GodRAT - New RAT targeting financial institutions

By Saurabh Sharma

Published: 2025-08-19 · Archived: 2026-04-05 22:05:23 UTC

Summary

In September 2024, we detected malicious activity targeting financial (trading and brokerage) firms through the distribution of malicious .scr (screen saver) files disguised as financial documents via Skype messenger. The threat actor deployed a newly identified Remote Access Trojan (RAT) named GodRAT, which is based on the Gh0st RAT codebase. To evade detection, the attackers used steganography to embed shellcode within image files. This shellcode downloads GodRAT from a Command-and-Control (C2) server.

GodRAT supports additional plugins. Once installed, attackers utilized the FileManager plugin to explore the victim's systems and deployed browser password stealers to extract credentials. In addition to GodRAT, they also used AsyncRAT as a secondary implant to maintain extended access.

GodRAT is very similar to the AwesomePuppet, another Gh0st RAT-based backdoor, which we reported in 2023, both in its code and distribution method. This suggests that it is probably an evolution of AwesomePuppet, which is in turn likely connected to the Winnti APT.

As of this blog's publication, the attack remains active, with the most recent detection observed on August 12, 2025. Below is a timeline of attacks based on detections of GodRAT shellcode injector executables. In addition to malicious .scr (screen saver) files, attackers also used .pif (Program Information File) files masquerading as financial documents.

GodRAT shellcode injector executable MD5	File name	Detection date	Country/territory	Distribution
cf7100bbb5ceb587f04a1f42939e24ab	2023-2024ClientList&.scr	2024.09.09	Hong Kong	via Skype
e723258b75fee6fbd8095f0a2ae7e53c	2024-11-15_23.45.45 .scr	2024.11.28	Hong Kong	via Skype
d09fd377d8566b9d7a5880649a0192b4	2024-08-01_2024-12-31Data.scr	2025.01.09	United Arab Emirates	via Skype
a6352b2c4a3e00de9e84295c8d505dad	2025TopDataTransaction&.scr	2025.02.28	United Arab Emirates	NA
6c12ec3795b082ec8d5e294e6a5d6d01	2024-2025Top&Data.scr	2025-03-17	United Arab Emirates	via Skype
bb23d0e061a8535f4cb8c6d724839883	• Corporate customer transaction &volume.pif • corporate customer transaction &volume.zip	2025-05-26	• United Arab Emirates • Lebanon • Malaysia	NA

	• company self-media account application qualifications&.zip			
160a80a754fd14679e5a7b5fc4aed672	• 个人信息资料&.pdf.pif • informasi pribadi &pelanggan global.pdf.pif • global customers preferential deposit steps&.pif	2025-07-17	Hong Kong	NA
2750d4d40902d123a80d24f0d0acc454	2025TopClineData&1.scr	2025-08-12	United Arab Emirates	NA
441b35ee7c366d4644dca741f51eb729	2025TopClineData&.scr	2025-08-12	Jordan	NA

Technical details

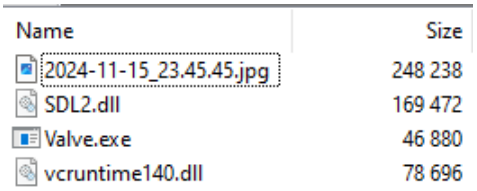
Malware implants

Shellcode loaders

We identified the use of two types of shellcode loaders, both of which execute the shellcode by injecting it into their own process. The first embeds the shellcode bytes directly into the loader binary, and the second reads the shellcode from an image file.

A GodRAT shellcode injector file named “2024-08-01_2024-12-31Data.scr” (MD5 d09fd377d8566b9d7a5880649a0192b4) is an executable that XOR-decodes embedded shellcode using the following hardcoded key: “OSEDBIU#IUSBDGKJS@SIHUDVNSO*SKJBKSDS#SFDBNXFCB”. A new section is then created in the memory of an executable process, where the decoded shellcode is copied. Then the new section is mapped into the process memory and a thread is spawned to execute the shellcode.

Another file, “2024-11-15_23.45.45 .scr” (MD5 e723258b75fee6fbd8095f0a2ae7e53c), serves as a self-extracting executable containing several embedded files as shown in the image below.



Content of self-extracting executable

Among these is “SDL2.dll” (MD5 512778f0de31fcce281d87f00affa4a8), which is a loader. The loader “SDL2.dll” is loaded by the legitimate executable Valve.exe (MD5 d6d6ddf71c2a46b4735c20ec16270ab6). Both the loader and Valve.exe are signed with an expired digital certificate. The certificate details are as follows:

- Serial Number: 084caf4df499141d404b7199aa2c2131
- Issuer Common Name: DigiCert SHA2 Assured ID Code Signing CA
- Validity: Not Before: Friday, September 25, 2015 at 5:30:00 AM; Not After: Wednesday, October 3, 2018 at 5:30:00 PM
- Subject: Valve

The loader “SDL2.dll” extracts shellcode bytes hidden within an image file “2024-11-15_23.45.45.jpg”. The image file represents some sort of financial details as shown below.

37158	3, 350, 000	1, 390, 000	−58. 51%	381, 717. 32		9050
HH801666	1, 970, 000	660, 000	−66. 50%	2, 789. 68		8. 61509E+
HU 3987	1, 010, 000	790, 000	−21. 78%	266, 100. 19		852 063
7750516	600, 000	1, 200, 000	100. 00%	38, 989. 05		33E+12
HU8	400, 000	1, 000, 000	150. 00%	35, 898. 38		8529
28003	1, 860, 000	620, 000	−66. 67%	258, 317. 29		3662
HH825228	2, 550, 000	120, 000	−95. 29%	2, 852. 96		8. 61393E+
HU 2610	590, 000	2, 770, 000	369. 49%	36, 992. 25		852 198
3196178	1, 373, 000	759, 000	−44. 72%	74, 951. 64		809190
HH8	3, 745, 000	60, 000	−98. 40%	32, 117. 82		8526
6783	720, 000	210, 000	−70. 83%	52, 571. 15		3633
HH811683	524, 000	62, 000	−88. 17%	13, 582. 69		852928111
HH 0202	1, 490, 000	260, 000	−82. 55%	5, 010. 19		852 569
3515693	430, 000	90, 000	−79. 07%	54. 87		779824
HU8	1, 740, 000	450, 000	−74. 14%	35, 372. 28		8529
7928	3, 810, 000	70, 000	−98. 16%	5, 029. 96		7579
HU838103	920, 000	550, 000	−40. 22%	9, 367. 08		614663990
HH 789	1, 360, 000	54, 000	−96. 03%	85, 569. 24		852 711
3587366	1, 260, 000	180, 000	−85. 71%	5, 681. 34		013801
HU8	1, 505, 000	315, 000	−79. 07%	17, 870. 73		8529
09299	2, 550, 000	660, 000	−74. 12%	42, 758. 38		8843
HH866780	1, 540, 000	40, 000	−97. 40%	7, 356. 45		852965429
HU 5100	760, 000	100, 000	−86. 84%	110, 798. 86		8. 1 +11
2638712	378, 000	47, 000	−87. 57%	26, 983. 20		711293
HH8	70, 000	828, 000	1082. 86%	42, 820. 35		8529
72690	2,644,000	756,000	-71.41%	849, 660. 51		2590

The loader allocates memory, copies the extracted shellcode bytes, and spawns a thread to execute it. We’ve also identified similar loaders that extracted shellcode from an image file named “2024-12-10_05.59.18.18.jpg”. One such loader (MD5 58f54b88f2009864db7e7a5d1610d27d) creates a registry load point entry at “HKCU\Software\Microsoft\Windows\CurrentVersion\Run\MyStartupApp” that points to the legitimate executable Valve.exe.

Shellcode functionality

The shellcode begins by searching for the string “godinfo,” which is immediately followed by configuration data that is decoded using the single-byte XOR key 0x63. The decoded configuration contains the following details: C2 IP address, port, and module command line string. The shellcode connects to the C2 server and transmits the string “GETGOD.” The C2 server responds with data representing the next (second) stage of the shellcode. This second-stage shellcode includes bootstrap code, a UPX-packed GodRAT DLL and configuration data. However, after downloading the second-stage

shellcode, the first stage shellcode overwrites the configuration data in the second stage with its own configuration data. A new thread is then created to execute the second-stage shellcode. The bootstrap code injects the GodRAT DLL into memory and subsequently invokes the DLL's entry point and its exported function "run." The entire next-stage shellcode is passed as an argument to the "run" function.

GodRAT

The GodRAT DLL has the internal name ONLINE.dll and exports only one method: "run". It checks the command line parameters and performs the following operations:

1. 1 If the number of command line arguments is one, it copies the command line from the configuration data, which was "C:\Windows\System32\curl.exe" in the analyzed sample. Then it appends the argument "-Puppet" to the command line and creates a new process with the command line "C:\Windows\System32\curl.exe -Puppet". The parameter "-Puppet" was used in AwesomePuppet RAT in a similar way. If this fails, GodRAT tries to create a process with the hardcoded command "%systemroot%\system2\cmd.exe -Puppet". If successful, it suspends the process, allocates memory, and writes the shellcode buffer (passed as a parameter to the exported function "run") to the allocated memory. A thread is then created to execute the shellcode, and the current process exits. This is done to execute GodRAT inside the curl.exe or cmd.exe process.
2. 2 If the number of command line arguments is greater than one, it checks if the second argument is "-Puppet." If true, it proceeds with the RAT's functionality; otherwise, it acts as if the number of command line arguments is one, as described in the previous case.

The RAT establishes a TCP connection to the C2 server on the port from the configuration blob. It collects the following victim information: OS information, local hostname, malware process name and process ID, user account name associated with malware process, installed antivirus software and whether a capture driver is present. A capture driver is probably needed for capturing pictures, but we haven't observed such behavior in the analyzed sample.

The collected data is zlib (deflate) compressed and then appended with a 15-byte header. Afterward, it is XOR-encoded three times per byte. The final data sent to the C2 server includes a 15-byte header followed by the compressed data blob. The header consists of the following fields: magic bytes (\x74\x78\x20) , total size (compressed data size + header size), decompressed data size, and a fixed DWORD (1 for incoming data and 2 for outgoing data). The data received from the C2 is only XOR-decoded, again three times per byte. This received data includes a 15-byte header followed by the command data. The RAT can perform the following operations based on the received command data:

- Inject a received plugin DLL into memory and call its exported method "PluginMe", passing the C2 hostname and port as arguments. It supports different plugins, but we only saw deployment of the FileManager plugin
- Close the socket and terminate the RAT process
- Download a file from a provided URL and launch it using the CreateProcessA API, using the default desktop (WinSta0\Default)
- Open a given URL using the shell command for opening Internet Explorer (e.g. "C:\Program Files\Internet Explorer\iexplore.exe" %1)
- Same as above but specify the default desktop (WinSta0\Default)
- Create the file "%AppData%\config.ini", create a section named "config" inside this file, and, create in that section a key called "NoteName" with the string provided from the C2 as its value

GodRAT FileManager plugin

The FileManager plugin DLL has the internal name FILE.dll and exports a single method called PluginMe. This plugin gathers the following victim information: details about logical drives (including drive letter, drive type, total bytes, available

free bytes, file system name, and volume name), the desktop path of the currently logged-on user, and whether the user is operating under the SYSTEM account. The plugin can perform the following operations based on the commands it receives:

- List files and folders at a specified location, collecting details like type (file or folder), name, size, and last write time
- Write data to an existing file at a specified offset
- Read data from a file at a specified offset
- Delete a file at a specified path
- Recursively delete files at a specified path
- Check for the existence of a specified file. If the file exists, send its size; otherwise, create a file for writing.
- Create a directory at a specified path
- Move an existing file or directory, including its children
- Open a specified application with its window visible using the ShellExecuteA API
- Open a specified application with its window hidden using the ShellExecuteA API
- Execute a specified command line with a hidden window using cmd.exe
- Search for files at a specified location, collecting absolute file paths, sizes, and last write times
- Stop a file search operation
- Execute 7zip by writing hard-coded 7zip executable bytes to “%AppData%\7z.exe” (MD5 eb8d53f9276d67afafb393a5b16e7c61) and “%AppData%\7z.dll” (MD5 e055aa2b77890647bdf5878b534fba2c), and then runs “%AppData%\7z.exe” with parameters provided by the C2. The utility is used to unzip dropped files.

Second-stage payload

The attackers deployed the following second-stage implants using GodRAT’s FileManager plugin:

Chrome password stealer

The stealer is placed at “%ALLUSERSPROFILE%\google\chrome.exe” (MD5 31385291c01bb25d635d098f91708905). It looks for Chrome database files with login data for accessed websites, including URLs and usernames used for authentication, as well as user passwords. The collected data is saved in the file “google.txt” within the module’s directory. The stealer searches for the following files:

- %LOCALAPPDATA%\Google\Chrome\User Data\Default\Login Data – an SQLite database with login and stats tables. This can be used to extract URLs and usernames used for authentication. Passwords are encrypted and not visible.
- %LOCALAPPDATA%\Google\Chrome\User Data\Local State – a file that contains the encryption key needed to decrypt stored passwords.

MS Edge password stealer

The stealer is placed at “%ALLUSERSPROFILE%\google\msedge.exe” (MD5 cdd5c08b43238c47087a5d914d61c943). The collected data is stored in the file “edge.txt” in the module’s directory. The module attempts to extract passwords using the following database and file:

- %LOCALAPPDATA%\Microsoft\Edge\User Data\Default\Login Data – the “Login Data” SQLite database stores Edge logins in the “logins” table.
- %LOCALAPPDATA%\Microsoft\Edge\User Data\Local State – this file contains the encryption key used to decrypt saved passwords.

AsyncRAT

The DLL file (MD5 605f25606bb925d61ccc47f0150db674) is an injector and is placed at “%LOCALAPPDATA%\bugreport\LoggerCollector.dll” or “%ALLUSERSPROFILE%\bugreport\LoggerCollector.dll”. It verifies that the module name matches “bugreport_.exe”. The loader then XOR-decodes embedded shellcode using the key “EG9RUOFIBVODSLFJBXLSVWKJENQWBIVUKDSZADVXBWEADSXZCXBVADZXVZXZXCBWES”. After decoding, it subtracts the second key “IUDSY86BVUIQNOEWSUFHGV87QCI3WEVBRSFUKIHVJQW7E8RBUYCBQO3WEIQWEXCSSA” from each shellcode byte.

A new memory section is created, the XOR-decoded shellcode is copied into it, and then the section is mapped into the current process memory. A thread is started to execute the code in this section. The shellcode is used to reflectively inject the C# AsyncRAT binary. Before injection, it patches the AMSI scanning functions (AmsiScanBuffer, AmsiScanString) and the EtwEventWrite function to bypass security checks.
AsyncRAT includes an embedded certificate with the following properties:

- Serial Number: df:2d:51:bf:e8:ec:0c:dc:d9:9a:3e:e8:57:1b:d9
- Issuer: CN = marke
- Validity: Not Before: Sep 4 18:59:09 2024 GMT; Not After: Dec 31 23:59:59 9999 GMT
- Subject: CN = marke

GodRAT client source and builder

We discovered the source code for the GodRAT client on a popular online malware scanner. It had been uploaded in July 2024. The file is named “GodRAT V3.5_____dll.rar” (MD5 04bf56c6491c5a455efea7dbf94145f1). This archive also includes the GodRAT builder (MD5 5f7087039cb42090003cc9dbb493215e), which allows users to generate either an executable file or a DLL. If an executable is chosen, users can pick a legitimate executable name from a list (svchost.exe, cmd.exe, cscript.exe, curl.exe, wscript.exe, QQMusic.exe and QQScLauncher.exe) to inject the code into. When saving the final payload, the user can choose the file type (.exe, .com, .bat, .scr and .pif). The source code is based on Gh0st RAT, as indicated by the fact that the auto-generated UID in “GodRAT.h” file matches that of “gh0st.h”, which suggests that GodRAT was originally just a renamed version of Gh0st RAT.

```
// GodRAT.h : main header file for the GODRAT application
//

#if !defined(AFX_GODRAT_H__C0496689_B41C_45DE_9F46_75A916C86D38__INCLUDED_)
#define AFX_GODRAT_H__C0496689_B41C_45DE_9F46_75A916C86D38__INCLUDED_
```

GodRAT.h

```
// gh0st.h : main header file for the GH0ST application
//

#if !defined(AFX_GH0ST_H__C0496689_B41C_45DE_9F46_75A916C86D38__INCLUDED_)
#define AFX_GH0ST_H__C0496689_B41C_45DE_9F46_75A916C86D38__INCLUDED_
```

gh0st.h

Conclusions

The rare command line parameter “puppet,” along with code similarities to Gh0st RAT and shared artifacts such as the fingerprint header, indicate that GodRAT shares a common origin with AwesomePuppet RAT, which we described in a private report in 2023. This RAT is also based on the Gh0st RAT source code and is likely connected with Winnty APT activities. Based on these findings, we are highly confident that GodRAT is an evolution of AwesomePuppet. There are some differences, however. For example, the C2 packet of GodRAT uses the “direction” field, which was not utilized in AwesomePuppet.

Old implant codebases, such as Gh0st RAT, which are nearly two decades old, continue to be used today. These are often customized and rebuilt to target a wide range of victims. These old implants are known to have been used by various threat actors for a long time, and the GodRAT discovery demonstrates that legacy codebases like Gh0st RAT can still maintain a long lifespan in the cybersecurity landscape.

Indicator of Compromise

File hashes

cf7100bbb5ceb587f04a1f42939e24ab
d09fd377d8566b9d7a5880649a0192b4 GodRAT Shellcode Injector
e723258b75fee6fbd8095f0a2ae7e53c GodRAT Self Extracting Executable
a6352b2c4a3e00de9e84295c8d505dad
6c12ec3795b082ec8d5e294e6a5d6d01
bb23d0e061a8535f4cb8c6d724839883
160a80a754fd14679e5a7b5fc4aed672
2750d4d40902d123a80d24f0d0acc454
441b35ee7c366d4644dca741f51eb729
318f5bf9894ac424fd4faf4ba857155e GodRAT Shellcode Injector
512778f0de31fcce281d87f00affa4a8 GodRAT Shellcode Injector
6cad01ca86e8cd5339ff1e8fff4c8558 GodRAT Shellcode Injector
58f54b88f2009864db7e7a5d1610d27d GodRAT Shellcode Injector
64dfcdd8f511f4c71d19f5a58139f2c0 GodRAT FileManager Plugin(n)
8008375eec7550d6d8e0eaf24389cf81 GodRAT
04bf56c6491c5a455efea7dbf94145f1 GodRAT source code
5f7087039cb42090003cc9dbb493215e GodRAT Builder
31385291c01bb25d635d098f91708905 Chrome Password Stealer
cdd5c08b43238c47087a5d914d61c943 MSEdge Password Stealer
605f25606bb925d61ccc47f0150db674 Async RAT Injector (n)
961188d6903866496c954f03ecff2a72 Async RAT Injector
4ecd2cf02bdf19cdbc5507e85a32c657 Async RAT
17e71cd415272a6469386f95366d3b64 Async RAT

File paths

C:\users\[username]\downloads\2023-2024clientlist & .scr
C:\users\[username]\downloads\2024-11-15_23.45.45 .scr
C:\Users\[username]\Downloads\2024-08-01_2024-12-31Data.scr
C:\Users\[username]\\Downloads\2025TopDataTransaction&.scr
C:\Users\[username]\Downloads\2024-2025Top&Data.scr
C:\Users\[username]\Downloads\2025TopClineData&1.scr
C:\Users\[username]\Downloads\Corporate customer transaction &volume.pif

C:\telegram desktop\Company self-media account application qualifications&.zip
C:\Users\[username]\Downloads\个人信息资料&.pdf.pif
%ALLUSERSPROFILE%\bugreport\360Safe2.exe
%ALLUSERSPROFILE%\google\chrome.exe
%ALLUSERSPROFILE%\google\msedge.exe
%LOCALAPPDATA%\valve\valve\SDL2.dll
%LOCALAPPDATA%\bugreport\LoggerCollector.dll
%ALLUSERSPROFILE%\bugreport\LoggerCollector.dll
%LOCALAPPDATA%\bugreport\bugreport_.exe

Domains and IPs

103[.]237[.]92[.]191 GodRAT C2
118[.]99[.]3[.]33 GodRAT C2
118[.]107[.]46[.]174 GodRAT C2
154[.]91[.]183[.]174 GodRAT C2
wuwu6[.]cfd AsyncRAT C2
156[.]241[.]134[.]49 AsyncRAT C2
https://holoohg.oss-cn-hongkong.aliyuncs[.]com/HG.txt URL containing AsyncRAT C2 address bytes
47[.]238[.]124[.]68 AsyncRAT C2
