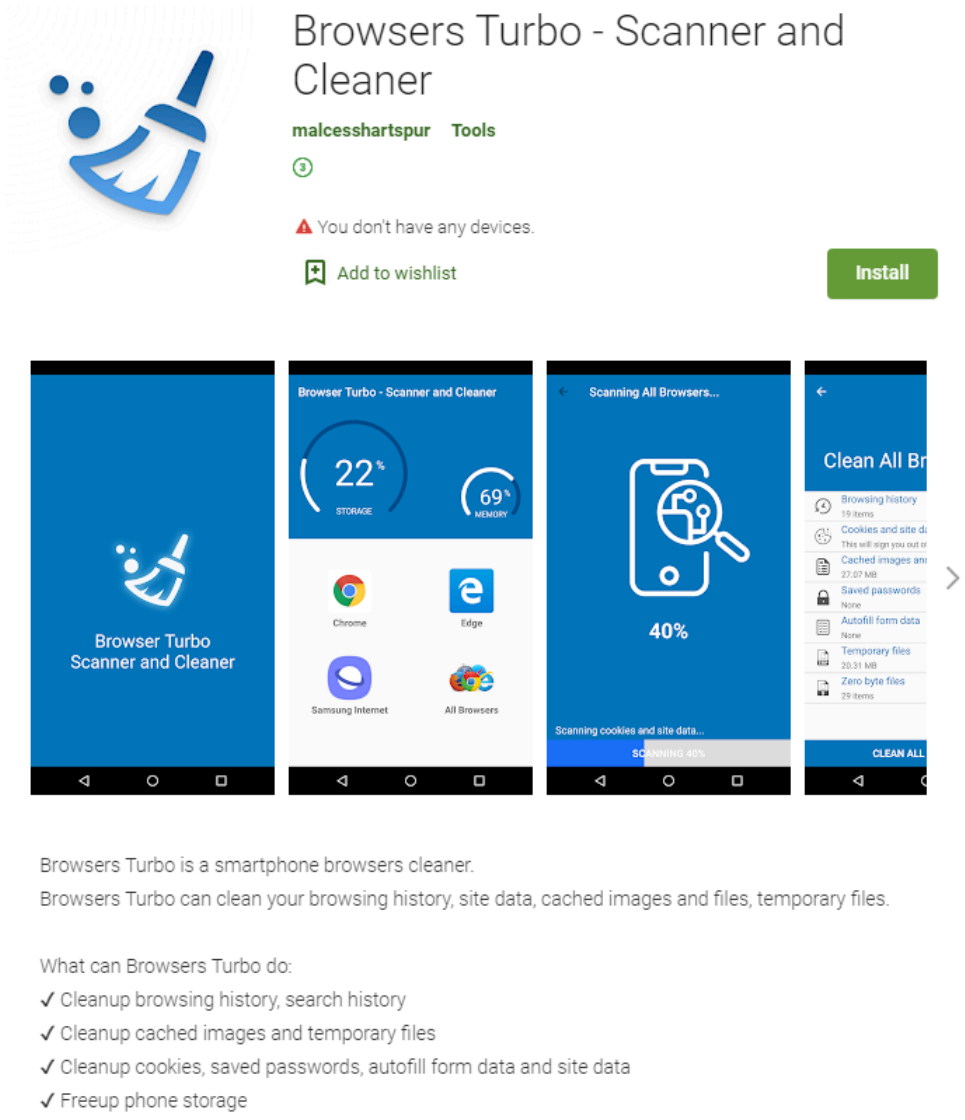
Hiding in plain sight: PhantomLance walks into a market

By Alexey Firsh

Published: 2020-04-28 · Archived: 2026-04-05 14:34:03 UTC

In July 2019, Dr. Web reported about a backdoor trojan in Google Play, which appeared to be sophisticated and unlike common malware often uploaded for stealing victims' money or displaying ads. So, we conducted an inquiry of our own, discovering a long-term campaign, which we dubbed "PhantomLance", its earliest registered domain dating back to December 2015. We found dozens of related samples that had been appearing in the wild since 2016 and had been deployed in various application marketplaces including Google Play. One of the latest samples was published on the official Android market on November 6, 2019. We informed Google of the malware, and it was removed from the market shortly after.



The latest example of spyware in Google Play disguised as a browser cleaner

During our investigation, we discovered various overlaps with reported OceanLotus APT campaigns. Thus, we found multiple code similarities with the previous Android campaign, as well as macOS backdoors, infrastructure overlaps with Windows backdoors and a few cross-platform resemblances.

Besides the attribution details, this document describes the actors' spreading strategy, their techniques for bypassing app market filters, malware version diversity and the latest sample deployed in 2020, which uses Firebase to decrypt the malicious payload. We also found out that Blackberry Cylance research team investigated this activity.

Our report is broken down into several sections.

1. 1 Malware versions – technical description of versions found, their features and relationships between them.
2. 2 Spread – information on specific tactics used by the threat actors for distributing their malware.
3. 3 Infrastructure – further details on uncovered infrastructure pieces as well as overlaps found.
4. 4 Victimology – thoughts on the actors' interests in choosing their targets.
5. 5 Overlaps with previous campaigns – details of similarities with all related campaigns that we have identified.

More information on PhantomLance is available to customers of Kaspersky Intelligence Reporting. For more information, contact intelreports@kaspersky.com

Malware versions

For the purposes of the research, we divided samples we found into a series of "versions" based on technical complexity: from the basic Version 1 to the highly sophisticated Version 3. Note that they do not fully correlate with the chronological order of their appearance ITW: for example, we observed Version 1 samples in late 2019 and in 2017, the year that we also saw Version 3.

Functionality of all samples are similar – the main purpose of spyware was to gather sensitive information. While the basic functionality was not very broad, and included geolocation, call logs, contact access and SMS access, the application could also gather a list of installed applications, as well as device information, such as model and OS version. Furthermore, the threat actor was able to download and execute various malicious payloads, thus, adapting the payload that would be suitable to the specific device environment, such as Android version and installed apps. This way the actor is able to avoid overloading the application with unnecessary features and at the same time gather information needed.

Version 1

We attribute the latest Google Play sample (MD5: 2e06bbc26611305b28b40349a600f95c) to this version. This is a clear payload, and unlike the other versions, it does not drop an additional executable file. Our main theory about the reasons for all these versioning maneuvers is that the attackers are trying to use diverse techniques to achieve their key goal, to bypass the official Google marketplace filters. And achieve it they did, as even this version passed Google's filters and was uploaded to Google Play Store in 2019 (see Spreading for details).

No suspicious permissions are mentioned in the manifest file; instead, they are requested dynamically and hidden inside the dex executable. This seems to be a further attempt at circumventing security filtering. In addition to that, there is a feature that we have not seen before: if the root privileges are accessible on the device, the malware can use a reflection call to the undocumented API function "setUidMode" to get permissions it needs without user involvement.

```
private static int a(Context arg8, int arg9, int arg10, int arg11, String arg12)
    Object v0 = arg8.getSystemService("appops");
    Class v1 = AppOpsManager.class;
    Class[] v3 = new Class[3];
    Class v4 = Integer.TYPE;
    v3[0] = v4;
    v3[1] = v4;
    v3[2] = v4;
    v1.getMethod("setUidMode", v3).invoke(v0, Integer.valueOf(arg9), Integer.valu
            .valueOf(arg11));
    return j.a(arg8, arg9, arg10, arg12);
}

private static int a(Context arg7, int arg8, int arg9, String arg10) {
    Object v7 = arg7.getSystemService("appops");
    Class v0 = AppOpsManager.class;
    Class[] v2 = new Class[3];
    Class v3 = Integer.TYPE;
    v2[0] = v3;
    v2[1] = v3;
    v2[2] = String.class;
    return v0.getMethod("checkOp", v2).invoke(v7, Integer.valueOf(arg8), Integer.
            arg10).intValue();
}

private static int a(Context arg5, String arg6) {
    return AppOpsManager.class.getMethod("permissionToOpCode", String.class).inv
            "appops"), arg6).intValue();
}
```

Note that this trick only works with Android SDK version 19 or higher.

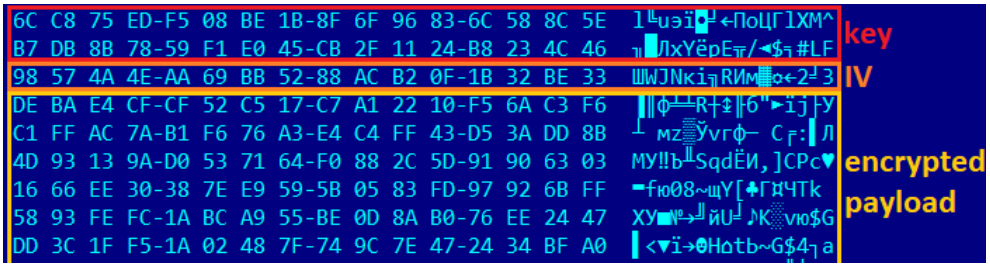
Most of the aforementioned operations naturally require root access, but we believe that the root exploit may be delivered as payload in a server response to collected device info. Also, some of the applications that the malware mimics will have notified the user that they only work on rooted devices. For instance, Browser Cleaner can only clean up the browser cache if it is given root permissions.

Version 2

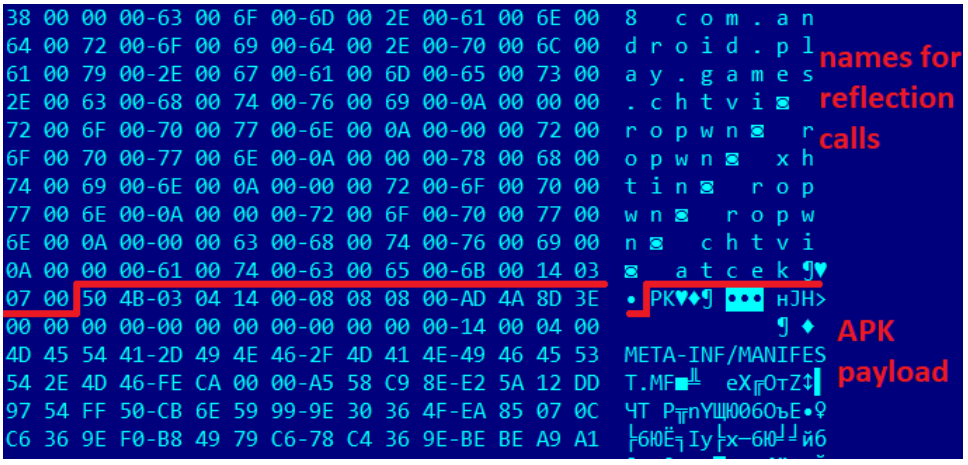
Specimens of this version were also detected in 2019 and earlier. One of the samples was located in Google Play Store in November 2019 and described in the Dr. Web blog. Based on our detection statistics and spotted version stamps, we believe that this version is a replacement for Version 3, which we did not observe in 2019.

Below are the most valuable points and main differences from the Version 1.

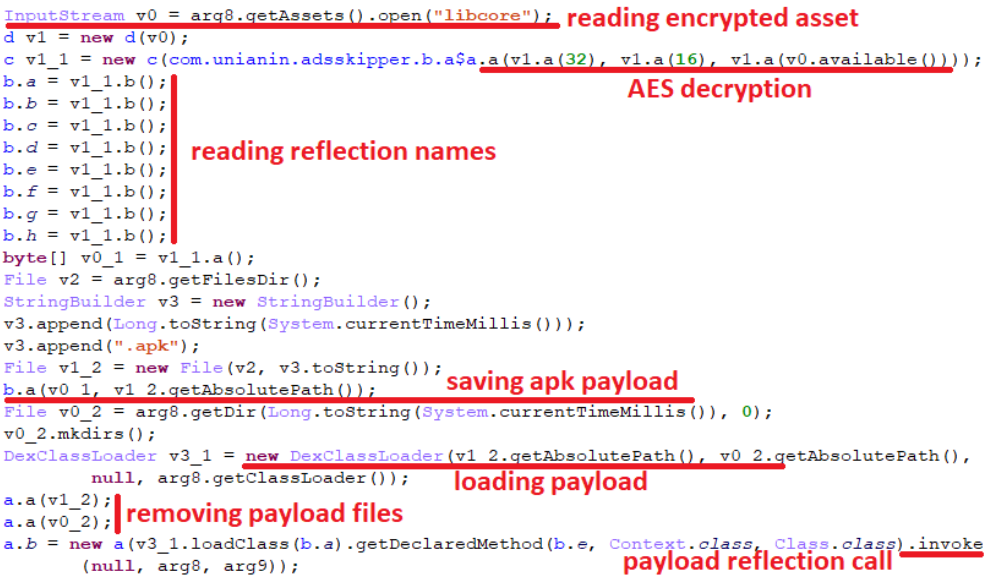
The malicious payload APK is now packed in an encrypted file in the assets directory and is decrypted by the first stage using an AES algorithm. A decryption key and initialization vector (IV) are located in the first 32 + 16 bytes of the encrypted payload.



After decryption, the asset file will look like this.



As you can see, before the APK magic, the file header contains strings that are used for making further reflection calls to payload methods. Here is the first-stage code fragment with explanations regarding the payload loading process.



All Version 2 payloads use the same package name, “com.android.play.games”, which probably mimics the official Google Play Games package, “com.google.android.play.games”.

Moreover, we spotted developer version stamps in decrypted payloads.

MD5	Developer version stamp
65d399e6a77acf7e63ba771877f96f8e	5.10.6084
6bf9b834d841b13348851f2dc033773e	5.10.6090
8d5c64fdaae76bb74831c0543a7865c3	5.10.9018
3285ae59877c6241200f784b62531694	5.10.9018
e648a2cc826707aec33208408b882e31	5.10.9018

It is worth mentioning payload manifests, which do not contain any permission requests. As stated in the description of Version 1, permissions required by the malicious features are granted via an undocumented Android API.

We have found two different certificates used for signing Version 2 payloads.

MD5	Certificate
6bf9b834d841b13348851f2dc033773e	Serial Number: 0xa4ed88e620b8262e Issuer: CN=Lotvolron Validity: from = Wed Jan 20 11:30:49 MSK 2010
65d399e6a77acf7e63ba771877f96f8e	
8d5c64fdaae76bb74831c0543a7865c3	Serial Number: 0xd47c08706d440384 Issuer: CN=Ventoplex Validity: from = Wed Apr 13 05:21:26 MSK 2011
3285ae59877c6241200f784b62531694	
e648a2cc826707aec33208408b882e31	

Although validity dates look spoofed in both cases and do not point to any real deployment times, by analyzing all payload certificates, we discovered that the second one (Ventoplex) was used to sign Version 3 payloads as well.

Version 2.1

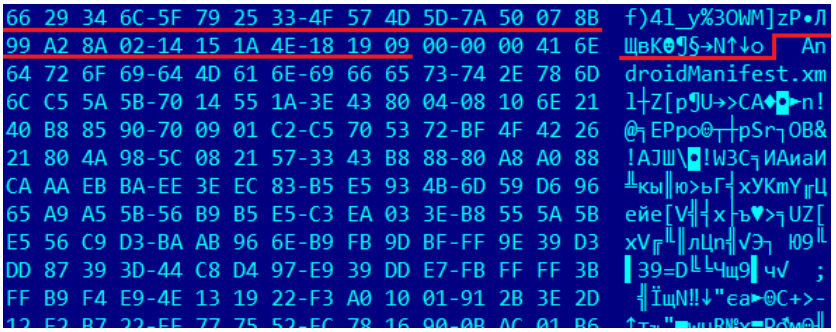
The latest samples of PhantomLance discovered in the early 2020 introduced a new technique for decrypting payloads: the malicious payload was shipped with its dropper, encrypted with AES. The key is not stored anywhere in the dropper itself but sent to the device using Google’s Firebase remote config system. The other technical features are very similar to the ones we observed in Version 2, so we tagged this generation as Version 2.1.

We were able to make a valid request to PhantomLance's Firebase API. The response consisted of a JSON struct containing the AES decryption key, where the "code_disable" value is the decryption key for payload.

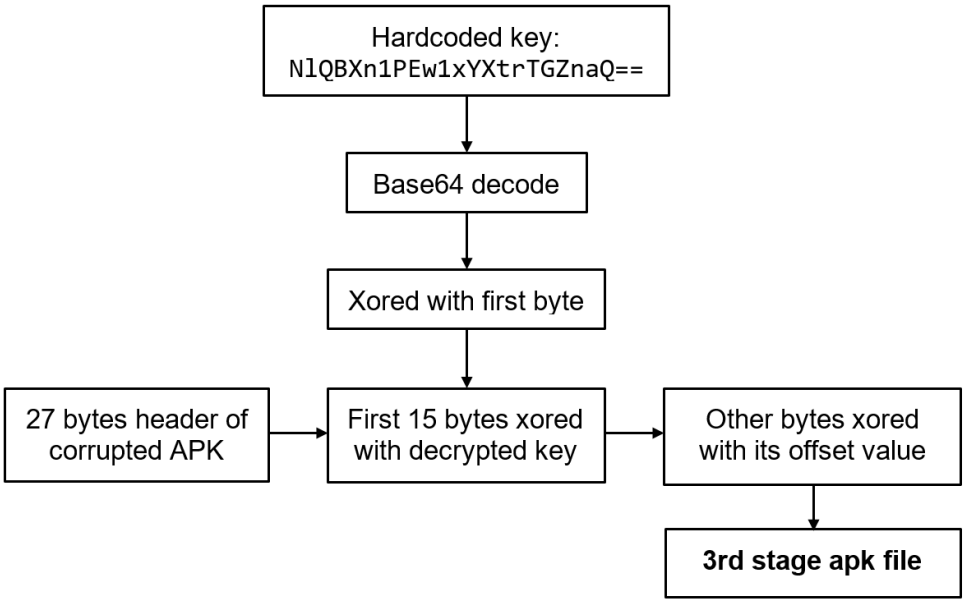
```
{
  "entries": {
    "code_disable": "27ypYitp1UFc9Tvh"
  },
  "appName": "com.ozerlab.callrecorder",
  "state": "UPDATE"
}
```

What is important, the dropper expects that the AES decryption key will be stored in a parameter named "code", so this specific variant should not function properly. Besides, we noticed that Firebase previously returned one more field, named "conf_disable", which has the same value as the "code_disable", so we assume that the actors are still tinkering with this new feature.

Another interesting technique that the actors are trying to implement is a third-stage payload implant. The second-stage payload (MD5: 83cd59e3ed1ba15f7a8cadfe9183e156) contains an APK file named "data" (MD5: 7048d56d923e049ca7f3d97fb5ba9812) with a corrupted header in the assets path.



The second stage reads this APK file, decrypts it and rewrites its first 27 bytes as described below.



This results in an APK file (MD5: c399d93146f3d12feb32da23b75304ba) that appears to be a typical PhantomLance payload configured with already known C2 servers (cloud.anofrio[.]com, video.viodger[.]com, api.anaehler[.]com). This third-stage APK is deployed with a custom native library named "data.raw", also stored at the assets path. This library is

used for achieving persistence on the infected device and appears to be a custom daemonized ELF executable based on the open-source "daemon.c" Superuser tool component, while in previous samples, we saw MarsDaemon used for this purpose.

```
if ( v3 < 0
  || sub_AB04()
  || (memset_0(&v28, 0, 110),
      v28 = 1,
      v4 = (const char *)sub_8222(&unk_176E0, 17),
      sprintf(v29, "%s/server", v4),
      memset_0(v29, 0, 108),
      v5 = sub_8222(&unk_176F2, 11),
      memcpy_0(v29, v5, 11),
      connect_0(v3, &v28, 110)) )
{
LABEL_2:
  exit(-1);
}
LODWORD(v6) = getenv("MOUNT_EMULATED_STORAGE");
HIDWORD(v6) = v6;
LODWORD(v6) = v6 - 1;
v21 = (v6 - __PAIR64__(v6, 1)) >> 32;
if ( getenv("SUPERUSER_SEND_TTY") )
  goto LABEL_37;
v7 = isatty(0);
v8 = (__PAIR64__(v7, v7 - 1) - __PAIR64__(v7 - 1,
if ( isatty(1) )
  v8 |= 2u;
if ( isatty(2) )
{
  LOBYTE(v8) = v8 | 4;
  goto LABEL_11;
}
```

```
sprintf(sun.sun_path, "%s/server", REQUESTOR_DAEMON_PATH);

memset(sun.sun_path, 0, sizeof(sun.sun_path));
memcpy(sun.sun_path, "\0" "SUPERUSER", strlen("SUPERUSER") + 1);

if (0 != connect(socketfd, (struct sockaddr*)&sun, sizeof(sun)))
    PLOGE("connect");
    exit(-1);
}

LOGD("connecting client %d", getpid());

int mount_storage = getenv("MOUNT_EMULATED_STORAGE") != NULL;

// Determine which one of our streams are attached to a TTY
int atty = 0;

// Send TTYs directly (instead of proxying with a PTY) if
// the SUPERUSER_SEND_TTY environment variable is set.
if (getenv("SUPERUSER_SEND_TTY") == NULL) {
```

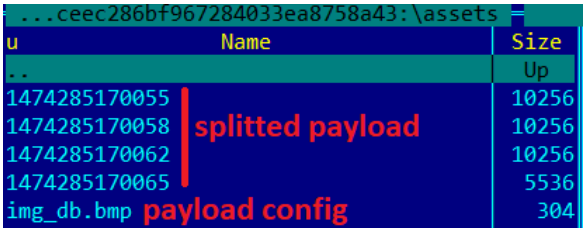
Code comparison of the library used to daemonize the third stage payload with daemon.c source code hosted on Github

Version 3

While we have found that Version 2 has been used as a replacement for this one, as we have not observed any new deployments of Version 3 in 2019, it still looks more advanced in terms of technical details than Version 2. According to our detection statistics and deployment dates on application markets, Version 3 was active at least from 2016 to 2018.

Below are the most valuable points and main differences between Version 3 and Version 2.

The first-stage dropper appears even more obfuscated than that in Version 2; it uses a similar way of decrypting the payload, but it has minor differences. The encrypted content is split into multiple asset files under 10256 bytes in size plus an encrypted config file, and contains payload decryption details.



Below is the payload decryption sequence.

1. 1 Decrypt the payload config file from the assets with both a hardcoded name and AES key.
2. 2 Read the following values from the decrypted payload config file in this order:
    - AES key for APK payload decryption
    - Class and method names for reflection calls to the payload
    - MD5 for APK payload integrity check
    - Number and names of the split APK payload parts
3. 3 Decrypt the APK payload header hardcoded in the first stage with the AES key from the payload config. Write it to the APK payload file.
4. 4 Using decrypted names of the split payload parts, decrypt their content and append them to the APK payload file one by one.
5. 5 Check the integrity of the resulting APK payload file by comparing with the MD5 value decrypted from the payload config.
6. 6 Load and run the APK payload.

The following reversed code fragment represents the actual payload decryption process.

```
byte[] v2 = g.aesDecrypt(c.openAssetFile(arg12, e.name), e.aes_key_asset);
if(v2 == null) {
    return v1;
}

b asset_config = new b(v2);
String aes_key_apk = asset_config.readString();
String class_name = asset_config.readString();
String method_name = asset_config.readString();
String md5_apk = asset_config.readString();
int number_parts_apk = asset_config.a();
String[] names_parts_apk = new String[number_parts_apk];
int v2_1;
for(v2_1 = 0; v2_1 < number_parts_apk; ++v2_1) {
    names_parts_apk[v2_1] = asset_config.readString();
}

asset_config.closeFile();
File apk_file = new File(arg12.getFilesDir(), Long.toString(System.currentTimeMillis()) + ".apk");
FileOutputStream v3_1 = new FileOutputStream(apk_file);
v3_1.write(g.aesDecrypt(e.b, aes_key_apk));
number_parts_apk = names_parts_apk.length;
while(v0 < number_parts_apk) {
    v3_1.write(g.aesDecrypt(c.openAssetFile(arg12, names_parts_apk[v0]), aes_key_apk));
    ++v0;
}

v3_1.flush();
v3_1.close();
if(!f.calculateMD5(apk_file.getAbsolutePath()).equals(md5_apk)) {
    return v1;
}

File loader_dir = arg12.getDir(Long.toString(System.currentTimeMillis()), 0);
loader_dir.mkdirs();
DexClassLoader v3_2 = new DexClassLoader(apk_file.getAbsolutePath(), loader_dir.getAbsolutePath(),
        null, arg12.getClassLoader());
a.remove(apk_file);
a.remove(loader_dir);
v0_1 = new a(v3_2.loadClass(class_name).getConstructor(Context.class, Service.class).newInstance(
        arg12, arg13), method_name);
```

payload config decryption

read config values

decrypt and write header

decrypt and write splitted parts

integrity check

load apk payload

remove payload files

payload reflection call

Each Version 3 payload has the same package name, “com.android.process.gpsp”, and is signed with the same certificate (CN=Ventoplex), used to sign some of the Version 2 payloads.

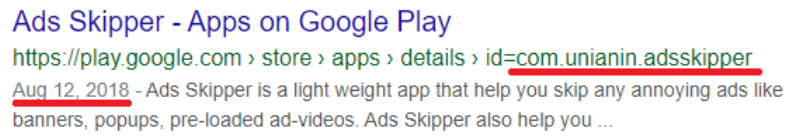
The only developer version stamp that we have found in Version 3 payloads is “10.2.98”.

Another notable finding is the 243e2c6433815f2ecc204ada4821e7d6 sample, which we believe belongs to a Version 3 payload. However, no related dropper has been spotted in the wild, and unlike the other payloads, it is signed with a debug certificate and not obfuscated at all, revealing all variable/class/method names and even BuildConfig values. Our guess that this is a debug developer version that somehow got leaked.

As a conclusion to this technical review, it is worth saying that all payloads across the different versions, even Version 1, which is in fact a clear payload without a dropper, share a code structure and locations where sensitive strings, such as C2 addresses, are stored.

Spread

The main spreading vector used by the threat actors is distribution through application marketplaces. Apart from the com.zimice.browserturbo, which we have reported to Google, and com.physlane.opengl, reported by Dr. Web, we have observed tracks indicating that many malicious applications were deployed to Google Play in the past and have now been removed.



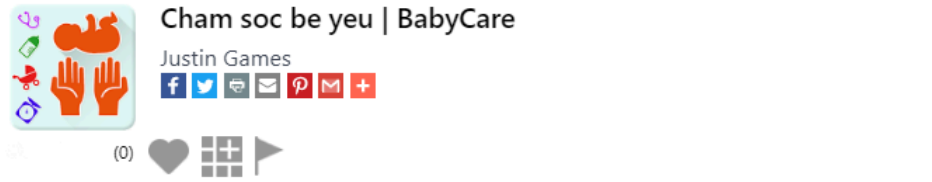
These search results contain a link to already-removed malware in Google Play

Some of the applications whose appearance in Google Play we can confirm.

Package name	Google Play persistence date (at least)
com.zimice.browserturbo	2019-11-06
com.physlane.opengl	2019-07-10

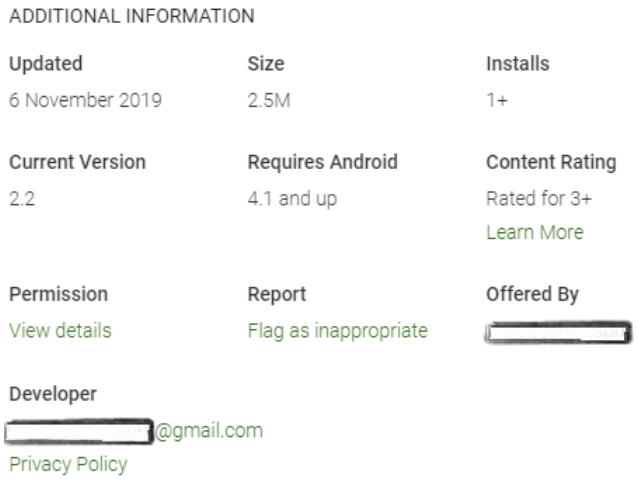
com.unianin.adsskipper	2018-12-26
com.codedexon.prayerbook	2018-08-20
com.luxury.BeerAddress	2018-08-20
com.luxury.BiFinBall	2018-08-20
com.zonjob.browsercleaner	2018-08-20
com.linevialab.ffont	2018-08-20

Besides, we have identified multiple third-party marketplaces that, unlike Google Play, still host the malicious applications, such as https://apkcombo[.]com, https://apk[.]support/, https://apkpure[.]com, https://apkpourandroid[.]com and many others.



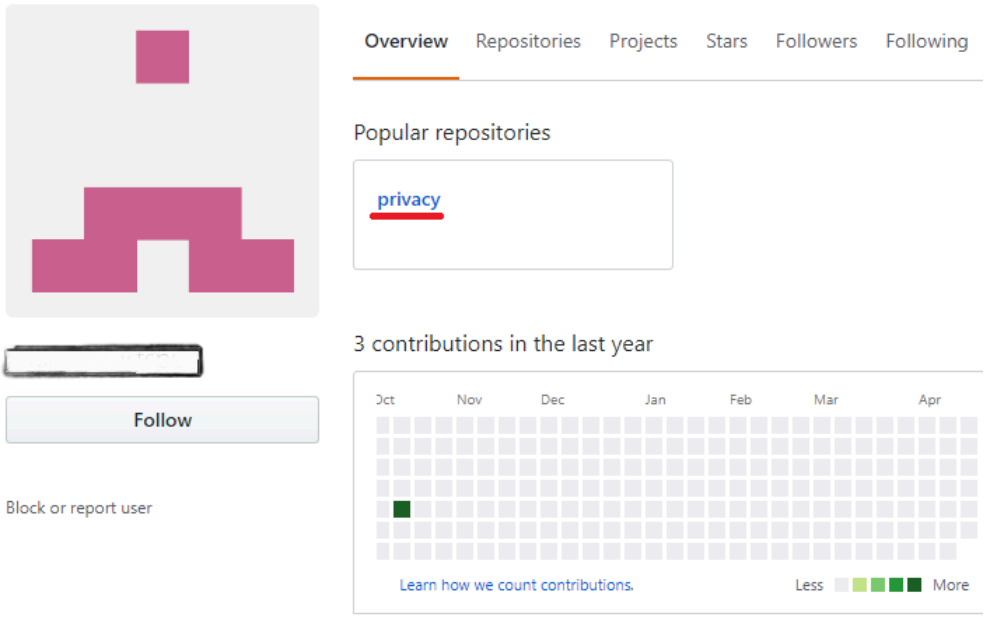
Example of a malicious application with a description in Vietnamese that is still available in a third-party marketplace (hxxps://androidappsapk[.]co/detail-cham-soc-be-yeu-babycare/)

In nearly every case of malware deployment, the threat actors try to build a fake developer profile by creating a Github account that contains only a fake end-user license agreement (EULA). An example is the one below, reported by us to Google.



This Google Play page contains a fake developer email

Here is a related Github account with the same handle, registered on October 17, 2019.

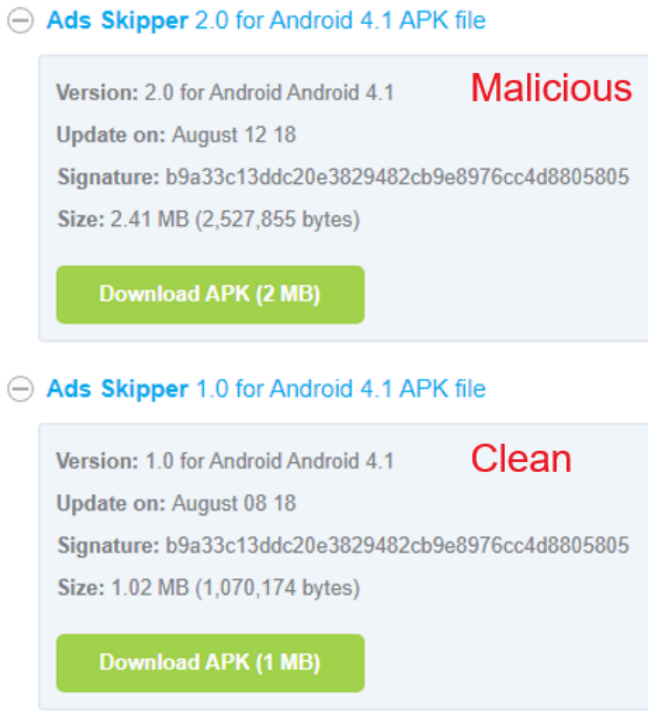


A Github profile that is part of the fake developer identity

The account contains only one report with one file described as some type of EULA.

During our extensive investigation, we spotted a certain tactic often used by the threat actors for distributing their malware. The initial versions of applications uploaded to app marketplaces did not contain any malicious payloads or code for dropping a payload. These versions were accepted because they contained nothing suspicious, but follow-up versions were updated with both malicious payloads and code to drop and execute these payloads. We were able to confirm this behavior in all of the samples, and we were able to find two versions of the applications, with and without a payload.

An example of this behavior can be seen in Ads Skipper (https://apkpure[.]ai/ads-skipper), in ApkPure.

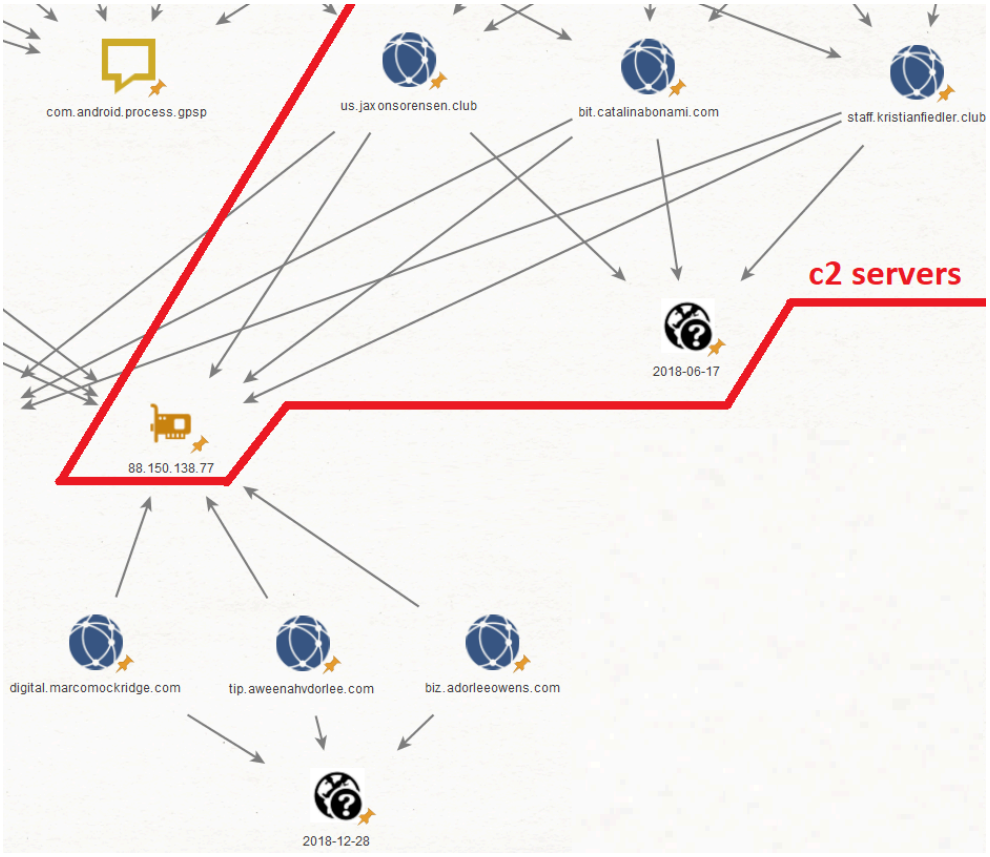


Versions of Ads Skipper with (v. 2.0) and without (v. 1.0) a malicious payload in ApkPure

Third-party marketplaces like those mentioned in the table above often serve as a mirror for Google Play: they simply copy applications and metadata from Google Play to their own servers. Therefore, it is safe to assume that the samples listed in the table were copied from Google Play as well.

Infrastructure

While analyzing the C2 server infrastructure, we quickly identified multiple domains that shared similarities with previous ones but were not linked to any known malware samples. This allowed us to uncover more pieces of the attackers' infrastructure.



Example of related infrastructure

Tracking PhantomLance's old infrastructure, which dated back four years, we noticed that the expired domain names had been extended. The maintenance suggested that the infrastructure might be used again in the future.

Domain	Registered	Last updated
osloger[.]biz	2015-12-09	2019-12-01
log4jv[.]info	2015-12-09	2019-11-26
sqllitlever[.]info	2015-12-09	2019-11-26
anofrio[.]com	2017-05-16	2020-03-30
anaehler[.]com	2017-05-16	2020-03-30
viodger[.]com	2017-05-16	2020-04-07

The PhantomLance TTPs indicate that samples are configured only with subdomains as C2 servers, while most, but not all, parent domains do not have their own IP resolution. We checked the ones that did have a valid resolution and found that they all resolved to the same IP address: 188.166.203[.]57. It belongs to the DigitalOcean cloud infrastructure provider and, according to Domaintools, hosts a total of 129 websites.

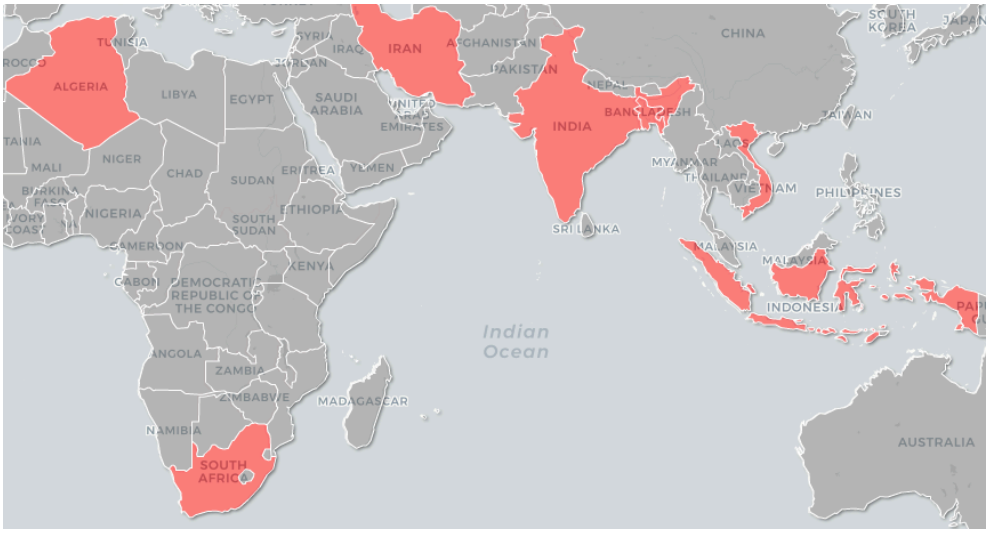
Looking up records for this IP address in our passive DNS database suggests that a few dozen of these websites are legitimate, as well as the aforementioned PhantomLance domains and two more interesting overlaps with OceanLotus infrastructure:

- browsersyn[.]com: known domain used as a C2 in a previously publicly reported sample (MD5: b1990e19efaf88206f7bffe9df0d9419) considered by the industry to be the OceanLotus APT.
- cerisecaird[.]com: privately received information indicates that this domain is related to OceanLotus as well.

Host	IP	Lastseen	Tags Host IP
elizongham.com	188.166.203.57	2019-08-31 00:53:29	PhantomLance
georgiatheiss.com	188.166.203.57	2019-11-24 19:45:44	PhantomLance
lancebarkerwa.com	188.166.203.57	2019-07-28 21:09:31	PhantomLance
meorgirilbert.com	188.166.203.57	2019-11-24 19:45:41	PhantomLance
remaariegarcia.com	188.166.203.57	2019-11-24 19:43:22	PhantomLance
senapusmireault.com	188.166.203.57	2019-08-19 11:03:54	PhantomLance
stralisemariegar.com	188.166.203.57	2019-11-24 19:42:14	PhantomLance
stralisemariegar.com	188.166.203.57	2019-11-24 19:42:14	PhantomLance
www.lancebarkerwa.com	188.166.203.57	2019-07-15 18:18:02	PhantomLance
social.cerisecaird.com	188.166.203.57	2020-04-06 01:49:23	OceanLotus
e.browsersyn.com	188.166.203.57	2019-03-19 16:32:54	OceanLotus

Victimology

We have observed around 300 infection attacks on Android devices in India, Vietnam, Bangladesh, Indonesia, etc. starting in 2016. Below is a rough cartographic representation of countries with top attempted attacks.



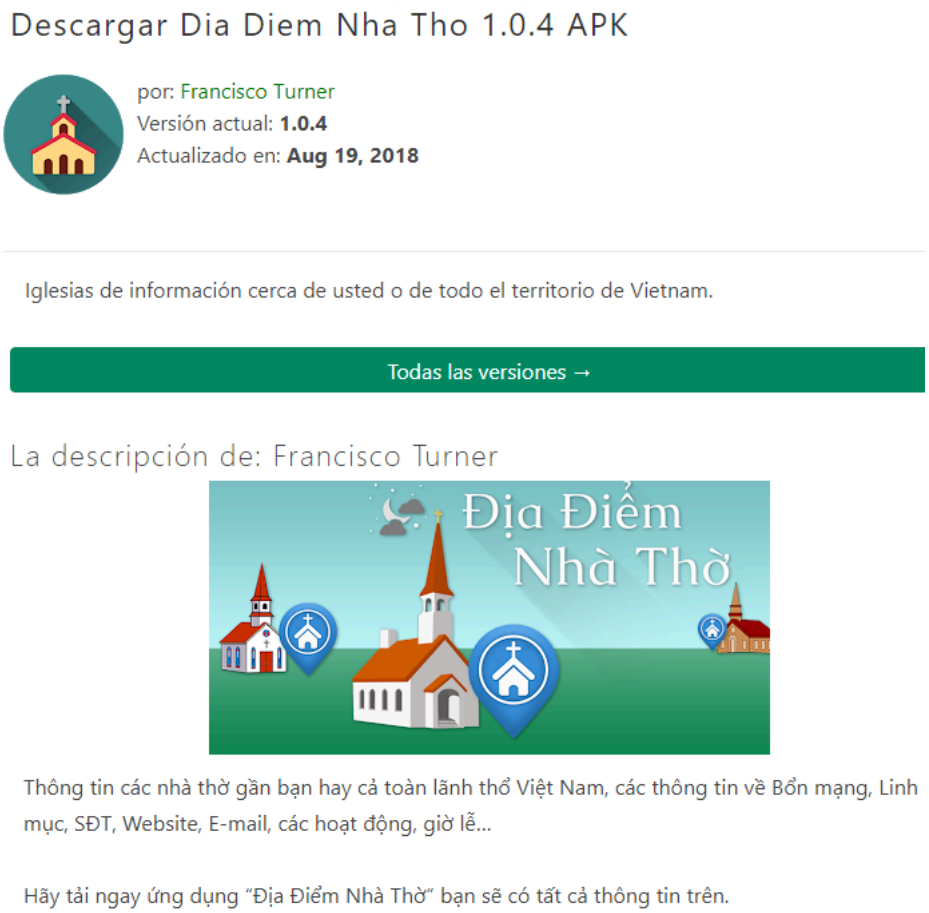
We have also seen a number of detections in Nepal, Myanmar and Malaysia. As you can see, this part of South Asia seems to be targeted by the actors the most.

Note that due to the chosen distribution vector (publication of malicious samples on publicly available application stores), there should be secondary infection of random victims not directly related to the actors' interests.

To get more details on targeted victims, we looked at the types of applications that the malware mimicked. Apart from common luring applications, such as Flash plugins, cleaners and updaters, there were those that specifically targeted Vietnam.

- luxury.BeerAddress – "Tìm quan nhau | Tìm quán nhậu" ("Find each other | Find pubs" in Vietnamese). An application for finding the nearest pub in Vietnam.

- codedexon.churchaddress – “Địa Điểm Nhà Thờ” (“Church Place”)



  Publisher description (hxxps://apk.support/app-en/com.codedexon.churchaddress) translated from Vietnamese: *Information about churches near you or the whole of Vietnam, information about patronies, priests, phone numbers, websites, email, activities, holidays…*

- bulknewsexpress.news – “Tin 247 – Đọc Báo Hàng Ngày” (“Read Daily Newspaper”)

Mimics the Vietnamese www.tin247.com mobile news application.

Overlaps with previous campaigns

In this section, we provide a correlation of PhantomLance’s activity with previously reported campaigns related to the OceanLotus APT.

OceanLotus Android campaign in 2014-2017

In May 2019, Antiy Labs published a report (https://www.antiy.net/p/analysis-of-the-attack-of-mobile-devices-by-oceanlotus/) in which they described an Android malware campaign, claiming that it was related to OceanLotus APT. We checked the provided indicators using information from our telemetry and found that the very first tracks of these samples date back to December 2014.

It is important to note that according to our detection statistics, the majority of users affected by this campaign were located in Vietnam, with the exception of a small number of individuals located in China.

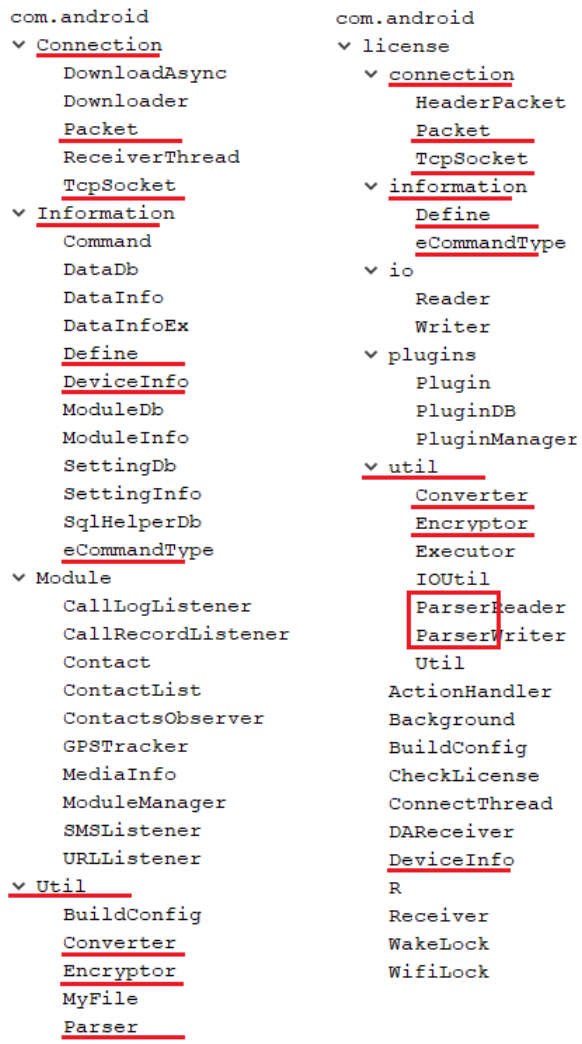
The main infection vector seems to be links to malicious applications hosted on third-party websites, possibly distributed via SMS or email spearphishing attacks. Examples below.

Referring URL for victim	**Malware URL**	**I**
hxxp://download.com[.]vn/android/download/nhaccuatui-downloader/31798	hxxp://113.171.224.175/videoplayer/NhacCuaTuiDownloader[.]apk	2

	hxxp://nhaccuatui.android.zyngacdn.com/NhacCuaTuiDownloader[.]apk	
hxxp://www.mediafire.com/file/1elber8zl34tag4/framaroot-xpro[.]apk	hxxp://download1825.mediafire.com/tyxddh46orzg/1elber8zl34tag4/framaroot-xpro[.]apk	

The latest registered malware download event occurred in December 2017. We observed a small amount of activity in 2018, but judging by the volume of hosted malware and the number of detections we observed, the main campaign took place from late 2014 to 2017.

To best visualize the similarities we discovered, we made a code structure comparison of the sample from the old reported OceanLotus Android campaign (MD5: 0e7c2adda3bc65242a365ef72b91f3a8) and the only unobfuscated (probably a developer version) PhantomLance payload v3 (MD5: 243e2c6433815f2ecc204ada4821e7d6).



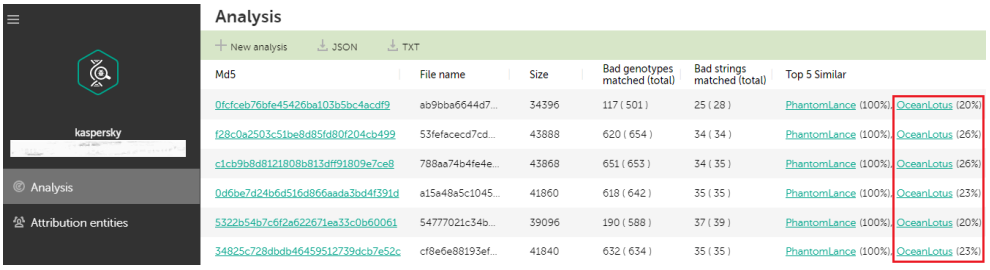
Code structure comparison of a sample linked to OceanLotus and PhantomLance payload v3.

Despite the multiple differences, we observed a similar pattern used in malware implementation. It seems that the developers have renamed “module” to “plugin”, but the meaning remains the same. Overlapping classes look quite similar and have the same functionality. For example, here is a comparison of the methods contained in the Parser classes.

Parser from 0e7c2adda3bc65242a365ef72b91f3a8	**ParserWriter/Reader from 243e2c6433815f2ecc204ada4821e7d6**
public void appendBoolean(boolean f)	public void appendBoolean(boolean value)
public void appendByte(byte data)	public void appendByte(byte value)
public void appendBytes(byte[] data)	public void appendBytes(byte[] value)

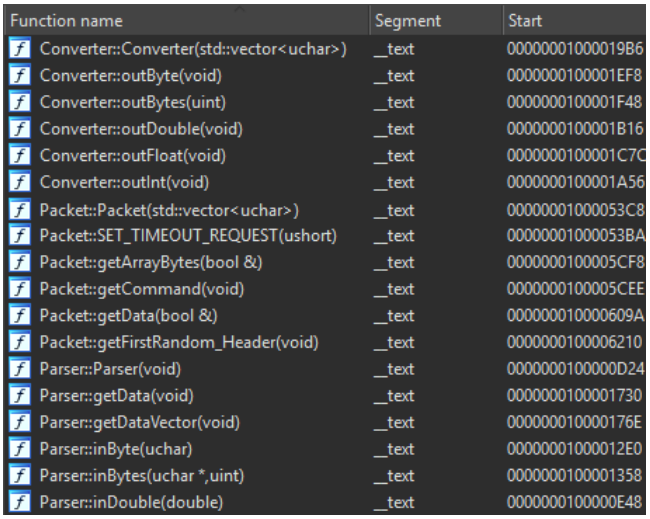
public void appendDouble(double val)	public void appendDouble(double value)
public void appendInt(int val)	public void appendInt(int value)
public void appendLong(long val)	public void appendLong(long value)
	private void appendNumber(Object value)
public void appendShort(short val)	public void appendShort(short value)
public void appendString(String str)	public void appendString(String value)
public byte[] getContents()	public byte[] getContents()
public void appendFloat(float val)	
public boolean getBoolean()	public boolean getBoolean()
public byte getByte()	public byte getByte()
public byte[] getBytes()	public byte[] getBytes()
public double getDouble()	public double getDouble()
public float getFloat()	
public int getInt()	public int getInt()
public long getLong()	public long getLong()
public short getShort()	public short getShort()
	byte getSignal()
public String getString()	public String getString()
	getStringOfNumber()

Using our malware attribution technology, we can see that the PhantomLance payloads are at least 20% similar to the ones from the old OceanLotus Android campaign.



OceanLotus macOS backdoors

There are multiple public reports of macOS backdoors linked by the industry to OceanLotus. We examined these in order to find possible overlaps, with the caveat that it was really difficult to compare malware implemented for two completely different platforms, since two different programming languages were obviously used for the implementation process. However, during the analysis of the macOS payload (MD5: 306d3ed0a7c899b5ef9d0e3c91f05193) dated early 2018, we were able to catch a few minor tracks of the code pattern used in the Android malware implementation described above. In particular, three out of seven main classes had the same names and similar functionality: “Converter”, “Packet” and “Parser”.



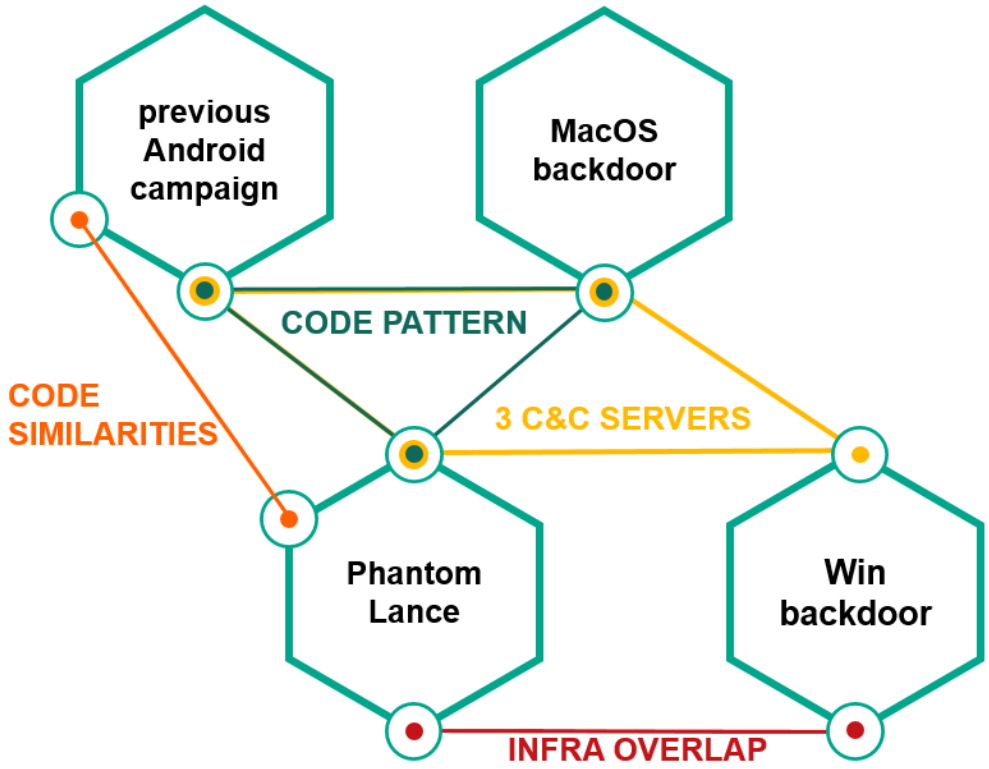
Function name	Segment	Start
Converter::Converter(std::vector<uchar>)	_text	00000001000019B6
Converter::outByte(void)	_text	0000000100001EF8
Converter::outBytes(uint)	_text	0000000100001F48
Converter::outDouble(void)	_text	0000000100001B16
Converter::outFloat(void)	_text	0000000100001C7C
Converter::outInt(void)	_text	0000000100001A56
Packet::Packet(std::vector<uchar>)	_text	00000001000053C8
Packet::SET_TIMEOUT_REQUEST(ushort)	_text	00000001000053BA
Packet::getArrayBytes(bool &)	_text	0000000100005CF8
Packet::getCommand(void)	_text	0000000100005CEE
Packet::getData(bool &)	_text	000000010000609A
Packet::getFirstRandom_Header(void)	_text	0000000100006210
Parser::Parser(void)	_text	0000000100000D24
Parser::getData(void)	_text	0000000100001730
Parser::getDataVector(void)	_text	000000010000176E
Parser::inByte(uchar)	_text	00000001000012E0
Parser::inBytes(uchar *,uint)	_text	0000000100001358
Parser::inDouble(double)	_text	0000000100000E48

Summary of overlaps

Another notable attribution token that applies to most of OceanLotus malware across platforms is usage of three redundant, different C2 servers by each sample, mostly subdomains. Below is an example of this from the samples examined above and OceanLotus Windows malware described in our private report.

MD5	C2 servers	Description
0d5c03da348dce513bf575545493f3e3	mine.remaariegarcia[.]com egg.stralisemariegar[.]com api.anaehler[.]com	PhantomLance Android
d1eb52ef6c2445c848157beaba54044f	sadma.knrowz[.]com ckoen.dmkatti[.]com itpk.mostmkru[.]com	OceanLotus Android campaign 2014-2017
306d3ed0a7c899b5ef9d0e3c91f05193	ssl.arkouthrie[.]com s3.hiahornber[.]com widget.shoreoa[.]com	OceanLotus MacOS backdoor
51f9a7d4263b3a565dec7083ca00340f	ps.andreagahuvrauvin[.]com paste.christienollmache[.]xyz att.illagedrivestralia[.]xyz	OceanLotus Windows backdoor

Based on the complete analysis of previous campaigns, with the actors' interests in victims located in Vietnam, infrastructure overlaps between PhantomLance and OceanLotus for Windows, multiple code similarities between an old Android campaign and MacOS backdoors, we attribute the set of the Android activity (campaign 2014-2017 and PhantomLance) to OceanLotus with medium confidence.



Considering the timeline of the Android campaigns, we believe that the activity reported by Antiy Labs is a previous campaign that was conducted by OceanLotus until 2017, and PhantomLance is a successor, active since 2016.

In summarizing the results of this research, we are able to assess the scope and evolution of the actors' Android set of activity, operating for almost six years.

IOC

Kaspersky Lab products verdicts

PhantomLance

HEUR:Backdoor.AndroidOS.PhantomLance.*
HEUR:Trojan-Dropper.AndroidOS.Dnolder.*

Android campaign linked to OceanLotus (2014-2017)

HEUR:Trojan.AndroidOS.Agent.eu
HEUR:Trojan.AndroidOS.Agent.vg
HEUR:Trojan-Downloader.AndroidOS.Agent.gv

macOS campaign linked to OceanLotus

HEUR:Backdoor.OSX.OceanLotus.*

MD5

PhantomLance malware

2e06bbc26611305b28b40349a600f95c
b1990e19efaf88206f7bffe9df0d9419
7048d56d923e049ca7f3d97fb5ba9812
e648a2cc826707aec33208408b882e31
3285ae59877c6241200f784b62531694
8d5c64fdaae76bb74831c0543a7865c3
6bf9b834d841b13348851f2dc033773e
0d5c03da348dce513bf575545493f3e3

0e7c2adda3bc65242a365ef72b91f3a8
a795f662d10040728e916e1fd7570c1d
d23472f47833049034011cad68958b46
8b35b3956078fc28e5709c5439e4dcb0
af44bb0dd464680395230ade0d6414cd
65d399e6a77acf7e63ba771877f96f8e
79f06cb9281177a51278b2a33090c867
b107c35b4ca3e549bdf102de918749ba
83cd59e3ed1ba15f7a8cadfe9183e156
c399d93146f3d12feb32da23b75304ba
83c423c36ecda310375e8a1f4348a35e
94a3ca93f1500b5bd7fd020569e46589
54777021c34b0aed226145fde8424991
872a3dd2cd5e01633b57fa5b9ac4648d
243e2c6433815f2ecc204ada4821e7d6

PhantomLance payload-free versions

a330456d7ca25c88060dc158049f3298
a097b8d49386c8aab0bb38bbfdf315b2
7285f44fa75c3c7a27bbb4870fc0cdca
b4706f171cf98742413d642b6ae728dc
8008bedaaebc1284b1b834c5fd9a7a71
0e7b59b601a1c7ecd6f2f54b5cd8416a

Android campaign 2014-2017

0e7c2adda3bc65242a365ef72b91f3a8
50bfd62721b4f3813c2d20b59642f022
5079cb166df41233a1017d5e0150c17a
810ef71bb52ea5c3cfe58b8e003520dc
c630ab7b51f0c0fa38a4a0f45c793e24
ce5bae8714ddfca9eb3bb24ee60f042d
d61c18e577cfc046a6252775da12294f
fe15c0eacdbf5a46bc9b2af9c551f86a
07e01c2fa020724887fc39e5c97eccee
2e49775599942815ab84d9de13e338b3
315f8e3da94920248676b095786e26ad
641f0cc057e2ab43f5444c5547e80976

Domains and IP addresses

PhantomLance

mine.remaariegarcia[.]com
egg.stralisemariegar[.]com
api.anaehler[.]com
cloud.anofrio[.]com
video.viodger[.]com
term.ursulapaulet[.]com
inc.graceneufville[.]com
log.osloger[.]biz
file.log4jv[.]info
news.sqllitlever[.]info
us.jaxonsorensen[.]club
staff.kristianfiedler[.]club
bit.catalinabonami[.]com
hr.halettebiermann[.]com
cyn.ettebiermahalet[.]com

Android campaign 2014-2017

mtk.baimind[.]com
ming.chujong[.]com
mokkha.goongnam[.]com
ckoen.dmkatti[.]com
sadma.knrowz[.]com
itpk.mostmkru[.]com
aki.viperse[.]com
game2015[.]net
taiphanmemfacebookmoi[.]info
nhaccuatui.android.zyngacdn[.]com
quam.viperse[.]com
jang.goongnam[.]com
