Storm Cloud on the Horizon: GIMMICK Malware Strikes at macOS

By mindgrub

Published: 2022-03-22 · Archived: 2026-04-05 13:24:37 UTC

In late 2021, Volexity discovered an intrusion in an environment monitored as part of its Network Security Monitoring service. Volexity detected a system running **frp**, otherwise known as fast reverse proxy, and subsequently detected internal port scanning shortly afterward. This traffic was determined to be unauthorized and the system, a MacBook Pro running macOS 11.6 (Big Sur), was isolated for further forensic analysis. Volexity was able to run Surge Collect to acquire system memory (RAM) and select files of interest from the machine for analysis. This led to the discovery of a macOS variant of a malware implant Volexity calls **GIMMICK**. Volexity has encountered Windows versions of the malware family on several previous occasions.

GIMMICK is used in targeted attacks by Storm Cloud, a Chinese espionage threat actor known to attack organizations across Asia. It is a feature-rich, multi-platform malware family that uses public cloud hosting services (such as Google Drive) for command-and-control (C2) channels. The newly identified macOS variant is written primarily in Objective C, with Windows versions written in both .NET and Delphi. Despite core differences in programming languages used and operating systems targeted, Volexity tracks the malware under the same name due to shared C2 architecture, file paths, and behavioral patterns used by all variants.

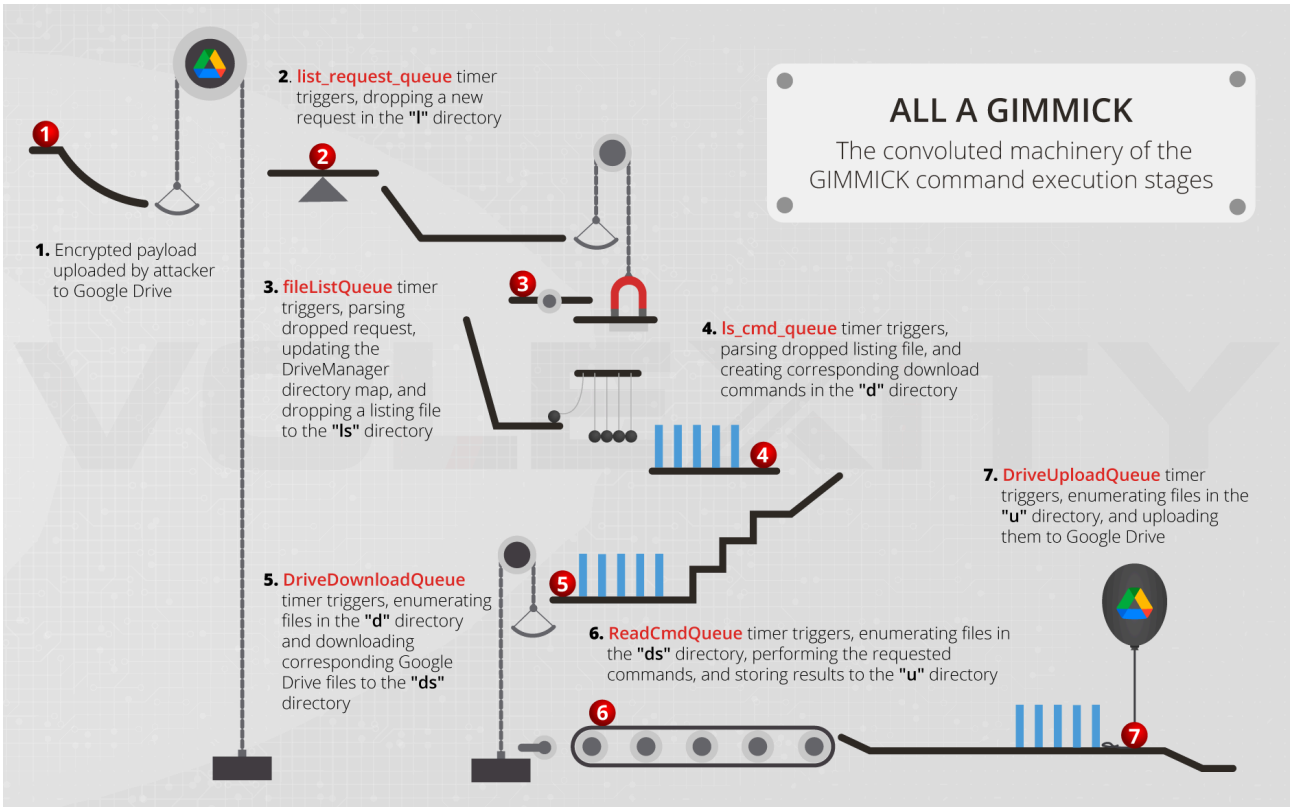

Figure 1. The GIMMICK workflow

This blog post provides an in-depth analysis of the macOS variant of GIMMICK, but also demonstrates the features and characteristics of the Windows variant. Volexity discovered this sample through memory analysis of the compromised system and was able to recover the implant from both memory and disk. The file name and install path were unique to the victim system and had been configured in a manner designed to blend in with job functions of the user. Additionally, GIMMICK was configured to only communicate with its Google Drive-based C2 server on working days in order to further blend in with network traffic in the target environment.

The SHA1 hash of the file Volexity was able to obtain from disk was “fe3a3e65b86d2b07654f9a6104c8cb392c88b7e8”.

Volexity worked closely with Apple to add protections for the GIMMICK malware across their userbase. On March 17, 2022, Apple pushed new signatures to XProtect and MRT to block and remove GIMMICK. Though on by default, users can confirm they are automatically protected by verifying the “Install system data files and security updates” box is checked in their Settings (instructions can be found here).

Startup and Initialization

On macOS, GIMMICK was found to support being launched as a daemon on the system or by a user. Should GIMMICK be launched directly by a user, rather than a daemon, it will install itself as a launch agent by dropping a PLIST file with contents, similar to that shown below, to */Users/<username>/Library/LaunchAgents*. The name of the binary, PLIST, and agent will vary per sample. In the case observed by Volexity, the implant was customized to imitate an application commonly launched by the targeted user. It is worth noting that the Windows versions of GIMMICK Volexity has observed have no concept of setting their own persistence.

```
<?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>Label</key>
<string>com. /[applicationname].va.plist</string>
<key>ProgramArguments</key>
<array>
<string>/Users/#####/Library/Preferences/[pathto/binary]>/</string>
</array>
<key>RunAtLoad</key>
<true/>
<key>StartInterval</key>
<integer>30</integer>
<key>ThrottleInterval</key>
<integer>2</integer>
<key>WorkingDirectory</key>
<string>/Users/<removed>/Library/Preferences/[applicationname]string>
</dict>
</plist>
```

Likewise, the implant provides an uninstall function accessible by adding the argument “uninstall” on the command line. This removes the implant and all associated files, and then kills the process.

During initialization, the sample decodes several pieces of data critical to the malware operation using a rotating addition algorithm.

The first decoding loop results in a JSON object containing OAuth2 credentials for establishing a session to Google Drive. An example JSON object is shown in Figure 2:

```
{
  "token": "ya29.a0ARrdaM_3h5LqnwTmt929m5mw4ul9pMVGICT-
  [snipped]
  OWLfpG2lwTVAovNlk2FOMJeMoxZdDVH_7gE48JMFtiAK1zuSOVOPTddm
  n1V",
  "expire": "2021-09-14 09:57:24",
  "clientid":
  "1092974474287-h18u696gfqp7998iab7v9i0ra2lmd38f.apps.
  googleusercontent.com",
  "refresh_token": "1//0e9-6utdQixCiCgYIARAA[snipped]
  -iM6KTYFwIxZ9otUvnbwyre3zkZKnPLSEfJBo6kS4KXg",
  "token_uri": "https://oauth2.googleapis.com/token",
  "scopes": ["https://www.googleapis.com/auth/drive"],
  "client_secret": "faOO[snipped]g89CXm7"
}
```


Figure 2. An example JSON object containing credentials required to authenticate with Google Drive

The second loop decodes the 32-byte string “943c3743f72f06e58e60fa147481db83”. This string is run through an additional conversion stage that converts two characters at a time into a numeric representation and writes the resulting byte to a buffer. This buffer is used as an AES key in several calls to CCCrypt() function.

```
idxAesKeyConvert = -2LL;
pgAesKey = __g_szAesKey;
do
{
  szKeyNum[2] = 0;
  szKeyNum[0] = 0;
  szKeyNum[1] = 0;
  sprintf(
    szKeyNum,
    "%c%c",
    (unsigned int)bytesAESKey[idxAesKeyConvert + 2],
    (unsigned int)bytesAESKey[idxAesKeyConvert + 3]);
  *pgAesKey = strtol(szKeyNum, 0LL, 16);
  idxAesKeyConvert += 2LL;
  ++pgAesKey;
}
while ( idxAesKeyConvert < 30 );
```

Figure 3. AES key conversion

The final decode is done in place and its result is a 200-byte binary blob of configuration data, with only a few seemingly visible data boundaries.

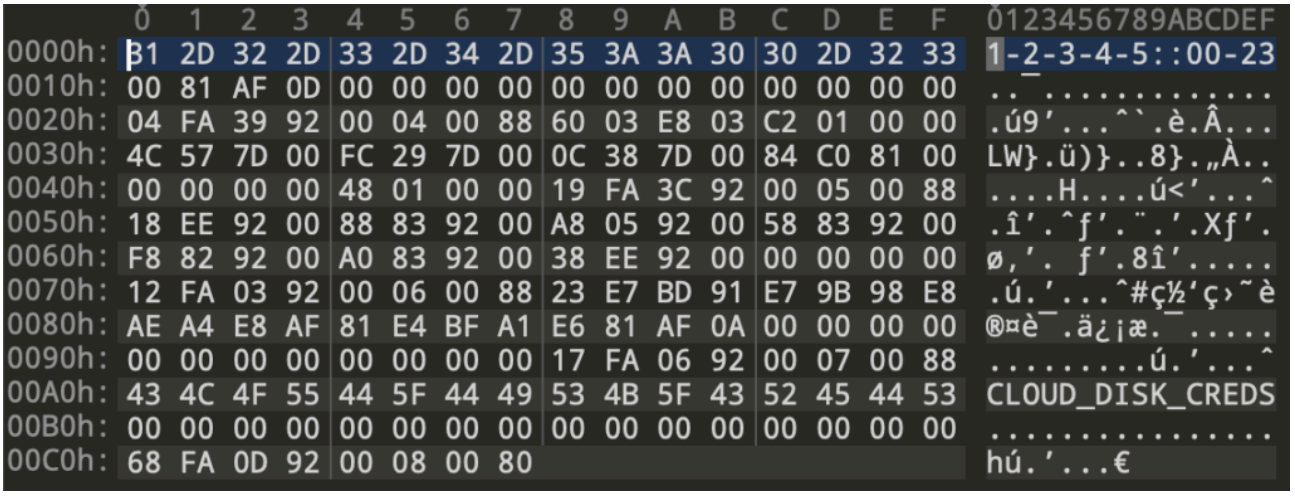

Figure 4: Config blob

Outside of this data obfuscation, and the use of AES for certain external files, the malware makes little attempt to obfuscate its functionality or presence on the system.

C2 Protocol

Post initialization, the operation of the GIMMICK malware is highly asynchronous. Prior variants of the malware written for Windows have managed this using thread pool techniques internal to the program, provided by Delphi's System.Threading.TThreadPool and .NET's System.Thread and System.Action. The macOS variant, however, manages the protocol using Apple's Grand Central Dispatch (GCD) technology. This feature allows developers to distribute tasks to a system-managed pool of threads for later processing. These tasks are encapsulated into self-contained objects called blocks which are scheduled on dispatch queues for processing. The precise structures and implementation details of GCD are fairly complicated and beyond the scope of this document; several resources are provided in the Appendix.

There are three custom ObjectiveC classes in the malware that manage critical aspects of the C2 protocol: DriveManager, FileManager, and GCDTimerManager.

DriveManager has several responsibilities:

- Manage the Google Drive and proxy sessions.
- Maintain a local map of the Google Drive directory hierarchy in memory.
- Manage locks for synchronizing tasks on the Google Drive session.
- Handle download and upload tasks to and from the Google Drive session.

Based on the way command files are enumerated by the malware, the Google Drive appears to be populated with a directory for each infected host. The name of this directory differs slightly by platform. Windows implants generate a unique GUID to operate as their ID, while the macOS implant uses Apple's own Hardware UUID for the task.

FileManager manages a local directory hierarchy containing C2 information and command tasks in various stages of completion. Older variants of GIMMICK used slightly different names for directories, but they have remained consistent across several recent variants. The macOS implant stores this hierarchy in the root directory of the application's main bundle in a directory named "MGD". Each folder within the directory structure is designated for holding a single type of file as it moves through the C2 process. A summary all directories and their purpose are given in the table below.

Name	Interpreted Meaning	Contents
tmp	Temporary	Temporary safe location for writing files; no dispatch code is checking files in this directory
c	Credentials	Stores the AES- encrypted credentials JSON decoded during initialization
e	Errors	Stores error logs as individual files; errors are reported as opaque integral values of usually four digits
p	Proxies	Stores proxy definition files consisting of a host and port separated by a ":"
u	Upload Command	Stored AES-encrypted command results pending upload
d	Download Command	Stores pending download command files, each containing the Google Drive path of a command file to be downloaded
ds	Download Success	Storage location for downloaded AES-encrypted command files awaiting processing
df	Download Failed	Temporary location for failed download command until they can be retried or cleared
l	List Command	Stores pending list of command files that indicate the directory of the Google Drive from which to download commands
ls	List Success	Stores temporary listing files containing paths of remote Google Drive files to be download
lf	List Failed	Temporary location for failed list commands until they can be retried or cleared

Not all variants of GIMMICK use all directories. For instance, the macOS implant does not use the "df" directory, and it creates, but does not access, the "lf" and "p" directories.

GCDTimerManager manages the various GCD objects that ensure the regular dispatching of work for the implant and holds collections of the dispatch timers along with their corresponding blocks. The malware creates several named dispatch queues for managing specific C2-related tasks:

Name	Purpose
SendBaseinfoQueue	Regularly generates and sends a system reconnaissance heartbeat message to the C2 containing the following: • Hardware UUID • MAC address of the eth0 interface • CPU model string • OS Version string
list_request_queue	Generates a list request file in the “l” directory containing a path in the format “/<HardwareUUID>”
ls_cmd_queue	Parses files from the “ls” directory and for each line, writes a corresponding download command file to the “d” directory
ReadCmdQueue	Decrypts and parses files from the “ds” directory, and executes the commands contained within, saving results to the “u” directory
CredsCheck	Checks for timeout of the Google Drive session, and re-authenticates if necessary
DriveClearTrashQueue	Regularly deletes the Google Drive trash file
DriveDownQueue	Parses files stored in the “d”, and downloads corresponding files from Google Drive to the “ds” directory
DriveUploadQueue	Uploads feedback files stored in the “u” directory
DriveFailUploadQueue	Second attempt to upload any failed upload items. Second attempt is marked successful regardless of result.
fileListQueue	Parses files stored in the “l” directory and for each, updates the DriveManager’s directory map of the Google Drive, and generates a listing of files to download in the “ls” directory

In addition, GCDTimerManager uses the static config information decoded during initialization to set a work period for the implant, limiting off-hour connections that might draw defender attention. It parses the work period from the string at the very start of the config data. This string starts with a set of single-digit numbers separated by hyphen characters, followed by two colon characters and two two-digit numbers separated again by a hyphen. The first set of numbers indicate the day number the malware will be active, with day 0 being Sunday. The second set of two-digit numbers indicate the range of active hours. Taking the initial value of “1-2-3-4-5::00-23”, the implant will be active from 12AM to 11PM on weekdays—this is the first data seen in the Configuration blob shown in Figure 4.

Command Lifetime

Due to the asynchronous nature of the malware operation, command execution requires a staged approach. Though the individual steps occur asynchronously, every command follows the same steps:

1. An encrypted payload is uploaded by the attacker to the Google Drive.
2. The dispatch timer on “list_request_queue” triggers.
    - New request file to be written to the “l” directory
3. The dispatch timer on te “fileListQueue” triggers.
    - Reads the list request from the “l” directory
    - Updates the DriveManager state from the Google Drive session
    - Drops a listing file to the “ls” directory
4. The dispatch timer on “ls_cmd_queue” triggers.
    - Parses the listing files from the “ls” directory
    - Drops download command files for each remote file in the “d” directory
    - Deletes listing files from the “ls” directory
5. The dispatch timer on “DriveDownloadQueue” triggers.
    - Enumerates the files in the “d” directory
    - Queues the download of command files to the “ds” directory
    - Queues deletion of remote Google Drive file and local download command file after download is complete
6. The dispatch timer on “ReadCmdQueue” triggers.
    - Reads and decrypts command files from “ds” directory
    - Handles command execution
    - Deletes local command file
    - Writes encrypted “feedback” files to “u” directory
7. The dispatch timer on “DriveUploadQueue” triggers.
    - Enumerates the files in the “u” directory
    - Queues the upload of the result files
    - Queues the deletion of local result files once upload is completed

Commands and Feedback

Commands reach the system as encrypted files in the “ds” directory which, once decrypted with the implant’s static AES key, result in a JSON object. There are only four JSON fields read by the command parser.

Name	**Type**
CMDType	Number
content	String
params	String
savepath	String

While each command JSON must have a CMDType field, the fields required vary from command to command. The table below summarizes the available commands and their required fields.

Enum	Description	Additional Required JSON Fields
0	Transmit base system information	None
1	Upload file to C2	params
2	Download file to client	content, savepath
3	Execute a shell command and write output to C2	params
4	Set client Google Drive timer interval	params
5	Set client timer interval for client info heartbeat message	params
6	Overwrite client work period information	params

Feedback to the C2 is also formatted as JSON, with fields fairly similar to the commands. However, all feedback JSON objects have one additional required field, “uuid”, which is populated with the device’s Hardware UUID.

Conclusion

Storm Cloud is an advanced and versatile threat actor, adapting its tool set to match different operating systems used by its targets. They make use of built-in operating system utilities, open-source tools, and custom malware implants to achieve their objectives. Leveraging cloud platforms for C2, such as using Google Drive, increases the likelihood of operating undetected by network monitoring solutions. This is especially true when coupled with the fact that the malware only beacons on victims’ working days.

Irrespective of platform, samples of the GIMMICK malware family are fairly large and complex, which is partly due to the complexity of their asynchronous design, such as the threading and locking mechanisms required. The work involved in porting this malware and adapting its systems to a new operating system (macOS) is no light undertaking and suggests the threat actor behind it is well resourced, adept, and versatile. It is worth noting that Volexity has only ever observed GIMMICK (macOS and Windows) in use by Storm Cloud. However, it is unknown if this malware implant is developed or otherwise used by them exclusively.

To generally prevent similar attacks from being successful, Volexity recommends the following:

- Regularly audit and monitor persistence locations, such as LaunchAgents and LaunchDaemons on endpoint macOS devices. This can be done through an EDR solution and/or with free tools such as BlockBlock and KnockKnock.
- Monitor network traffic for anomalous proxy activity and internal scanning.
- Ensure that XProtect and MRT from Apple are enabled and running on macOS systems.

To prevent these specific attacks from being successful, Volexity recommends the following:

- Use the rules provided to identify related activity, provided here.

Files related to this post are provided here.

This threat activity was detailed to Volexity Threat Intelligence customers in MAR-20220120.

Appendix

The following resources describe Apple's Grand Central Dispatch:

- https://www.amazon.com/dp/099105556X/ref=cm_sw_em_r_mt_dp_RYJ6VS3327WSY7SE551Y?_encoding=UTF8&psc=1 -> ISDN-13: 978-0991055562
- https://www.amazon.com/dp/0321706250/ref=cm_sw_em_r_mt_dp_7J0VBS0DW5NWAZAFT5ZF -> ISDN-13: 978-0321706256
- https://www.galloway.me.uk/2012/10/a-look-inside-blocks-episode-1/
- https://opensource.apple.com/source/libclosure/libclosure-67/BlockImplementation.txt.auto.html

---