```
git clone https://github.com/5ud0ch0p/linux-privesc
```

Linux Privilege Escalation

Troy (@5ud0ch0p)
Andrew (@HillsBraindead)

linux privesc

- privilege model
- recon
- auth weaknesses
- weak file permissions
- built-in escalation mechanisms && misconfiguration
- service misconfiguration
- artefact exploitation
- escaping restrictions
- *advanced (SELinux, LD_PRELOAD)**

struct

- technique
- hands-on
- hints
- review

30 mins

- hands-on; 3 levels
  - intro
  - intermediate
  - annoying*

mindset

- some might seem:
  - simple
  - strange
  - confusing
  - impossible

Welcome to hacking!

- almost all challenges are representative
- some (not many) are contrived

0cc4m$ r4z0r

ask questions!

wat do

```
root@wopr:~$
```

basics && reconnaissance

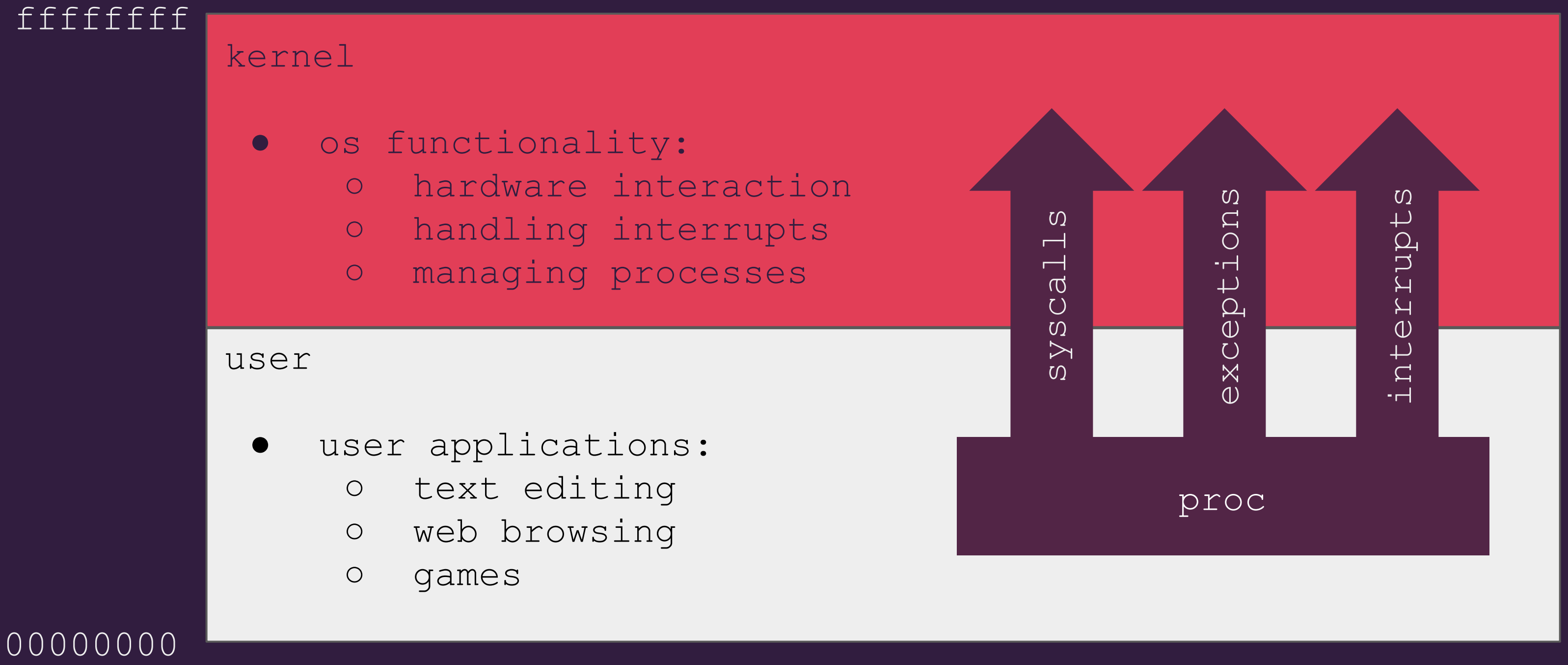
uid(0) / ‘root’

- highest user privilege on a *nix device
- as an attacker, high value:
  - read+write access to all data
  - access to all functionality
  - full control

uid(0) / ‘root’ - privs

- power control
- control over peripherals and components
- creating/starting/stopping services
- user management
- (un)installation of packages
- device configuration
- binding to privileged ports (1-1024)

usr vs krnl



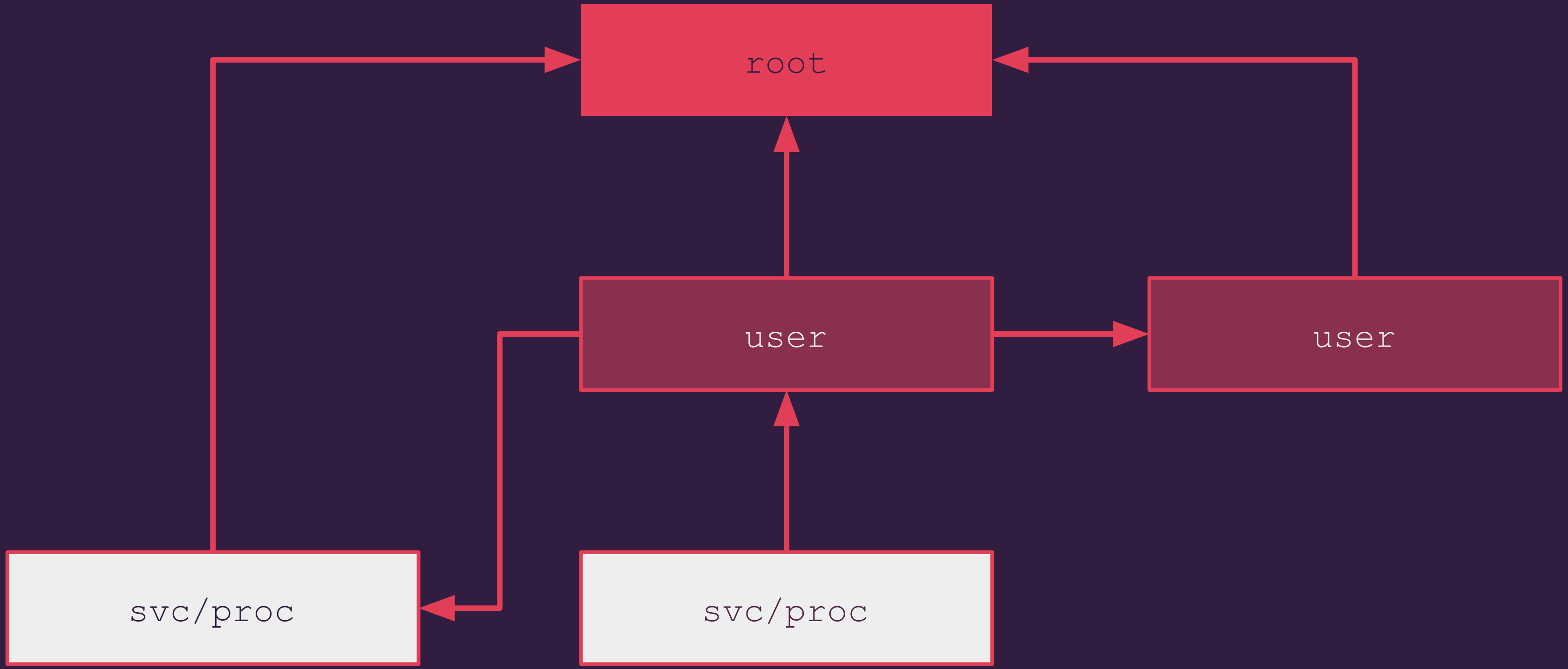
kernel

- os functionality:
  - hardware interaction
  - handling interrupts
  - managing processes

user

- user applications:
  - text editing
  - web browsing
  - games

priv esc



users / authentication

- /etc/passwd
- /etc/shadow
- /etc/group
- pam
- svc-specific (~/.ssh/authorized_keys)
- su/sudo*

privs && access control

- “EvErYtHiNg Is A fIlE”
- discretionary access control
  - rwx
- mandatory access control
  - SELinux/apparmor, etc.

discretionary access control

- e.g.: rwxr-xr-x

$$\overbrace{\texttt{rwx}}^{\texttt{user}}\ \overbrace{\texttt{r-x}}^{\texttt{group}}\ \overbrace{\texttt{r-x}}^{\texttt{others}}$$

```
-rw-r--r- 1 root root 1734 Jun 14 08:58 /etc/passwd
drwx------ 2 dhcpcd dhcpcd 4096 Jun 14 08:58 dhcpcd
```

mandatory access control

- SELinux, apparmor, etc.
- rwx may not == rwx
- generally, two modes:
  - report-only
  - enforce
- implementation specific:
  - SELinux - contexts/policies
  - Apparmor - path-dependent, mixing of modes

suid / sgid

- suid : rwsr-xr-x
  - executes as **user** who owns file

```
-rwsr-xr-x 1 root root 63640 Feb 4 23:31 /usr/bin/passwd
```

- sgid : rwxr-sr-x
  - executes as **group** who owns file

```
-rwxr-sr-x 1 root tty 34784 May 24 18:09 /usr/bin/wall
```

privs && svcs && procs

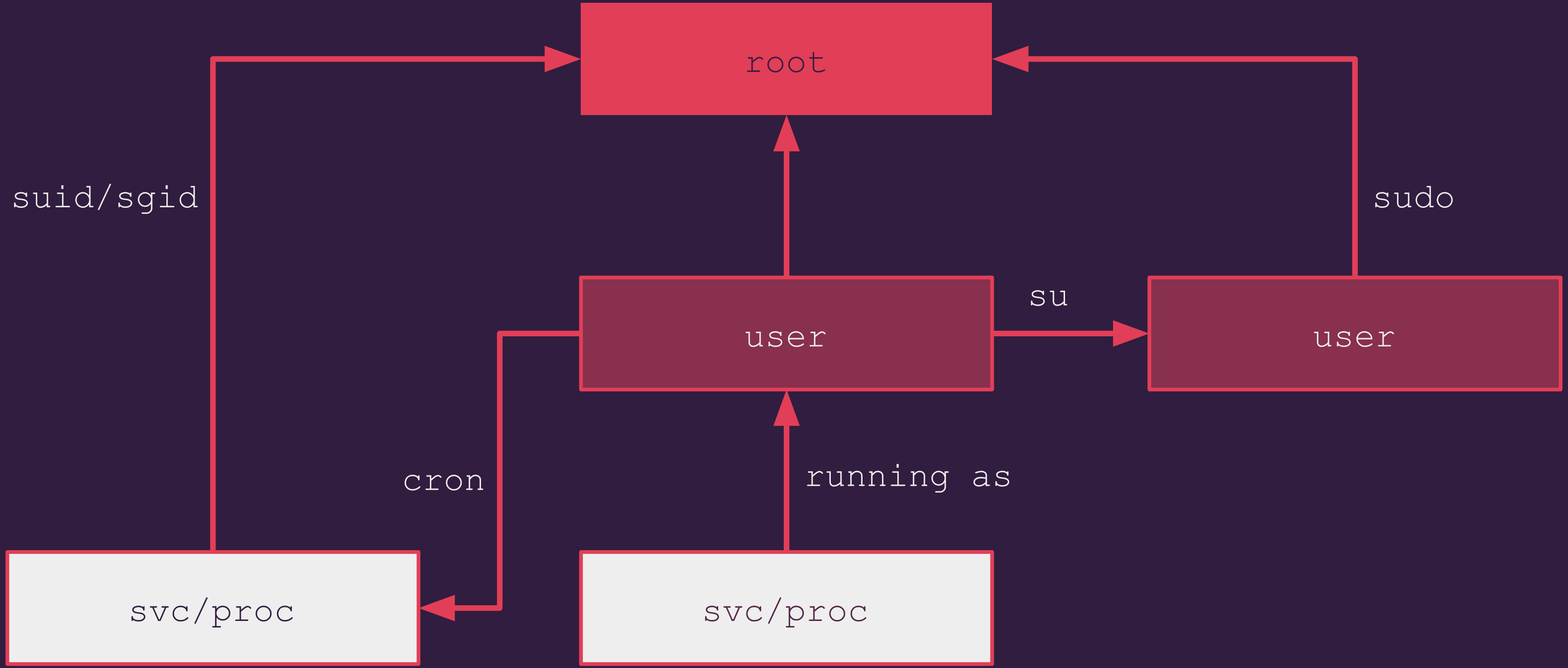
- everything runs as a ‘user’

- svcs/procs need privs
  - running as a priv account
  - run as root
  - e.g. cron, systemd, etc.

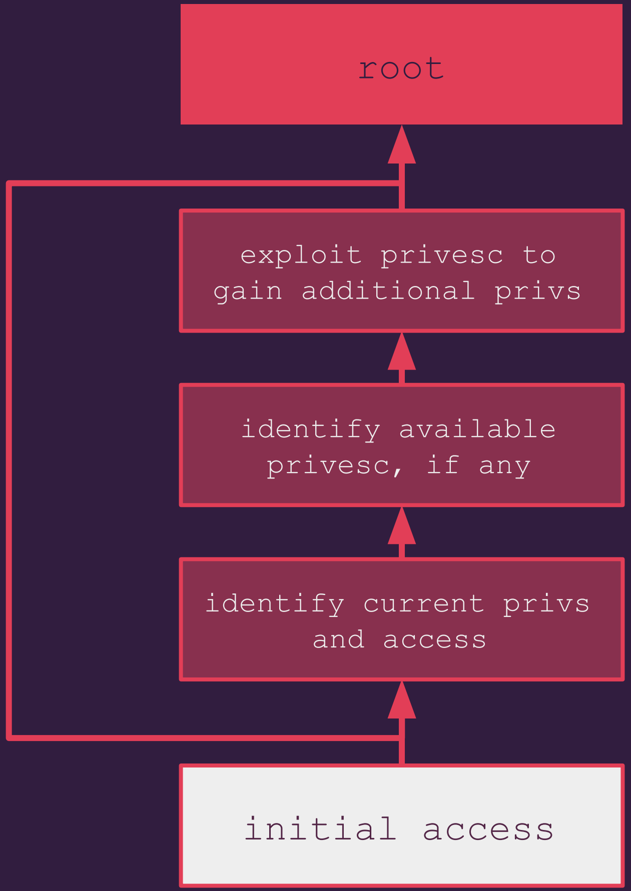
suid / sgid, etc.

- svcs might also shed privs
  - run as low-priv as possible
  - apache, nfs, etc.
    - wwwdata

priv esc


root
suid/sgid
sudo
su
user
user
cron
running as
svc/proc
svc/proc

method


root

exploit privesc to gain additional privs

identify available privesc, if any

identify current privs and access

initial access

recon

- users && groups
  - /etc/passwd, /etc/shadow, /etc/groups, etc.
- processes (&& configs)
  - ps/ss, env, documentation!
- services (&& configs)
  - netstat, systemctl, documentation!
- files && contents
  - ls, find, grep, strings, cat, etc.
- binaries && execution perms
  - ls, find, etc.

git / docker setup

docker setup && test

- docker setup
  - check the README in the repo

- lowpriv:lowpriv

- run some recon
  - get used to OEL!

practical 0x00

- don’t need automated tooling
  - **strong recommend not to use**
  - recommend cat, ls, find, grep, ps, etc.

- don’t diff the image!
  - yeah you’ll solve the challenge
  - but often can’t do this in the wild!

authentication weaknesses

authentication

- ssh, su/sudo, telnet, etc.
  - authentication often required (passwds, keys, pam)

- auth data often:
  - stored incorrectly
    - (plaintext, weak hash functions)
  - generally weak
    - (root:root, weak key lengths)

authentication

- /etc/passwd

```
user1:x:1000:1000:User 1:/home/user1/:/bin/bash
user2:<hash>:1001:1001:User 2:/home/user2/:/bin/bash
```

- /etc/shadow

```
user1:$hashid$salt$hash:12345:0:90:10:::
user2:$hashid$salt$hash:11223:0:90:10:::
```

- /etc/group

```
group1:x:40:user1,user2
group2:x:41:user2
```

practical 0x01

practical 0x01

- ssh as lowpriv
- **~/configure-privescs**
  - to configure practical and difficulty
- get ‘root’

review

- weak creds
  - pwd-based auth only as good as pwd
- key OS files
- cred storage mechanisms
- weak cred storage

file permissions

file usage

- general user
  - scripts, backups, debug data
- service
  - conf, auth
- type
  - plaintext (freetext, etc.)
  - specific formats (zip, pcap, configs, etc.)
  - binary (programs, dump, etc.)
  - special (directories, socket, link, etc.)

file perms (umask)

- file *and* directory
  - not all behave as expected!
  - s on directory?
    - suid - *generally* ignored
    - sgid - new files/subdirs inherit gid

- rwx vs octal
  - think of binary bits

$$\begin{array}{ccc} \overbrace{\texttt{rwx}}^{\text{user}} & \overbrace{\texttt{r-x}}^{\text{group}} & \overbrace{\texttt{r-x}}^{\text{others}} \\ 111 & 101 & 101 \\ 7 & 5 & 5 \\ & 755 & \end{array}$$

dir structure

- /   ‘root’ directory
    - /bin/   ‘core’ binaries (ls, cat, cd, etc.)
    - /etc/   config files
    - /home/   user home directories
    - /tmp/   temporary files
    - /usr/   user-land programs/data
    - /var/   things likely to change (e.g. logs)
    - (non-exhaustive)

- some have unique properties
    - /tmp/ - non-boot-persistent, 777

practical 0x02

practical 0x02

- ssh as lowpriv
- get ‘root’

review

- importance of:
  - acls
  - specific files/dirs

- identifying:
  - files of interest
  - and how to find them

built-in escalation mechanisms

perms plz

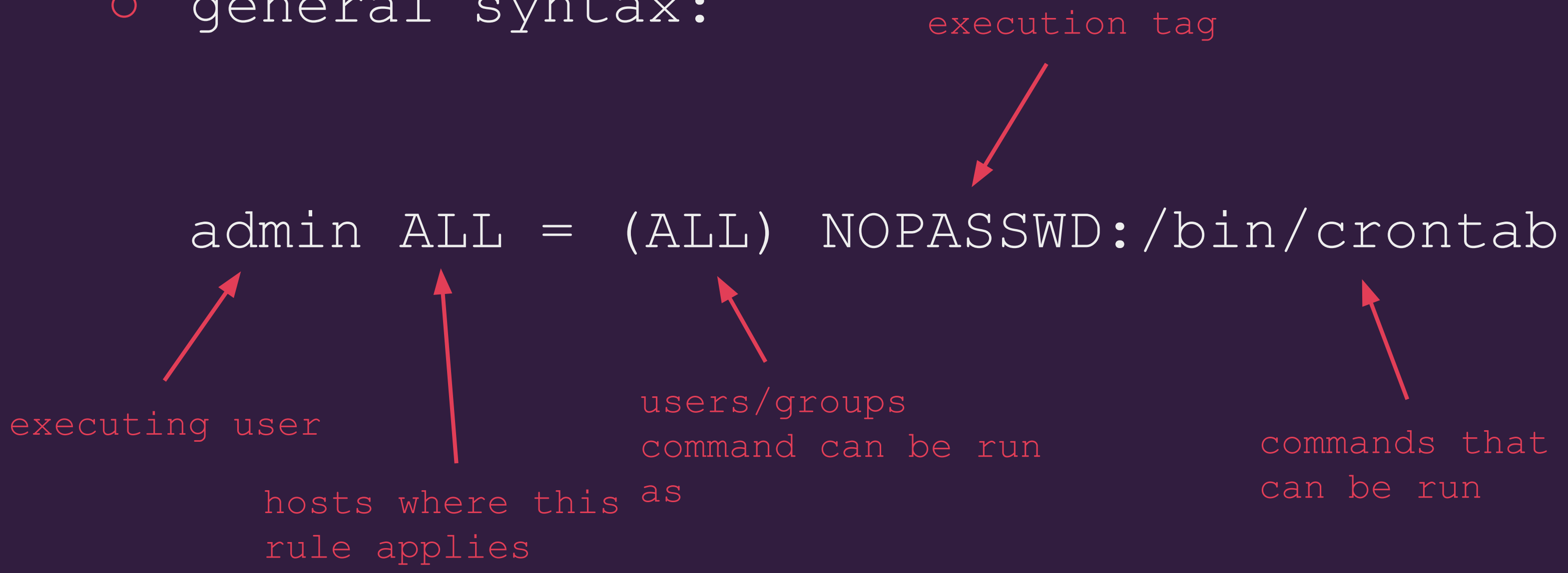
- usr needs additional permissions *temporarily*
- might be:
    - (re)starting/stopping services
    - (un)installing packages
    - rebooting machine

su(do)

- su
  - substitutes user and group IDs
  - spawns a shell (by default)
  - requires **target user** auth success
- sudo
  - launches single command (by default)
  - requires **current user** auth success
- default behaviours
  - su -c ≈ sudo
  - su ≈ sudo /bin/bash

sudo

- /etc/sudoers
  - who can execute what, as who
  - general syntax:



```
admin ALL = (ALL) NOPASSWD:/bin/crontab
```

execution tag

executing user

hosts where this rule applies

users/groups command can be run as

commands that can be run

sudo

- /etc/sudoers, e.g.:

```
%wheel ALL = (ALL) ALL
%admins ALL = (ALL) NOPASSWD:/tmp/admin_tools/*
lowpriv localhost = root /home/*/*/config
```

sudo

- `sudo -l`
- `/etc/sudoers.d/*`
- `man sudoers`

practical 0x03

practical 0x03

- ssh as lowpriv
- get ‘root’
- **remember; file access permissions!**

suid / sgid

- Provides functionality using effective uid/gid.
- e.g.:
  - user changing their password
    - /etc/shadow, owned by root:root, rw-------
  - mount a drive
    - hardware IO; needs to run as root
  - configuring cron (more on this later)
    - /etc/cron* owned by root

suid / sgid

- suid : rwsr-xr-x
  - executes as **user** who owns file

```
-rwsr-xr-x 1 root root 63640 Feb 4 23:31 /usr/bin/passwd
```

- sgid : rwxr-sr-x
  - executes as **group** who owns file

```
-rwxr-sr-x 1 root tty 34784 May 24 18:09 /usr/bin/wall
```

suid / sgid

- find -perm:
  - -4000 = suid
  - -2000 = sgid
  - -6000 = both
- don't need sudo

practical 0x04

practical 0x04

- ssh as lowpriv
- get ‘root’

review

- built-in escalation mechanisms
  - sudo/su/suid/sgid
- dangers of sudoers misconfigurations
- awareness of suid/sgid
  - how this can be abused.

artefacts and remnants

footprints in the sand

- usr actions leave traces
  - /tmp/, /home/usr/, etc.
  - histfile
  - syslog

- sysadmin processes might not clean-up
  - user creation/deletion
  - software installation/removal

recon is important!

- specific files often of interest:
  - ~/.bash_history
  - /var/log/*

- sysadmins leave remnants too:
  - *.bak
  - ./sysadmin.sh
  - mysql -u root -p <whatever>
  - orphaned uids owning files
    - find / [-nouser | -nogroup]

practical 0x05

practical 0x05

- ssh as lowpriv
- get ‘root’

review

- usr behaviour, sysadmin processes can have unintended consequences
- context dependent, but can be of use
    - often opportunistic, but carpe diem!

escaping restricted execution environments

lockdown

- usr account might have:
    - limited permissions
    - limited allowed command set
        - login places usr in limited exec env

- exploitation of svc might grant access to limited features we want more!

breakout

- what are we running in?
- what can we do in our restricted env?
- documented escapes?
- can we fundamentally bypass restrictions?

- what kind of input might break controls?
  - ${}, ${{}}, ||, &&, <, >, ;, etc.

practical 0x06

practical 0x06

- ssh as lowpriv
- get ‘root’
- **get interactive, command-line access (/bin/bash or similar) as root!**

review

- execution restrictions or controls
- awareness of ‘extra’ functionality
- not always secure by default
    - shouldn’t be assumed either!

service misconfiguration

svcs

- proc running (often backgrounded)
- run as a specific user
    - sometimes to shed perms (www-data)
    - sometimes because they need perms (root)

svcs

- often provides a... service

db	mysql, postgresql
http	apache, nginx
scheduled jobs	cron, systemd
remote management	ssh, telnet
file sharing	ftp, ssh
networking	dns, dhcp

- configs can be complex and tricky
  - can introduce vulns, privescs
  - mostly file-based (/etc/)

svcs

- remember - recon!

- what is running?
- what is it running as?
- where/how is it configured?

- docs/manpages
- distro-dependent

cron

- run X at time/frequency
- crontab
  - -l = show user crontab
  - -e = edit user crontab
- /etc/cron*
  - crontab (file)
  - cron.[hourly|daily|weekly]
  - cron.d/
  - cron.[deny|allow]

cron

- /var/spool/cron/
  - file per user - (‘crontab’)
  - editable: crontab -e
  - default perms: 600
- in OEL (docker container):
  - /etc/cron*
  - /var/spool/cron

file syntax

```
<m> <h> <day of month> <month> <day of week> <command>
```

```
*/20 * * * * zip -r logs.bak.zip /var/log/
2 5 * * * systemctl restart networking
0 9 9 6 * /root/start.sh
```

practical 0x07

practical 0x07

- ssh as lowpriv
- get ‘root’

review

- svcs highly specific
- remember:
  - recon
  - approach/methodology/questions
- docs/manpages!
  - some behaviours not obvious

advanced:
shared objects

shared objects/libs

- compiled collections of functions, code, etc.
  - libc
  - libcrypt
  - libusb

- can be used by multiple programs

- given lib can have 2 names:
  - “library name”/”soname” - libc.so.6
  - filename - /usr/lib/libc.so.6

- similar concept to DLLs in Windows

basic .so

- basic code structure of a shared library (in C):
  - header file (something.h)
  - source (something.c)

something.h

```
#ifndef ...
#define ...

extern void something();

#endif
```

something.c

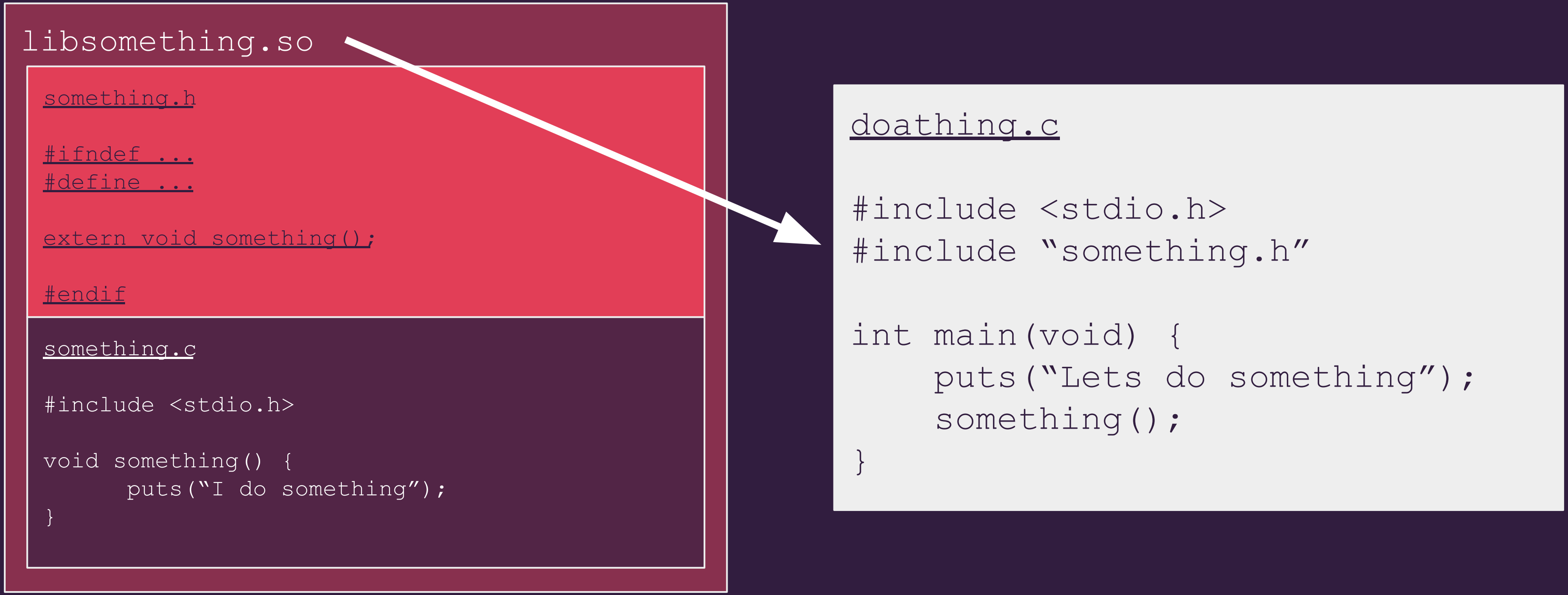
```
#include <stdio.h>

void something() {
    puts(“I do something”);
}
```

- then compile as shared object

basic .so

- we can #include our SO in other code:



libsomething.so

something.h

```
#ifndef ...
#define ...

extern void something();

#endif
```

something.c

```
#include <stdio.h>

void something() {
      puts(“I do something”);
}
```

doathing.c

```
#include <stdio.h>
#include “something.h”

int main(void) {
    puts(“Lets do something”);
    something();
}
```

shared objects/libs

- SOs **linked** during compilation, load, or runtime
- list SO dependencies:
  - `ldd <binary>`
- list exported symbols from an SO:
  - `objdump -T /path/to/lib.so`
  - `nm -D /path/to/lib.so`
    - T prefix indicates export

linking

- static
  - all libs copied into binary
  - code/libs/etc placed into memory at once by OS
  - once linked, libs are static
    - changes need recompilation of binary

- **dynamic**
  - **names** of libs placed into binary
  - OS loads main binary/libs separately
  - libs can change*
    - no need to recompile binary

load order

- OS looks for dynamically-linked libs in various locations:
  - DT_RPATH in dynamic section of binary
  - LD_LIBRARY_PATH
  - DT_RUNPATH
  - /etc/ld.so.cache
  - /lib*
  - /usr/lib*

* Can also be /lib64, /usr/lib64

https://man7.org/linux/man-pages/man8/ld.so.8.html

search path manipulation

- LD_LIBRARY_PATH
- RPATH
- LD_PRELOAD

- can be hacky solutions to dependency hell
- often used for debugging
  - can be left behind after debugging!

LD_LIBRARY_PATH

- envvar
- *nix-specific (not all *nix, only some)
- contains colon-delimited list of dirs
  - searched before typical search order directories
- how could this be problematic?

```
LD_LIBRARY_PATH=

DT_RPATH
LD_LIBRARY_PATH
DT_RUNPATH
/etc/ld.so.cache
/lib*
/usr/lib*
```

```
LD_LIBRARY_PATH=/tmp/

/tmp/
DT_RPATH
LD_LIBRARY_PATH
DT_RUNPATH
/etc/ld.so.cache
/lib*
/usr/lib*
```

RPATH

- similar to LD_LIBRARY_PATH
- compiled within binary
  - not dependent upon usr envvars

```
-rpath=/path/to/something
```

- LD_RUN_PATH is envvar equivalent

RPATH

- `objdump -x /path/to/binary | grep RPATH`
- can we write to this location?
- can another user write to this location?

LD_PRELOAD

- envvar
- lists SOs that override other SOs
- can be used alongside sudo:

```
Defaults envkeep += LD_PRELOAD
```

- why might this be problematic?

practical 0x08

practical 0x08

- ssh as lowpriv
- get ‘root’
- msfvenom *might* be helpful here
  - but not required here
- automated tooling can help find these privescs
  - but not required here

review

- .so
- load order important!
- LD_LIBRARY_PATH, RPATH, LD_PRELOAD

advanced:

selinux

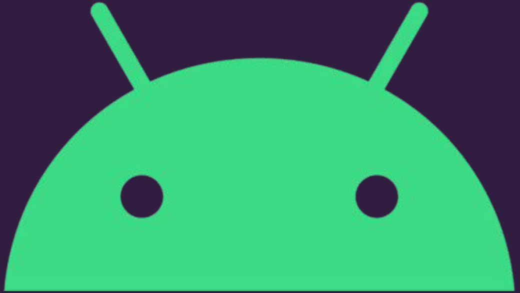
caveat

- following section is (very) high level summary
- selinux is complex
- providing absolute basics
  - (incase you find yourselves in selinux-land)
- no practicals

irl

- available on multiple distros
  - (good) support: RHEL/CentOS, Fedora, Gentoo, et. al
  - in repos for: Ubuntu, Debian
    - (apparmor normally used instead)
  - no official support: arch (only krnl modules)

- in use by default on, e.g.:



https://source.android.com/docs/security/features/selinux

mandatory access control

- selinux, apparmor, etc.
- rwx may not == rwx
- generally, two modes:
    - report-only
    - enforce
- implementation specifics:
    - selinux - contexts/policies
    - apparmor - path-dependent, mixed modes

selinux

- confines user programs, system services
- ideally, minimize privs for a given proc
- enforced by krnl
- no inherent ‘root’
    - root also subject to selinux criteria!
- selinux users != linux users
    - OS maps linux -> selinux users
    - can also map to roles

access control mechanisms

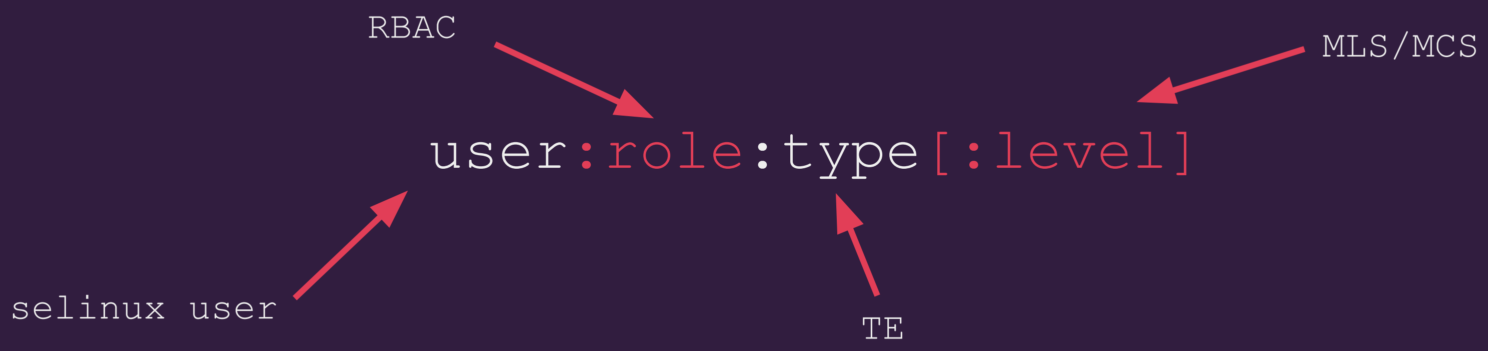
- type enforcement
  - all subjects, objects are allocated a type
- role-based access control
  - selinux users associated to 1(+) roles
- multi-level security
  - uses ‘security level’ to enforce policies
    - e.g. “Top_Secret”, “Confidential”, etc.
- multi-category security
  - categorises objects to enforce selinux policies
    - “Log_Files”, “Customer_Data”, etc.

access control mechanisms

- contexts:
  - username
  - role
  - domain (or type)
  - level

- (almost) everything assigned a label
  - network ports, files, hw, etc.
  - access between labelled objects controlled by policy files
    - can be manually adjusted (!)

access control mechanisms



RBAC

MLS/MCS

```
user:role:type[:level]
```

selinux user

TE

```
-rw-rw-r-- lowpriv lowpriv standard_u:access_r:user_home_t:s0 notes.txt
```

inheritance

- default: context inheritance allowed
  - files created in dir of context dir_t also created with dir_t
  - child procs spawned from proc with exec_t also are exec_t
- different from DAC
  - dir = rw-, file created follows user default mask
- how could this be bad?

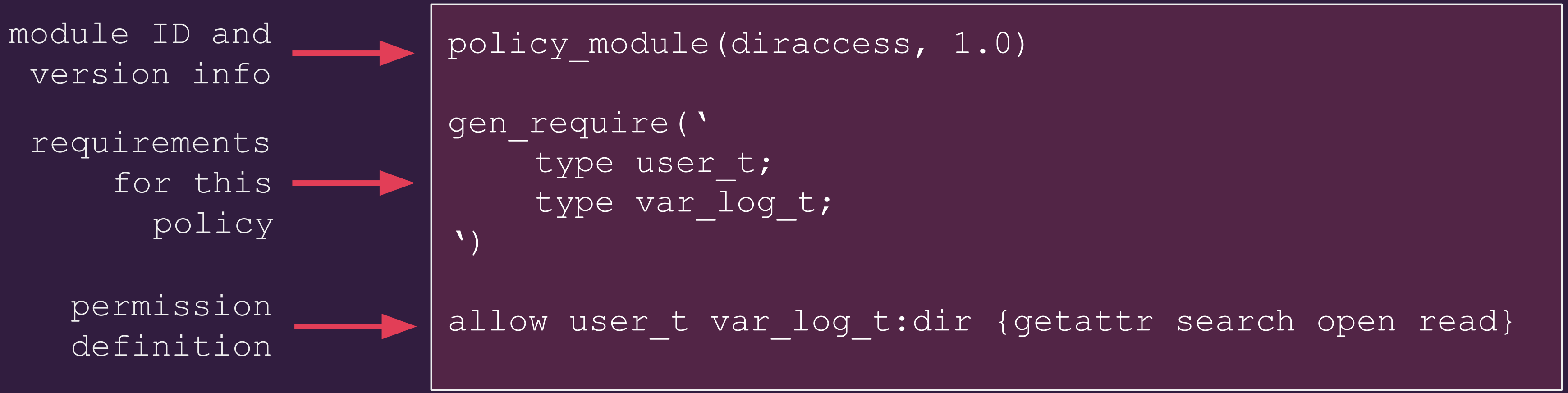
policies

- grouping of rules of explicit perms, e.g.:
    - read/execute
    - bind/connect to port
- typical policy consists of:
    - mapping file (.te)
    - “file contexts” file (.fc) [opt]
    - interface file (.if) [opt]
- compiled into .pp binaries which are krnl loaded
- collectively, define domain transition
- default policies exist, but specific

enforcement

- policy controls access between labelled processes and objects
- different enforcement modes, e.g.:
  - disabled
  - permissive (warnings shown on violation)
  - enforcing (access denied/logged)
  - targeted
    - confines system procs
    - all other procs run in unconfined
    - protects key processes without harming UX

policy (.te)



https://wiki.gentoo.org/wiki/SELinux/Tutorials/Creating_your_own_policy_module_file

module ID and version info

requirements for this policy

permission definition

```
policy_module(diraccess, 1.0)

gen_require(`
    type user_t;
    type var_log_t;
')

allow user_t var_log_t:dir {getattr search open read}
```

https://selinuxproject.org/page/ObjectClassesPerms

domain transitions

- 3 conditions:
  - policy allows transition from origin domain to target
  - origin domain has execute on file
  - file context is defined as target domain entry point

```
type_transition <process> <origin_context> : <target_context>
```

```
type_transition backup_t backup_exec_t : fileaccess_t
```

https://wiki.gentoo.org/wiki/SELinux/Tutorials/How_does_a_process_get_into_a_certain_conte

tooling

- -Z
- getenforce/sestatus - selinux status
- chcon
- semanage - core selinux mgmt
  - user-role association
  - change security context of target
  - proc permission assignment
- seinfo - query policy components
- ssh <user>/<selinux_role>@hostname
- ++

privesc...?

- looking for:
    - selinux permissive (!), or disabled(!)
    - overly-permissive policy entries/typedefs
    - usr/procs with incorrect context
    - type_transition of interest
- think about:
    - what can we achieve with current context?
    - do we need to access additional perms?
        - if only interested in data, httpd_* might ok
        - but need to be running as this or a child
            - no migrating!

privesc...!

- if successful:
  - disable selinux enforcing!
  - write to etc_t/shadow_t
  - load krnl modules
  - etc.

misc

- /etc/selinux/config
- logging:
  - /var/log/messages
  - /var/log/audit/audit.log
  - /var/lib/setroubleshoot/se_troubleshoot_database.xml
  - systemd

misc

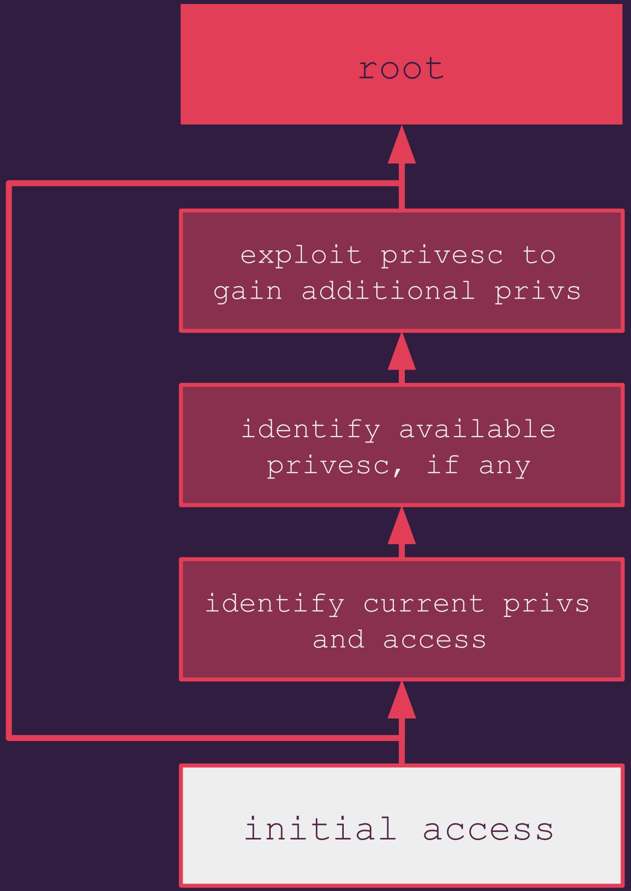
- selinux allocates context to an object
- objects only interact with their context
- can transition to other domains via policy

final challenges

review

- privilege model
- recon
- auth weaknesses
- weak file permissions
- built-in escalation mechanisms && misconfiguration
- service misconfiguration
- artefact exploitation
- escaping restrictions
- ** advanced (SELinux, LD_PRELOAD)*

method



root

exploit privesc to gain additional privs

identify available privesc, if any

identify current privs and access

initial access

review

- importance of recon
- importance of methodology

?

^D

Linux Privilege Escalation

Troy (@5ud0ch0p)
Andrew (@HillsBraindead)