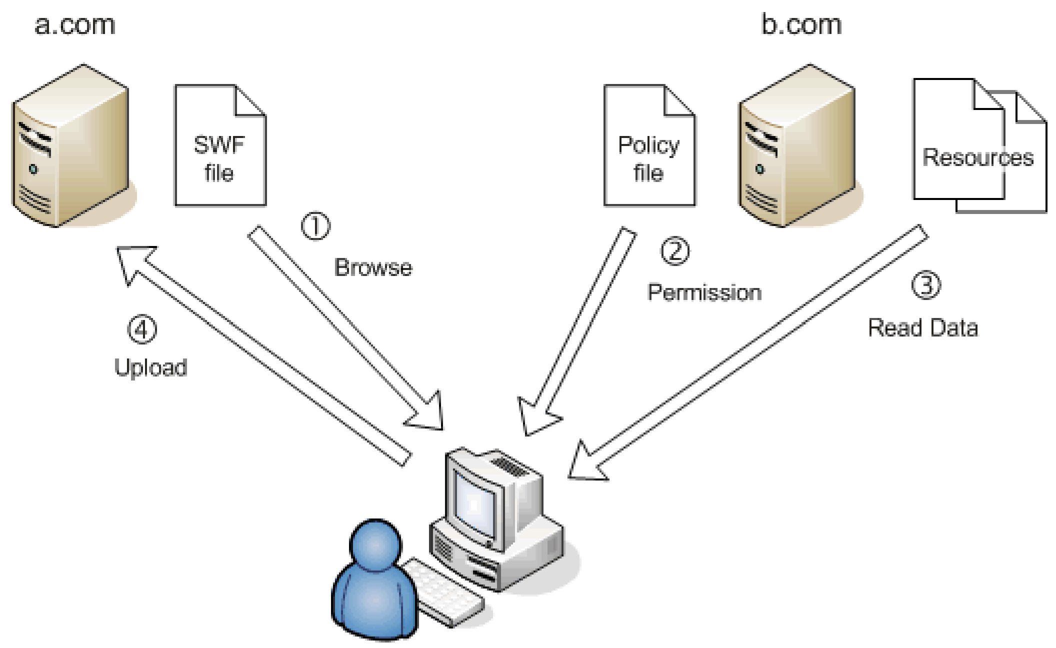


Figure 3. Data flow enabled by a policy file [From 15]

As a result of these vulnerabilities, the attack scenario using Adobe Flash Player 9.0.48.0 was documented by Stanford security researchers, providing a feasible DNS rebinding attack as shown in Figure 4, described in the following steps:

- The client’s Web browser visits a malicious Web site attacker.com that embeds a SWF movie.
- The SWF movie opens a socket to the attacker’s IP address according to the attacker’s DNS server response, which also is configured with a short TTL.
- Flash Player in the victim’s machine sends back a policy request file using the current socket connection.
- The attacker’s server responds with a policy file as shown in Table 2, which permits all domains to make socket connections to all ports.
- The SWF file tries to make a new connection to attacker.com, but due to the low TTL, the DNS entry has expired and Adobe Flash Player queries again for an IP address. At this point, DNS rebinding appears because the attacker responds with the IP address of the target server [3].

```
        <?xml version="1.0"?>
    <cross-domain-policy>
    <allow-access-from domain="*" to-ports="*" />
    </cross-domain-policy>
```

Table 2. Vulnerable Adobe Flash Player policy file [From 3]

As stated previously, Adobe Flash Player 9 does not perform DNS pinning, hence it permits the connection to the new IP address.

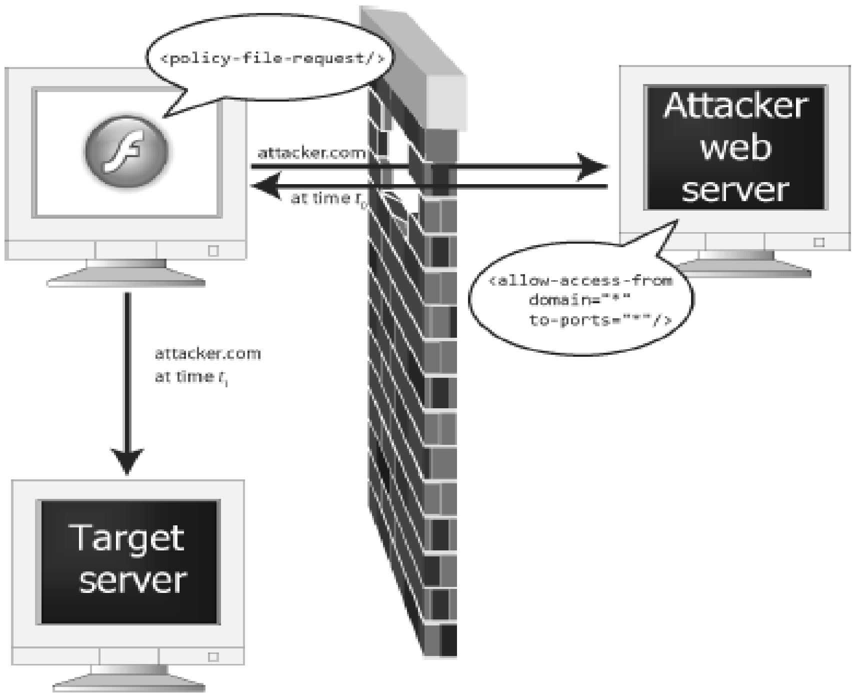


Figure 4. DNS rebinding attack using Adobe Flash Player [From 3]

F. PLUG-INS AND MULTI-PIN VULNERABILITIES

Modern browsers use several plug-ins for interacting with Web pages. In fact, the user has no other option than to install and use these platforms in order to be capable of browsing his favorite sites. For example, Java, Adobe Flash Player, and Adobe Acrobat are tools that every Web browser must have installed. On the other hand, all these useful tools have introduced new vulnerabilities that can be exploited by DNS rebinding among many other types of attacks that are out of the scope of this research.

Many plug-ins permit direct socket access back to their origins and the fact that they maintain separate DNS pin databases introduces the basis for the third generation of DNS rebinding attacks. The paradox is that the lack of separate DNS pinning in Adobe Flash Player 9.0.48.0 led to the attack scenario described previously, whereas the presence of separate DNS pinning leads to new vulnerabilities and exposures. Indeed, if a plug-in platform pins to the attacker’s IP address and the Web browser pins to the target’s IP address, the attacker can take advantage of the communication capabilities between them to get around same origin policy restrictions.

For example, Java Virtual Machine (JVM) maintains DNS pins separately from the browser, opening up the possibility of DNS rebinding vulnerabilities. According to Stanford security team researchers, Java is exposed to the following weaknesses [3]:

> *LiveConnect* bridges JavaScript and the JVM in Firefox and Opera, permitting script access to the Java standard library, including the Socket class, without loading an applet. The browser pins to the attacker’s IP address, but the JVM spawned by *LiveConnect* does a second DNS resolve and pins to the target’s IP address. The attacker’s JavaScript can exploit this pin mismatch to open and communicate on a socket from the client machine to an arbitrary IP address on an arbitrary destination port, including UDP sockets with a source port number $\geq$ 1024.
>
> Applets with Proxies are also vulnerable to a multi-pin attack, regardless of which browser the client uses. If the client uses an HTTP proxy to access the Web, there is yet another DNS resolver involved—the proxy. When the JVM retrieves an applet via a proxy, it requests the applet by host name, not by IP address. If the applet opens a socket, the JVM does a second DNS resolve and pins to the target’s IP address.
>
> *Relative Paths* can cause multi-pin vulnerabilities. If a server hosts an HTML page that embeds an applet using a relative path with the parameter *mayscript* set to *true*, that machine can be the target of a multi-pin attack. The browser pins to the target, retrieves the HTML page, and instructs the JVM to load the applet. The JVM does a second DNS resolve, pins to the attacker, and retrieves a malicious applet. The applet instructs the browser, via JavaScript, to issue *XMLHttpRequests* to the target’s IP address [3].

Another example is Adobe Flash Player, whose version 9 exposure due to lack of DNS pinning has already been described. However, Flash could additionally be vulnerable to multi-pin attacks even if it uses DNS pinning. The reason is due to the operation of the Flash platform.

When the user visits a Web page that contains Flash content, the browser downloads the content (a movie, for example) and initiates Flash Player, transferring the movie's origin by host name. When the attacker's movie attempts to open a socket, Flash Player does a second DNS query back to the movie's origin and the attacker could respond with the target's IP address, which is the IP address the Flash Player would use for pinning [3].

Although the methods described so far require the attacker to send the IP address of the target machine, there is also an alternate method. The attacker can simply guess the internal host name of the target server, for example cs.nps.edu, and rebind attacker.com to a CNAME record which maps to that host name. When the attacker guesses an existing host name, the client's own recursive DNS resolver will complete the resolution and return the IP address of the target. Otherwise, the attacker has to search for IP addresses to find an interesting target using a scanning tool such as Nmap or Superscan [3].

It is important to note that the essential initial condition for mounting a DNS rebinding attack is to attract the victim to visit the attacker's Web page. This is subject to techniques based on Social Engineering, which is out of the scope of this research. For the purpose of this study, the author assumes that the potential victim has already been attracted to visit the malicious Web page.

G. IMPACT OF DNS REBINDING ATTACKS

The exploitation of the various DNS rebinding vulnerabilities involves several components: the DNS service itself, the security policies applied (or not) in browsers (same origin policy and DNS pinning), and the operation of plug-ins that interact with all modern Web browsers. An attacker can take advantage of these vulnerabilities. Depending on the result achieved, the impact of these attacks has been separated into two categories: firewall circumvention and IP hijacking [3].

1. Firewall Circumvention

A firewall bounds network traffic between computer networks in different zones of trust. For example, a firewall can be configured to block connections from the public Internet to a LAN's internal machines and negotiate connections even from internal machines to servers with sensitive data according to the administrator's wish. Firewall circumvention attacks bypass this prevention on inbound connections, allowing the attacker to connect to internal servers while the user is visiting the attacker's Web page.

a. Stealing Data

Web servers inside corporate firewalls often contain confidential documents, relying on the firewall to prevent non-legitimate users from retrieving these documents. However, using a DNS rebinding attack, an attacker can control the victim client's browser to read and get these documents.

b. Exploiting Unpatched Machines

Note that the presence of a firewall often leads network administrators to the erroneous impression that this defense is enough to keep away attackers. Considering also that the patching process is time-consuming and expensive, administrators may not patch internal machines as quickly as Internet-facing machines. Using DNS rebinding, an attacker can attempt to exploit known vulnerabilities in unpatched machines on the internal network. If an exploit succeeds, the attacker can form a presence within the firewall that persists even after the victim closes the Web browser.

c. Exploiting Internal Open Services

Similarly, relying on the firewall, internal networks support many open services intended only for the organization's use. Moreover, users inside firewalls often feel comfortable creating file shares or FTP servers accessible to anonymous users under the assumption that the servers will be available only to clients within the network. Finally, routers are often installed without changing the default password or encrypting route information packets.

Consequently, if an unsuspicious internal user visits the attacker's external Web page, the implementation of DNS rebinding may succeed in exposing these services to unfortunate results. For example, internal network printers may print until they exhaust their supplies in paper and ink, or shared documents can be stolen. Finally, routers with default passwords can be accessed so that attacker can change the configuration files [3].

2. IP Hijacking

DNS rebinding attacks can also target machines on the public Internet. In this case, an attacker uses a victim's browser, taking advantage of the trust that public services have in the victim's IP address. Once the attacker has taken control of the victim's IP address using the DNS rebinding techniques, there are several attacks that can be mounted.

a. Click Fraud

Click Fraud can be defined as any click done in bad faith, which means any click where there is no intention by the user to buy, browse or get information from the Web page visited. Click fraud happens in the case where the purpose of a click is to either drain funds or generate profits [17].

b. Spam Messages

Many e-mail servers blacklist IP addresses known to send spam e-mail. By hijacking a client's IP address, an attacker can send spam from trusted IP addresses. In order to send spam e-mail, the attacker has to send packets to SMTP servers on port 25. Although most browsers do not permit this action, it is permitted by Adobe Flash Player and Java [3].

c. Breaking IP-based Authentication

Many Internet services still employ IP-based authentication. After hijacking an authorized IP address, the attacker can access the service, defeating the authentication mechanism. Because the communication originates from an IP address actually authorized to use the service, the service provider fails to recognize the security breach.

d. Framing Clients

An attacker who hijacks an IP address can perform illegal actions and set up the victim. For example, if an attacker gains unauthorized access to a computer system using the victim's hijacked IP address, the logs will associate the client to that illegal action instead of the attacker because the attack seems to originate from the victim's machine. [3]

THIS PAGE INTENTIONALLY LEFT BLANK

III. EXPERIMENTS–SETUP AND RESULTS

A. CHAPTER OVERVIEW

The purpose of this chapter is to describe the experiments conducted for this thesis and present the main results obtained from them. Section B describes the network testbed in detail; the testbed was built using the available hardware components in the Networks Laboratory of the Computer Science department. Section C, provides the description and experimentation with a DNS rebinding attack scenario and the results are provided. In Section D, the research focused on vulnerabilities to DNS rebinding for different Adobe Flash Player versions and experiments using socket programming between a client and a server in an effort to address security issues with the same origin policy and DNS-pinning with respect to DNS rebinding attacks.

B. CONFIGURATION OF TESTBED

The testbed represented a network topology for experiments. As shown in Figure 5, it consisted of the attacker's LAN with domain name joker.lab, the target LAN with domain name angel.lab, which hosted some important enterprise services, and a third LAN in the angel.lab domain, which simulated a group of authorized clients for the services (e.g., Web) hosted on the target LAN. The testbed was designed to operate locally, isolated from the Internet for testing purposes. The following sections describe in detail the hardware, software, and configuration of the testbed components.

1. Hardware

The architecture of the testbed consisted of the following components:

- Three (3) Cisco 2600 series routers.
- Three (3) Catalyst 1900 series switches.
- Six (6) DELL Optiplex personal computers with a 1.86 GHz Intel Core2 CPU.

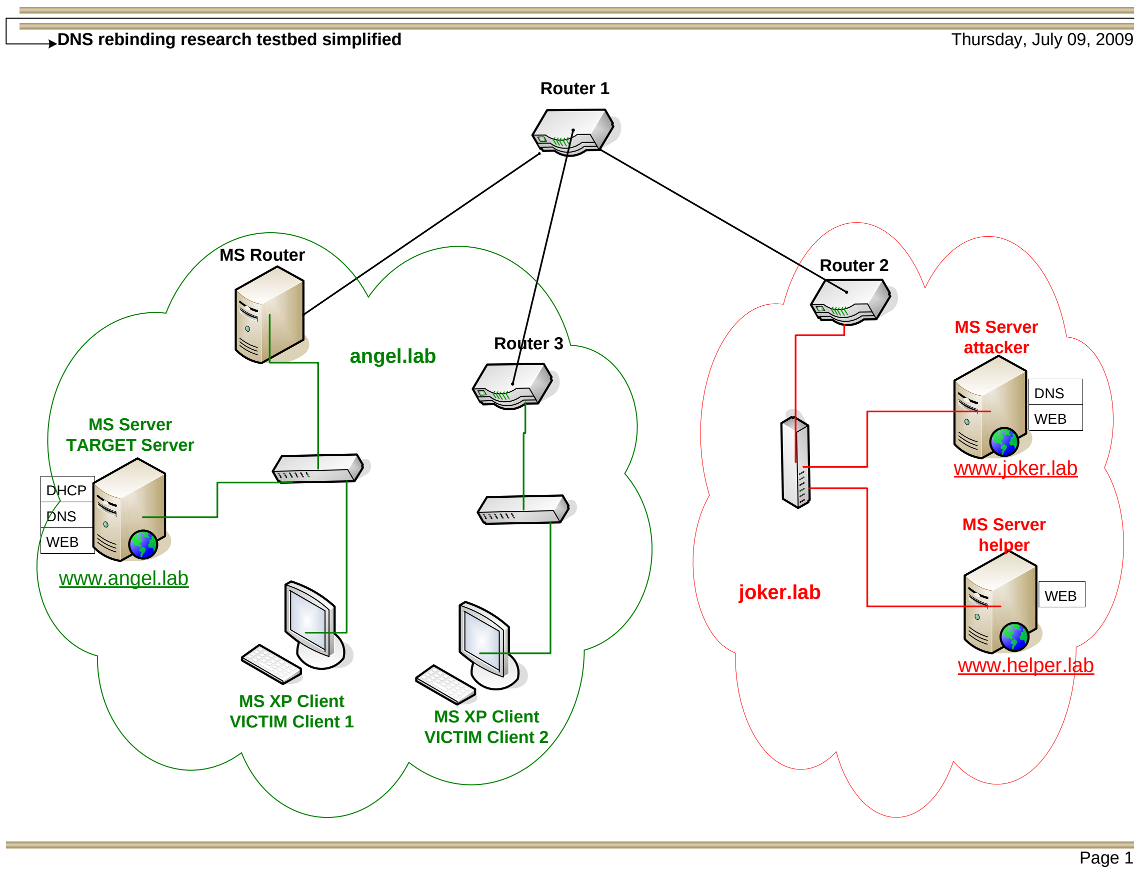


Figure 5. Research network testbed

2. Software

The following operating systems and applications were installed:

- All servers run Microsoft Windows Server 2003 R2 Enterprise Edition SP2, which was provided by MSDNAA Software Center, of which NPS CS students can be members. Servers provide DNS and DHCP services to their LANs, as well as Web server capabilities for hosting and managing their Web pages using IIS 6.0, which is embedded in MS Server 2003 software.
- Clients run Microsoft Windows XP Service Pack 2.
- Clients had installed the most popular Web browsers and an application that permits the execution of previous versions of Internet Explorer. Web browsers included:

    - Mozilla Firefox 3.0.8
    - Google Chrome 1.0.154.65
    - Safari 4 Public Beta (528.16) release for Windows
    - Internet Explorer 7
    - An application which is an installer that contains multiple versions of IE, specifically IE 4.01, IE 5, IE 5.5, IE 6.0, which is very useful in order to test a Web site in various versions of Internet Explorer [20]. The application was successfully tested in Windows XP SP2 client machines.
- All PCs had Java(TM) SE Runtime Environment (build 1.6.0.13) installed. Additionally, Eclipse Platform 3.4.2 was installed on both attacker host and target server.
- Adobe Flex Builder 3 built on Eclipse was installed in attacker server for Flash application development and execution [21].
- Archived Flash Player 9 package, which contains all the Flash Player 9 versions, as well as the necessary uninstaller for testing purposes, was downloaded [22].
- Network protocol analyzer Wireshark 1.0.6 installed on all the testbed machines.

3. Network Topology Description

The research testbed represents the network topology shown in detail in Figure 6. It consists of two sides, the attacker LAN and the target LAN.

The target LAN (left side of the diagram) is the angel.lab domain (192.168.23.8/29), which contains a server that plays the role of the target server and a client which plays the role of a victim client (victim 1). There is also an additional LAN under Router 3, which includes another victim client (victim 2). This LAN (192.168.23.0/29) has access to angel.lab, which means that victim 2 is a legitimate user who is permitted access to the angel.lab server, so this client is not firewalled.

The angel.lab target server provides DHCP, DNS and WEB services to the LAN. It hosts and maintains the Web page www.angel.lab, which is meant to be accessible only to legitimate clients of this LAN. Thus, in this network topology, legitimate clients of that target server are victim client 1 and victim client 2 only. MS Router is the default gateway of this LAN, which is a server configured as a router and which additionally has the responsibility of applying the firewall rules that protects the LAN from outsiders.

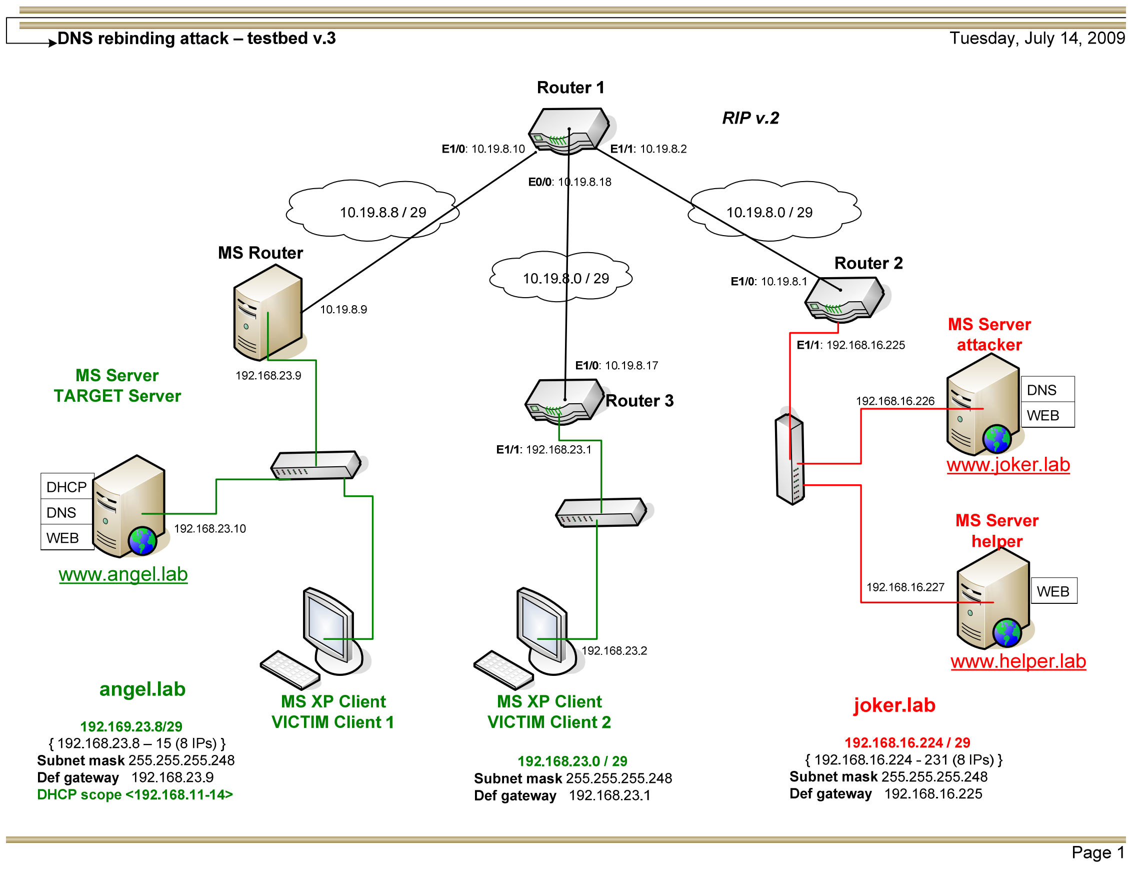


Figure 6. Detailed network testbed architecture

The right side shows the joker.lab domain (192.168.23.16/29), which is the attacker's side under Router 2. It consists of the attacker server, which provides DNS and WEB services, hosting and maintaining the www.joker.lab page. Additionally, the attacker controls a second Web server in the same domain, which hosts the www.helper.lab page. That server is required to cooperate with the primary attacker server in several ways, such as receiving the results of an attack. Router 2 is the default gateway of the attacker's LAN.

All LANs are provided with switches with factory default configuration. Router 1 is directly connected to all LANs, providing the necessary connectivity between the components of the network topology. Appendix A contains the configuration files of the Cisco routers (1, 2, and 3). The testbed's routers utilize dynamic routing configured to use the RIP version 2 routing protocol.

The correct operation of the network topology is fundamental for the implementation of experiments and the export of scientifically argued conclusions. For that reason, a script was written to test the connectivity between the testbed's LANs. The script is a .bat file running in Windows, which pings all the interfaces of the testbed's hosts in order to check that all hosts are connected and communicate properly with each other. The code of mspinger.bat is contained in Appendix A, as well as an output of its execution.

C. DNS REBINDING ATTACK SCENARIO VIA JAVASCRIPT

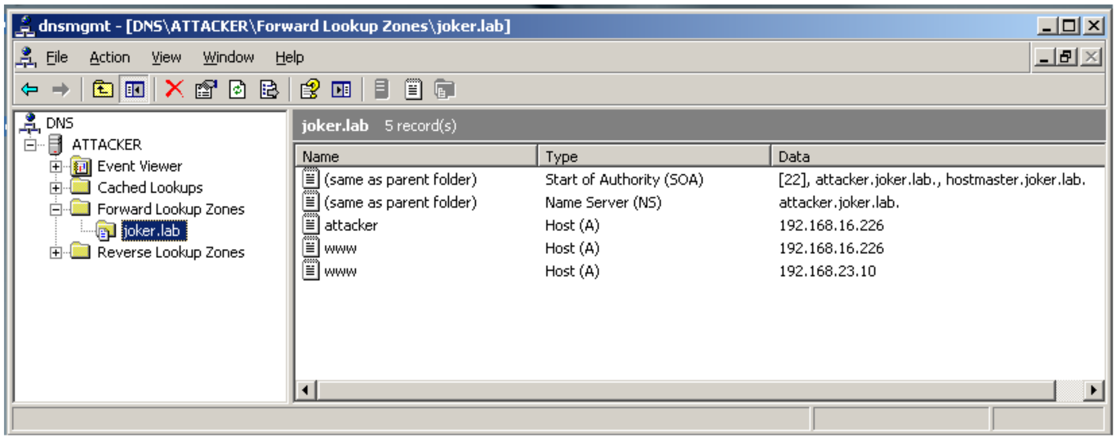
The DNS rebinding attack scenario used during this research was provided by Collin Jackson of Stanford Web security team. It is an attack method that uses round-robin DNS and the `XmlHttpRequest` object as described in detail in the following sections.

1. Attack Scenario

The scenario is about getting around a firewall (firewall circumvention). The attacker is firewalled outside the target LAN, and tries to intrude in order to perform malicious actions, specifically stealing data from the target Web server. According to the scenario, the unsuspecting victim visits the attacker's malicious Web site and the attacker then channels HTTP traffic through the victim in order to get to the target server inside the firewall. The attacker's purpose is to read the content of http://www.angel.lab*,* which he cannot do normally with HTML, but which he can do using `XmlHttpRequest` object and DNS rebinding.

The attacker’s Web server hosts the www.joker.lab HTML page; it contains a function whose purpose is to make an `XmlHttpRequest` to the root directory and alerts the result, showing the HTML content of the page on the screen. The attacker’s DNS server needs two IP addresses on the round-robin DNS. The first IP address maps to the attacker’s Web server (192.168.16.226) and the second IP address maps to the victim’s Web server (192.168.23.10). This configuration is shown in Figure 7.

The attacker uses DNS rebinding to trick the victim’s Web browser into thinking that 192.168.16.226 and 192.168.23.10 are in the same origin, so that the Web page loaded from first IP (www.joker.lab) will be able to perform an `XmlHttpRequest` to the second IP (192.168.23.10), which corresponds to the internal server’s Web page (www.angel.lab).



Name	Type	Data
(same as parent folder)	Start of Authority (SOA)	[22], attacker.joker.lab., hostmaster.joker.lab.
(same as parent folder)	Name Server (NS)	attacker.joker.lab.
attacker	Host (A)	192.168.16.226
www	Host (A)	192.168.16.226
www	Host (A)	192.168.23.10

Figure 7. Attacker’s DNS server records A

In this way, when the victim client visits the attacker’s domain name, he retrieves the attacker’s page (www.joker.lab), as he was supposed to do. The attacker’s server is then physically disconnected from the network, so that the original IP address for that page is no longer available. The browser will make a new DNS lookup by itself after a time interval, which depends on the browser, in order to attempt to reconnect. However, this time, due to round-robin DNS, it will use the second IP address, which belongs to the internal target Web server. As long as the attacker server is disconnected, there must be a second Web server under the attacker's control, to which the result of the attack will be redirected.

Thus, the key components for this DNS rebinding attack scenario are:

- The attacker's Web HTML page www.joker.lab containing the malicious code.
- The round-robin DNS configuration in the attacker's DNS server with two IP addresses, where the second is the target server's IP address.
- The unavailability of the attacker's Web server, which takes place after the victim's initial visit to the attacker's Web page.

2. Attack Process

The procedure is carried out by following the steps shown pictorially in the timing diagram in Figure 8 and described as follows:

- Initially, the client's browser queries for www.joker.lab IP address asking its own local DNS server (time t1).
- The DNS server sends back to client the response received from the authoritative DNS server, which is under the attacker's control. Thus, the client receives 192.168.16.226 and 192.168.23.10, in that order (time t2).
- The client's browser uses the first IP address, sends a GET message and receives the HTML content of www.joker.lab (time t3).
- At this point, the attacker physically disconnects the Web server he controls from the network in order to make it unavailable. The server can also be made unavailable using other techniques, such as those described in previous sections. For example, the Web page may contain an instruction to find an image in a closed port, so the connection will fail and cause the victim browser to initiate a new DNS query.

- The client's browser should make a new query after an amount of time (dt) depending on the Web browser, according to Stanford security team tests, as discussed in previous sections. As a result of round-robin DNS, this time the client uses the second IP address 192.168.23.10 in order to visit the same Web page as before, the attacker's www.joker.lab.
- The same origin policy breaks because the client sends a GET message to 192.168.23.10, which corresponds to www.angel.lab although the client intends to visit www.joker.lab (time t4).
- In the case where the client's browser successfully receives the HTML content from www.angel.lab, the intrusion is successful, because now data can be sent back to the attacker's domain (time t5).

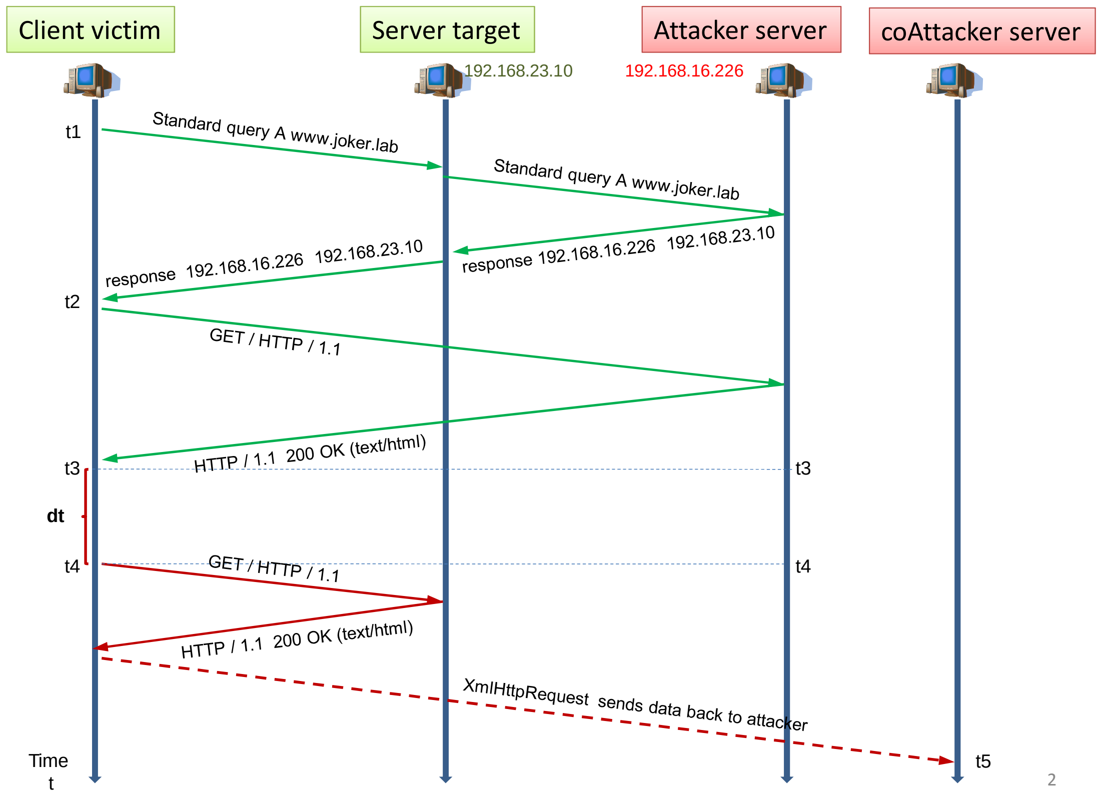

Client victim
Server target
Attacker server
coAttacker server
192.168.23.10
192.168.16.226
t1
Standard query A www.joker.lab
Standard query A www.joker.lab
response 192.168.16.226 192.168.23.10
response 192.168.16.226 192.168.23.10
t2
GET / HTTP / 1.1
HTTP / 1.1 200 OK (text/html)
t3
t3
dt
t4
t4
GET / HTTP / 1.1
HTTP / 1.1 200 OK (text/html)
XmlHttpRequest sends data back to attacker
Time
t
t5
2

Figure 8. Timing diagram of DNS rebinding attack scenario

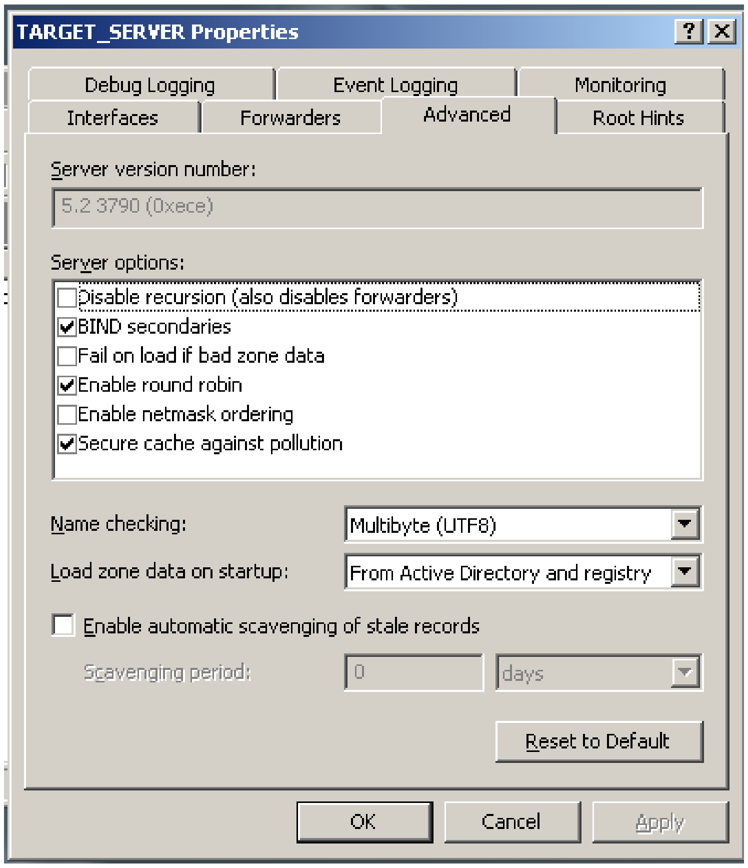
3. Testing–Results

The testbed uses Wireshark to follow the steps of the attack procedure and to watch the conversations between clients and servers. Initially, at time t1 the victim client makes a DNS query for the attacker's page, which goes to its DNS server in the LAN to which it belongs. The DNS server queries for the Web page, and the query gets a response from the DNS server, which the attacker controls. The response finally reaches the client at time t2. As shown in Figure 8, the response contains the two IP addresses, first the attacker's Web server, and second the target server's IP address.

a. Round-robin DNS Configuration

At this first step of the attack process, the DNS server configuration of both target and attacker plays an important role. Although the DNS server in Windows 2003 works in round-robin by default, there is a **subnet mask ordering option**, which is chosen by default. This option ensures that clients are directed to the nearest server that matches their request. This utility prevents the implementation of the attack, because during the experiments, victim client 1 is directed to the nearest server; this is the target server due to its IP address belonging to the same subnet as the client and being the nearest. Figure 9 shows the target server's DNS configuration, where the **enable netmask ordering** option is unchecked in order to make the attack scenario feasible.

After deactivating that option, the round-robin DNS works on both attacker and target servers, in the way the attacker desires. Thus, the DNS server gives back to the client the attacker's IP address as its first option rather than the nearest, which is the second. Round-robin can be seen in action after that interference using NSLOOKUP, as shown in the Figure 10, which details NSLOOKUP running on the target server machine. However, the client machine's behavior may prevent round-robin from working, as described in the following section.



Server version number: 5.2 3790 (0xece)

Server options:

- [ ] Disable recursion (also disables forwarders)
- [✓] BIND secondaries
- [ ] Fail on load if bad zone data
- [✓] Enable round robin
- [ ] Enable netmask ordering
- [✓] Secure cache against pollution

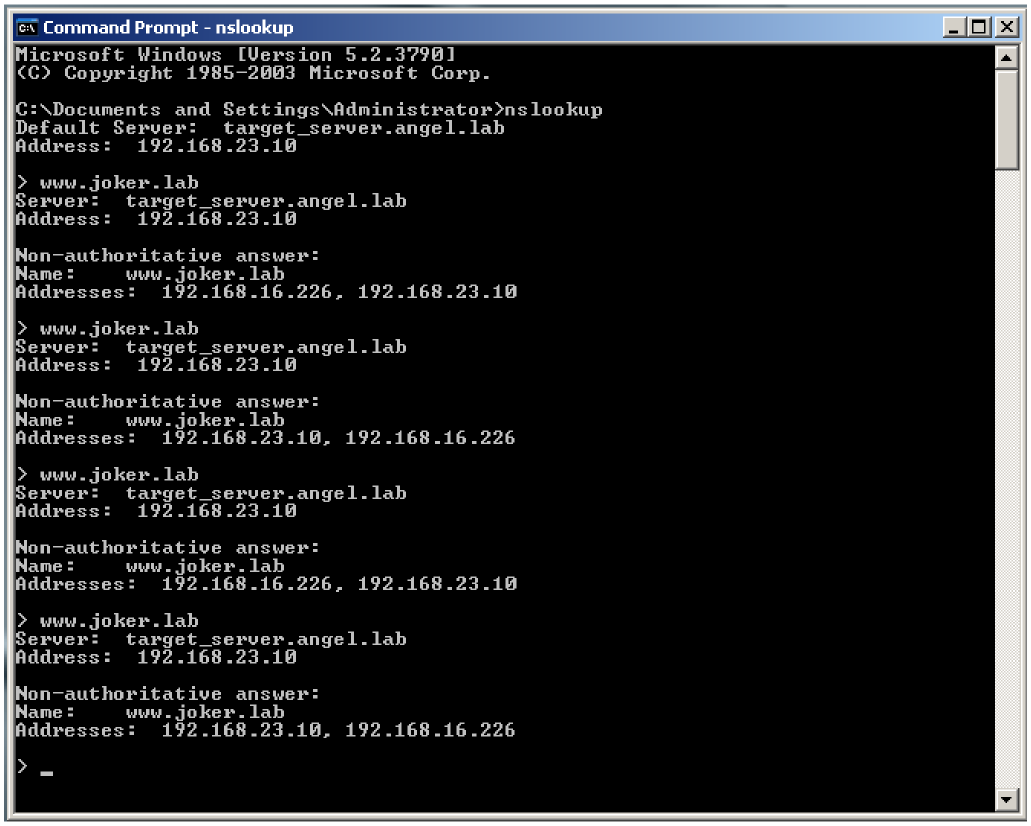
Name checking: Multibyte (UTF8)

Load zone data on startup: From Active Directory and registry

- [ ] Enable automatic scavenging of stale records

Scavenging period: 0 days

Figure 9. Netmask ordering option



```
Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

C:\Documents and Settings\Administrator>nslookup
Default Server:  target_server.angel.lab
Address:  192.168.23.10

> www.joker.lab
Server:  target_server.angel.lab
Address:  192.168.23.10

Non-authoritative answer:
Name:    www.joker.lab
Addresses:  192.168.16.226, 192.168.23.10

> www.joker.lab
Server:  target_server.angel.lab
Address:  192.168.23.10

Non-authoritative answer:
Name:    www.joker.lab
Addresses:  192.168.23.10, 192.168.16.226

> www.joker.lab
Server:  target_server.angel.lab
Address:  192.168.23.10

Non-authoritative answer:
Name:    www.joker.lab
Addresses:  192.168.16.226, 192.168.23.10

> www.joker.lab
Server:  target_server.angel.lab
Address:  192.168.23.10

Non-authoritative answer:
Name:    www.joker.lab
Addresses:  192.168.23.10, 192.168.16.226

> _
```

Figure 10. NSLOOKUP running on target server showing round-robin DNS response for “www.joker.lab”

b. Subnet Prioritization

The second observation comes from victim client 1’s behavior. Victim client 1 is a host that belongs to the same LAN as the target server. When a browser tries to visit the attacker’s Web page, it uses the IP address of its subnet, which is the second IP address and not the first one that it received from its DNS server. This is another constraint for the scenario, which happens because the Windows XP DNS resolver uses **Subnet Prioritization** by default. According to Microsoft support documentation about this issue [23]:

> If the resolver receives multiple IP address mappings (A resource records) from a DNS server, and some of the records have IP addresses from networks to which the computer is directly connected, the resolver places those resource records first. This

behavior reduces network traffic across subnets by forcing computers to connect to network resources that are closer to them [23].

Thus, subnet prioritization on the client's machine prevents round-robin DNS from working victim 1. There is a way to disable this feature by adding the `PrioritizeRecordData` registry entry with a value of 0 in the registry key: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services \Dnscache\Parameters` [23].

The procedure that has to be followed [24] is shown in Table 3, which was tested successfully on victim client 1.

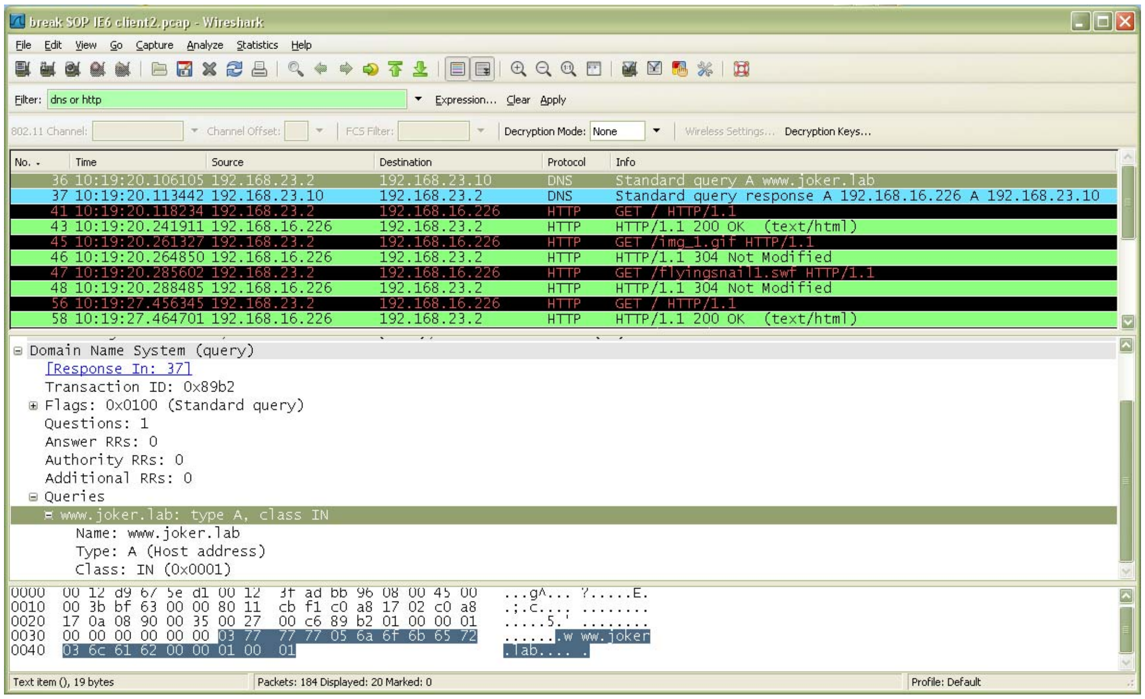
1. Start a registry editor (e.g., regedit.exe) on each client machine.
2. Navigate to the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services \Dnscache \Parameters registry subkey.
3. From the Edit menu, select New, DWORD Value.
4. Enter the name PrioritizeRecordData, then press Enter.
5. Double-click the new value, set it to 0, then click OK.
6. Close the registry editor.
7. Reboot the machine for the change to take effect.

To re enable subnet prioritization, either delete the PrioritizeRecordData registry value or set this value to 1.

Table 3. Disable subnet prioritization procedure [From 24]

Consequently, the round-robin DNS works on victim client 1 if and only if a change in its registry occurs as described above. This condition makes it impossible to mount the attack scenario using this client, because the attacker would have to be able to change the victim's registry remotely even before the victim visits his malicious Web page (www.joker.lab).

However, the attack scenario is still feasible as regards victim client 2. This client does not belong to the same LAN as the target server and he has access to angel.lab, as detailed in the setup configuration description. Therefore, the default subnet prioritization feature does not prevent the round-robin from working. In Figure 11, the Wireshark snapshot from victim client 2 shows that phase of the attack scenario, where the client visits the attacker’s Web page using the first IP address of his DNS query response.


break SOP IE6 client2.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: dns or http
Expression... Clear Apply
802.11 Channel: Channel Offset: FCS Filter: Decryption Mode: None Wireless Settings... Decryption Keys...

```
No. . Time               Source          Destination     Protocol Info
   36 10:19:20.106105 192.168.23.2    192.168.23.10    DNS      Standard query A www.joker.lab
   37 10:19:20.113442 192.168.23.10   192.168.23.2     DNS      Standard query response A 192.168.16.226 A 192.168.23.10
   41 10:19:20.118234 192.168.23.2    192.168.16.226   HTTP     GET / HTTP/1.1
   43 10:19:20.241911 192.168.16.226  192.168.23.2     HTTP     HTTP/1.1 200 OK  (text/html)
   45 10:19:20.261327 192.168.23.2    192.168.16.226   HTTP     GET /img_1.gif HTTP/1.1
   46 10:19:20.264850 192.168.16.226  192.168.23.2     HTTP     HTTP/1.1 304 Not Modified
   47 10:19:20.285602 192.168.23.2    192.168.16.226   HTTP     GET /flyingsnail1.swf HTTP/1.1
   48 10:19:20.288485 192.168.16.226  192.168.23.2     HTTP     HTTP/1.1 304 Not Modified
   56 10:19:27.456345 192.168.23.2    192.168.16.226   HTTP     GET / HTTP/1.1
   58 10:19:27.464701 192.168.16.226  192.168.23.2     HTTP     HTTP/1.1 200 OK  (text/html)

⊟ Domain Name System (query)
    [Response In: 37]
    Transaction ID: 0x89b2
  ⊞ Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊟ www.joker.lab: type A, class IN
        Name: www.joker.lab
        Type: A (Host address)
        Class: IN (0x0001)

0000  00 12 d9 67 5e d1 00 12  3f ad bb 96 08 00 45 00   ...g^... ?.....E.
0010  00 3b bf 63 00 00 80 11  cb f1 c0 a8 17 02 c0 a8   .;.c.... ........
0020  17 0a 08 90 00 35 00 27  00 c6 89 b2 01 00 00 01   .....5.' ........
0030  00 00 00 00 00 00 03 77  77 77 05 6a 6f 6b 65 72   .......w ww.joker
0040  03 6c 61 62 00 00 01 00  01                        .lab.... .

Text item (), 19 bytes    Packets: 184 Displayed: 20 Marked: 0    Profile: Default
```

Figure 11. Victim client 2 retrieving attacker home Web page

Until this point, the tests on this testbed have shown that the attack scenario with round-robin DNS is possible if and only if the following conditions are fulfilled:

- The enable netmask ordering option has to be unchecked in the target DNS server
- The victim client cannot be a member of the same subnet as the target server because the Windows XP DNS resolver uses subnet prioritization by default.

c. Server’s Unavailability

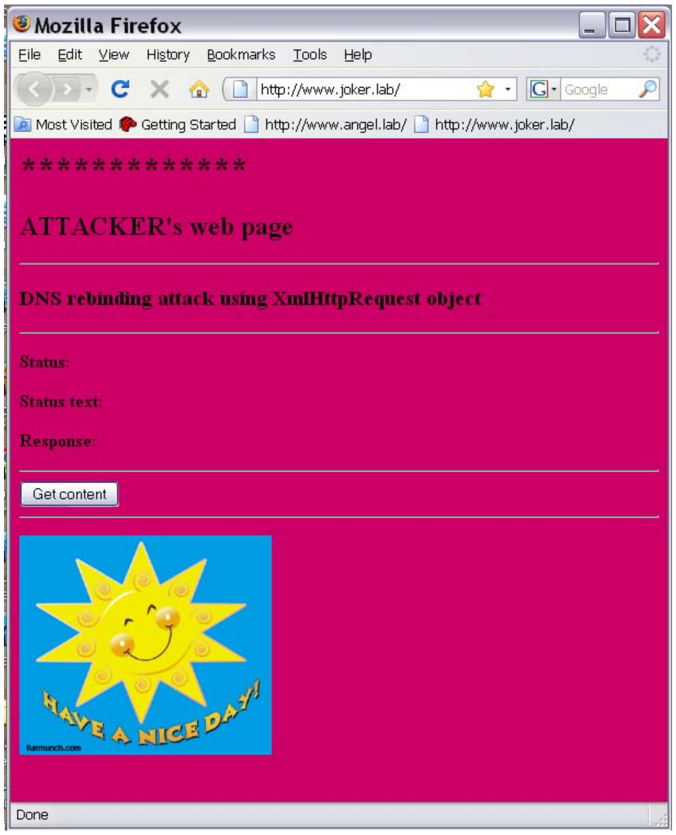
At time t3, victim client 2 has retrieved the attacker’s Web page, www.joker.lab. The attacker then disconnects its Web server physically from the network, so that the IP address that the client used initially is no longer available.

According to Stanford's security team tests [3], the victim's Web browser should try a different IP address within several seconds, contrary to DNS pinning, which is implemented in current Web browsers and which prevents host names from referring to multiple IP addresses.

However, despite extensive testing with all current major Web browsers, as well as with previous versions of Internet Explorer, no such behavior was noticed on this testbed. The victim client's Web browser does not react to the attacker's server unavailability. In an effort to explain this difference in the results, the author forwarded these observations to Collin Jackson from the Stanford security team, who noted that it was probably due to the differences in the configurations between their research's testbed and the this research's testbed. The browser's reaction was simulated using the Web browser's refresh option, which enabled the process to continue. This means that instead of waiting for the Web browser to try a different IP address, the user refreshes the Web page, which leads to the same results because the browser will not be able to connect to the unavailable IP address and will eventually try a different one.

d. Breaking Same Origin Policy

Therefore, when the scenario reaches time t3, the victim client has retrieved the attacker's Web page and the attacker Web server becomes unavailable by physically disconnecting it. For instance, in Figure 12 is shown the attacker's Web page www.joker.lab as it is retrieved from victim using Firefox 3.0.8 browser.



****

ATTACKER's web page

DNS rebinding attack using XmlHttpRequest object

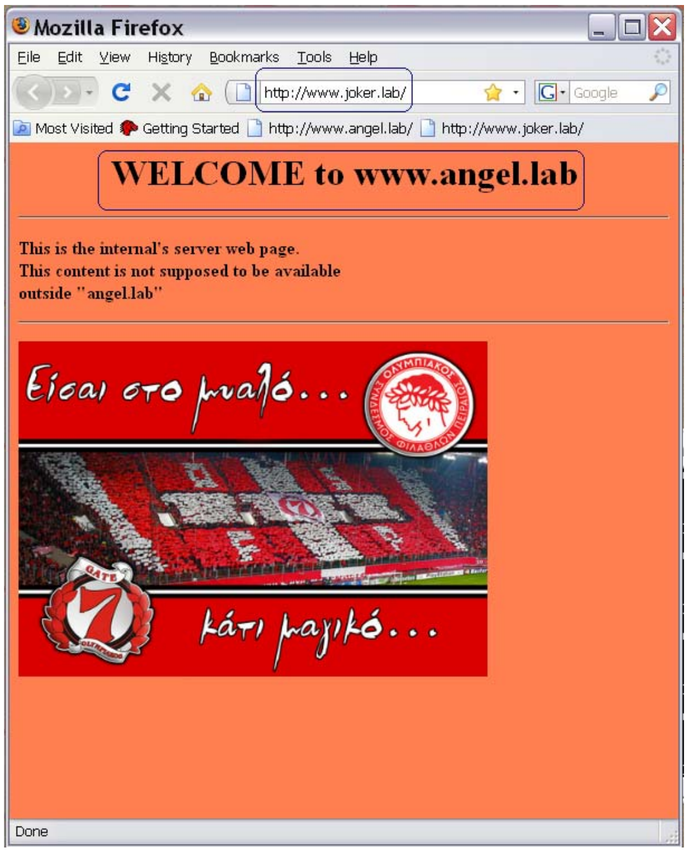
Status:

Status text:

Response:

Figure 12. Victim client 2 retrieves attacker's home Web page

The user then refreshes the Web page but the initial IP address in no longer available. After some period of time dt, the browser uses the second IP address (192.168.23.10), which corresponds to the target server's Web page, www.angel.lab. Figure 13 shows that although the browser thinks that it has retrieved www.joker.lab, in reality it has retrieved the HTML content of the target server's Web page. This is a violation of the same origin policy of the Web browser, which is accomplished using DNS rebinding. The communication between victim client 2 and servers was recorded using Wireshark. For example, Figure 14 shows how the same origin policy in Firefox browser is violated after dt = 1:55 min.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.joker.lab/
Most Visited Getting Started http://www.angel.lab/ http://www.joker.lab/
Done

WELCOME to www.angel.lab

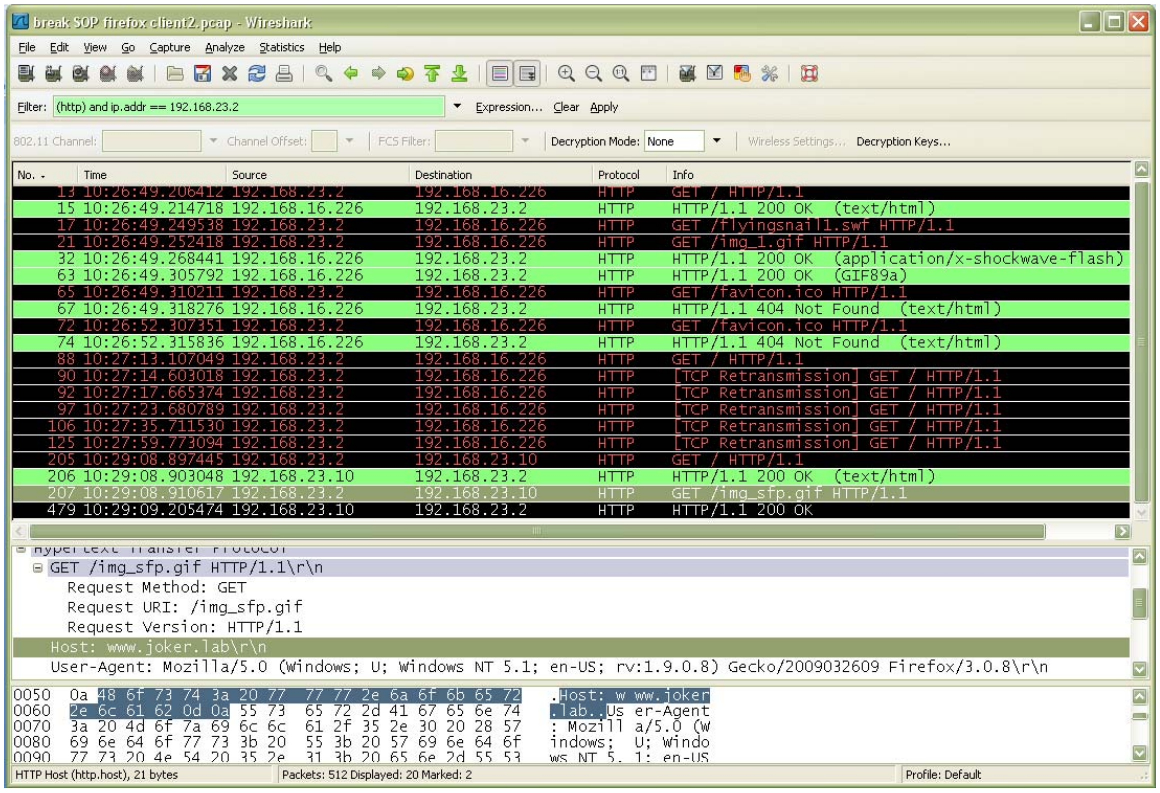
This is the internal's server web page.
This content is not supposed to be available
outside "angel.lab"

Είσαι στο μυαλό...
κάτι μαγικό...

Figure 13. Content from internal Web server: Violation of same origin policy

Tests were carried out using several different Web browsers on victim client 2. The results in Table 4 show that the same origin policy was successfully violated in every case, with only a small difference in the time interval dt. This means that by using DNS rebinding, Web browsers are trapped into thinking that 192.168.16.226 and 192.168.23.10 are the same origin, which is very important because the `XmlHttpRequest` object can only make same origin requests.

Finally, it is important to note that using victim client 2 there is no need to change the default configuration of the target DNS server by disabling the subnet mask ordering option. This client does not belong to the same subnet as the target server, so the default configuration does not prevent round-robin functionality. The experiments using victim client 2 were carried out using the default configuration in the target Windows 2003 Web server, which leads to the next important observation about Host Header Checking and its role in the mitigation of the same origin policy violation described above.


break SOP firefox client2.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Expression... Clear Apply
Wireless Settings... Decryption Keys...

Filter: (http) and ip.addr == 192.168.23.2

802.11 Channel: Channel Offset: FCS Filter: Decryption Mode: None

```
No. -  Time             Source          Destination     Protocol Info
   13 10:26:49.206412 192.168.23.2    192.168.16.226  HTTP     GET / HTTP/1.1
   15 10:26:49.214718 192.168.16.226  192.168.23.2    HTTP     HTTP/1.1 200 OK  (text/html)
   17 10:26:49.249538 192.168.23.2    192.168.16.226  HTTP     GET /flyingsnail1.swf HTTP/1.1
   21 10:26:49.252418 192.168.23.2    192.168.16.226  HTTP     GET /img_1.gif HTTP/1.1
   32 10:26:49.268441 192.168.16.226  192.168.23.2    HTTP     HTTP/1.1 200 OK  (application/x-shockwave-flash)
   63 10:26:49.305792 192.168.16.226  192.168.23.2    HTTP     HTTP/1.1 200 OK  (GIF89a)
   65 10:26:49.310211 192.168.23.2    192.168.16.226  HTTP     GET /favicon.ico HTTP/1.1
   67 10:26:49.318276 192.168.16.226  192.168.23.2    HTTP     HTTP/1.1 404 Not Found  (text/html)
   72 10:26:52.307351 192.168.23.2    192.168.16.226  HTTP     GET /favicon.ico HTTP/1.1
   74 10:26:52.315836 192.168.16.226  192.168.23.2    HTTP     HTTP/1.1 404 Not Found  (text/html)
   88 10:27:13.107049 192.168.23.2    192.168.16.226  HTTP     GET / HTTP/1.1
   90 10:27:14.603018 192.168.23.2    192.168.16.226  HTTP     [TCP Retransmission] GET / HTTP/1.1
   92 10:27:17.665374 192.168.23.2    192.168.16.226  HTTP     [TCP Retransmission] GET / HTTP/1.1
   97 10:27:23.680789 192.168.23.2    192.168.16.226  HTTP     [TCP Retransmission] GET / HTTP/1.1
  106 10:27:35.711530 192.168.23.2    192.168.16.226  HTTP     [TCP Retransmission] GET / HTTP/1.1
  125 10:27:59.773094 192.168.23.2    192.168.16.226  HTTP     [TCP Retransmission] GET / HTTP/1.1
  205 10:29:08.897445 192.168.23.2    192.168.23.10   HTTP     GET / HTTP/1.1
  206 10:29:08.903048 192.168.23.10   192.168.23.2    HTTP     HTTP/1.1 200 OK  (text/html)
  207 10:29:08.910617 192.168.23.2    192.168.23.10   HTTP     GET /img_sfp.gif HTTP/1.1
  479 10:29:09.205474 192.168.23.10   192.168.23.2    HTTP     HTTP/1.1 200 OK
```

```
⊟ Hypertext Transfer Protocol
  ⊟ GET /img_sfp.gif HTTP/1.1\r\n
      Request Method: GET
      Request URI: /img_sfp.gif
      Request Version: HTTP/1.1
    Host: www.joker.lab\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.8) Gecko/2009032609 Firefox/3.0.8\r\n
```

```
0050  0a 48 6f 73 74 3a 20 77  77 77 2e 6a 6f 6b 65 72   .Host: w ww.joker
0060  2e 6c 61 62 0d 0a 55 73  65 72 2d 41 67 65 6e 74   .lab..Us er-Agent
0070  3a 20 4d 6f 7a 69 6c 6c  61 2f 35 2e 30 20 28 57   : Mozill a/5.0 (W
0080  69 6e 64 6f 77 73 3b 20  55 3b 20 57 69 6e 64 6f   indows;  U; Windo
0090  77 73 20 4e 54 20 35 2e  31 3b 20 65 6e 2d 55 53   ws NT 5. 1; en-US
```

HTTP Host (http.host), 21 bytes | Packets: 512 Displayed: 20 Marked: 2 | Profile: Default

Figure 14. Wireshark snapshot confirming violation of same origin policy

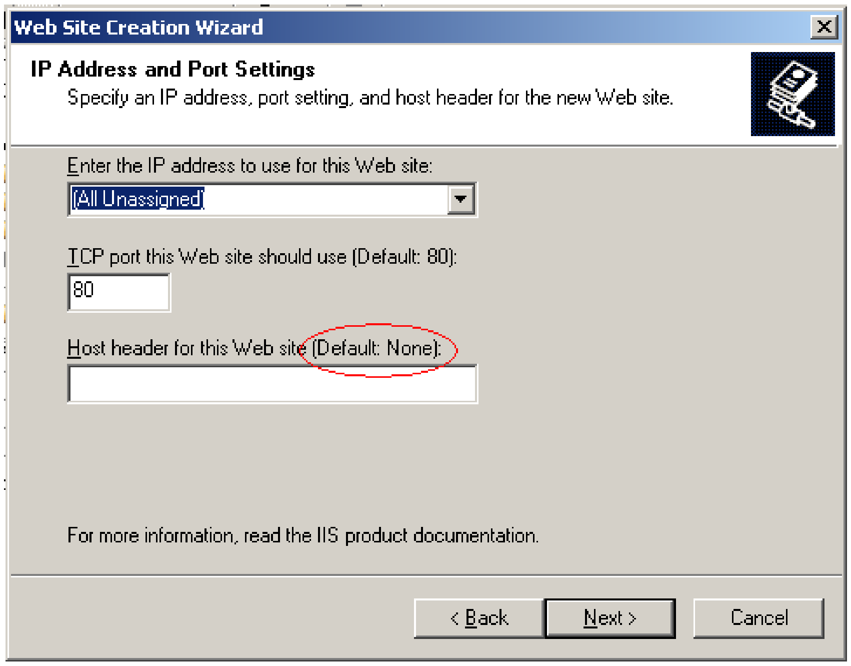
BREAKING SAME ORIGIN POLICY		
BROWSER TESTED	**RESULT**	**TIME NEEDED**
Google Chrome 1.0.154.65	Success	02:52 min
Safari 4 Public Beta (528.16)	Success	00:40 sec
Firefox 3.0.8	Success	01:55 min
Internet Explorer 7	Success	01:40 min
Internet Explorer 6	Success	01:37 min
Internet Explorer 5.5	Success	01:05 min
Internet Explorer 5.01	Success	02:18 min

Table 4. Summary of results with different Web browsers

e. *Host Header Checking*

The tests described above successfully led to the violation of the same origin policy because the target Web Server does not have any option except to permit the user’s browser to visit and retrieve the HTML content of its www.angel.lab page, although the browser asks for www.joker.lab, but with a different IP address. The target Web server does not have any clue that this query is a violation of same origin policy because the Web site has not any host name value to be compared with.

The creation of the target’s Web page was carried out using the default configuration steps shown in Figure 15, where the host name is optional. It means that by using IIS 6.0 of Web Server 2003, the administrator may create the www.angel.lab Web site without any host name or any notification that this configuration is vulnerable to DNS rebinding attacks. If the Web page has a host name, then the violation above is not possible because the Web server rejects any request that carries a host header that does not match the host name of the Web page it maintains.


Web Site Creation Wizard

IP Address and Port Settings

Specify an IP address, port setting, and host header for the new Web site.

Enter the IP address to use for this Web site:

(All Unassigned)

TCP port this Web site should use (Default: 80):

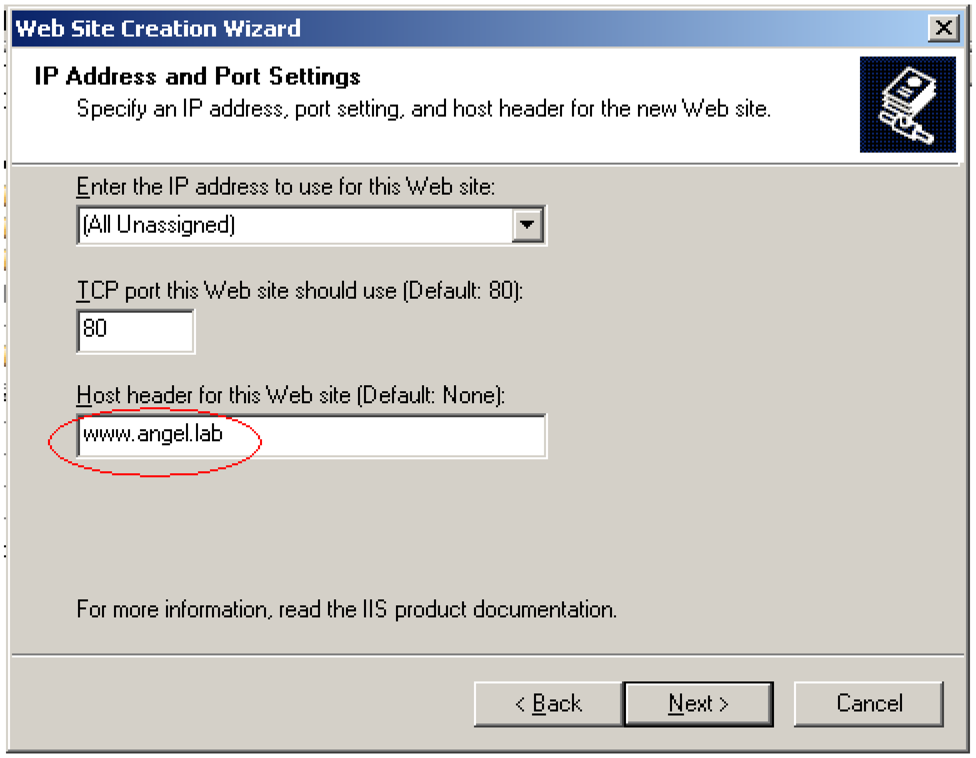
80

Host header for this Web site (Default: None):

For more information, read the IIS product documentation.

Figure 15. Default host header configuration for Web site www.angel.lab

In other words, when the victim client makes requests to the IP address 192.168.23.10 with the wrong host header www.joker.lab, the target Web server will compare that host name with its own host name, which is mapped to the Web page www.angel.lab and the IP address 192.168.23.10. The host header checking performed will result in dismissal of the request and thus the same origin policy will not be violated.


Web Site Creation Wizard

IP Address and Port Settings

Specify an IP address, port setting, and host header for the new Web site.

Enter the IP address to use for this Web site: (All Unassigned)

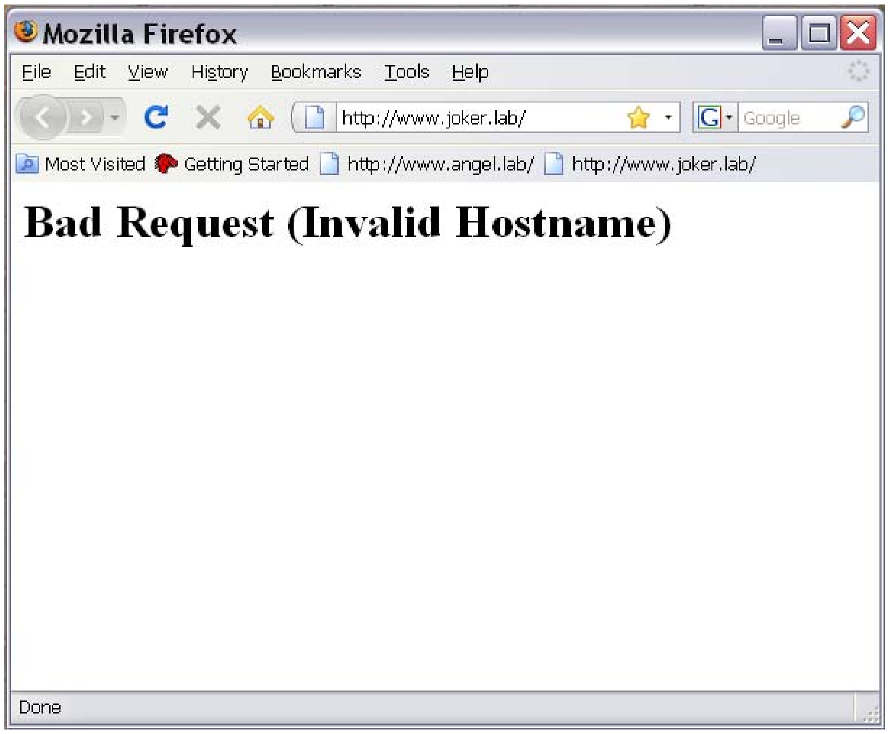
TCP port this Web site should use (Default: 80): 80

Host header for this Web site (Default: None): www.angel.lab

For more information, read the IIS product documentation.

Figure 16. Assigning a host name using the host header option

In order to verify the statement above, tests were again carried out, but this time the target's Web page www.angel.lab was created using a host name (www.angel.lab), shown in Figure 16. The procedure was exactly the same as before, but the results in this case were completely different. When victim client 2 makes the second request for www.joker.lab using the second IP address (192.168.23.10), which corresponds to the internal target Web server, access is prohibited, returning the message "Bad Request (Invalid Hostname)," as shown in Figure 17. The Wireshark capture of the communication between the victim client and the target server (Figure 18) clearly shows the wrong value in the host value (www.joker.lab) of the victim browser's request. It also shows the target server's response, which prevents the client browser from retrieving the Web page, because the host names do not match. This time the target Web server was able to perform host header checking, which is an efficient defense against the same origin policy violation.

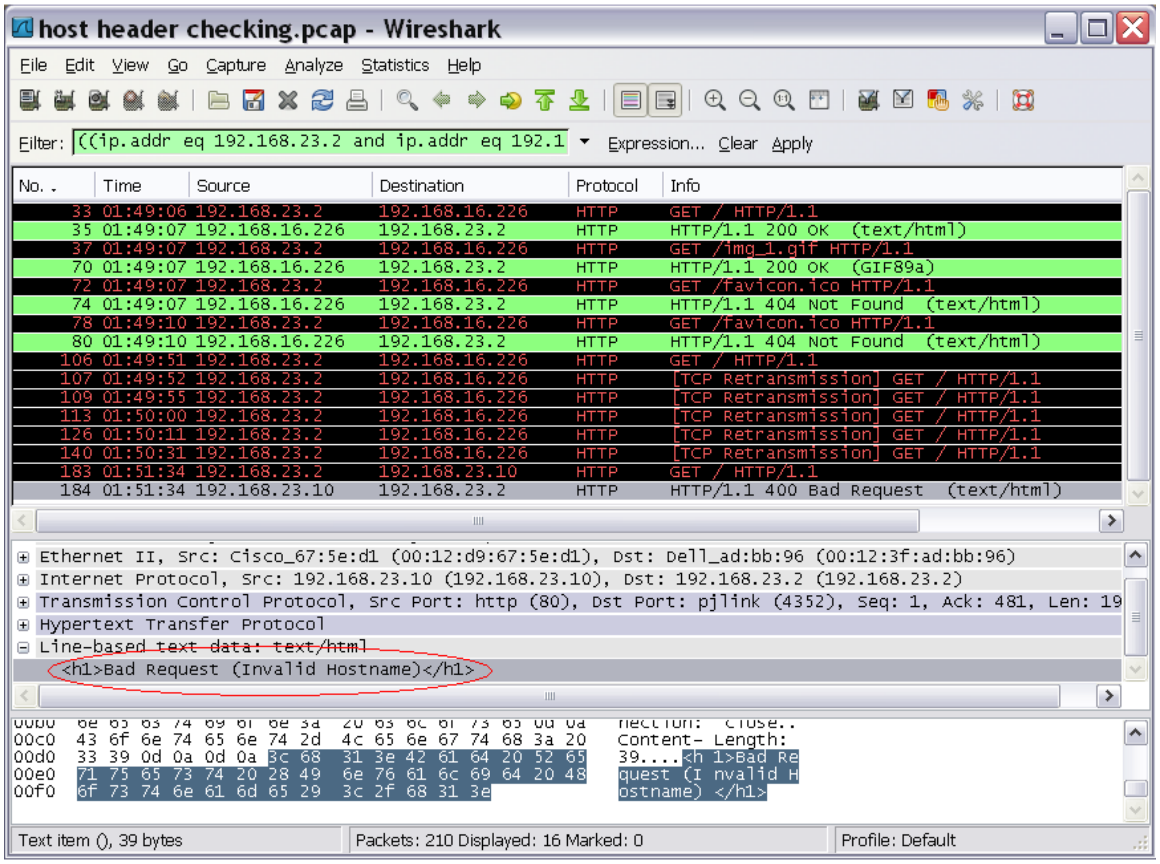


Bad Request (Invalid Hostname)

Figure 17. Message illustrating the failure of attack

The tests were performed using all the available Web browsers, as shown in Table 4. The results had no differences; in every case, the Web server rejects the browser's invalid request due to successful host header checking.

In conclusion, the DNS rebinding attack scenario that is described in this section can proceed if and only if the target Web server does not perform host header checking; this is very likely, because the host name value is optional by default in IIS 6.0 of Windows Server 2003 as well as in IIS 7.0 of Windows Server 2008, as was discovered during this research.


host header checking.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Expression... Clear Apply

```
Filter: ((ip.addr eq 192.168.23.2 and ip.addr eq 192.1

No. .  Time     Source          Destination     Protocol  Info
   33 01:49:06 192.168.23.2    192.168.16.226  HTTP      GET / HTTP/1.1
   35 01:49:07 192.168.16.226  192.168.23.2    HTTP      HTTP/1.1 200 OK  (text/html)
   37 01:49:07 192.168.23.2    192.168.16.226  HTTP      GET /img_1.gif HTTP/1.1
   70 01:49:07 192.168.16.226  192.168.23.2    HTTP      HTTP/1.1 200 OK  (GIF89a)
   72 01:49:07 192.168.23.2    192.168.16.226  HTTP      GET /favicon.ico HTTP/1.1
   74 01:49:07 192.168.16.226  192.168.23.2    HTTP      HTTP/1.1 404 Not Found  (text/html)
   78 01:49:10 192.168.23.2    192.168.16.226  HTTP      GET /favicon.ico HTTP/1.1
   80 01:49:10 192.168.16.226  192.168.23.2    HTTP      HTTP/1.1 404 Not Found  (text/html)
  106 01:49:51 192.168.23.2    192.168.16.226  HTTP      GET / HTTP/1.1
  107 01:49:52 192.168.23.2    192.168.16.226  HTTP      [TCP Retransmission] GET / HTTP/1.1
  109 01:49:55 192.168.23.2    192.168.16.226  HTTP      [TCP Retransmission] GET / HTTP/1.1
  113 01:50:00 192.168.23.2    192.168.16.226  HTTP      [TCP Retransmission] GET / HTTP/1.1
  126 01:50:11 192.168.23.2    192.168.16.226  HTTP      [TCP Retransmission] GET / HTTP/1.1
  140 01:50:31 192.168.23.2    192.168.16.226  HTTP      [TCP Retransmission] GET / HTTP/1.1
  183 01:51:34 192.168.23.2    192.168.23.10   HTTP      GET / HTTP/1.1
  184 01:51:34 192.168.23.10   192.168.23.2    HTTP      HTTP/1.1 400 Bad Request  (text/html)

⊞ Ethernet II, Src: Cisco_67:5e:d1 (00:12:d9:67:5e:d1), Dst: Dell_ad:bb:96 (00:12:3f:ad:bb:96)
⊞ Internet Protocol, Src: 192.168.23.10 (192.168.23.10), Dst: 192.168.23.2 (192.168.23.2)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: pjlink (4352), Seq: 1, Ack: 481, Len: 19
⊞ Hypertext Transfer Protocol
⊟ Line-based text data: text/html
    <h1>Bad Request (Invalid Hostname)</h1>

00b0  6e 65 63 74 69 6f 6e 3a  20 63 6c 6f 73 65 0d 0a   nection: close..
00c0  43 6f 6e 74 65 6e 74 2d  4c 65 6e 67 74 68 3a 20   Content- Length:
00d0  33 39 0d 0a 0d 0a 3c 68  31 3e 42 61 64 20 52 65   39....<h 1>Bad Re
00e0  71 75 65 73 74 20 28 49  6e 76 61 6c 69 64 20 48   quest (I nvalid H
00f0  6f 73 74 6e 61 6d 65 29  3c 2f 68 31 3e            ostname) </h1>

Text item (), 39 bytes    Packets: 210 Displayed: 16 Marked: 0    Profile: Default
```

Figure 18. Wireshark snapshot confirming successful host header checking

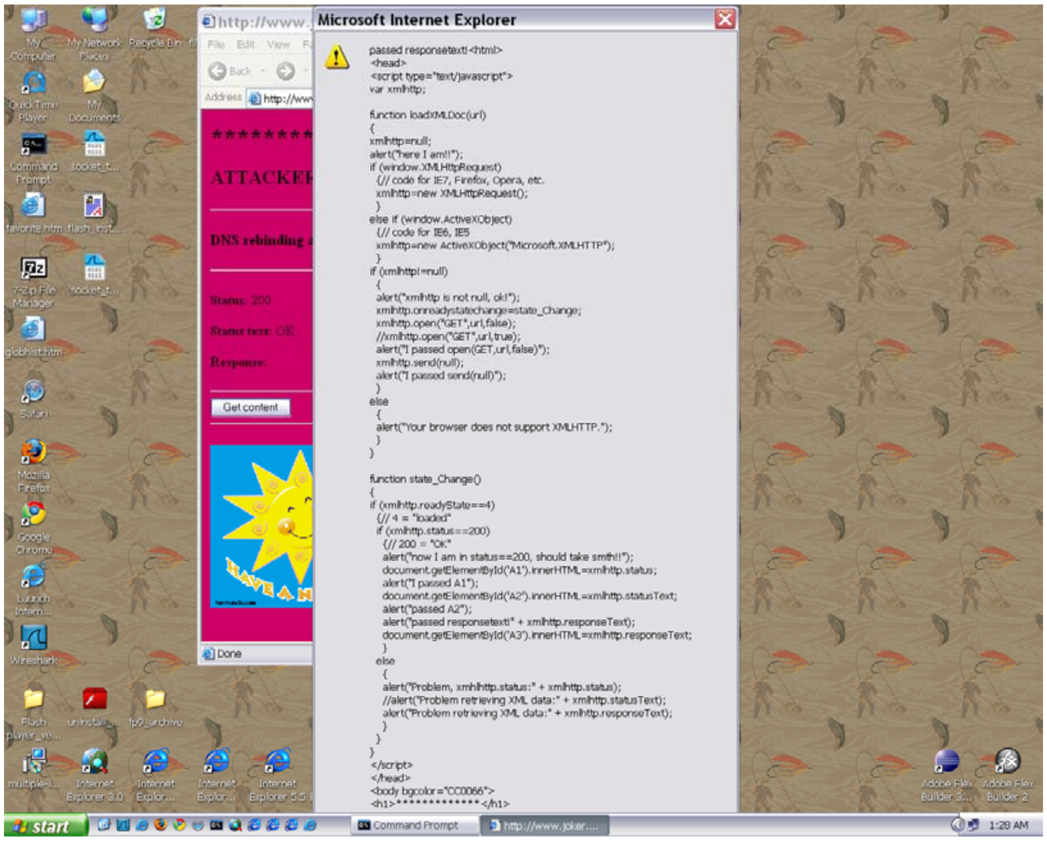
f. XmlHttpRequest

The `XmlHttpRequest` object is a mechanism that is used for reading HTML or XML documents. It is important that the `XmlHttpRequest` object can only make same origin requests. As discussed above, the attacker's goal is to read the HTML content of www.angel.lab*,* which he cannot do normally with HTML as he is firewalled. Instead, he tries to accomplish this through DNS rebinding and an `XmlHttpRequest` using victim client 2 as a proxy.

As an example of the `XmlHttpRequest` object, consider that the attacker's Web page www.joker.lab contains a script (written in JavaScript) where an XmlHttpRequest for http://www.joker.lab is executed*.* Figure 19 shows the result, which is the HTML content of that Web page. (The code is based on examples found at w3schools.com [25], and is shown in Appendix C.) Assuming that the attack process has reached the point that victim client 2 retrieves the HTML content of the target server's www.angel.lab Web page (time t4), thus violating the same origin policy described above, then the `XmlHttpRequest` for http:// www.joker.lab will result in the HTML content of the target rather than the attacker's page, because the same origin policy has been violated. In this final phase of the scenario, the data retrieved can be sent back to the attacker's domain using the appropriate malicious script inside attacker's Web page HTML code. According to Collin Jackson, a script like that shown in Table 5 could be used successfully to send data back to the attacker's corporate Web server.

In conclusion, tests have shown that by using DNS rebinding, the same origin policy can be violated in all major browsers with the following conditions:

- The victim client belongs to a different subnet than the target server
- The target's Web server does not perform host header checking
- The attacker's page contains a malicious script that uses `XmlHttpRequest` to connect to attacker's www.joker.labthat can be resolved to the target server's IP address, which the browser thinks belongs to the same origin.



```
passed responsetext! <html>
<head>
<script type="text/javascript">
var xmlhttp;

function loadXMLDoc(url)
{
xmlhttp=null;
alert("here I am!!");
if (window.XMLHttpRequest)
  {// code for IE7, Firefox, Opera, etc.
  xmlhttp=new XMLHttpRequest();
  }
else if (window.ActiveXObject)
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
if (xmlhttp!=null)
  {
  alert("xmlhttp is not null, ok!");
  xmlhttp.onreadystatechange=state_Change;
  xmlhttp.open("GET",url,false);
  //xmlhttp.open("GET",url,true);
  alert("I passed open(GET,url,false)");
  xmlhttp.send(null);
  alert("I passed send(null)");
  }
else
  {
  alert("Your browser does not support XMLHTTP.");
  }
}

function state_Change()
{
if (xmlhttp.readyState==4)
  {// 4 = "loaded"
  if (xmlhttp.status==200)
    {// 200 = "OK"
    alert("now I am in status==200, should take smth!!");
    document.getElementById('A1').innerHTML=xmlhttp.status;
    alert("I passed A1");
    document.getElementById('A2').innerHTML=xmlhttp.statusText;
    alert("passed A2");
    alert("passed responsetext!" + xmlhttp.responseText);
    document.getElementById('A3').innerHTML=xmlhttp.responseText;
    }
  else
    {
    alert("Problem, xmhlhttp.status:" + xmlhttp.status);
    //alert("Problem retrieving XML data:" + xmlhttp.statusText);
    alert("Problem retrieving XML data:" + xmlhttp.responseText);
    }
  }
}
</script>
</head>
<body bgcolor="CC0066">
<h1>*************</h1>
```

Figure 19. `XmlHttpRequest` script embedded in attacker's Web page

```
<form action='http://www.helper.lab/'><input type='hidden'
value='secret stolen info'></form>
<script>document.forms[0].submit()</script>
```

Table 5. Example code segment for sending to attacker

D. DNS REBINDING ATTACKS VIA ADOBE FLASH PLAYER

This section focuses on DNS rebinding vulnerabilities in Adobe Flash Player. As already discussed in Chapter II, Adobe Flash Player 9.0.48.0 is vulnerable to these attacks because it does not perform any DNS pinning, nor does it use a browser's pinning [16]. Although in recent years many researchers have discussed DNS rebinding attacks, there are only a few proofs-of-concept focusing on Adobe Flash Player applications. This author found Tadashi Kanatoko Satoh's experiments on jumperz.net [18] and the Stanford security team's research [3] to be the most important in this respect. Additionally, this section provides information about the security updates that have been made by Adobe in order to mitigate these attacks. Finally, the section describes the tests that were carried out during this research that verify the changes in the Flash Player's security policy.

1. Satoh's DNS Rebinding Attack via Adobe Flash Player 9

In 2007, Tadashi Kanatoko Satoh uploaded a demo on his Web site, www.jumperz.net [18], which showed that Flash Player 9 was affected by DNS rebinding attacks. With DNS rebinding, his demo broke the same origin policy. He is quite possibly the first researcher who noticed that not only JavaScript, but also Flash Player and the Java Applet, are affected by these attacks because most of the studies on the subject were made later and include Satoh's attack as a reference.

Satoh took advantage of the `Socket` class that was introduced in Flash Player version 9.0 or higher. The class allows Flash applications to make socket connections, therefore reading and writing data, which is useful for working with servers that use binary data. However, that is a functionality that can be exploited by attackers [18]. As Satoh states, with DNS rebinding and socket connections, the attacker can perform several malicious actions, including:

- Scan any IP addresses and any ports on Intranets or Internet.
- Make the user's browser send shell codes to any hosts.
- Make the user's browser send spam emails.
- Use the user's browser as a proxy.
- Break any IP address based authentication.
- Exploit protocols other than HTTP [18].

More specifically, Satoh's demo contains a Flash application, which is embedded in his Web page plays the role of the attacker's Web site. The Flash application executes a port scanning to a given IP address, gets banners, and sends the extracted data back to the attacker's Web site. Table 6 outlines the browsers and operating systems that were successfully tested in 2007.

SATOH'S TESTING RESULTS		
BROWSER TESTED	**OS**	**RESULTS**
Internet Explorer 6.0	Windows XP / SP2	success
Internet Explorer 6.0	Windows2000 / SP4	success
Firefox 2.0	Windows2000 / SP4	success
Firefox 2.0	Windows XP / SP2	success
Netscape 8.2.1	Windows XP / SP2	success
Opera 9.0.2	Windows 2000 / SP4	success

Table 6. Satoh's attack results using Adobe Flash Player [From 18]

It is important to note that Satoh's Web site is the only source available to the public that not only discusses the DNS rebinding issue, but also provides an online demonstration of the attack using three different methods, including the Flash Player usage that was presented in this section. Moreover, the source code of the Flash application that was used for the attack is provided on his Web site. The author attempted to communicate with Satoh in order to obtain more information about DNS rebinding and Flash socket connections, especially about the source code used, but received no response. For instance, there is no information about the exact Flash Player version used for his tests, what kind of server he used, or when his attack became obsolete.

2. Stanford Security Team's DNS Rebinding in Adobe Flash Player 9.0.48.0

The Stanford security team was credited with the discovery of Adobe Flash Player's DNS rebinding vulnerability [26]. The National Vulnerability

Database contains this vulnerability with the identifier CVE-2007-5275 in the Common Vulnerabilities and Exposures system (CVE), with the following description [27]:

> The Adobe Macromedia Flash 9 plug-in allows remote attackers to cause a victim machine to establish TCP sessions with arbitrary hosts via a Flash (SWF) movie, related to lack of pinning of a hostname to a single IP address after receiving an allow-access-from element in a cross-domain-policy XML document, and the availability of a Flash Socket class that does not use the browser's DNS pins, aka DNS rebinding attacks, a different issue than CVE-2002-1467 and CVE-2007-4324 [27].

Briefly describing their method, they developed proof-of-concept exploits for DNS rebinding vulnerabilities in Flash 9, `LiveConnect`, Java applets with proxy servers, and the browser itself. Their system consisted of an authoritative DNS server for their domain, a Flash policy server, and a standard Apache Web server. They tested DNS rebinding experimentally, by running a Flash advertisement on a small advertising network. The Flash advertisement exploited the vulnerability, described in the previous chapter about Flash 9.0.48.0, in order to load an XML document from the target server. The attack required the client to visit the attacker's Web page and view the Flash application, which was an advertisement. In describing the results of the attack [1], the security team wrote:

> We ran the ad beginning at midnight EDT on three successive nights in late April 2007. Our experimental results show that DNS rebinding vulnerabilities are widespread and cost effective to exploit on a large scale [1].

The Stanford security team published the results of their experiments in DNS rebinding attacks in 2007 [1]. As they include Satoh's proof-of-concept in their references, it is clear that their attack was implemented later than Satoh's. Their experiments were documented, published, and presented to the 14th ACM Conference on Computer and Communication Security (CCS 2007) [28]. Their research proved the existence and exploitation of DNS rebinding vulnerabilities using Flash Player 9.0.48.0 and led Adobe to develop security updates in order to mitigate DNS rebinding attacks, as described in the following section.

3. Adobe Security Updates for DNS Rebinding

The DNS rebinding vulnerability concerning Flash Player 9 was a security hole that was finally fixed by Adobe in November 2008 when version 9.0.151.0 was released. In its APSB08-20 security bulletin [29], Adobe characterized that update as critical for mitigating security issues that included DNS rebinding. The company suggested that the users of Flash Player 9.0.124.0 and earlier versions upgrade to the newest version 10.0.12.36. However, for users that were not able to do so, Adobe developed a patched version of Flash Player 9, Flash Player 9.0.151.0. All Flash Player 9 versions, sorted by order of release with the necessary information about the DNS rebinding vulnerability, are shown in Table 7.

According to Adobe [30], Flash Player relies on browsers to provide HTTP networking, so any DNS rebinding vulnerabilities that involve only HTTP must be solved by browsers. However, Flash Player also provides socket level networking (via the ActionScript `Socket` and `XMLSocket` classes), which may be exploited in DNS rebinding attacks, as Satoh and the Stanford security team researchers proved in their experiments.

For this reason, Adobe applied several changes beginning with Flash 9.0.115.0 in order to mitigate DNS rebinding attacks, which can be summarized as follows:

- Strict policy file rules that always require permission from a socket policy file in order to make a socket connection, even when the socket server appears to be the same as a connecting SWF file's domain of origin.
- In addition, beginning with version 9.0.115.0, Flash Player matches socket connections to their corresponding socket policy files based on IP addresses, not just domain names.
- Detailed record policy file events from debug versions of Flash Player. All failures and successes in loading and processing policy

files are reported, as well as failures and successes of operations that depend on those policy files. This should help not only with the transition to the stricter rules, but also generally in finding and solving problems with any policy file deployment.

- A fixed socket master policy file port, TCP port 843, which is assigned by default for socket policy files. This provides a standard way to serve socket policy files, in contrast with the random port that was used previously.
- An option to require strong client authentication for local sockets. Socket policy files served from localhost sockets may now specify that only HTTPS Flash applications from given domains may connect, using the `secure="true"` declaration previously reserved only for HTTPS policy files. This can help secure hybrid applications that combine online Flash content with local applications [30].

In conclusion, it is important to note that Adobe clearly states that all of the above mechanisms were developed and applied in three phases. These developments began with 9.0.115.0 and were extended in 9.0.124.0, but were only fully implemented in version 9.0.151.0 [30]. As a result, the first Flash Player version 9 that is considered patched against DNS rebinding attacks is 9.0.151.0 [31] as shown in Table 7. It is interesting to note that it took more than one year for Adobe to fix this security hole in Flash Player.

Flash Player 9 versions		
Version	**Operating System**	**DNS rebinding**
9.0.16.0	Windows, Mac OS	vulnerable
9.0.20.0	Mac OS	vulnerable
9.0.28.0	Windows, Mac OS	vulnerable
9.0.31.0	Linux	vulnerable
9.0.45.0	Windows, Mac OS	vulnerable
9.0.47.0	Windows, Mac OS, Solaris	vulnerable
9.0.48.0	Linux	vulnerable
9.0.115.0	Windows, Mac OS, Solaris, Linux	vulnerable
9.0.124.0	Windows, Mac OS, Solaris, Linux	vulnerable
9.0.125.0	Solaris	vulnerable
9.0.151.0	Windows, Mac OS, Solaris, Linux	Fixed
9.0.152.0	Linux	Fixed
9.0.159.0	Windows, Mac OS, Solaris, Linux	Fixed

Table 7. Vulnerability of Adobe Flash Player 9 versions to DNS rebinding attacks [From 30]

4. Testing Flash Socket Application and Policy File Server

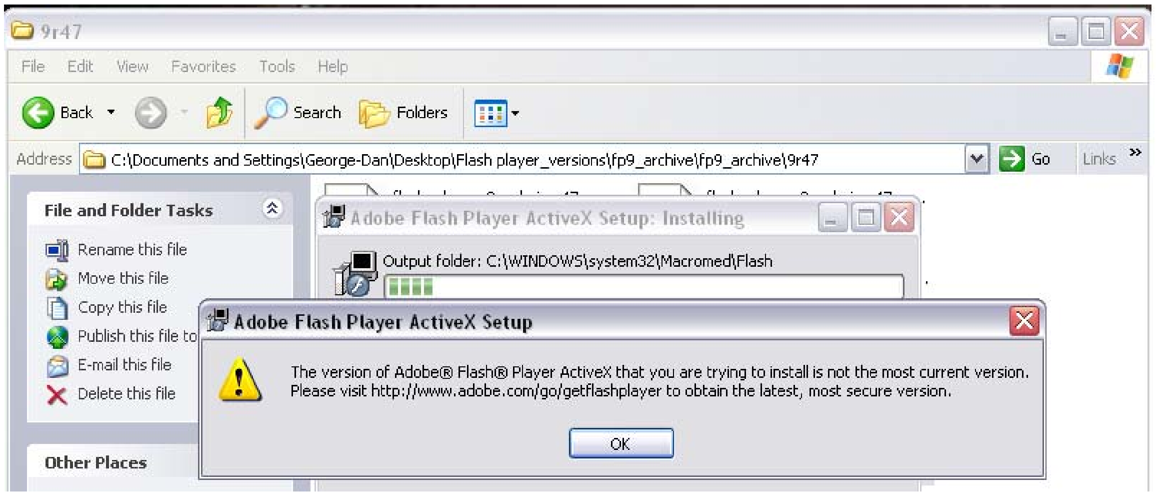
This last section describes the experiments that were carried out during this research in order to verify the difference in behavior between the vulnerable Flash Player 9 versions and the current Flash version 10.0.22.87 [32], which is compatible with all operating systems and in which all security updates have been applied in to order to mitigate DNS rebinding vulnerabilities.

All tests were carried out using the same testbed shown in Figure 6 and described above, as well as the Web browsers used for previous tests shown in Table 4. The tests were limited to the use of Windows XP SP2 in clients and Windows Server 2003 in servers.

a. *Archived Flash Player Versions*

The testbed uses the archived Flash Players that Adobe has made available for testing purposes. These archived versions of Flash Player are provided specifically for Flash developers who are accessing their sites from the perspective of users with different versions of Flash Player. Thus, the testbed uses Flash Player 9 versions as shown in Table 7, which were downloaded from Adobe TechNote 14266 [33].

On the other hand, there is a security restriction when installing Flash Player in a Windows XP SP2 machine, as was observed during tests. If there was a previously installed version of Flash Player, the user was prevented from installing an earlier version. For instance, if the user installs Flash Player 9.0.151.0 and then attempts to install version 9.0.48.0, then the installation fails and an error message is displayed, (see Figure 20).



C:\Documents and Settings\George-Dan\Desktop\Flash player_versions\fp9_archive\fp9_archive\9r47

Output folder: C:\WINDOWS\system32\Macromed\Flash

The version of Adobe® Flash® Player ActiveX that you are trying to install is not the most current version. Please visit http://www.adobe.com/go/getflashplayer to obtain the latest, most secure version.

Figure 20. Flash Player installation error message

According to the corresponding Adobe TechNote (kb402435) [34], this issue is due to registry key settings applied by the Flash Player security model. The restriction is intended to prevent users from downgrading minor Flash Player versions. The same TechNote provides the solution to this problem, which is the use of an uninstaller for Flash Player [34]. The procedure to be followed is presented in Table 8; however, it is important to note that the successful process of downgrading Flash Player also requires the user to restart the computer before attempting to install a different version, probably due to the need for the computer to apply changes to the registry.

1. Click Start > Run.
2. Type cmd in the Open box, and then press Enter.
3. In the command window, type the following:

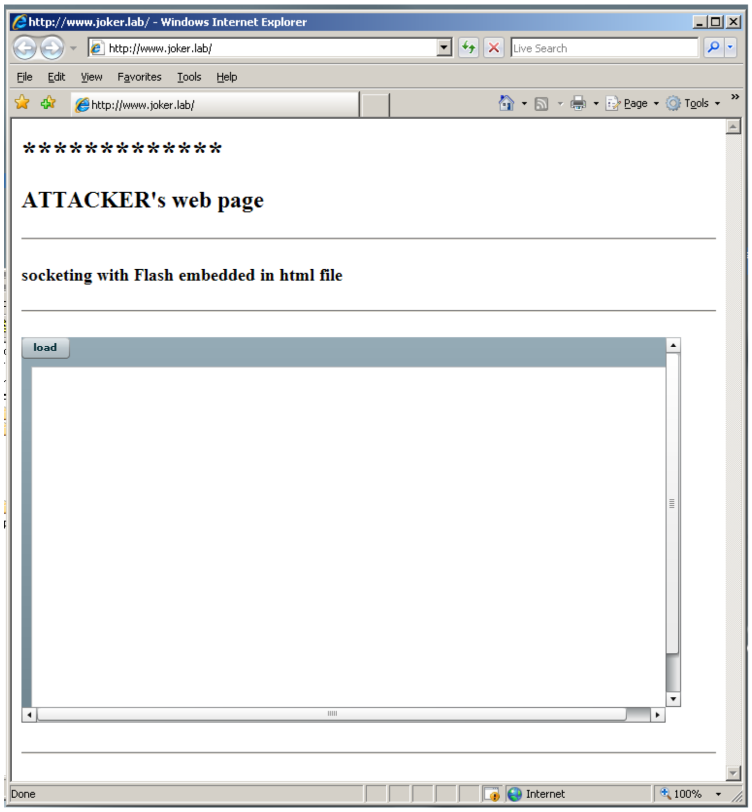
   ```
   uninstall_flash_player.exe /clean
   ```
4. Restart the computer

Table 8. Procedure for downgrading Adobe Flash Player [After 34]

b. Socket Flash Application Embedded in HTML

The HTML code that was developed for the www.joker.lab Web page, as well as the source code for the Flash application, is shown in Appendix D. The HTML code demonstrates how a Flash application can be embedded in the HTML source code of any Web page.

The Flash application source code was developed using the Adobe Flex 3 tool; specifically, the version that works as a plug-in of the Eclipse IDE. The application uses the `Socket` class that Flash provides in order to open a socket connection between the client and the server. The application is connected with an active button (LOAD). When the client who visits www.joker.lab presses the button, the application executes opening a socket between the client and the server. The Web page is shown in Figure 21. The application executes a GET command to the root directory of the Web server in order to retrieve the HTML content of the Web page. Using the functions `writeUTFBytes` and `readUTFBytes`, included in `Socket` class, the HTML file is presented in the designated display area. In the case that a security error occurs, the application detects it and throws an error on the display area using the event listeners contained in the `Socket` class, such as `SecurityErrorEvent.SECURITY_ERROR`, which throws a corresponding message in the display area when such an error occurs.


http://www.joker.lab/ - Windows Internet Explorer
http://www.joker.lab/
Live Search
File Edit View Favorites Tools Help
Page Tools
load
Done
Internet
100%

****

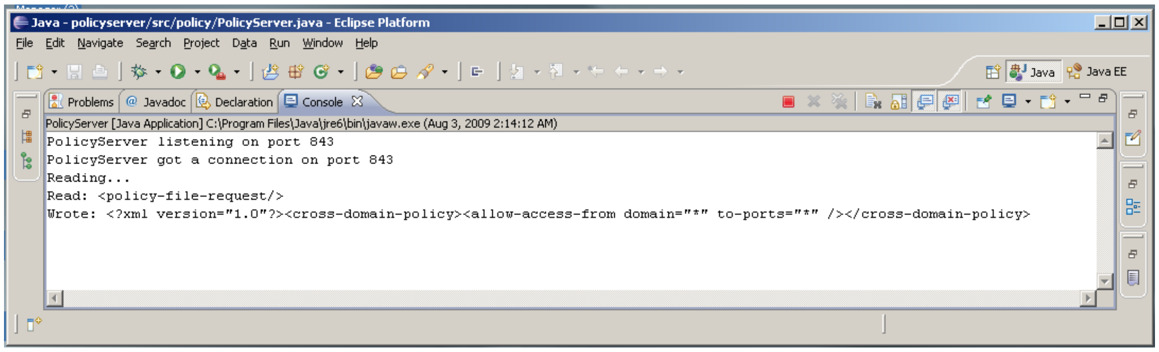
ATTACKER's web page

socketing with Flash embedded in html file

Figure 21. www.joker.lab Web page with Flash application embedded

c. *Flash Policy Server in Java*

The server that hosts the Web page also runs a Flash policy server that listens to port 843; this is the designated port for this purpose. The source code for the policy server is based on the code that was developed by Thomas Meyer [35]. The code was modified so that it serves a cross-domain policy, permitting connections from all domains on all ports as shown in Table 2. Figure 22 shows the Flash policy server in action as it listens to port 843 and responds to a client's request for a policy file.


Java - policyserver/src/policy/PolicyServer.java - Eclipse Platform
File Edit Navigate Search Project Data Run Window Help
Java
Java EE
Problems
Javadoc
Declaration
Console

```
PolicyServer [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Aug 3, 2009 2:14:12 AM)
PolicyServer listening on port 843
PolicyServer got a connection on port 843
Reading...
Read: <policy-file-request/>
Wrote: <?xml version="1.0"?><cross-domain-policy><allow-access-from domain="*" to-ports="*" /></cross-domain-policy>
```

Figure 22. Flash policy server in action running in Eclipse

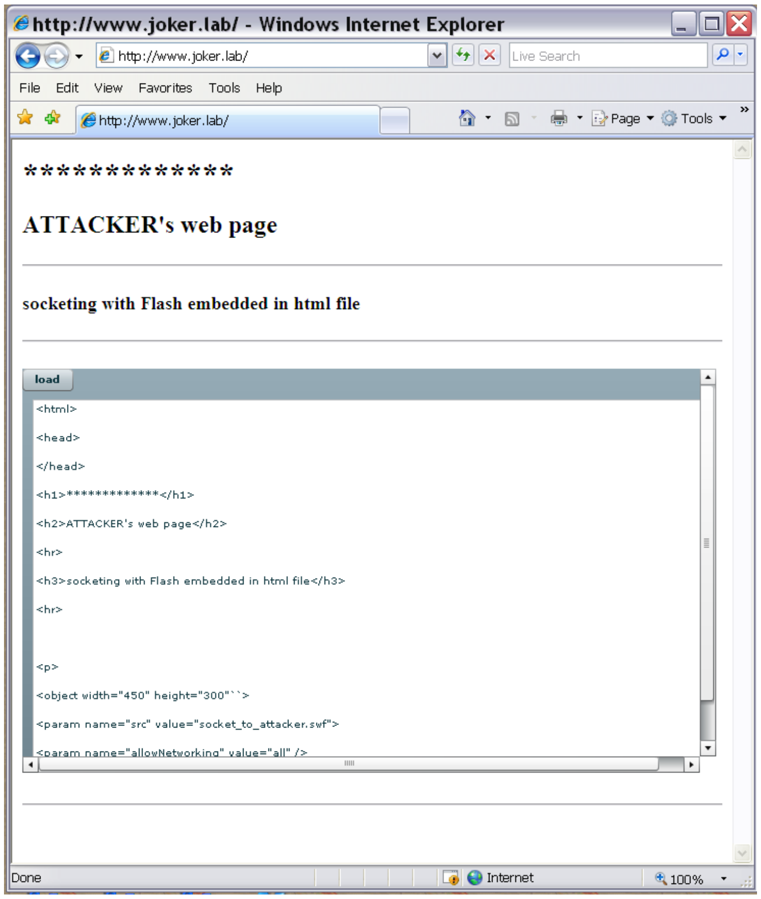
d. Testing–Results

Tests were carried out using the testbed's clients who visited www.joker.lab Web page using different Web browsers and Flash Player 9 versions. It was also used the latest Flash Player version, 10.0.22.87 [32]. Again, the Wireshark network protocol analyzer was used to record and observe the client server communication in the various cases.

First, the tests showed that some Web browsers do not accept the manually installed Flash versions. More specifically, Mozilla Firefox 3.0.8, Google Chrome 1.0.154.65, and Safari 4 Public Beta (528.16) do not accept any manual installation of the Flash Player plug-in. They require a download of the missing plug-in from the official Adobe site in order to display the www.joker.lab Web page and run the Flash code. Adobe responds with the latest version, which means that the above-listed Web browsers contain an additional security feature; this is in contrast with Internet Explorer Web browsers, which accept every Flash Player version 9 installed in the client Windows XP machine following the procedure described above. For this reason, tests were carried out with various versions of Internet Explorer.

Second, the Flash application cannot execute when a client uses Flash Player versions previous to 9.0.115.0. This result is reasonable because the testbed uses the policy server to listen to the fixed TCP port 843 in order to provide the `crossdomain.xml` policy. That port was assigned by default for socket policy files beginning with Flash Player 9.0.115.0.

Third, the client can normally execute the Flash application using all available Internet Explorer versions (7, 6, 5.01, and 5.5) in parallel with the Flash Player versions 9.0.115.0 and 9.0.124.0. Figure 23 shows the output, which is the HTML code of www.joker.lab that is retrieved from the Web server when Flash code executes the `GET \/index.html\n\n` command as shown in Appendix E. This Figure shows a test where the client uses Internet Explorer 7 and the Flash plug-in that is installed with version 9.0.115.



http://www.joker.lab/ - Windows Internet Explorer
http://www.joker.lab/
Live Search
File Edit View Favorites Tools Help
http://www.joker.lab/
Page Tools
load
Done
Internet
100%

****

ATTACKER's web page

socketing with Flash embedded in html file

```
<html>

<head>

</head>

<h1>*************</h1>

<h2>ATTACKER's web page</h2>

<hr>

<h3>socketing with Flash embedded in html file</h3>

<hr>



<p>

<object width="450" height="300"``>

<param name="src" value="socket_to_attacker.swf">

<param name="allowNetworking" value="all" />
```

Figure 23. Flash application normal execution

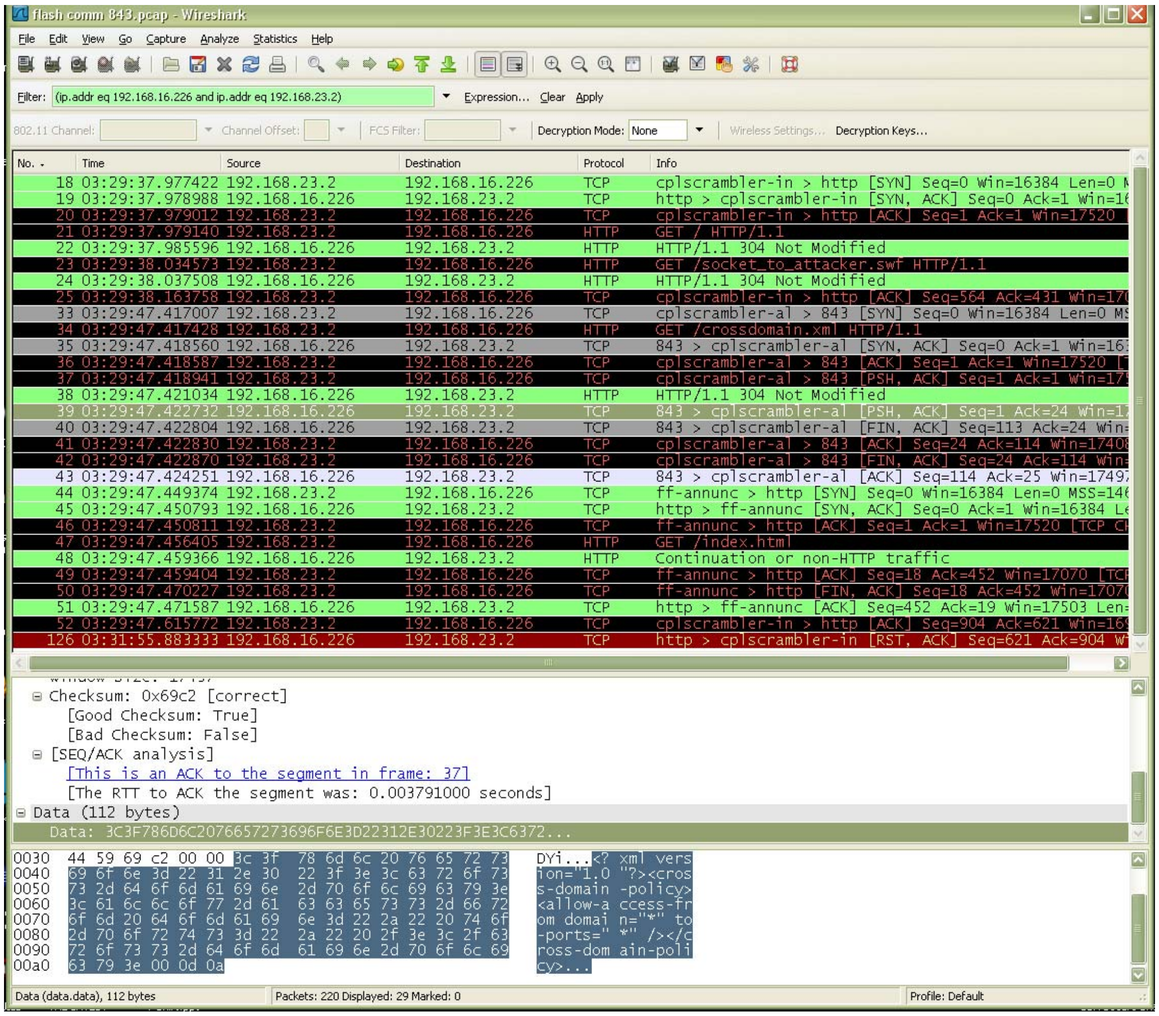
The communication between client and server during this procedure is shown in Figure 24, where the client (192.168.23.2) asks for a policy file using port 843 immediately after the user attempts to execute the Flash application by pressing the active button. The server then responds in the same port with the policy file and the client accepts and executes the code.

Figure 25 shows specifically the TCP flow of packets between client and server, which occurs between a client’s port 1088 (above 1024) and the server’s port 843. This stream contains the policy file that the client receives, which permits the successful execution of the embedded Flash application.

However, when a client uses Flash Player 10.0.22.87, a security error is displayed, as shown in Figure 26. This alert is thrown by the security checking that the application performs, and is a result of the security updates that have been developed by Adobe in order to mitigate Flash Player vulnerabilities, including those that lead to DNS rebinding attacks.

In conclusion, Adobe has fixed the security hole created by the DNS rebinding vulnerability; however, the time needed was more than one year because the vulnerability was discovered in Flash 9.0.48.0 in 2007 and it wasn’t fixed until the release of Flash 9.0.151.0 in 2008. In the meantime, versions 9.0.115.0 and 9.0.124.0 were still vulnerable to DNS rebinding attacks. Adobe finally applied all the mechanisms described above, which led to the successful mitigation of these attacks.

It is important to note that during the research, this author communicated several times with Collin Jackson [36], the member of the Stanford security team who is credited with the discovery and exploitation of the DNS rebinding attacks. Jackson provided useful information about the subject of this research, most importantly the confirmation that Adobe has indeed fixed the DNS rebinding issue beginning with Flash 9.0.151.0. Users have to use the latest Flash Player version (10.0.22.87 in July 2009), or Flash 9.0.151.0 or above if there is a need for Flash 9.



flash comm 843.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Expression... Clear Apply
802.11 Channel: Channel Offset: FCS Filter:
Wireless Settings... Decryption Keys...

Filter: (ip.addr eq 192.168.16.226 and ip.addr eq 192.168.23.2)

Decryption Mode: None

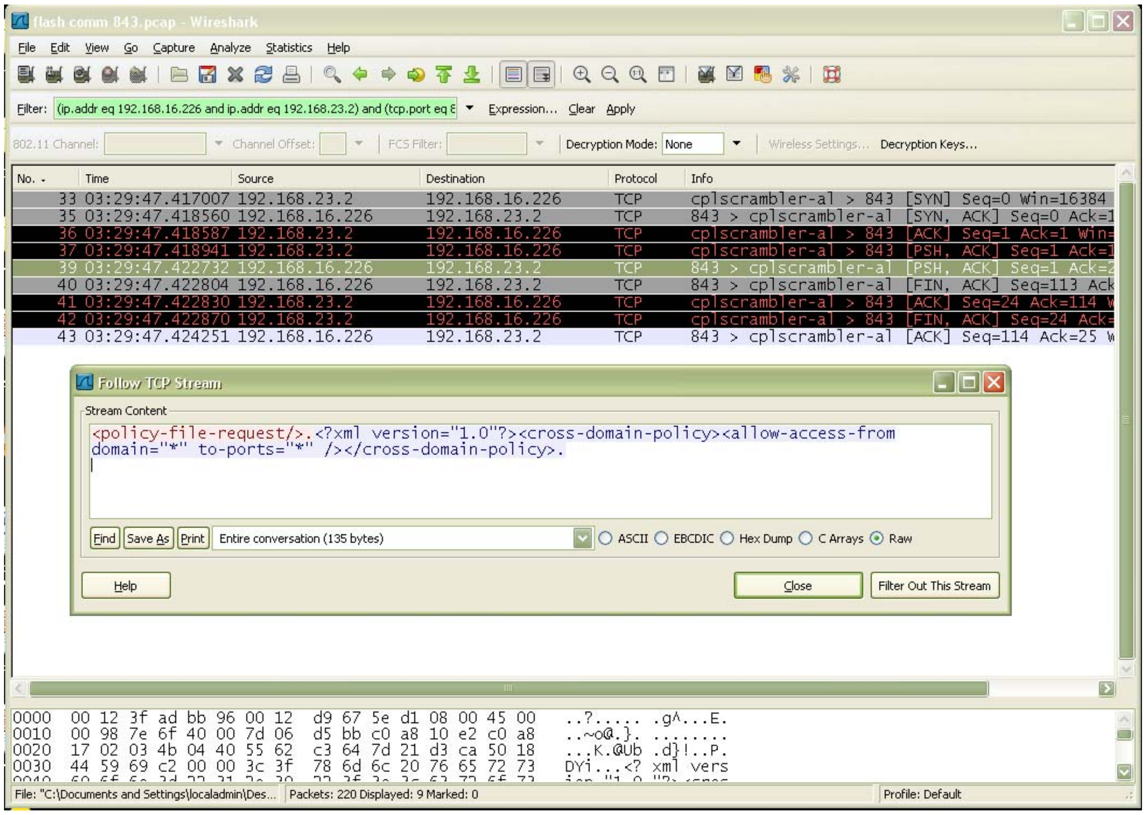
```
No.  Time             Source          Destination     Protocol  Info
 18 03:29:37.977422 192.168.23.2    192.168.16.226  TCP       cplscrambler-in > http [SYN] Seq=0 Win=16384 Len=0 M
 19 03:29:37.978988 192.168.16.226  192.168.23.2    TCP       http > cplscrambler-in [SYN, ACK] Seq=0 Ack=1 Win=16
 20 03:29:37.979012 192.168.23.2    192.168.16.226  TCP       cplscrambler-in > http [ACK] Seq=1 Ack=1 Win=17520
 21 03:29:37.979140 192.168.23.2    192.168.16.226  HTTP      GET / HTTP/1.1
 22 03:29:37.985596 192.168.16.226  192.168.23.2    HTTP      HTTP/1.1 304 Not Modified
 23 03:29:38.034573 192.168.23.2    192.168.16.226  HTTP      GET /socket_to_attacker.swf HTTP/1.1
 24 03:29:38.037508 192.168.16.226  192.168.23.2    HTTP      HTTP/1.1 304 Not Modified
 25 03:29:38.163758 192.168.23.2    192.168.16.226  TCP       cplscrambler-in > http [ACK] Seq=564 Ack=431 Win=17
 33 03:29:47.417007 192.168.23.2    192.168.16.226  TCP       cplscrambler-al > 843 [SYN] Seq=0 Win=16384 Len=0 MS
 34 03:29:47.417428 192.168.23.2    192.168.16.226  HTTP      GET /crossdomain.xml HTTP/1.1
 35 03:29:47.418560 192.168.16.226  192.168.23.2    TCP       843 > cplscrambler-al [SYN, ACK] Seq=0 Ack=1 Win=16
 36 03:29:47.418587 192.168.23.2    192.168.16.226  TCP       cplscrambler-al > 843 [ACK] Seq=1 Ack=1 Win=17520 [
 37 03:29:47.418941 192.168.23.2    192.168.16.226  TCP       cplscrambler-al > 843 [PSH, ACK] Seq=1 Ack=1 Win=17
 38 03:29:47.421034 192.168.16.226  192.168.23.2    HTTP      HTTP/1.1 304 Not Modified
 39 03:29:47.422732 192.168.16.226  192.168.23.2    TCP       843 > cplscrambler-al [PSH, ACK] Seq=1 Ack=24 Win=1
 40 03:29:47.422804 192.168.16.226  192.168.23.2    TCP       843 > cplscrambler-al [FIN, ACK] Seq=113 Ack=24 Win=
 41 03:29:47.422830 192.168.23.2    192.168.16.226  TCP       cplscrambler-al > 843 [ACK] Seq=24 Ack=114 Win=1740
 42 03:29:47.422870 192.168.23.2    192.168.16.226  TCP       cplscrambler-al > 843 [FIN, ACK] Seq=24 Ack=114 Win=
 43 03:29:47.424251 192.168.16.226  192.168.23.2    TCP       843 > cplscrambler-al [ACK] Seq=114 Ack=25 Win=1749
 44 03:29:47.449374 192.168.23.2    192.168.16.226  TCP       ff-annunc > http [SYN] Seq=0 Win=16384 Len=0 MSS=146
 45 03:29:47.450793 192.168.16.226  192.168.23.2    TCP       http > ff-annunc [SYN, ACK] Seq=0 Ack=1 Win=16384 Le
 46 03:29:47.450811 192.168.23.2    192.168.16.226  TCP       ff-annunc > http [ACK] Seq=1 Ack=1 Win=17520 [TCP CH
 47 03:29:47.456405 192.168.23.2    192.168.16.226  HTTP      GET /index.html
 48 03:29:47.459366 192.168.16.226  192.168.23.2    HTTP      Continuation or non-HTTP traffic
 49 03:29:47.459404 192.168.23.2    192.168.16.226  TCP       ff-annunc > http [ACK] Seq=18 Ack=452 Win=17070 [TCP
 50 03:29:47.470227 192.168.23.2    192.168.16.226  TCP       ff-annunc > http [FIN, ACK] Seq=18 Ack=452 Win=1707
 51 03:29:47.471587 192.168.16.226  192.168.23.2    TCP       http > ff-annunc [ACK] Seq=452 Ack=19 Win=17503 Len=
 52 03:29:47.615772 192.168.23.2    192.168.16.226  TCP       cplscrambler-in > http [ACK] Seq=904 Ack=621 Win=16
126 03:31:55.883333 192.168.16.226  192.168.23.2    TCP       http > cplscrambler-in [RST, ACK] Seq=621 Ack=904 W
```

```
  ⊟ Checksum: 0x69c2 [correct]
      [Good Checksum: True]
      [Bad Checksum: False]
  ⊟ [SEQ/ACK analysis]
      [This is an ACK to the segment in frame: 37]
      [The RTT to ACK the segment was: 0.003791000 seconds]
⊟ Data (112 bytes)
    Data: 3C3F786D6C2076657273696F6E3D22312E30223F3E3C6372...
```

```
0030  44 59 69 c2 00 00 3c 3f  78 6d 6c 20 76 65 72 73   DYi...<? xml vers
0040  69 6f 6e 3d 22 31 2e 30  22 3f 3e 3c 63 72 6f 73   ion="1.0 "?><cros
0050  73 2d 64 6f 6d 61 69 6e  2d 70 6f 6c 69 63 79 3e   s-domain -policy>
0060  3c 61 6c 6c 6f 77 2d 61  63 63 65 73 73 2d 66 72   <allow-a ccess-fr
0070  6f 6d 20 64 6f 6d 61 69  6e 3d 22 2a 22 20 74 6f   om domai n="*" to
0080  2d 70 6f 72 74 73 3d 22  2a 22 20 2f 3e 3c 2f 63   -ports=" *" /></c
0090  72 6f 73 73 2d 64 6f 6d  61 69 6e 2d 70 6f 6c 69   ross-dom ain-poli
00a0  63 79 3e 00 0d 0a                                  cy>...
```

Data (data.data), 112 bytes | Packets: 220 Displayed: 29 Marked: 0 | Profile: Default

Figure 24. Client–server communication in Wireshark


flash comm 843.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Expression... Clear Apply
Wireless Settings... Decryption Keys...
Follow TCP Stream
Stream Content
Find Save As Print
Help Close Filter Out This Stream

Filter: (ip.addr eq 192.168.16.226 and ip.addr eq 192.168.23.2) and (tcp.port eq 8

802.11 Channel: Channel Offset: FCS Filter: Decryption Mode: None

```
No. - Time            Source         Destination    Protocol Info
   33 03:29:47.417007 192.168.23.2   192.168.16.226 TCP      cplscrambler-al > 843 [SYN] Seq=0 Win=16384
   35 03:29:47.418560 192.168.16.226 192.168.23.2   TCP      843 > cplscrambler-al [SYN, ACK] Seq=0 Ack=1
   36 03:29:47.418587 192.168.23.2   192.168.16.226 TCP      cplscrambler-al > 843 [ACK] Seq=1 Ack=1 Win=
   37 03:29:47.418941 192.168.23.2   192.168.16.226 TCP      cplscrambler-al > 843 [PSH, ACK] Seq=1 Ack=1
   39 03:29:47.422732 192.168.16.226 192.168.23.2   TCP      843 > cplscrambler-al [PSH, ACK] Seq=1 Ack=2
   40 03:29:47.422804 192.168.16.226 192.168.23.2   TCP      843 > cplscrambler-al [FIN, ACK] Seq=113 Ack
   41 03:29:47.422830 192.168.23.2   192.168.16.226 TCP      cplscrambler-al > 843 [ACK] Seq=24 Ack=114 W
   42 03:29:47.422870 192.168.23.2   192.168.16.226 TCP      cplscrambler-al > 843 [FIN, ACK] Seq=24 Ack=
   43 03:29:47.424251 192.168.16.226 192.168.23.2   TCP      843 > cplscrambler-al [ACK] Seq=114 Ack=25 W
```

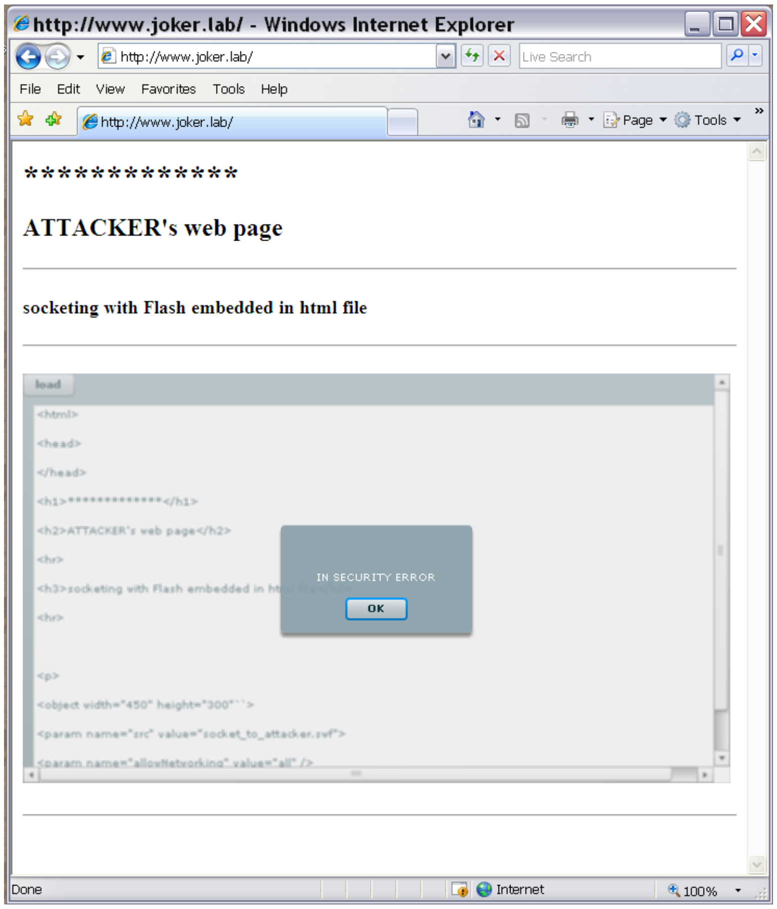
```
<policy-file-request/>.<?xml version="1.0"?><cross-domain-policy><allow-access-from
domain="*" to-ports="*" /></cross-domain-policy>.
```

Entire conversation (135 bytes) ( ) ASCII ( ) EBCDIC ( ) Hex Dump ( ) C Arrays [✓ Raw]

```
0000  00 12 3f ad bb 96 00 12  d9 67 5e d1 08 00 45 00   ..?..... .g^...E.
0010  00 98 7e 6f 40 00 7d 06  d5 bb c0 a8 10 e2 c0 a8   ..~o@.}. ........
0020  17 02 03 4b 04 40 55 62  c3 64 7d 21 d3 ca 50 18   ...K.@Ub .d}!..P.
0030  44 59 69 c2 00 00 3c 3f  78 6d 6c 20 76 65 72 73   DYi...<? xml vers
```

File: "C:\Documents and Settings\localadmin\Des... Packets: 220 Displayed: 9 Marked: 0 Profile: Default

Figure 25. The TCP stream for policy file request–response


http://www.joker.lab/ - Windows Internet Explorer
http://www.joker.lab/
Live Search
File Edit View Favorites Tools Help
Page
Tools
load
OK
Done
Internet
100%

****

ATTACKER's web page

socketing with Flash embedded in html file

```
<html>
<head>
</head>
<h1>*************</h1>
<h2>ATTACKER's web page</h2>
<hr>
<h3>socketing with Flash embedded in html file</h3>
<hr>

<p>
<object width="450" height="300"``>
<param name="src" value="socket_to_attacker.swf">
<param name="allowNetworking" value="all" />
```

IN SECURITY ERROR

Figure 26. Flash 10.0.22.87 throws security error

IV. DEFENSES AGAINST DNS REBINDING ATTACKS

A. CHAPTER OVERVIEW

This chapter provides the defenses that have been developed in order to mitigate the DNS rebinding attacks. Section B describes the defenses for plug-ins and their adoption from vendors, focusing on Adobe Flash Player and JVM. It follows the description of the firewall defenses in parallel with proposed changes in DNS server configuration. It also describes how the Web server can protect itself against DNS rebinding attacks. Section C reviews all the current effective defenses against DNS rebinding attacks discussing the concept of Defense-in-Depth in Network Security. Section D compares the analysis for defenses with DNS Security Technical Implementation Guidance (STIG) by Defense Information Systems Agency (DISA) for Department of Defense (DoD). The analysis is also compared with Client Configuration Guidance as contained in the Microsoft Vista Specialized Security-Limited Functionality Template and Split-DNS architecture.

B. DEFENSES AGAINST DNS REBINDING ATTACKS

1. Adobe Flash Player and Java Plug-ins Defenses

Adobe Flash Player and Java are the most widely deployed Web browser plug-ins. Figure 27 shows the latest statistics about Adobe Flash Player and Java penetration. According to Adobe [37]:

> Adobe ® Flash ® Player is the world's most pervasive software platform, used by over 2 million professionals and reaching 99.0% of Internet-enabled desktops in mature markets as well as a wide range of devices. Mature Markets include US, Canada, UK, France, Germany and Japan [37].

Consequently, one of the first priorities for defending against DNS rebinding attacks is to fix the DNS rebinding vulnerabilities in Web browser plug-ins. For that purpose, a malicious Web page has to be prevented from achieving socket level access to a random IP address. The Stanford security team proposed modifications to the socket access policies of Adobe Flash Player and Java. More specifically, they proposed changes in socket level access depending on the way plug-ins provide socket access, which may be denied or allowed by default. As they stated [3]:

> Plug-ins comprise a particular source of complexity in defending against DNS rebinding attacks because they enable subsecond attacks, provide socket-level network access, and operate independently from browsers. Plug-ins that grant socket-level access to Web content either default to allowing socket connection or they default to denying socket-level access. Different DNS rebinding defenses are appropriate for these different socket access paradigms [3].

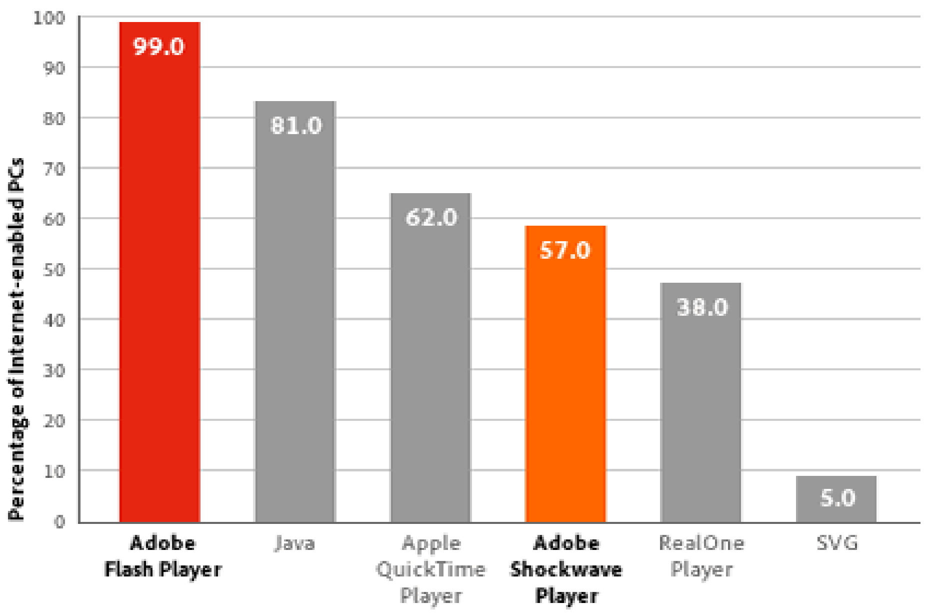


Figure 27. Adobe Flash Player and Java plug-ins penetration [From 37]

a. Java Socket Connection Proposed Changes

Java plug-in for Web browsers allows by default socket connections back to Web server's content. If the Java plug-in does not retrieve the content directly from an IP address then it can defend against DNS rebinding attacks by verifying that the destination Web server agrees to accept socket connections from the content’s host name. Successful deployment of this defense requires a mechanism that takes as input a pair of IP address and host name and determines whether the server at that IP address authorizes that host name [3].

DNS rebinding requires that the IP address is authoritative for authorizing host names. In contrast, forward DNS queries require that the host name is authoritative for responding to queries. So, according to Stanford security team suggestions, the reverse DNS system can be extended to authorize host names without sacrificing backwards compatibility. For instance, the owner of an IP address, such as 192.168.23.10, can authorize a host name, such as www.angel.lab, by including the following Pointer DNS Record Type (PTR record):

```
10.23.168192.in-addr.arpa. IN PTR www.angel.lab
```

This PTR record has the format as the existing reverse DNS. Its purpose is to include a security policy implementation into the existing use of reverse DNS. The only disadvantage is that this record format can encode only one host name per IP address. In many cases such as big organizations, the Web server has to map several host names to a one IP address [3].

Stanford security team also proposed that the Java plug-in consult reverse DNS records of the following form:

```
www.angel.com.auth.10.23.168.192.in-addr.arpa. IN A
192.168.23.10
```

This proposal uses the `auth` domain to authorize a set of host names for an IP address [3].

b. Adobe Flash Player Socket Connection Proposed Changes

Adobe Flash Player represents the case where the socket level access is denied by default. The destination Web server accepts socket connections if and only if it provides an XML policy file using, for example, a policy file server like the one that was used during the experiments described above and presented in Appendix E.

According to the Stanford security team, Adobe Flash Player plug-in can defend against DNS rebinding attacks applying the following changes:

- Policy files have to be considered as valid only for the IP address from which they were obtained.
- If an Adobe Flash application attempts to connect to another IP address, even one with the same host name, Adobe Flash Player should request another policy file.
- If an attacker attempts to rebind to attacker's domain (joker.com in the lab) to a target server (angel.lab in testbed), Flash Player will request another XML policy from the target server in order to determine whether or not to open the socket connection.

These modifications to the policy behavior are also compatible with previous security policy usage because all legitimate servers that share a host name are expected to serve the same policy file [3].

Especially for the case of port numbers greater than or equal to 1024, Adobe Flash Player 9.0.48.0 allows by default socket access to the origin Web server that hosts the Adobe Flash application. Although the majority of services an attacker can valuably target are hosted on low-numbered ports (as SMTP: 25, HTTP: 80, HTTPS: 443, SSH: 22, FTP: 20, 21), there are services such as MySQL, BitTorrent, IRC, and HTTP proxies that listen on high numbered ports. In this case, the Stanford security team proposed that Adobe Flash Player deny by default socket level access and use the XML policy file mechanism to prevent DNS rebinding attacks [3].

c. Plug-ins Socket Access Changes Adoption

Sun has put into practice the above changes in the JVM, fixing DNS rebinding vulnerabilities. The updated Java plug-in rejects network requests if the host name is not explicitly authorized in reverse DNS. According to Sun, the Java versions that are vulnerable to these attacks, as well as the updated and secured versions, are shown in Table 9 [38].

Adobe has also adopted the above proposals and has patched Flash Player to prevent socket level DNS rebinding vulnerabilities, updating its security policy using more strict rules as they were analyzed in previous the chapter and were shown in action through the experiments that were carried out and described above. As a result, the current Adobe Flash Player 10.0.22.87, as well as Flash Player 9.0.151.0 version and above, are secured with respect to the known DNS rebinding attack methods.

Vulnerable Java Releases
JDK and JRE 6 Update 2 and earlier
JDK and JRE 5.0 Update 12 and earlier
SDK and JRE 1.4.2_15 and earlier
SDK and JRE 1.3.1_20 and earlier

Table 9. Vulnerable to DNS rebinding Java releases [After 38]

In the same way, Microsoft has adopted the proposed defenses for Adobe Flash Player in order to secure the Silverlight plug-in Socket API [39].

To summarize, Sun, Adobe and Microsoft, the vendors of the most widely used Web browser plug-ins, have deployed the necessary defenses in patches preventing the currently known firewall circumvention and IP hijacking types of DNS rebinding attacks.

2. Firewall Defenses and Modified DNS Server

Networks like the one described and used for experiments during this research (angel.lab) can protect themselves against firewall circumvention using DNS rebinding by preventing external host names from resolving to internal IP addresses thereby preventing the attacker from naming an internal target server (angel.lab in the testbed) to its own host name (joker.lab in the testbed).

If the attacker is not able to perform the above action, the same origin policy cannot be violated and the victim client will not be transformed to a proxy under control of the attacker. These malicious DNS bindings can be blocked either by using filtering packets at the firewall or by modifying the DNS resolvers used by hosts on the network [3].

a. Modified DNS Server

The Stanford security team, which developed and proposed these defenses, developed a modified DNS server that prevents external names from resolving to internal IP addresses. They implemented this method in a program written using the C programming language called `dnswall` [40], which is defined as follows:

> *`dnswall`* is a daemon that filters out private IP addresses in DNS responses. It is designed to be used in conjunction with an existing recursive DNS resolver in order to protect networks against DNS rebinding attacks [3].

This modified DNS server works in parallel with an organization's firewall in order to protect its local area networks (LANs). More specifically, a network administrator can apply a firewall rule to block outbound traffic on port 53, which is the port designated to the DNS Internet service. Then, all internal machines, including HTTP proxies and virtual private network (VPN) clients, can be forced to use the modified DNS server that prevents external names from resolving to internal IP addresses.

According to the Stanford security team, `dnswall` has been tested and embedded in Berkeley Internet Name Domain (BIND) with success. BIND is the most widely used DNS server on the Internet. So, when `dnswall` operates in conjunction with BIND, it changes DNS responses that try to connect external names with internal IP addresses, successfully preventing the firewall circumvention type of DNS rebinding attacks [3].

b. Firewall and DNS Server Defenses Adoption

Several organizations have deployed the above-modified DNS server to protect their corporate networks. The FreeBSD operating system, which is a version of UNIX, runs on Intel microprocessors and powers the servers of the Web's largest sites, includes `dnswall` in order to defend against DNS rebinding attacks [41].

Consumer firewalls, like those produced by Linksys, defend their private networks from firewall circumvention by using `dnswall` to block DNS responses that contain private IP addresses. These firewalls can implement this defense without user configuration because these devices often manage the allocation of private IP addresses. Moreover, the vendors of these devices encourage the patching of their products because DNS rebinding attacks can be used to access the private configuration interface of these devices and potentially reconfigure them to mount additional attacks on their owners [3].

Finally, several open source consumer firewall projects have adopted the protection using `dnswall`, including Dnsmasq , Open-Wrt and DD-WRT [3].

3. Web Server Defense

A single Web server can defend itself against DNS rebinding attacks effectively if it checks and confirms the authenticity of the HTTP host header. Consequently, it has to reject requests that contain an unexpected host header value. Without socket level access, the attacker’s Web content is unable to spoof the host header. This defense is suitable for servers that trust the Web browser's IP address [3].

a. Host Header Checking

Hypertext Transfer Protocol (HTTP/1.1) requires that a user's Web browser includes a host header in HTTP requests that specifies the host name of the server. More specifically, according to RFC 2616 regarding HTTP:

> A "host" without any trailing port information implies the default port for the service requested (e.g., "80" for an HTTP URL). For example, a request on the origin server for:
>
> http://www.w3.org/pub/www/
>
> would properly include:
>
> GET /pub/www/ HTTP/1.1
> Host: www.w3.org
>
> A client MUST include a Host header field in all HTTP/1.1 request messages. If the requested URI does not include an Internet host name for the service being requested, then the Host header field MUST be given with an empty value. An HTTP/1.1 proxy MUST ensure that any request message it forwards does contain an appropriate Host header field that identifies the service being requested by the proxy. All Internet-based HTTP/1.1 servers MUST respond with a 400 (Bad Request) status code to any HTTP/1.1 request message, which lacks a Host header field [42].

This feature is used widely by HTTP proxies and by Web servers to host many virtual hosts on one IP address. Servers can use the host header to defend themselves against DNS rebinding attacks. During a DNS rebinding attack, the browser sends a host header with the attacker's host name to the target server. Using a Web browser's Application Programming Interfaces (APIs), such as `XMLHttpRequest`, Web content can specify HTTP headers but cannot change the host header. So, a server can protect itself from DNS rebinding attacks by rejecting HTTP requests that contain an unrecognized or unexpected host header [3].

For example, according to the Stanford security team [3], Apache servers can defend themselves using the following `ModSecurity` rule:

```
  SecRule REQUEST_HEADERS:Host!^www\.example\.com(:\d+)?$
deny,status:403
```

This rule confirms that the host header contains the expected value, rejecting requests with unexpected or missing host headers [3].

Finally, the tests during this research, as described above, showed the importance of the target server's host Header checking. The DNS rebinding attack scenario that was executed can be effective if and only if the target Web server does not perform host header checking, which is very likely because the host name value is optional by default in IIS 6.0 of Windows Server 2003, as well as in IIS 7.0 of Windows Server 2008 as was discovered during this research.

b. Host Header Checking Limitations

Host header checking also has some restrictions. For instance, checking the host header is problematic for servers that do not know their host name. Assuming that these servers are behind firewalls, the firewall defenses as described above are more suitable for this case. Another example is home routers, where most of them do not know the host name of their Web configuration interface [3].

It is also important to note that a browser vulnerable to DNS rebinding plug-in, as Adobe Flash Player 9.0.48.0, can be used to spoof the host header, so the Web server's host header checking becomes inefficient to mitigate the attack. Using as an example the network topology of this research, even if the target Web server (angel.lab in Figure 6) performs host header checking, it can be vulnerable to a DNS rebinding attack in the case where the victim client (victim client 2 in Figure 6) has a vulnerable plug-in installed in its Web browser.

That happens because the older versions of Adobe Flash Player and the JVM with DNS rebinding vulnerabilities, beyond for the fact they can be used to violate the Web browser's same origin policy (as described and tested in

Chapter III), they can also allow an attacker to spoof the host header preventing the host header checking from protecting against DNS rebinding attacks [43].

In brief, Web server host header checking must be combined with the usage of the latest Adobe Flash Player and Java plug-ins in all internal clients' machines in order to be effective. In the case that host header checking cannot be performed because the Web server does not know its host name, then the defense against DNS rebinding attacks relies only on firewall and modified DNS server methods, which prevent external host names from resolving to internal IP addresses.

C. DEFENSE-IN-DEPTH AGAINST DNS REBINDING ATTACKS

Defense-in-Depth in Network Security is a strategy for achieving Information Assurance in today's highly networked environments. It is based on the implementation of all the currently known methods and technologies that can be used in order to protect an organization's assets. The strategy recommends a balance between the protection capability and cost, performance, operational considerations and it was conceived by National Security Agency (NSA) [44].

Defense-in-Depth has two dimensions: using more than one protective mechanism, (for example using Intrusion Detection System (IDS), firewall, encryption; and using more than one type of a specific protective mechanism (for example using both a signature-based and behavior-based IDS). In other words, the purpose is to avoid relying only on one defensive mechanism. [45].

Each protection mechanism is expected to have its own flaws and restrictions. But together, the mechanisms provide multiple layer defenses in order to make the attacker's job much harder. For example, the host header checking, which is a Web server's defense method against DNS rebinding attacks, cannot prevent the attacks successfully as a standalone security measure, because in the case that the victim's browser has a plug-in installed, that is vulnerable to DNS rebinding then the server is vulnerable to the attack, as described in previous section.

According to a Defense-in-Depth strategy, the mitigation of DNS rebinding attacks requires the implementation of multiple layer defenses that contain all the effective known defense methods, which are summarized as follows:

- Clients Web browsers have to update Adobe Flash Player and Java plug-ins to the most recent versions, which prevent socket level DNS rebinding vulnerabilities
- Firewalls have to prevent external host names from resolving to internal IP addresses working in parallel with a modified DNS server, as described above
- Web servers have to perform host header checking in order to reject requests that contain an unexpected host header value.

D. DEPARTMENT OF DEFENSE (DOD) GUIDELINES ANALYSIS

1. DNS Security Technical Implementation (STIG) V4R1

a. Background and Scope of DNS STIG

The Security Technical Implementations Guides (STIGs) and the National Security Agency (NSA) Guides are the configuration standards for DoD Information Assurance (IA) for support of IA-enabled devices or systems. The DNS STIG was designed in October 2007 by Defense Information Systems Agency (DISA) to assist administrators with the configuration of DNS server software and related portions of the underlying operating system. It is provided under the authority of DOD Directive 8500.1. More specifically:

> The intent of this STIG is to include security considerations at the network level needed to provide an acceptable level of risk for information as it is transmitted throughout an enclave [46].

The DNS STIG is a requirement for all DoD administered systems and all systems connected to DoD networks. These requirements are intended to support Security Managers (SMs), Information Assurance Managers (IAMs), Information Assurance Officers (IAOs), and System Administrators (SAs) responsible for configuring and maintaining security controls. It details DoD DNS security practices and procedures applicable to all DoD name servers, including authoritative and recursive servers. The STIG specifically addresses issues and configuration choices for the following implementations of DNS [46]:

- BIND 9.3.1 and above
- BIND 9.3.2 for Microsoft Windows 2000, Windows XP and Windows 2003 Server
- Microsoft Windows 2000/2003 Server and DNS subsystem
- Cisco CSS DNS

It is important to note that the DNS STIG does not address the DNS configuration of DNS clients (for example, the workstations, servers, and network devices that question name servers). According to the STIG:

> Each of these clients runs DNS stub resolver software. Any requirements concerning those resolvers would be addressed in the STIG corresponding to the underlying technology such as the Desktop Services or Operating System STIGs (e.g., Desktop) [46].

The DNS Security Checklist contains the procedures that enable qualified personnel to verify the compliance with the DNS STIG. Client DNS configuration is outside the scope of this checklist, which focuses on DNS servers and related administrative, technical, and physical controls. The current version of this checklist (version 4, release 1.7) was recently updated (15 August 2009) [47].

b. Evaluation of DNS STIG and Suggestions

The DNS STIG and the corresponding DNS security checklist do not include any instruction related to the mitigation of the DNS rebinding attacks. There is no reference to these attacks, unlike other known attacks such as DNS poisoning. In the security checklist that corresponds to DNS STIG there are several checks that are described as mitigating factors against DNS poisoning or

Denial of Service attacks. Conversely, there is no check that is directly or even indirectly described as a security measure against DNS rebinding attacks.

The prevention of the resolution of external host names to internal IP addresses, which is the effective defense method that can be applied by a firewall in parallel with a modified DNS server, is not included in this STIG or the security checklist, although the STIG refers to firewall rules and their important role in security. So, the DNS STIG has to be updated in order to include the above mechanism that is currently known as an effective defense, at least against the firewall circumvention DNS rebinding attack.

Additionally, DNS rebinding attacks prevention requires combined defense methods, including Web browsers, DNS servers, Web servers, firewall rules and client machines. It means that Defense-in-Depth requires that DNS STIG should mention or refer to the rest of the defenses, which can be part of the same STIG or can be included in other more relevant STIGs. For example, there has to be an instruction about the Web server's host header checking, which may not be part of the DNS STIG but it is another fundamental defense method.

The nature of DNS rebinding attacks combines several different technologies from the Internet, so that a STIG about a single aspect of the Internet, e.g., the DNS service, cannot effectively eliminate this danger. Also, the DNS STIG should be tested with DNS rebinding attack scenarios like the scenario that was used during this research for safer results, which is part of the future work that results from this research.

2. Microsoft Windows Vista Client Security Guidance

a. Background and Scope of Microsoft Vista Client Specialized Security Limited Functionality (SSLF)

The Microsoft Windows Vista Client Security Guidance includes the Client Specialized Security Limited Functionality (SSLF), which is a guide that satisfies the creation of highly secure environments for computers running Windows Vista. The demand for security is so great in these environments that a significant loss of functionality and manageability is acceptable. The SSLF security settings are not intended for the majority of enterprise organizations. The configuration for these settings has been developed for organizations where *security is more important than functionality* [48].

The SSLF implements security restrictions that reduce user's functionality because it limits users to only the specific functions that they require to complete necessary tasks. Access is limited to approved applications, services, and infrastructure environments. The areas of higher security and limited functionality that the SSLF enforces are:

- Restricted services and data access
- Restricted network access
- Strong network protection [48].

b. Evaluation of Microsoft Windows Vista Client Specialized Security Limited Functionality (SSLF)

According to the SSLF the user has to disable Java and all the other plug-ins. More specifically, in "Appendix A: Security Group Policy," under the "Internet Control Panel \ Security page \ Restricted Sites Zone," the list with the recommended settings suggest the user disable: active scripting, file downloads, installation of desktop items, signed and unsigned ActiveX controls, Java permissions, launching applications and files in an `IFRAME`, running ActiveX controls and plug-ins, and scripting of Java applets [49].

This security measure is very effective against DNS rebinding attacks. The attacker has no way to subvert the victim client to a proxy in order to gain access to the internal server because the client cannot download and execute any malicious code from the attacker's page. For instance, the attack scenario that was tested during this research with round-robin DNS coupled with the `XmlHttpRequest` object. This attack cannot be successful if the victim client deploys the SSLF template. The reason is that even if the same origin policy could be violated, the JavaScript malicious code that tries to extract data from the internal Web server would never be executed due to victim client's restriction.

Additionally, this client security template combined with the host header checking that has to be performed by the internal Web server result in a system that is hardened enough to defend against the currently known DNS rebinding attacks.

However, the deployment of the SSLF template has serious drawbacks because it is not productive to configure machines that have no access to Web pages that use plug-ins. As stated in Chapter III and shown in Figure 27, almost 99% of users have Adobe Flash Player plug-in installed and more than 80% have Java plug-in installed in their machines in order to be able to enjoy the advanced capabilities that today's Internet provides.

Disabling all plug-ins from the user's Web browser is a kind of win for the attackers because they succeed in a Denial of Service attack only by the fear of the attack. Instead of disabling these features the approach of updating to secure versions and continuously watching for new security updates is a more feasible and reasonable solution against DNS rebinding attacks.

3. Split-DNS Architecture

a. Background and Scope of Split-DNS Architecture

In the Split-DNS architecture, the administrator creates two zones for the same domain. One is for the internal network while the other will be used by the external network. Split-DNS logically and physically separates the external and internal IP address spaces. Information that is necessary for external hosts on the Internet is maintained on the external DNS servers, while information about the internal hosts and IP space is maintained and resolved using the internal DNS servers [50].

b. Evaluation of Split-DNS

The Split-DNS architecture as a standalone security measure against DNS rebinding attacks provides only the protection of hiding the IP address space of the internal network. It makes harder the attacker's job of discovering the target's IP address in order to bind it with his own.

However, there is still the possibility for an unpatched internal machine to be transformed to a proxy when it visits attacker's Web page. For example, the internal Web server in the tested attack scenario is not compromised directly, but using a victim client as a proxy. The attacker succeeds in that by simply attracting the user to visit his Web page and he responds with legitimate DNS responses. The DNS rebinding attack is actually initialized by the unsuspicious victim client. Moreover, if the protected network does not use the proposed firewall and modified DNS server mechanism in order to prevent external host names from resolving to internal IP addresses the network is still vulnerable at least to firewall circumvention DNS rebinding attacks.

V. CONCLUSIONS–FUTURE WORK

A. CONCLUSIONS

DNS rebinding attacks compromise the same-origin policy that is designed to protect Web browsers from malicious Web contents. This thesis used a real network testbed to experiment with feasible attack scenarios, focusing on firewall circumvention and Adobe Flash Player socket vulnerabilities. The experimental results are summarized as follows:

- The same origin policy of all major Web browsers was violated in a DNS rebinding attack scenario with round robin DNS and JavaScript. Considering that the attack was successful using the default configuration for the creation of the target's Web page, it is apparent that the existence of Web servers that are vulnerable to this attack scenario is very likely because the usage of the default settings is very common case for most users or administrators.
- Host header checking proved to be an effective defense against attacks based on round robin DNS. The tests were carried out using IIS 6.0 of Windows Server 2003, where the host name value is not mandatory for a Web server so the users have to explicitly configure the Web server in order to perform host header checking. The research discovered that neither IIS 7.0 of Windows server 2008 has been updated to deploy this security measure by default.
- In the second scenario, a malicious Flash application successfully made a socket connection from the browser to the internal Web server without any notification when earlier versions of the Adobe Flash Player plug-in were used. More specifically, Flash Player 9.0.151.0 and above, as well as the current 10.0.22.87 version have deployed new security rules that eliminate the DNS rebinding vulnerabilities. This was verified through the tests.

- Moreover, the tests revealed additional security features in some Web browsers and the security model of Flash Player. Except for all Microsoft Internet Explorer versions, all the major Web browsers that were tested (Firefox, Safari, Google Chrome) forbad the manual installation of an outdated version of Flash Player, which should help reduce their exposure to DNS rebinding attacks. In contrast, all archived Internet Explorer versions accepted the installation of vulnerable Flash Player versions, which is a security weakness. The security model of Flash Player stores a value in the user's registry, which prevents downgrading to a previous version. This is an additional security feature provided from Adobe.

The results of the tests as well as an extensive literature survey also point to the following observations about the current defenses against DNS rebinding attacks:

- First, the vendors of the most widely used plug-ins have fixed the known DNS rebinding vulnerabilities by updating their security policies with more strict rules. So, the users should ensure that their Web browsers have installed the current versions of Adobe Flash Player, JVM and Microsoft Silverlight plug-ins.
- Second, the Web servers should be configured to perform host header checking in order to reject HTTP requests that contain unexpected host name values.
- Third, firewall rules in conjunction with a modified DNS server should be applied in order to prevent external host names from resolving to internal IP addresses of a protected network. This defense method specifically prevents firewall circumvention attacks.

Finally, this thesis evaluated the effectiveness of the following U.S. DoD guidelines against the DNS rebinding attacks, with the following results:

- The DNS STIG published by DISA and the corresponding security checklist, which in fact was recently updated (August 2009), do not include any instruction or security check related to the mitigation of the DNS rebinding attacks. DNS STIG should be updated in order to include a security measure that will prevent external host names from resolving to internal IP addresses.
- The Microsoft Windows Vista Client Security Guidance, which includes the Client Specialized Security Limited Functionality (SSLF), has been developed for organizations where security is more important than functionality. It means that SSLF reduces significantly the user's functionality with respect to specific tasks. Although this security measure is not feasible in the vast majority of users, it is effective against DNS rebinding attacks because the user's Web browser cannot install any plug-in so the attacker has no way to subvert it and turn it into a proxy.
- The Split-DNS architecture as a standalone security measure against DNS rebinding attacks provides only the protection of the IP address space of the internal network. It makes harder for the attacker to discover the target's IP address in order to rebind it to his own domain. However, there is still the possibility of an unpatched internal machine that can be transformed to a proxy when the user visits the attacker's malicious Web page.

In summary, the mitigation of DNS rebinding attacks requires the implementation of a defense-in-depth strategy because they involve different Internet services than just DNS servers. Specifically, it also targets Web servers and various plug-in technologies. A multiple layer defense including all the currently effective defenses against these attacks must be applied in order to protect an organization's digital assets.

The Network Security environment is extremely dynamic. As research on these attacks continues, additional defenses are likely to develop. Unfortunately, it is also likely that new vulnerabilities will be revealed. Thus, vigilance must be maintained.

B. FUTURE WORK

The experiments conducted herein can be extended using different software, both for servers and clients. For instance, BIND can be used as the DNS server, Apache as the Web server and the Linux operating system for the clients with the purpose of evaluating their existing defenses.

Alternate DNS rebinding attack scenarios belonging to the IP Hijacking category can be used for tests, as well as the evaluation of different plug-ins like Apple QuickTime Player.

The Split-DNS architecture can be deployed in the network topology of the current testbed in order to discover any attack scenarios that could be mitigated. In the same way, Microsoft Vista SSLF Client Security Guide can be applied to the testbed for further evaluation.

Finally, future work may continue the evaluation of DNS STIG in depth using the testbed for experiments. The target DNS server can be configured according to the STIG and then be tested using DNS rebinding attack methods, including the method that was used during this research.

APPENDIX A. SCRIPT FOR CONNECTIVITY CHECK

Appendix A presents the script that was written to check the testbed's connectivity.

MSPINGER.BAT

```
cls
@echo off
echo "*********************************************************"
echo "****** This a PINGER. Let's ping our entire system ******"
pause
echo ""
echo "****** Ping edge MS Router of angel.lab LAN *************"
ping -n 1 -w 1 10.19.8.9
ping -n 1 -w 1 192.168.23.9
echo ""
echo "****** Ping MS Server in angel.lab LAN ******************"
ping -n 1 -w 1 192.168.23.10
echo ""
echo "****** Ping central Router1 *****************************"
ping -n 1 -w 1 10.19.8.10
ping -n 1 -w 1 10.19.8.2
ping -n 1 -w 1 10.19.8.18
echo ""
echo "****** Ping edge Router3 of Client 2 LAN ****************"
ping -n 1 -w 1 10.19.8.17
ping -n 1 -w 1 192.168.23.1
echo ""
echo "****** Ping MS XP Client 2 in its LAN *******************"
ping -n 1 -w 1 192.168.23.2
echo ""
echo "****** Ping edge Router2 of joker.lab LAN ***************"
ping -n 1 -w 1 10.19.8.1
ping -n 1 -w 1 192.168.16.225
echo ""
echo "****** Ping MS Server in joker.lab LAN ******************"
ping -n 1 -w 1 192.168.16.226
echo ""
echo "*********************************************************"
pause
```

OUTPUT OF *MSPINGER.BAT*

```
"*********************************************************"
"****** This a PINGER. Let's ping our entire system ******"
Press any key to continue . . .
""
"****** Ping edge MS Router of angel.lab LAN *************"

Pinging 10.19.8.9 with 32 bytes of data:

Reply from 10.19.8.9: bytes=32 time<1ms TTL=128

Ping statistics for 10.19.8.9:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

Pinging 192.168.23.9 with 32 bytes of data:

Reply from 192.168.23.9: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.23.9:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
""
"****** Ping MS Server in angel.lab LAN ******************"

Pinging 192.168.23.10 with 32 bytes of data:

Reply from 192.168.23.10: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.23.10:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
""
"****** Ping central Router1 *****************************"

Pinging 10.19.8.10 with 32 bytes of data:

Reply from 10.19.8.10: bytes=32 time=1ms TTL=254

Ping statistics for 10.19.8.10:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms

Pinging 10.19.8.2 with 32 bytes of data:
```

```
Reply from 10.19.8.2: bytes=32 time=1ms TTL=254

Ping statistics for 10.19.8.2:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms

Pinging 10.19.8.18 with 32 bytes of data:

Reply from 10.19.8.18: bytes=32 time=1ms TTL=254

Ping statistics for 10.19.8.18:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms
""
"****** Ping edge Router3 of Client 2 LAN ****************"

Pinging 10.19.8.17 with 32 bytes of data:

Reply from 10.19.8.17: bytes=32 time=2ms TTL=253

Ping statistics for 10.19.8.17:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 2ms, Average = 2ms

Pinging 192.168.23.1 with 32 bytes of data:

Reply from 192.168.23.1: bytes=32 time=3ms TTL=253

Ping statistics for 192.168.23.1:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 3ms, Maximum = 3ms, Average = 3ms
""
"****** Ping MS XP Client 2 in its LAN *******************"

Pinging 192.168.23.2 with 32 bytes of data:

Reply from 192.168.23.2: bytes=32 time=2ms TTL=125

Ping statistics for 192.168.23.2:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 2ms, Average = 2ms
""
"****** Ping edge Router2 of joker.lab LAN ***************"
```

```
Pinging 10.19.8.1 with 32 bytes of data:

Reply from 10.19.8.1: bytes=32 time=2ms TTL=253

Ping statistics for 10.19.8.1:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 2ms, Average = 2ms

Pinging 192.168.16.225 with 32 bytes of data:

Reply from 192.168.16.225: bytes=32 time=2ms TTL=253

Ping statistics for 192.168.16.225:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 2ms, Average = 2ms
""
"****** Ping MS Server in joker.lab LAN ******************"

Pinging 192.168.16.226 with 32 bytes of data:

Reply from 192.168.16.226: bytes=32 time=1ms TTL=125

Ping statistics for 192.168.16.226:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms
""
"******************************************************"
Press any key to continue . . .
```

APPENDIX B. ROUTERS' CONFIGURATION FILES

Appendix B presents the configuration files for the testbed's network routers.

ROUTER 1: CONNECTING LANS

```
Current configuration : 986 bytes
!
version 12.1
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname router1
!
enable secret 5 $1$xDh4$nAo0mJGi44vzTtJH2YKMq1
!
!
!
!
!
memory-size iomem 15
ip subnet-zero
!
!
!
!
!
!
interface Ethernet0/0
 description link to client_LAN
 ip address 10.19.8.18 255.255.255.248
!
interface Ethernet1/0
 description link to MSRouter, defender side
 ip address 10.19.8.10 255.255.255.248
!
interface Ethernet1/1
 description link to Router 2, attacker side
 ip address 10.19.8.2 255.255.255.248
!
interface Ethernet1/2
 no ip address
 shutdown
!
interface Ethernet1/3
```

```
 no ip address
 shutdown
!
router rip
 version 2
 network 10.0.0.0
!
ip classless
no ip http server
!
banner login ^C
Hi George, how are you today? Don't worry, I am routing fine!
^C
banner motd ^C
This ROUTER1, the middle router, that connects all the LANS
^C
!
line con 0
 transport input none
line aux 0
line vty 0 4
 password geored36
 login
!
no scheduler allocate
End
```

ROUTER 2: DIRECTLY CONNECTED TO ATTACKER’S LAN

```
Current configuration : 1022 bytes
!
version 12.3
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname router2
!
boot-start-marker
boot-end-marker
!
enable secret 5 $1$iEbK$h49pSZqW2Xm6nawo14TkQ.
!
memory-size iomem 10
no aaa new-model
ip subnet-zero
ip cef
```

```
!
!
!
!
!
!
!
interface Ethernet0/0
 no ip address
 shutdown
 half-duplex
!
interface Ethernet1/0
 description link to ROUTER 1
 ip address 10.19.8.1 255.255.255.248
 half-duplex
!
interface Ethernet1/1
 description link to switch-joker
 ip address 192.168.16.225 255.255.255.248
 half-duplex
!
interface Ethernet1/2
 no ip address
 shutdown
 half-duplex
!
interface Ethernet1/3
 no ip address
 shutdown
 half-duplex
!
router rip
 version 2
 network 10.0.0.0
 network 192.168.16.0
!
no ip http server
ip classless
!
!
banner login ^C
Hi George, how is it going? Take a look at my routing table.
^C
banner motd ^Codt #
This is ROUTER2, the DFGW for attacker's LAN
^C
!
line con 0
line aux 0
line vty 0 4
 password geored36
 login
!
!
End
```

ROUTER 3: DIRECTLY CONNECTED TO VICTIM CLIENT 2 LAN

```
Current configuration : 1218 bytes
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname Router3
!
boot-start-marker
boot-end-marker
!
enable secret 5 $1$l3fU$zu6z7kFWjdyGYeAUqMkWM.
!
no aaa new-model
no network-clock-participate slot 1
no network-clock-participate wic 0
ip cef
!
!
ip auth-proxy max-nodata-conns 3
ip admission max-nodata-conns 3
!
!
!
!
!
!
!
!
!
!
!
!
interface FastEthernet0/0
 no ip address
 shutdown
 duplex auto
 speed auto
!
interface FastEthernet0/1
 no ip address
 shutdown
 duplex auto
 speed auto
!
interface Ethernet1/0
 description Link to ROUTER1
 ip address 10.19.8.17 255.255.255.248
 half-duplex
!
interface Ethernet1/1
 description Link to SWITCH client_victim
 ip address 192.168.23.1 255.255.255.248
 half-duplex
!
interface Ethernet1/2
 no ip address
 shutdown
 half-duplex
!
interface Ethernet1/3
 no ip address
 shutdown
 half-duplex
!
router rip
 version 2
 network 10.0.0.0
 network 192.168.23.0
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
!
!
control-plane
!
!
!
banner motd ^C
This is Router3 the DFGW for clint_victim's LAN
^C
!
line con 0
line aux 0
line vty 0 4
 password geored36
 login
!
!
End
```

APPENDIX C. HTML CODE FOR XMLHTTPREQUEST

Appendix C presents the HTML modified code that was used for the attacker's www.joker.lab Web page in order to show the functionality of the XmlHttpRequest object. The original code was extracted from w3schools.com online tutorials [25].

HTML CODE WITH XMLHTTPREQUEST OBJECT

```
<html>
<head>
<script type="text/JavaScript">
var xmlhttp;

function loadXMLDoc(url)
{
xmlhttp=null;
alert("here I am!!");
if (window.XMLHttpRequest)
  {// code for IE7, Firefox, Opera, etc.
  xmlhttp=new XMLHttpRequest();
  }
else if (window.ActiveXObject)
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
if (xmlhttp!=null)
  {
  alert("xmlhttp is not null, ok!");
  xmlhttp.onreadystatechange=state_Change;
  xmlhttp.open("GET,"url,false);
  //xmlhttp.open("GET,"url,true);
  alert("I passed open(GET,url,false)");
  xmlhttp.send(null);
  alert("I passed send(null)");
  }
else
  {
  alert("Your browser does not support XMLHTTP.");
  }
}

function state_Change()
{
if (xmlhttp.readyState==4)
  {// 4 = "loaded"
  if (xmlhttp.status==200)
    {// 200 = "OK"
    alert("now I am in status==200, should take smth!!");
    document.getElementById('A1').innerHTML=xmlhttp.status;
    alert("I passed A1");
    document.getElementById('A2').innerHTML=xmlhttp.statusText;
    alert("passed A2");
    alert("passed responsetext!" + xmlhttp.responseText);
    document.getElementById('A3').innerHTML=xmlhttp.responseText;
    }
  else
    {
    alert("Problem, xmhlhttp.status:" + xmlhttp.status);
    //alert("Problem retrieving XML data:" + xmlhttp.statusText);
    alert("Problem retrieving XML data:" + xmlhttp.responseText);
    }
  }
}
</script>
</head>
<body bgcolor="CC0066">
<h1>************</h1>
<h2>ATTACKER's web page</h2>
<hr>
<h3>DNS rebinding attack using XmlHttpRequest object</h3>
<hr>
<p><b>Status:</b>
<span id="A1"></span>
</p>
<p><b>Status text:</b>
<span id="A2"></span>
</p>
<p><b>Response:</b>
<br /><span id="A3"></span>
</p>
<hr>
<button onclick="loadXMLDoc('http://www.joker.lab')">Get content</button>
```

```
<hr>

<p>
<img src="img_1.gif"
width="230" height="200">
</p>

</body>
</html>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. FLASH APPLICATION EMBEDDED IN HTML CODE

Appendix D presents the HTML source code that was used for the www.joker.lab Web page when embedding a Flash application. It follows the code that was developed for the Flash application using the Flex 3 plug-in for Eclipse IDE.

HTML CODE EMBEDDING FLASH APPLICATION
<html> <head> </head> <h1>*************</h1> <h2>ATTACKER's web page</h2> <hr> <h3>socketing with Flash embedded in html file</h3> <hr> <p> <object width="450" height="300"``> <param name="src" value="socket_to_attacker.swf"> <param name="allowNetworking" value="all" /> <embed src="socket_to_attacker.swf" width="680" height="395"> <param name="allowNetworking" value="all" /> </embed> </object> </p> <hr> </body> </html>

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
<mx:Script>
      <![CDATA[
            import mx.controls.Alert

            private function loadsec():void{
            Alert.show("enter loadsecurity function");
```

```
Security.loadPolicyFile
                  ("http://www.joker.lab/crossdomain.xml");
flash.system.Security.loadPolicyFile
                  ("http://www.joker.lab/crossdomain.xml");
Security.allowDomain("*");
}

private var socket:Socket = new Socket();

private function load():void{
   loadsec();
   Alert.show("enter main function");
   socket.connect('192.168.16.226', 80);
   Alert.show("just after socket connection");
   socket.addEventListener
     (SecurityErrorEvent.SECURITY_ERROR, secalert_Handler);
   socket.addEventListener(IOErrorEvent.IO_ERROR,
                                         ioerror_Handler);
   socket.addEventListener(ProgressEvent.SOCKET_DATA,
                                        progress_Handler);
   socket.addEventListener(Event.CONNECT,
                                        connectedHandler);
   socket.addEventListener(ProgressEvent.SOCKET_DATA,
                                        inputDataHandler);
}

private function secalert_Handler
                          (event:SecurityErrorEvent):void {
   Alert.show("IN SECURITY ERROR");
   trace("securityErrorHandler: " + event);
}

private function ioerror_Handler(event:IOErrorEvent):void {
   Alert.show("IN IOerror event ERROR");
   trace("ioErrorHandler: " + event);
}

private function progress_Handler(event:ProgressEvent):void
{
   Alert.show("IN progress event");
}

private function inputDataHandler(event:ProgressEvent):void
{
   Alert.show("now in function that writes data in
                                                 string");
   var bytesLoaded:int = socket.bytesAvailable;
   var str:String =
                            socket.readUTFBytes(socket.bytesAvailable);
               myT.text+=str;
            }

            private function connectedHandler(event:Event):void {
               Alert.show("just entered GET \/index  function....");
               socket.writeUTFBytes("GET \/index.html\n\n");
               socket.flush();
            }
      ]]>
</mx:Script>
   <mx:Button click="load()" label="load"/>
   <mx:TextArea x="10" y="30" width="680" height="395" id="myT"/>
</mx:Application>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. JAVA POLICY SERVER

Appendix E presents the modified source code for the Flash Policy Server that was used during tests with the Flash Player application. The original code was developed by Thomas Meyer [35].

```
package policy;

import java.io.BufferedReader;
import java.io.EOFException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.InterruptedIOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * Class PolicyServer
 * Starts a PolicyServer on the specified port.
 * Can be started as main class, passing the port number as the first
command line argument
 * @author Thomas Meyer, Less Rain (thomas@lessrain.com)
 *
 */
public class PolicyServer extends Thread
{
   /**
   * If no argument is passed the server will listen on this port
     for connections
   */
   public static final int DEFAULT_PORT = 843;//1008;
   public static final String[] DEFAULT_POLICY = new String[] { "*" };

   /**
    * The character sequence sent by the Flash Player to request a
      _policy file
   */
   public static final String POLICY_REQUEST =
                                 "<policy-file-request/>";
   public static final boolean DEBUG = true;

   /**
    * @param args Use the first command line argument to set the port
      the server will listen on for connections
   */
   public static void main(String[] args)
   {
      int port = DEFAULT_PORT;
   try
      {
         if (args.length>0) port = Integer.parseInt(args[0]);
      }
         catch (NumberFormatException e) {}

      // Start the PolicyServer
      (new PolicyServer( port , new String[] { "*" })).start();
                                                 //{ "*:80" }))
}

/*
 * PolicyServer class variables
*/
private int _port;
private boolean _listening;
private ServerSocket _socketServer;
private String _policy;

/**
 * PolicyServer constructor
 * @param port_      Sets the port that the PolicyServer listens on
*/
public PolicyServer( int port_, String[] allowedHosts_ )
{
   _port = port_;
   _listening=true;
   if (allowedHosts_==null) allowedHosts_ = DEFAULT_POLICY;
   _policy = buildPolicy(allowedHosts_);
}

private String buildPolicy( String[] allowedHosts_ )
{
   StringBuffer policyBuffer = new StringBuffer();

   policyBuffer.append("<?xml version=\"1.0\"?><cross-domain-
                                                   policy>");
   for (int i = 0; i < allowedHosts_.length; i++) {
      String[] hostInfo = allowedHosts_[i].split(":");
      String hostname = hostInfo[0];
      String ports;
      if (hostInfo.length>1) ports = hostInfo[1];
      else ports = "*";

      policyBuffer.append("<allow-access-from
                domain=\""+hostname+"\" to-ports=\""+ports+"\" />");
   }
      policyBuffer.append("</cross-domain-policy>");

      return policyBuffer.toString();
}
/**
 * Thread run method, accepts incoming connections and creates
   SocketConnection objects to handle requests
*/
```

```
public void run()
{
   try
   {
      _listening=true;
      // Start listening for connections
      _socketServer = new ServerSocket(_port,50);
      if (DEBUG) System.out.println("PolicyServer listening on port
                                                               "+_port);
      while(_listening)
      {
         // Wait for a connection and accept it
         Socket socket = _socketServer.accept();

         try
         {
            if (DEBUG) System.out.println("PolicyServer got a
                                          connection on port "+_port);
            // Start a new connection thread
            (new SocketConnection(socket)).start();
         }
         catch (Exception e)
         {
            if (DEBUG) System.out.println("Exception:"
                                                   +e.getMessage());
         }
         try
         {
         // Wait for a sec until a new connection is accepted to
            avoid flooding
         sleep(1000);
         }
         catch (InterruptedException e) {}
         }
   }
   catch(IOException e)
   {
   if (DEBUG) System.out.println("IO Exception: "+e.getMessage());
   }
}

/**
 * Local class SocketConnection
 * For every accepted connection one SocketConnection is created.
 * It waits for the _policy file request, returns the _policy file
   and closes the connection immediately
 * @author Thomas Meyer, Less Rain (thomas@lessrain.com)
 *
 */
 class SocketConnection extends Thread
 {
    private Socket _socket;
    private BufferedReader _socketIn;
    private PrintWriter _socketOut;
```

```
/**
 * Constructor takes the Socket object for this connection
 * @param socket_    Socket connection to a client created by
   the PolicyServer main thread
 */
 public SocketConnection(Socket socket_)
 {
     _socket = socket_;
 }

 /**
  * Thread run method waits for the _policy request, returns the
    poilcy file and closes the connection
  */
  public void run()
  {
     try
     {
        // initialize socket and readers/writers
        _socket.setSoTimeout(10000);
        _socketIn = new BufferedReader(new InputStreamReader
                                      (_socket.getInputStream()));
        _socketOut =new PrintWriter(_socket.getOutputStream(),
                                                          true);
     }
     catch (IOException e)
     {
        if (DEBUG) System.out.println("IO Exception "
                                               +e.getMessage());
           return;
     }

     readPolicyRequest();
  }

  /**
   * Wait for and read the _policy request sent by the Flash Player
   * Return the _policy file and close the Socket connection
   */
   private void readPolicyRequest()
   {
      try
      {
         // Read the request and compare it to the request string
                                     defined in the constants.
         // If the proper _policy request has been sent write out
                                              the _policy file
         if (POLICY_REQUEST.equals(read()))
            write(_policy);
      }
      catch (Exception e)
      {
         if (DEBUG) System.out.println("Exception "+e.getMessage());
      }
      close();
```

```
}

/**
 * Read until a zero character is sent or a maximum of 100 character
 * @return The character sequence read
 * @throws IOException
 * @throws EOFException
 * @throws InterruptedIOException
 */
 private String read() throws IOException, EOFException,
                                                InterruptedIOException
 {
    StringBuffer buffer = new StringBuffer();
    int codePoint;
    boolean zeroByteRead=false;

    if (DEBUG) System.out.println("Reading...");
      do
      {
         codePoint=_socketIn.read();
         if (codePoint==0) zeroByteRead=true;
         else buffer.appendCodePoint( codePoint );
      }
      while (!zeroByteRead && buffer.length()<100);
         if (DEBUG) System.out.println("Read: "+buffer.toString());

            return buffer.toString();
 }

 /**
  * Writes a String to the client
  * @param msg Text to be sent to the client (_policy file)
  */
  public void write(String msg)
  {
     _socketOut.println(msg+"\u0000");
     _socketOut.flush();
     if (DEBUG) System.out.println("Wrote: "+msg);
  }

  /**
   * Close the Socket connection an set everything to null. Prepared
     for garbage collection
   */
   public void close()
   {
      try
      {
         if (_socket!=null) _socket.close();
         if (_socketOut!=null) _socketOut.close();
         if (_socketIn!=null) _socketIn.close();
      }
      catch (IOException e) {}

         _socketIn=null;
```

```
            _socketOut=null;
            _socket=null;
        }

    }

}
```

LIST OF REFERENCES

[1] J. Ruderman, “Same origin policy for JavaScript,” 2008, https://developer.mozilla.org/en/Same_origin_policy_for_JavaScript. Retrieved May 2009.

[2] F. Hsu, “OMash: Enabling Secure Web Mashups via Object Abstractions,” 2008, http://www.cs.ucdavis.edu/~hchen/paper/ccs08-slide.pdf. Retrieved May 2009.

[3] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh “Protecting Browsers from DNS Rebinding Attacks,” 2007, http://crypto.stanford.edu/dns/dns-rebinding.pdf. Retrieved May 2009.

[4] D. Dean, E. W. Felten, and D. S. Wallach “Java Security: From HotJava to Netscape and Beyond,” 1996, http://www.cs.princeton.edu/sip/pub /secure96.html. Retrieved May 2009.

[5] Princeton University, CS Department, “DNS Attack Scenario,” 1996, http://www.cs.princeton.edu/sip/news/dns-scenario.html. Retrieved May 2009.

[6] World Wide Web Consortium, “The XMLHttpRequest object,” 2008, http://www.w3.org/TR/XMLHttpRequest/#xmlhttprequest. Retrieved May 2009.

[7] Wikipedia, “XMLHttpRequest,” 2005, http://en.wikipedia.org/wiki /XMLHttpRequest. Retrieved May 2009.

[8] D. Byrne, Black Hat Briefings, 2007, “Intranet Invasion through Anti-DNS Pinning,” https://www.blackhat.com/presentations/bh-usa-07/Byrne /Presentation/bh-usa-07-byrne.pdf. Retrieved May 2009.

[9] C. Matthies, “DNS Pinning explained,” 2007, http://christ1an.blogspot.com /2007/07/dns-pinning-explained.html. Retrieved May 2009.

[10] M Johns, “(somewhat) breaking the same-origin policy by undermining dns-pinning,” 2006, http://seclists.org/bugtraq/2006/Aug/0290.html. Retrieved May 2009.

[11] “Feature: Understanding TCP Reset Attacks, Part I,” 2004, http://kerneltrap.org/node /3072. Retrieved May 2009.

[12] Kanatoko Anvil, “Stealing Information Using Anti-DNS Pinning (DNS Rebinding): Online Demonstration,” 2006, http://www.jumperz.net /index.php?i=2&a=1&b=7. Retrieved May 2009.

[13] Wikipedia, “Adobe Flash Player,” 2009, http://en.wikipedia.org/wiki /Flash_player. Retrieved May 2009.

[14] Adobe, “Security Bulletin,” 2007, http://www.adobe.com/support /security/bulletins/apsb07-20.html. Retrieved May 2009.

[15] Adobe, “Policy file changes in Flash Player 9 and Flash Player 10,” 2008, http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security_03. html. Retrieved May 2009.

[16] Common Vulnerabilities and Exposures, “CVE-2007-5275,” 2007, http://cve.mitre.org/cgi-bin/cvename.cgi?name=2007-5275, Retrieved May 2009.

[17] Click Fraud Report, “What is Click Fraud,” 2005, http://www.clickfraudreport.com/1.html. Retrieved May 2009.

[18] Kanatoko Anvil, “Anti-DNS Pinning (DNS Rebinding) + Socket in Flash,” 2006, http://www.jumperz.net/index.php?i=2&a=3&b=3. Retrieved May 2009.

[19] Adobe, “Flash Player Penetration,” 2009, http://www.adobe.com/products /player_census/flashplayer/. Retrieved May 2009.

[20] TredoSoft, 2006, http://tredosoft.com/Multiple_IE. Retrieved June 2009.

[21] Download Adobe Flex Builder 3, http://www.adobe.com/cfusion /entitlement /index.cfm?e=flexbuilder3. Retrieved June 2009.

[22] Archived Flash Players available for testing purposes, http://kb2.adobe.com/cps/142/tn_14266.html. Retrieved June 2009.

[23] Microsoft Help and Support, “How to Disable Client-Side DNS Caching in Windows XP and Windows Server 2003,” 2007, http://support.microsoft.com/kb/318803. Retrieved June 2009.

[24] WindowsITPro, “How can I enable or disable subnet prioritization on a client machine,” 2002, http://windowsitpro.com/article/articleid/27026/how-can-i-enable-or-disable-subnet-prioritization-on-a-client-machine.html. Retrieved June 2009.

[25] W3schools.com, “The XmlHttpRequest object,” http://www.w3schools.com /xml/xml_http.asp. Retrieved June 2009.

[26] SecurityFocus, “Adobe Flash Player DNS Rebinding Vulnerability,” 2007 http://www.securityfocus.com/bid/26930/info. Retrieved July 2009.

[27] National Vulnerability Database, “Vulnerability Summary for CVE-2007-5275,” http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-5275, Retrieved July 2009.

[28] 14th ACM Conference on Computer and Communication Security (CCS), 2007, http://www.sigsac.org/ccs/CCS2007/paper-list.html. Retrieved July 2009.

[29] Adobe, “Flash Player update available to address security vulnerabilities,” 2008, http://www.adobe.com/support/security/bulletins/apsb08-20.html. Retrieved July 2009.

[30] Adobe, “Policy file changes in Flash Player 9 and Flash Player 10,” 2008, http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security.html. Retrieved July 2009.

[31] The H, “Adobe releases Flash Player 9.0.151.0 after all,” 2008, http://www.honline.com/news/Adobe-releases-Flash-Player-9-0-151-0-after-all--/111902. Retrieved July 2009.

[32] Adobe, “Version test for Adobe Flash,” 2009, http://kb2.adobe.com/cps/155/tn_15507.html. Retrieved July 2009.

[33] Adobe, “Archived Flash Players available for testing purposes,” 2009, http://kb2.adobe.com/cps/142/tn_14266.html#ionComHeading. Retrieved July 2009.

[34] Adobe Technote, “Safe versions security restrictions when installing Flash Player,” 2009, http://kb2.adobe.com/cps/402/kb402435.html. Retrieved July 2009.

[35] Lessrain blog, “AS3 + Java: Socket connections to ports below 1024,” 2006, http://www.blog.lessrain.com/as3-java-socket-connections-to-ports-below-1024/. Retrieved July 2009.

[36] Collin Jackson, 2009, http://www.collinjackson.com/. Retrieved July 2009.

[37] Adobe, “Flash Player Penetration,” 2009, http://www.adobe.com/products /player_census /flashplayer/. Retrieved July 2009.

[38] Sun, “Sun Alert,” 2008, http://sunsolve.sun.com/search /document.do?assetkey=1-26-103078-1. Retrieved August 2009.

[39] MSDN, “Network Security Access Restrictions in Silverlight,” 2008, http://msdn.microsoft.com/en-us/library/cc645032(VS.95).aspx. Retrieved August 2009.

[40] Google code, “google-dnswall,” 2008, http://code.google.com/p/google-dnswall/. Retrieved August 2009.

[41] FreeBSD software, “The FreeBSD Ports Archive,” 2009, http://www.freebsdsoftware.org/dns/. Retrieved August 2009.

[42] RFC 2616, “Hypertext Transfer Protocol-HTTP/1.1,” 1999, http://www.w3.org/Protocols/rfc2616/rfc2616.html. Retrieved August 2009.

[43] SecurityFocus, “Host header cannot be trusted as an anti anti DNS-pinning measure,” 2006, http://www.securityfocus.com/archive/1 /445490/30/0/threaded. Retrieved August 2009.

[44] NSA, “Defense in Depth,” http://www.nsa.gov/ia/_files/support /defenseindepth.pdf. Retrieved August 2009.

[45] J.D.Fulp, “Network Security Core Principles,” 2009, CS3690 Network Security Course Notes. Retrieved August 2009.

[46] Security Technical Implementation Guides (STIG) and supporting documents, “Domain Name System V4R1,” 2007, http://iase.disa.mil/stigs /stig/dns_stig_v4r1_20071017.pdf. Retrieved August 2009.

[47] Security Technical Implementation Guides (STIG) and supporting documents, “Domain Name System Security Checklist V4R1.7,” 2009, http://iase.disa.mil/stigs/checklist/dns-checklist-vr4r1-7_20090815.pdf. Retrieved August 2009.

[48] MicrosoftTechNet, “Chapter 5: Specialized Security – Limited Functionality,” 2009, http://technet.microsoft.com/en-us/library /bb629464.aspx. Retrieved August 2009.

[49] MicrosoftTechNet, “Appendix A: Security Group Policy Settings,” 2009, http://technet.microsoft.com/en-us/library/bb679962.aspx#_internet_ control_pane-security. Retrieved August 2009.

[50] Docstoc, “What is a Split DNS,” 2003, http://www.docstoc.com/docs /2260813/What-is-a-Split-DNS. Retrieved August 2009.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Defense Information Systems Agency
   Arlington, VA

3. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

4. Geoffrey Xie
   Naval Postgraduate School
   Monterey, CA

5. John H. Gibson
   Naval Postgraduate School
   Monterey, CA

6. Georgios Kokkinopoulos
   Naval Postgraduate School
   Monterey, CA

7. Trent Pitsenbarger
   National Security Agency
   Fort Meade, MD

8. Terry Dossey
   National Security Agency
   Fort Meade, MD