



iALERT White Paper

Brute-Force Exploitation of Web Application Session IDs

By David Endler

Director, iDEFENSE Labs

dendler@idefense.com

November 1, 2001

iDEFENSE Inc.

14151 Newbrook Drive

Suite 100

Chantilly, VA 20151

Main: 703-961-1070

Fax: 703-961-1071

<http://www.idefense.com>

Copyright © 2001, iDEFENSE Inc.

"The Power of Intelligence" is trademarked by iDEFENSE Inc.

iDEFENSE and iALERT are Service Marks of iDEFENSE Inc.

TABLE OF CONTENTS

Introduction	3
Session IDs.....	4
Some Session ID Examples	4
COOKIES	4
STATIC URL WITH SESSION ID	5
HIDDEN INPUT FIELDS WITH SESSION ID	5
Susceptibility of Session IDs to Attack.....	6
Session ID Exploitation Mechanics.....	8
A URL Session ID Cracking Example.....	8
Bring on the Perl	10
Cracking More URLs.....	11
Hijacking Register.com (or the \$35 hack)	11
EXPLOITATION	14
Peeping at Others' Movies	16
Let's Crash The Party!.....	17
Cracking Cookies	19
Free Servers Indeed	19
Cracking Slash	21
Cracking Apache IDs.....	24
Conclusion.....	26
Acknowledgements.....	27
Resources.....	28
Appendix A: Cookie Collection Script.....	30
Appendix B: Sample Script to Brute-force 123greetings.com.....	31
Appendix C: Brute-forcing Register.com Domain Manager.....	32
Appendix D: Cracking Freeservers.com.....	33
Appendix E: More Session ID Sampling.....	34

INTRODUCTION

Almost all of today's "stateful" web-based applications use session IDs to associate a group of online actions with a specific user. This has security implications because many state mechanisms that use session IDs also serve as authentication and authorization mechanisms — purposes for which they were not well designed.

It is already well known that a user's web session is vulnerable to hijacking in a replay attack if these session IDs are captured or sniffed by an attacker. A replay attack involving a web application means that an attacker can use a session ID to log on to a user's account without the appropriate username or password. For example, by sniffing a URL that contains the session ID string, an attacker may be able to hijack a session simply by pasting this URL back into a web browser.

What is not well known is just how easily many of these session IDs can be guessed or brute-forced in order to conduct a replay attack. This eliminates the need for an attacker to guess someone's username and password on one of these websites.

Session IDs are usually long random alphanumeric strings transmitted between client and server either within cookies or directly in URLs. Once a user has logged into an application (e.g., Hotmail, Amazon, eBay, etc.), these session IDs can serve as stored authentication mechanisms so that the user does not have to retype a password after each click within the website. Ideally, during logon, a session ID is generated on the web server in such a manner that a potential attacker could not guess or calculate its value while the user's session is still active.

When strong cryptographic algorithms are used for this purpose, it is almost impossible to predict the next ID in a sequence generated by the same application. However, many of these applications generate session IDs in a linear or predictable manner, allowing an attacker to guess or brute-force them using automated programs. If a session ID can be forged or guessed, it saves the attacker from having to brute-force a user's legitimate logon credentials in order to access the account or hijack the active session.

This paper focuses on the ease with which many of these session IDs can be brute-forced, allowing an attacker to steal a legitimate web application user's credentials.

SESSION IDS

A session ID is an identification string used to associate specific web page activity with a specific user so that a sense of state is preserved for a web application. Session IDs can be used to preserve knowledge of the user across many pages and across historical sessions, enabling websites to provide features such as site personification (my.yahoo.com), online retail shopping carts (cdnow.com) and web-based e-mail (mail.yahoo.com, hotmail.com). Some web servers will generate a session ID for users after they visit any page on that server for the first time (Microsoft IIS, Apache, etc.). Additionally, other applications running on that web server (ATG Dynamo, BEA Weblogic, PHPNuke, etc.) may also generate more and different types of session IDs once the user has successfully authenticated.

Session IDs are often stored in a cookie held by the browser. Sometimes the cookies that store session IDs are set to expire (i.e., be deleted) immediately upon closing the browser; these are typically called “session cookies.” However, persistent cookies last beyond a user’s session. For instance, if the user has selected the “Remember Me” option on a website. Persistent cookies are usually stored on the user’s hard drive in a location according to the particular operating system and browser (for instance, c:\program files\netscape\users\username\cookies.txt for Netscape and c:\Documents and Setting\username\Cookies for IE on Windows 2000).

Session IDs can also be embedded in a static URL, dynamically rewritten URL, hidden in the HTML of a web page or some combination of these. Some examples of session IDs stored in static URLs include online greeting cards (bluemountain.com), invitations (evite.com) or password changing mechanisms (register.com) to name a few.

Some Session ID Examples

COOKIES

A typical cookie used to store a session ID (for redhat.com for example) looks much like:

www.redhat.com	FALSE	/	FALSE	1154029490	Apache	64.3.40.151.16018996349247480
----------------	-------	---	-------	------------	--------	-------------------------------

The columns above illustrate the six parameters that can be stored in a cookie.

From left-to-right, here is what each field represents:

domain: The website domain that created and that can read the variable.

flag: A TRUE/FALSE value indicating whether all machines within a given domain can access the variable.

path: Pathname of the URL(s) capable of accessing the cookie from the domain.

secure: A TRUE/FALSE value indicating if an SSL connection with the domain is needed to access the variable.

expiration: The Unix time that the variable will expire on. Unix time is defined as the number of seconds since 00:00:00 GMT on Jan 1, 1970. Omitting the expiration date signals to the browser to store the cookie only in memory; it will be erased when the browser is closed.

Session Hijacking

Exploiting TCP, UDP and HTTP Sessions

Shray Kapoor
shray.kapoor@gmail.com

Preface

With the emerging fields in e-commerce, financial and identity information are at a higher risk of being stolen. The purpose of this paper is to illustrate a common and valiant security threat to which most systems are prone to i.e. *Session Hijacking*. Sensitive user information is constantly transported between sessions after authentication and hackers are putting their best efforts to steal them. In this paper I will discuss mechanics of the act of session hijacking in TCP and UDP sessions i.e. hijacking at the network level and at Application levels i.e. hijacking HTTP sessions.

Table of Contents

Background

Introduction to TCP

Introduction to UDP

Introduction to HTTP

Hijacking at Network Levels

TCP session hijack

IP spoofing

Packet Sniffing (Middle Man Attack)

Blind attacks

UDP session Hijack Hijacking
at Application levels

HTTP session hijack

Obtaining Session ID's Countermeasures Summary

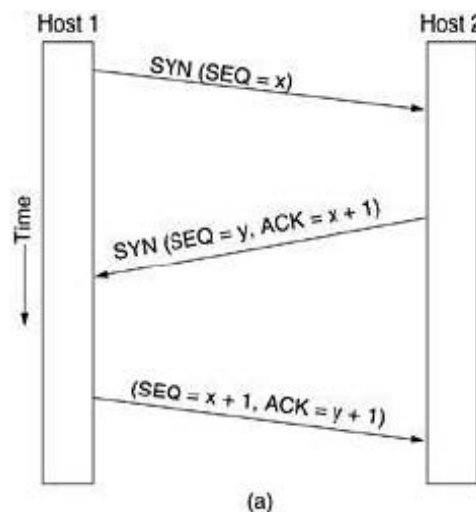
Background

Session hijacking can be done at two levels: *Network Level* and *Application Level*. Network layer hijacking involves TCP and UDP sessions, whereas Application level session hijack occurs with HTTP sessions. Successful attack on network level sessions will provide the attacker some critical information which will then be used to attack

application level sessions, so most of the time they occur together depending on the system that is attacked. Network level attacks are most attractive to an attacker because they do not have to be customized on web application basis; they simply attack the data flow of the protocol, which is common for all web applications.

Introduction to TCP

TCP an abbreviation for *Transmission Control Protocol*, one of the main connections oriented protocol in a TCP/IP network. TCP was formally defined in RFC 793 (while extensions are given in RFC 1323), as a protocol for providing a reliable end-to-end communication on a non-reliable network. To establish a session or a connection with a TCP server, a client must have to follow a structured system for session management; this system is known as “*Three Way Handshake*”. For two machines to communicate via TCP they must have to synchronize their session through Synchronize and Acknowledgement Packets. Every single packet is given a sequence number which helps the receiving host to synchronize and reassemble the stream of packets back into their original and intended order. TCP session establishment is shown in figure:



(Figure and session establishment summary taken from **Computer Networks by Andrew S. Tanenbaum, Prentice hall**)

1. Client sends a SYN request to server with initial sequence number X.
2. Server sends the SYN/ACK packet publishing its own Sequence number SEQ y and Acknowledgement number ACK for the client's original SYN packet. The ACK indicates the next SEQ number expected from client by the server.
3. Client acknowledges the receipt of the SYN/ACK packet from the server by sending the ACK number which will be the next sequence number expected from the server, y+1 in this case.

The following example shows the three-way handshake, using TCP dump to display the exchange:

```
tclient.net.39904 > telnet.com.23: S 733381829:733381829(0) win 8760 <mss  
1460> (DF)
```

```
telnet.com.23 > tclient.net.39904: S 1192930639:1192930639(0) ack 733381830  
win 1024 <mss 1460> (DF)  
tclient.net.39904 > telnet.com.23: . ack 1 win 8760 (DF)
```

(Reference from **New Riders Intrusion Detection 3rd** edition)

tclient at port 39904 attempting to establish session with telnet.com at 23 port with SEQ number marked by S (start:end(bytes)) flag ; publishing its Window size which is the buffer size i.e. 8760 in this case and also publishing Maximum Segment Size(mss).

Rest all communication follows the standard handshake mechanism After the session establishment its mere a matter of sending and receiving packets and increasing the sequence and the acknowledgement numbers accordingly.

Introduction to UDP

UDP is a User Datagram Protocol, unlike TCP, it does not provide connection oriented service. UDP does not use sequencing for session establishment and sending packets instead it is used for broadcasting messages across the network or for DNS or ARP queries. UDP is our second hijacking stage in Network level hijack attacks.

Introduction to HTTP

Hyper Text Transfer Protocol (HTTP) is a stateless protocol used by World Wide Web ; which defines how messages are formatted and transmitted between client and servers, and what actions Web servers and browsers should take in response to various commands. For establishing a connection with a server over HTTP: one has to establish a TCP connection on port 80 on the servers machine. Every session maintains a unique Session ID for the current live session with the server; which can be the target for stealing sessions. This is the last stage of session hijacking.

Hijacking at Network levels

Network level session attacks are done with TCP and UDP sessions, which are discussed in detail in the following sections.

TCP Session Hijack

TCP hijacks are meant to intercept the already established TCP sessions between any two communicating parties and then pretending to be one of them, finally redirecting the TCP traffic to it by injecting spoofed IP packets so that your commands are processed on behalf of the authenticated host of the session. It desynchronizes the session between the actual communicating parties and by intruding itself in between. As authentication is only required at the time of establishing connection, an already established connection can be easily stolen without going through any sort of authentication or security measures concerned. TCP session hijacks can be implemented in two different ways: Middle Man Attack (suggested by Lam, LeBlanc, and Smith) and the Blind attack. Before moving further there is need to understand IP spoofing which is discussed in the next subsection.

IP Spoofing: Assuming the identity

Spoofing is pretending to be someone else. This is a technique used to gain unauthorized access to the computer with an IP address of a trusted host. The trusted host in case of session hijacking is the client with whose IP address we will spoof our packets so that our packets will become acceptable to the server maintaining the session with the client. In implementing this technique session hijacker has to obtain the IP address of the client and inject his own packets spoofed with the IP address of client into the TCP session, so as to fool the server that it is communicating with the victim i.e. the original host.

What remains untouched is how to alter the sequence and the acknowledgement numbers of the spoofed packets which the server is expecting from the client. Once it is altered, hijacker injects its own forged packet in the established session before the client can respond, ultimately desynchronizing the original session, because now our server will expect a different sequence number, so the original packet will be trashed. Based on the anticipation of sequence numbers there are two types of TCP hijacking: Man in the Middle and Blind hijacking.

Man in the Middle attack using Packet Sniffers

This technique involves using a packet sniffer to intercept the communication between client and the server. Packet sniffer comes in two categories: Active and Passive sniffers. Passive sniffers monitors and sniffs packet from a network having same collision domain i.e. network with a hub, as all packets are broadcasted on each port of hub. Active sniffers works with Switched LAN network by ARP spoofing (For more information on Active Sniffers refer *Ethical Hacking and Countermeasures EC Council Exam 312 50 (OSB-2004)*). Once the hijacker reads the TCP header, he can know the sequence number expected by the server, the acknowledgement number, the ports and the protocol numbers; so that hijacker can forge the packet and send it to the server before the client does so.

Another way of doing so is to change the default gateway of the client's machine so that it will route its packets via the hijacker's machine. This can be done by ARP spoofing (i.e. by sending malicious ARP packets mapping its MAC address to the

default gateways address so as to update the ARP cache on the client , to redirect the traffic to hijacker).

Blind Attack

If you are not able to sniff the packets and guess the correct sequence number expected by server, you have to implement “*Blind Session Hijacking*”. You have to brute force 4 billion combinations of sequence number which will be an unreliable task.

UDP Session Hijacking

Since UDP does not use packet sequencing and synchronizing; it is easier than TCP to hijack UDP session. The hijacker has simply to forge a server reply to a client UDP request before the server can respond. If sniffing is used than it will be easier to control the traffic generating from the side of the server and thus restricting server’s reply to the client in the first place.

Hijacking Application Levels

At this level a hijacker can not only hijack already existing sessions but can also create new sessions from the stolen data.

HTTP Session Hijack

Hijacking HTTP sessions involves obtaining Session ID’s for the sessions, which is the only unique identifier of the HTTP session. Session ID’s can be found at three places

1. 1. In the URL received by the browser for the HTTP GET request.
2. 2. With cookies which will be stored in clients computer.
3. 3. Within the form fields.

Obtaining Session ID’s

One way to obtain the Session ID is by sniffing, which is same as the *Man in middle attack*. Cookies and URL’s can be sniffed from the packets and if unencrypted can provide critical user logon information.

Another way is by *Brute Forcing* the Session ID’s which involves trying a set of session id’s based on some pattern. Brute forcing is a time consuming task but worked on some algorithm can produce results rather quickly.

Countermeasures

To defend your network with session hijacking, a defender has to implement both security measures at Application level and Network level. Network level hijacks can be prevented by **Ciphering the packets** so that the hijacker cannot decipher the packet headers, to obtain any information which will aid in spoofing. This encryption can be provided by using protocols such as **IPSEC**, **SSL**, **SSH** etc. Internet security protocol (IPSEC) has the ability to encrypt the packet on some shared key between the two parties involved in communication. IPsec runs in two modes: Transport and Tunnel. In Transport Mode only the data sent in the packet is encrypted while in Tunnel Mode both packet headers and data are encrypted, so it is more restrictive.

To prevent your Application session to be hijacked it is recommended to use **Strong Session ID's** so that they cannot be hijacked or deciphered at any cost. **SSL (Secure Socket layer) and SSH (Secure Shell)** also provides strong encryption using SSL certificates so that session cannot be hijacked, but tools such as Cain & Bell can spoof the SSL certificates and decipher everything! **Expiring sessions** after a definite period of time requires re-authentication which will futile the hacker's tricks.

Summary

Session hijacking is a serious threat to Networks and Web applications on web as most of the systems are vulnerable to it. Although above explanation and countermeasures will give insight to the defender to protect his /her network, but it will also raise the security bar and will force the hijackers to apply more complex attacks to compromise the system. Networks should be tested and monitored continuously in order to make them impenetrable by the intruders.

The '*Strange Attractors and TCP/IP Sequence Number Analysis*' gives a great insight on Pseudo-Random Number Generators (PRNG) to avoid Blind attacks by guessing sequence numbers. This is available at:

<http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm>

An Introduction to ARP Spoofing

Sean Whalen

arpspoof@gmx.net

<http://chocobospore.org/arpspoof>

April, 2001

Revision 1.8

P U R P O S E

This paper deals with the subject of ARP spoofing. ARP spoofing is a method of exploiting the interaction of IP and Ethernet protocols. It is only applicable to Ethernet networks running IP.

The subject will be addressed such that anyone with basic networking experience can understand key points of the subject. Knowledge of the TCP/IP reference model is vital to full understanding, as is a familiarity with the operation of switched and non-switched networks. Some background will be presented in the "Introduction" section, but experienced readers may wish to skip to "Operation".

I N T R O D U C T I O N

A computer connected to an IP/Ethernet LAN has two addresses. One is the address of the network card, called the MAC address. The MAC, in theory, is a globally unique and unchangeable address which is stored on the network card itself. MAC addresses are necessary so that the Ethernet protocol can send data back and forth, independent of whatever application protocols are used on top of it. Ethernet builds "frames" of data, consisting of 1500 byte blocks. Each frame has an Ethernet header, containing the MAC address of the source and the destination computer.

The second address is the IP address. IP is a protocol used by applications, independent of whatever network technology operates underneath it. Each computer on a network must have a unique IP address to communicate. IP addresses are virtual and are assigned via software.

IP and Ethernet must work together. IP communicates by constructing "packets" which are similar to frames, but have a different structure. These packets cannot be delivered without the data link layer. In our case they are delivered by Ethernet, which splits the packets into frames, adds an Ethernet header for delivery, and sends them down the cable to the switch. The switch then decides which port to send the frame to, by comparing the destination address of the frame to an internal table which maps port numbers to MAC addresses.

When an Ethernet frame is constructed, it must be built from an IP packet. However, at the time of construction, Ethernet has no idea what the MAC address of the destination machine is, which it needs to create an Ethernet header. The only information it has available is the destination IP from the packet's header. There must be a way for the Ethernet protocol to find the MAC address of the destination machine, given a destination IP.

This is where ARP, the Address Resolution Protocol, comes in.

O P E R A T I O N

ARP operates by sending out "ARP request" packets. An ARP request asks the question, "Is your IP address x.x.x.x? If so, send your MAC back to me." These packets are broadcast to all computers on the LAN, even on a switched network. Each computer examines the ARP request, checks if it is currently assigned the specified IP, and sends an ARP reply containing its MAC address.

To minimize the number of ARP requests being broadcast, operating systems keep a cache of ARP replies. When a computer receives an ARP reply, it will update its ARP cache with the new IP/MAC association. As ARP is a stateless protocol, most operating systems will update their cache if a reply is received, regardless of whether they have sent out an actual request.

ARP spoofing involves constructing forged ARP replies. By sending forged ARP replies, a target computer could be convinced to send frames destined for computer A to instead go to computer B. When done properly, computer A will have no idea that this redirection took place. The process of updating a target computer's ARP cache with a forged entry is referred to as "poisoning".

A T T A C K S

S N I F F I N G

Switches determine which frames go to which ports by comparing the destination MAC on a frame against a table. This table contains a list of ports and the attached MAC address. The table is built when the switch is powered on, by examining the source MAC from the first frame transmitted on each port.

Network cards can enter a state called "promiscuous mode" where they are allowed to examine frames that are destined for MAC addresses other than their own. On switched networks this is not a concern, because the switch routes frames based on the table described above. This prevents sniffing of other people's frames. However, using ARP spoofing, there are several ways that sniffing can be performed on a switched network.

A "man-in-the-middle" attack is one of these. When a MiM is performed, a malicious user inserts his computer between the communications path of two target computers. Sniffing can then be performed. The malicious computer will forward frames between the two target computers so communications are not interrupted. The attack is performed as follows (where X is the attacking computer, and T1 and T2 are targets):

- X poisons the ARP cache of T1 and T2.
- T1 associates T2's IP with X's MAC.
- T2 associates T1's IP with X's MAC.
- All of T1 and T2's IP traffic will then go to X first, instead of directly to each other.

This is extremely potent when we consider that not only can computers be poisoned, but routers/gateways as well. All Internet traffic for a host could be intercepted with this method by performing a MiM on a target computer and the LAN's router.

Another method of sniffing on a switched network is MAC flooding. By sending spoofed ARP replies to a switch at an extremely rapid rate, the switch's port/MAC table will overflow. Results vary by brand, but some switches will revert to broadcast mode at this point. Sniffing can then be performed.

B R O A D C A S T I N G

Frames can be broadcast to the entire network by setting the destination address to FF:FF:FF:FF:FF:FF, also known as the broadcast MAC. By sweeping a network with spoofed ARP replies which set the MAC of the network gateway to the broadcast address, all external-bound data will be broadcast, enabling sniffing.

If a host were to listen for ARP requests and generate a reply containing the broadcast address, potentially crippling amounts of data could be broadcast on large networks.

D O S

Updating ARP caches with non-existent MAC addresses will cause frames to be dropped. These could be sent out in a sweeping fashion to all clients on the network in order to cause a Denial of Service attack. This is also a side effect of post-MiM attacks, since targeted computers will continue to send frames to the attacker's MAC address even after they remove themselves from the communication path. To perform a clean MiM attack, the target computers would have to have the original ARP entries restored by the attacking computer.

H I J A C K I N G

Connection hijacking allows an attacker to take control of a connection between two computers, using methods similar to the MiM attack. This transfer of control can result in any type of session being transferred. For example, an attacker could take control of a telnet session after a target computer has logged in to a remote computer as administrator.

C L O N I N G

MAC addresses were intended to be globally unique identifiers for each network interface produced. They were to be burned into the ROM of each interface, and not be changed. Today, however, MAC addresses are easily changed. Linux users can even change their MAC without spoofing software, using a single parameter to "ifconfig", the interface configuration program for the OS.

An attacker could DoS a target computer, then assign themselves the IP and MAC of the target computer, receiving all frames intended for the target.

T O O L S



A R P O I S O N

[HTTP://WEB.SYR.EDU/~SABUER/ARPOISON/](http://web.syr.edu/~sabuer/arpoison/)

ARPoison is a command-line tool for UNIX which creates spoofed ARP replies. Users can specify the source and destination IP/MAC addresses.



E T T E R C A P

[HTTP://ETTERCAP.SOURCEFORGE.NET](http://ettercap.sourceforge.net)

Ettercap is a powerful UNIX program employing a text-mode GUI, easy enough to be used by "script kiddies". All operations are automated, and the target computers are chosen from a scrollable list of hosts detected on the LAN.

Ettercap can perform four methods of sniffing: IP, MAC, ARP, and Public ARP. It also automates the following procedures:

- Injecting characters into connections
- Sniffing encrypted SSH sessions
- Password collection
- OS fingerprinting
- Connection killing



Parasite is a daemon which watches a LAN for ARP requests, and automatically sends spoofed ARP replies. This places the attacking computer as the MiM for any computer that broadcasts and ARP request. Eventually, this results in a LAN-wide MiM attack and all data on the switch can be sniffed.

Parasite does not do a proper clean up when stopped. This results in a DoS of all poisoned computers because their ARP caches are pointing to a MAC address that is no longer forwarding their frames. Poisoned ARP entries must expire before normal operation can resume.

D E F E N S E S

There is no universal defense against ARP spoofing. In fact, the only possible defense is the use of static (non-changing) ARP entries. Since static entries cannot be updated, spoofed ARP replies are ignored. To prevent spoofing, the ARP tables would have to have a static entry for each machine on the network. The overhead in deploying these tables, as well as keeping them up to date, is not practical for most LANs. Also of note is the behavior of static routes under Windows. Tests found that Windows still accepts spoofed ARP replies and updates the static entry with the forged MAC, sabotaging the purpose of static routes.

MAC cloning can be prevented by a feature found on high-end switches called Port Security (also known as Port Binding or MAC Binding). Port Security prevents changes to the MAC tables of a switch, unless manually performed by a network admin. It is not suitable for large networks, or networks using DHCP. Port Security does not prevent ARP spoofing.

Aside from these two methods, the only remaining defense is detection. Arpwatch is a free UNIX program which listens for ARP replies on a network. It will build a table of IP/MAC associations and store them in a file. When the MAC address associated with an IP changes (referred to as a flip-flop), an email is sent to an administrator.

Tests showed that running Parasite on a network caused a flood of flip-flops, leaving the MAC of the attacker present in Arpwatch's emails. Ettercap caused several flip flops, but would be difficult to detect on a DHCP-enabled network where flip flops occur at regular intervals.

MAC cloning can be detected by using RARP (Reverse ARP). RARP requests the IP address of a known MAC address. Sending a RARP request for all MAC addresses on a network could determine if any computer is performing cloning, if multiple replies are received for a single MAC address.

If a MAC flood is performed and the switch reverts to broadcast mode, a computer will have to enter promiscuous mode to examine the broadcast frames. Many methods exist for detecting machines in promiscuous mode. These can be found in the Sniffing FAQ, at <http://www.robertgraham.com/pubs/sniffing-faq.html>. Note that you can perform ARP spoofing without being in promiscuous mode since redirected frames will be routed to your MAC.

It is important to remember that Operating systems have their own TCP/IP stacks, and Ethernet cards have their own drivers, each with their own quirks. Even different versions of the same operating system have variations in behavior. Solaris is unique in its treatment of ARP replies. Solaris only accepts ARP updates after a timeout period. To poison the cache of a Solaris box, an attacker would have to DoS the second target machine in order to avoid a race condition after the timeout period. This DoS may be detected if the network has an Intrusion Detection System in place.

C L O S I N G

ARP spoofing is one of several vulnerabilities which exist in modern networking protocols, which allow a knowledgeable individual free reign over a network. IP spoofing, TCP sequence prediction, and ICMP redirects are just a few examples of other current weaknesses in these protocols. It is unlikely that these problems will be addressed until they are abused on a wide enough scale to force a change in the status quo. The problem is poised to grow as broadband Metropolitan Area Networks are implemented using Ethernet as the protocol of choice.

Information in this paper was heavily influenced by the Ettercap and Parasite projects. Proof of concept tests were performed with the tools mentioned in this paper, against Linux, Windows NT, and Windows 2000 machines.

Alberto Ornaghi <alor@antifork.org>
Marco Valleri <naga@antifork.org>

Man in the middle attacks

- What they are
- How to achieve them
- How to use them
- How to prevent them

Table of contents

Different attacks in different scenarios:

LOCAL AREA NETWORK:

- ARP poisoning
- Port stealing
- DNS spoofing
- STP mangling

FROM LOCAL TO REMOTE (through a gateway):

- ARP poisoning
- ICMP redirection
- DNS spoofing
- IRDP spoofing
- DHCP spoofing
- route mangling

REMOTE:

- DNS poisoning
- traffic tunneling
- route mangling

Once in the middle...

Sniffing

- It is the easiest attack to launch since all the packets transit through the attacker.
- All the “plain text” protocols are compromised (the attacker can sniff user and password of many widely used protocol such as telnet, ftp, http)

Hijacking

- Easy to launch
- It isn't blind (the attacker knows exactly the sequence numbers of the TCP connection)

Injecting

- Possibility to add packets to an already established connection (only possible in full-duplex mitm)
- The attacker can modify the sequence numbers and keep the connection synchronized while injecting packets.
- If the mitm attack is a “proxy attack” it is even easier to inject (there are two distinct connections)

Filtering

- The attacker can modify the payload of the packets by recalculating the checksum
- He/she can create filters on the fly
- The length of the payload can also be changed but only in full-duplex (in this case the seq has to be adjusted)

DNS Cache Poisoning: Definition and Prevention

Tom Olzak
March 2006

The Internet would grind to a halt – would not be possible – without a Domain Name System (DNS). As you'll see in this paper, the proper operation of DNS is fundamental to the maintenance and distribution of the addresses for the vast number of nodes around the globe. So it would be too much to hope for crackers (malicious hackers) to ignore DNS as they continuously look for new ways to circumvent your security.

There are several facets to DNS security. In this paper we focus on one of the most dangerous types of attack – DNS cache poisoning. To provide a complete picture of this threat, we'll explore how DNS works, two ways crackers facilitate cache poisoning, what impact this type of attack can have on your organization, and steps you can take to protect your information assets.

What is DNS?

In the world of the Internet and [TCP/IP](#), IP addresses are used to route [packets](#) from source to destination. A single IP address, for example 203.192.135.234, is not difficult to remember. But trying to learn or track thousands of these addresses, including which server/node is associated with each address, is a daunting task. So instead, we use [domain names](#) to refer to systems with which we want to communicate.

A real-world Internet domain name example is Google.com. When you enter the Google domain name into the address bar of your browser, the Google page appears. This is because your PC executed a process to resolve Google.com to an IP address. Only by having the IP address is a system able to initiate a session with another system across the Internet. Let's look at two ways IP address resolution can occur.

Figure 1 depicts the domain name/IP address resolution process when the target system and DNS server are internal. In this example, a workstation must establish a session with a server with a domain name of *Farpoint.company.com*. In order for a workstation to implement DNS, it must be running a DNS Client or Client *Resolver*. The resolver initiates the following process, resulting in the conversion of the domain name to an IP address (Microsoft TechNet, 2005).

Step 1: The resolver checks the resolver cache in the workstation's memory to see if it contains an entry for Farpoint.company.com. The entry would be present if the workstation had resolved the name to an IP address since the last time it was powered on, and the [Time to Live](#) of the entry had not been exceeded. In this example, no entry is found.

Step 2: Having found no entry in the resolver cache, the resolver sends a resolution query to the internal DNS server.

Step 3: When the DNS server receives the query, it first checks to see if it's authoritative for the company.com domain. In other words, is it responsible for managing the [zone](#) in which the company.com domain resides? If it is, the server performs a lookup in its internal zone table. In this case, it finds a host [Resource Record](#) (RR) that includes the IP address for Farpoint.company.com.

Step 4: The IP address of Farpoint.company.com is returned to the resolver.

Step 5: The resolved domain name and IP address are placed into the resolver cache. Figure 2 is an actual listing of the contents of a workstation's resolver cache. The IP address is used to contact Farpoint.

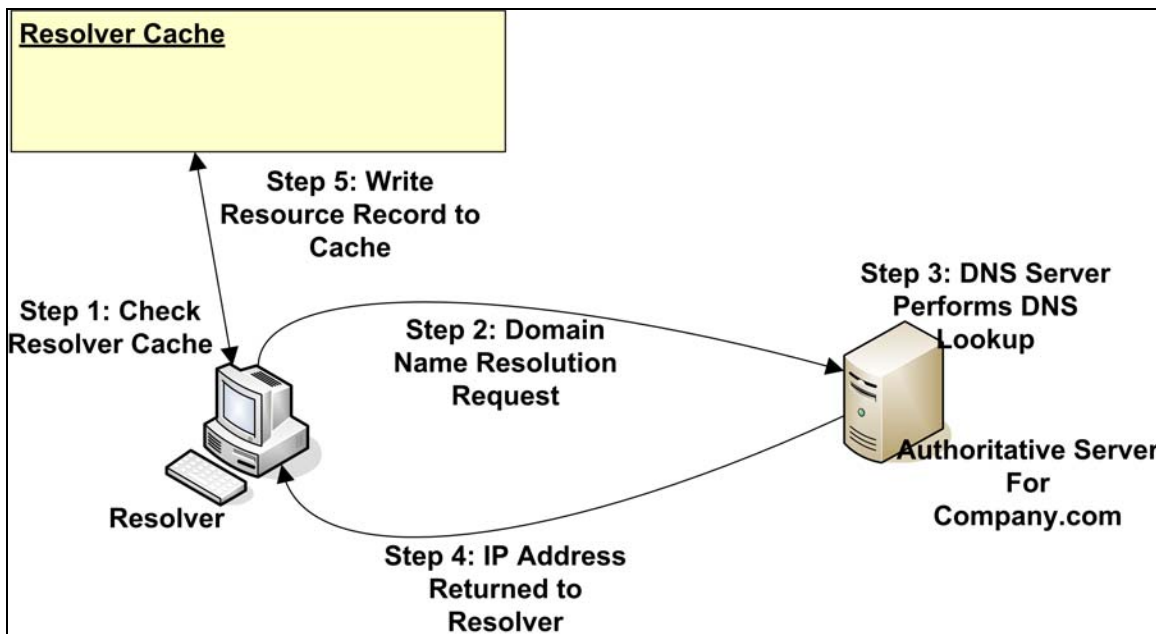


Figure 1: Internal DNS Server Lookup

In the previous example, the target server was located within the requestor's network. But there are many instances in which the target device is located somewhere on the Internet. In these cases, the process is somewhat different. Please refer to Figure 3 as we step through this second DNS resolution process.

Step 1: The resolver checks the resolver cache in the workstation's memory to see if it contains an entry for Farpoint.companyA.com.

Step 2: Having found no entry in the resolver cache, the resolver sends a resolution request to the internal DNS server.

```

F:\>ipconfig /displaydns

Windows IP Configuration

    1.0.0.127.in-addr.arpa
-----
Record Name . . . . . : 1.0.0.127.in-addr.arpa.
Record Type . . . . . : 12
Time To Live . . . . . : 575706
Data Length . . . . . : 4
Section . . . . . : Answer
PTR Record . . . . . : localhost

    technet2.microsoft.com
-----
Record Name . . . . . : technet2.microsoft.com
Record Type . . . . . : 1
Time To Live . . . . . : 314
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . : 207.46.196.114

    google.com
-----
Record Name . . . . . : google.com
Record Type . . . . . : 1
Time To Live . . . . . : 32
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . : 64.233.167.99

Record Name . . . . . : google.com
Record Type . . . . . : 1
Time To Live . . . . . : 32
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . : 72.14.207.99

```

Figure 2: Actual Resolver Cache Listing

Step 3: When the DNS server receives the request, it first checks to see if it's authoritative. In this case, it isn't authoritative for companyA.com. The next action it takes is to check its local cache to see if an entry for Farpoint.companyA.com exists. It doesn't. So in Step 4 the internal DNS server begins the process of iteratively querying external DNS servers until it either resolves the domain name or it reaches a point at which it's clear that the domain name entry doesn't exist.

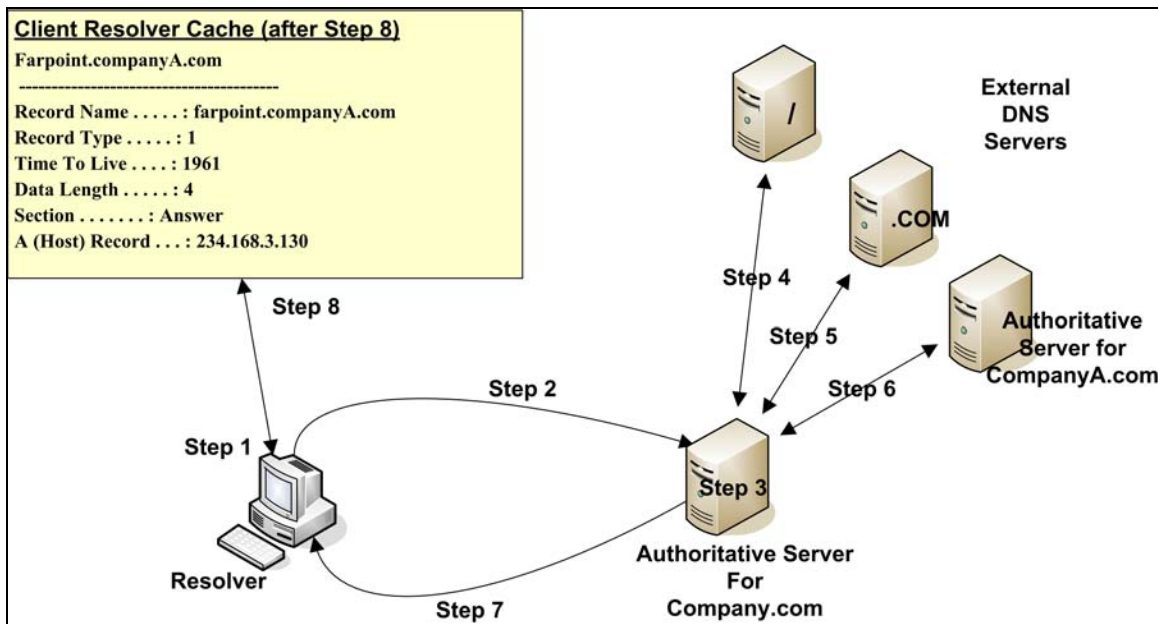


Figure 3: Recursive DNS Query

Step 4: A request is sent to one of the [Internet “root” servers](#). The root server returns the address of a server authoritative for the .COM Internet space.

Step 5: A request is sent to the authoritative server for .COM. The address of a DNS server authoritative for the companyA.com domain is returned.

Step 6: A request is sent to the authoritative server for companyA.com. The IP address of Farpoint.companyA.com is returned.

Step 7: The IP address for Farpoint is returned to the client resolver.

Step 8: An entry is made in the resolver cache, and a session is initiated with Farpoint.companyA.com.

This process, from the client resolver perspective, is known as a recursive query. With the proper software, the client resolver could perform the iterative query illustrated by Steps 4, 5, and 6.

Now that we have a good idea how DNS is supposed to work, it’s time to look at how this process can be used to co-opt one or more DNS caches.

What is DNS Cache Poisoning?

DNS cache poisoning consists of changing or adding records in the resolver caches, either on the client or the server, so that a DNS query for a domain returns an IP

address for an attacker's domain instead of the intended domain. To demonstrate how this might work, let's step through Figure 4.

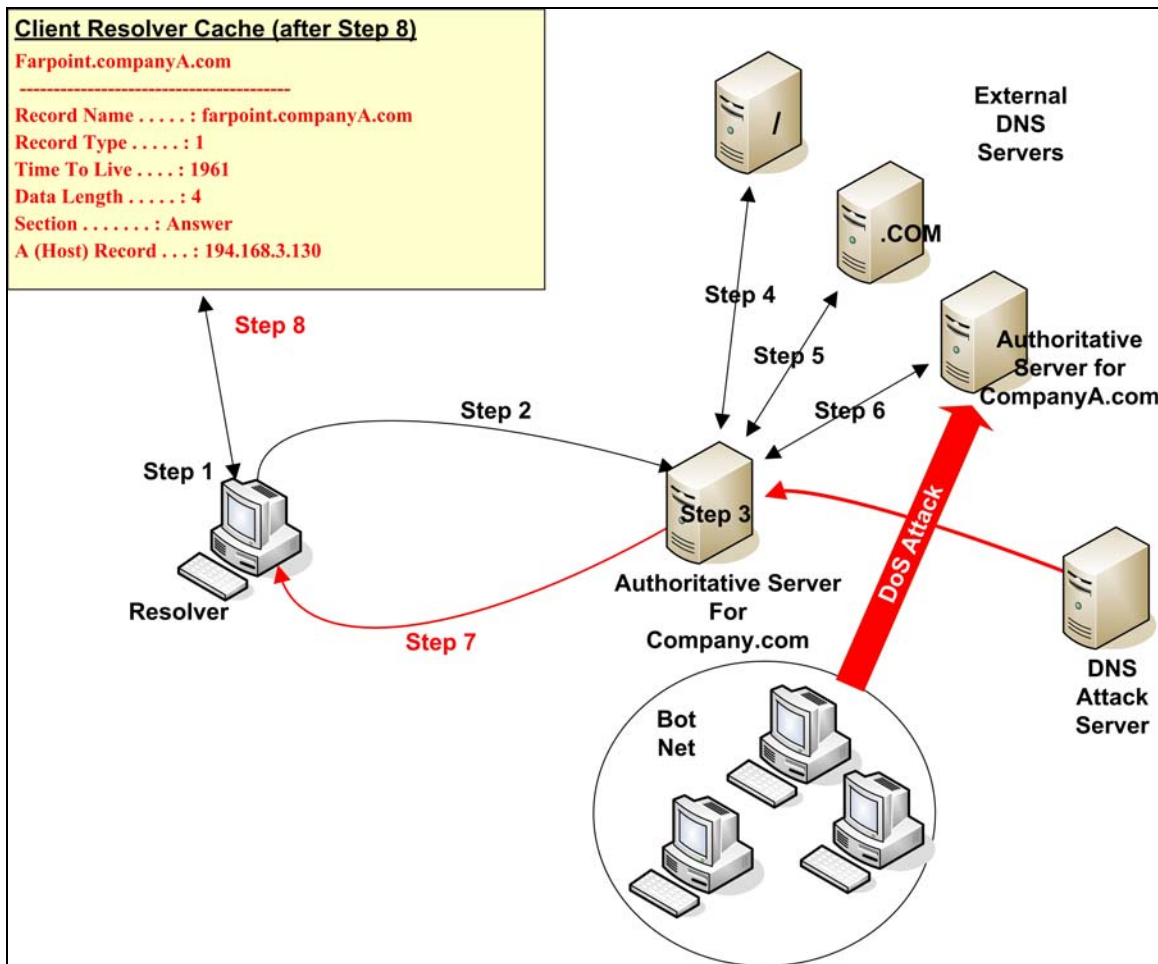


Figure 4: DNS Cache Poisoning

Step 1: The resolver checks the resolver cache in the workstation's memory to see if it contains an entry for Farpoint.companyA.com.

Step 2: Having found no entry in the resolver cache, the resolver sends a resolution request to the internal DNS server.

Step 3: When the DNS server receives the request, it first checks to see if it's authoritative. In this case, it isn't authoritative for companyA.com. The next action it takes is to check its local cache to see if an entry for Farpoint.companyA.com exists. It doesn't. So in Step 4 the internal DNS server begins the process of iteratively querying external DNS servers until it either resolves the domain name or it reaches a point at which it's clear that the domain name entry doesn't exist.

Step 4: A request is sent to one of the Internet root servers. The root server returns the address of a server authoritative for the .COM Internet space.

Step 5: A request is sent to the authoritative server for .COM. The address of a DNS server authoritative for the companyA.com domain is returned.

Step 6: A request is sent to the authoritative server for companyA.com. This is identical to the standard process for an iterative query – with one exception. A cracker has decided to poison the internal DNS server’s cache. In order to intercept a query and return malicious information, the cracker must know the transaction ID. Once the transaction ID is known, the attacker’s DNS server can respond as the authoritative server for companyA.com.

Although this would be a simple matter with older DNS software (e.g. BIND 4 and earlier), newer DNS systems have build-in safeguards. In our example, the transaction ID used to identify each query instance is randomized. But figuring out the transaction ID is not impossible. All that’s required is time. To slow the response of the real authoritative server, our cracker uses a [botnet](#) to initiate a Denial of Service (DoS) attack. While the authoritative server struggles to deal with the attack, the attacker’s DNS server has time to determine the transaction ID.

Once the ID is determined, a query response is sent to the internal DNS server. But the IP address for Farpoint.companyA.com in the response is actually the IP address of the attacker’s site. The response is placed into the server’s cache.

Step 7: The rogue IP address for Farpoint is returned to the client resolver.

Step 8: An entry is made in the resolver cache, and a session is initiated with the attacker’s site. At this point, both the workstation’s cache and the internal DNS server’s cache are poisoned. Any workstation on the internal network requesting resolution of Farpoint.companyA.com will receive the rogue address listed in the internal DNS server’s cache. This continues until the entry is deleted.

Another method used to poison a DNS cache is the use of a recursive query sent by the attacker. The query can force the target server to connect to the authoritative source of the domain in the query. Once connected, rogue information about one or more domains might be sent to the querying server and posted to the server’s cache.

There are other methods attackers use to poison DNS caches, but the objective is the same. Now we’ll explore the consequences of using a poisoned DNS cache.

Potential Consequences of Cache Poisoning

Pharming is the primary risk associated with cache poisoning. Crackers employ pharming for four primary reasons (Hyatt, 2006): identity theft, distribution of malware, dissemination of false information, and man-in-the-middle attacks.

Identity Theft

Once an attacker gets you to his site, he'll try to trick you into leaving behind information he can use to impersonate you. One way to do this in our first example is to create a site identical to the real Farpoint.companyA.com. When the user connects using the poisoned cache information, she might be fooled into entering information about herself through apparently legitimate requests for her name, social security number, address, etc.

Distribution of Malware

Another objective of attackers using cache poisoning is the automatic distribution of malware. Instead of releasing malicious code into the Internet and realizing random results, the use of rogue IP addresses to redirect unsuspecting users to the attacker's site can be a more focused attack vector. Once a workstation initiates a session with the malicious site, malware is uploaded to the workstation without intervention by or the knowledge of the user.

Dissemination of False Information

This aspect of pharming is useful to attackers who want to spread self-serving information about an organization. It's also been used to manipulate stock prices in an attempt to realize a large profit.

Man-in-the-middle Attack

In this attack type, the workstation initiates a session with the attacker's server. The attacker's server initiates a session with the actual target site. All information flowing between the workstation and the genuine site passes through and is intercepted by the cracker's server.

There can be serious consequences when security is an afterthought during the configuration and deployment of DNS servers. The next section provides guidelines that can help prevent cache poisoning.

Protecting Your Assets

The first layer of defense against cache poisoning is the use of the latest version of DNS. DNS based on BIND 9.3.x or Microsoft Windows Server 2003 is far more secure than DNS implemented with earlier versions. Successful completion of our first poisoning example would have been more difficult, because these systems also randomize the [port](#) used for the DNS query in addition to the transaction ID.

Recursive queries should be limited to internal DNS servers. If Internet facing recursive queries are required, only queries from internal addresses should be accepted. This will help prevent outside systems from sending queries with malicious intent. The following list provides additional guidance (Hyatt, 2005):

- ▲ Physically separate external and internal DNS servers
- ▲ Restrict [zone transfers](#) to authorized (secondary servers) devices

- ▲ Use [TSIG](#) to digitally sign zone transfers and zone updates – one of the best ways to prevent poisoning is to force identification of the sending authoritative source
- ▲ Restrict [dynamic DNS updates](#) when possible
- ▲ Hide the version of BIND being used on the DNS servers
- ▲ Remove unnecessary services running on the DNS servers
- ▲ Where possible, use dedicated appliances instead of multi-purpose servers

Conclusion

DNS cache poisoning is a large risk for organizations running early versions of DNS solutions. Elevated risk extends to companies that carelessly deploy DNS, regardless of the DNS software version used. Careful attention to version management and the secure configuration of DNS devices should reduce risk from cache poisoning to an acceptable level.

Works Cited

- Hyatt, M. (2005, June). *Five steps to minimize DNS cache poisoning*. Retrieved March 14, 2006 from http://searchwindowssecurity.techtarget.com/tip/1,289483,sid45_gci1101998,00.html
- Hyatt, R. (2006, January). Keeping DNS trustworthy. *The ISSA Journal*. January 2006, p. 37-38.
- Microsoft TechNet (2005, January). *How DNS query works*. Retrieved March 14, 2006 from <http://technet2.microsoft.com/WindowsServer/en/Library/1fd5b3be-ca29-43d0-b5e2-8a65192d5b781033.msp>

[Prevention from Session Hijacking](http://hydtechie.blogspot.com/2008/08/prevention-from-session-hijacking.html)

<http://hydtechie.blogspot.com/2008/08/prevention-from-session-hijacking.html>

Session hijacking is a technique that involves intercepting a TCP session initiated between two machines in order to take over the communication channel. The term session hijacking refers to the exploitation of a valid computer session. The Session's most important part is its session key. An attacker usually exploits this session related data to gain unauthorized access to information or services in a computer system. This technique is used to steal the cookies from a target system also called as a magic cookie, which is used to authenticate a user to a remote server. Sessions are of great importance to web developers, as the HTTP cookies used to maintain a session on most of the web sites. These cookies can be easily stolen by an attacker using an intermediary computer or with access to the saved cookies on the victim's computer. In this particular mode of hijacking the authentication check is performed only when the session is open, a hijacker who successfully launches this attack is able to take control of the connection throughout the duration of the session. If an attacker is able to steal the session cookie, he can pretend to be the same user, or hijack the session during its lifetime.

There are three primary techniques for hijacking sessions:

- Brute Force - the attacker tries multiple IDs until successful.
- Calculate - IDs are generated in a non-random manner and can be calculated.
- Steal - using different types of techniques, the attacker can steal the Session ID.

Methods to prevent session hijacking for Developers of a Website:

1. Regenerating the session id after a successful login. This prevents session fixation because the hacker/ attacker does not know the session id of the user after he has logged in.
2. Use a long random number or string as the session key. This reduces the risk that an hacker/ attacker could guess a valid session key through trial and error or brute force attacks.
3. Encryption of the data passed between the user and the web servers , specially the session key.
4. A web server could check with each request made matches the IP address of the use from previous sessions.
5. Can have services which change the value of the cookie with every request received.
6. Prevent Eavesdropping within the network.
7. Expire the session as soon as the use logs out .
8. Reduce the life span of a session or a cookie.

Methods to prevent session hijacking for USERS of a Website:

1. Do not click on the links forwarded to you through mails or IM's.

2. Make sure you flush out cookies and session from your browser after every confidential and sensitive transactions.
3. Do not simply close the browser , make sure that you click the log out button before closing the browser.
4. Use Firewalls
5. Restricts cookies to the maximum extend possible using the browser and firewall settings.
6. Clear History and offline content for every few days.
7. Prefer https rather than http for sensitive and confidential transactions.
8. Make sure that the website is certified by the certifying authorities.