Global Information Assurance Certification Paper

Advanced Incident Handling and Hacker Exploits

GCIH Practical Assignment
Version 2.1 Option 2

Support for the Cyber Defense Initiative
Port 22 – SSH
Man-in-the-Middle Attack

By:

Ronny LEPLAE, CISSP

© SANS Institute 2000 - 2005 Author retains full rights.

August 2002

Table of contents

Executive Summary

Cyber Defense Initiative

The CDI (Cyber Defense Initiative) is an initiative to identify frequently targeted services and the exploits commonly used against them[1]. By identifying the most widely-exploited vulnerabilities, focused lists can be created of problems that should be addressed immediately. A good example of such a list is the SANS Top Twenty.

How it works

The Internet Storm Center is a well-known example that uses techniques based on collecting a large number of log files from firewalls and intrusion detection systems. Currently the log files of more than 3000 firewalls and Intrusion Detection Systems in more than sixty countries are processed. Correlation and visualization techniques are used to analyze this data.[2]

The result

The result of this analysis is a global view on attack patterns, widely attacked services, top attacking systems and top attacked ports. This valuable information allows understanding the behavior of attackers.

About this paper

This paper will discuss the SSH (Secure Shell) protocol as an example of a widely attacked service on the Internet. The first part of the paper explains the protocol, the usage of the protocol and commonly known vulnerabilities. The second part explains a particular vulnerability of SSH when subject to a MiM (Man-in-the-Middle) attack.

[1] The SANS Institute http://www.sans.org

[2] The Internet Storm Center http://www.incidents.org/

Part I - Targeted Port

This section will discuss port 22, the services associated with port 22 and the known exploits for port 22.

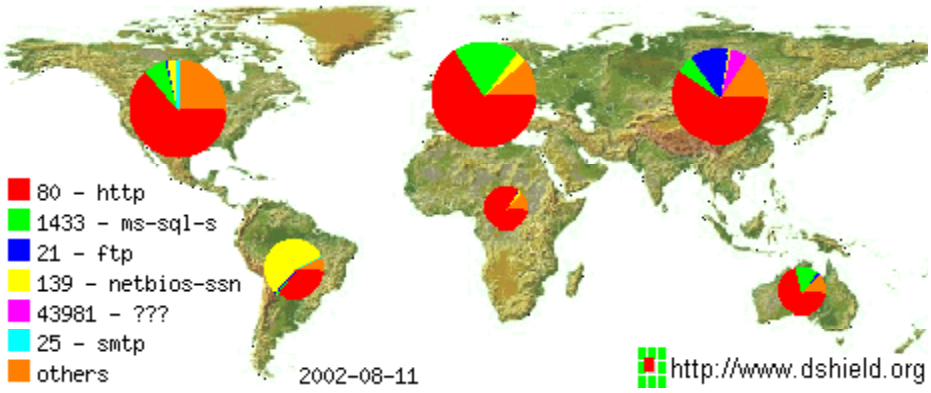
Why Port 22?

The Cyber Defense Initiative is an initiative that helps to understand, analyze and avoid the exploit of common vulnerabilities. One of the goals of the initiative is to compile lists of common attacked services. This is possible through the collection of log files, submitted by many participants, to a central server. This allows having a much broader view on what is going on.

Incidents.org is a virtual organization of advanced intrusion detection analysts, forensics experts and incident handlers from across the globe. The organization's mission is to provide real time "threat-driven" security intelligence and support to organizations and individuals.[3]

On the website of incidents.org it is possible to view the Top 10 of attacked services. The site gives both a geographical overview and a table per service.

The following diagram was downloaded from http://www.incidents.org on 11th August 2002. It shows the distribution of the most common attacks per continent.

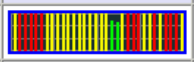
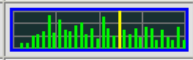
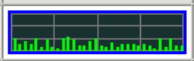
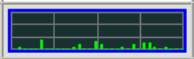
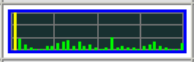
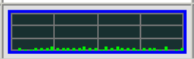
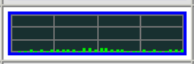
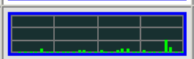
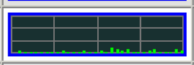
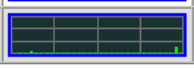


According to the information on the site of incidents.org, the overview is compilation of log file data, from more than 3000 firewalls and intrusion detection systems spread over sixty countries.

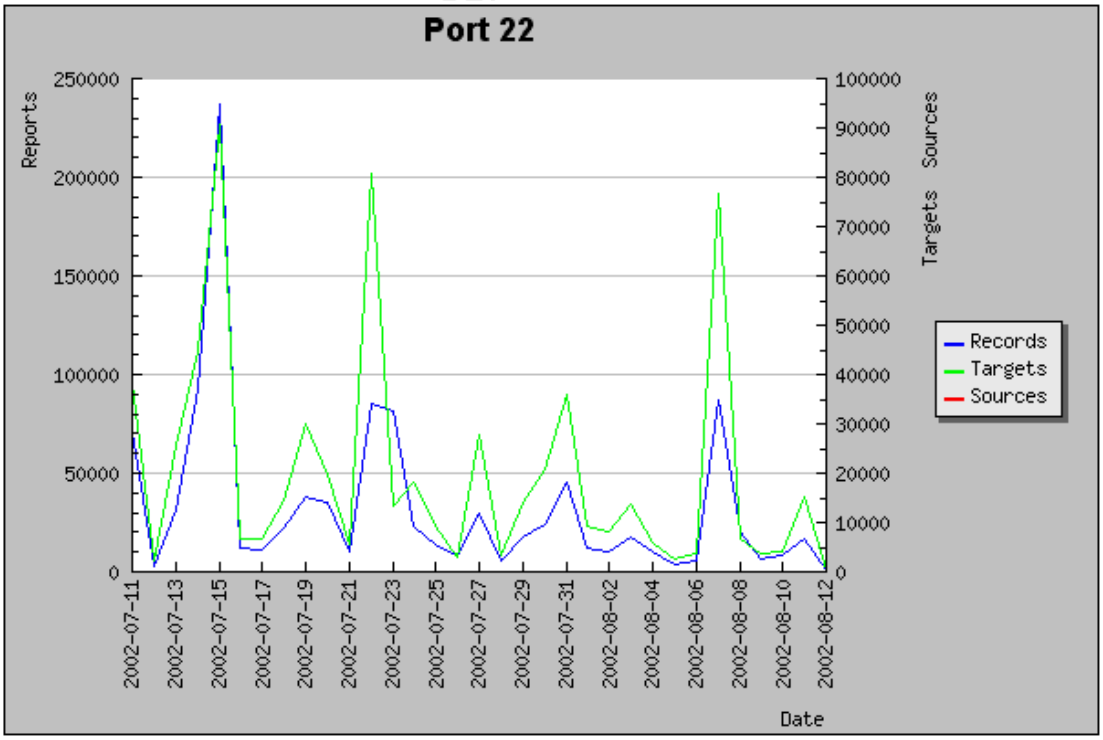
[3] http://www.incidents.org

To see the top 10 most attacked ports, a selection in the menu of the site http://www.incidents.org produces the following diagram:

Top 10 Ports

Service Name	Port Number	30 day history	Explanation
http	80		HTTP Web server
ms-sql-s	1433		Microsoft SQL Server
ftp	21		FTP servers typically run on this port
sunrpc	111		RPC. vulnurable on many Linux systems. Can get root
netbios-ssn	139		Windows File Sharing Probe
???	43981		Netware/IP
smtp	25		Mail server listens on this port.
ssh	22		Secure Shell, old versions are vulnerable
asp	27374		Scan for Windows SubSeven Trojan
???	6112		

On the above diagram, port 22 is ranked in 8th position. Clicking on the port number gives an attack history for port 22. The diagram shows the last month's results.



Services and Applications Running on Port 22

SSH (Secure Shell)

Well-known TCP-IP ports are registered by IANA and a list of currently well-known port numbers is available on the IANA web site. According to the IANA list, port 22 is a well-known port and is most commonly used for SSH.

PC Anywhere

PC Anywhere is a remote management software product sold by Symantec. The product allows for remotely administering computers.

Some older versions of this product used port 22. These versions are no longer supported by Symantec and have been replaced by newer versions, using other ports.

Adore SSHD

Adore SSHD is a Trojan Horse. A Trojan Horse is malicious software, hidden in another program. In this case the malicious code is hidden in the SSH daemon. This particular Trojan Horse uses Adore, a kernel RootKit to hide itself. A RootKit is software installed on a machine that will change the behavior of system calls making the software invisible to the administrator. The system calls will lie to the tools that call them. In this way, the Trojan Horse becomes invisible to a command to list active processes like: ps.

Adore SSHD allows hackers to enter the system with the password h4ck3d!

More information about Adore SSHD can be found on the following URL:
http://www.simovits.com/trojans/tr_data/y48.html

Shaft

Shaft is a Trojan Horse and a Distributed Denial of Service (DDoS) program. It can send packets in TCP, UDP or ICMP flood. Shaft can also steal passwords.

The next URL provides a detailed description on Shaft:
http://security.royans.net/info/posts/bugtraq_ddos3.shtml

Detailed Description of SSH

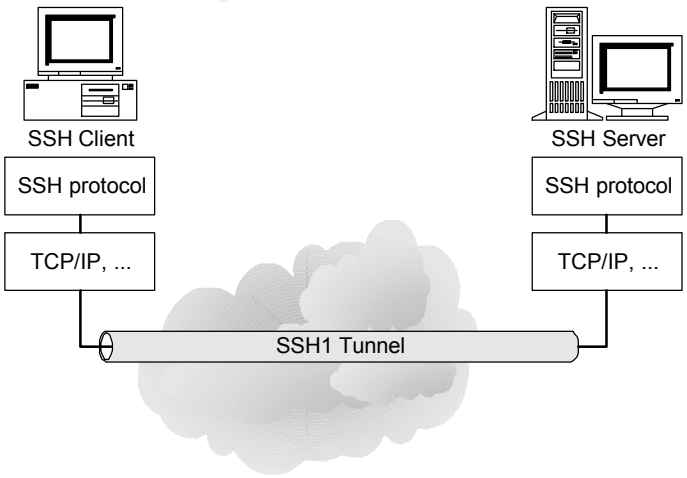
In order to discuss the SSH protocol we need to make a difference between SSH1 and SSH2. SSH2 was mainly developed as an answer to many previously discovered weaknesses in the SSH1 protocol. SSH2 is very different from SSH1.

SSH was developed in the first place to address the risks involved with using protocols like telnet, rlogon, rexec and rsh. All of these protocols are vulnerable to eavesdropping when using a sniffer. A sniffer is an application that allows inspecting network packets traveling over a network. Because protocols like telnet, rlogon, rexec and rsh do not encrypt their communication, very sensitive information such as usernames and passwords could fall in the hands of an eavesdropper.

SSH1

SSH1 was the first version of SSH and is a packet based protocol that can work on top of any transport layer. SSH1 provides an encrypted connection protecting all information exchanged between the server and the client from eavesdropping.

The following diagram shows an overview of the components:



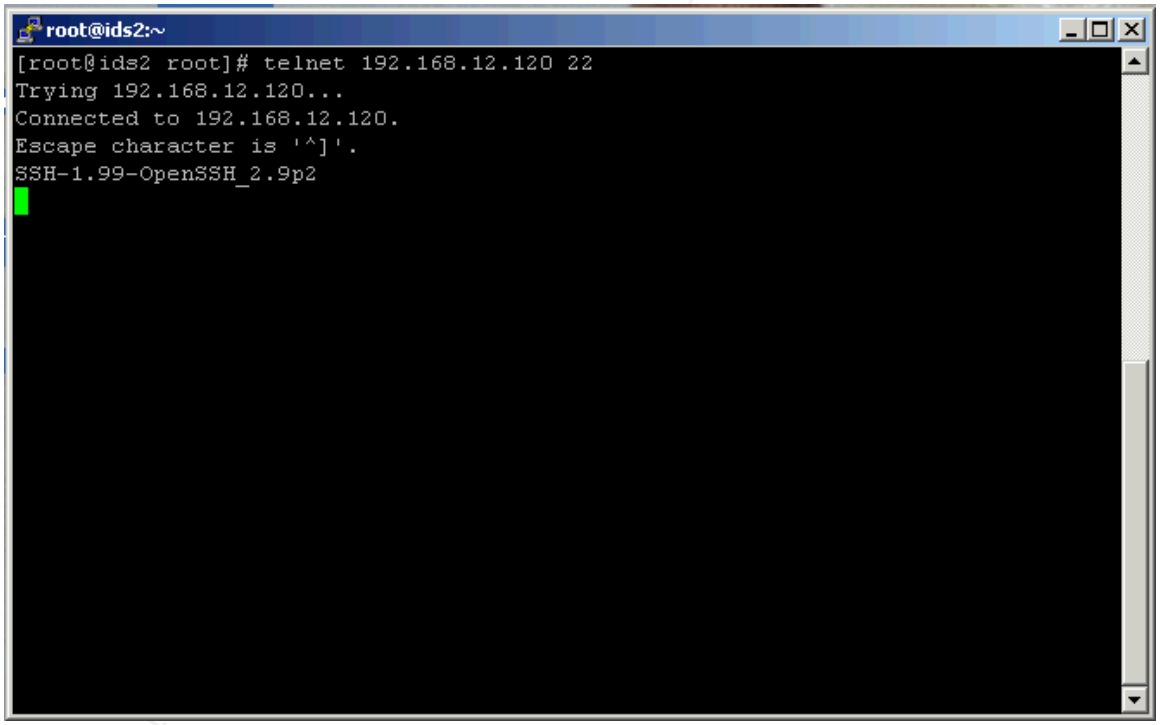
The diagram clearly shows the client part and the server part. When an SSH connection is established all information is sent and received encrypted. This creates a virtual pipe in which messages are passed encrypted between client

and server. Because the messages are encrypted they are not readable to an eavesdropper sniffing the traffic.

Before any data can be exchanged between the client and the server, a connection has to be established. A connection is negotiated between the client and the server. This is a very straightforward process and includes the following steps[4]:

1. **The SSH client contacts the SSH server and the SSH server sends a version string.**

   The client will open a socket connection to the server on port 22. This can be simulated with a telnet client overwriting the default port to use port 22 to connect. The following image is a screenshot of this simulation:



   As soon as the TCP connection is established, the server replies with a version string. In our simulation, we see "SSH-1.99-OpenSSH_2.9p2". This string tells the SSH client which version(s) of SSH is (are) supported by the server.

   SSH-1.5-… means the server is only capable of accepting SSH1.
   SSH-1.99-… means the server is capable of handling both SSH1 & SSH2.
   SSH-2.00-… means the server is only capable of accepting SSH2.

[4] SSH, The Secure Shell: The Definitive Guide, O'Reilly

In our simulation the SSH-1.99- part of the reply means that the server is capable of using both SSH1 and SSH2.

The part of the string after the second dash is disclosing information on the software used by the server. Often administrators will remove this label. This message is said to be 'leaking' additional information to hackers. Knowing the exact version and brand of the SSH server can make the job a lot easier to look for known vulnerabilities.

2. **The SSH client selects an SSH version and sends its version string to the SSH server.**

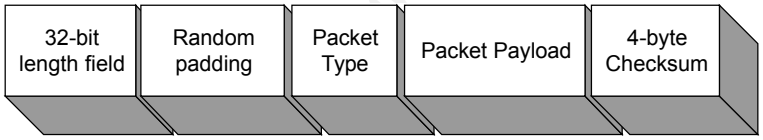
   Depending on the settings of the SSH client, the SSH client will choose a version of the SSH protocol to be used between client and server.

   A string is sent back to the SSH server similar to:

SSH-1.5-client	to establish an SSH-1 session
SSH-2.0-client	to establish an SSH-2 session

3. **The SSH client and SSH server switch to a packet-based protocol.**

   The SSH-1 packet layout looks like the following diagram:



   The random padding adds one to eight bytes of padding to thwart a known-plaintext attack. The packet type is one byte.

4. **The SSH server identifies itself to the SSH client and sends host and server key.**

   The server sends the following information to the client:
   - The RSA public key of the host called host key.
   - A second - hourly changing - RSA public key called server key.
   - Eight random bytes called check bytes.
   - List of encryption, compression and authentication methods supported by the server.

   Upon receipt of the host key, the SSH-1 client will check if it previously received the host key. If an SSH-1 session was previously never

established with the host, the SSH-1 client will ask the user to accept the host key after displaying the server's key fingerprint.

The following screen shows an SSH-1 client asking to accept the host key:

The host key is stored for future reference in a caching database at the client. If the host was found in the caching database and differs from the current received host key, the SSH-1 client will issue a warning. The next screen shows this warning:

5. The SSH client sends the SSH server a secret (session) key.

In the next step the SSH client will generate a symmetric key.

A symmetric key means that the same key is used to encrypt and decrypt

the message. Symmetric cryptography is much faster than asymmetric cryptography. Asymmetric cryptography allows encrypting a message with one key that can only be decrypted with the corresponding second key. Both keys are related and called a key-pair.

Because the SSH client received the public key from the SSH server it is able to securely transmit the session key encrypted with this public key. Only the SSH server has the corresponding private key and is able to decrypt the session key. In the implementation of SSH the session key is encrypted twice. It is encrypted once using the server's public host key and a second time with the server key.

The key is called a session key because every session is using a newly generated key.

6. **Both sides turn on encryption and complete server authentication.**

   Both sides can now encrypt and decrypt with the session key. The SSH client waits for the server to send a packet encrypted with the session key. This provides server authentication because only the server could have decrypted the session key with the private RSA key.

7. **The secure connection is established.**

   Since both ends of the connection now have the session key, messages can be exchanged encrypted. This completes the SSH connection establishment.

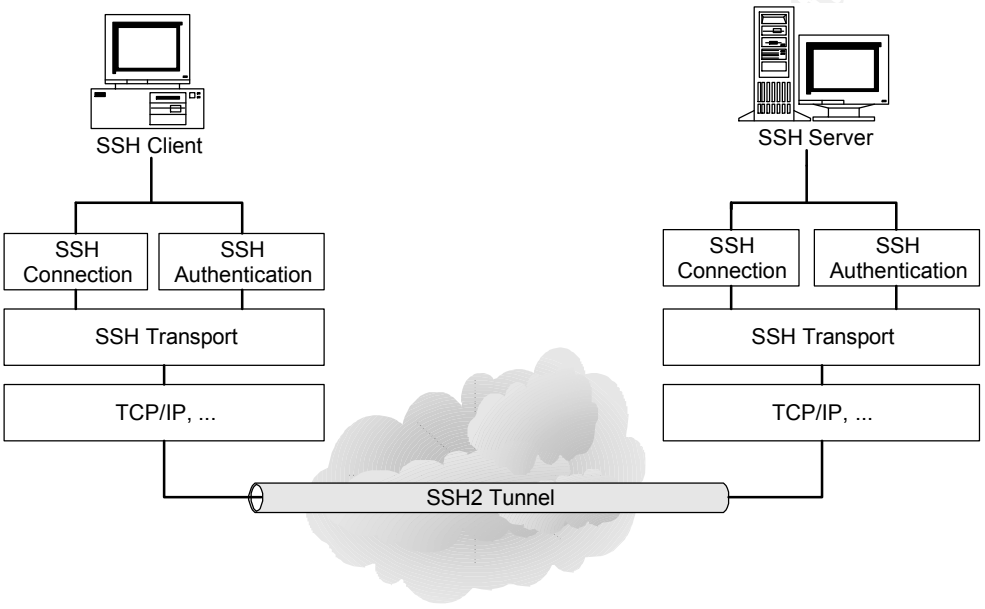
After the secure SSH1 connection is established, the server will want to authenticate the user before allowing access to any services. A mechanism called "User Authentication" is used for this. SSH1 supports the following user authentication mechanisms:

- Password
- Public-key

There are some other variants: Kerberos (central authentication server), SecureID (using a cryptographic hardware token), Rhosts (trusted host), TIS (OTP system from Trusted Information Systems, Inc).

SSH2

SSH2 is a different protocol from SSH1. SSH2 was developed to solve the weaknesses in the SSH1 protocol. Another difference is that SSH2 is no longer one layer, but is divided in 3 layers as shown in the following diagram:



Each of these layers is specified in a separate document and the overall architecture is described in a 4th document. The specifications can be found on the web site: http://www.ietf.org/internet-drafts/

The layers are called:

- SSH-TRANS. This layer is responsible for the server authentication, session key negotiation, privacy and integrity.
- SSH-AUTH. This layer is responsible for the client authentication.
- SSH-CONN. This layer is responsible for the multiplexing and various connection related issues.

SSH2 has a number of improvements over SSH1. The following is a list of some of these improvements:

- An important improvement of SSH2 is an extendable namespace for algorithms. The method offers a mechanism to support both standard and non-standard algorithms. The standard or registered names do not contain the "@"sign. Non-registered names include the "@" sign in the name. IANA[5] is the organization responsible for registering the names of the protocols.

- Another improvement offered by SSH2 over SSH1 is the support for multiple key exchange methods. The key exchange method is used to send the session key from the SSH client to the SSH server. Remember that in SSH1 the session key is sent double encrypted to the SSH server. SSH2 offers support for methods, which use a different way to exchange the session key. Currently SSH2 only uses Diffie-Hellman and all implementations must support this key exchange algorithm. Diffie-Hellman is a key exchange method based on exchanging a derivate of the session key, resulting in both sides ending up with the same secret without exchanging this secret.

- SSH2 also offers a stronger integrity check compared to SSH1. SSH1 is using the rather weak CRC32 integrity check. SSH2 supports Message Authentication Code (MAC) algorithms. A MAC is a cryptographic hash signed with a symmetric key.

- SSH2 offers session re-keying. This means that either side of an SSH2 connection can request to change the session key. The reason being that when a symmetric key is used for too long, the key could be discovered by a potential attacker. A large amount of encrypted data can facilitate a crypto attack. A solution for this weakness is periodically changing the session key, thereby making it a lot more difficult to break the session key.

Important to know is that most of the available SSH clients support both SSH1 and SSH2. This is because, from a commercial point of view, the offering is stronger if the product supports both protocols. In the second section of this paper, we will discuss the weakness resulting from this dual support and how this can be exploited in a Man-in-the-Middle attack.

[5] http://www.iana.org

Known Vulnerabilities of SSH

The NIST offers an excellent resource for searching vulnerabilities. The following table is an extract from http://icat.nist.gov/icat.cfm:

CVE-1999-0013	1/22/1998	Stolen credentials from SSH clients via SSH-agent program, allowing other local users to access remote accounts belonging to the SSH-agent user.
CVE-1999-1085	6/12/1998	SSH 1.2.25, 1.2.23, and other versions, when used in CBC (Cipher Block Chaining) or CFB (Cipher Feedback 64 bits) modes, allows remote attackers to insert arbitrary data into an existing stream between an SSH client and server by using a known plaintext attack and computing a valid CRC-32 checksum for the packet, aka the "SSH insertion attack."
CVE-1999-0310	9/1/1998	SSH 1.2.25 on HP-UX allows access to new user accounts.
CVE-1999-1321	11/5/1998	Buffer overflow in SSH 1.2.26 client with Kerberos V enabled could allow remote attackers to cause a denial of service or execute arbitrary commands via a long DNS hostname that is not properly handled during TGT ticket passing.
CVE-1999-1159	12/29/1998	SSH 2.0.11 and earlier allows local users to request remote forwarding from privileged ports without being root.
CAN-1999-0398	1/1/1999	In some instances of SSH 1.2.27 and 2.0.11 on Linux systems, SSH will allow users with expired accounts to login.
CAN-1999-0547	1/1/1999	An SSH server allows authentication through the .rhosts file.
CVE-1999-0248	1/1/1999	SSHD 1.2.17 can be compromised through the SSH protocol.
CAN-1999-1029	5/13/1999	SSH server (sshd2) before 2.0.12 does not properly record login attempts if the connection is closed before the maximum number of tries, allowing a remote attacker to guess the password without showing up in the audit logs.
CAN-1999-1231	6/9/1999	SSH 2.0.12, and possibly other versions, allows valid user names to attempt to enter the correct password multiple times, but only prompts an invalid user name for a password once, which allows remote attackers to determine user account names on the server.
CVE-1999-0787	9/17/1999	The SSH authentication agent follows symlinks via a UNIX domain socket.
CVE-1999-1010	12/14/1999	An SSH 1.2.27 server allows a client to use the "none" cipher, even if it is not allowed by the server policy.
CAN-2000-0143	2/11/2000	The SSH protocol server SSHD allows local users without shell access to redirect a TCP connection through a service that uses the standard system password database for authentication, such as POP or FTP.
CVE-2000-0217	2/24/2000	The default configuration of SSH allows X forwarding, which could allow a remote attacker to control a client's X sessions via a malicious xauth program.

CVE-2000-0532	6/7/2000	A FreeBSD patch for SSH on 2000-01-14 configures ssh to listen on port 722 as well as port 22, which might allow remote attackers to access SSH through port 722 even if port 22 is otherwise filtered.

CVE-2000-0525	6/8/2000	OpenSSH does not properly drop privileges when the UseLogin option is enabled, which allows local users to execute arbitrary commands by providing the command to the SSH daemon.
CAN-2000-0535	6/12/2000	OpenSSL 0.9.4 and OpenSSH for FreeBSD do not properly check for the existence of the /dev/random or /dev/urandom devices, which are absent on FreeBSD Alpha systems, which causes them to produce weak keys which may be more easily broken.
CVE-2000-0575	7/5/2000	SSH 1.2.27 with Kerberos authentication support stores Kerberos tickets in a file which is created in the current directory of the user who is logging in, which could allow remote attackers to sniff the ticket cache if the home directory is installed on N
CAN-2000-0784	10/20/2000	SSHD program in the Rapidstream 2.1 Beta VPN appliance has a hard-coded "rsadmin" account with a null password, which allows remote attackers to execute arbitrary commands via SSH.
CAN-2000-0999	12/11/2000	Format string vulnerabilities in OpenBSD SSH program (and possibly other BSD-based operating systems) allow attackers to gain root privileges.
CVE-2000-0992	12/19/2000	Directory traversal vulnerability in scp in SSHD 1.2.xx allows a remote malicious scp server to overwrite arbitrary files via a .. (dot dot) attack.
CVE-2000-1169	1/9/2001	OpenSSH SSH client before 2.3.0 does not properly disable X11 or agent forwarding, which could allow a malicious SSH server to gain access to the X11 display and sniff X11 events, or gain access to the SSH-agent.
CVE-2001-0080	2/12/2001	Cisco Catalyst 6000, 5000, or 4000 switches allow remote attackers to cause a denial of service by connecting to the SSH service with a non-SSH client, which generates a protocol mismatch error.
CVE-2001-0144	3/12/2001	CORE SDI SSH1 CRC-32 compensation attack detector allows remote attackers to execute arbitrary commands on an SSH server or client via an integer overflow.
CVE-2001-0155	6/2/2001	Format string vulnerability in VShell SSH gateway 1.0.1 and earlier allows remote attackers to execute arbitrary commands via a long user name.
CVE-2001-0156	6/2/2001	VShell SSH gateway 1.0.1 and earlier has a default port forwarding rule of 0.0.0.0/0.0.0.0, which could allow local users conduct arbitrary port forwarding to other systems.
CVE-2001-0259	6/2/2001	ssh-keygen in ssh 1.2.27 - 1.2.30 with Secure-RPC can allow local attackers to recover a SUN-DES-1 magic phrase generated by another user, which the attacker can use to decrypt that user's private key file.

CAN-2001-0471	6/27/2001	SSH daemon version 1 (aka SSHD-1 or SSH-1) 1.2.30 and earlier does not log repeated login attempts, which could allow remote attackers to compromise accounts without detection via a brute force attack.

CVE-2001-0361	6/27/2001	The SSH version 1.5 protocol allows a remote attacker to decrypt and/or alter traffic via an attack on PKCS#1 version 1.5 knows as a "Bleichenbacher attack". OpenSSH up to version 2.3.0, AppGate, and SSH Communications Security ssh-1 up to version 1.2.31 have the vulnerability present, although it may not be exploitable due to configurations.
CVE-2001-0364	6/27/2001	SSH Communications Security sshd versions 2.4 for Windows allows a remote attacker to create a denial of service via a large number of simultaneous connections.
CVE-2001-0529	8/14/2001	OpenSSH version 2.9 and earlier, with X forwarding enabled, allows a local attacker to delete any file named 'cookies' via a symlink attack.
CVE-2001-0553	8/14/2001	SSH Secure Shell 3.0.0 on Unix systems does not properly perform password authentication to the sshd2 daemon, which allows local users to gain access to accounts with short password fields, such as locked accounts that use "NP" in the password field.
CAN-2001-0572	8/22/2001	The SSH protocols 1 and 2 (aka SSH-2) as implemented in OpenSSH and other packages have various weaknesses which can allow a remote attacker to obtain the following information via sniffing: (1) password lengths or ranges of lengths, which simplifies brute force password guessing, (2) whether RSA or DSA authentication is being used, (3) the number of authorized_keys in RSA authentication, or (4) the lengths of shell commands.
CAN-2001-1029	9/20/2001	libutil in OpenSSH on FreeBSD 4.4 and earlier does not drop privileges before verifying the capabilities for reading the copyright and welcome files, which allows local users to bypass the capabilities checks and read arbitrary files by specifying alternate copyright or welcome files.
CAN-2001-1382	9/27/2001	The "echo simulation" traffic analysis countermeasure in OpenSSH before 2.9.9p2 sends an additional echo packet after the password and carriage return is entered, which could allow remote attackers to determine that the countermeasure is being used.
CAN-2001-1380	10/18/2001	OpenSSH before 2.9.9, while using keypairs and multiple keys of different types in the ~/.ssh/authorized_keys2 file, may not properly handle the "from" option associated with a key, which could allow remote attackers to login from unauthorized IP addresses.
CVE-2001-0816	12/6/2001	OpenSSH before 2.9.9, when running sftp using sftp-server and using restricted keypairs, allows remote authenticated users to bypass authorized_keys2 command= restrictions using sftp commands.
CVE-2001-0872	12/21/2001	OpenSSH 3.0.1 and earlier with UseLogin enabled does not properly cleanse critical environment variables such as LD_PRELOAD, which allows local users to gain root privileges.

CVE-2002-0083	3/15/2002	Off-by-one error in the channel code of OpenSSH 2.0 through 3.0.2 allows local users or remote malicious servers to gain privileges.

CAN-2002-0575	6/18/2002	Buffer overflow in OpenSSH before 2.9.9, and 3.x before 3.2.1, with Kerberos/AFS support and KerberosTgtPassing or AFSTokenPassing enabled, allows remote and local authenticated users to gain privileges.
CAN-2002-0639	7/3/2002	Integer overflow in sshd in OpenSSH 2.9.9 through 3.3 allows remote attackers to execute arbitrary code during challenge response authentication enabled (ChallengeResponseAuthentication) when OpenBSD is using SKEY or BSD_AUTH authentication.
CAN-2002-0640	7/3/2002	Buffer overflow in sshd in OpenSSH 2.3.1 through 3.3 may allow remote attackers to execute arbitrary code via a large number of responses during challenge response authentication when OpenBSD is using PAM modules with interactive keyboard authentication (PAMAuthenticationViaKbdInt).
CAN-2002-0460	8/12/2002	Bitvise WinSSHD before 2002-03-16 allows remote attackers to cause a denial of service (resource exhaustion) via a large number of incomplete connections that are not properly terminated, which are not properly freed by SSHd.
CAN-2002-0765	8/12/2002	sshd in OpenSSH 3.2.2, when using YP with netgroups and under certain conditions, may allow users to successfully authenticate and log in with another user's password.

Part II Specific Exploit on SSH: Man-in-the-Middle

Exploit Details

Description

The specific exploit discussed in this paper is a Man-in-the-Middle attack between an SSH client and an SSH server. In the introduction it was explained that the SSH protocol has a protection mechanism, warning the user of the SSH client about a Man-in-the-Middle attack. This warning is mainly because the SSH server key is different from the one used during previous SSH sessions.

A new vulnerability was found and published by Sebastian Krahmer on 1st July 2002, which allows avoiding the control mechanisms implemented in SSH clients to detect a Man-in-the-Middle attack. In early August 2002, Sebastian Krahmer also published a proof of concept application, demonstrating this vulnerability. In this exploit the SSH client does not issue the MiM warning to the user[6].

Variants

There have been other MiM attacks on SSH. The first MiM attack on SSH was probably done by Dug Song and is called monkey in the middle and is part of dsniff. More information on dsniff is found at the following URL: http://www.monkey.org/~dugsong/dsniff

Also ettercap is a sniffing tool that supports MiM attack on SSH. More detailed information on ettercap is available at the URL http://ettercap.sourceforge.net/

The main difference between these variants is that dsniff and ettercap are based on analyzing the packets, assembling, reassembling and forwarding them again.

The exploit discussed in this paper works on the banners. It also does not trigger the MiM warning included in the SSH client implementations. Therefore it goes almost undetected by the user.

[6] http://stealth.7350.org/ssharp.pdf - http://stealth.7350.org/7350ssharp.tgz

Systems Impacted

The vulnerability is based on the fact that an SSH client does not use the MiM attack warning when it receives a new key for the host and its cached key was for another signature algorithm or another SSH protocol version. Most SSH client software products support both SSH1 and SSH2 versions and many different cryptographic functions. By using this exploit and translating the SSH communication the MiM warning can be avoided.

Therefore, the problem is applicable to a very large number of systems using SSH. This includes but is not limited to:

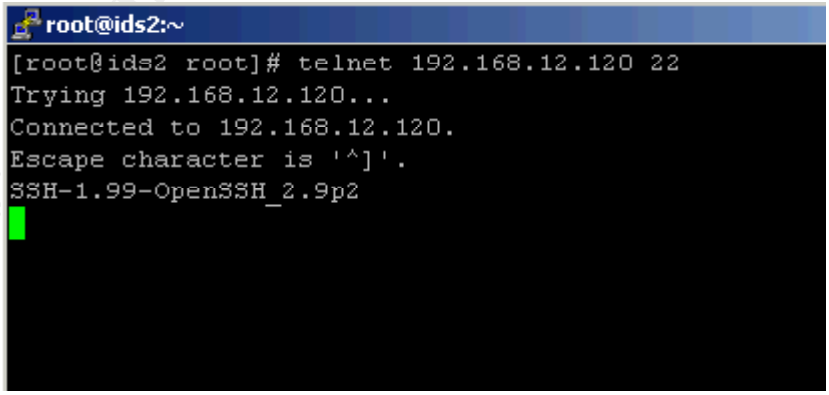
- Unix systems
- Windows systems having an SSH server
- Network appliances supporting SSH connections (Cisco, …)
- All SSH clients not updated for this exploit

Protocol Description

We explained in the introduction how SSH1 and SSH2 are used to make an encrypted network connection between an SSH client and a SSH server.

In the first step, when setting up an SSH connection, the client connects to the server and receives a string from the server. This string tells the client which versions of SSH the server is able to support.

The string can be obtained with a telnet tool as shown in this screenshot:



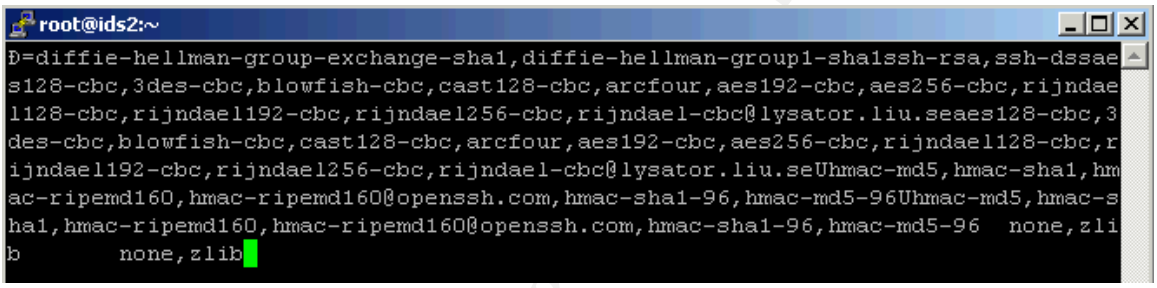
The server can reply with a number of possibilities:

SSH-1.5-… means the server is only capable of accepting SSH1.
SSH-1.99-… means the server is capable of handling both SSH1 & SSH2.
SSH-2.00-… means the server is only capable of accepting SSH2.

The client replies with a similar string to tell the server its choice about the SSH version to use. A string is sent back to the SSH server similar to:

SSH-1.5-client	to establish an SSH-1 session
SSH-2.0-client	to establish an SSH-2 session

The next screenshot shows the supported crypto algorithms received from the SSH server:



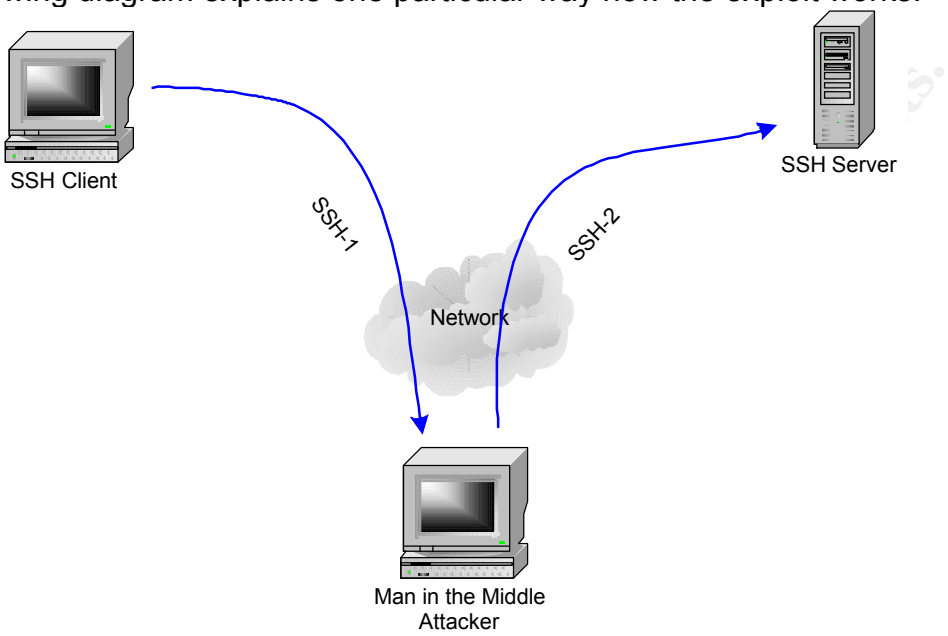
The client selects a crypto algorithm and a key-exchange is initiated. When both client and server have exchanged keys and have authenticated each other the SSH connection is established.

How it Works

The discussed exploit of Man-in-the-Middle will typically alter these messages and when receiving the strings from the server alter them or send other strings to the client. As such, the SSH version chosen by the client and the crypto algorithms used by the client are different from the SSH version and the crypto algorithms used on the server.

It is important to understand that this Man-in-the-Middle attack is based on a weakness in the SSH client. The SSH client stores the server key associated with the version of SSH and the version of digital signature used. If the key changes and the same type of connection is used a warning banner “Security breach” is displayed. If the key changes and the type of connection also changes then a banner is displayed asking the user to accept or refuse the new key.

The following diagram explains one particular way how the exploit works:



Because the SSH server key is stored at the SSH client and associated with the type of connection, a new connection, using another version of the protocol, will not issue the MiM attack warning screen. Instead, the client will ask the user to accept the new key and display a screen showing the fingerprint of the new key, issued by the Man-in-the-Middle server.

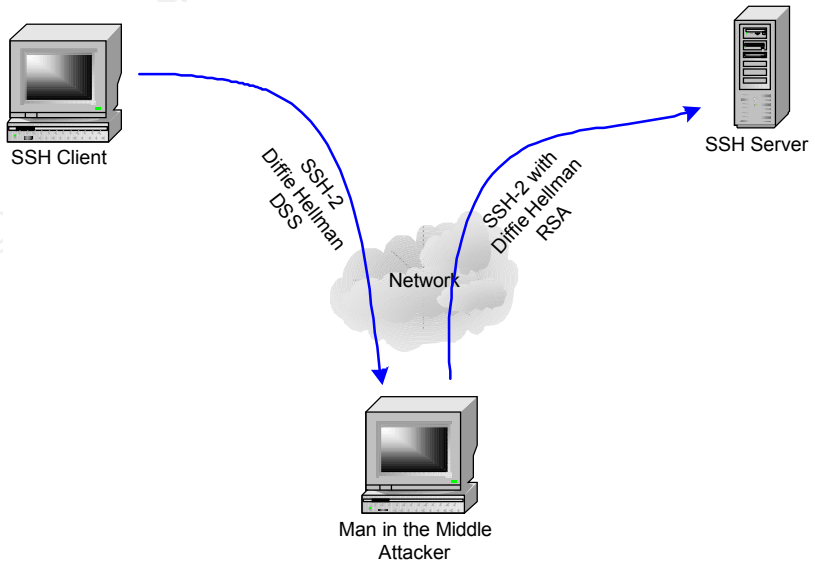
This is the screen, which should be displayed in the case of a MiM attack:

However, when the attacking server changes the version of the protocol, as in the previously explained diagram, a minor warning screen shows, telling the user no key is known:

It is important to note that the very first time the user connects to a server he also gets this screen and is told to accept the key. Because of this previous situation, the odds are that the user will again press 'YES'.

Changing the version of SSH between the client and the server is not the only way of exploiting and preventing the Security breach banner from being displayed. It is possible to have an SSH2 connection with the client and an SSH2 connection with the server. When the MiM attacker changes the digital signature algorithm, the key issued by the MiM system will also result in a request to the user, to accept the new key rather than issuing a MiM warning banner.

The following diagram explains this alternative exploit flow:



How to use the Exploit

In the next section, we will simulate and discuss the details of this exploit. We have logged all the packets and we will explain both the tools used and the logs captured for this exploit.

The Configuration

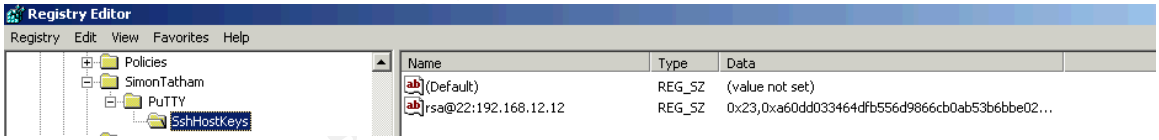
Our lab configuration consists of a small network with 3 machines:

- An SSH Server: Linux Redhat 7 with SSH server
- An SSH Client: W2000 With Putty SSH Client
- Attacking host: Linux Redhat 7 with SSharp from TESO.
  arpspoof from Dug Song.
  mss-server and mss-client from TESO.

The Various Steps

1st Step. The SSH client connects directly to the SSH server

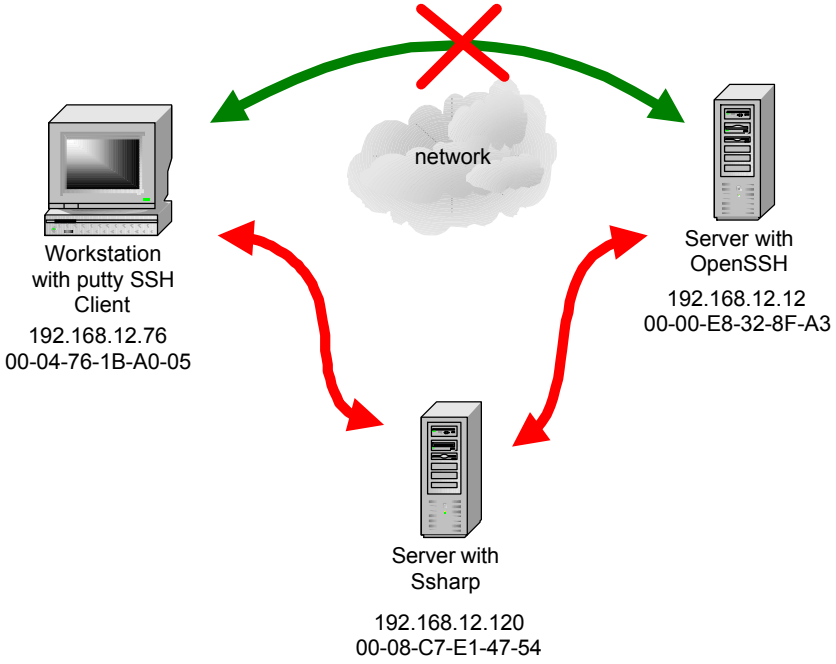
This is not part of the actual attack but it is important to look at this step because part of the MiM Attack protection mechanism is based on the fact that the SSH client stores the key of the SSH server the first time it connects to the SSH server. The next screen shot shows the key stored in the registry:



The SSH client will use this key when reconnecting to the SSH server and if it receives a different key it will issue a MiM attack warning. The weakness of this mechanism is that the key will only be compared if the SSH client is reconnecting with the same version of SSH to the server and if the SSH is using the same digital signature algorithm.

2nd Step. Reroute the traffic.

Normally the SSH client would connect directly to the SSH server (green arrow). In our lab configuration, we want all traffic rerouted to our attacking server, with SSharp[7] from TESO installed. The next diagram shows all the servers, their IP address and their MAC address.



In order to be able to modify traffic, we have to reroute the traffic to our system. Once the traffic is rerouted to our system, we can modify it. In order to have the SSH client and the SSH server send their traffic for each other to the attacking host, we need to do an ARP spoofing. An ARP spoofing works by sending ARP reply packets to both systems fooling them about the IP address of the other system.

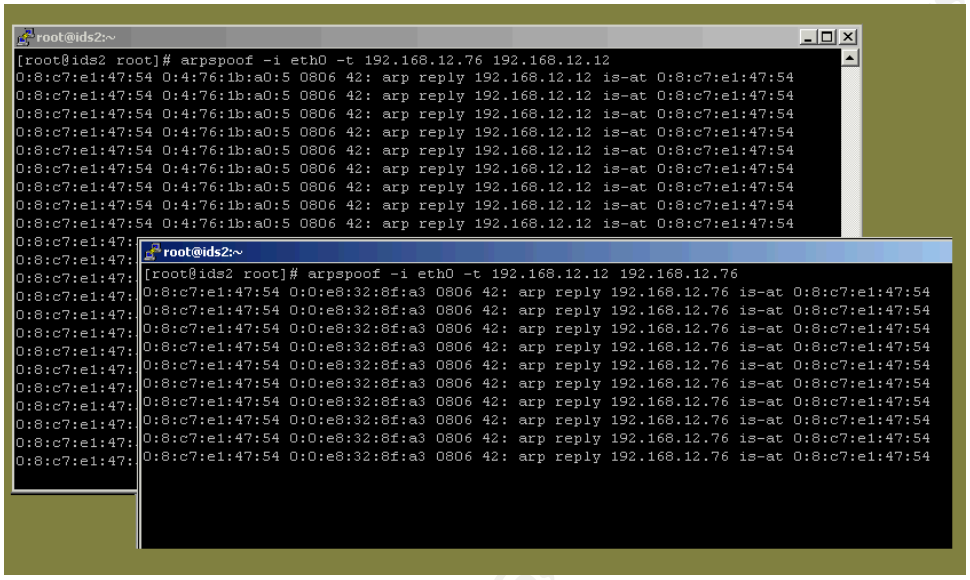
In order to establish this traffic deviation we use the tool that comes with dsniff called arpspoof. The syntax is very simple and looks like:

arpspoof –i eth0 –t 192.168.12.76 192.168.12.12
arpspoof –i eth0 –t 192.168.12.12 192.168.12.76

Both commands are running in separate windows. As long as the commands are running all traffic from the SSH client to the SSH server will be sent to the attacking host. All traffic from the SSH server to the SSH client will also be sent to the attacking host. When the attacking host has IP forwarding installed, neither party will notice anything at all.

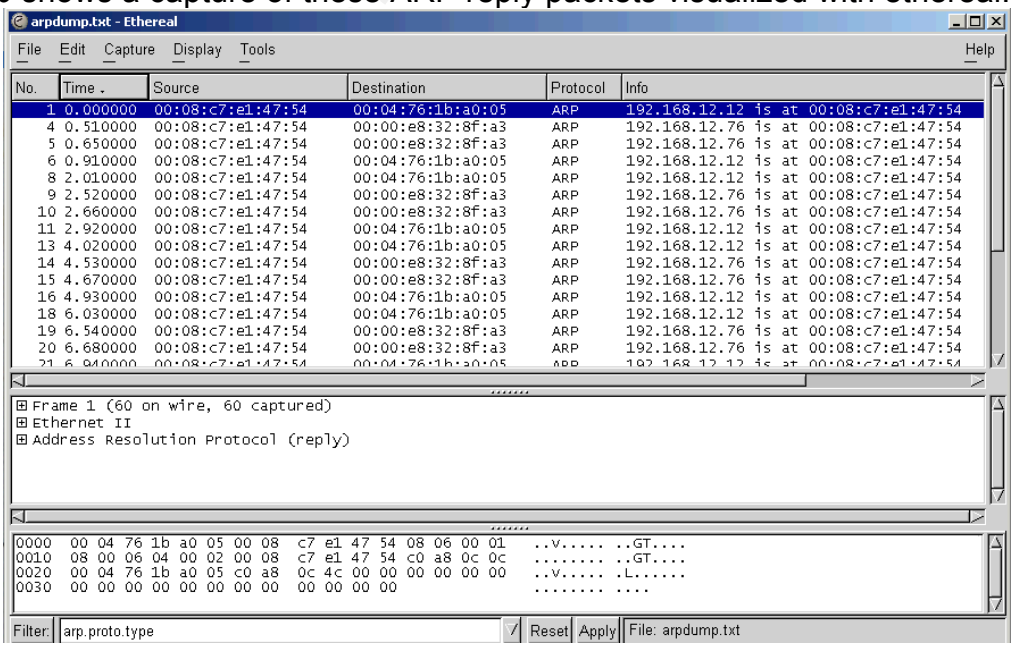
[7] SSharp is a tool that allows for MIM attack on SSH. It is available at http://stealth.7350.org/

Our first objective has been achieved; we have successfully rerouted all traffic to our attacking host. On each window at the attacking host, you can regularly see an ARP reply broadcast. The following is a screenshot from the attacking host:



Note the MAC address 00-08-C7-E1-47-54 which is used by both the SSH client and SSH server to identify each other. This is the MAC address from the attacking host.

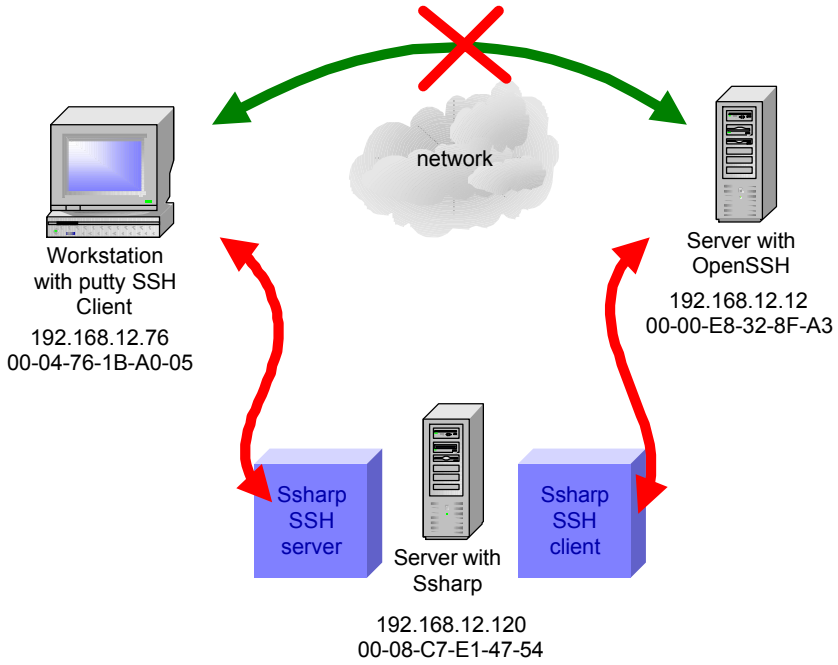
When we sniff the network, the ARP reply packets can be logged. The following dump shows a capture of these ARP reply packets visualized with ethereal.



3rd Step. Intercept the traffic.

In this step we install the Ssharp tool available from TESO at http://www.7350.org. The tool is based on a modified version of an SSH daemon. It consists of two parts: an SSH server part, that accepts incoming SSH connections, and a client part that connects to the original intended server. This mechanism makes it possible to have access to the communication in an unencrypted form between the SSH server and the SSH client on the attacking host.

The following diagram shows how it works:



Because we are interested in hijacking the session, we have configured it with the advanced USE_MSS option.

To make the Ssharp work, we run it as a daemon and make it listen on port 11000.
This can be done with the following command:

 # sshd -4 –p 11000

The traffic coming from the SSH client will be directed to port 22 and our daemon is listening to port 11000. To route the incoming traffic for port 22 to port 11000 we add a rule in iptables with the following command:

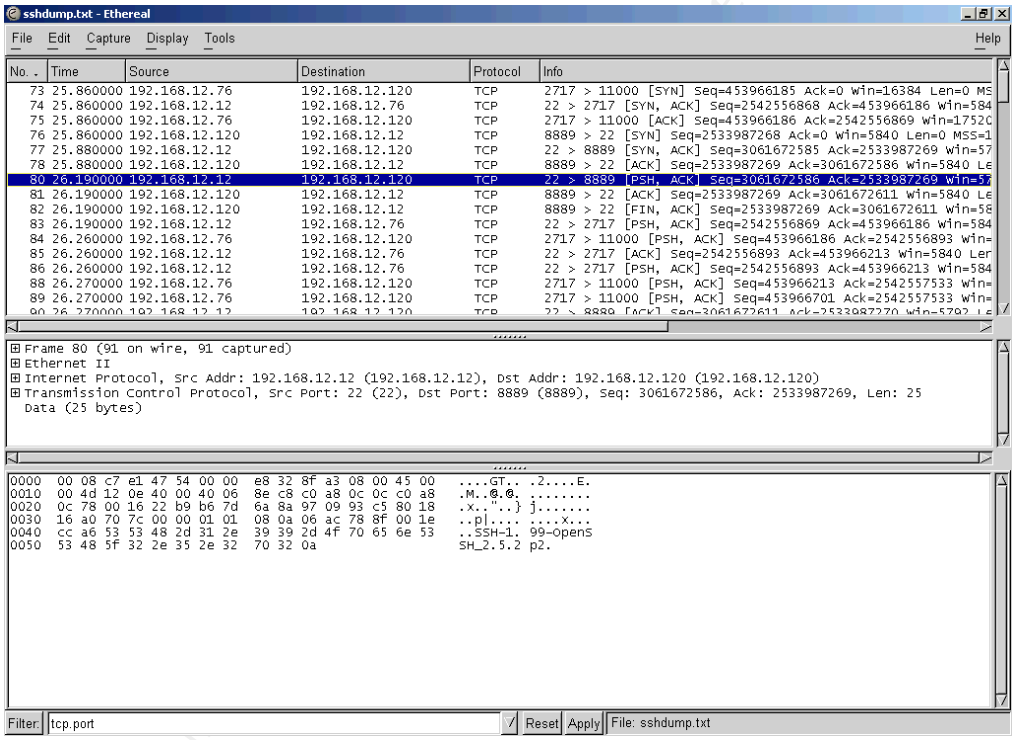
 # iptables –t –nat –A PREROUTING –p tcp --dport 22 --sport 1000:8000 –j

REDIRECT --to-ports 11000

We have now successfully installed our software and are ready to intercept any SSH connections made from the SSH client to the SSH server with our attacking host.

When the SSH client connects to the SSH server a pop-up window appears requesting to accept the new key. The user has no concern about this and clicks yes.

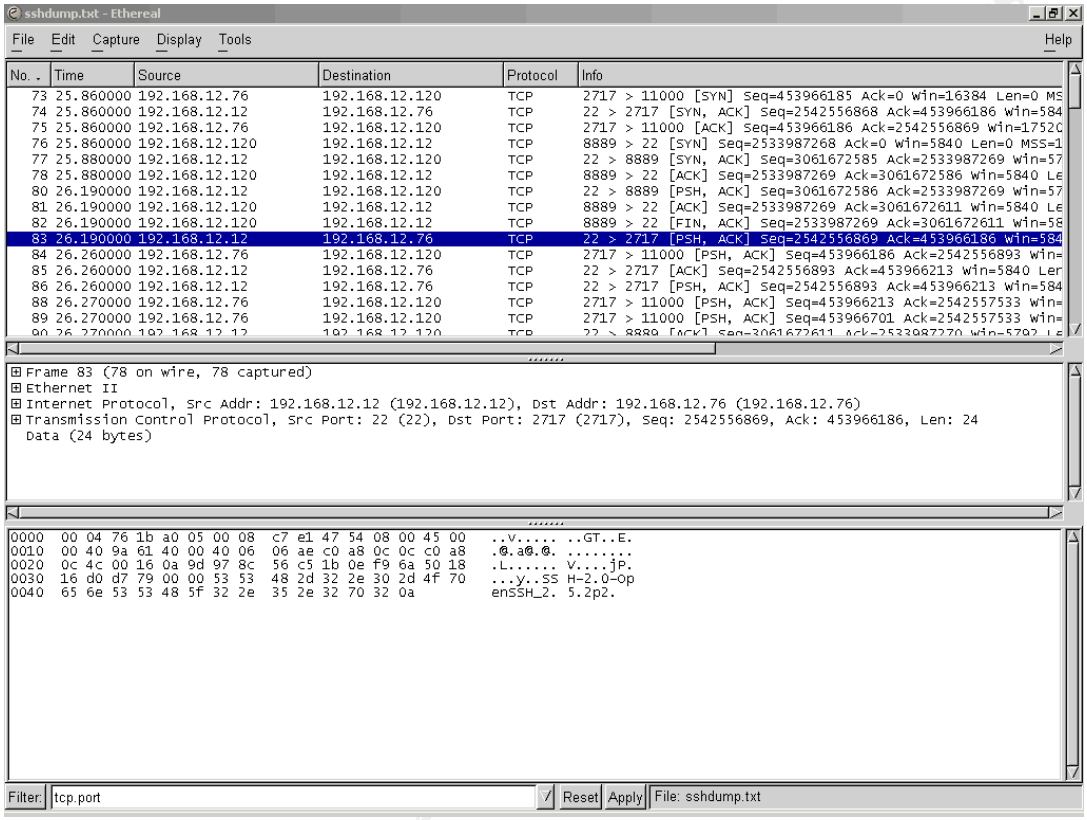
We are now successfully in between the SSH client and SSH server with our attacking host! If we look at a dump of the network packets, we see the following:



The first six lines are the TCP connection setup between the SSH client and the attacking host and between the attacking host and the SSH server. Note that in the second line, the source address is spoofed. This is the Ssharp on the attacking host replying to the SSH client and not the SSH server.

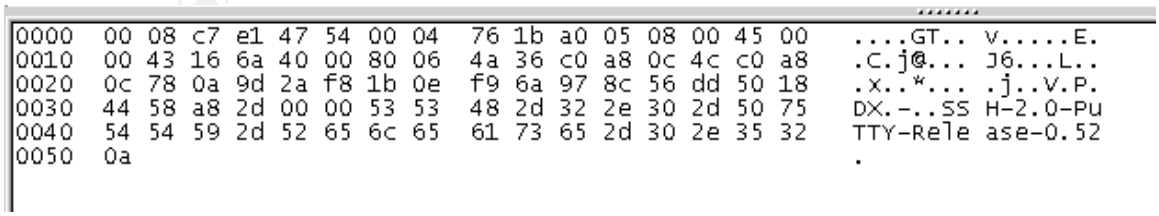
As soon as the connection between the attacking host and the SSH server is established, the SSH server replies with a packet on line 7. The detail can be seen in the lowest window of the previous Ethereal screen capture. We see in the contents of the packet the SSH server replying with “SSH-1.99-OPENSSH_2.5.2 p2”. This line means that the SSH server supports both SSH1 and SSH2.

The next interesting line is line 10 where the attacking host is replying to the SSH client with a spoofed IP address (as being the SSH server) and now look at the contents of the packet on the next screen capture!



The packet data has been altered by Ssharp and tells the SSH client the host is only capable of supporting SSH2. This is achieved with the string “SSH-2.0-OpenSSH_2. 5.2p2.”

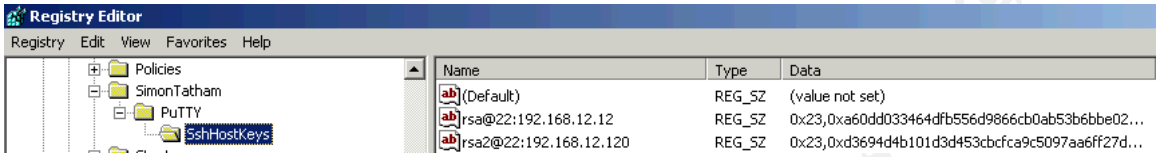
In the next packet the SSH client replies with the string “SSH-2.0-PuTTY-Release-0.52” confirming an SSH2 connection between the SSH client and the attacking host.



Now the attacking host and SSH client will exchange the protocols they both support and agree on a key exchange and cipher block protocol. Using this technique, Ssharp can send a new key to the SSH client. A window appears

asking the user to accept the new key. The user does not expect anything malicious and accepts the key. The logon prompt appears and the user has a working SSH connection to the server.

A close look to the registry shows that our SSH client has now two keys:



A second key is now stored in the registry and is used for the SSH2 connection to the Ssharp MiM attacking host.

4th Step. Hijacking the connection.

The user is not worried about anything and works on the SSH server.

At the attacking host, everything is in place to hijack the session. The directory /tmp contains a number of interesting files.

Password hijacking

The first interesting file is */tmp/ssharp.* This file stores all username/password pairs used by user sessions where the attack host was successfully in the middle.

```
[root@ids2 tmp]# ls -l
total 92
-rw-r--r--    1 root     root        11513 Aug  8 18:27 install.log
-rw-r--r--    1 root     root            0 Aug  8 18:11 install.log.syslog
drwx------    2 root     root         4096 Aug 17 14:10 orbit-root
-rw-------    1 apache   apache      18238 Aug  9 15:56 sess_a1f0a7b4712b01d61f2eb73f19595b63
-rw-------    1 apache   apache      25501 Aug  9 15:20 sess_c0e9519949c4d5a377b5b9b6f79a0be5
-rw-------    1 apache   apache      18236 Aug  9 15:10 sess_fcf5c468d81eb7aecd1355d6cc72596b
-rw-------    1 root     root            0 Aug 17 12:17 session_mm.sem
drwxr-xr-x    2 root     root         4096 Aug 12 09:46 ssh
-rw-r--r--    1 root     root           65 Aug 17 16:05 ssharp
srwxr-xr-x    1 nobody   nobody          0 Aug 17 16:04 ssharp-192.168.12.120.1846
srwxr-xr-x    1 nobody   nobody          0 Aug 17 15:41 ssharp-192.168.12.12.1834
[root@ids2 tmp]#
[root@ids2 tmp]# cat ssharp
192.168.12.12:22 [root:xxxxxxxx]
192.168.12.120:22 [root:xxxxxx]
[root@ids2 tmp]#
```

Session hijacking

More interesting is hijacking the user session with the following command:

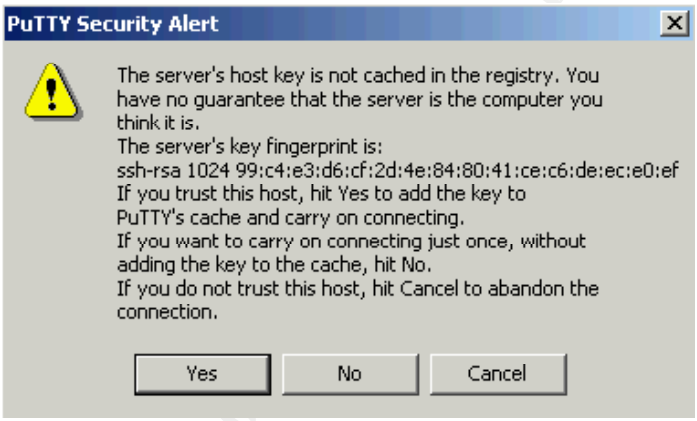
mss-client ssharp-192.168.12.12.1834

The attacker has now access to the user's session and can take over control and even kill the connection to the user. The session has been successfully hijacked!

Signatures

The signature of the attack is represented as a new key to the user. The user should be trained not to accept any new key from the SSH server without prior notice.

When a new unknown key is sent to the SSH client a screen appears as follows:



How to Protect Against

This is a good example of an attack where user awareness training can be extremely useful. Users who are trained to report any anomalies or react on suspicious events can assist considerably in detecting this event at an early stage.

- The best solution against the vulnerability is to patch the SSH client applications to issue a more severe warning when a new key is presented and another key was found for the same host using a different protocol:

- Alternatively the SSH server and the SSH client can be configured to use one specific protocol to avoid the possibility of a Man-in-the-Middle attacking system to have a two leg situation.

- Another protection against Man-in-the-Middle is to replace passwords with hardware identification tokens. It was explained that the Ssharp exploit stores the username and password. The attacker can use this information to gain access to the server.

  By using a hardware identification token the password is replaced with a challenge/response mechanism. The SSH server asks a different question on every session preventing the attacker from using a previously captured response.

- Users should be trained to be aware of the login banner. The banner provides information on the last session. This could give a clue about other people using a stolen username/password.

Source Code – Pseudo Code

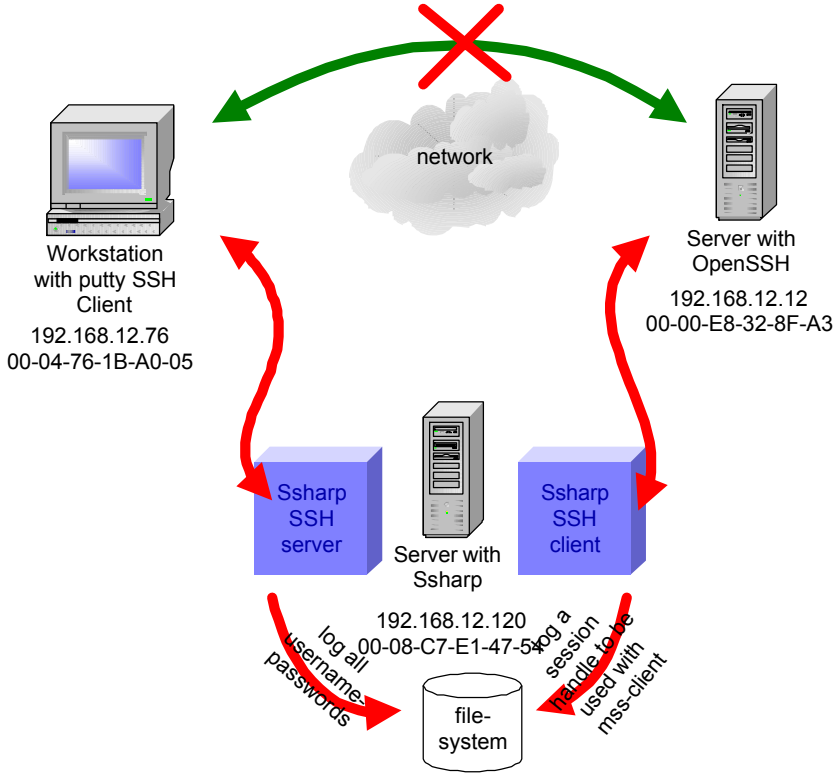
Source code for this vulnerability is available at http://stealth.7350.org

Some effort went into compiling the source code and make it work. The documentation provided with the source code is the bare minimum, but the tool works. The traces show how this proof of concept demonstrates vulnerability to exploitation.

The source contains an SSH server and an SSH client as explained on page 26. The SSH server listens for incoming calls and the SSH client continues the outgoing SSH connection to the original SSH server.

The package will create a different type of connection between the attacking host and the SSH client and between the attacking host and the SSH server. This allows exploiting the fact that the SSH client receives a key for the SSH server, which it never received before prompting the user to accept it. Because the key is associated with another context of connection (version / key algorithm) the severe security breach warning is not issued.

The following is a diagram of the Sshap package:



Important are the logging for username/password and the session handling that allows to hijack the SSH session.

Additional Information

Additional information on the subject can be found at various websites:

TESO website contains a proof of concept and the original paper of Sebastian Krahmer.
http://stealth.7350.org/

The internet engineering task force publishes a number of draft documents including standards and specifications on the SSH2 protocol.
http://www.ietf.org/internet-drafts/

Dug Song is probably the first having developed a MiM implementation for SSH. His development is part of the well-known package dsniff.
http://www.monkey.org/~dugsong/dsniff

Discussion of MiM attack on SSH in the Phrack 59 from 07-28-2002.
http://www.phrack.com/show.php?p=59&a=11

References

IANA provides a list of internet assigned TCP and UPD port numbers also referred to as well-known ports:
http://www.iana.org/assignments/port-numbers

Symantec support website provides more info on the port(s) used by the various PC Anywhere products:
http://service4.symantec.com/SUPPORT/pca.nsf/pfdocs/1998122810210812

The NIST provides an excellent website to allow querying known vulnerabilities in products:
http://icat.nist.gov/vt_portal.cfm

The internet engineering task force publishes all standards. The documents with details on the SSH protocol can also be found on this website:
http://www.ietf.org/internet-drafts/