Secure Shell: SSH

Secure Shell: SSH

- Let's move up the stack and look at ssh
- Partly a tool, partly an application
- We'll discuss the original version of the protocol

Features of SSH

Secure Shell: SSH
Secure Shell: SSH
Features of SSH
Simple Login Sequence
The Server's Two Keys
Authenticating the Server
Sample Initial Login
An Attack?
What is the Security Guarantee?
What Should Users Do?
A List of Ciphers
Client Authentication
Connection-Forwarding
Deployability
Limitations

- Encrypted login and shell connection
- Easy, drop-in replacement for `rlogin`, `rsh`, `rcp`
- Multiple means of authentication
- Interesting case study in deployability

Simple Login Sequence

- Client contacts server
- Server sends its public RSA "host" key (at least 1024 bits), an RSA "server" key (768 bits), and a list of ciphers
- (The server key is changed hourly)
- The client authenticates the server
- The client generates a session key and encrypts it using both the host and server key
- The server decrypts it and uses it for traffic encryption
- The client authenticates to the host

The Server’s Two Keys

- Why are two keys used?
- The longer key is for authentication: only the genuine host will be able to decrypt it
- The shorter key provides an approximation to perfect forward secrecy: if the host is compromised more than one hour after the session starts, there’s no way for the attacker to recover it and read old sessions
- But why not use Diffie-Hellman? Speed? 768-bit RSA is faster than 1024-bit Diffie-Hellman, and computers were slower then. Actually, it’s because Tatu Ylönen, the author, was an inspired amateur in 1995. . .

Authenticating the Server

- How does the client authenticate the server?
- More precisely, why should it trust the server's key?
- Note well: the server is sending a *key*, not a *certificate* — no one is vouching for the key
- The first time a key is received, the user is prompted about whether or not to accept it
- The result is cached in a "known hosts" file

Sample Initial Login

```
$ ssh foo
The authenticity of host ’foo (192.168.77.222)’ can
RSA key fingerprint is cf:26:92:6c:01:c1:05:c7:51:de
Are you sure you want to continue connecting (yes/no
Warning: Permanently added ’foo (RSA) to the list of
```

An Attack?

Secure Shell: SSH
Secure Shell: SSH
Features of SSH
Simple Login Sequence
The Server's Two Keys
Authenticating the Server
Sample Initial Login
An Attack?
What is the Security Guarantee?
What Should Users Do?
A List of Ciphers
Client Authentication
Connection-Forwarding
Deployability
Limitations

```
$ ssh foo
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    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 RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
f1:68:d8:0d:0a:1b:78:2c:48:3a:aa:1b:4a:8c:cb:ca.
Please contact your system administrator.
Add correct host key in /home/smb/.ssh/known_hosts to get rid of this message.
Offending key in /home/smb/.ssh/known_hosts:86
RSA host key for foo has changed and you have requested strict checking.
Host key verification failed.
```

What is the Security Guarantee?

- We don't *know* that the key is correct
- We do know that the key is *the same as it was last time*
- The vulnerability is on the initial login *only*
- But — users must be taught what to do about that message. . .

What Should Users Do?

Secure Shell: SSH
Secure Shell: SSH
Features of SSH
Simple Login Sequence
The Server's Two Keys
Authenticating the Server
Sample Initial Login
An Attack?
What is the Security Guarantee?
What Should Users Do?
A List of Ciphers
Client Authentication
Connection-Forwarding
Deployability
Limitations

- The system administrator can populate a system-wide known hosts file
- System administrators can publish a digitally-signed list of their hosts' keys (see `http://www.psg.com/ssh-keys.html`
- Users can check a piece of paper or ask each other
- Do people actually do this?
- Note: MITM attacks against ssh have been seen in the wild. . .

A List of Ciphers

- The server transmits a list of ciphers at the start
- The client picks one
- What if an attacker substituted a list containing only weak or cracked ciphers?
- This is known as a *rollback* or *downgrade* attack
- Solution: after starting the encryption, send an authenticated list of the algorithms you originally proposed

Client Authentication

Client Authentication

- How does the client authenticate itself to the host?
- Many possible ways — in fact, *very* many possible ways. . .
- We'll look at just a few

Password Authentication

- Simplest form: ordinary username and password
- The password is protected from eavesdropping
- There is no protection against brute-force password guessing

Password Guessing Attacks on SSH

```
00:01:36 foo sshd: Invalid user duane from 206.231.8
00:01:37 foo sshd: Invalid user murray from 206.231.
00:01:38 foo sshd: Invalid user kovic from 206.231.8
00:01:39 foo sshd: Invalid user mitchell from 206.23
00:01:40 foo sshd: Invalid user nance from 206.231.8
00:01:41 foo sshd: Invalid user liberty from 206.231
00:01:42 foo sshd: Invalid user alan from 206.231.8.
00:01:43 foo sshd: Invalid user wilfe from 206.231.8
00:01:45 foo sshd: Invalid user ruthy from 206.231.8
00:01:46 foo sshd: Invalid user oriana from 206.231.
00:01:47 foo sshd: Invalid user mauzone from 206.231
00:01:48 foo sshd: Invalid user leopold from 206.231
```

Public Key Authentication

- Client has a public/private key pair, and sends the public key to the server
- Server encrypts a 256-bit random number with that key
- Client decrypts it and sends back an MD5 hash of the random number

Trusting the Client's Key

- Again, this is a simple key, not a certificate
- There is a per-client list of *authorized keys*
- If the client's key is in that list, it's accepted (provided, of course, that the challenge/response works)

Host-Based Authentication

- The client's host can have a public/private key pair
- If this host is listed in an authorized hosts file, the userid is simply accepted
- Note: this is only useful if the two machines are under common administration and are secure against insider attacks

Storing Private Keys

- How are private keys stored?
- If a private key is compromised, all security bets are off
- Note: must cope with NFS-mounted home directories

The Minimum

- All private key files must be read-protected
- But if users store their keys under their home directories and use NFS, someone can eavesdrop on the NFS traffic
- Solution: encrypt the private key with some symmetric cipher; prompt the user for a passphrase as needed

Too Many Prompts!

- If people use ssh heavily, they'll be prompted for passwords constantly
- Solution: *ssh agent*
- Run a process that prompts for the passphrase once, decrypts the keys in memory, and performs the public key operations on behalf of the *proper* ssh client
- How do we secure that channel?

Securing the SSH Agent

- All communications to it are via a Unix-domain socket, which lives in the file system
- Not all systems enforce file permissions on Unix-domain sockets, since they're seen as communications channels rather than as files
- But — all systems verify permissions on containing directories
- Put the socket in a protected directory; use shell environment variables to pass the location to clients

Using SSH Agent

Secure Shell: SSH
Client Authentication
Client Authentication
Password Authentication
Password Guessing Attacks on SSH
Public Key Authentication
Trusting the Client's Key
Host-Based Authentication
Storing Private Keys
The Minimum
Too Many Prompts!
Securing the SSH Agent
Using SSH Agent
Connection-Forwarding
Deployability
Limitations

```
$ set|grep SSH
SSH_AGENT_PID=363
SSH_AUTH_SOCK=/tmp/ssh-00000418aa/agent.418
$ ls -la /tmp/ssh-00000418aa
total 8
drwx------  2 smb   wheel   20 Oct 11 03:15 .
drwxrwxrwt  4 root  wheel  260 Oct 12 00:13 ..
srwxr-xr-x  1 smb   wheel    0 Oct 10 20:57 agent.41
```

Connection-Forwarding

Connection-Forwarding

- Ssh can forward TCP connections from the local machine to the remote, or vice-versa
- Can be used to access resources through an ssh firewall
- Talking to an internal POP3 server:

  ```
  ssh -L 110:mbox:110 firewall
  ```

  followed by (in another window)

  ```
  telnet 127.0.0.1 110
  ```

- Or, of course, configure your mailer to talk to 127.0.0.1
- Can forward remote connections to the local machine, too

Secure Shell: SSH
Client Authentication
Connection-Forwarding
Connection-Forwarding
Violating Security Policy with SSH
Forwarding the Authentication Agent
Forwarding the Authentication Agent
The Risks of Agent Forwarding
X11 Forwarding
Authenticating X11 Connections
X11 Forwarding
Cookie Change
The Risks of X11 Forwarding
Deployability
Limitations

Violating Security Policy with SSH

- Policy 1: ssh to the firewall is the only inbound service allowed
- Policy 2: all ssh connections must be authenticated by a SecurID token
- Violation:

  ```
       ssh -L 2222:insidehost:22
  firewall
  ```
- Connects port 2222 on some outside machine to port 22 — ssh — on some inside server
- To log in without using a SecurID token, just connect to 2222 on that outside machine
- Similar violations can be initiated from the inside, if outbound ssh is permitted

Forwarding the Authentication Agent

- Alice use `ssh-agent` to log in to host Foo. From Foo, she logs in to Bar. How does she authenticate?
- She could have a separate private/public key bar stored on Foo, and use it to log in to Bar
- Alternatively, she could use a special form of connection-forwarding to forward access to the authentication agent
- Note: the private key itself is not transmitted; all cryptographic operations are still done by the same agent process

Forwarding the Authentication Agent

```
$ ssh-add -l
1024 7c:01:66:d8:4b:3d:bc:36:1e:97:92:8e:48:d5:0f:37
b132$ ssh berkshire
NetBSD 4.99.3 (BERKSHIRE) #0: Sun Sep 24 16:30:08 ED

b129$ ssh-add -l
1024 7c:01:66:d8:4b:3d:bc:36:1e:97:92:8e:48:d5:0f:37
b130$ set|grep SSH
SSH_AUTH_SOCK=/tmp/ssh-00028833aa/agent.28833
SSH_CLIENT=’192.168.2.79 65051 22’
SSH_CONNECTION=’192.168.2.79 65051 192.168.2.163 22
SSH_TTY=/dev/ttyp4
```

The Risks of Agent Forwarding

- Suppose that host Foo is insecure
- An attacker with root privileges on Foo can contact Alice’s authentication agent
- It is thus possible for the attacker to log in as Alice anywhere that key is accepted
- *Never* do connection-forwarding to an insecure machine

X11 Forwarding

- Ssh can be used to forward X11 window system connections, too
- How X11 works: with X11, the *X server* controls the keyboard, screen, and mouse
- X applications open a connection — via Unix-domain sockets or TCP — to the server
- The environment variable DISPLAY tells the application what to do
- How is this connection authenticated?

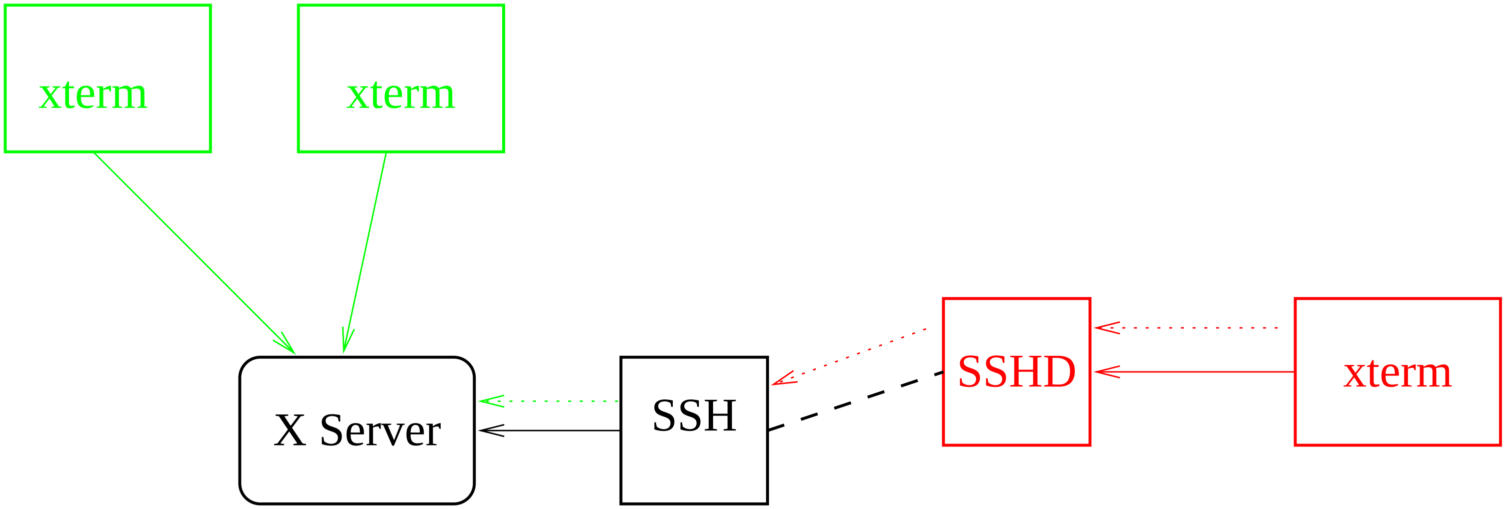
Authenticating X11 Connections

- Some people don't — so attackers can read the screen, and send synthetic keypress and mouse events. Oops. . .
- Can be done with odd Kerberos facilities
- Normal way: use "magic cookie" mode — the application has to read a (secret) value from a file, and send that to the X server

X11 Forwarding

- The remote sshd generates a new, random cookie and stores it in that file for applications
- It sets DISPLAY to point to itself
- When an X11 application attempts to connect to the X server, it actually connects to sshd and sends that magic cookie
- The sshd server verifies the cookie, and forwards the connection over the ssh channel to the client
- The client replaces the remote cookie with the local one, and contacts the local X server

Cookie Change



The Risks of X11 Forwarding

- Again, assume that Foo is insecure and is penetrated
- An attacker can read the cookie, connect to Alice's X server, and read the screen, send events, etc.
- Moral: don't forward X11 to an insecure machine

Deployability

Why Did SSH Succeed?

- No infrastructure needed
- No PKI, no CAs, no central server
- A site could deploy SSH on as many or as few machines as needed

Usability

- It was a drop-in replacement for `rlogin`
- It could even be configured with the same host-based trust model
- It required little in the way of user training
- It provided some nice features, such as connection- and X11-forwarding, compression, etc.

Security

- It defended against real attacks
- It provided extra functionality not in other packages, such as connection-forwarding
- It included add-ons such as scp
- It ran on more Unix variants than its competitors did

Limitations

SSH Doesn’t Solve All Problems

- Cryptographic mistakes (i.e., using a CRC instead of MD5)
- Compromised hosts
- Password-guessing
- Deliberate user misbehavior
- Ssh worms

Compromised Hosts

- The ssh and sshd commands can be Trojaned, and used to steal passwords
- X11 and authentication agent forwarding can be captured by the bad guys

Ssh Worms

- The known host file indicates connectivity patterns
- More importantly, it tends to indicate *trust* patterns
- An attacker who has compromised your machine can not only use your ssh keys, but can also look at the known hosts list to see where you've connected via ssh
- Transitive trust patterns help the attack spread
- (Btw, studies suggest that many users don't encrypt their private keys. . .)

Conclusions

- A professional cryptographer would have designed a system around certificates issued by properly-isolated and secured CAs
- In a very real sense, that would have been more secure — and it would likely have been undeployable
- We got more *real security* from a partially-secure implementation that better matched deployment patterns