



CSC 405

Linux Security

Aleksandr Nahapetyan
anahape@ncsu.edu

(Slides adapted from Dr. Kapravelos)

Logistics update

- Reminder: HW1 is due end of next week. You must log-in into the hw1 server by sunday end of day!
- I can not release lecture recordings
 - Solution: Start with an in-class exercise
- We've now had two snow days!
 - We have lost a good bit of the in-class workdays to this, so I will be hosting office hours next Tuesday in EBII 2240



In-class practice

- Difficulty: Easy
- Files: go.ncsu.edu/csc405-s26-04
- Spot the bug!

```
.section .text
.globl _start

_start:
    xor %rax, %rax
    push %rax

    movq $0x2F62696E2F7368, %rbx
    push %rbx

    mov %rsp, %rdi

    push %rax
    push %rdi
    mov %rsp, %rsi

    xor %rdx, %rdx

    mov $59, %al
    syscall
```

We are done with machine code!

We are done with machine code!

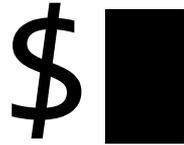
for now...

We are done with machine code!

for now...

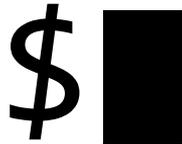
*Not really HW1 is not
canceled*

Reason



Having access to the shell means you have full control over the system

Reason



Having access to the shell means you have full control over the system
Which means all we're ever trying to do is reach **THAT** again

Reason

And it means we have access to all the tools available to Linux

\$1s

Having access to the shell means you have full control over the system
Which means all we're ever trying to do is reach **THAT** again

Reason

And it means we have access to all the tools available to Linux

```
$ cd  
ls
```

Having access to the shell means you have full control over the system
Which means all we're ever trying to do is reach **THAT** again

Reason

And it means we have access to all the tools available to Linux

```
$ rm -rf /  
cd  
ls
```

Having access to the shell means you have full control over the system
Which means all we're ever trying to do is reach **THAT** again

And it means we



is available to Linux

```
$rm -rf /
```

This is also a friendly reminder that some of the control we gain in this class can break a system

Having (please remember to always test things out on your VMs) the system gain

Shells (there are many)

- sh: basics, installed on everything
 - Might be dash under the hood
- bash: old, but has some quality of life things
 - !! will just execute the told command
 - \$? will give you the return code of the previous command
- zsh
 - oh-my-zsh is a set of plugins fancy shells
- fish/elvish/nu/xonsh
 - new, l33t, non-standard compliant.

```

ohmyzsh.sh demo

→ projects omz theme use eastwood
[~/projects]$ take omz-demo && git init
Initialized empty Git repository in /Users/robbyrussell/projects/omz-demo/.git/
[main] [~/projects/omz-demo]$ echo "TODO: This is my new README" > README.md
[main] [~/projects/omz-demo]$ git add README.md
* [main] [~/projects/omz-demo]$ git commit -m "Adding a README to new repo" --quiet
[main] [~/projects/omz-demo]$ echo "Wow, Oh My Zsh looks neat" >> README.md
* [main] [~/projects/omz-demo]$ git add -p
diff --git a/README.md b/README.md
index fc97e80..b0939f1 100644
--- a/README.md
+++ b/README.md
@@ -1,1,2 @@
- TODO: This is my new README
+Wow, Oh My Zsh looks neat
(1/1) Stage this hunk [y,n,q,a,d,e,?]? y
* [main] [~/projects/omz-demo]$ git commit -m "Updating README in the repo" --quiet
[main] [~/projects/omz-demo]$ git checkout -b feature/openai-integration
Switched to a new branch 'feature/openai-integration'
[feature/openai-integration] [~/projects/omz-demo]$ git checkout main
Switched to branch 'main'
[main] [~/projects/omz-demo]$ ..
[~/projects]$
  
```

Linux

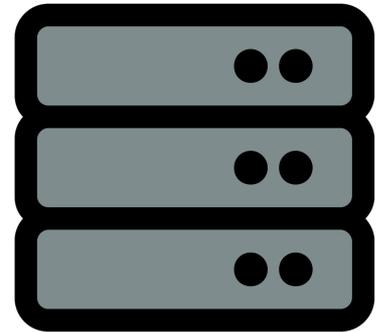
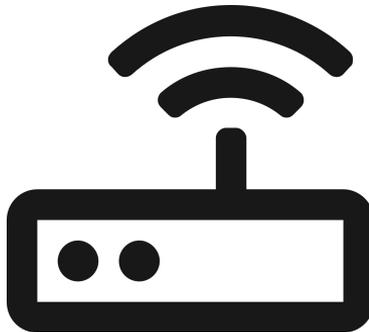
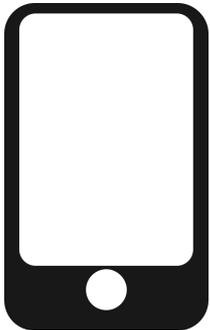
The most deployed operating system in the world

What are three devices that explain why?

Linux

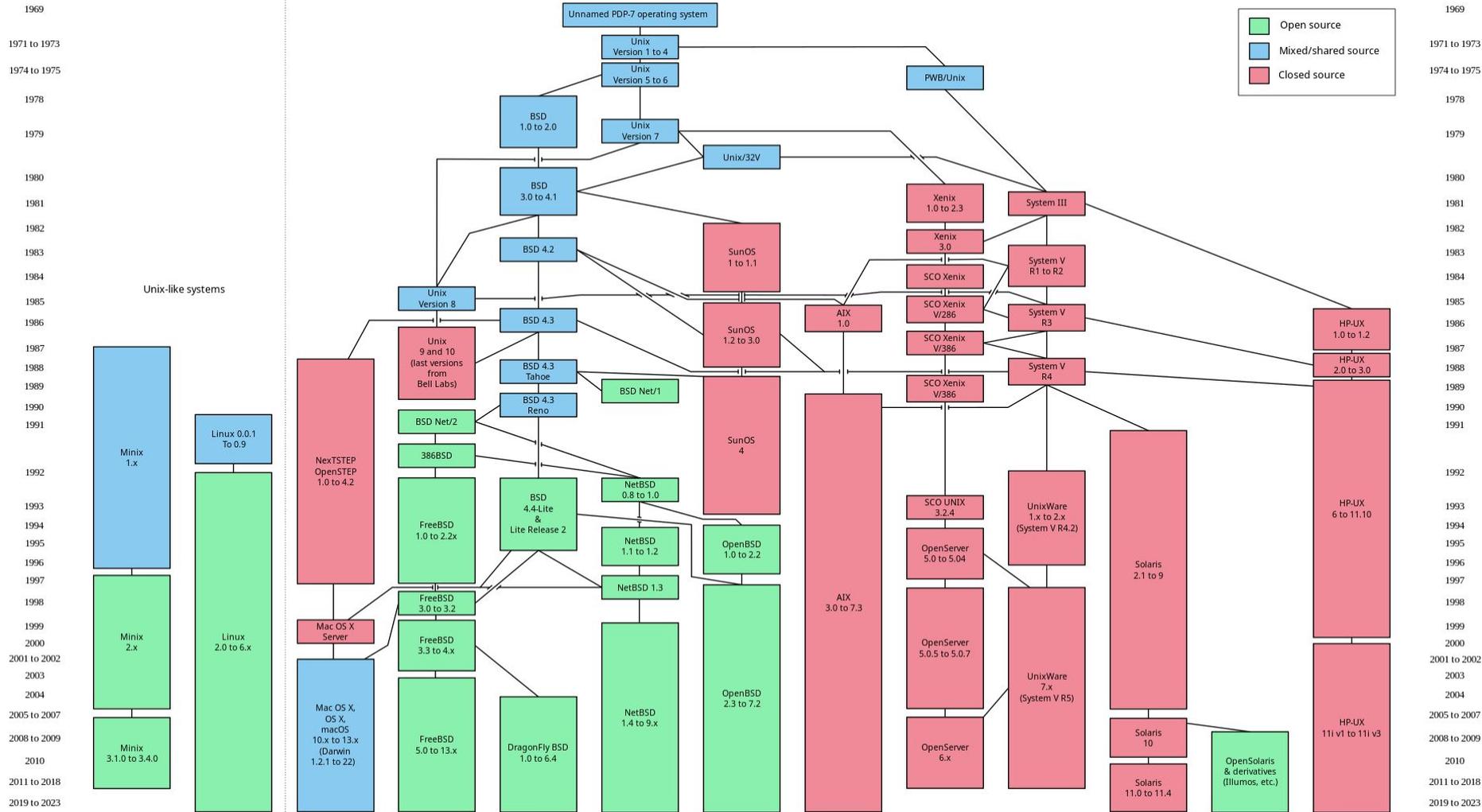
The most deployed operating system in the world

What are three devices that explain why?



A History of Linux

In the beginning,
there was **UNIX**[®]
An Open Group Standard



Unix

- Started in 1969 at AT&T / Bell Labs
- Split into a number of popular branches
 - BSD, System V (commercial, AT&T), Solaris, HP-UX, AIX
- Inspired a number of Unix-like systems
 - Linux, Minix, macOS
- Standardization attempts
 - POSIX, Single Unix Specification (SUS), Filesystem Hierarchy Standard (FHS), Linux Standard Base (LSB), ELF

A History of Linux

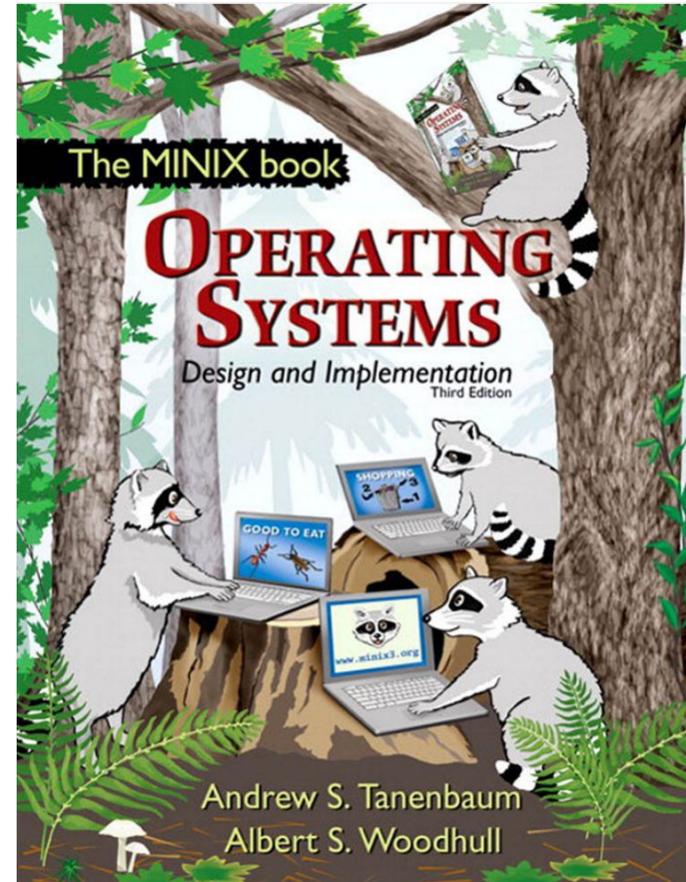
Linus Torvalds



A History of Linux

Linus developed the first iteration of Linux while in college (~1987) coding in Minix and thought...

"there must be a better way"

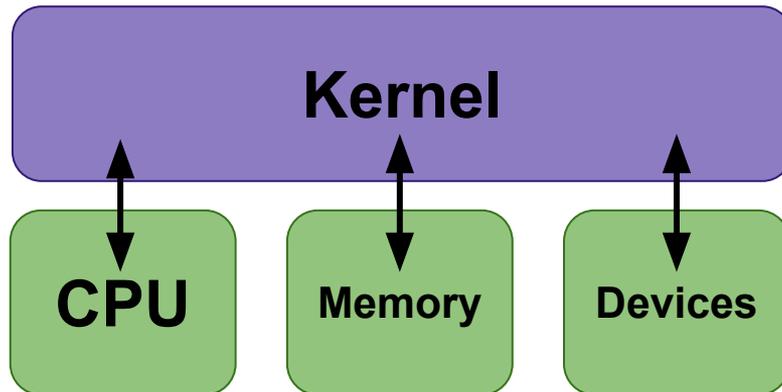


The Kernel

Core component to the operating system

Manages system resources

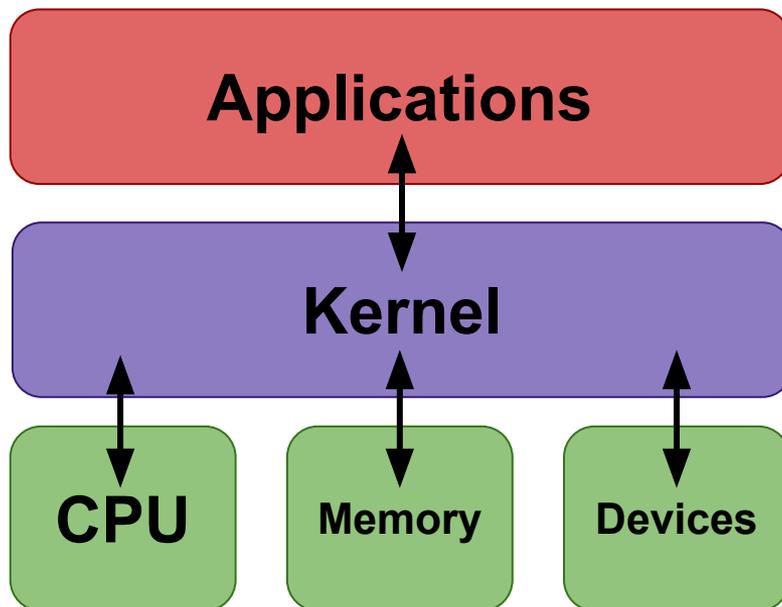
Provides essential services like scheduling, drivers, memory management, and **system calls**



The Kernel

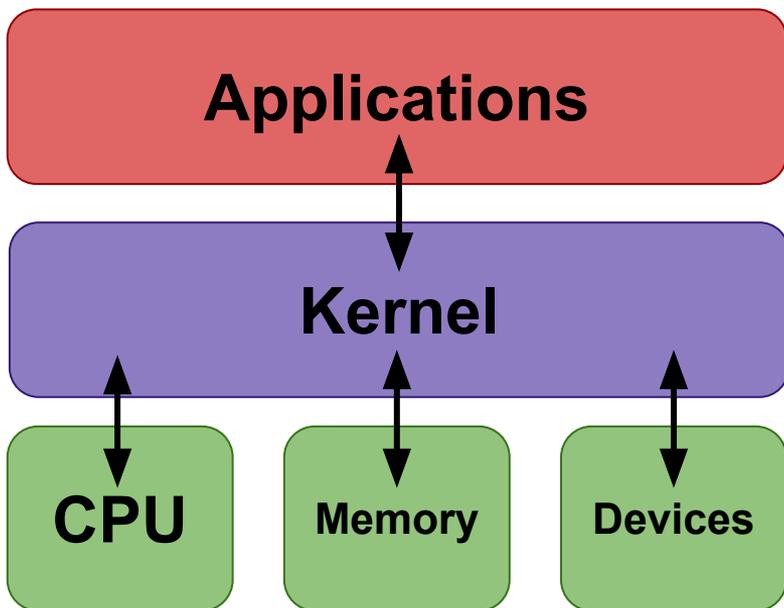
Serves as the bridge between software and hardware

Facilitates communication between them



The Kernel

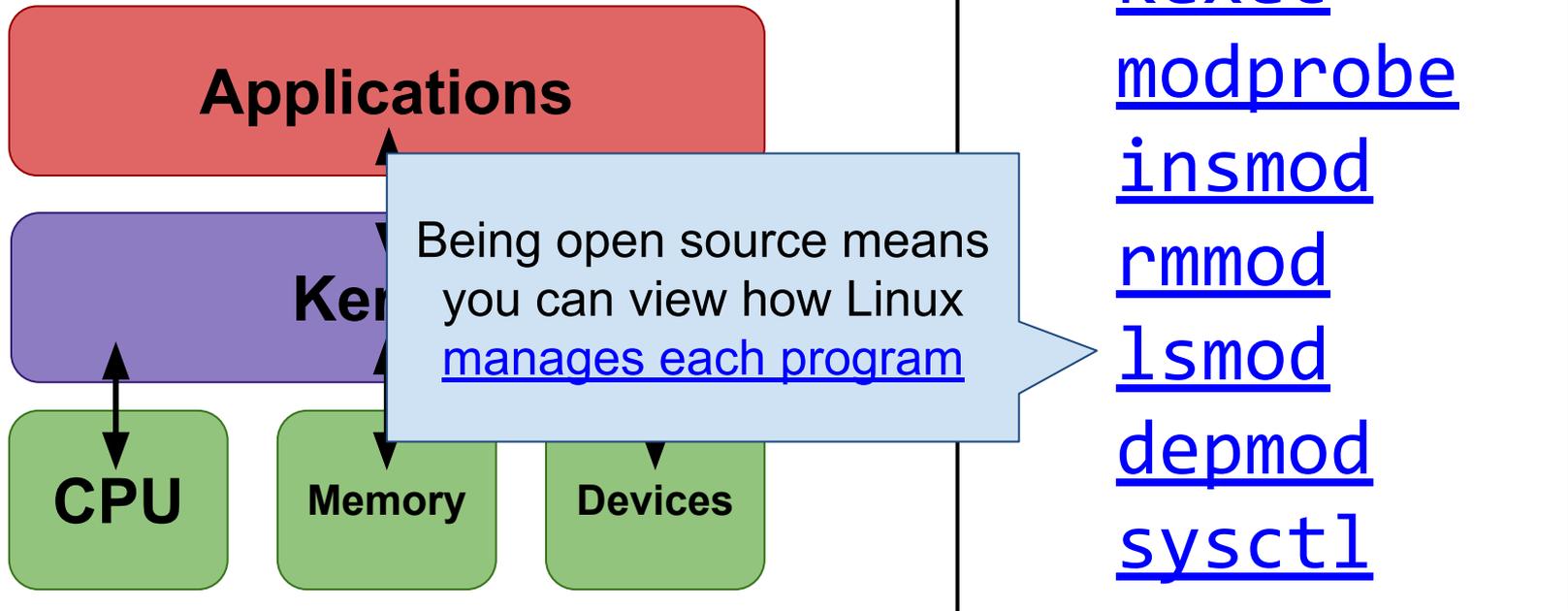
Linux for example is a collection of C binaries for handling the kernel



grub
kexec
modprobe
insmod
rmmod
lsmod
depmod
sysctl

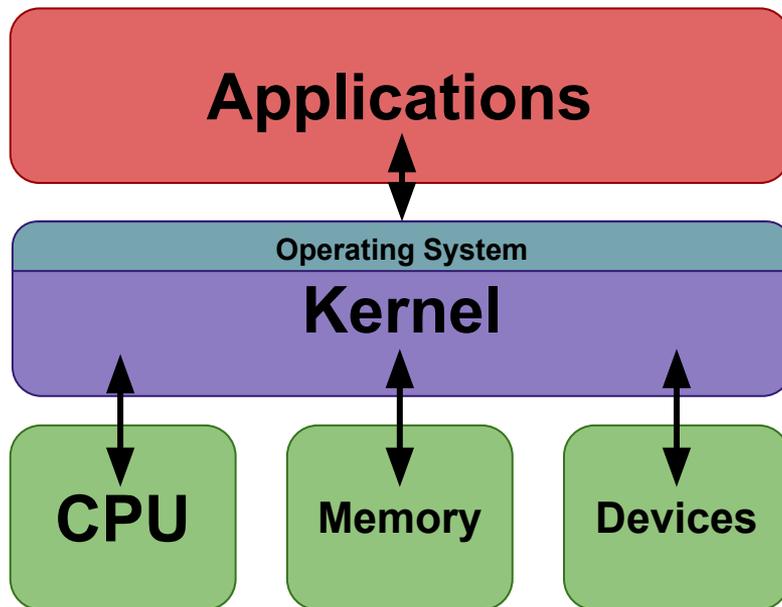
The Kernel

Linux for example is a collection of C binaries for handling the kernel



The Operating System

The operating system, on the other hand, is essentially built around the kernel to provide a user-friendly interface



Kernel vulnerabilities

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2017-12762	119		Overflow	2017-08-09	2017-08-25	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p>In /drivers/isdn/4l/isdn_net.c: A user-controlled buffer is copied into a local buffer of constant size using strcpy without a length check which can cause a buffer overflow. This affects the Linux kernel 4.9-stable tree, 4.12-stable tree, 3.18-stable tree, and 4.4-stable tree.</p>														
2	CVE-2017-11176	416		DoS	2017-07-11	2017-08-07	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p>The mq_notify function in the Linux kernel through 4.11.9 does not set the sock pointer to NULL upon entry into the retry logic. During a user-space close of a Netlink socket, it allows attackers to cause a denial of service (use-after-free) or possibly have unspecified other impact.</p>														
3	CVE-2017-8890	415		DoS	2017-05-10	2017-05-24	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p>The inet_csk_clone_lock function in net/ipv4/inet_connection_sock.c in the Linux kernel through 4.10.15 allows attackers to cause a denial of service (double free) or possibly have unspecified other impact by leveraging use of the accept system call.</p>														
4	CVE-2017-7895	189			2017-04-28	2017-05-11	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p>The NFSv2 and NFSv3 server implementations in the Linux kernel through 4.10.13 lack certain checks for the end of a buffer, which allows remote attackers to trigger pointer-arithmetic errors or possibly have unspecified other impact via crafted requests, related to fs/nfsd/nfs3xdr.c and fs/nfsd/nfsxdr.c.</p>														
5	CVE-2017-0648	264		Exec Code	2017-06-14	2017-07-07	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
<p>An elevation of privilege vulnerability in the kernel FIQ debugger could enable a local malicious application to execute arbitrary code within the context of the kernel. This issue is rated as High due to the possibility of a local permanent device compromise, which may require reflashing the operating system to repair the device. Product: Android. Versions: Kernel-3.10. Android ID: A-36101220.</p>														
6	CVE-2017-0605	264		Exec Code	2017-05-12	2017-05-19	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
<p>An elevation of privilege vulnerability in the kernel trace subsystem could enable a local malicious application to execute arbitrary code within the context of the kernel. This issue is rated as Critical due to the possibility of a local permanent device compromise, which may require reflashing the operating system to repair the device. Product: Android. Versions: Kernel-3.10, Kernel-3.18. Android ID: A-35399704. References: QC-CR#1048480.</p>														
7	CVE-2017-0564	264		Exec Code	2017-04-07	2017-07-10	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
<p>An elevation of privilege vulnerability in the kernel ION subsystem could enable a local malicious application to execute arbitrary code within the context of the kernel. This issue is rated as Critical due to the possibility of a local permanent device compromise, which may require reflashing the operating system to repair the device. Product: Android. Versions: Kernel-3.10, Kernel-3.18. Android ID: A-34276203.</p>														
8	CVE-2017-0563	264		Exec Code	2017-04-07	2017-07-10	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
<p>An elevation of privilege vulnerability in the HTC touchscreen driver could enable a local malicious application to execute arbitrary code within the context of the kernel. This issue is rated as Critical due to the possibility of a local permanent device compromise, which may require reflashing the operating system to repair the device. Product: Android. Versions: Kernel-3.10. Android ID: A-32089409.</p>														
9	CVE-2017-0561	264		Exec Code	2017-04-07	2017-08-15	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p>A remote code execution vulnerability in the Broadcom Wi-Fi firmware could enable a remote attacker to execute arbitrary code within the context of the Wi-Fi SoC. This issue is rated as Critical due to the possibility of remote code execution in the context of the Wi-Fi SoC. Product: Android. Versions: Kernel-3.10, Kernel-3.18. Android ID: A-34199105. References: B-RB#110814.</p>														
10	CVE-2017-0528	264		Exec Code Bypass	2017-03-07	2017-07-17	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
<p>An elevation of privilege vulnerability in the kernel security subsystem could enable a local malicious application to execute code in the context of a privileged process. This issue is rated as High because it is a general bypass for a kernel level defense in depth or exploit mitigation technology. Product: Android. Versions: Kernel-3.18. Android ID: A-33351919.</p>														

Kernel vulnerabilities

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2018-20961	415		DoS	2019-08-07	2019-08-27	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p>In the Linux kernel before 4.16.4, a double free vulnerability in the <code>f_midi_set_alt</code> function of <code>drivers/usb/gadget/function/f_midi.c</code> in the <code>f_midi</code> driver may allow attackers to cause a denial of service or possibly have unspecified other impact.</p>														
2	CVE-2019-10125	94			2019-03-27	2019-06-14	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p>An issue was discovered in <code>aio_poll()</code> in <code>fs/aio.c</code> in the Linux kernel through 5.0.4. A file may be released by <code>aio_poll_wake()</code> if an expected event is triggered immediately (e.g., by the close of a pair of pipes) after the return of <code>vfs_poll()</code>, and this will cause a use-after-free.</p>														
3	CVE-2019-11683	399		DoS Mem. Corr.	2019-05-02	2019-06-14	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p><code>udp_gro_receive_segment</code> in <code>net/ipv4/udp_offload.c</code> in the Linux kernel 5.x before 5.0.13 allows remote attackers to cause a denial of service (slab-out-of-bounds memory corruption) or possibly have unspecified other impact via UDP packets with a 0 payload, because of mishandling of padded packets, aka the "GRO packet of death" issue.</p>														
4	CVE-2019-11811	416			2019-05-07	2019-05-31	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p>An issue was discovered in the Linux kernel before 5.0.4. There is a use-after-free upon attempted read access to <code>/proc/ioprots</code> after the <code>ipmi_si</code> module is removed, related to <code>drivers/char/ipmi/ipmi_si_intf.c</code>, <code>drivers/char/ipmi/ipmi_si_mem_io.c</code>, and <code>drivers/char/ipmi/ipmi_si_port_io.c</code>.</p>														
5	CVE-2019-15292	416			2019-08-21	2019-09-02	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p>An issue was discovered in the Linux kernel before 5.0.9. There is a use-after-free in <code>atalk_proc_exit</code>, related to <code>net/appletalk/atalk_proc.c</code>, <code>net/appletalk/ddp.c</code>, and <code>net/appletalk/sysctl_net_atalk.c</code>.</p>														
6	CVE-2019-15504	415			2019-08-23	2019-09-04	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p><code>drivers/net/wireless/rsi/rsi_91x_usb.c</code> in the Linux kernel through 5.2.9 has a Double Free via crafted USB device traffic (which may be remote via usbip or usbredir).</p>														
7	CVE-2019-15505	125			2019-08-23	2019-09-04	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
<p><code>drivers/media/usb/dvb-usb/technisat-usb2.c</code> in the Linux kernel through 5.2.9 has an out-of-bounds read via crafted USB device traffic (which may be remote via usbip or usbredir).</p>														
8	CVE-2019-15926	125			2019-09-04	2019-09-14	9.4	None	Remote	Low	Not required	Complete	None	Complete
<p>An issue was discovered in the Linux kernel before 5.2.3. Out of bounds access exists in the functions <code>ath6kl_wmi_pstream_timeout_event_rx</code> and <code>ath6kl_wmi_cac_event_rx</code> in the file <code>drivers/net/wireless/ath/ath6kl/wmi.c</code>.</p>														
9	CVE-2018-20836	416			2019-05-07	2019-05-08	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
<p>An issue was discovered in the Linux kernel before 4.20. There is a race condition in <code>smp_task_timedout()</code> and <code>smp_task_done()</code> in <code>drivers/scsi/libsas/sas_expander.c</code>, leading to a use-after-free.</p>														
10	CVE-2019-11815	362			2019-05-08	2019-06-07	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
<p>An issue was discovered in <code>rds_tcp_kill_sock</code> in <code>net/rds/tcp.c</code> in the Linux kernel before 5.0.8. There is a race condition leading to a use-after-free, related to net namespace cleanup.</p>														

Kernel Security Research is Active

Papers from USENIX Security 2023

- [PhyAuth: Physical-Layer Message Authentication for ZigBee Networks](#)
- [Auditory Eyesight: Demystifying \$\mu\$ s-Precision Keystroke Tracking Attacks on Unconstrained Keyboard Inputs](#)
- [Improving Logging to Reduce Permission Over-Granting Mistakes](#)
- [Know Your Cybercriminal: Evaluating Attacker Preferences by Measuring Profile Sales on an Active, Leading Criminal Market for User Impersonation at Scale](#)

Kernel Security is also “rapidly” changing

Rust was added to Linux v6.1

- Compiles to machine code via rustc
- Provides stronger memory safety guarantees
- Performs comparable to C and C++

Aka, a lot of the most basic attacks may change

Users

Unix is **user-centric**

- no roles

Running code is **always linked** to a certain identity

- **security checks** and **access control** decisions are based on user identity

Users

User

- identified by username (**UID**), group name (**GID**)

```
amgaweda amgaweda 4.0K Jan 29 21:04 .  
amgaweda amgaweda 4.0K Jan 29 21:03 ..  
amgaweda amgaweda 0 Jan 29 21:04 example.txt
```

Users

User

- identified by username (**UID**), group name (**GID**)

```
amgaweda amgaweda 4.0K Jan 29 21:04 .  
amgaweda amgaweda 4.0K Jan 29 21:03 ..  
amgaweda amgaweda 0 Jan 29 21:04 example.txt
```

- typically authenticated by password (stored encrypted)

```
sudo cat /etc/shadow
```

```
...
```

```
amgaweda:$y$notOnYourLifeBubYoullNeverGuessBubbles:0:99999:7:::
```

Users

User

- identified by username (**UID**), group name (**GID**)
- typically authenticated by password (stored encrypted)

User root

```
root root 4.0K Apr 18 2022 boot
```

- superuser, system administrator
- special privileges (access resources, modify OS)
- **cannot decrypt user passwords**

Process Management

Process (PID)

- implements user-activity
- entity that executes a given piece of code
- has its own execution stack, memory pages, and file descriptors table
- separated from other processes using the virtual memory abstraction



htop

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1	root	20	0	2456	1864	1756	S	0.0	0.0	0:00.01	/init
4	root	20	0	2456	932	896	S	0.0	0.0	0:00.00	plan9 --
9	amgaweda	20	0	6180	5156	3396	S	0.0	0.0	0:00.11	-bash
163	amgaweda	20	0	5364	3780	3116	R	0.0	0.0	0:00.01	htop

Process Management

Thread

- separate stack and program counter
- share memory pages and file descriptor table
- processes are also executed through threads and have their own thread ids (**LWP**) and count (**NLWP**)

```
$ ps -eLf
```

UID	PID	PPID	LWP	C	NLWP	STIME	TTY	TIME	CMD
root	1	0	1	0	2	21:02	hvc0	00:00:00	/init
amgaweda	9	8	9	0	1	21:02	pts/0	00:00:00	-bash
amgaweda	164	9	164	0	1	21:24	pts/0	00:00:00	ps -eLf



Process Management

Process Attributes

- process ID (**PID**)
 - uniquely identified process
- user ID (**UID**)
 - ID of owner of process
- effective user ID (**EUID**)
 - ID used for permission checks (e.g., to access resources)
- saved user ID (**SUID**)
 - to temporarily drop and restore privileges
- lots of management information
 - scheduling, memory management, resource management

Process Management

Switching between IDs

- uid-setting system calls
- `int setuid(uid_t uid)`
- `int seteuid(uid_t uid)`
- `int setresuid(uid_t ruid, uid_t euid, uid_t suid)`

Can be tricky

- [POSIX 1003.1](#):
If the process has appropriate privileges, the `setuid(newuid)` function sets the real user ID, effective user ID, and the [saved user ID] to `newuid`.
- what are appropriate privileges?
Solaris: `EUID = 0`; FreeBSD: `newuid = EUID`; Linux: `SETUID` capability

Sudo Change Time

- **user** logs in
 - their **UID** is set to a non-**root** value, indicating they have regular user permissions

Sudo Change Time

- **user** logs in
 - their **UID** is set to a non-**root** value, indicating they have regular user permissions
- **user** runs **date** to change the system time
 - Doing this requires escalated privileges (**root**)
 - **date** is executed but the kernel checks the **EUID** of the process to see if it matches the root's **UID**
 - Since it doesn't, the process is halted

Sudo Change Time

- **user** logs in
 - their **UID** is set to a non-**root** value, indicating they have regular user permissions
- **user** runs **date** to change the system time
 - Doing this requires escalated privileges (**root**)
 - **date** is executed but the kernel checks the **EUID** of the process to see if it matches the root's **UID**
 - Since it doesn't, the process is halted
- **user** runs **sudo date**
 - **sudo** elevates the **EUID** of **date** to **root** temporarily, allowing it to change the time

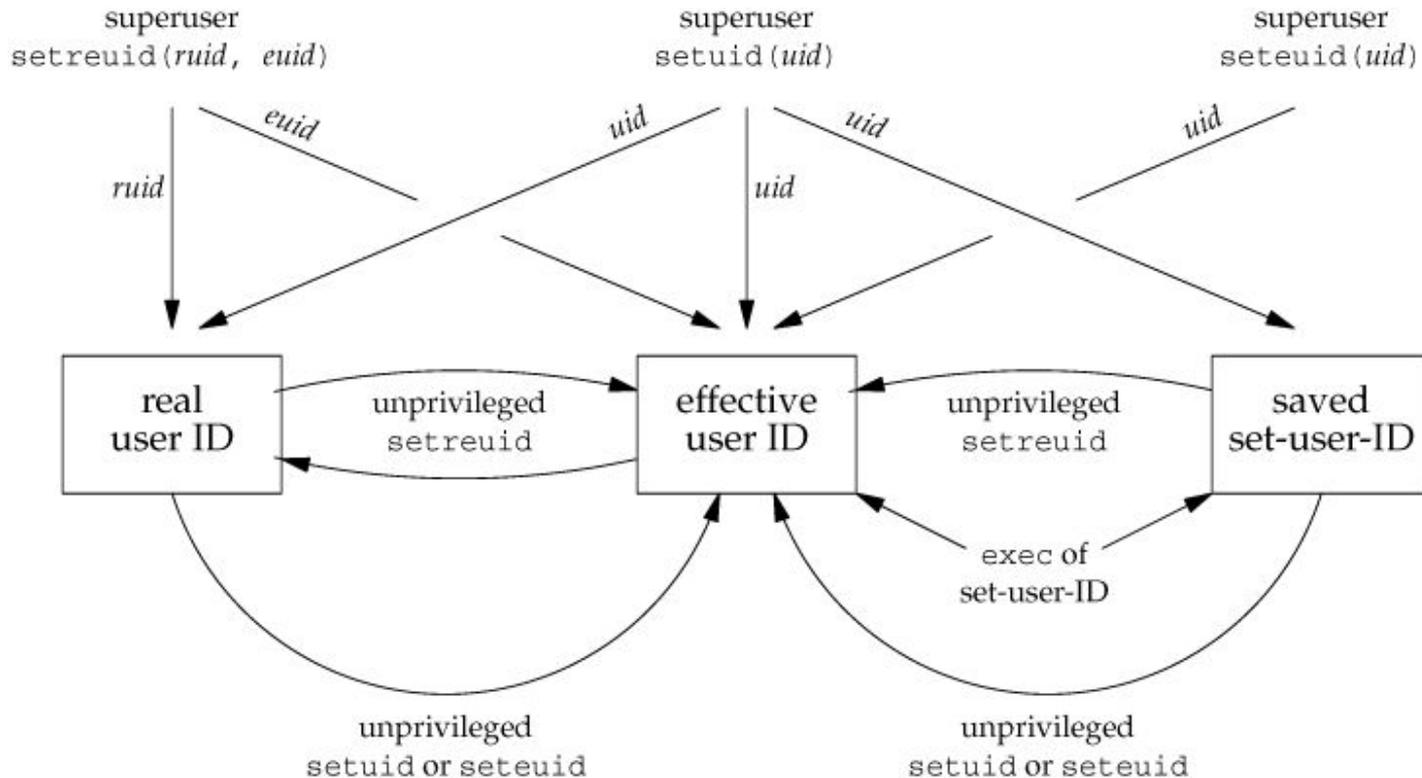
Obligatory XKCD



Obligatory alt-text:

<https://xkcd.com/838/>

Summary of all the Functions that Set the User IDs



Early Example of Privilege Escalation

[Bug in sendmail 8.10.1:](#)

- call to `setuid(getuid())` to clear privileges (effective **UID** is **root**)
- on Linux, attacker could clear **SETUID** capability
- call clears **EUID**, but **SUID** remains root

Further reading

[Setuid Demystified](#), Hao Chen, David Wagner, and Drew Dean
11th USENIX Security Symposium, 2002

User Authentication

How does a process get a user ID?

User Authentication

How does a process get a user ID?

Authentication

User Authentication

Passwords

- user passwords are used as keys for crypt() function
- uses SHA-512
- 8-byte “salt”
 - chosen from date, not secret
 - prevent same passwords to map onto same string
 - make dictionary attacks more difficult

```
sudo cat /etc/shadow  
kali:$y$j9T$1R7REZ4XgU56yXN19PFiN/$oI3B/0eQGx0oTb7opQ.azBM0gG2IM0neRj4MN3HCqQ.:19331:0:99999:7:::
```

User

SHA-512 encryption of "kali"

More on salting passwords in our Web Security lectures

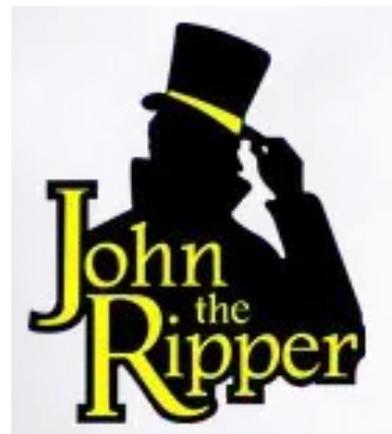
User Authentication

Password Cracking

- dictionary attacks (try common passwords)
- rainbow tables (efficiently try common passwords)
- simple brute force (inefficiently try all passwords)

Password Crackers

- Crack
- JohnTheRipper



User Authentication

Shadow passwords

- password file is needed by many applications to map user ID to user names
- encrypted passwords are not

```
kali:$y$j9T$1R7REZ4XgU56yXN19PFiN/$oI3B/OeQGx0oTb7opQ.azBM0gG2IM0neRj4MN3HCqQ.:19331:0:99999:7:::
```

/etc/shadow

- holds encrypted passwords
 - account information
 - last change date (**19331**)
 - minimum change frequency (**0, 99999**)
 - number of days before expiration (**7**)
 - readable only by superuser and privileged programs
 - ~~SHA-512~~ [yescrypt](#) hashed passwords (default on Ubuntu) to slow down guessing
- 

User Authentication

Shadow passwords

- a number of other encryption / hashing algorithms were proposed
- blowfish, SHA-1, ...

Other authentication means possible

- Linux PAM (pluggable authentication modules)
- Kerberos
- Active directory (Windows)

Group Model

Users belong to one or more **groups**

- primary group (stored in `/etc/passwd`)
- additional groups (stored in `/etc/group`)
- become group member with `newgrp`
- can also to set group password (none by default)

```
/etc/group (groupname : password : group id : additional users)
root:x:0:root
bin:x:1:root,bin,daemon
users:x:100:akprav
```

Special group `wheel/sudo` (like on Ubuntu)

- protect root account by limiting user accounts that can perform `su`



CSC 405

Linux Security

Aleksandr Nahapetyan
anahape@ncsu.edu

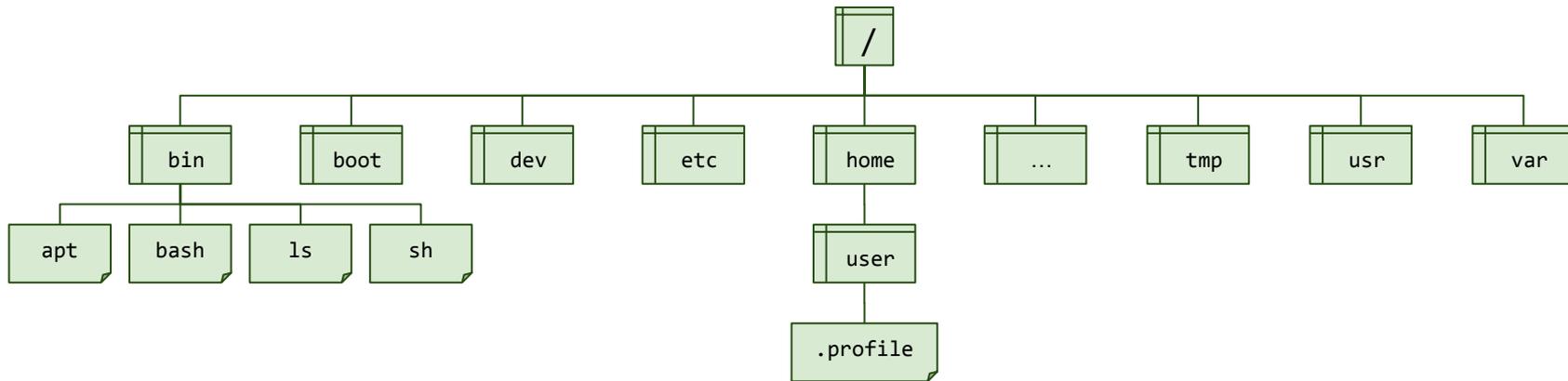
(Slides adapted from Dr. Kapravelos)

In-class practice

- Difficulty: Medium
- There is a server hosted **152.14.199.163** on port **8089**
- The server reads in 80 bytes of shellcode and executes it.
 - Reminder: 0x90 is the opcode for NOP (no-operation)
- Read the flag @ /flag
- Want to automate it?
 - Try out [pwntools!](#)
- How do you connect to the server: [Netcat!](#)

File System

- File Hierarchy Tree - primary repository of information
 - directories contain file system objects (FSO)



- File system object
 - files, directories, symbolic links (shortcuts), sockets, device files
 - referenced by **inode** (index node)

Denial of Service through Inodes

```
#!/bin/bash

# Directory to create files in
target_dir="/tmp/exhaust_inodes"
mkdir -p $target_dir

# Loop to create one million small files
for i in {1..1000000}; do
    # Create a small file with a unique name, exhausting 1 inode
    echo "This is file $i" > "$target_dir/file_$i.txt"
done

# Wait for user input to keep files in place for inspection
# Could DOS processes waiting to creating files on the system
# if the script exhausts all available inodes, even if there
# is still disk space on the drive
read -p "Press any key to delete files and clean up..." -n 1 -r

# Clean up: Remove files and directory
rm -rf $target_dir
echo "Cleanup complete."
```

Do exercise **caution**, this is one of those "attack" scripts

We aren't responsible if you break your machine

`df -i` to see how many inodes your system has

File Permissions

Access Control

- permission bits
- `chmod`, `chown`, `chgrp`, `umask`
- permission structure:

```

      -      rwX      rwX      rwX      -
(file type) (user)   (group) (other) (sticky)

```

Type	r	w	x	s	t
File	read access	write access	execute	suid / sgid inherit id	sticky bit
Directory	list files	insert and remove files	stat / execute files, chdir	new files have dir-gid	files/dirs only delete-able by owner

File Permissions

Access Control

- permission bits
- `chmod`, `chown`, `chgrp`, `umask`
- permission structure:

-
rwX
rwX
rwX
 (file type) (user) (group) (other)

s inherits the permissions of the binary owner

When you execute `passwd`, it inherits root permissions

Type	r	w	x	s	t
File	read access	write access	execute	suid / sgid inherit id	sticky bit
Directory	list				

Find files w/ root setuid

```
find / -type f -perm /4000 -exec stat -c "%U %n" {} + | grep root
```

Find available binaries on the system

```
dpkg -get-resources
```

Sticky bit

No effect on files (on Linux)

When used on a directory, all the files in that directory will be modifiable **only by their owners**

What's a very common directory with sticky bit?

Sticky bit

No effect on files (on Linux)

When used on a directory, all the files in that directory will be modifiable **only by their owners**

What's a very common directory with sticky bit?

```
$ ls -ld /tmp
drwxrwxrwt 26 root root 69632 Sep  7 15:24 /tmp

$ ls -l
-rw-rw-r-- 1 username username 0 Sep  7 15:29 test

$ chmod +t test; ls -l
-rw-rw-r-t 1 username username 0 Sep  7 15:29 test
```

SUID Programs

Each process has real and effective user / group ID

- usually identical
- real IDs
 - determined by current user
 - authentication (login, **su**)
- effective IDs
 - determine the “rights” of a process
 - system calls (e.g., **setuid()**)
- **suid / sgid** bits
 - to start process with effective ID different from real ID
 - attractive target for attacker

Never use suid shell scripts (multiplying problems)

- many operating systems ignore the **setuid** attribute when applied to executable shell scripts
- you need to patch the kernel to enable it

File System

Shared resource

- susceptible to [race condition problems](#)

Time-of-Check, Time-of-Use (**TOCTOU**)

- common race condition problem
- problem:
 - Time-Of-Check (t_1): validity of assumption **A** on entity **E** is checked
 - Time-Of-Use (t_2): assuming **A** is still valid, **E** is used
 - Time-Of-Attack (t_3): assumption **A** is invalidated

$$t_1 < t_3 < t_2$$

TOCTOU

- Steps to access a resource
 - obtain reference to resource
 - query resource to obtain characteristics
 - analyze query results
 - if resource is fit, access it
- Often occurs in Unix file system accesses
 - check permissions for a certain file name (e.g., using [access\(2\)](#))
 - open the file, using the file name (e.g., using [fopen\(3\)](#))
 - four levels of indirection (symbolic link - hard link - **inode** - file descriptor)
- Windows uses file handles and includes checks in the API [open](#) call

TOCTOU Example

```
/* access returns 0 on success */
if(!access(file, W_OK)) {
    f = fopen(file, "wb+");
    write_to_file(f);
} else {
    fprintf(stderr, "Permission denied \
                when trying to open %s.\n", file);
}
```

W_OK: Flag meaning test for write permission.
access return value is 0 if the access is permitted

Application checks if a file is safe to write to, if so then writes to it.

TOCTOU Example

```
/* access returns 0 on success */  
if(!access(file, W_OK)) {  
    f = fopen(file, "wb+");  
    write_to_file(f);  
} else {  
    fprintf(stderr, "Permission denied \  
                when trying to open %s.\n", file);  
}
```

W_OK: Flag meaning test for write permission.
access return value is 0 if the access is permitted

Application checks if a file is safe to write to, if so then writes to it.

```
$ touch dummy; ln -s dummy pointer  
$ rm pointer; ln -s /etc/passwd pointer
```

Attack creates symbolic link to **dummy**
Application makes `access()` call on **dummy**
System says **dummy** is okay to write to

TOCTOU Example

```
/* access returns 0 on success */  
if(!access(file, W_OK)) {  
    f = fopen(file, "wb+");  
    write_to_file(f);  
} else {  
    fprintf(stderr, "Permission denied \  
                when trying to open %s.\n", file);  
}
```

W_OK: Flag meaning test for write permission.
access return value is 0 if the access is permitted

Application checks if a file is safe to write to, if so then writes to it.

```
$ touch dummy; ln -s dummy pointer  
$ rm pointer; ln -s /etc/passwd pointer
```

Attack creates symbolic link to **dummy**
Application makes **access()** call on **dummy**
System says **dummy** is okay to write to

Before **fopen()** operation occurs, attacker
deletes the symbolic link on **dummy** and creates it
on **/etc/passwd**

TOCTOU Example

- **setuid** Scripts

- `exec()` system call invokes `seteuid()` call prior to executing program
- program is a script, so command interpreter is loaded first
- program interpreted (with **root** privileges) is invoked on script name
- attacker can replace script content between step 2 and 3

```
#!/bin/bash

# Check if the user has read permissions on sensitive_file
if [ -r "sensitive_file" ]; then
    echo "User has read permissions. Executing privileged operation..."
    # Perform privileged operation
    cat "sensitive_file"
else
    echo "User does not have read permissions. Operation aborted."
fi
```

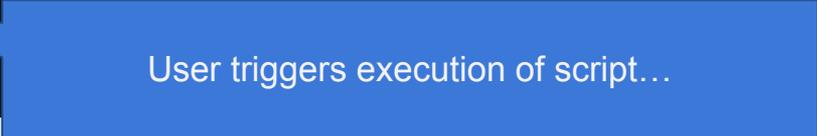
TOCTOU Example

- **setuid** Scripts

- `exec()` system call invokes `seteuid()` call prior to executing program
- program is a script, so command interpreter is loaded first
- program interpreted (with **root** privileges) is invoked on script name
- attacker can replace script content between step 2 and 3

```
#!/bin/bash

# Check if the user has read permissions on sensitive_file
if [ -r "sensitive_file" ]; then
    echo "User has read permissions. Executing privileged operation..."
    # Perform privileged operation
    cat "sensitive_file"
else
    echo "User triggered execution of script... aborted."
fi
```



TOCTOU Example

- **setuid** Scripts

- `exec()` system call invokes `seteuid()` call prior to executing program
- program is a script, so command interpreter is loaded first
- program interpreted (with **root** privileges) is invoked on script name
- attacker can replace script content between step 2 and 3

```
#!/bin/bash

# Check if the user has read permissions on sensitive_file
if [ -r "sensitive_file" ]; then
    echo "User has read permissions. Executing privileged operation..."
    # Perform privileged operation
    cat "sensitive_file"
else
    echo "User does not have read permissions. Exiting."
fi
```

But before execution, attacker creates a symbolic link named `sensitive_file` pointing to `/etc/passwd`

```
$ ln -s /etc/passwd sensitive_file
```

TOCTOU Example

- Directory operations
 - `rm` can remove directory trees, traverses directories depth-first
 - issues `chdir("..")` to go one level up after removing a directory branch
 - by relocating subdirectory to another directory, arbitrary files can be deleted

```
#!/bin/bash

# Create a temporary file
touch /tmp/example

# Check if the directory exists
if [ -f "/tmp/example" ]; then
    # Prompt the user before removing
    echo "File exists. Are you sure? (y/n)"
    read answer
    if [ "$answer" == "y" ]; then
        # Remove the file
        rm -rf /tmp/example
        echo "File removed."
    else
        echo "File not removed."
    fi
else
    echo "File does not exist."
fi
```

TOCTOU Example

- Directory operations

- `rm -rf /etc` After checking the file exists...
 - 1. Attacker deletes `/tmp/example`
 - 2. Creates a symbolic link `ln -s /etc /tmp/example`
 - 3. Process proceeds to execute `rm -rf /etc`
- `traverse`
- `issue`
- `up a`
- by relocating subdirectory to another directory, arbitrary files can be deleted

```
#!/bin/bash

# Create a temporary file
touch /tmp/example

# Check if the directory exists
if [ -f "/tmp/example" ]; then
    # Prompt the user before removing
    echo "File exists. Are you sure? (y/n)"
    read answer
    if [ "$answer" == "y" ]; then
        # Remove the file
        rm -rf /tmp/example
        echo "File removed."
    else
        echo "File not removed."
    fi
else
    echo "File does not exist."
fi
```

TOCTOU Example

- Temporary files
 - commonly opened in `/tmp` or `/var/tmp`
 - often guessable file names
 - if the attacker can intercept the process between permission check and operation, and the `/tmp` file is trivially named, they may be able to manipulate it

Common Trivial Names:

- `cache.dat`
- `temp_file`
- `data.txt`
- `apache2.pid`
- `sshd.pid`

Temporary Files

- "Secure" procedure for creating temporary files
 - pick a prefix for your filename
 - generate **at least** 64 bits of high-quality randomness
 - base64 encode the random bits
 - concatenate the prefix with the encoded random data
 - set `umask` appropriately (0066 is usually good, readable/writable only by you)
 - use [`fopen\(3\)`](#) to create the file, opening it in the proper mode
 - delete the file immediately using [`unlink\(2\)`](#) (deletes file after you're done with it)
 - perform reads, writes, and seeks on the file as necessary
 - finally, close the file

Prevention

- Immutable bindings
 - rather than using the file's variable, operate on file descriptors ([fstat](#))

```
int main() {  
    ...  
    int fd = open(filename, O_RDONLY);  
    ...  
    struct stat st;  
    fstat(fd, &st)  
    ...  
    if (!S_ISREG(st.st_mode)) { ... }  
    ...  
    printf("File size: %ld bytes\n", st.st_size);  
    close(fd);  
    return 0;  
}
```

Ensures that we're not attempting to work with a special file type (directory, symbolic link)

Prevention

- Use the `O_CREAT` | `O_EXCL` flags to create a new file with [open\(2\)](#)
 - be prepared to have the open call fail

```
int main() {  
    ...  
    int fd = open(filename, O_WRONLY | O_CREAT | O_EXCL, 0066);  
    ...  
    close(fd);  
    return 0;  
}
```

Automatically creates the file if it does not exist and fails if it does

Prevention

Series of papers on the access system call

[Fixing races for fun and profit: how to use access\(2\)](#)

D. Dean and A. Hu

Usenix Security Symposium, 2004

[Fixing races for fun and profit: how to abuse atime](#)

N. Borisov, R. Johnson, N. Sastry, and D. Wagner

Usenix Security Symposium, 2005

[Portably Solving File TOCTTOU Races with Hardness Amplification](#)

D. Tsafir, T. Hertz, D. Wagner, and D. Da Silva

Usenix Conference on File and Storage Technologies (FAST), 2008

Locking

- Ensures exclusive access to a certain resource
- Used to circumvent accidental race conditions
 - advisory locking (processes need to cooperate)
 - not mandatory, therefore not secure
- Often, files are used for locking
 - portable (files can be created nearly everywhere)
 - “stuck” locks can be easily removed
- Simple method
 - create file using the O_EXCL flag

```
struct flock lock;

// Open or create a file
fd = open("example.txt",
         O_RDWR | O_CREAT,
         0666);

// Prepare lock structure
lock.l_type = F_WRLCK; // Write lock
lock.l_whence = SEEK_SET;
lock.l_start = 0;
lock.l_len = 0; // Lock entire file

// Try to acquire the lock
if (fcntl(fd, F_SETLK, &lock) == -1) {
    // error
}

// Do some operations

// Unlock the file
lock.l_type = F_UNLCK;
```

Shell

- Shell
 - one of the core Unix application
 - both a command language and programming language
 - provides an interface to the Unix operating system
 - rich features such as control-flow primitives, parameter passing, variables, and string substitution
 - communication between shell and spawned programs via redirection and pipes
 - different flavors
 - **bash** and **sh**, **tcsh** and **csh**, **ksh**, **zsh**

Shell Attacks

- Environment Variables

- **\$HOME** and **\$PATH** can modify behavior of programs that operate with relative path names

- **\$IFS** – internal field separator

- used to parse tokens
- usually set to `[\t\n]` but can be changed to `"/"`
- `"/bin/ls"` is parsed as `"bin ls"` calling bin locally
- IFS now only used to split expanded variables

- preserve attack (`/usr/lib/preserve` is **SUID**)

- called `"/bin/mail"` when `vi` crashes to preserve file
- change IFS, create `bin` as link to `/bin/sh` then `kill vi`

Used to be super common but IFS has been removed since actual use is rare

```
$ IFS=';' ./vulnerable_script.sh
Enter a filename:
/tmp/secret_file; ls /
```

```
IFS=$'\n'
ln -s /bin/sh
    /usr/lib/preserve/bin
vi /usr/lib/preserve/some_file
```

Shell Attacks

- Control and escape characters
 - can be injected into command string
 - modify or extend shell behavior
 - user input used for shell commands has to be rigorously sanitized
 - easy to make mistakes
 - classic examples are ';' and '&'
- Applications that are invoked via shell can be targets as well
 - increased vulnerability surface
- Restricted shell
 - invoked with `-r` or `rbash`
 - more controlled environment

```
find /some_path -name "filename.txt; ls /"
```

Shell Attacks

- `system(char *cmd)`
 - function called by programs to execute other commands
 - invokes shell
 - executes string argument by calling `/bin/sh -c string`
 - makes binary program vulnerable to shell attacks
 - especially when user input is utilized
- `popen(char *cmd, char *type)`
 - forks a process, opens a pipe and invokes shell for cmd

explainshell.com

```
rm(1) -rf /
```

remove files or directories

-r, -R, --recursive
remove directories and their contents recursively

-f, --force
ignore nonexistent files, never prompt

Remove (unlink) the FILE(s).

File Descriptor Attacks

- SUID program (everyone uses, root permissions) opens file
- forks external process
 - sometimes under user control
- **on-execute** flag
 - if **close-on-exec** flag is not set, then new process inherits file descriptor
 - malicious attacker might exploit such weakness
- Linux Perl 5.6.0
 - **getpwuid()** leaves **/etc/shadow** opened (June 2002)
 - could attack this with Apache or `mod_perl`

Resource Limits

- File system limits
 - quotas
 - restrict storage blocks and number of inodes
 - hard limit
 - can never be exceeded (operation fails)
 - soft limit
 - can be exceeded temporarily
 - can be defined per mount-point
 - defend against resource exhaustion (denial of service)
- Process resource limits
 - number of child processes, open file descriptors

```
#!/bin/bash

# Limit CPU time to 10 seconds
ulimit -t 10

# Limit virtual memory to 100 MB
ulimit -v 100000

# Infinite loop consumes CPU and memory
while true; do
    :
done
```

Signals

Signal

- asynchronous notification; simple form of interrupt
- can happen anywhere for process in user space
- used to deliver segmentation faults, reload commands, ...
- kill command

Signal handling

- process can install signal handlers
- when no handler is present, default behavior is used
 - ignore or kill process
- possible to catch all signals except SIGKILL (-9)

```
#!/bin/bash

# Start the vulnerable script in the background
./vulnerable_script.sh &

# Obtain the PID of the vulnerable script
pid=$!

# Wait for a few seconds to ensure the
vulnerable script is running
sleep 2

# Send a SIGINT signal to the vulnerable script
echo "Sending SIGINT signal to PID $pid..."
kill -2 $pid
```

Signals

- Security issues
 - code has to be re-entrant (code running, signal jump, then come back)
 - atomic modifications
 - no global data structures
 - race conditions
 - unsafe library calls, system calls
 - examples
 - wu-ftpd 2001, sendmail 2001 + 2006, stunnel 2003, [ssh 2006](#)
- Secure signals
 - write handler as simple as possible
 - block signals in handler

Shared Libraries

- Library
 - collection of object files
 - included into (linked) program as needed
 - code reuse
- Shared library
 - multiple processes share a single library copy
 - save disk space (program size is reduced)
 - save memory space (only a single copy in memory)
 - used by virtually all Unix applications (at least libc.so)
 - check binaries with **ldd**

Shared Libraries

- Static shared library
 - address binding at link-time
 - not very flexible when library changes
 - code is fast
- Dynamic shared library
 - address binding at load-time
 - uses procedure linkage table (PLT) and global offset table (GOT) to hold references to code
 - code is slower (redirection)
 - loading is slow (binding has to be done at run-time)
 - classic .so or .dll libraries
- PLT and GOT entries are very popular attack targets
 - buffer overflows

<https://www.technovelty.org/linux/plt-and-got-the-key-to-code-sharing-and-dynamic-libraries.html>

Shared Libraries

- Management
 - stored in special directories (listed in /etc/ld.so.conf)
 - manage cache with ldconfig
- Preload
 - override (substitute) with other version
 - use /etc/ld.so.preload
 - can also use environment variables for override
 - possible security hazard
 - [Wanna hack a game?](#)



Advanced Security Features

- Address space protection
 - address space layout randomization (ASLR)
 - non-executable stack (based on NX bit or PAX patches)
- Mandatory access control extensions
 - SELinux/AppArmor
 - role-based access control extensions
 - capability support
- Miscellaneous improvements
 - hardened chroot jails
 - better auditing
- <https://wiki.ubuntu.com/Security/Features>

in-class lab

Go to <https://pwn.college/linux-luminarium/>
Solve 12 challenges (they are really easy :))