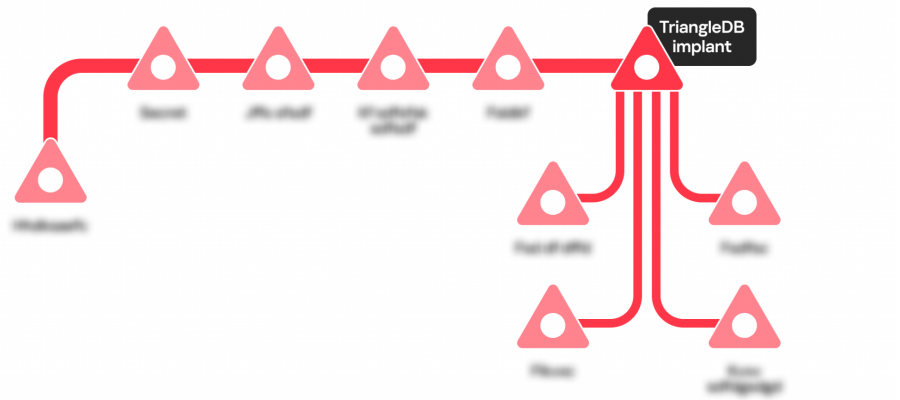
Dissecting TriangleDB, a Triangulation spyware implant

Authors

- Georgy Kucherin
- Leonid Bezvershenko
- Igor Kuznetsov

Over the years, there have been multiple cases when iOS devices were infected with targeted spyware such as Pegasus, Predator, Reign and others. Often, the process of infecting a device involves launching a chain of different exploits, e.g. for escaping the iMessage sandbox while processing a malicious attachment, and for getting root privileges through a vulnerability in the kernel. Due to this granularity, discovering one exploit in the chain often does not result in retrieving the rest of the chain and obtaining the final spyware payload. For example, in 2021, analysis of iTunes backups helped to discover an attachment containing the FORCEDENTRY exploit. However, during post-exploitation, the malicious code downloaded a payload from a remote server that was not accessible at the time of analysis. Consequently, the analysts lost "the ability to follow the exploit."

In researching Operation Triangulation, we set ourselves the goal to retrieve as many parts of the exploitation chain as possible. It took about half a year to accomplish that goal, and, after the collection of the chain had been completed, we started an in-depth analysis of the discovered stages. As of now, we have finished analyzing the spyware implant and are ready to share the details.



The Operation Triangulation infection chain

The implant, which we dubbed TriangleDB, is deployed after the attackers obtain root privileges on the target iOS device by exploiting a kernel vulnerability. It is deployed in memory, meaning that all traces of the implant are lost when the device gets rebooted. Therefore, if the victim reboots their device, the attackers have to reinfect it by sending an iMessage with a malicious attachment, thus launching the whole exploitation chain again. In case no reboot occurs, the implant uninstalls itself after 30 days, unless this period is extended by the attackers.

Meet TriangleDB

The TriangleDB implant is coded using Objective-C, a programming language that preserves names of members and methods assigned by the developer. In the implant's binary, method names are not obfuscated; however, names of class members are uninformative acronyms, which makes it difficult to guess their meaning:

Class method examples	Class member examples
-[CRConfig populateWithFieldsMacOSOnly]	NSString *pubKI;
-[CRConfig populateWithSysInfo]	NSData *pubK;
-[CRConfig extendFor:]	signed __int64 iDa;
-[CRConfig getCInfoForDump]	signed __int64 uD;
+[CRConfig sharedInstance]	NSString *deN;
+[CRConfig unmungeHexString:]	NSSTring *prT;
-[CRConfig init]	NSString *seN;
-[CRConfig getBuildArchitecture]	NSString *uDI;
-[CRConfig cLS]	NSString *iME;
-[CRConfig setVersion]	NSString *meI;
-[CRConfig swapLpServerType]	NSString *osV;
-[CRConfig setLpServerType:]	CRPwrInfo *pwI;

In some cases, it is possible to guess what the acronyms mean. For example, osV is the iOS version, and iME contains the device's IMEI.

The strings in the implant are HEX-encoded and encrypted with rolling XOR:

```
1  id +[CRConfig unmungeHexString:](id a1, SEL a2, id stringToDecrypt) {
2    // code omitted
3    while (1) {
4  hexByte[0] = stringBytes[i];
5  hexByte[1] = stringBytes[i + 1];
6  encryptedByte = strtoul(hexByte, &__endptr, 16);
7  if (__endptr == hexByte)
8          break;
9  i += 2LL;
10 if (j)
11       decryptedString[j] = encryptedByte ^ previousByte;
12 else
13       decryptedString[0] = encryptedByte;
14 ++j;
15 previousByte = encryptedByte;
16 if (i >= stringLength)
17         break;
18  }
19  decryptedString[j] = 0;
```

```
20   // code omitted
21 }
```

The rolling XOR algorithm implemented in the implant for string decryption

C2 communications

Once the implant launches, it starts communicating with the C2 server, using the Protobuf library for exchanging data. The configuration of the implant contains two servers: the primary and the fallback (contained in the IS and ISf configuration fields). Normally, the implant uses the primary server, and, in case of an error, it switches to the fallback server by invoking the -[CRConfig swapLpServerType:] method.

Additionally, the sent and received messages are encrypted with symmetric (3DES) and asymmetric (RSA) cryptography. All messages are exchanged via the HTTPS protocol in POST requests, with the cookie having the key g and a value that is a digit string from the pubKI configuration parameter.

The implant periodically sends heartbeat beacons that contain system information, including the implant version, device identifiers (IMEI, MEID, serial number, etc.) and the configuration of the update daemon (whether automatic downloads and installations of updates are enabled).

```
maR: "1"
miR: "7"
paR: "0"
buR: "5"
eBU: "[redacted]"
eBA: "arm64e"
eCU: "[redacted]"
eIU: "[redacted]"
eSU: "[redacted]"
pI: "[redacted]"
oF: "iPhone"
oN: "iOS"
oV: "15.3.1"
```

Heartbeat beacon snippet, implant v1.7.0.5 running on iOS 15.3.1

TriangleDB commands

The C2 server responds to heartbeat messages with commands. Commands are transferred as Protobuf messages that have type names starting with CRX. The meaning of these names is obscure: for example, the command listing directories is called CRXShowTables, and changing C2 server addresses is handled by the command CRXConfigureDBServer. In total, the implant we analyzed has 24 commands designed for:

- Interacting with the filesystem (creation, modification, exfiltration and removal of files);
- Interacting with processes (listing and terminating them);
- Dumping the victim's keychain items, which can be useful for harvesting victim credentials;
- Monitoring the victim's geolocation;
- Running additional modules, which are Mach-O executables loaded by the implant. These executables are reflectively loaded, with their binaries stored only in memory.

One of the interesting commands we discovered is called CRXPollRecords. It monitors changes in folders, looking for modified files that have names matching specified regular expressions. Change monitoring is handled by obtaining a Unix file descriptor of the directory and assigning a vnode event handler to it. Whenever the implant gets notified of a change, the event handler searches for modified files that match the regex provided by the attacker. Such files are then scheduled for uploading to the C2 server.

The parameters of this command are as follows:

Parameter name	**Parameter description**
p	Directory path
m	Filename regex
sDC	Specifies whether the command should exfiltrate files that were modified

	before monitoring started.
eWo	Specifies whether file contents should be exfiltrated only via Wi-Fi.

Below, we describe the implant's commands, specifying the developer-assigned command names along with their numerical identifiers when possible.

Command ID	Developer-assigned name	Description
0xFEED	CRXBlank	No operation
0xF001	N/A	Uninstalls the implant by terminating its process.
0xF301	CRXPause	Makes the implant sleep for a specified number of seconds.
0xFE01	N/A	Sleeps for a pseudorandom time defined by the configuration parameters caS and caP. The sleeping time is chosen between caP – caS and caP + caS.
0xFB01	CRXForward	Changes the caP configuration value for the 0xFE01 command.
0xFB02	CRXFastForward	Changes the caS configuration value for the 0xFE01 command.
0xF201	CRXConfigureDBServer	Changes the addresses of the primary and fallback C2 servers.
0xF403	CRXUpdateConfigInfo	Changes the implant's configuration parameters. The arguments of this command contain the identifier of the parameter to be changed and its new value. Note that the parameter identifiers are number strings, such as "nineteen" or "twentyone".
0xF101	CRXExtendTimeout	Extends the implant lifetime by a specified number of seconds (the default implant lifetime is 30 days).
0xF601	CRXQueryShowTables	Obtains a listing of a specified directory with the fts API.
0xF801	CRXFetchRecordInfo	Retrieves metadata (attributes, permissions, size, creation, modification and access timestamps) of a given file.
0xF501	CRXFetchRecord	Retrieves contents of a specified file.
0xFC10	CRXPollRecords	Starts monitoring a directory for files whose names match a specified regex.
0xFC11	CRXStopPollingRecords	Stops execution of the CRXPollRecords command.
0xFC01	CRXFetchMatchingRecords	Retrieves files that match a specified regex.
0xF901	CRXUpdateRecord	Depending on the command's iM argument, either writes data to a file or adds a new module to the implant.
0xFA02	CRXRunRecord	Launches a module with a specified name by reflectively loading its Mach-O executable.
0xF902	CRXUpdateRunRecord	Adds a new module to the implant and launches it.
0xFA01	CRXDeleteRecord	Depending on the command's arguments, either removes an implant module or deletes a file with a specified name.
0xF402	CRXGetSchemas	Retrieves a list of running processes.
0xFB44	CRXPurgeRecord	Kills a process with a specified PID, either with SIGKILL or SIGSTOP, depending on the command's arguments.
0xFD01	N/A	Retrieves information about installed iOS applications
0xFB03	CRXGetIndexesV2	Retrieves keychain entries of the infected device. It starts monitoring the screen lock state, and, when the device is unlocked, dumps keychain items from the genp (generic passwords), inet (Internet passwords), keys and cert tables (certificates, keys and digital identity) from the /private/var/Keychains/keychain-2.db database. Note here that the implant's code can work with different keychain versions, starting from the ones used in iOS 4.
0xF401	N/A	Retrieves the victim's location information: coordinates, altitude, bearing (the direction in which the device is moving) and speed. By default, this command works only if the device screen is off. However, the implant operator can override this restriction with a configuration flag.

Odd findings

While researching the TriangleDB implant, we found a lot of curious details:

- The developers refer to string decryption as "unmunging" (as the method performing string decryption is named +[CRConfig unmungeHexString:]);
- Throughout the code, we observed that different entities were given names from database terminology, which is the reason why we dubbed the implant TriangleDB:

Entity	Developer-used terminology for the entity
Directory	Table
File Implant module	Record
Process	Schema
Keychain entry	Index, row
C2 server	DB Server

Geolocation information	DB Status
Heartbeat	Diagnostic data
Process of exchanging data with C2 server	Transaction
Request to C2 server	Query
iOS application	Operation

- While analyzing TriangleDB, we found that the class CRConfig (used to store the implant's configuration) has a method named populateWithFieldsMacOSOnly. This method is not called anywhere in the iOS implant; however, its existence means that macOS devices can also be targeted with a similar implant;
- The implant requests multiple entitlements (permissions) from the operating system. Some of them are not used in the code, such as access to camera, microphone and address book, or interaction with devices via Bluetooth. Thus, functionalities granted by these entitlements may be implemented in modules.

To be continued

That's it for TriangleDB, a sophisticated implant for iOS containing multiple oddities. We are continuing to analyze the campaign, and will keep you updated with all details about this sophisticated attack.

TriangleDB indicators of compromise

MD5 063db86f015fe99fdd821b251f14446d
SHA-1 1a321b77be6a523ddde4661a5725043aba0f037f
SHA-256 fd9e97cfb55f9cfb5d3e1388f712edd952d902f23a583826ebe55e9e322f730f