50 Shades of Sudo Abuse

- About
- A very brief crash course on Sudo, TTPs, and Sudoers
- Enumeration + Recon
- Example + Explanation
  - 1.1: Open Sudo
  - 1.2: Application Abuse (the easy stuff)
  - 1.3: Application Abuse pt2 (more subtle gotcha's)
  - 1.4: Application Abuse pt3 (leveraging bugs)
  - 2.0: Abusing Pagers
  - 3.0: Cron Abuse
  - 4.0: LD_PRELOAD
  - 5.0: Installers (pip)
  - 6.0: Path in Sudoers (wildcards) and *secure_path* in sudoers
  - 7.0: Abusing editors (vim, nano, sudoedit)
  - 8.0: CVE-2019-18634: Pwdfeedback
  - 9.0: Limited Shells
  - 10.0: CVE-2019-14287: security bypass

About: Me

- Agg (class of 09)
- CommO (MOS: 0602) MACS4 / 1MAW
- Network Engineer
- Security Engineer

Tyler Boykin
@CaptBoykin
Capt.boykin.0602@gmail.com

SUDO : ~#

Crash course Sudo pt 1

- "**sudo** allows a permitted user to execute a *command* as the superuser or another user, as specified by the security policy."

- " The security policy determines what privileges, if any, a user has to run **sudo**. The policy may require that users authenticate themselves with a password or another authentication mechanism. If authentication is required, **sudo** will exit if the user's password is not entered within a configurable time limit. This limit is policy-specific; the default password prompt timeout for the *sudoers* security policy is 5 minutes."
    - Ref: https://www.sudo.ws/man/1.8.13/sudo.man.html

Crash course Sudo pt 2

- "When **sudo** executes a command, the security policy specifies the execution environment for the command. Typically, the real and effective user and group and IDs are set to match those of the target user, as specified in the password database, and the group vector is initialized based on the group database (unless the **-P** option was specified)…"
- "...real and effective user ID, real and effective group ID, supplementary group IDs, the environment list, current working directory, file creation mode mask (umask), SELinux role and type, Solaris project, Solaris privileges, BSD login class, scheduling priority (aka nice value)"
  - Ref: https://www.sudo.ws/man/1.8.13/sudo.man.html

Crash course Sudo pt3

- "There is no easy way to prevent a user from gaining a root shell if that user is allowed to run arbitrary commands via sudo. Also, many programs (such as editors) allow the user to run commands via shell escapes, thus avoiding sudo's checks. However, on most systems it is possible to prevent shell escapes with the sudoers(5) plugin's noexec functionality."
  - Ref: https://www.sudo.ws/man/1.8.13/sudo.man.html

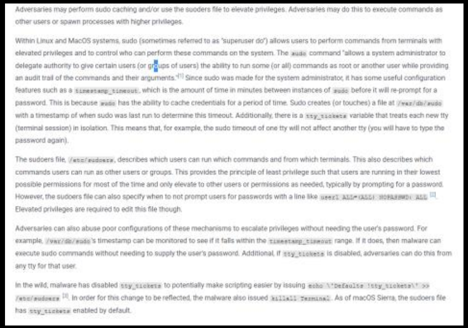
TTPs:~#

T1548: Abuse Elevation Control Mechanism

" Adversaries may circumvent mechanisms designed to control elevate privileges to gain higher-level permissions. Most modern systems contain native elevation control mechanisms that are intended to limit privileges that a user can perform on a machine. Authorization has to be granted to specific users in order to perform tasks that can be considered of higher risk. An adversary can perform several methods to take advantage of built-in control mechanisms in order to escalate privileges on a system."

- Ref: https://attack.mitre.org/techniques/T1548/003/

T1548.003: Abuse Elevation Control Mechanism: Sudo and Sudo Caching



Adversaries may perform sudo caching and/or use the suoders file to elevate privileges. Adversaries may do this to execute commands as other users or spawn processes with higher privileges.

Within Linux and MacOS systems, sudo (sometimes referred to as "superuser do") allows users to perform commands from terminals with elevated privileges and to control who can perform these commands on the system. The `sudo` command "allows a system administrator to delegate authority to give certain users (or groups of users) the ability to run some (or all) commands as root or another user while providing an audit trail of the commands and their arguments."[1] Since sudo was made for the system administrator, it has some useful configuration features such as a `timestamp_timeout`, which is the amount of time in minutes between instances of `sudo` before it will re-prompt for a password. This is because `sudo` has the ability to cache credentials for a period of time. Sudo creates (or touches) a file at `/var/db/sudo` with a timestamp of when sudo was last run to determine this timeout. Additionally, there is a `tty_tickets` variable that treats each new tty (terminal session) in isolation. This means that, for example, the sudo timeout of one tty will not affect another tty (you will have to type the password again).

The sudoers file, `/etc/sudoers`, describes which users can run which commands and from which terminals. This also describes which commands users can run as other users or groups. This provides the principle of least privilege such that users are running in their lowest possible permissions for most of the time and only elevate to other users or permissions as needed, typically by prompting for a password. However, the sudoers file can also specify when to not prompt users for passwords with a line like `user1 ALL=(ALL) NOPASSWD: ALL` [1]. Elevated privileges are required to edit this file though.

Adversaries can also abuse poor configurations of these mechanisms to escalate privileges without needing the user's password. For example, `/var/db/sudo`'s timestamp can be monitored to see if it falls within the `timestamp_timeout` range. If it does, then malware can execute sudo commands without needing to supply the user's password. Additional, if `tty_tickets` is disabled, adversaries can do this from any tty for that user.

In the wild, malware has disabled `tty_tickets` to potentially make scripting easier by issuing `echo \'Defaults !tty_tickets\' >> /etc/sudoers` [2]. In order for this change to be reflected, the malware also issued `killall Terminal`. As of macOS Sierra, the sudoers file has `tty_tickets` enabled by default.

https://attack.mitre.org/techniques/T1548/003/

T1548.003: Abuse Elevation Control Mechanism: Sudo and Sudo Caching

- “Adversaries may perform sudo caching and/or use the suoders file to elevate privileges. Adversaries may do this to execute commands as other users or spawn processes with higher privileges.”
- “Adversaries can also abuse poor configurations of these mechanisms to escalate privileges without needing the user's password. “

SUDOERS:~#

```
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/
instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/s
bin:/usr/bin:/sbin:/bin"


# Host alias specification


# User alias specification


# Cmnd alias specification
```

Super high level Sudoers pt1

- *<u>The</u>* primary configurable file by which sudo permissions are made
- Resides in /etc/sudoers by default
- It *can* be edited with whatever editor by root
- It *should* be edited using something more secure (ie *visudo*)
- It's picky on syntax, errors affects all users.
  - Best to make a copy, perform edits, and then use visudo for syntax checking
  - **`visudo -cf /var/tmp/sudoers.new`**

Super high level Sudoers pt2

- Error in syntax

```
>>> /etc/sudoers.myedits: syntax error near line 15 <<<
parse error in /etc/sudoers.myedits near line 15
```

- OK

```
/etc/sudoers.myedits: parsed OK
/etc/sudoers.d/README: parsed OK
```

Sudoers pt1

- "The *sudoers* security policy requires that most users authenticate themselves before they can use **sudo**. A password is not required if the invoking user is root, if the target user is the same as the invoking user, or if the policy has disabled authentication for the user or command. "
- "Unlike su(1), when *sudoers* requires authentication, it validates the invoking user's credentials, not the target user's (or root's) credentials. This can be changed via the *rootpw*, *targetpw* and *runaspw* flags, described later."
  - Ref: https://www.sudo.ws/man/1.8.15/sudoers.man.html

Sudoers pt2

- "The sudoers file is composed of two types of entries: aliases (basically variables) and user specifications (which specify who may run what). "
- "When multiple entries match for a user, they are applied in order. Where there are multiple matches, the last match is used (which is not necessarily the most specific match)."
  - Ref: https://www.sudo.ws/man/1.8.15/sudoers.man.html

Sudoers pt3

Aliases: There are four kinds of aliases: User_Alias, Runas_Alias, Host_Alias and Cmnd_Alias.

```
Alias_Type NAME = item1, item2, item3 : NAME = item4, item5
```

- Defaults: Certain configuration options may be changed from their default values at run-time via one or more Default_Entry lines. These may affect all users on any host, all users on a specific host, a specific user, a specific command, or commands being run as a specific user.
  - Ref: https://www.sudo.ws/man/1.8.15/sudoers.man.html

Sudoers pt4

- sudo allows shell-style wildcards (aka meta or glob characters) to be used in host names, path names and command line arguments in the sudoers file. Wildcard matching is done via the glob(3) and fnmatch(3) functions as specified by IEEE Std 1003.1 ("POSIX.1").

***** **Matches any set of zero or more characters (including white space).**

? **Matches any single character (including white space).**

[...] **Matches any character in the specified range.**

[!...] **Matches any character not in the specified range.**

\x **For any character 'x', evaluates to 'x'. This is used to escape special characters such as: '*', '?', '[', and ']'.**

- **Note that these are not regular expressions.**
- Unlike a regular expression there is no way to match one or more characters within a range.
  - Ref: https://www.sudo.ws/man/1.8.15/sudoers.man.html

Sudoers pt5

- User specification: A user specification determines which commands a user may run (and as what user) on specified hosts. By default, commands are run as root, but this can be changed on a per-command basis. The basic structure of a user specification is

```
who where = (as_whom) what
```

- Runas_Spec: A Runas_Spec determines the user and/or the group that a command may be run as. A fully-specified Runas_Spec consists of two Runas_Lists (as defined above) separated by a colon (‘:’) and enclosed in a set of parentheses. A Runas_Spec sets the default for the commands that follow it. What this means is that for the entry:

```
Me Server = (operator) /bin/ls, /bin/kill, /usr/bin/lprm
```

- Ref: https://www.sudo.ws/man/1.8.15/sudoers.man.html

Sudoers pt6

- Tag_Spec: A command may have zero or more tags associated with it. There are ten possible tag values: EXEC, NOEXEC, FOLLOW, NOFOLLOW, LOG_INPUT, NOLOG_INPUT, LOG_OUTPUT, NOLOG_OUTPUT, MAIL, NOMAIL, PASSWD, NOPASSWD, SETENV, and NOSETENV. Once a tag is set on a Cmnd, subsequent Cmnds in the Cmnd_Spec_List, inherit the tag unless it is overridden by the opposite tag (in other words, PASSWD overrides NOPASSWD and NOEXEC overrides EXEC).
  - **NOPASSWD** often used
  - **SETENV** preserves original user's env stuff

    NOEXEC prevents dynamically linked programs from running other commands itself

  - Ref: https://www.sudo.ws/man/1.8.15/sudoers.man.html

ENUMERATION:~#

Manually

- sudo –l (might be as easy as that)
- /etc/sudoers (long shot, might be readable)
- /usr/bin/sudo -V (version) + google
- /etc/group (find other potential users in sudo group)
- /var/log/auth.log

Automated

- MSF
  - post/multi/recon/sudo_commands
- Scripts
  - https://github.com/rebootuser/LinEnum
  - https://github.com/n0w4n/CVE-2019-14287
  - https://github.com/Critical-Start/FallofSudo

EXAMPLES:~#

"

The Problem with Sudo

With basic knowledge of Sudo and /etc/sudoers the hunt for vulnerabilities is on. During engagements, I consistently see 2 scenarios as it relates to Sudo rules.

- A complete lack of configuration where the Sudo rules don’t follow the least privilege model and allow for many users to execute commands as root.
- A relatively secure set of Sudo rules, but the command assigned is a rarely used argument that allows the execution of shell commands.

Ref: https://www.criticalstart.com/fall-of-sudo-a-pwnage-collection/#:~:text=Finding%20Linux%20servers%20heavily%20reliant,quickly%20become%20security's%20worst%20nightmare.

Fairly secure settings, but unforeseen bugs or obscure options

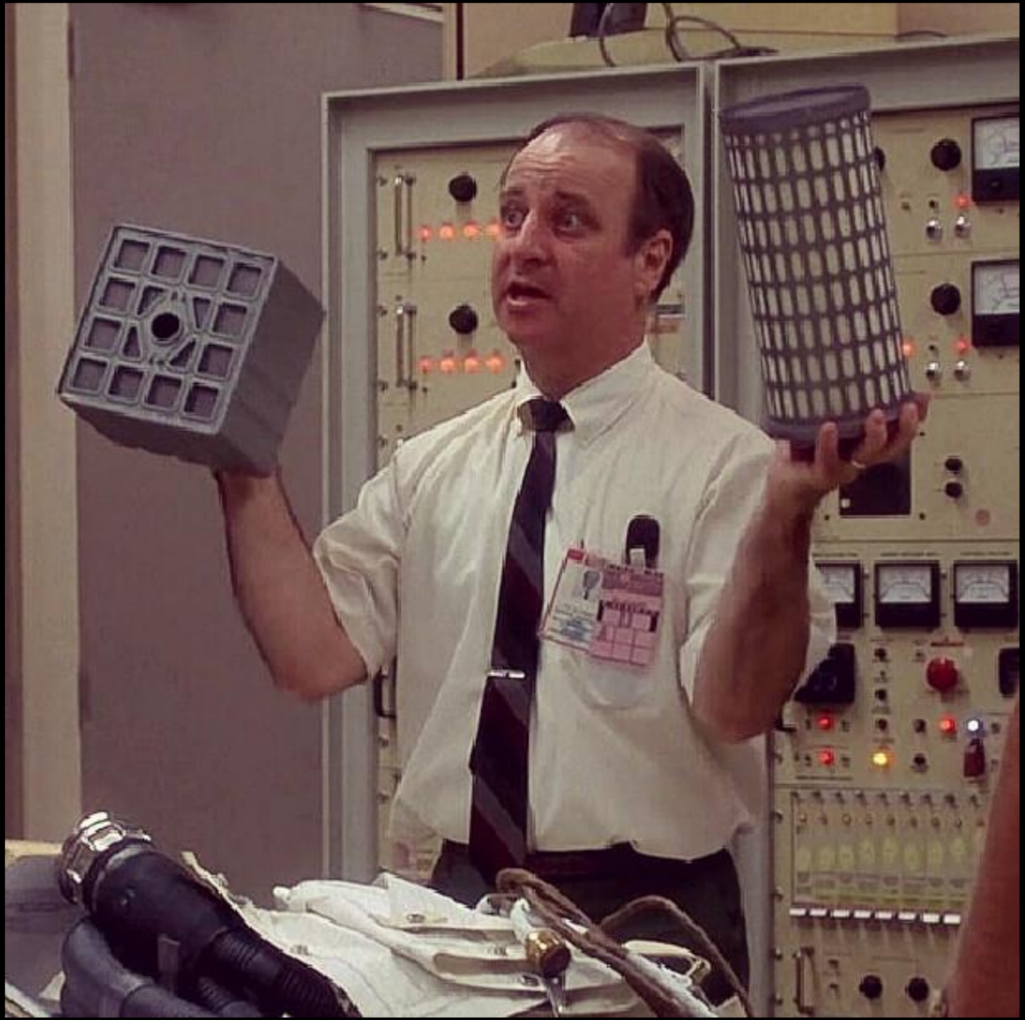
Middle area that's a combination of inconsistently applied rules and too lax of rules

Lowest of the fruit

Overall avenues of adventures

Get creative and inventive

- "How can I do something not permitted?"
- "How can I read something off limits?"
- "How can I write something off limits?"
- "Are there any features that are often overlooked?"
- "Can I escalate laterally?"
- "Can I escalate vertically?"



https://live.staticflickr.com/5485/9049864455_1c081998ac_z.jpg

Entering
Low Hanging Fruit

1 Low hanging fruit: Open Sudo

- "Wide open". Default for root. Can run anything on the system as root as long as they can authenticate (even worse if NOPASSWD is used)

```
# User privilege specification
root    ALL=(ALL:ALL) ALL
section1_opensudo ALL=(ALL:ALL) ALL
```

1 Low hanging fruit: App Abuse pt1

- Anything that directly spawns shells, (sh, ksh, zsh, dash, bash, etc)
- Any programming language (python, ruby, php, perl, java, etc etc)

```
# User privilege specification
section1_apps ALL=(root) /bin/su
section1_apps ALL=(root) /bin/sh
section1_apps ALL=(root) /usr/bin/python
section1_apps ALL=(root) /usr/bin/ruby
```

1 Low hanging fruit: App Abuse pt2

- Anything with features that were intended for a benevolent purpose but can be twisted
- Anything with features that function as intended but have side effects that can benefit the attacker

```
# User privilege specification
section1_apps2 ALL=(root) /usr/bin/wget
Section1_apps2 ALL=(root) /usr/bin/curl
section1_apps2 ALL=(root) /usr/bin/base64
section1_apps2 ALL=(root) /usr/bin/tree
section1_apps2 ALL=(root) /usr/bin/gcc
```

1 Low hanging fruit: App Abuse pt2 cont.

`/usr/bin/wget` **–File Read/ File Write (wget –i /root/root.txt)**

```
 -i,  --input-file=FILE   download URLs found in local or external FILE
```

`/usr/bin/curl` **–File Read/ File Write (curl -O -C - file:///root/root.txt)**

```
 -C, --continue-at <offset>
         Continue/Resume  a  previous  file transfer at the given offset.
         Use  "-C  -" to tell curl to automatically find out where/how to
         resume the transfer
   -O, --remote-name
         Write  output to a local file named like the remote file we get.
```

`/usr/bin/base64` **–File Read/ File Write (base64 /root/root.txt | base64 –d)**

`/usr/bin/gcc` **–Commands (gcc -wrapper /bin/sh,-s .)**

```
-wrapper  Invoke all subcommands under a wrapper program.The name of
  the wrapper program and its parameters are passed as a comma
  separated list.
```

1 Low hanging fruit: App Abuse pt3 (leveraging bugs)

- Applications with bugs running sudo can be leveraged.

```
# User privilege specification
section1_apps3 ALL=(root)
/home/section10_apps3/mycat
```

- In the example above, a command injection vulnerable can be leveraged to run arbitrary commands (in addition to other avenues)

Entering Pager Abuse

DEMO

2. Abusing Pagers

- "A terminal pager, or paging program, is a computer program used to view (but not modify) the contents of a text file moving down the file one line or one screen at a time. Some, but not all, pagers allow movement up a file. A popular cross-platform terminal pager is more. More can move forwards and backwards in text files but cannot move backwards in pipes.[1] less is a more advanced pager that allows movement forward and backward, and contains extra functions such as search.[2]"
  - **Examples: more, less, man, systemctl, service**

```
# User privilege specification
section2_pagers ALL=(root) /usr/bin/man
section2_pagers ALL=(root) /usr/sbin/service
```

Ref: https://en.wikipedia.org/wiki/Terminal_pager

2. Abusing Pagers

- Editors also incorporate a Terminal Pager (vim, nano, etc)

- Basically anything that lets you use your arrow keys to 'scroll' the content of the file left, right, up, down could be a pager.

DEMO

Entering Cron Abuse

3. Cron Abuse

- Services that run as root are an avenue of approach for someone looking to escalate
- If the user can manipulate the service itself either through crontab or directly affecting the underlying applications

```
# User privilege specification
section3_cron ALL=(root) /usr/sbin/service cron *
section3_cron ALL=(root) /usr/bin/crontab
```

DEMO

Entering LD_PRELOAD

4. LD_PRELOAD pt1

- "The LD_PRELOAD trick exploits functionality provided by the dynamic linker on Unix systems that allows you to tell the linker to bind symbols provided by a certain shared library *before other libraries*. For this, remember that upon program execution, the operating system's ***dynamic loader*** will first load dynamic libraries you link to into the process's memory (address space), such that the *dynamic linker* can then resolve symbols at load or run time and bind them to actual definitions."
    - **Ref:** http://www.goldsborough.me/c/low-level/kernel/2016/08/29/16-48-53-the_-ld_preload-_trick/

4. LD_PRELOAD pt2

- TL;DR or still unfamiliar with dynamic linking
  - Methods and functions that are called by the executable can be overwritten with **custom functionality.**

    "**custom functionality**" **+** "**sudo**" === "**fun**“

- Sudoers file require a particular default in place for this to be utilized.
  - `Defaults:section4_ldpreload        env_keep+=LD_PRELOAD`

- Syntax for the above
  - *`Sudo LD_PRELOAD=/path/to/my/shared_obj <binary>`*

DEMO

Entering Installers

5. Abusing Installers (pip, apt-get, etc)

- Installers can also be manipulated to run extra commands or install attacker-controlled software
- Installing software at any level is risky... let alone sudo

```
# User privilege specification
section5_installers ALL=(root) /usr/bin/pip
section5_installers All=(root) /usr/bin/apt-get
Section5_installers All=(root) /usr/bin/dpkg
```

Ref: https://kulinacs.com/pip-install-is-code-execution/

Ref: https://root4loot.com/post/pip-install-privilege-escalation/

5. Abusing Installers (pip, apt-get, etc)

- Apt-get:
  - Pager
  - During install
  - During update
- Pip
  - Setup.py has various avenues to put in custom functionality or even backdoor stuff
- dpkg
  - Pager
  - During install with a custom .deb

DEMO

Entering $PATH and secure_path

6. Misconfigured executable path in Sudoers (wildcards) pt1

- Recall the section on wildcards...

***** Matches any set of zero or more characters (including white space).

? Matches any single character (including white space).

[...] Matches any character in the specified range.

[!...] Matches any character not in the specified range.

\x For any character ‘x’, evaluates to ‘x’. This is used to escape special characters such as: ‘*’, ‘?’, ‘[’, and ‘]’.

6. Misconfigured executable path in Sudoers (wildcards) pt2

```
# User privilege specification
section8_path ALL=(root) /bin/*
```

```
Has a similar effect as a '.' or '*' in $PATH.
```

6. Misconfigured ***secure_path*** in sudoers

"secure_path: Path used for every command run from sudo. If you don't trust the people running sudo to have a sane PATH environment variable you may want to use this. Another use is if you want to have the “root path” be separate from the “user path”. Users in the group specified by the exempt_group option are not affected by secure_path. This option is not set by default. "

- If the secure_path is set and misconfigured === potential for fun :)

- **Command path beats secure path**

```
Defaults:section6_path2
secure_path=/tmp:/usr/local/sbin:/usr/local/bin:/usr
/sbin:/usr/bin:/sbin:/bin
```

DEMO

Entering editors and sudoedit

7. Abusing editors(vim, nano, sudoedit) pt1

- An obvious avenue for exploitation, could result in sensitive file read/write and up to elevated shells
- Some editors also incorporate pagers.

```
# User privilege specification
(ALL : ALL) NOPASSWD: /home/section7_editors/bin/sudoedit
  /home/section7_editors/*/*/protected.txt
(ALL : ALL) NOEXEC: NOPASSWD:
  /usr/bin/vim /home/section7_editors/YOUCANONLYEDITTHISFILE
  .txt
(ALL : ALL) NOPASSWD: /usr/bin/nano
```

Ref: https://root4loot.com/post/pip-install-privilege-escalation/

7. Abusing editors(vim, nano, sudoedit) pt2

- Sudo 1.8.14 (RHEL 5/6/7 / Ubuntu) - 'Sudoedit' Unauthorized Privilege Escalation

"It seems that sudoedit does not check the full path if a wildcard is used twice (e.g. /home/*/*/file.txt), allowing a malicious user to replace the file.txt real file with a symbolic link to a different location (e.g. /etc/shadow)"

```
# User privilege specification
section9_editors
ALL=(ALL:ALL) /home/section9_editors/bin/sudoedit /h
ome/section9_editors/*/*/protected.txt
```

Ref: https://www.exploit-db.com/exploits/37710

DEMO

Entering pwfeedback (CVE-2019-18634)

8. CVE-2019-18634: Pwdfeedback

" In Sudo before 1.8.26, if pwfeedback is enabled in /etc/sudoers, users can trigger a stack-based buffer overflow in the privileged sudo process. (pwfeedback is a default setting in Linux Mint and elementary OS; however, it is NOT the default for upstream and many other packages, and would exist only if enabled by an administrator.) The attacker needs to deliver a long string to the stdin of getln() in tgetpass.c."

TL;DR: A pretty simple buffer overflow with ascii commands at the end :)

```
# User privilege specification
section11_pwfeedback ALL=(ALL:ALL) ALL
```

Ref: https://nvd.nist.gov/vuln/detail/CVE-2019-18634

DEMO

Entering Limited Shells

9. Overcoming Limited Shells pt1

- In some instances, the attacker/tester has a limited shell and for whatever reason cannot elevate to a full TTY.
- If any kind of password can be provided, *expect* can be leveraged to run commands as a sudo/su user.

- In this demo, a limited shell complicates leveraging an open sudo as authenticating requires a TTY. Expect provides another option.

```
expect -c 'spawn sudo -S cat "/root/root.txt";expect
"*password*";send "section12_limited_shell";send
"\r\n";interact'
```

Ref: https://likegeeks.com/expect-command/

9. Overcoming Limited Shells pt2

"Commands are listed alphabetically so that they can be quickly located. However, new users may find it easier to start by reading the descriptions of **spawn**, **send**, **expect**, and **interact**, in that order."

"The spawn command is used to start a script or a program like the shell, **FTP**, Telnet, SSH, SCP, and so on.

The send command is used to send a reply to a script or a program.

The Expect command waits for input.

The interact command allows you to define a predefined user interaction"

Ref: https://likegeeks.com/expect-command/

Ref: https://linux.die.net/man/1/expect

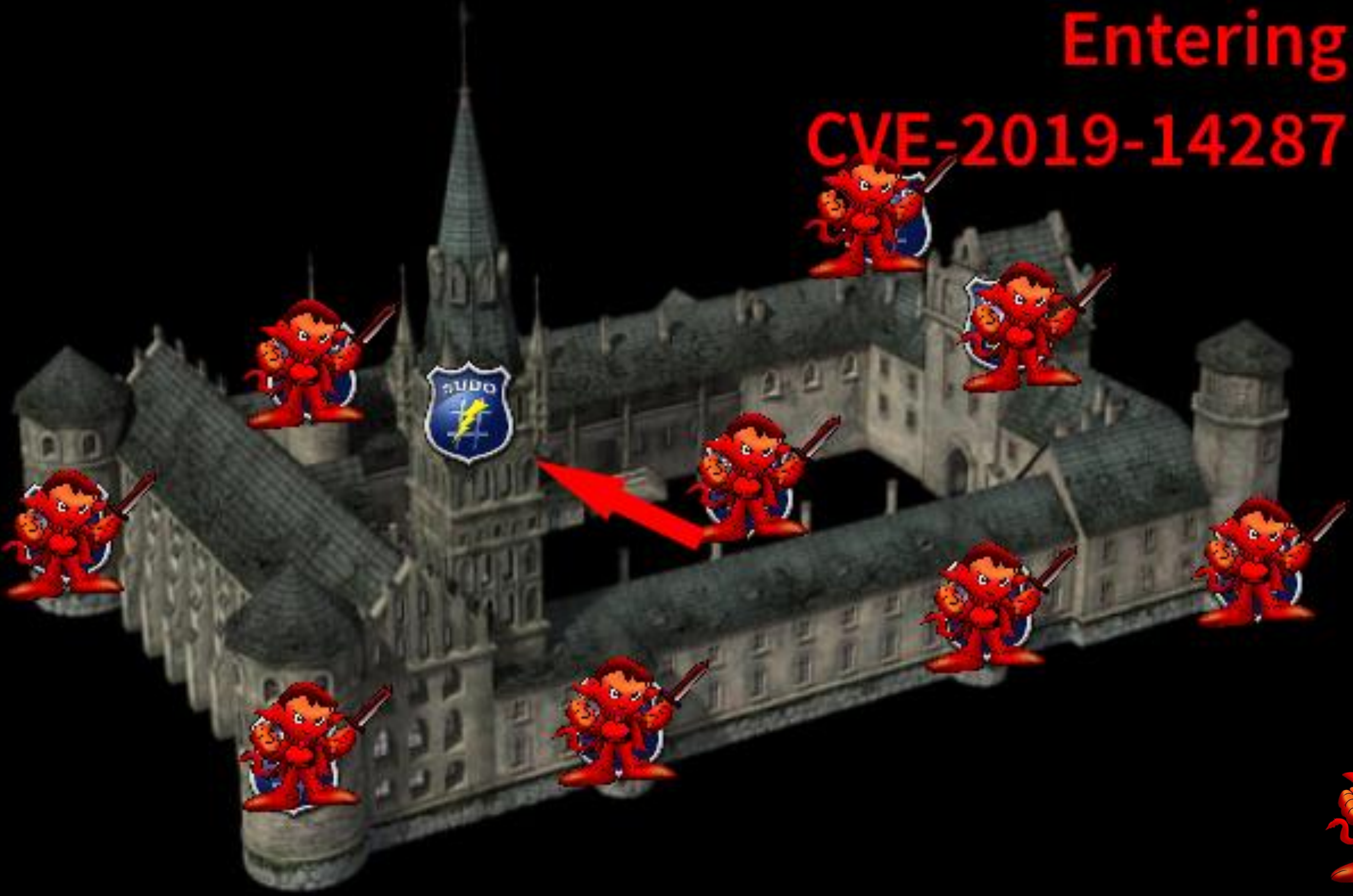
9. Overcoming Limited Shells pt3

```
expect -c 'spawn sudo -S cat "/root/root.txt";expect
"*password*";send "section12_limited_shell";send
"\r\n";interact'
```

- `expect –c`
  - A command to be executed before any in the script...
- `spawn sudo -S cat "/root/root.txt";`
  - 'Spawn' sudo and supply the command "cat...."
- `expect "*password*";`
  - 'Expect' many permuations of password...
- `send "section9_limited_shell";`
  - 'Send' the supplied password to STDOUT...
- `send "\r\n";`
  - 'Send' \r\n (enter)...
- `Interact`
  - 'Interact' thereby outputting what expect see's from the terminal

DEMO

Entering CVE-2019-14287



10. CVE-2019-14287: security bypass

"Description :Sudo doesn't check for the existence of the specified user id and executes the with arbitrary user id with the sudo priv -u#-1 returns as 0 which is root's id and /bin/bash is executed with root permission"

```
# User privilege specification
section14_2019-14287 ALL=(ALL,!root) /bin/sh
```

sudo –u#-1 /bin/sh

Ref: https://www.exploit-db.com/exploits/47502

DEMO

Recap

- A very brief crash course on Sudo and Sudoers
  - Sudo
  - Sudoers File
- Example + Explanation
  - 1.1. Open Sudo
  - 1.2. Application Abuse (the easy stuff)
  - 1.3. Application Abuse pt2 (more subtle gotcha's)
  - 2.0. Abusing Pagers
  - 3.0. Abusing Cron
  - 4.0. LD_PRELOAD
  - 5.0. Abusing Installers (pip, apt-get)
  - 6.0. Misconfigured executable path in Sudoers (wildcards)
  - 7.0. Editors (vim, nano, sudoedit)
  - 8.0. Application Abuse pt3 (leveraging bugs)
  - 9.0. CVE-2019-18634: Pwdfeedback
  - 10.0. Overcoming Limited Shells
  - 11.0. Misconfigured ***secure_path*** in sudoers
  - 12.0. CVE-2019-14287: security bypass

References, Links, Goodies

- Sudo logo
  - https://commons.wikimedia.org/wiki/File:Sudo_logo.png
- Sudo 1.8.14 (RHEL 5/6/7 / Ubuntu) - 'Sudoedit' Unauthorized Privilege Escalation
  - https://www.exploit-db.com/exploits/37710
- Sudo 1.8.25p - 'pwfeedback' Buffer Overflow (PoC)
  - https://www.exploit-db.com/exploits/47995
- Sudo Manual
  - https://www.sudo.ws/man/1.8.15/sudoers.man.html
- Sudo Manual – Sudoers File
  - https://www.sudo.ws/man/1.8.15/sudoers.man.html
- GTFO Bins
  - https://gtfobins.github.io/
- Python Pip Priv Esc via Sudo
  - https://root4loot.com/post/pip-install-privilege-escalation/

- Sudo (LD_PRELOAD) (Linux Privilege Escalation)
  - Touhid Shaikh, April 12, 2018: https://touhidshaikh.com/blog/2018/04/sudo-ld_preload-linux-privilege-escalation/
- sudo 1.8.27 - Security Bypass
  - https://www.exploit-db.com/exploits/47502
- Abusing SUDO
  - https://recipeforroot.com/abusing-sudo/
- Terminal Pager
  - https://en.wikipedia.org/wiki/Terminal_pager
- Expect Command
  - https://likegeeks.com/expect-command/
  - https://linux.die.net/man/1/expect
  - Pentester Academy AttackDefense Labs
- GCC –wrapper
  - https://stackoverflow.com/questions/14039669/what-does-gccs-wrapper-flag-do

- LinEnum
  - https://github.com/rebootuser/LinEnum
- " Fall of Sudo – A Pwnage Collection" - Quentin Rhoads-Herrera | Offensive Security Manager, CRITICALSTART, May 21, 2018
  - https://www.criticalstart.com/fall-of-sudo-a-pwnage-collection/#:~:text=Finding%20Linux%20servers%20heavily%20reliant,quickly%20become%20security's%20worst%20nightmare
- “Is it possible to use wget for copying files in my own system?” - Askubuntu - April 12, 2018:
  - https://askubuntu.com/questions/530686/is-it-possible-to-use-wget-for-copying-files-in-my-own-system