

OS & Application Security

Michael A. Specter

Assistant Professor Georgia Tech /// SPDR Lab

What is computer security?

Two definitions

1. Security as systems design (akin to medicine)
2. Security as ecosystem analysis (akin to public health)

Two definitions (For this class)

1. Security as systems design (akin to medicine)

2. Security as ecosystem analysis (akin to public health)

Security as *systems* design.

Security Defined

Security is the design of systems that achieve some property against an active adversary.

Process:

define a system,
define security goals,
and prove that the system provides these goals.



Goals & Threat Model

Goals:

- You shouldn't be able to read my files
- No process should be able to read another process's memory

Threat model:

- Assumptions about what the attacker can do
- E.g. can one guess passwords?

Threat model specifies what's in/out of scope.

Goal defines what you want to achieve.

What could go wrong?

What could go wrong?

1. Bad Goals
2. Bad Threat Models
3. Bad Mechanisms

What could go wrong?
Bad Threat Models

Getting a threat model right can be
subtle.

For example:
I assume the adversary does
not know how my system works.

Kerckhoff's Principle

A system is only secure if the adversary knows everything about the system, and it is still secure.

The enemy knows the system.

Inverse: “Security through obscurity”
Does. Not. Work.



Bad Threat Models:

- Stuxnet
 - Iranian nuclear facility centrifuges were destroyed
 - Used an airgapped network
 - Assumed that malware couldn't jump the airgap
 - ...This assumption was bad, malware was installed via USB
- Computational assumptions
 - Cryptographic primitives may not survive as computation gets better – e.g. DES
 - Cryptography & Quantum Computers: Shor's algorithm breaks RSA

Bad Threat Model Examples: Just Trust Your Hardware

- Farewell dossier
 - During the cold war, Soviets tended to steal US plans / tech. Often lacked technical knowhow
 - Problem: the US had a defector, “Farewell”, in that group.

Gus Weiss, WH’s NSC Director of International Econ:

“Contrived computer chips found their way into Soviet military equipment, flawed turbines were installed on a gas pipeline, and defective plans disrupted the output of chemical plants and a tractor factory. The Pentagon introduced misleading information pertinent to stealth aircraft, space defense, and tactical aircraft. The Soviet Space Shuttle was a rejected NASA design.”

Bad Threat Model Examples: Just Trust Updates

- Apple v. FBI (now, I guess, the UK)
- NPM's Left-Pad Incident
- Chrome extension bought by malware authors
 - <https://arstechnica.com/security/2014/01/malware-vendors-buy-chrome-extensions-to-send-adware-filled-updates/>

Bad Threat Model Examples: Just Trust Your Software

- 2015: Chinese iOS Developers Mirrored xcode
 - Took too long to download from the U.S.
 - Some of these mirrors had malware'd versions of xcode
 - Injected malware into apps
 - (look up: xcodeGhost)

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

What *e/se* could go wrong?
Bad Mechanisms

Let's talk about Memory Safety!

```
> cat test.c
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerable_function(argv[1]);
    return 0;
}
```

What's wrong with this code?

Let's talk about Memory Safety!

```
> cat test.c
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerable_function(argv[1]);
    return 0;
}
```

What happens when you
make a call to `vulnerable_function`?

Let's say the input is '1234567890'

Let's talk about Memory Safety!

```
> cat test.c
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerable_function(argv[1]);
    return 0;
}
```



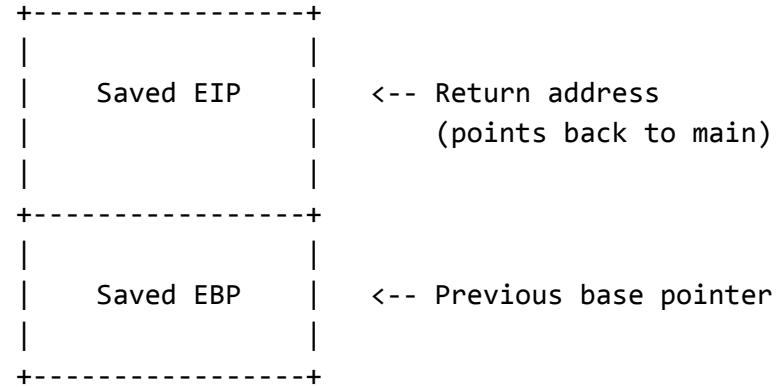
<-- Previous base pointer

Let's talk about Memory Safety!

```
> cat test.c
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerable_function(argv[1]);
    return 0;
}
```

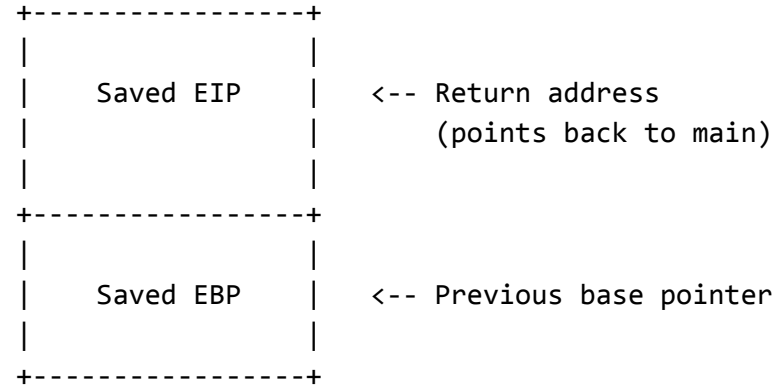


Let's talk about Memory Safety!

```
> cat test.c
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerable_function(argv[1]);
    return 0;
}
```

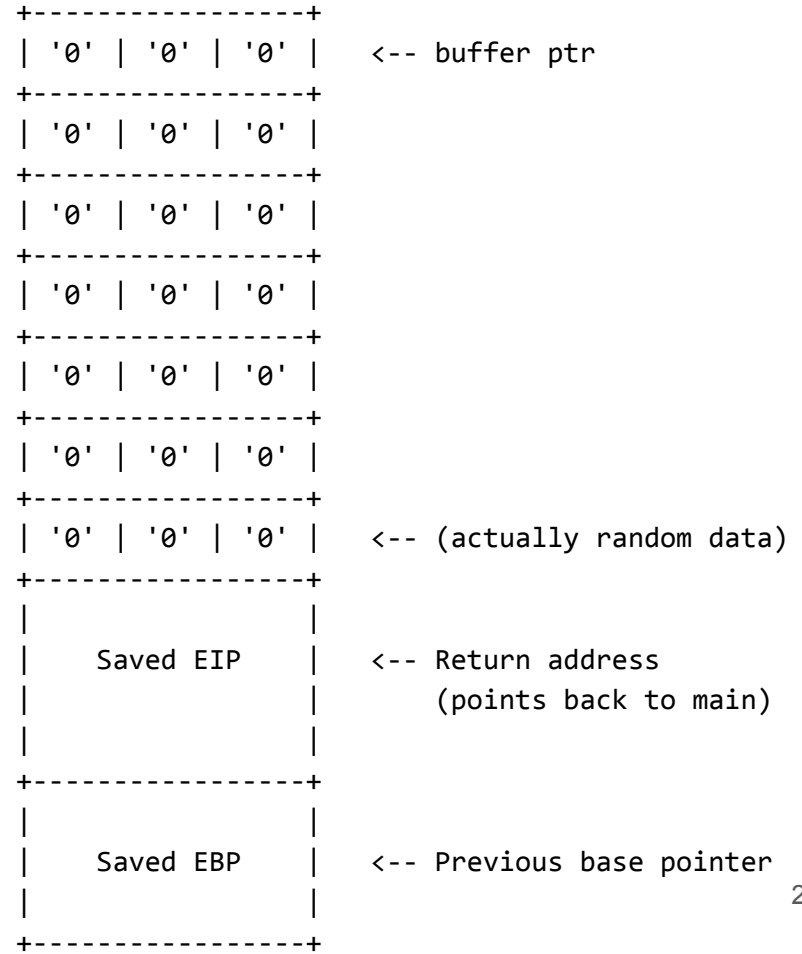


Let's talk about Memory Safety!

```
> cat test.c
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerable_function(argv[1]);
    return 0;
}
```

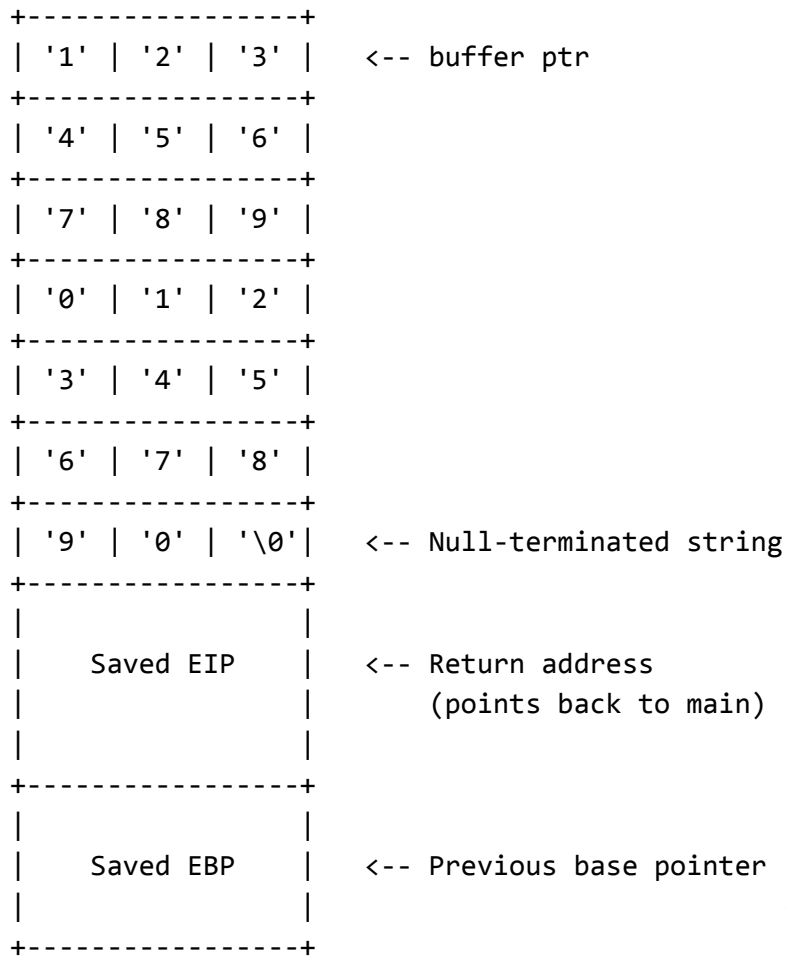


Let's talk about Memory Safety!

```
> cat test.c
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerable_function(argv[1]);
    return 0;
}
```



Let's talk about Memory Safety!

```
> cat test.c
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerable_function(argv[1]);
    return 0;
}
```

What happens when you
make a call to `vulnerable_function`?

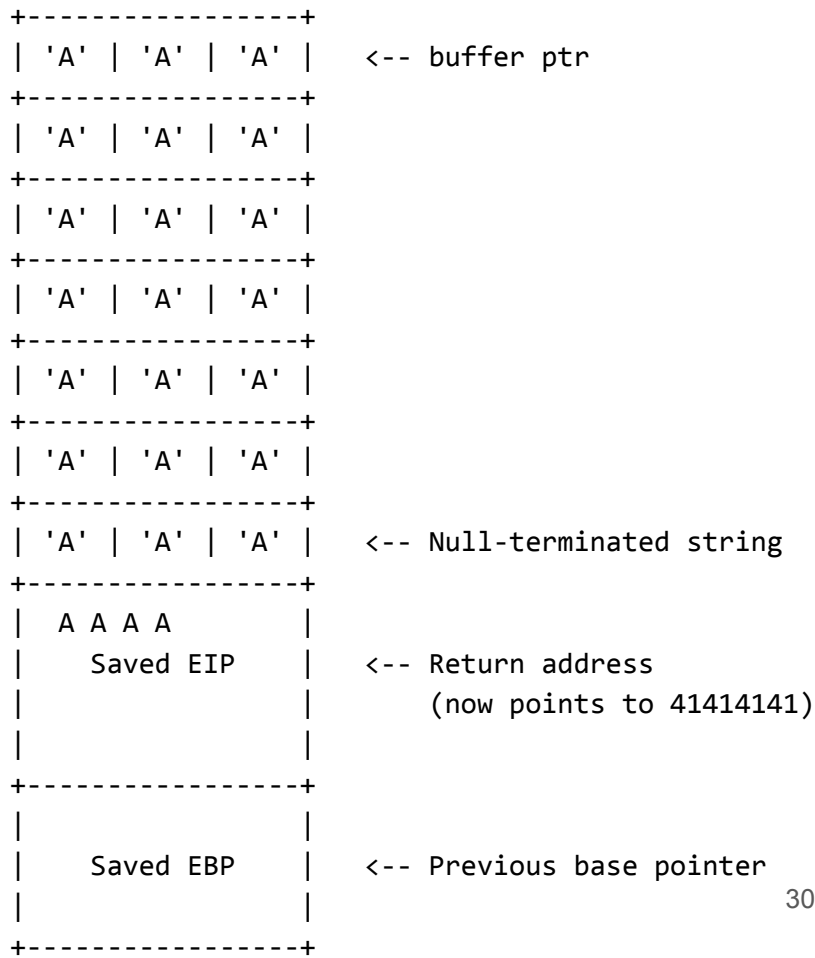
Now let's say the input is 'A' * 100

Let's talk about Memory Safety!

```
> cat test.c
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerable_function(argv[1]);
    return 0;
}
```



Once I control EIP (RIP)

In this case, A's will return to hex 41414141, which isn't useful, but:

- Return to code I wrote on the stack
- Return to known functions (e.g. known addresses of things in libc)
- Game is generally over =(

Generally called memory safety issues

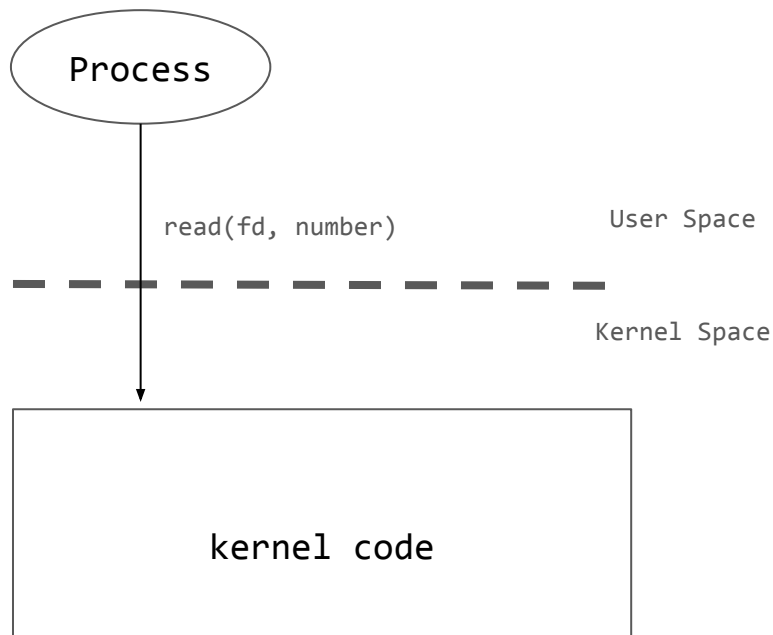
- Buffer overreads, writes, etc
- Use-after-free
- Happens both on the stack & heap
- For more (& a history) check out:

Szekeres et al, SoK: Eternal War in Memory, IEEE S&P 2013

Relevant to Kernel Devs: Syscall security

A *syscall* is how userspace programs call into the kernel

E.g. a *process* wants to read a file



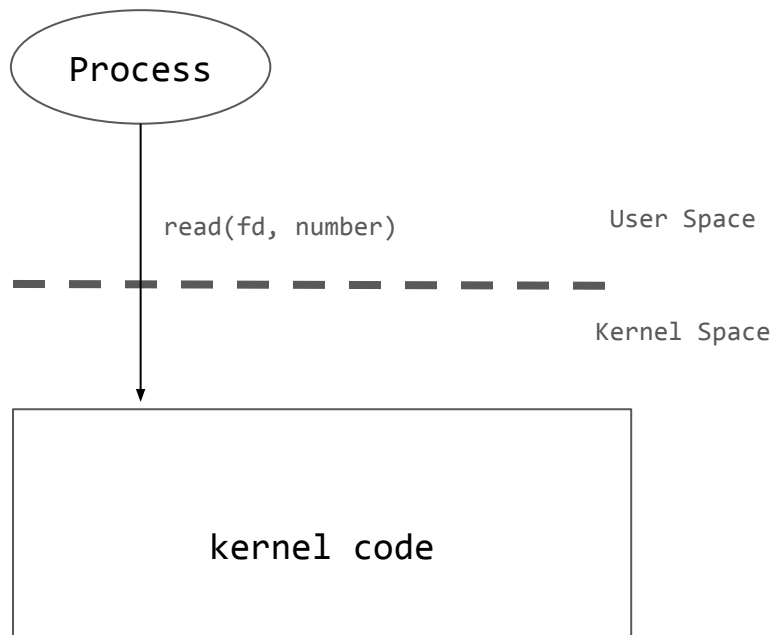
Relevant to Kernel Devs: Syscall security

There are ~456 syscalls in Linux

<https://filippo.io/linux-syscall-table/>

Linux kernel has ~40M lines of code.

Device drivers are written by OEMs
...And, generally, not written well.



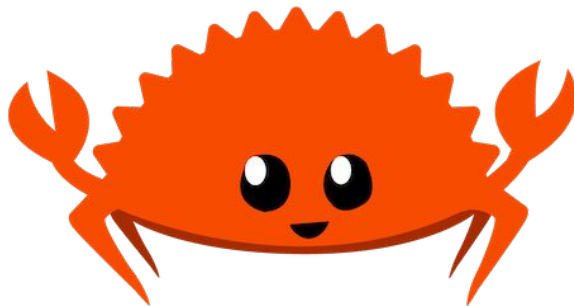
Current, unfixed bugs in linux

open (1402):							
Title	Repro	Cause bisect	Fix bisect	Count	Last	Reported	Discussions
WARNING: bad unlock balance in __rtnl_unlock <code>[net]</code>	C	done		17	12h54m	2h34m	PATCH [2h31m]
WARNING in taprio_get_start_time <code>[net]</code>				1	9h40m	5h29m	
INFO: trying to register non-static key in sock_def_readable <code>[net]</code> ...	syz			1	8h51m	5h29m	
BUG: corrupted list in __sk_destruct(2) <code>[net]</code> <code>[ext4]</code>	syz			1	19h50m	5h29m	
BUG: sleeping function called from invalid context in __bio_release_...				2	4d08h	8h22m	
KASAN: slab-out-of-bounds Read in validate_bset_keys <code>[bcachefs]</code>	C			5	1d16h	1d00h	
WARNING: bad unlock balance in __mm_populate <code>[xfs]</code> <code>[mm]</code>	C			13	1h49m	1d22h	☹ 4 [22m]
KASAN: slab-use-after-free Read in nf_tables_trans_destroy_work(2...	syz			1	2d17h	2d03h	PATCH [3h35m]
KASAN: global-out-of-bounds Read in get_mem_cgroup_from_mm ...	C			1	2d16h	2d03h	
kernel BUG in try_to_unmap_one <code>[mm]</code>				1	6d16h	2d16h	☹ 3 [22h37m]
WARNING in exit_itimers <code>[kernel]</code>				1	7d02h	3d02h	
BUG: unable to handle kernel paging request in io_schedule <code>[ntfs3]</code>	C			4	7d00h	3d07h	
INFO: task hung in invalidate_inode_pages2_range(3) <code>[bcachefs]</code>	C	done		14	21h53m	4d07h	
kernel BUG in filemap_fault <code>[mm]</code>				1	8d08h	4d08h	
INFO: task hung in pipe_release(6) <code>[bcachefs]</code>	syz	done		4	6d22h	4d19h	
KASAN: slab-use-after-free Read in task_work_run <code>[kernel]</code>	syz	error		2	8d19h	4d22h	☹ 1 [3d15h]
INFO: reu detected stall in call_timer_fn(5) <code>[net]</code>	C			1	9d11h	5d11h	
WARNING: bad unlock balance in traverse <code>[fs]</code>	C			13	1d01h	5d17h	
possible deadlock in ipv6_sock_ac_close(4) <code>[net]</code>				2	1d23h	5d23h	
KMSAN: uninit-value in cfg80211_tx_mlme_mgmt <code>[wireless]</code>				1	9d23h	5d23h	PATCH [5d06h]
WARNING in ieee80211_free_keys <code>[wireless]</code>				2	7d17h	6d02h	
WARNING in ieee80211_set_disassoc <code>[wireless]</code>				1	10d	6d04h	
WARNING in get_dump_page <code>[mm]</code> <code>[ext4]</code>	C	done		5	6d23h	6d04h	
WARNING in hcj_send_cmd(2) <code>[bluetooth]</code>				1	10d	6d09h	
general protection fault in native_tss_update_io_bitmap <code>[kernel]</code>	C	error		7	6d23h	6d09h	PATCH [6d00h]
possible deadlock in l2cap_info_timeout <code>[bluetooth]</code>				1	10d	6d19h	
KCSAN: data-race in netfs_advance_write/netfs_write_collection_w...				7	3d00h	7d04h	
KASAN: slab-out-of-bounds Read in fbcon_prepare_logo <code>[fbdev]</code>	syz			7	3d02h	7d08h	
possible deadlock in crypto_exit_scomp_ops_async <code>[crypto]</code>				1	11d	7d17h	PATCH [5d13h]
KASAN: slab-out-of-bounds Read in hfsplus_bmap_alloc <code>[hfs]</code>	C			1	7d22h	7d21h	
INFO: task hung in ...	syz	done		1	8d07h	7d23h	

<https://syzkaller.appspot.com/upstream>

Solutions:

- Bounds checking (e.g. strncpy vs strcpy)
- compiler safety helpers ASLR, DEP, and other
- Some OS Primitives can be helpful
- The real solution is memory safety (e.g. rust, not c)





FEBRUARY 26, 2024

PRESS RELEASE: Future Software Should Be Memory Safe



► ONCD ► BRIEFING ROOM ► PRESS RELEASE

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report [here](#)

WASHINGTON – Today, the White House Office of the National Cyber Director (ONCD) released a report calling on the technical community to proactively reduce the attack surface in cyberspace. ONCD makes the case that technology manufacturers can prevent entire classes of vulnerabilities from entering the digital ecosystem by adopting memory safe programming languages. ONCD is also encouraging the research community to address the problem of software measurability to enable the development of better diagnostics that measure cybersecurity quality.

Passwords!

How should you store a password on-disk?

Solution #1:

Just store it in plaintext!

What could go wrong?

How should you store a password on-disk?

Solution #2: Let's talk about hashes!

$$h: \{0,1\}^* \rightarrow \{0,1\}^d$$

In words: Hash function h maps a bit string of any size to a bit string of length d .

Goals: Efficient, deterministic, random, public

Examples:

MD5	$d=128$
SHA 256	$d=256$
SHA3-256	$d=256$
SHAKE	$d=\text{variable}$

How should you store a password on-disk?

Solution #2: Let's talk about hashes!

$$h: \{0,1\}^* \rightarrow \{0,1\}^d$$

Other Goals:

Collision Resistance:

Given x , cannot find x' s.t. $h(x) = h(x')$ (assuming $x \neq x'$)

One-wayness:

Given $y \leftarrow h(x)$, where x is random, cannot find x' s.t. $h(x') = y$

How should you store a password on-disk?

Just store the password as $h(\text{password})$!

What could go wrong?

Brute force attack:

1. Create a database of likely passwords $P=\{p_1 \dots p_n\}$
2. Calculate $\{(h(p_1), p_1) \dots (h(p_n), p_n)\}$
3. You can now recover most “likely” passwords.
4. Works **even if $|P|$ is prohibitively large**: **Rainbow Tables**
 - a. Exploits time/space tradeoff to limit size of $|P|$

WEB / POLICY / BREAKING

Adobe suffers major cyber attack, says data on 2.9 million customers was compromised

By [Chris Welch](#), a reviewer specializing in personal audio and home theater. Since 2011, he has published nearly 6,000 articles, from breaking news and reviews to useful how-tos.

Source [Adobe](#)

Oct 3, 2013, 5:09 PM EDT



Comments (0 New)

TECH

Over 150 million breached records from Adobe hack have surfaced online

By [Chris Welch](#), a reviewer specializing in personal audio and home theater. Since 2011, he has published nearly 6,000 articles, from breaking news and reviews to useful how-tos.

Source [Naked Security \(Sophos blog\)](#)

Nov 7, 2013, 6:08 PM EST



0 Comments (0 New)

ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:

USER PASSWORD	HINT	
4e18acc1ab2762d6	WEATHER VANE SWORD	
4e18acc1ab2762d6		
4e18acc1ab2762d6 a0a2876eb1ea1fca	NAME 1	
8babbb6299e06eb6d	DUH	
8babbb6299e06eb6d a0a2876eb1ea1fca		
8babbb6299e06eb6d 85e9da81a8a78adc	57	
4e18acc1ab2762d6	FAVORITE OF 12 APOSTLES	
1ab29ae86da6e5ca 7a2d6a0a2876eb1e	WITH YOUR OWN HAND YOU HAVE DONE ALL THIS	
a1f9b2b299e7a2b e0dec1e6ab797397	SEXY EARLOBES	
a1f9b2b299e7a2b 617ab0277727ad85	BEST TOS EPISODE	
39738b7adb0b8af7 617ab0277727ad85	SUGARLAND	
1ab29ae86da6e5ca	NAME + JERSEY #	
877ab7889d3862b1	ALPHA	
877ab7889d3862b1		
877ab7889d3862b1		
877ab7889d3862b1	OBVIOUS	
877ab7889d3862b1	MICHAEL JACKSON	
38a7c9279c0deb44 9dca0d79d4dec6d5		
38a7c9279c0deb44 9dca0d79d4dec6d5	HE DID THE MASH, HE DID THE PURLOINED	
38a7c9279c0deb44 8a0e545c71e7ef7a 9dca0d79d4dec6d5	FAV. LATER-3 POKEMON	

THE GREATEST CROSSWORD PUZZLE
IN THE HISTORY OF THE WORLD

This is now a real thing.

<https://zed0.co.uk/crossword/>

Adobe Crossword



A crossword based on the Adobe password leak.
Inspired by [xkcd #1286: Encryptic](#)

Password popularity:

1-100

101-200

201-300

301-400

401-500

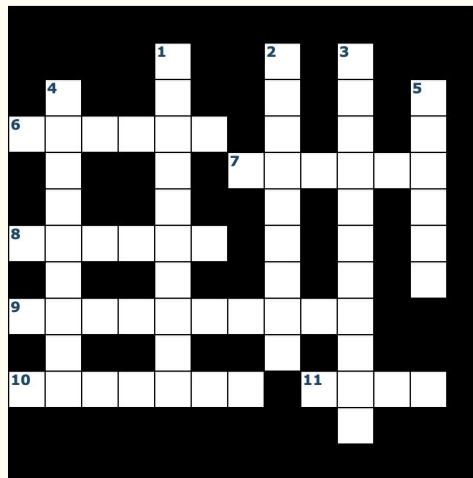
501-600

601-700

701-800

801-900

901-1000



Reveal

Check

Hide

Across

▼ 6: zk8NJgAOqc4=

dog; cat; pet;
dark; dogs name;
Dog; my dog;
black dog; dog
name; dog's
name; darkness;
black cat; sonic;
black; Kitty; horse;
Cat; pets name;
sombra; puppy;
cats name; old
dog; shade; first
dog; pet name;
doggy; hedgehog;
cat's name; bike;
my cat; nickname;
Pet; me; light;
favorite pet;
usual; sha;
doggie; pet's
name; first pet;
animal; sh; shad;
s; car; first cat;
Dog's name;
chien; favorite
dog; ombre

Down

▼ 1:
2aZl4Ouarwm52NYYl9c

adobe; adobe2;
adobe2; adobe
twice; twice;
adobetwice;
adobe2x; site;
??????; name;
software; 2x;
company;
2xadobe;
programa; adobe
x 2; program;
adobe x2; ???;
ad; adobe*2;
????; Adobe;
double;
namenname;
2adobe; ?????;
x2; a; name twice;
photoshop;
company name;
adobe adobe;
adobe?; ado; aa;
company twice; 2;
marca; website;
none; adobe 2x;

How should you store a password on-disk?

Real solution: Hash and Add Salt!

Get a random number r :

$$r \leftarrow \$ \{0,1\}^d$$

Store:

$$h(\text{password} \parallel r), r$$

Attacker cannot easily precompute a table!

Modern techniques also include using intentionally slow hash functions

See bcrypt

Solutions

Let's run random, untrusted
code from the Internet



On-device Access Control Problem

Once a program is running on a system, how do we limit what it can do?

- E.g. multiple users ssh'd in to one system
- E.g. untrusted app running locally
- E.g. untrusted web renderer

Strategies: DAC & MAC

Discretionary Access Control (DAC)

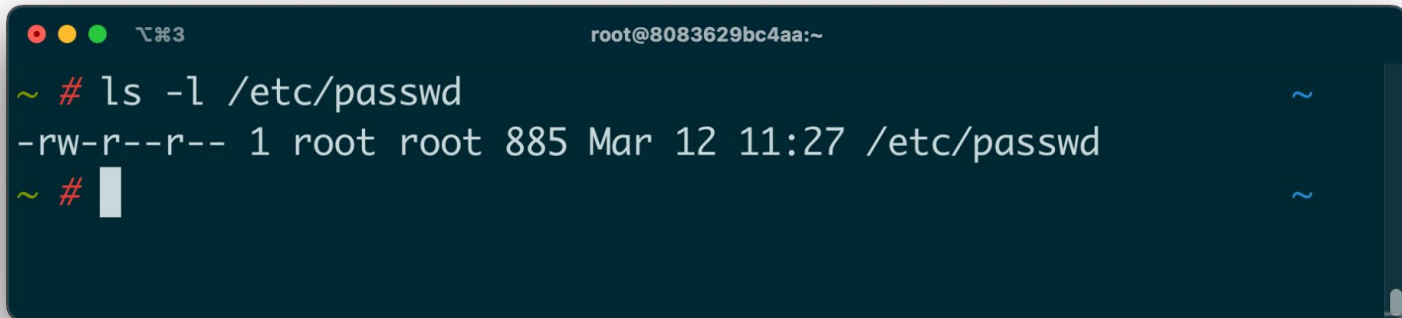
- Owner of a resource defines its access properties
- Can transfer ownership, control, etc

Mandatory Access Control (MAC)

- Policy for all resources is set at the organizational level / e.g. the computer

DAC Example: Unix Files

- Each user has:
 - A User ID (UID)
 - Any number of Group ID's (GID)
- All files have associated ACL metadata:
 - User owner
 - Group owner
 - Read, Write, eXecute bits for {user, group, other}
- Stored in filesystem's metadata



```
root@8083629bc4aa:~  
~ # ls -l /etc/passwd  
-rw-r--r-- 1 root root 885 Mar 12 11:27 /etc/passwd  
~ #
```

DAC Example: Unix Files

```
root@8083629bc4aa:~  
~ # ls -l /etc/passwd  
-rw-r--r-- 1 root root 885 Mar 12 11:27 /etc/passwd  
~ #
```

User

Group

Other

User Owner

Group Owner

DAC Problem: SetUID

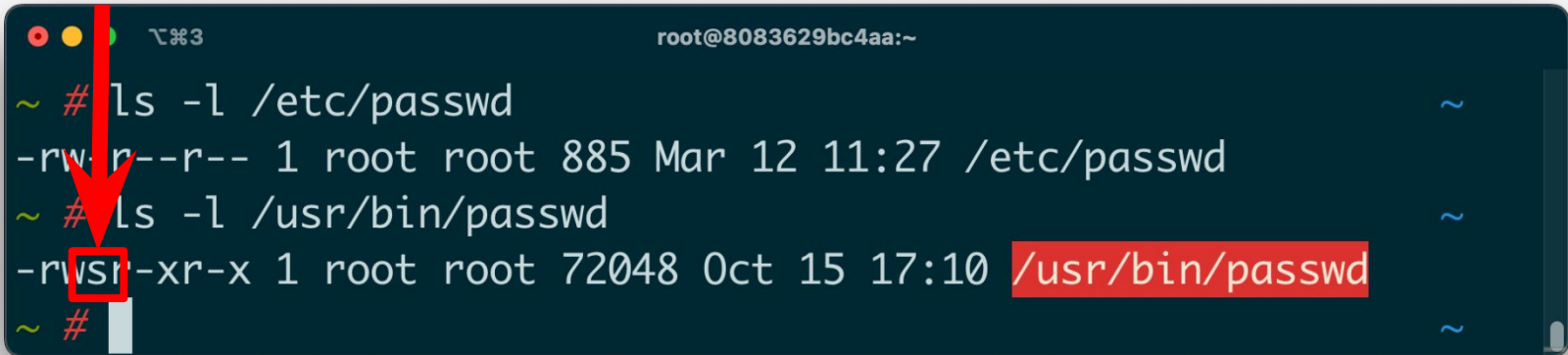
What if I want to change my password?

On linux, need to change /etc/passwd and /etc/shadow files

But, if I can edit these, I can change other folks' passwords, too
How do I change my own password?

Solution:

setuid binaries can be executed by a lower perm user



```
root@8083629bc4aa:~  
~ # ls -l /etc/passwd  
-rw-r--r-- 1 root root 885 Mar 12 11:27 /etc/passwd  
~ # ls -l /usr/bin/passwd  
-rwsr-xr-x 1 root root 72048 Oct 15 17:10 /usr/bin/passwd  
~ #
```

What could go wrong?

What happens if you exploit the setuid program?

Confused Deputy Problem

What happens if you exploit passwd?

Confused Deputy

Tricks a priv process into doing something

Inherent problem w/ DAC-only systems



Real world:
We combine MAC & DAC

Ex: Android!



Android Application Sandboxing: Goals

- Maximally Limit Untrusted Applications
- Apps can be written in any language
 - Cannot rely on PL primitives
- Apps *can* be malicious
- Also, System Decomposition:
 - E.g. Separating out media processing, wifi, bluetooth

Goal: Getting EIP on *one* component != Owning the OS

Android Application Sandboxing

DAC

- Standard Linux file permissions
- Each App has its own UID

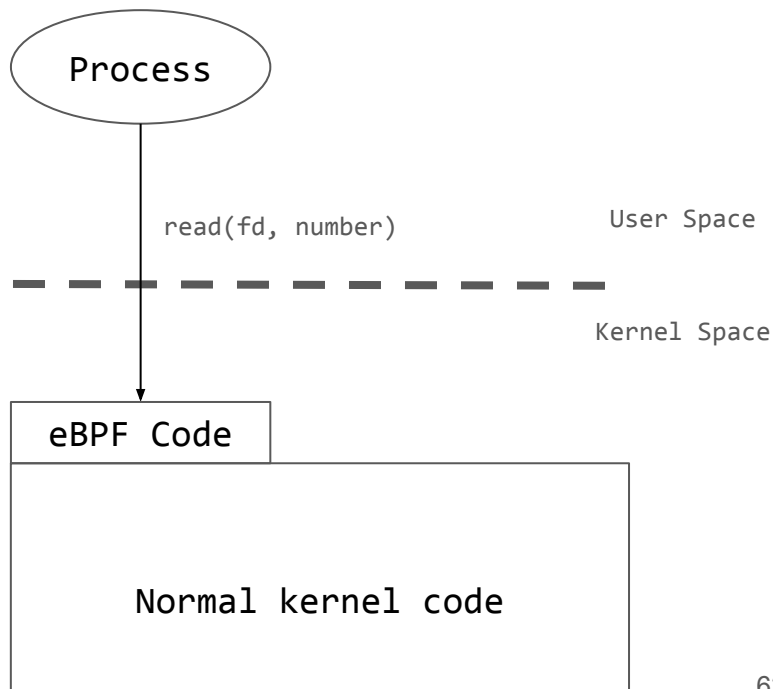
MAC == SELinux

- All processes on Android get a *context tag*
- At compile time, the kernel gets a configuration describing:
 - What files a process with each *context* has access to
 - What operations are allowed (e.g. read/write/execute)
- Everything else is **disallowed**
 - E.g. one could have a “root” process w/ access to nothing.

Android Application Sandboxing: Seccomp

Beyond MAC & DAC, there's SECCOMP!

- Goal: filter *specific* kernel calls
 - R/W/Open/IOCTL are too coarse
 - E.g. allow read but only for specific sections of a file
- eBPF
 - Non-turing complete language
 - Can “filter” specific inputs
 - Also, block
 - Injected in per-process basis





System Decomposition

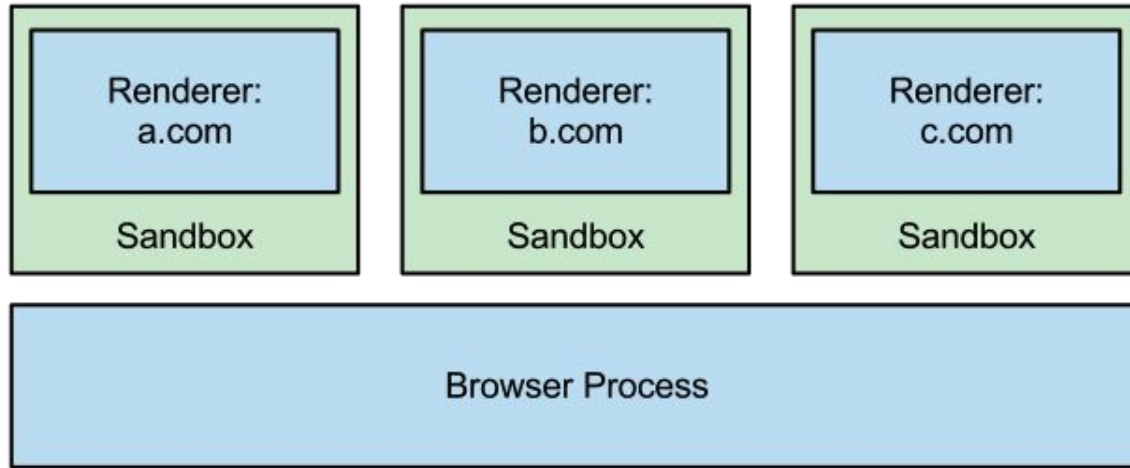
Theory: Make the attacker find multiple vulns

Example: Chrome

- Renderer deals with a ton of untrusted code
 - Javascript
- For every new page:
 - Fork
 - Drop to a new limited UID
 - Apply new SEPolicy Context
 - Apply seccomp filter
 - Only now we render.



System Decomposition Result



Another Take: Capabilities

Theory:

- MAC system
- OS provides explicit access *keys*
- Keys give an invoked process the ability to perform an *action* on some specific *object*

The Confused Deputy

(or why capabilities might have been invented)

Norm Hardy
Senior Architect

Key Logic
5200 Great America Parkway
Santa Clara, CA 95054-1108

Compiler allows a flag for output

Wrote to things

A compiler would get {output_fd, RW},
the OS checks if the invoker (e.g. the user) has the right to give those capabilities

Aside: Finding bugs

- Fuzzing
 - Send random data to a program to cause it to crash
 - Many strategies (e.g. maximizing coverage of input, novelty, etc)
- Many systems to play with:
 - AFL, libfuzzer
- OS-level Fuzzing from userspace -> kernel space:
 - Syzkaller!
 - Current bugs: <https://syzkaller.appspot.com/>
- Learn how to fuzz:
 - <https://github.com/google/oss-fuzz>
- Smashing the Stack exercises:
 - <https://microcorruption.com/>
 - CSAW CTF
 - GT Club: Grey Hat, WREKCTF

Readings

Intro to Security

- Original Paper Intro:
Smashing the Stack for Fun and Profit
- Memory Corruption History:
SOK: Eternal War in Memory

OS Primitives for Security

- System Decomp.:
Building Secure High-Performance Web Services with OKWS
- Android:
The Android Platform Security Model (2023)
- Capabilities:
Security in KeyKos (1986)

Final thoughts

1. Designing for security can be subtle
2. Mechanisms can and will fail, learn to design around them
3. Sandboxing and Access Control Can Help