National Cyber Security Centre
a part of GCHQ

Cheeky Chipmunk

Malware Analysis Report

Version 1.0

24th January 2022

National Cyber Security Centre
a part of GCHQ

Cheeky Chipmunk

Windows malware implemented as an RPC server

Executive summary

- Cheeky Chipmunk is a malicious Microsoft Windows Remote Procedure Call (RPC) server, installed via a PowerShell loader.
- Command and control (C2) is performed using a separate standalone executable RPC client.
- The loader implements an Antimalware Scanning Interface (AMSI) avoidance technique and is protected using layered obfuscation, encryption and compression.
- Functionality includes execution of shell commands and upload/download of files.
- The RPC connection is set up using named pipes, for which numerous names have been observed across multiple versions of Cheeky Chipmunk.
- Cheeky Chipmunk is added as a Windows service to maintain persistence.

Introduction

Cheeky Chipmunk is a malicious Microsoft Windows RPC server which is installed via a PowerShell loader and tasked by a separate RPC client. It can be tasked to execute shell commands and to upload/download files, enabling data exfiltration and providing a vector for execution of additional payloads. Cheeky Chipmunk employs multiple defence evasion techniques and maintains persistence as a Windows service.

National Cyber Security Centre
a part of GCHQ

Malware details

Metadata

Filename	mstcf.ps1
Description	Cheeky Chipmunk service loader
Size	181950 bytes
MD5	165be7620b78fe37cf25c797ee5b49e7
SHA-1	50c0bf9479efc93fa9cf1aa99bdca923273b71a1
SHA-256	22a8b4a7c7a467ea7fcf0a3930c99ecb482095093839683b400f58e2cdda176f

Filename	securlsa.chk
Description	Cheeky Chipmunk RPC server
Size	206848 bytes
MD5	38abeb8a68e9207da3e6ead88a9682ec
SHA-1	52c8cbd0545caab7596c1382c7fc5a479209851d
SHA-256	454e6c3d8c1c982cd301b4dd82ec3431935c28adea78ed8160d731ab0bed6cb7
Compile time	2019-03-26 11:59:38

Filename	mscsp.exe
Description	Cheeky Chipmunk RPC client
Size	55808 bytes
MD5	b03caf2f86a6b7adc06550bf666309a4
SHA-1	865f4c457bf86ff03456de828044f6ed6d6cf96a
SHA-256	d86f7d63bf8faa7070ba096523cf114947b55570dc50b0583797c9897ba3559c
Compile time	2013-12-09 10:37:34

Filename	
Description	Cheeky Chipmunk RPC client - DLL
Size	144896 bytes
MD5	50c98af90563bae4f89219d50feba38f
SHA-1	5d5825b14377c5e5fe96816dd72a90bd13dc9fc8
SHA-256	722fa0c893b39fef787b7bc277c979d29adc1525d77dd952f0cc61cd4d0597cc
Compile time	2019-11-15 12:19:44

Filename	mscsp.exe
Description	Cheeky Chipmunk RPC client
Size	50176 bytes
MD5	48afbdc27f3ff243ac2689e2ebd9f33c
SHA-1	7e4a6bac09ad214e98801bc199f96c265d7b6b48
SHA-256	27e7d2054a68510c974add24f33c1c7ef06ef68028cca021cd6b5e67363e2bea
Compile time	2008-07-19 08:33:09

National Cyber Security Centre
a part of GCHQ

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	Cheeky Chipmunk maintains persistence by adding a new Windows service (`pnrssp`) for the RPC server component.
Execution	T1059.001	Command and Scripting Interpreter: PowerShell	Open-source PowerShell runner code is embedded in the Cheeky Chipmunk RPC server and used to invoke commands stored in the registry.
	T1059.003	Command and Scripting Interpreter: Windows Command Shell	The Cheeky Chipmunk RPC server can be sent Windows shell commands by the RPC client.
	T1559.001	Inter-Process Communication: Component Object Model	The Cheeky Chipmunk RPC server uses COM to call embedded PowerShell runner code.
	T1047	Windows Management Instrumentation	Cheeky Chipmunk implements an RPC server to receive tasking.
Defense Evasion	T1140	Deobfuscate/Decode Files or Information	The Cheeky Chipmunk loader uses built-in and open-source PowerShell libraries to Base64-decode, 3DES-decrypt, and GZIP-decompress the RPC server.
	T1070.004	Indicator Removal on Host: File Deletion	The Cheeky Chipmunk loader is deleted from disk once the RPC server is running.
			Cheeky Chipmunk deletes the log file containing details of its RPC server's execution, following exfiltration.
	T1036.005	Masquerading: Match Legitimate Name or Location	Cheeky Chipmunk masquerades as a legitimate Windows service path (`netsvcs`) to maintain persistence.
			The Cheeky Chipmunk log file (`NTuser.log`) is named similarly to a legitimate Windows log file (`NTuser.dat.log`).
			The Cheeky Chipmunk service (`Peer Name Resolution Provider`) is named similarly to a legitimate Windows service (`Peer Name Resolution Protocol`).
	T1027	Obfuscated Files or Information	Cheeky Chipmunk's import names are obfuscated by XORing with `0x55`.
	T1562.001	Impair Defenses: Disable or Modify Tools	Cheeky Chipmunk implements an AMSI bypass technique.
Collection	T1005	Data from Local System	Cheeky Chipmunk can be tasked to collect files.

Functionality

Overview

Cheeky Chipmunk consists of a loader and an RPC server which receives C2 from a separate RPC client. In the analysed sample the loader is a PowerShell script; other loaders have also been observed, including a Windows DLL, but this will not be covered in detail in this report.

Cheeky Chipmunk gains persistence on the victim machine by adding itself as a Windows service, whereas the RPC client is a standalone executable binary which tasks the RPC server based on command line arguments. Other RPC clients have also been observed, including a Windows DLL, but this will not be covered in detail in this report.

The RPC server can be tasked to upload files and download files as well as to execute shell commands, as detailed in the 'Functionality (Tasking)' section of this report.

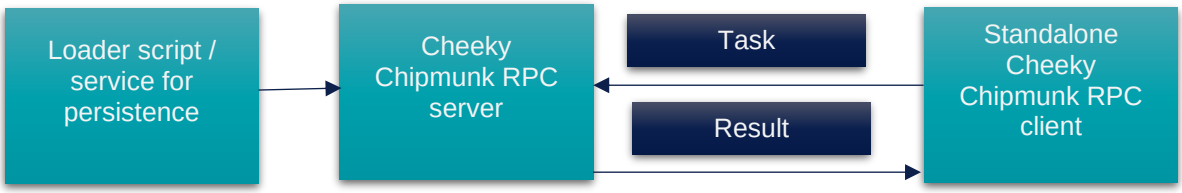

Figure 1: Cheeky Chipmunk components

Installation

The Cheeky Chipmunk loader consists of a PowerShell script (shown as 'script A' in Figure 2) containing a second embedded PowerShell script (shown as 'script B' in Figure 2), which in turn contains the Cheeky Chipmunk RPC server binary. It requires PowerShell to be executed with Administrator privileges to run successfully and uses obfuscated variable names to mask the behaviour of the script.

The loader implements an AMSI avoidance technique, as described in the 'Functionality (Defence evasion)' section of this report.

It creates the following registry key, adding itself to the `NullSessionPipes` list meaning it can access the named pipe anonymously, allowing unauthenticated access to null sessions even if restricted by security policies. The RPC server checks that this registry key is present when it starts up.

```
\\HKLM\SYSTEM\CurrentControlSet\services\LanmanServer\Parameters\NullSessionPipes\pnrsvc
```

Enabling the 'Network access: Restrict anonymous access to Named Pipes and Shares' security policy setting restricts null session access to unauthenticated users to all server pipes and shared folders except those listed in the NullSessionPipes and NullSessionShares registry entries.[1]

[1] https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/network-access-restrict-anonymous-access-to-named-pipes-and-shares

The embedded PowerShell script is Base64-encoded, 3DES-CBC-encrypted, and GZIP-compressed. The encryption key is derived using the built-in PowerShell function `Cryptography.PasswordDeriveBytes` with a hash name of `sha1`, using the key `SMQ0763eeg` and Initialisation Vector (IV) `NVAO8351dihi`. After Base64-decoding, this key is used with the IV `YDJFFTLPESPKMGUS` to decrypt the embedded script, which is then GZIP decompressed using the open-source PowerShell function `[MemoryZipper]::UnZip`.

Once decoded, the embedded PowerShell script contains a Base64-encoded Cheeky Chipmunk RPC server x64 binary. This is decoded and written to `C:\Windows\security\database\securlsa.chk`. This script also implements a persistence mechanism which is described in the 'Functionality (Persistence)' section of this report, and this is used to run the decoded RPC server. To clean up, the embedded script forcibly removes its own path, so the loader script will no longer be present on disk. A visual description of this is shown below in Figure 2.

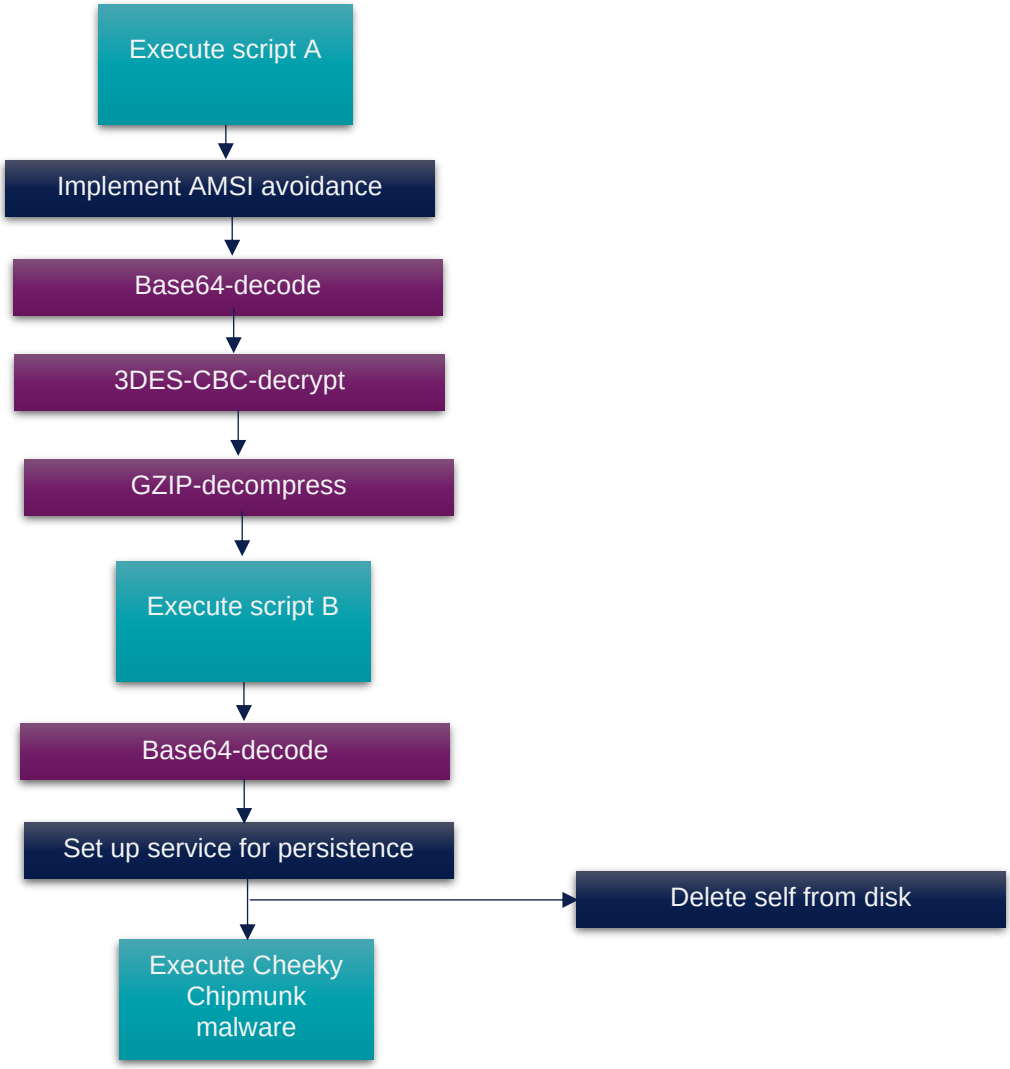

Figure 2: Cheeky Chipmunk loading process

RPC server

The core Cheeky Chipmunk functionality is implemented as a Windows RPC server, which is tasked by a corresponding stand-alone RPC client as described in the 'Functionality (Tasking)' section of this report. The RPC server is registered with the `RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH` flag set.

---

*When an RPC server is registered with the **`RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH`** flag, the RPC runtime invokes the registered security callback for all calls, regardless of identity, protocol sequence, or authentication level of the client[2].*

The RPC communications are described in the 'Communications' section of this report.

PowerShell execution

The malware contains an embedded .NET binary, containing `PowerShellRunner` code, which is open-source and allows PowerShell scripts to be executed without running `powershell.exe`. When the RPC server starts up, Cheeky Chipmunk checks that the process "explorer.exe" is running. Assuming this is found, COM is used to load the embedded `PowerShellRunner` binary and run a command which will Base64-decode and execute the values stored in the following registry locations:

- `\\HKLM\SOFTWARE\Microsoft\Network\Media\Enable`
- `\\HKLM\SOFTWARE\Microsoft\Network\Media\(Default)`

These registry keys are not populated by default by Cheeky Chipmunk.

When the PowerShell command completes, a message is logged to `%PUBLIC%\NTuser.log`

If the command was successful, the following message is logged. The '`%d`' format specifier is replaced with an internal return code, which is always `0`:

- `PE %d\n`

Otherwise, if an error occurred while executing the command, an internal error code describing the failure is logged.

Additionally, the malware contains two strings that are intended to be logged but which, due to a mistake, will never be written:

- "`Invoke payload (%d)...`" where %d will be the length of the payload
- "`status = %d\n`" where %d is an internal error

The PowerShell execution functionality runs concurrently with the RPC server in a loop which will check every 10-12 minutes to see if the `explorer.exe` process has been restarted. If it has, it will attempt to re-run the command from registry. However, due to a programming error, if the `explorer.exe` process is restarted, Cheeky Chipmunk will instead crash, resulting in the RPC service being stopped without alerting the user.

It is unclear why the payload is intended to be re-executed if explorer restarts, but it could suggest that the payload is intended to interact with it.

---

[2] https://docs.microsoft.com/en-us/windows/win32/rpc/interface-registration-flags

Tasking

The RPC client for Cheeky Chipmunk is a standalone executable which can task the malware as required.

RPC is an interprocess communication (IPC) mechanism that enables data exchange and invocation of functionality residing in a different process. That different process can be on the same machine, on the local area network, or across the Internet[3].

The client takes the following command line arguments:

- Network address of the machine running the RPC server
- Command flag (as shown in Table 1)
- Command parameter (as described in Table 1)

Command flag	ID	Command	Parameter
`/p`	0x01	Upload	Local file path to upload to RPC server.
`/g`	0x02	Download	Remote file path to download from RPC server.
`/c`	0x03	Execute	Command line to execute on RPC server.
N/A	0x04	Get execution output	Timeout value, hard-coded in client as 12 seconds – called immediately after an 'execute' command.

Table 1: Tasking commands

The tasking is sent from the RPC client to the RPC server as described in the 'Communications (Tasking communications)' section of this report.

When each tasking request completes, either a relevant error string or a success string is printed to the console by the RPC client. A list of these messages is included in the 'Appendix (Client console log messages)' section of this report. The exception to this is the 'get result' command which will print the retrieved result to the command line.

Upload

The 'upload' command sends the specified file path and contents to the RPC server.
The server checks that the file doesn't already exist, creates the file and writes the provided content to it.

If the file exists, an internal error code `0x50` is returned to the client and no further action is taken. If the upload fails for any other reason the result of `GetLastError` is returned by the RPC server.

---

[3] https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wpo/7d2df784-557e-4fde-9281-9509653a0f17

Download

The 'download' command sends the specified file path to the RPC server. The RPC server reads the file, checks that it is not empty and returns the content to the client.

If the file is empty, an internal error code `0x18` is returned to the client. If anything else fails, the RPC server returns the result of `GetLastError`.

Assuming the file is successfully downloaded, it is written to the same path on the RPC client system. If requested by the client, the RPC server will delete the file after it has been downloaded, however the RPC client does not support this option.

Execute

The 'execute' command sends the specified command line to the RPC server.

The RPC server prepends the command line with `cmd /c` and runs it as a child process. It creates a temporary file in the '%temp%' directory or, if that fails, in the current working directory, with the `FILE_FLAG_WRITE_THROUGH` flag set. The child process' `stdout` and `stderr` output is written to this file.

When a file is created with the `FILE_FLAG_WRITE_THROUGH` flag set, any file writes will go directly to disk without going through any intermediate cache[4]. This means no evidence of the file will remain in any cached locations after the file is deleted.

Get result

The 'get result' command is sent automatically by the RPC client after each 'execute' command. The RPC client calls it with a timeout value, which is hard-coded to be 12 seconds. The RPC server waits the requested time for the child process to complete and the temporary file to be written, then returns the content of the temporary file to the RPC client and deletes the file. If an error occurs, a message is returned to the RPC client, as shown in Table 2.

Error	Message	Clean up
No output from executed command	`Empty output \n`	Temporary file is deleted.
Command not completed within the timeout	`Timeout. Handle %s manually\n` (The '%s' format specifier is replaced by the temporary file name.)	None.
Temporary file not found	`Output not found\n`	Temporary file deleted on reboot.

Table 2: Execute command error messages

[4] https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea

If the temporary file cannot be found, Cheeky Chipmunk cannot delete the file immediately, so instead it schedules the file to be deleted on reboot using `MoveFileExW`. This ensures that if the execution exceeded the timeout value and the file is written after the clean-up is supposed to have occurred, the artefact will be removed on reboot.

The Windows API function MoveFileExW with destination NULL and the flag MOVEFILE_DELAY_UNTIL_REBOOT causes the file to be deleted when the machine is rebooted[5].

Persistence

Cheeky Chipmunk maintains persistence using a Windows service. Details of this service are shown in Table 3.

Name	Description
Name	Pnrssp
Binary Path Name	`%SystemRoot%\system32\svchost.exe -k netsvcs`
Display Name	Peer Name Resolution Provider
Description	Uses the NTLM MS-CHAP protocol to encapsulate and negotiate options in order to resolve domain names
Depends On	RpcSs

Table 3: Service details

[5] https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-movefileexw

Defence evasion

Cheeky Chipmunk employs several methods to evade detection.

Import resolution

Windows API functions are dynamically resolved by looking up the function name in the relevant module export tables. These function names, and the associated module names, are XOR-encoded with `0x55` to prevent static detection.

Blending with OS behaviour

The service is added to an existing service group, allowing it to blend with normal operating system behaviour. The log and service names are also named similarly to Windows files in an attempt to appear legitimate: `NTuser.dat.log` is a legitimate file, `NTuser.log` is a Cheeky Chipmunk file, the `Peer Name Resolution Protocol` is a legitimate service, the `Peer Name Resolution Provider` is the Cheeky Chipmunk service.

Removing files

The loader script forcibly removes itself, so the initial `mstcf.ps1` script will no longer be present on disk once it has loaded the RPC server and created the Windows service.

AMSI Avoidance

The loader script contains an embedded Base64-encoded .NET binary for Antimalware Scan Interface (AMSI) detection.

If AMSI is detected, then Cheeky Chipmunk uses an avoidance technique, where the .NET binary loads `amsi.dll` and finds `AmsiScanBuffer`, then overwrites it with either x86 or x64 shellcode to return `0x01`, causing all scripts to pass the AMSI check. This will stop the malware installation being detected and potentially blocked by AMSI.

The Windows Antimalware Scan Interface (AMSI) is a feature which can be integrated into any antimalware product which is present on a machine and is used to scan scripts for malicious content[6].

---

[6] https://docs.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal

Communications

Overview

Communications occur via Remote Procedure Call (RPC) using a named pipe. Multiple named pipes have been observed in use by different versions of Cheeky Chipmunk, including:

- `\pipe\atctl`
- `\pipe\pnrsvc`
- `\pipe\msbrws`

In addition to this, internal RPC structures contain a GUID which can be used to identify RPC client and RPC server interfaces. The GUID observed in use by the Cheeky Chipmunk malware is `{7DF02564-C31E-4A68-A688-72D0EC840746}`.

When any tasking is requested the internal network traffic will be as shown in Figure 3. Of note, this contains the GUID mentioned above. The SMB packet is shown in Figure 4.

```
SMB       213 Negotiate Protocol Request
SMB2      506 Negotiate Protocol Response
SMB2      232 Negotiate Protocol Request
SMB2      566 Negotiate Protocol Response
SMB2      220 Session Setup Request, NTLMSSP_NEGOTIATE
SMB2      401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
SMB2      318 Session Setup Request, NTLMSSP_AUTH, User: \
SMB2      159 Session Setup Response
SMB2      172 Tree Connect Request Tree: \\[redacted]\IPC$
SMB2      138 Tree Connect Response
SMB2      190 Create Request File: pnrsvc
SMB2      210 Create Response File: pnrsvc
SMB2      162 GetInfo Request FILE_INFO/SMB2_FILE_STANDARD_INFO File: pnrsvc
SMB2      154 GetInfo Response
DCERPC    286 Bind: call_id: 2, Fragment: Single, 2 context items: 7df02564-c31e-4a68-a688-72d0ec840746 V1.5 (32bit NDR), 7df02564-c31e-4a68-a688-72d0ec84074
SMB2      138 Write Response
SMB2      171 Read Request Len:1024 Off:0 File: pnrsvc
DCERPC    230 Bind_ack: call_id: 2, Fragment: Single, max_xmit: 4280 max_recv: 4280, 2 results: Acceptance, Negotiate ACK
DCERPC    254 Request: call_id: 2, Fragment: Single, opnum: 1, Ctx: 0 7df02564-c31e-4a68-a688-72d0ec840746 V1
DCERPC    234 Response: call_id: 2, Fragment: Single, Ctx: 0 7df02564-c31e-4a68-a688-72d0ec840746 V1
SMB2      146 Close Request File: pnrsvc
SMB2      182 Close Response
```

Figure 3: Cheeky Chipmunk traffic example

```
SMB2 (Server Message Block Protocol version 2)
  SMB2 Header
      Server Component: SMB2
      Header Length: 64
      Credit Charge: 1
      Channel Sequence: 0
      Reserved: 0000
      Command: Ioctl (11)
      Credits requested: 1
    > Flags: 0x00000030, Priority
      Chain Offset: 0x00000000
      Message ID: Unknown (9)
      Process Id: 0x0000feff
    > Tree Id: 0x00000001  \\[redacted]\IPC$
    > Session Id: 0x0000080000000045 Acct: Domain: Host:[redacted]
      Signature: 00000000000000000000000000000000
      [Response in: 32]
```

Figure 4: Cheeky Chipmunk SMB packet overview

The ioctl request contains the identifying service name as highlighted in Figure 5.

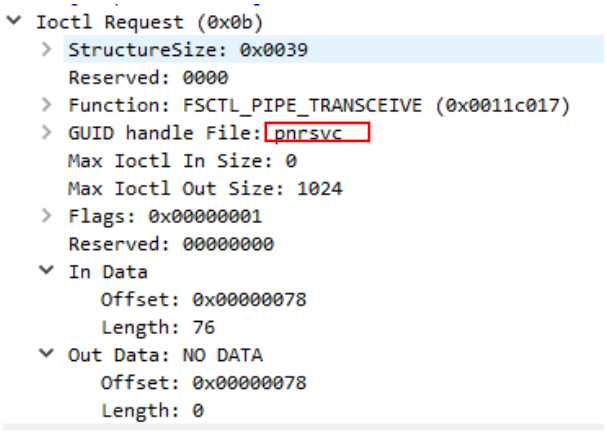

Figure 5: Cheeky Chipmunk ioctl request

The RPC packet is shown in Figure 6. Of note is the `Opnum` field which indicates the requested command. This value maps to the command ID shown in Table 1 in the 'Functionality (Tasking)' section of this report. Command parameters are encoded in the `Stub data` field, examples of which are included in the 'Communications (Tasking communications)' section of this report.

```
Distributed Computing Environment / Remote Procedure Call (DCE/RPC) Request, Fragment: Single,
   Version: 5
   Version (minor): 0
   Packet type: Request (0)
 > Packet Flags: 0x03
 > Data Representation: 10000000 (Order: Little-endian, Char: ASCII, Float: IEEE)
   Frag Length: 76
   Auth Length: 0
   Call ID: 2
   Alloc hint: 52
   Context ID: 0
   Opnum: 1
   [Response in frame: 32]
   Stub data: 120000000000000012000000043003a005c00740065007300...
```

Figure 6: Cheeky Chipmunk RPC packet

Tasking communications

This section details the arguments for each tasking request, whether these arguments are sent by the RPC client or returned by the RPC server, and what the RPC stub data looks like for each tasking command. Where [out] is specified next to an argument, this means the value is returned by the RPC server based on the result of the task processing.

Upload

An example of the data sent to the RPC server to request a file 'upload' is shown in Table 4.

The 'upload' function arguments are:

- Unicode file name to upload
- Size of file
- File contents

Upload stub data for request
11 00 00 00 00 00 00 00 11 00 00 00 43 00 3A 00 5C 00 74 00 65 00 73 00 74 00 5C 00 74 00 65 00 73 00 74 00 2E 00 74 00 78 00 74 00 00 00 00 00 15 00 00 00 00 00 02 00 15 00 00 00 63 6F 6E 74 65 6E 74 73 20 6F 66 20 74 65 73 74 20 66 69 6C 65
RPC interpreted data | File path | File contents

Table 4: Cheeky Chipmunk upload communication to RPC server

The contents of the file will be dependent upon the type of file requested. In the above example this is a text file, below is the text representation of the file path data and file content data shown in Table 4:

```
C:\test\test.txt
contents of test file
```

The possible return values which can be sent by the RPC server following this command are:

- `00 00 00 00` (success)
- `50 00 00 00` (file already exists)
- Result of `GetLastError` (any other failure)

Download

Examples of the traffic for a successful ‘download’ task to and from the RPC server are shown in Table 5 and Table 6.

The 'download’ function arguments are:

- Unicode file name to download
- [out] File size
- [out] File contents
- Delete flag

If the delete flag is set, the file will be deleted from the RPC server machine once the file has been sent from the RPC server to the RPC client.

Download stub data for request	
12 00 00 00 00 00 00 00 12 00 00 00 43 00 3A 00 5C 00 74 00 65 00 73 00 74 00 5C 00 74 00 65 00 73 00 74 00 32 00 2E 00 74 00 78 00 74 00 00 00 00 00 00 00	
RPC interpreted data	File path

Table 5: Cheeky Chipmunk download communication to RPC server

Download stub data for response	
17 00 00 00 00 00 02 00 17 00 00 00 63 6F 6E 74 65 6E 74 73 20 6F 66 20 74 65 73 74 20 66 69 6C 65 20 32 00 00 00 00 00	
RPC interpreted data	File contents

Table 6: Cheeky Chipmunk upload communication to RPC client

The contents of the file will be dependent upon the type of file requested. In the above example this is a simple text file, below is the text representation of the file path shown in Table 5 and the returned file contents shown in Table 6.

```
C:\test\test2.txt
contents of test file 2
```

Execute

An example of the traffic for an execution task request is shown in Table 7.

The 'execute' function only has one argument:

- The Unicode command to run

Execute stub data for request
20 00 00 00 00 00 00 00 20 00 00 00 43 00 3A 00 5C 00 57 00 69 00 6E 00 64 00 6F 00 77 00 73 00 5C 00 53 00 79 00 73 00 74 00 65 00 6D 00 33 00 32 00 5C 00 6D 00 73 00 70 00 61 00 69 00 6e 00 74 00 2E 00 65 00 78 00 65 00 00 00
RPC interpreted data | File Path

Table 7: Cheeky Chipmunk execute communication to RPC server

The text representation of the file path data shown in Table 7 is:

```
C:\Windows\System32\mspaint.exe
```

The return value for a successful run and an unsuccessful run are as follows:

- `00 00 00 00`
- Result of `GetLastError`

Get result

The 'get result' function arguments are:

- Timeout interval (in seconds)
- [out] Length result string
- [out] Result string

An example of the response traffic where a timeout occurred is shown in Table 8.

Get result stub data for response
34 00 00 00 00 00 02 00 34 00 00 00 54 69 6D 65 6F 75 74 2E 20 48 61 6E 64 6C 65 20 43 3A 5C 57 69 6E 64 6F 77 73 5C 54 45 4D 50 5C 31 41 36 39 2E 74 6D 70 20 6D 61 6E 75 61 6C 6C 79 5C 6E 00 00 00 00 00
RPC interpreted data | Result string

Table 8: Cheeky Chipmunk get result communication to client

The text representation of the returned result string data shown in Table 8 is:

```
Timeout. Handle C:\Windows\TEMP\1A69.tmp manually\n
```

The possible result strings for the get result command are all hard coded in the server binary as ASCII strings. All possible values for this are covered in the 'Functionality (Tasking)' section of this report.

Conclusion

Cheeky Chipmunk is assessed to be of medium sophistication and is unusual in its use of RPC for C2.

The loader is a self-deleting PowerShell script with the malware embedded inside, underneath multiple layers of encryption, encoding and compression. The loader implements an AMSI avoidance technique to ensure that the loading process can occur un-interrupted.

Cheeky Chipmunk implements many defence evasion techniques, including single-byte XOR-encoding of imported function names with `0x55`, which has been previously observed in regular use by Turla.

Networks infected with this malware risk files being exfiltrated. In addition to this, with the ability to upload, download and execute files on the victim machine, this has the potential to be a vector for downloading and running additional malware.

Although in theory the RPC client could be external to the network on which the RPC server is running, in many organisations this traffic would be blocked at the network boundary. As there are no other communication methods available in either the Cheeky Chipmunk server or client, it is likely to be used alongside other components.

Detection

Indicators of compromise

Type	Description	Values
Registry value name	Registry value required for the running of the service - this will be present on a victim machine	\\HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost\netsvcs\Pnrssp
Filename	Name of the Cheeky Chipmunk log file that will be present on a victim machine	%PUBLIC%\NTUser.log
Registry value name	Registry value required for the running of the service - this will be present on a victim machine	\\HKLM\SYSTEM\CurrentControlSet\services\pnrssp\ServiceMain\ServiceMain
Registry value name	Registry value required for the running of the service - this will be present on a victim machine	\\HKLM\SYSTEM\CurrentControlSet\services\pnrssp\ServiceDll\%SYSTEMROOT\security\database\securlsa.chk
Registry value name	NullSessionPipes registry value - this will be present on a victim machine	\\HKLM\SYSTEM\CurrentControlSet\services\LanmanServer\Parameters\NullSessionPipes\Pnrsvc
Filename	Name of the RPC server that will be present on a victim machine	%SYSTEMROOT%\security\database\securlsa.chk
Named pipes	Named pipes used for RPC communications	\\.\pipe\pnrsvc \\.\pipe\atctl \\.\pipe\msbrws
GUID	RPC GUID, seen in network traffic and hardcoded in binaries	7DF02564-C31E-4A68-A688-72D0EC840746

National Cyber Security Centre
a part of GCHQ

Rules and signatures

Description	Detects Cheeky Chipmunk argument checking code to determine command to process.
Precision	No false positives seen from VirusTotal retro-hunts
Rule type	YARA

```
rule CheekyChipmunk_command_argument_check
{
    meta:
        author = "NCSC"
        description = "Detects Cheeky Chipmunk argument checking code to
determine command to process. "
        date = "2022-01-24"
        hash1 = "7e4a6bac09ad214e98801bc199f96c265d7b6b48"

    strings:
            $1 = {66 83 F8 70 75 ?? C7 85 ?? FD FF FF 01 00 00 00 EB ??}
                  //cmp     ax, 70h ; 'p'
                  //jnz     short loc_401C39
                  //mov     [ebp+Switch], 1
                  //jmp     short loc_401C7F

            $2 = {66 83 F8 63 75 ?? C7 85 ?? FD FF FF 03 00 00 00 EB ??}
                  //cmp     ax, 63h ; 'c'
                  //jnz     short loc_401C4B
                  //mov     [ebp+Switch], 3
                  //jmp     short loc_401C7F

            $3 = {66 83 F8 67 75 ?? C7 85 ?? FD FF FF 02 00 00 00 EB ??}
                  //cmp     ax, 67h ; 'g'
                  //jnz     short loc_401C5D
                  //mov     [ebp+Switch], 2
                  //jmp     short loc_401C7F

            $4 = {66 83 F8 72 75 ?? C7 85 ?? FD FF FF 01 00 00 00 EB ??}
                  //cmp     ax, 72h ; 'r'
                  //jnz     short loc_401C6F
                  //mov     [ebp+var_300], 1
                  //jmp     short loc_401C7F02}

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

Description	Detects the Cheeky Chipmunk RPC GUID
Precision	No false positives seen from VirusTotal retro-hunts
Rule type	YARA

```
rule CheekyChipmunk_GUID_bytes
{
    meta:
        author = "NCSC"
        description = "Detects the Cheeky Chipmunk RPC GUID"
        date = "2022-01-24"
        hash1 = "52c8cbd0545caab7596c1382c7fc5a479209851d"
        hash2 = "865f4c457bf86ff03456de828044f6ed6d6cf96a"
        hash3 = "5d5825b14377c5e5fe96816dd72a90bd13dc9fc8"
        hash4 = "7e4a6bac09ad214e98801bc199f96c265d7b6b48"

    strings:
        $GUID = {64 25 F0 7D 1E C3 68 4A A6 88 72 D0 EC 84 07 46}

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

Description	Detects the Cheeky Chipmunk RPC service control handler
Precision	No false positives seen from VirusTotal retro-hunts
Rule type	YARA

```
rule CheekyChipmunk_control_handler_code
{
    meta:
        author = "NCSC"
        description = "Detects the Cheeky Chipmunk RPC service control
handler"
        date = "2022-01-24"
        hash1 = "52c8cbd0545caab7596c1382c7fc5a479209851d"

    strings:
        $servicecode = {41 B8 01 00 00 00 33 D2 B9 03 00 00 00 E8 ?? ??
?? ?? E8 ?? ?? ?? ?? 45 33 C0 33 D2 B9 01 00 00 00 E8 ?? ?? ?? ?? EB ??
41 B8 01 00 00 00 33 D2 B9 06 00 00 00 E8 ?? ?? ?? ?? 45 33 C0 33 D2 B9
07 00 00 00 E8 ?? ?? ?? ?? EB ?? 41 B8 01 00 00 00 33 D2 B9 05 00 00 00
E8 ?? ?? ?? ?? 45 33 C0 33 D2 B9 04 00 00 00 E8 ?? ?? ?? ??}

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

National Cyber Security Centre
a part of GCHQ

Description	Detects the stack string for the Cheeky Chipmunk log file name
Precision	No false positives seen from VirusTotal retro-hunts
Rule type	YARA

```
rule CheekyChipmunk_logfilename_stackstring
{
    meta:
        author = "NCSC"
        description = "Detects the stack string for the Cheeky Chipmunk
log file name"
        date = "2022-01-24"
        hash1 = "52c8cbd0545caab7596c1382c7fc5a479209851d"

    strings:
        $name = {B8 5C 00 00 00 66 89 84 24 ?? 00 00 00 B8 4E 00 00 00 66
89 84 24 ?? 00 00 00 B8 54 00 00 00 66 89 84 24 ?? 00 00 00 B8 55 00 00
00 66 89 84 24 ?? 00 00 00 B8 73 00 00 00 66 89 84 24 ?? 00 00 00 B8 65
00 00 00 66 89 84 24 ?? 00 00 00 B8 72 00 00 00 66 89 84 24 ?? 00 00 00
B8 2E 00 00 00 66 89 84 24 ?? 00 00 00 B8 6C 00 00 00 66 89 84 24 ?? 00
00 00 B8 6F 00 00 00 66 89 84 24 ?? 00 00 00 B8 67 00 00 00 66 89 84 24
?? 00 00 00 33 C0 66 89 84 24 ?? 00 00 00}

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

Description	Detects Cheeky Chipmunk loader AMSI avoidance strings
Precision	No false positives seen from VirusTotal retro-hunts
Rule type	YARA

```
rule CheekyChipmunk_amsi_avoidance_strings
{
    meta:
        author = "NCSC"
        description = "Detects Cheeky Chipmunk loader AMSI avoidance
strings"
        date = "2022-01-24"
        hash1 = "50c0bf9479efc93fa9cf1aa99bdca923273b71a1"
    strings:
        $functionname = "FindAmsiFun"
        $x86found = "x32 protection detected"
        $x64found = "x64 protection detected"

    condition:
        all of them
}
```

Description	Detects stack string for Cheeky Chipmunk named pipe
Precision	No false positives seen from VirusTotal retro-hunts
Rule type	YARA

```
rule CheekyChipmunk_namedpipe_stackstring
{
    meta:
        author = "NCSC"
        description = "Detects stack string for Cheeky Chipmunk named
pipe"
        date = "2022-01-24"
        hash1 = "52c8cbd0545caab7596c1382c7fc5a479209851d"
    strings:
        $stackstring = {B8 5C 00 00 00 66 89 84 24 80 00 00 00 B8 70 00
00 00 66 89 84 24 82 00 00 00 B8 69 00 00 00 66 89 84 24 84 00 00 00 B8
70 00 00 00 66 89 84 24 86 00 00 00 B8 65 00 00 00 66 89 84 24 88 00 00
00 B8 5C 00 00 00 66 89 84 24 8A 00 00 00 B8 70 00 00 00 66 89 84 24 8C
00 00 00 B8 6E 00 00 00 66 89 84 24 8E 00 00 00 B8 72 00 00 00 66 89 84
24 90 00 00 00 B8 73 00 00 00 66 89 84 24 92 00 00 00 B8 76 00 00 00 66
89 84 24 94 00 00 00 B8 63 00 00 00 66 89 84 24 96 00 00 00 33 C0 66 89
84 24 98 00 00 00}

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

Description	Detects Cheeky Chipmunk XOR function
Precision	No false positives seen from VirusTotal retro-hunts
Rule type	YARA

```
rule CheekyChipmunk_xor_function_code
{
    meta:
        author = "NCSC"
        description = "Detects Cheeky Chipmunk XOR function"
        date = "2022-01-24"
        hash1 = "52c8cbd0545caab7596c1382c7fc5a479209851d"

    strings:
        $funccode = {8B 44 24 04 48 8B 4C 24 20 0F B6 04 01 8B 0C 24 03
C8 8B C1 89 04 24 8B 44 24 04 48 8B 4C 24 20 0F B6 04 01 83 F0 55 8B 4C
24 04 48 8B 54 24 20 88 04 0A B8 01 00 00 00 48 6B C0 00 48 8B 4C 24 20
0F B6 04 01 8B 0C 24 03 C8 8B C1 89 04 24}

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

Description	Detects Cheeky Chipmunk client print format strings
Precision	No false positives seen from VirusTotal retro-hunts
Rule type	YARA

```
rule CheekyChipmunk_print_format_strings
{
    meta:
        author = "NCSC"
        description = "Detects Cheeky Chipmunk client print format
strings"


    strings:
        $SN = "0x%04d: SN %S: "
        $RC = "0x%04d: RC %S: "
        $EX = "0x%04d: EX %S: "
        $SNSuccess = "SN %S: success"
        $RCSuccess = "RC %S: success"

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

Description	Detects network traffic containing Cheeky Chipmunk Default Command Timeout
Precision	The rules have been tested and no false positives identified
Rule type	Snort

```
alert tcp any any -> any 445
(msg:"CheekyChipmunk_default_command_timeout"; \

      content:"|fe|SMB"; offset:4; depth:4; \

      # Command: IOCTL

      byte_test:2,=,11,8,relative,little; \

      content:"|e0 2e 00 00|"; fast_pattern; offset:148; depth:4; \

      sid: 1000001; rev:1; classtype: malware-cnc;)
```

Description	Detects network traffic containing Cheeky Chipmunk RPC GUID
Precision	The rules have been tested and no false positives identified
Rule type	Snort

```
alert tcp any any -> any 445 (msg:"CheekyChipmunk_RPC_GUID"; \

      content:"|fe|SMB"; offset:4; depth:4; \

      # Command: Write Request

      byte_test:2,=,9,8,relative,little; \

      content:"d%|f0|}|1e c3|hJ|a6 88|r|d0 ec 84 07|F"; offset:148;
depth:16;  \

      sid: 1000002; rev:1; classtype: malware-cnc;)
```

Description	Detects network traffic containing Cheeky Chipmunk named pipe
Precision	The rules have been tested and no false positives identified
Rule type	Snort

```
alert tcp any any -> any 445 (msg:"CheekyChipmunk_namedpipe_pnrsvc"; \
      content:"|fe|SMB"; offset:4; depth:4; \
      # Command: Create Request
      byte_test:2,=,5,8,relative,little; \
      content:"p|00|n|00|r|00|s|00|v|00|c|00|"; offset:124; depth:12; \
      sid:1000003; rev:1; classtype: malware-cnc;)
```

Description	Detects network traffic containing Cheeky Chipmunk named pipe
Precision	The rules have been tested and no false positives identified
Rule type	Snort

```
alert tcp any any -> any 445 (msg:"CheekyChipmunk_namedpipe_atctl"; \
      content:"|fe|SMB"; offset:4; depth:4; \
      # Command: Create Request
      byte_test:2,=,5,8,relative,little; \
      content:"a|00|t|00|c|00|t|00|l|00|"; offset:124; depth:10; \
      sid:1000004; rev:1; classtype: malware-cnc;)
```

National Cyber Security Centre
a part of GCHQ

Description	Detects network traffic containing Cheeky Chipmunk named pipe
Precision	The rules have been tested and no false positives identified
Rule type	Snort

```
alert tcp any any -> any 445 (msg:"CheekyChipmunk_namedpipe_msbrws"; \
      content:"|fe|SMB"; offset:4; depth:4; \
      # Command: Create Request
      byte_test:2,=,5,8,relative,little; \
      content:"m|00|s|00|b|00|r|00|w|00|s|00|"; offset:124; depth:12; \
      sid:1000005; rev:1; classtype: malware-cnc;)
```

Appendix

Client console log messages

Some of the error messages printed to console by the RPC client may vary slightly between versions, but the general format and abbreviations are consistent (`SN` = send, `RC` = receive, `EX` = execute, `GR` = get result).

Upload

- `0x%04d: SN %S: err size...aborting`
- `0x%04d: SN %S: failed %d...aborting`
- `0x%04d: SN %S: err %d...aborting`
- `SN %S: success`

Download

- `0x%04d: RC %S: zero length.`
- `0x%04d: RC %S: failed %d...aborting`
- `RC %S: success`

Execute

- `0x%04d: EX %S: failed %d...aborting`
- `EX: wait for ~%d seconds for output...`

Get result

- `0x%04d: GR %S: failed %d...aborting`
- `In success case this will print the results of the file execution`

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.