

# Sandboxing

# logistics

CHALLENGE assignment — take-home portion of the final  
next class — final exam review

# CHALLENGE (1)

expect to release before Saturday; due by written final

probably complete all but two

five of seven or four of six

(waiting for TA feedback to calibrate difficulty)

similar format to “attack” homeworks

create a program that produces input

you are responsible for figuring out what scenario applies

# CHALLENGE (2)

some very similar to prior HWs, some not

reference solutions to OVER, ROP, FORMAT will be available  
you may modify and use these

you can ask about general strategies, but not specific challenges  
e.g. ask TAs/students to go through examples of how to do stack  
smashing  
e.g. ask TAs/students how to tell if pointer subterfuge would work

web page

# web security summary (1)

browser as OS:

- websites are like programs

cross-site scripting

- command injection for the web

- not just stuff to display — program code for website
- problem: runs with website permissions (e.g. cookies)

# web security summary (2)

isolation mechanism: same origin policy

decision: everything on domain name is “the same”

cross-site request forgery

consequence of statelessness

**all requests** send cookie (password-equivalent)

extra token to distinguish “user initiated” or not

# on user tracking

embedding one web page in another enables tracking users across website

example: multiple webpages include i frame with a google ad

your browser sends request to Google with same cookie

Google reliably gets excerpt of web history

reason: websites cooperated with Google

users often don't like this

what can browsers do about this?



# changing the cookie policy (1)

idea: no “third-party” cookies

only send cookies for URL in address bar

# changing the cookie policy (1)

idea: no “third-party” cookies

only send cookies for URL in address bar

now embedded Google calendar can't use my credentials

what about websites that use multiple domains?

## changing the cookie policy (2)

current Firefox “tracking protection” approach:

manually(?) created list of sites that do tracking

...and can be ignored without breaking things

# changing the cookie policy (3)

EFF Privacy Badger: heuristic approach

create score using

- amount of info in cookies

- number of places third-party appears

block requests to third-party or filter cookies if score too high

hard-coded exceptions for common false positives/tricky cases

- 'surrogate' code to avoid breaking website by blocking

  - tracking code has callbacks to third-party

  - e.g. facebook.com and fbcdn.com

# tracking without cookies

websites can do tracking even with no cookies

- information in URLs — add ?sessionID to all links
- other forms of browser storage — e.g. via Flash

websites can “fingerprint” browser and machine

- version, fonts, screen resolution, plugins, graphics features, ...
- caching of previously downloaded resources
- almost unique a surprising amount of the time

have IP addresses, too — very good hints

# tracking without cookies

websites can do tracking even with no cookies

- information in URLs — add ?sessionID to all links
- other forms of browser storage — e.g. via Flash

websites can “fingerprint” browser and machine

- version, fonts, screen resolution, plugins, graphics features, ...
- caching of previously downloaded resources
- almost unique a surprising amount of the time

have IP addresses, too — very good hints

# Web Frameworks

tools for making writing interactive websites help

e.g. Django (Python):

- default to anti-embedding HTTP header (no clickjacking)

- default to HttpOnly cookies

- default to requiring CSRF token for POSTs

usually provide “templates” which escape HTML properly by default

template: `<p>Name: {{name}} (placeholder in {{...}})`

if name is `<script>... result is`

`<p>Name: &lt;script>...</p>`

# recall: UAF triggering code

earlier in semester: exploit in Chrome **browser** itself

```
// in HTML near this JavaScript:  
// <video id="vid"> (video player element)  
function source_opened() {  
    buffer = ms.addSourceBuffer('video/webm;_codecs="vorbis,vp8"');  
    vid.parentNode.removeChild(vid);  
    gc(); // force garbage collector to run now  
    // garbage collector frees unreachable objects  
    // (would be run automatically, eventually, too)  
    // buffer now internally refers to delete'd player object  
    buffer.timestampOffset = 42;  
}  
ms = new WebKitMediaSource();  
ms.addEventListener('webkitsourceopen', source_opened);  
vid.src = window.URL.createObjectURL(ms);
```



# recall: UAF triggering code

earlier in semester: exploit in Chrome **browser** itself

```
// in HTML near this JavaScript:  
// <video id="vid"> (video player element)  
function source_opened() {  
    buffer = ms.addSourceBuffer('video/webm;_codecs="vorbis,vp8"');  
    vid.parentNode.removeChild(vid);  
    gc(); // force garbage collector to run now  
    // garbage collector frees unreachable objects  
    // (would be run automatically, eventually, too)  
    // buffer now internally refers to delete'd player object  
    buffer.timestampOffset = 42;  
}  
ms = new WebKitMediaSource();  
ms.addEventListener('webkitsourceopen', source_opened);  
vid.src = window.URL.createObjectURL(ms);
```

# recall: UAF triggering code

earlier in semester: exploit in Chrome **browser** itself

```
// in HTML near this JavaScript:  
// <video id="vid"> (video player element)  
function source_opened() {  
    buffer = ms.addSourceBuffer('video/webm;_codecs="vorbis,vp8"');  
    vid.parentNode.removeChild(vid);  
    gc(); // force garbage collector to run now  
    // garbage collector frees unreachable objects  
    // (would be run automatically, eventually, too)  
    // buffer now internally refers to delete'd player object  
    buffer.timestampOffset = 42;  
}  
ms = new WebKitMediaSource();  
ms.addEventListener('webkitsourceopen', source_opened);  
vid.src = window.URL.createObjectURL(ms);
```

# recall: UAF triggering code

earlier in semester: exploit in Chrome **browser** itself

```
// in HTML near this JavaScript:  
// <video id="vid"> (video player element)  
function source_opened() {  
    buffer = ms.addSourceBuffer('video/webm;_codecs="vorbis,vp8"');  
    vid.parentNode.removeChild(vid);  
    gc(); // force garbage collector to run now  
    // garbage collector frees unreachable objects  
    // (would be run automatically, eventually, too)  
    // buffer now internally refers to delete'd player object  
    buffer.timestampOffset = 42;  
}  
ms = new WebKitMediaSource();  
ms.addEventListener('webkitsourceopen', source_opened);  
vid.src = window.URL.createObjectURL(ms);
```

# browsers and exploits

browsers are in a particularly dangerous position for exploits

**routinely run untrusted code** (JavaScript on websites)

huge amounts of code, often written in C/C++

WebKit (part of Chrome, Safari) has millions of lines of code

# malvertising

could trick user into visiting your website

or pay for ad — embed your webpage in another!  
can run whatever script you like

Readers of popular websites targeted by  
stealthy Stegano exploit kit hiding in pixels of  
malicious ads

BY ESET RESEARCH POSTED 6 DEC 2016 - 12:00PM

EXPLOIT KIT

# modern advertising landscape (1)

website ads are often **sold in realtime**

conceptual idea: **mini-auction** for every ad

major concerns about fraud

are you really showing my ad?

ad operators want to do own tracking

get better idea what to show/bid

## modern advertising landscape (2)

website operators **typically don't host ads**

- don't build own realtime auction infrastructure
- not trusted to report number of ad views correctly

ads often sold indirectly

- middleman handles bidding/etc.
- website operators sell to multiple ad operators

# browsers and exploit mitigations

modern browsers employ many of the mitigations we talked about

- full ASLR

- write XOR execute (with exceptions for runtime-compiled code)

- stack canaries

also some **other mitigations**



# least privilege

why can code running for a webpage install software?

**never** needs to do that

concept: let's run it without those permissions

# multi-user OSs

```
cr4bd@labunix01:~$ cp myprogram.exe /bin/ls  
cp: cannot create regular file '/bin/ls' : Permission denied
```

programs have **limited privileges**

# permission enforcement

```
struct Process {
    int user_id;
    ...
};
int handle_open_system_call(char *filename, ...) {
    Process* currentProcess = GetCurrentProcess();
    File* file = GetFileByFilename(filename);
    if (!file->UserCanAccess(currentProcess->user_id)) {
        return ERROR_PERMISSION_DENIED;
    }
    ...
}
```

# multi-user OSs

```
cr4bd@labunix01:~$ cp myprogram.exe /bin/ls  
cp: cannot create regular file '/bin/ls' : Permission denied
```

programs have **limited privileges**

OS tracks “user” of running every program

result: malware I installed shouldn't be able to effect other users

idea 1: reuse this support for web browsers

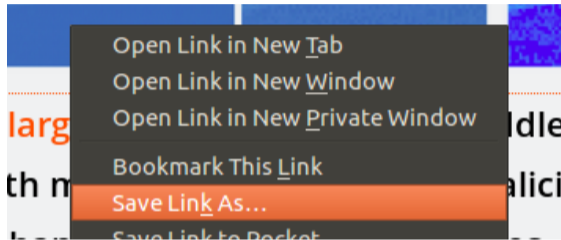
- webpage should run as “different user”

- malware should only affect web browser?

# things browsers need

what things should browser be able to do?

# things browsers need



save files

have your webmail password

...

# the privilege separation idea

can't make whole browser run as "different user"  
still need to save files, read password, etc.

how about just the parts that are "dangerous"?  
part that runs scripts, parses HTML

# simple privilege separation

simple example: want to show videos

video decoding library is tens of thousands of lines of code  
often buggy, includes hard-to-check hand-written assembly

what does video decoding library do?

read video file as input  
output images as output



# simple privilege separation

setup: create new user

start video decoder as new user

communicate via “pipes”

like terminal to be used by program

# simple privilege separation

```
/* dangerous video decoder to isolate */
int main() {
    /* switch to right user */
    SetUserTo("user-without-privileges");
    while (fread(videoData, sizeof(videoData), 1, stdin) > 0) {
        doDangerousVideoDecoding(videoData, imageData);
        fwrite(imageData, sizeof(imageData), 1, stdout);
    }
}

/* code that uses it */
FILE *fh = RunProgramAndGetFileHandle("./video-decoder");
for (;;) {
    fwrite(getNextVideoData(), SIZE, 1, fh);
    fread(image, sizeof(image), 1, fh);
    displayImage(image);
}
```

# issues with privilege separation (1)

“other user” can still do too much

read unprotected files

most of them?

write temporary files?

open network connections

use all your memory

...

## issues with privilege separation (2)

awkward to do

switching users requires special permissions

seperate user for **each** video decoder, audio decoder, web page renderer?

users can debug processes from same user

slowdown — extra copying

# recall: process virtual machine

process has isolated memory + CPU

communicating outside? needs **system calls**  
analagous to using I/O devices

**OS controls what process can do**

# Linux system call filtering API

privilege separation support: system call filtering

simple API: `seccomp(SECCOMP_SET_MODE_STRICT, 0, 0)`

“The only system calls the calling thread is permitted to make are `read`, `write`, `_exit`, and `sigreturn`. Other system calls [kill the program].”

read/write only work on **already open files**

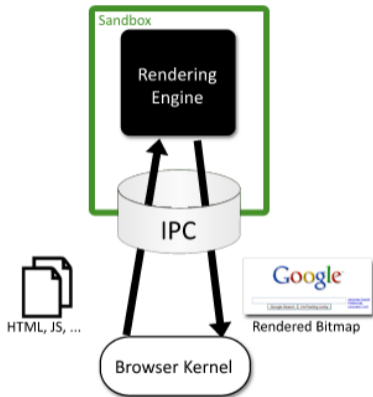
# “sandboxing”

result of filtering called a “sandbox”

idea: attacker can play in sandbox as much as they want

can't do anything harmful

# Chrome architecture



**Figure 1:** The browser kernel treats the rendering engine as a black box that parses web content and emits bitmaps of the rendered document.



# talking to the sandbox

browser kernel sends commands to sandbox

sandbox sends commands to browser kernel

idea: commands only allow necessary things

# original Chrome sandbox interface

sandbox to browser “kernel”

- show this image on screen

  - (using shared memory for speed)

- make request for this URL

- download files to local FS

- upload user requested files

browser “kernel” to sandbox

- send user input

# original Chrome sandbox interface

sandbox to browser “kernel”

show this image on screen

(using shared memory for speed)

make request for this URL

download files to local FS

upload user requested files

browser ' needs filtering — at least no file: (local file) URLs

send user input

# original Chrome sandbox interface

sandbox to browser “kernel”

show this image on screen

(using shared memory for speed)

make request for this URL

download files to local FS

upload user requested files

browser “kernel” to sandbox

send user input

can still read any website!  
still sends normal cookies!

# original Chrome sandbox interface

sandbox to browser “kernel”

show this image on screen

(using shared memory for speed)

make request for this URL

download files to local FS

upload user requested files

browser “kernel” to

send user input

files go to download directory only  
can't choose arbitrary filenames

# original Chrome sandbox interface

sandbox to browser “kernel”

show this image on screen

(using shared memory for speed)

make request for this URL

download files to local FS

upload user requested files

browser “kernel” to sandbox

send user input

browser kernel displays file chooser  
only permits files selected by user

# process per site

Chrome **almost** does process-per-site

idea: one sandbox process per site

with one **huge exception**

recall: same-origin policy

# recall: operations not requiring same origin

loading images, stylesheets (CSS), video, audio

loading scripts

but not getting syntax errors

accessing with “permission” of other website

submitting forms to other webpages

displaying other webpages (but not reading contents)



# browser kernel security

the browser kernel is not simple

needs to **securely** implement **special protocol**

UI, networking code overall more complicated than before

hope: writing secure browser kernel easier than secure whole-browser

# OpenSSH privilege separation

OpenSSH uses privilege separation for its SSH server

what runs on the lab machines when you log into them

separate network processing code from authentication code

separate process per connection — users don't share

# OpenSSH privsep protocol

sandboxed process tells “monitor” to:

perform **cryptographic operations**

- long-term keys never in sandboxed process

- commands to ask for cryptographic messages they need

ask to switch to user — if given user password, etc.

- monitor process verifies** login information

after authentication: new process running as logged-in user

- (normally) no issues with special privileges

# privilege seperation overall

large application changes

OpenSSH: 3k lines of code for communication/etc. added  
OpenSSH: 2% of existing code (950 of 44k lines) changed  
(but most changes simple)

lots of application knowledge

what is a meaningful separation of 'privileged' and 'unprivileged'?

better application design anyways?

# application confinement

confining whole browsers was hard

we trust them to do a lot of things — e.g. write arbitrary files

but maybe we can do this for simpler applications?

idea 1: applications send system calls to OS

**limit syscalls** like we limited browser kernel commands  
constructing command language “in reverse”

# filtering system calls?

example: video player VLC playing a local file on my laptop

uses **73 unique system calls**

opens many files that **are not the video file**

- libraries

- fonts

- configuration files

- translations of messages

can I limit the files my video player can read?

how do I come up with a useful filter?

# OS X sandboxing

OS X (tries to) implement system call filtering

main challenge: what about files?

user can open a file anywhere — we expect that to work

# OS X sandboxing

OS X (tries to) implement system call filtering

main challenge: what about files?

user can open a file anywhere — we expect that to work

OS X solution: OS service displays file-open dialog

OS knows user really choose a file

application can ask to remember file was chosen previously

not chosen/remembered — can't access

requires changes to how applications open files



# another sandboxing OS: Qubes

Qubes: heavily sandboxed OS

runs **seperate VMs** instead of filtering syscalls

UI that clearly shows what VM each window is from

advantage: easier to gaurentee isolation

many, many more bugs in system call filtering than VMs

disadvantage: harder to share between VMs

disadvantage: much more runtime overhead

# Qubes screenshot

The screenshot displays a Qubes VM environment. The primary window is a PDF viewer showing the 'ENCLAVE OPERATION' section of the 'Software Guard Extensions Programming Reference'. The PDF content includes:

**ENCLAVE OPERATION**

After AEX has completed, the logical processor is no longer in enclave mode and the exiting event is processed normally. Any new events that occur after the AEX has completed are treated as having occurred outside the enclave (e.g. a #PF in dispatching to an interrupt handler).

**3.2.3 Resuming Execution after AEX**

After system software has serviced the event that caused the logical processor to exit an enclave, the logical processor can re-start execution using ERESUME. ERESUME restores registers and returns control to where execution was interrupted.

If the cause of the exit was an exception or a fault and was not resolved, the event will be triggered again if the enclave is re-entered using ERESUME. For example, if an enclave performs a divide by 0 operation, executing ERESUME will cause the enclave to attempt to re-execute the faulting instruction and result in another divide by 0 exception. In order to handle an exception that occurred inside the enclave, software can enter the enclave at a different location and invoke the exception handler within the enclave by executing the BENTER instruction. The exception handler within the enclave can attempt to resolve the faulting condition or simply return and indicate to software that the enclave should be terminated (e.g. using EXIT).

**3.2.3.1 ERESUME interaction**

ERESUME restores registers depending on the mode of the enclave (32 or 64 bit).

- In 32-bit mode (IA32\_EFER.LMA = 0 || CS.L = 0), the low 32-bits of the legacy registers (EAX, EBX, ECX, EDI, ESP, EBP, ESI, EDI, EIP and EFLAGS) are restored from the thread's GPR area of the current SSA frame. Neither the upper 32-bits of the legacy registers nor the 64-bit registers (R8 ... R15) are loaded.
- In 64-bit mode (IA32\_EFER.LMA = 1 && CS.L = 1), all 66-bits of the general processor registers (RAX, RBX, RCG, RDX, RSP, RBP, RSI, RDI, RIB, R15, RIP and RFLAGS) are loaded.

Extended features specified by SECS\_ATTRIBUTES.XFRM are restored from the XSAVE area of the current SSA frame. The layout of the x87 area depends on the current values of IA32\_EFER.LMA and CS.L:

- IA32\_EFER.LMA = 0 || CS.L = 0
  - 32-bit load in the same format that XSAVE/FXSAVE uses with these values.
- IA32\_EFER.LMA = 1 && CS.L = 1
  - 64-bit load in the same format that XSAVE/FXSAVE uses with these values plus REX.W = 1

**3.3 CALLING ENCLAVE PROCEDURES**

**3.3.1 Calling Convention**

In standard call conventions subroutine parameters are generally pushed onto the stack. The called routine, being aware of its own stack layout, knows how to find parameters based on compile-time-computable offsets from the SP or BP register (depending on runtime conventions used by the compiler).

Because of the stack switch when calling an enclave, stack-located parameters cannot be found in this manner. Entering the enclave requires a modified parameter passing convention. For example, the caller might push parameters onto the untrusted stack and then pass a pointer to those parameters in RAX to the enclave software. The exact choice of calling conventions is up to the writer of the edge routines; be those routines hand-coded or compiler-generated.

**3.3.2 Register Preservation**

As with most systems, it is the responsibility of the callee to preserve all registers except that used for returning a value. This is consistent with conventional usage and tends to optimize the number of register save/restore operations.

34 of 156 | Software Guard Extensions Programming Refer... | 88.66% | 329298-001.pdf

10:00 | [Domo] Qubes VM Manager

Name	State	NetVM	CPU Graph	MEM
dom0	Running	n/a		2598 MB
sys-net	Running	n/a		301 MB
sys-firewall	Running	sys-net		301 MB
varia	Running	sys-firewall		979 MB
work-web	Running	sys-firewall		1173 MB
work-mutt	Running	sys-firewall		604 MB
keys-it-e-mail	Running	---		478 MB
work	Running	sys-firewall		607 MB
personal	Running	sys-firewall		750 MB

[user@work ~]\$

10:00 | [user@work ~]\$

34 of 156 | Software Guard Extensions Programming Refer... | 88.66% | 329298-001.pdf

10:00 | [user@work ~]\$

# quick review

part 1: malware and anti-malware

part 2: (memory) vulnerabilities and exploits and mitigations

part 3: bug-finding/prevention and misc. vulnerabilities and exploits

# malware — evil software

tricks itself onto victim machines

- e.g. masquerade as useful software

- e.g. embed in legitimate software (viruses)

- e.g. attack vulnerabilities in software to spread

- e.g. arrange to run automatically on disk insert

cat-and-mouse game — antivirus software to detect malware

- patterns, heuristics to detect

- tricks to appear like normal software

# memory vulnerabilities and exploits

buffer overflow/underflow — program writes outside of array

if “important” data, attacker can gain control

usual goal: overwrite pointer to code

use-after-free — program uses data as wrong type

attacker controls data as one type

ideally, misinterpreted (via dangling pointer) to contain pointer to code

# memory exploit mitigations

bounds-checking — don't allow outside-of-array writes

doesn't solve use-after-free

single object with array and pointers?

stack canaries — detect writes next to return addresses

ASLR — make it so program can't make up useful pointers?

problem: memory bugs can print out pointers

W xor X — make it so attacker can't write new code

problem: attack can reuse existing code (return-oriented programming)

# bug-finding

systematic testing — find crashes ( $\approx$  vulnerability)

- fuzz testing — generate random tests

- coverage-guided fuzz-testing — random tests, weighted by what runs

- symbolic execution — solve for input to reach each possibility

static analysis — look for dangerous patterns

- usually false positives and/or negatives

- typically examine potential paths through program

# bug-prevention

ownership — enforceable rule to prevent use-after-free

- never free while object is owned

- one writer (could be changing internal pointers) or many readers

- readers and writers can borrow from owner

- language (e.g. Rust) can track borrowing lifetimes to make safe

alternate safe policies — reference counting, etc.

- have runtime overhead, but can be used only when needed

escape hatch — only check small amount of unsafe code

- ideally implements policies that make sense

- at least limits the code one needs to check



# command injection/web security

command injection — type confusion problems

try to embed constant/etc., end up embedding commands

lots of languages to embed in — command line, SQL, HTML, ...

web security

same origin policy (SOP) — isolate by domain name (mostly)

XSS — command injection for the web

trusting client inputs — the attacker controls their browser

CSRF — innocent browser submits bad request (w/ cookies) for attacker

clickjacking — “steal” user’s click to make request

## next time

final exam review: bring questions