Linux Privilege Escalation

Troy Defty

Lukasz Gogolkiewicz

who

- Lukasz

who

- Troy (@5ud0ch0p)
- 8 years in industry
  - 5.5 in UK, 2.5 AU
- mostly red team
- recently blue team
  - Google – SecEng manager
- DEFCON 27, AISA, etc.

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*

caveat

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

welcome to hacking!

- almost all challenges are representative of real scenarios we’ve encountered
- some (not many) are contrived to provide a means of practice

basics && reconnaissance

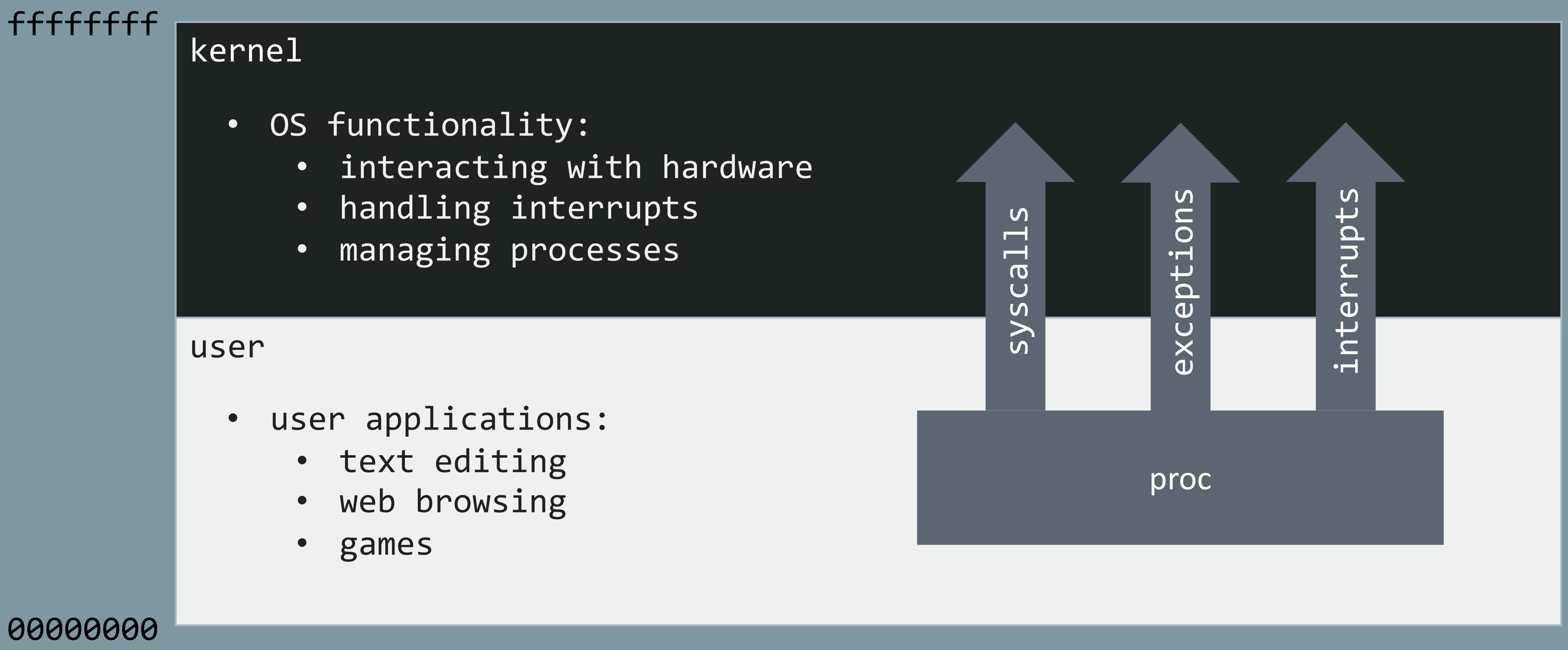
uid(0)

- ‘root’
- highest user privilege on a *nix device
- as an attacker, high value:
  - read+write access to all data
    - credentials
    - database contents
    - defacement of webapps, etc
  - access to all functionality
    - repurposing device (!)
    - network pivot

uid(0)

- typical high privs:
  - device 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



ffffffff

kernel

- OS functionality:
    - interacting with hardware
    - handling interrupts
    - managing processes

user

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

00000000

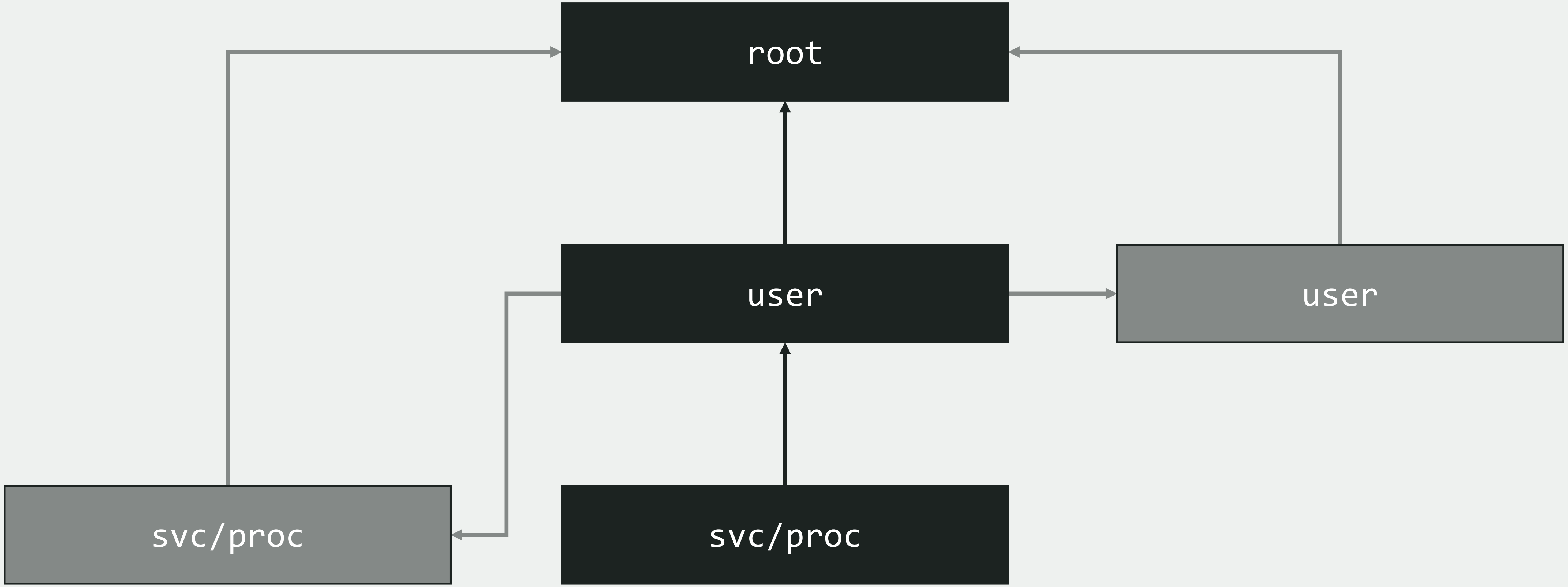
syscalls

exceptions

interrupts

proc

privilege escalation



users and authentication

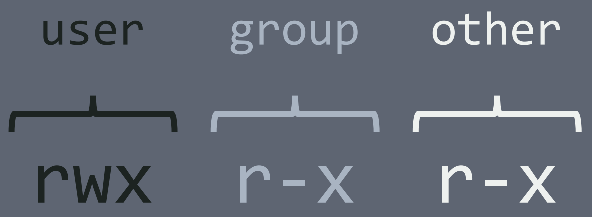
- /etc/passwd, /etc/shadow, /etc/group
- pam
- svc specific mechanisms (.ssh/authorized_keys)
- su/sudo*

privs && access control

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

privs && access control

- DAC : rwxr-xr-x



user group other

rwx r-x r-x

```
-rw-r—r– 1 root root 1734 Jun 14 08:58 /etc/passwd

drwx------ 2 dhcpcd dhcpcd 4096 Jun 14 08:58 dhcpcd
```

privs && access control

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

privs && access control

- 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 (*effective* gid)

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

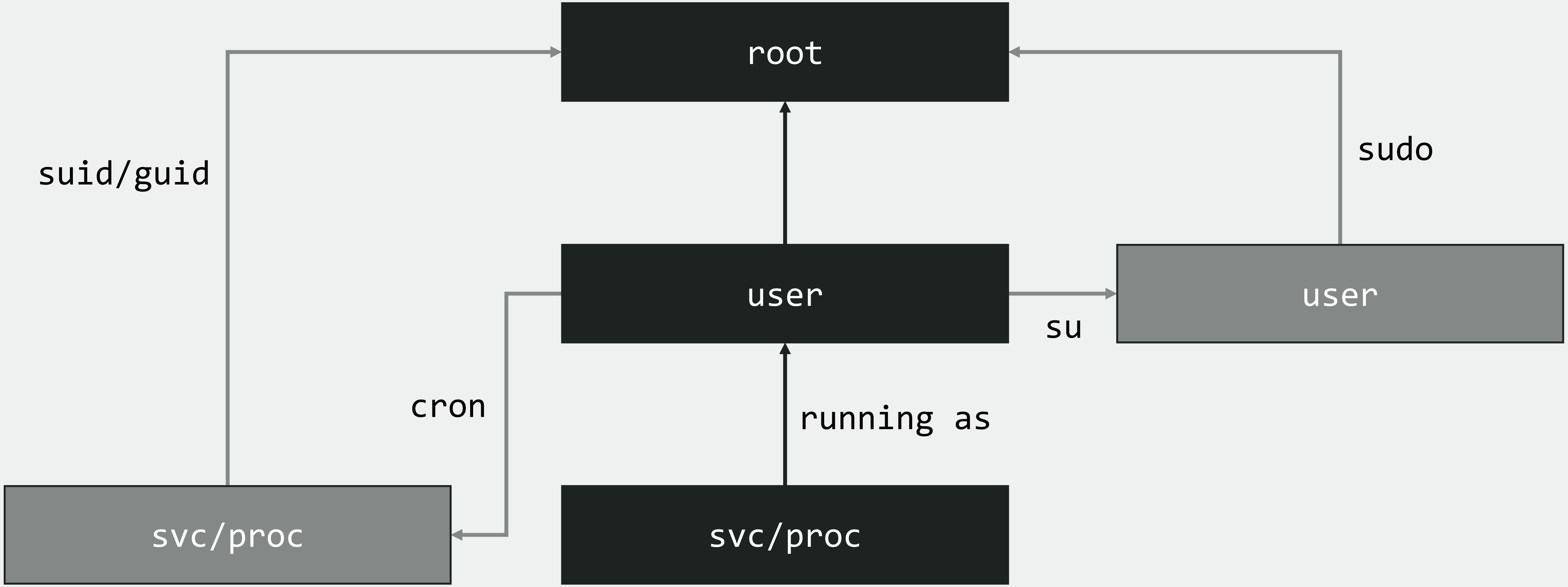
privs && svcs && procs

- everything runs as a ‘user’
- svcs and procs need privs
  - running as priv. account
  - run as root
  - e.g. cron, systemd, etc

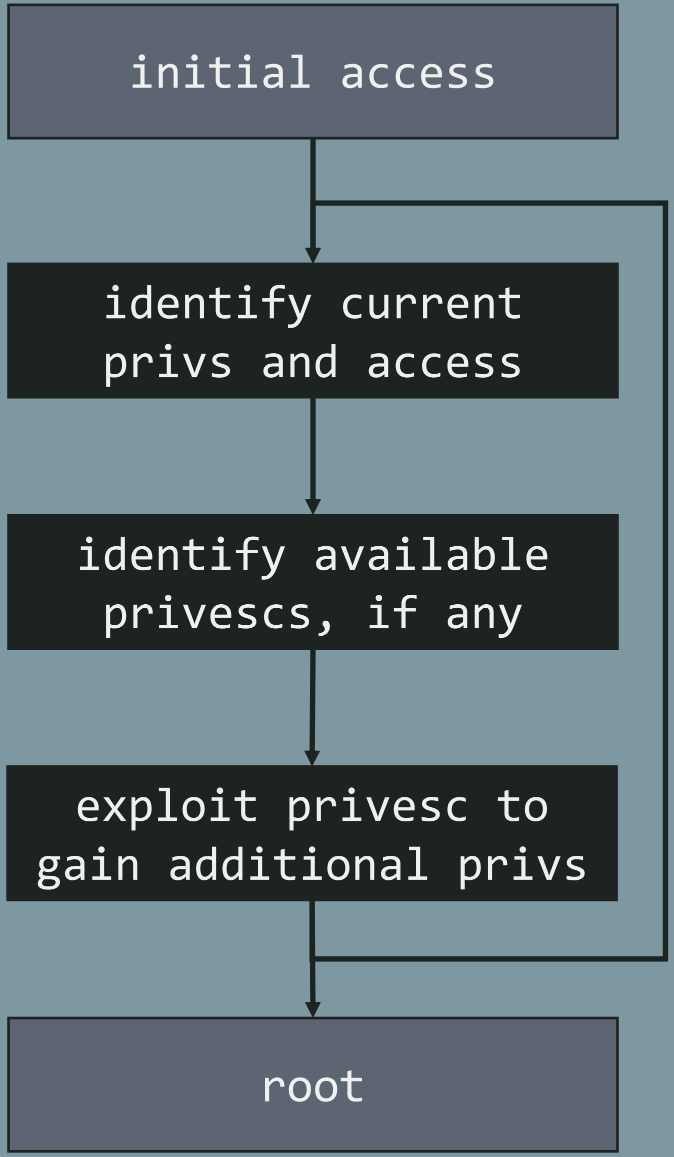
suid/sgid, e.g.

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

privilege model



method



initial access

identify current privs and access

identify available privescs, if any

exploit privesc to gain additional privs

root

recon

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

git && docker
good

quick git

```
git config --global core.autocrlf input

git clone https://github.com/5ud0ch0p/auscert2020-
linux-privesc.git
```

docker setup && test

- docker setup
- lowpriv:lowpriv
- run some recon
  - get used to CentOS!

practical 0x00

- don’t need automated tooling, but feel free
  - would strongly recommend to do without
- recommend:
  - ssh, cat, ls, find, grep, ps, netstat, etc.
  - documentation!
- don’t diff the image!
  - yeah you’ll solve the challenge
  - but we’re here to learn!
  - likely can’t ask the admin to roll the box back and then configure everything again in the real world!

authentication weaknesses

authentication

- ssh, su/sudo, telnet, etc.
  - authentication often required
    - passwds
    - keys
    - pam
- auth data often mishandled, stored incorrectly, generally weak
  - “password”, “<username>”, “”
  - weak key lengths, permissions
  - plaintext storage
  - hash functions

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

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

review

- weak pwds
  - pwd-based auth only as good as pwd
- key OS files
- pwd storage mechanisms, hashing
- weak pwd storage

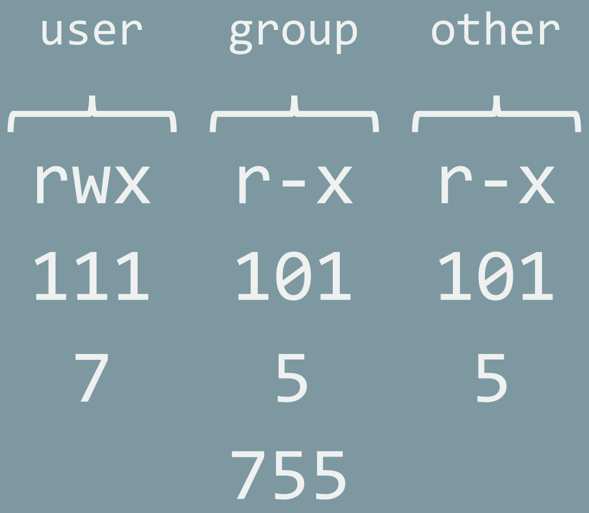
file permissions

file usage

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

file permissions

- file *and* directory
  - not all behave as expected!
  - s on directory?
    - suid *generally* ignored
    - sgid – new files and subdirs inherit groupID of dir, rather than usr
- rwx vs octal
  - think of binary bits



user	group	other
rwx	r-x	r-x
111	101	101
7	5	5
	755	

dir structure

- /
    - /bin/ - ‘core’ binaries (ls, cat, cd, etc.)
    - /etc/ - configuration files
    - /home/ - user home directories
    - /tmp/ - temporary files
    - /usr/ - user-land programs and data
    - /var/ - things likely to change; logs, e.g.
    - (non-exhaustive)
- depends on distro!
- some have unique properties
    - /tmp/ - non-boot-persistent, 777

practical 0x02

- ssh as lowpriv
- get ‘root’

review

- importance of:
    - acls
    - specific files/dirs.
- identifying:
    - files of interest
    - and ways of finding them

built-in escalation mechanisms

perms plz

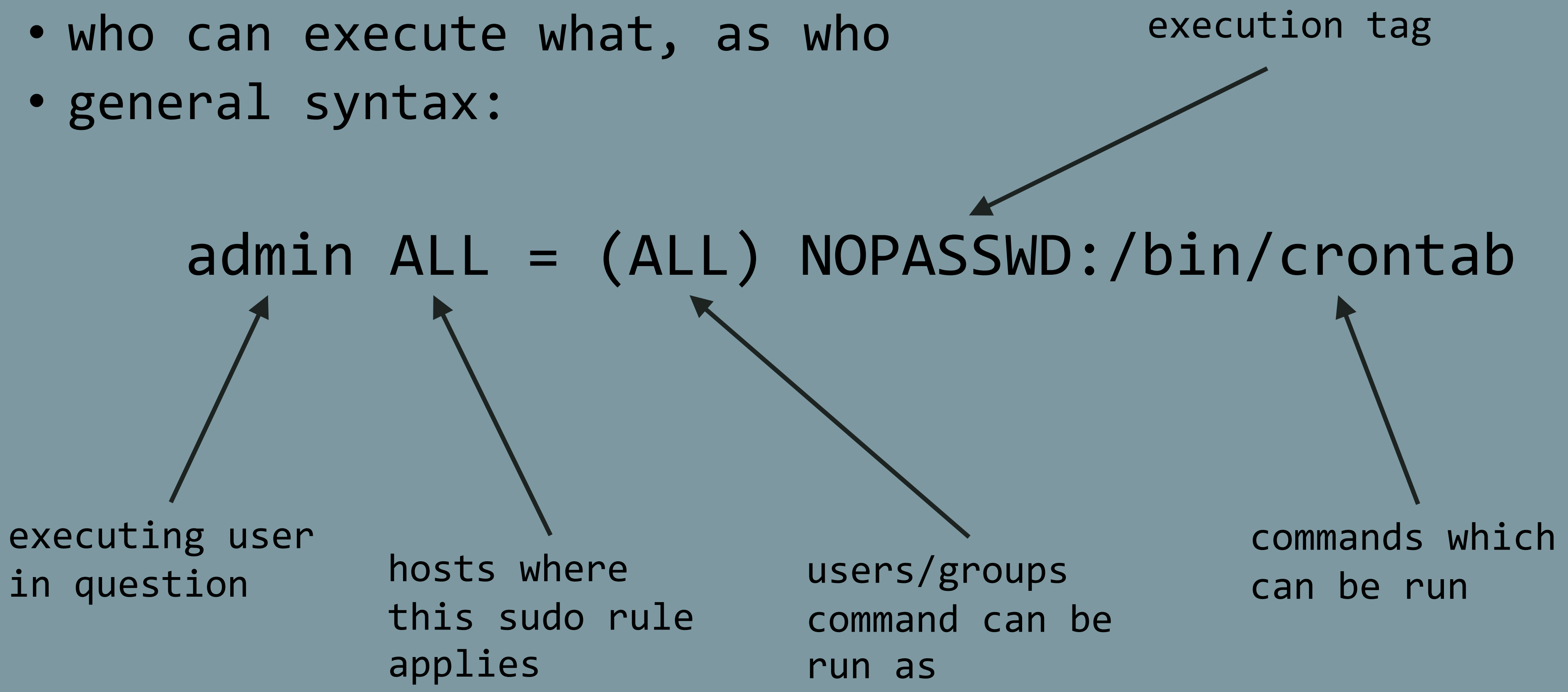
- usr needs additional permissions temporarily
  - doesn’t need full root all the time
- usr:
  - (re)starting/stopping services
  - (un)installing packages
  - reboot machine

su(do)

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

sudo

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



execution tag

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

executing user in question

hosts where this sudo rule applies

users/groups command can be run as

commands which can be run

sudo

- /etc/sudoers
  - e.g.:

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

sudo

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

practical 0x03

- ssh as lowpriv
- get ‘root’

- **remember; file access permissions!**

suid/sgid

- provides limited functionality using effective uID/gIDs
- usr:
  - change their password
    - changes /etc/shadow, owned by root:root
    - passwd needs to run as root!
  - mount a drive
    - privileged action
    - needs to run as root!
  - configure cron
    - /etc/cron* owned by root (as it contains all user crons!)
    - needs to run as 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:
  - -2000 = sgid
  - -4000 = suid
  - -6000 = both suid and sgid
- don’t need sudo/su

practical 0x04

- ssh as lowpriv
- get ‘root’

review

- built-in escalation mechanisms
- dangers of sudo configs
- awareness of suid/sgid

artefacts and remnants

footprints in the sand

- usr actions leave traces
  - /tmp/, /home/usr/, etc.
  - histfile
  - syslog
- sysadmin processes can be insecure/incomplete in cleanup
  - user creation/deletion
  - software installation/removal

footprints in the sand

- recon is important!
- specific files often of interest
  - .bash_history
  - /var/log/*
- sysadmin processes leave remnants
  - *.bak
  - ./sysadmin.sh
  - mysql –u root –p <whatever>
  - orphaned userIDs
    - find / [-nouser|-nogroup]

practical 0x05

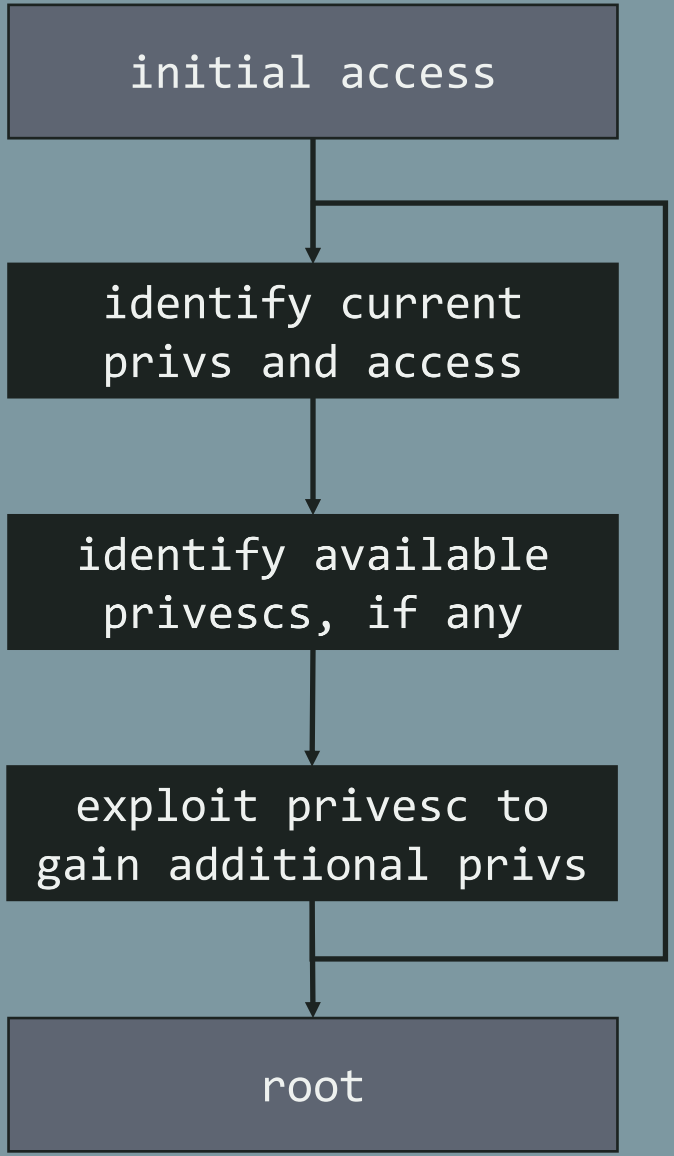
- ssh as lowpriv
- get ‘root’

review

- sysadmin processes can introduce unintended consequences
- context dependent, but can be of use!

EOF

review



initial access
identify current privs and access
identify available privescs, if any
exploit privesc to gain additional privs
root

review

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

review

- importance of recon
- importance of methodology

^D

Linux Privilege Escalation

Troy Defty

Lukasz Gogolkiewicz

$DAY=$((DAY+1))

review

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

escaping restricted execution environments

lockdown

- usr account might have limited permissions
- or a limited allowed command set
  - login places usr in limited execution environment
  - exploitation of service might grant access to limited features
- we want more privs/access!
  - escaping locked down environments
  - bypassing restrictions

breakout

- what are we running in?
- what can we do in our restricted environment?
- are there any documented escapes for this environment?
- is there a way we can fundamentally bypass the restrictions in place?
- what kind of input might break the restrictions?
    - ${}, ${{}}, ||, &&, <, >, etc.

practical 0x06

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

review

- awareness of ‘extra’ functionality
- not always secure by default!
    - and this shouldn’t be assumed!

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, etc.)
  - http (apache, nginx, etc.)
  - scheduled jobs (cron, systemd, etc.)
  - remote management (telnet, SSH, etc.)
  - file sharing (ftp, etc.)
  - networking (dns, dhcp, etc.)
- configs can be complex and tricky
  - can introduce vulns, privescs!
  - mostly file-based (often /etc/)

identifying the privesc

- remember recon!

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

- documentation/manpages
- often distro-dependent

cron

- crontab
  - -l = list user crontab
  - -e = edit user crontab
- /etc/cron*:
  - crontab (the file)
  - cron.hourly/, cron.daily/, cron.weekly/, cron.hourly/
  - cron.d/ - contains system cronjobs for various users
  - cron.deny or cron.allow – crontab access controls

cron

- /var/spool/cron/:
  - file per user (‘crontab’)
  - editable via crontab –e
  - default perms 600
- on CentOS (and in our container):
  - /etc/cron* generally used by anacron
  - /var/spool/cron generally used by cron

cron

- file syntax:

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

- ssh as lowpriv

review

- services can be highly specific
- remember recon
- remember approach/methodology
- docs/manpages very helpful
- some behaviours not obvious

advanced:
shared objects

shared libs/objects

- compiled collections of functions, code, etc.
  - libc
  - libcrypt
  - libusb
- can be used by multiple programs
- given lib can have two “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(void);

#endif
```

something.c

```
#include <stdio.h>

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

- then compile as a shared object

using our basic .so

- we can #include our SO similar to a core lib

libsomething.so

```
something.h

#ifndef ...
#define ...

extern void something(void);

#endif
```

```
something.c

#include <stdio.h>

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

```
doathing.c

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

int main(void) {
        puts(“Lets do something”);
        something();
        return 0;
}
```

shared libs/objects

- SOs are **linked** during compilation, load time or run time
- list shared object dependencies for a given binary:
    - ldd <binary>
- list exported symbols from a lib:
    - objdump –T /path/to/lib.so
    - nm –D /path/to/lib.so
        - T prefix indicates export

linking

- static
  - all libs copied into main binary
  - all code, libs, etc. placed into memory at once by OS
  - once linked, libs are static and changes require recompilation
- **dynamic**
  - names of libs placed into binary
  - OS then loads main binary and libs separately at runtime
  - libs can change (within reason!) and main binary does not require recompilation

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

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

RPATH

- similar to LD_LIBRARY_PATH
- but 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?

- DT_RUNPATH – similar RPATH, compiled into binary

LD_PRELOAD

- envvar
- lists SOs that override all normal shared objects
- often problematic with sudo:

```
Defaults      envkeep += LD_PRELOAD
```

- why?

practical 0x08

- ssh as lowpriv
- metasploit/msfvenom may be of help here
  - not necessary; can be done solely in C!
    - (if you have more time, would recommend giving it a go in C!)
    - why?
- automated tooling can make identifying this kind of privesc easier (sometimes!)

review

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

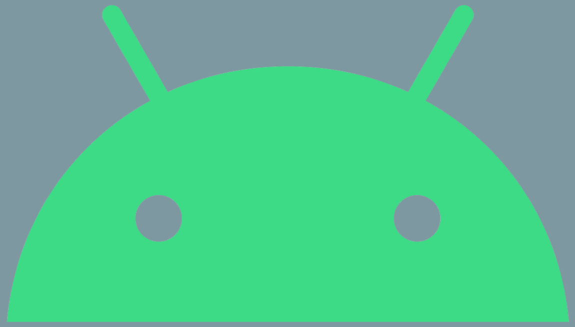
advanced:
selinux

caveat

- following section is a (very) high level summary
- selinux is complex
- section is to provide the absolute basics, should you find yourselves encountering selinux
- no practicals

in the real world

- available on multiple distros
  - (good) support: RedHat/CentOS, Fedora, Gentoo, et al.
  - in repos for: Ubuntu, Debian, (apparmor normally used) et al.
  - no official support: Arch (only kernel modules supported), et al.
- in use by default on, for example:



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

mandatory access control

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

selinux

- confines user programs, system services
- ideally, minimize required privileges for a given proc
- enforced by the kernel
- no inherent concept of ‘root’
  - root is subject to SELinux criteria!
- selinux users != linux users
  - OS maps linux users to selinux users
  - often used to also map to roles

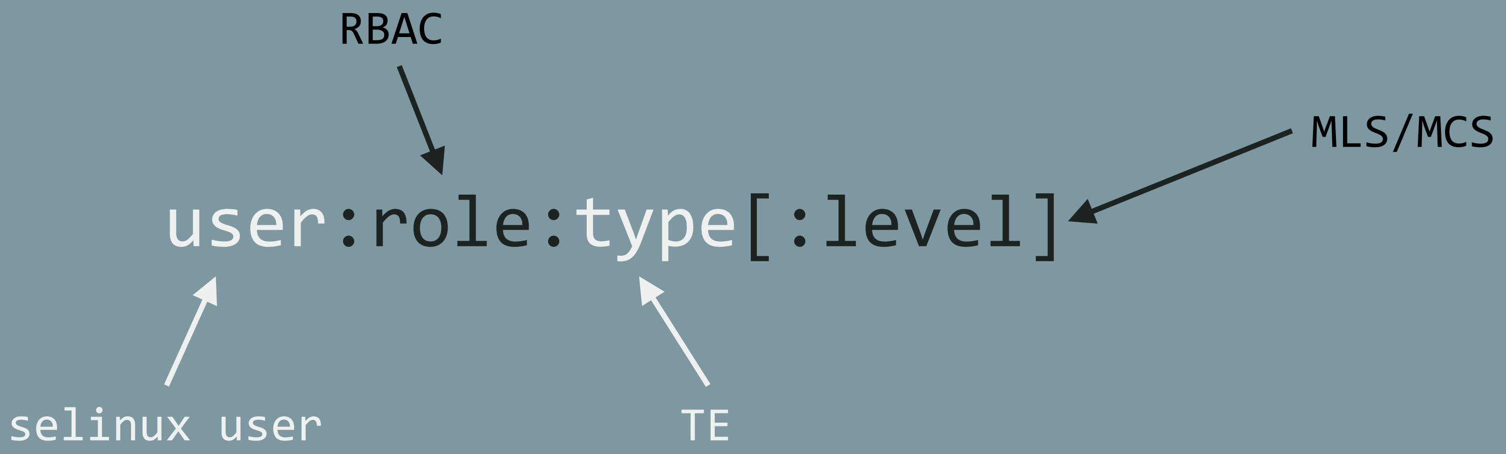
access control mechanisms

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

access control and contexts

- contexts:
  - username
  - role
  - domain (or type)
  - level
- (almost) everything is assigned a label
  - network ports, files, hardware, etc.
  - access between labelled objects controlled by policy files, but can be manually adjusted (!)

contexts



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 within a directory of context dir_t are also created with dir_t
  - child processes spawned from proc with exec_t also have exec_t
- different from DAC!
  - dir = rw-, file created file follows user default umask
- how could this be bad?

policies

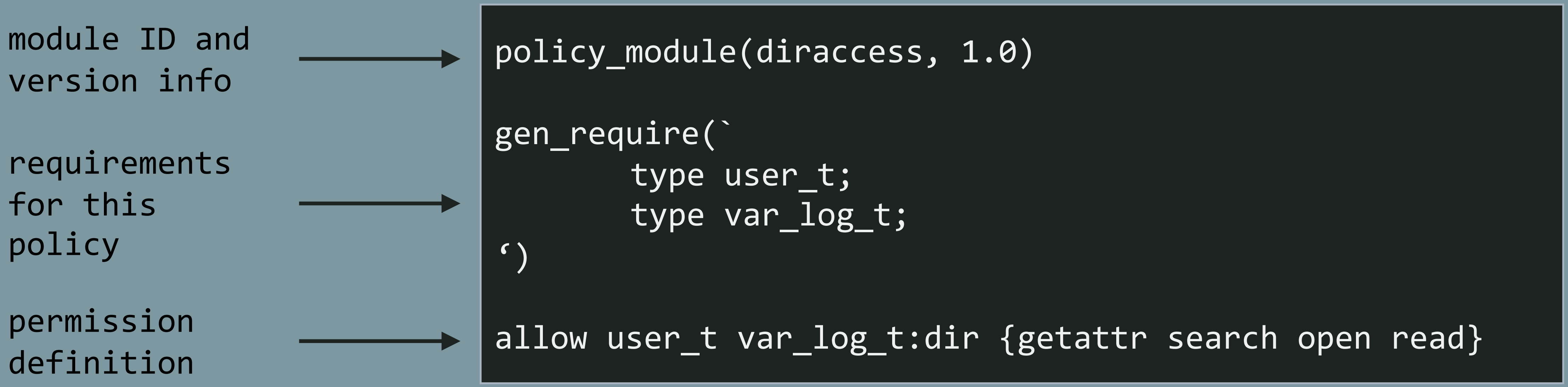
- grouping of rules of explicit permissions, e.g.:
  - read/execute
  - bind/connect to a port
- typical policy consists of:
  - mapping file (.te)
  - “file contexts” file (.fc) [optional]
  - interface file (.if) [optional]
- compiled into .pp binaries to be loaded into kernel space
- collectively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

enforcement

- policy controls access between a labelled process and labelled objects
- different enforcement modes (non-exhaustive):
  - disabled – no policy loaded
  - permissive – warnings printed on policy violation
  - enforcing – access denied, logged, on policy violation
  - targeted (default on CentOS):
    - confines specific system processes (httpd, named, dhcpd, mysqld)
    - all other system and user processes run in unconfined domain
    - designed to protect key processes without harming UX

policy

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

- three 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_context

tooling

- -Z
- getenforce/sestatus – selinux status
- chcon – similar to chmod/chown but temporary
- semanage – core selinux management, e.g.:
  - user role association
  - change security context of a target
  - proc permission management
- seinfo – query policy components
- ssh <user>/<selinux_role>@hostname
- many, many more

privesc...?

- looking for:
  - selinux permissive(!), or disabled(!!)
  - overly-permissive policy entries
  - overly-permissive type definitions
  - users/processes with incorrect context
  - type_transition of interest
- think about:
  - what can we achieve with our current selinux context?
    - how can we find this out?
  - do we need to try and access additional permissions?
    - if we are only interested in data, httpd_* might be sufficient!
    - but we need to be running in this or child processes; no migrating!

privesc...!

- if successful:
  - disable selinux enforcing mode!
  - writing to etc_t/shadow_t
  - loading kernel modules
  - etc.

misc.

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

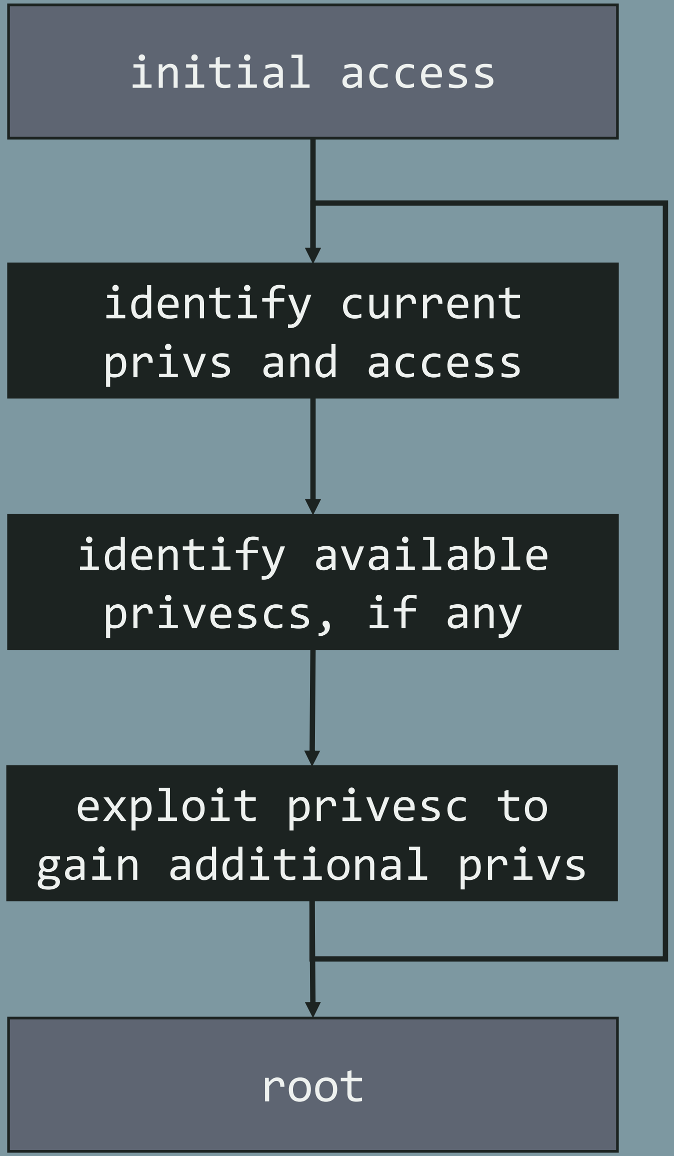
review

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

final challenges

EOF

review


initial access
identify current privs and access
identify available privescs, if any
exploit privesc to gain additional privs
root

review

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

review

- importance of recon
- importance of methodology

^D