

SSH for fun and profit

Sebastian Kraemer *kraemer@cs.uni-potsdam.de*

July 1, 2002

Abstract

The Secure Shell (SSH) protocol which itself is considered strong is often weakly implemented. Especially the SSH1/SSH2 interoperability as implemented in most SSH clients suffers from certain weak points as described below. Additionally the SSH2 protocol itself is also flexible enough to contain some interesting parts for attackers.

Disclaimer

This material ¹ is for educational purposes only. They demonstrate weaknesses in certain SSH implementations. All described tests and captures have been done in my own private LAN. It is strongly recommended that you do not test these programs without prior written permission of the local authorities.

The described attack-program will be made available one week after releasing this paper to give vendors time for fixes (which are rather trivial) to limit the possibility of abuse.

1 Introduction

In this paper I will describe how SSH clients can be tricked into thinking they are missing the hostkey ² even though they already have it in their list of known keys. This is possible due to some points in the SSH drafts which makes life of SSH developers harder but which was ment to offer special protection or more flexibility.

I assume you have a basic understanding of how SSH works. However it is not necessary to understand it all in detail because the attacks succeeds in the handshake where only a few packets have been exchanged. I also assume you are familiar with the common attacking scenarios in networks like "Man in the Middle" attacks, hijacking attacks against plaintext protocols, replay attacks and so on.

2 Playing with the banner

The SSH draft demands that both, client and server, exchange a banner before negotiating the key used for encrypting the communication channel. This is indeed needed for both sides to see which version of the protocol they have to speak. A banner commonly looks like

SSH-1.99-OpenSSH_2.2.0p1

¹this paper and provided programs

²for the host they connected to

A client obtaining such a banner reads this as "speak SSH1 or SSH2 to me". This is due to the 1 after the dash, the so called remote major version. It allows the client to choose SSH1 for key negotiation and further encryption. However it is also possible for the client to continue with SSH2 packets as the 99 tells him which is also called the remote minor version³. Depending on the clients configuration files and commandline options he decides to choose one of both protocols. Assuming the user does not force a protocol with either of the -1 or -2 switch most clients should behave the same way. This is due to the configuration files which do not differ that much across the various SSH vendors and often contain the line

```
Protocol 1,2
```

which makes the client choose SSH protocol version 1. It is obvious what follows now. Since the SSH client used to use SSH1 to talk to the server it is likely that he never spoke SSH2 before. This may be exploited by attackers to prompt a banner like

```
SSH-2.00-TESSO-SSH
```

to the client. The client looks up his database of known hosts and misses the hostkey because it only finds the SSH1 key of the server which does not help much because according to the banner he is not allowed to speak SSH1 anymore.⁴ Instead of presenting a warning like

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@   WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA1 host key has just been changed.
The fingerprint for the RSA1 key sent by the remote host is
f3:cd:d9:fa:c4:c8:b2:3b:68:c5:38:4e:d4:b1:42:4f.
Please contact your system administrator.
```

if someone tries MiM attacks against it without the banner-hack, it asks the user to just accept the new key:

```
Enabling compatibility mode for protocol 2.0
The authenticity of host 'lucifer (192.168.0.2)' can't be established.
DSA key fingerprint is ab:8a:18:15:67:04:18:34:ec:c9:ee:9b:89:b0:da:e6.
Are you sure you want to continue connecting (yes/no)?
```

It is much easier now for the user to type *yes* instead of editing the `known_hosts` file and restarting the SSH client. Once accepted, the attackers SSH server would record the login and password and would forward the SSH connection so the user does not notify his account was just compromised.

The described attack is not just an upgrade attack. It also works to downgrade SSH2 speaking clients to SSH1. If the banner would contain 2.0 the client only spoke SSH2 to the original server and usually can not know the SSH1 key of the server because he does not speak SSH1 at all. However our MiM server speaks SSH1 and prompts the client once again with a key he cannot know.

This attack will not work for clients which just support one protocol (likely to be SSH1) because it only implements one of them. These clients should be very seldom and most

³It is a convention that a remote-minor version of 99 with a remote-major version of 1 means both protocols

⁴since the remote major version number is 2

if not all ssh clients support both versions, indeed it is even a marketing-pusher to support both versions.

If the client uses RSA authentication there is no way for the attacker to get in between since he cant use the RSA challenges presented to him by the server because he is talking a different protocol to the client. In other words, the attacker is never speaking the same version of the protocol to both parties ⁵ and thus cannot forward or intercept RSA authentication.

A sample MiM program (*ssharp*) which mounts the banner-hack and records logins can be found at [7350ssharp].

3 Playing with the keys

It would be nice to have a similar attack against SSH without a version switch. This is because the version switch makes it impossible to break the RSA authentication.

Reading the SSH2 draft shows that SSH2 does not use the hostkey for encryption anymore ⁶. Instead the client obtains the hostkey to check whether any of the exchanged packets have been tampered with by comparing the server sent MAC⁷ with his own computed hash. The SSH2 draft is flexible enough to offer more than just one static algorithm to allow MAC computation. Rather it specifies that during key exchange the client and the server exchange a list of preferred algorithms they use to ensure packet integrity. Commonly DSA and RSA are used:

```
stealth@liane:~> telnet 192.168.0.2 22
Trying 192.168.0.2...
Connected to 192.168.0.2.
Escape character is '^]'.
SSH-1.99-OpenSSH_2.2.0p1
SSH-2.0-client
`$es%924D=)ydiffie-hellman-group1-shalssh-dss...
```

I deleted a lot of characters and replaced it with "...” because the interesting part is the `ssh-dss` which denotes the servers favorite algorithm used for MAC computation. Clients connecting to 192.168.0.2 cannot have a RSA key for computation because the server does not have one! Ofcourse the attackers MiM program has a RSA key and offers only RSA to ensure integrity:

```
stealth@liane:~> telnet 192.168.0.2 22
Trying 192.168.0.2...
Connected to 192.168.0.2.
Escape character is '^]'.
SSH-2.0-OpenSSH_2.9p1
SSH-2.0-client
at      seu>vME=diffie-hellman-group-exchange-sha1,
diffie-hellman-group1-shalssh-rsa...
```

A ssh client connecting to our MiM server will once again prompt the user to accept the new key instead of issuing the MiM warning.

The MiM server connected to the original server and got to know that he is using DSA. He then decided to face the user with a RSA key. If the original server offers DSA and RSA

⁵client and server

⁶as with SSH1 where the hostkey was sent to the client which sent back the sessionkey encrypted with the hostkey

⁷Message Authentication Code; the server computes a hash of the packets exchanged and signs it using the negotiated algorithm

the MiM server will wait until the client sends his preferred algorithms and will choose an algorithm the client is naming for his second choice. A RFC compliant SSH2 server has to choose the first algorithm from the client list he is supporting, our MiM server will choose the next one and thus produces a key-miss on client-side. This will again produce a yes/no prompt instead of the warning message. [7350ssharp] also supports the key-hack mode.

4 Countermeasures

Having the RSA hostkey for a server offering a DSA hostkey means nothing for today's clients. They ignore the fact that they have a valid hostkey for that host but in a different keytype. SSH clients should also issue the MiM warning if they find hostkeys for the server where either the version or type does not match. It's very likely someone is playing MiM games. In my eyes it is definitely a bug in the SSH client software.

5 An Implementation


There already exist some MiM implementations for SSH1 such as [dsniff] or [ettercap]. Usually they understand the SSH protocol and put much effort into packet assembling and deassembling or forwarding. Things are much simpler. *ssharp* is based on a normal OpenSSH daemon which was modified to accept any login/password pair and starts a special shell for these connections: a SSH client which is given the username/password and the real destination IP. It logs into the remote host without user-interaction and since it is bound to the mim servers pty it looks for the user like he enters his normal shell. This way it is not needed to mess with SSH1 or SSH2 protocol or to replace keys etc. We just play with the banner or the signature algorithm negotiation the way described above.

If compiled with USE_MSS option enabled, *ssharp* will slip the SSH client through a screen-like session which allows attaching of third parties to existing (mimed) SSH1 or SSH2 connections. It is also possible to kick out the legitimate user and completely take control over the session. Following a screenshot of such a hijacked SSH connection:



```
stealth@liane:~> /usr/bin/ssh -2 root@192.168.0.2
The authenticity of host '192.168.0.2 (192.168.0.2)' can't be established.
DSA key fingerprint is c9:8f:fb:79:7f:be:22:0f:2e:6d:cd:f0:ec:11:92:73.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.2' (DSA) to the list of known hosts.
root@192.168.0.2's password:
Welcome to lucifer.
Last login: Sun Jun 16 10:14:29 2002 from 192.168.0.2
Have a lot of fun...
lucifer:~ # id
uid=0(root) gid=0(root) groups=0(root),1(bin),14(uucp),15(shadow),16(dialout),17
(audio),504(cvs),65533(nobody),65534(nogroup)
lucifer:~ #
lucifer:~ #
lucifer:~ # echo $TERM
dumb
lucifer:~ # █
```

Figure 1: User as he sees his hijacked SSH session.

A terminal window titled "ssharp-192.168.0.2.2740" showing a hijacked SSH session. The session starts with a welcome message and login information for user 'lucifer'. The user runs the 'id' command, revealing root access. Subsequent commands show the terminal type is 'dumb'.

```
ssharp-192.168.0.2.2740
Welcome to lucifer.
Last login: Sun Jun 16 10:14:29 2002 from 192.168.0.2
Have a lot of fun...
lucifer:~ # id
uid=0(root) gid=0(root) groups=0(root),1(bin),14(uucp),15(shadow),16(dialout),17
(audio),504(cvs),65533(nobody),65534(nogroup)
lucifer:~ #
lucifer:~ #
lucifer:~ # echo $TERM
dumb
lucifer:~ # █
```

Figure 2: The terminal of the attacker with hijacked SSH session inside

Acknowledgments

Folks from the segfault dot net consortium ;-) for discussing and offering test environments. If you like to donut some hardware or money to these folks let me know. It would definitely help to let continue research on this and similar topics.

Also thanks to various other folks for discussing SSH with me.

This paper is also available in ASCII format in the incredible phrack magazine. Phrack is at <http://www.phrack.org>.

References

[dsniff] Dug Song *As far as I know the first SSH1 MiM implementation "monkey in the middle" part of dsniff package.*

<http://www.monkey.org/~dugsong/dsniff>

[ettercap] *Good sniffer/mim combo program for lazy hackers ;-).*

<http://ettercap.sourceforge.net>

[7350ssharp] TESO *An implementation of the attacks described in this paper.*

<http://stealth.7350.org/7350ssharp.tgz>