

# SSH features

*Things waiting to happen when you run a ssh server*

# structure

- speech  
30 minutes
- demonstration  
20 minutes
- discussion  
20+ minutes

# Things that can happen

- MiM attacks (main topic)
- authentication token guessing/stealing  
password bruteforcing,  
RSA key theft
- timing attacks
- protocol attacks

# Things less interesting

- buffer overflows
- etc.

# Man in the Middle (MiM)

- common attack against public key systems
- attacker spoofs public key of communicating parties
- with SSH this means: sending fake hostkeys to client and negotiating a known session key to sniff communication
- easy with LANs and WLANs

# Hostkeys?

- integral part of SSHv1 and SSHv2
- used with SSHv1 to encrypt secret session key
- used with SSHv2 to sign the session key negotiation (DH)
- transferred in both cases at beginning of conversation
- accepting a wrong hostkeys is like using telnet

# Wrong Hostkey? MiM!

- for each host connected to, the hostkey is saved
- upon each connect, ssh client tries to find corresponding hostkey

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@      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.
```

# The known hosts problem I

- ssh client records host-keys of hosts already connected to
- there are 3 different key-types (OpenSSH)
  - ssh-rsa1 (SSHv1)
  - ssh-rsa (SSHv2)
  - ssh-dss (SSHv2)
- for host to be known, the host AND keytype have to match
- => lets us produce key misses via unknown keytypes!



# The known hosts problem II

```
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)?
```

# The known hosts problem III

```
RSA key fingerprint c1:12:f4:27:5f:ef:21:89:c0:33:fa:1a:57:20:e8:5f.  
The authenticity of host '192.168.0.2 (192.168.0.2)' can't be established  
but keys of different type are already known for this host.  
DSA key fingerprint is 9f:d3:fc:99:64:b0:93:a2:81:66:55:93:7d:ea:ed:e8.  
Are you sure you want to continue connecting (yes/no)?
```

# hostkey selection in SSHv1

- SSH server banner shows client which SSH versions are supported
- SSH-1.99-OpenSSH\_2.2.0p1
- there is only one key-type: RSA
- if SSH client used to have SSHv1 key in known\_hosts file, prompt with a SSHv2 only banner
- this is the case for SSHv1 only servers

# hostkey selection in SSHv2

- client and server send list of supported algorithms
- common: ssh-rsa and ssh-dss
- algorithm is selected as follows (RFC):  
"The first algorithm on the client list that satisfies the requirements and is also supported by the server MUST be chosen."
- Client: rsa, dss Server: dss => choose dss

# Client > Server Algorithm negotiation

The screenshot shows a Wireshark capture of an SSHv2 Key Exchange packet. The packet details pane is expanded to show the 'Keys' section, which lists various algorithms for negotiation. A red circle highlights the 'server\_host\_key\_algorithms' field, which contains 'ssh-rsa,ssh-dss'.

No.	Time	Source	Destination	Protocol	Info
2	0.000192	192.0.0.4	192.0.0.7	TCP	ssh > 1038 [SYN, ACK] Seq=1862825
3	0.001129	192.0.0.7	192.0.0.4	TCP	1038 > ssh [ACK] Seq=2399661709 Ack
4	0.019113	192.0.0.4	192.0.0.7	SSH	Server Protocol: SSH-2.0-OpenSSH_2.
5	0.020137	192.0.0.7	192.0.0.4	TCP	1038 > ssh [ACK] Seq=2399661709 Ack
6	0.023297	192.0.0.7	192.0.0.4	SSH	Client Protocol: SSH-2.0-OpenSSH_3.
7	0.023375	192.0.0.4	192.0.0.7	TCP	ssh > 1038 [ACK] Seq=1862848 Ack=23
8	0.034227	192.0.0.7	192.0.0.4	SSHv2	Client: Key Exchange

Transmission Control Protocol, Src Port: 1038 (1038), Dst Port: ssh (22), Seq: 2399661748, Ack: 186284

SSH Protocol

- SSH Version 2
  - Packet Length: 532
  - Padding Length: 11
  - Key Exchange
    - Msg code: Key Exchange (20)
    - Keys
      - Cookie: 0\225\200P\004\202GNWE7i\237]\027
      - kex\_algorithms length: 61
      - kex\_algorithms string: diffie-hellman-group-exchange-sha1,diffie-hellman-group1-sha1
      - server\_host\_key\_algorithms length: 15
      - server\_host\_key\_algorithms string: ssh-rsa,ssh-dss
      - encryption\_algorithms\_client\_to\_server length: 102
      - encryption\_algorithms\_client\_to\_server string: aes128-cbc,3des-cbc,blowfish-cbc,cast128-c
      - encryption\_algorithms\_server\_to\_client length: 102
      - encryption\_algorithms\_server\_to\_client string: aes128-cbc,3des-cbc,blowfish-cbc,cast128-c
      - mac\_algorithms\_client\_to\_server length: 85

# Server > Client Algorithm negotiation

The screenshot displays a network capture in Wireshark. The main packet list shows a sequence of packets related to an SSH connection. Packet 12 is highlighted, showing a 'Server: Key Exchange' message. The details pane for this packet is expanded to show the 'Key Exchange' section, which includes a list of supported algorithms. A red circle highlights the 'server\_host\_key\_algorithms' field, which contains the string 'ssh-rsa,ssh-dss'. The packet bytes pane at the bottom shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Info
11	0.055452	192.0.0.4	192.0.0.1	TCP	ssh > 33086 [ACK] Seq=3911334278
12	0.061356	192.0.0.4	192.0.0.1	SSHv2	Server: Key Exchange
13	0.069808	192.0.0.1	192.0.0.4	SSHv2	Client: Diffie-Hellman GEX Request
14	0.105740	192.0.0.4	192.0.0.1	TCP	ssh > 33086 [ACK] Seq=3911334822
15	0.143926	192.0.0.4	192.0.0.1	SSHv2	Server: Key Reply

Key Exchange  
Msg code: Key Exchange (20)  
Keys  
Cookie: a\232\ri\0PG\027±\020.f±\201\031  
kex\_algorithms length: 61  
kex\_algorithms string: diffie-hellman-group-exchange-sha1,diffie-hellman-group1-sha1  
server\_host\_key\_algorithms length: 15  
server\_host\_key\_algorithms string: ssh-rsa,ssh-dss  
encryption\_algorithms\_client\_to\_server length: 102  
encryption\_algorithms\_client\_to\_server string: aes128-cbc,3des-cbc,blowfish-cbc,cast128-  
encryption\_algorithms\_server\_to\_client length: 102  
encryption\_algorithms\_server\_to\_client string: aes128-cbc,3des-cbc,blowfish-cbc,cast128-  
mac\_algorithms\_client\_to\_server length: 85  
mac\_algorithms\_client\_to\_server string: hmac-md5,hmac-sha1,hmac-ripemd160,hmac-ripemd160

0000 00 50 22 88 29 93 00 50 22 88 29 9b 08 00 45 00 .P"..P"..)...E.  
0010 02 54 70 13 40 00 40 06 48 8b c0 00 00 04 c0 00 .Tp.@.@.H.A...A.  
0020 00 01 00 16 81 3e e9 22 39 86 2f b5 6c 59 80 18 .....>é" 9./µlY..  
0030 19 80 25 54 00 00 01 01 08 0a 00 00 49 85 00 21 ..%T.... ....I..!  
0040 7f 86 00 00 02 1c 09 14 61 9a 0d ec a6 f2 de 47 ..... a..il0PG



# MiM > Client Algorithm negotiation

The screenshot displays a network capture in Wireshark. The packet list pane shows a series of packets related to an SSH connection. Packet 10 is highlighted, showing a Key Exchange message from the client (192.0.0.4) to the server (192.0.0.7). The packet details pane for this packet is expanded, showing the following fields:

- Transmission Control Protocol, Src Port: ssh (22), Dst Port: 1038 (1038), Seq: 1862848, Ack: 239966228
- SSH Protocol
  - SSH Version 2
    - Packet Length: 628
    - Padding Length: 9
  - Key Exchange
    - Msg code: Key Exchange (20)
    - Keys
      - Cookie: "Ä-9VÜ³\röÉ9d&\2060\_
      - kex\_algorithms length: 61
      - kex\_algorithms string: diffie-hellman-group-exchange-sha1,diffie-hellman-group1-sha1
      - server\_host\_key\_algorithms length: 7
      - server\_host\_key\_algorithms string: ssh-dss
      - encryption\_algorithms\_client\_to\_server length: 150
      - encryption\_algorithms\_client\_to\_server string: aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc
      - encryption\_algorithms\_server\_to\_client length: 150
      - encryption\_algorithms\_server\_to\_client string: aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc
      - mac\_algorithms\_client\_to\_server length: 85

A red circle is drawn around the 'server\_host\_key\_algorithms string' field, highlighting the value 'ssh-dss'.

# hostkey choosing I

- need to produce key miss via unknown key type
- how do we know which algorithm to choose?
- connect to real server upon each MiM connection
- look for supported hostkeys (pre-connect)
- peek at client stream
- look for supported hostkeys (peeking)
- choose the right one



# hostkey choosing II

client	server	RFC	MiM
rsa	rsa,dss	rsa	no way
dss	rsa,dss	dss	no way
rsa,dss	rsa	rsa	dss via pre-connect
rsa,dss	dss	dss	rsa via pre-connect
rsa,dss	rsa,dss	rsa	dss via peeking
dss,rsa	dss,rsa	dss	rsa via peeking
dss,rsa	rsa,dss	dss	rsa via peeking

# Implementation

- reuse existing OpenSSH code
- patch server to accept any login/password and to start special shell *ssharpclient* on a pty
- *ssharpclient* logs into remote-host => yields shell in pty
- optionally: slip *ssharpclient* through screen-like program => hunt for SSH

# Attackers view



```
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:~ # █
```

# How to protect?

- *unknown hostkey* messages should make you scared
- use RSA authentication
- do not use SSH1

# References

- SSH Timing attacks  
<http://www.openwall.com/advisories/>  
<http://www.openwall.com/presentations/>
- ettercap SSH sniffer  
<http://ettercap.sourceforge.net>
- SSHarp  
<http://stealth.openwall.net/SSH>