National Cyber Security Centre
a part of GCHQ

Wishful Woodchuck

Malware Analysis Report

Version 1.0

28 July 2021

National Cyber Security Centre
a part of GCHQ

Wishful Woodchuck

Non-persistent Windows keylogger

Executive summary

- Wishful Woodchuck logs keystrokes to a file on disk
- Windows API calls are used to hook the keyboard
- The log file is XOR-encoded with a multi-byte key which is embedded in the binary
- This keylogger contains no persistence or communications functionality

Introduction

Wishful Woodchuck is a non-persistent keylogger which contains some capability previously observed in use by Turla. This includes XOR 0x55 obfuscation of function names and use of the `TVer` versioning string.

Many variants of this keylogger have been observed with different methods of execution, including as a Windows executable file and as a DLL that is dropped and loaded by a separate executable. This includes variants compiled for x86 and x64 Windows architectures.

In all analysed variants the functionality is the same: Windows APIs are used to hook the keyboard, and keystrokes are then written to a file on disk in an encoded format.

National Cyber Security Centre
a part of GCHQ

Malware details

Metadata

Filename	msvrt.exe
Description	Dropper for keylogger DLL
Size	122880 bytes
Type	Windows Executable (PE) x86
MD5	4f97351fd325ecc4b1bcf10b67dfa885
SHA-1	8aac9cb8f12ffcf98a8bad5391e33ccf1021f60c
SHA-256	5af100c1781e80ecedc09b43430e58fe38319e944b65c0f622c71f42985957a1
Compile time	2013-06-26 14:22:03

Filename	lMM32.dll
Description	Keylogger DLL for x86, dropped by msvrt.exe, called by exports
Size	62464 bytes
Type	Windows Executable (PE) x86
MD5	887fba081dc7be123be7126cedae3e57
SHA-1	c4453b2dd4887ccdd2eccd19bad830d71c300943
SHA-256	740b27fc5552e5ac3c3655e9c598ed5711cfce442cc64e39af7dca8c468aad09
Compile time	2013-06-26 14:21:28

Filename	hlpapi.dll
Description	Keylogger DLL for x86 architecture
Size	61440 bytes
Type	Windows Executable (PE) x86
MD5	ca578729316f35db156849ba397df0a8
SHA-1	bc14776cd0b4f1cf7b42f2dd1ab931143ace9efb
SHA-256	3b7060063814ff7dbdda98b30d35282a5686e0b965e79ee89b1d9d279b5c125a
Compile time	2015-03-24 13:22:04

Filename	Manualmap_injector.exe
Description	Keylogger DLL for x64 architecture
Size	220160 bytes
Type	Windows Executable (PE) x64
MD5	f7570b7ec498bcd086a5318d9a9bcb0f
SHA-1	24abbcc1a326249d2e0f4ba159edbba43b15345b
SHA-256	dd40335044873fef29fca893a06eac4da0b1750651251c118b40e558c767c993
Compile time	2015-03-31 19:48:33

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
Collection	T1056.001	Input Capture: Keylogging	Wishful Woodchuck uses Windows API calls to log keystrokes
Defense Evasion	T1027	Obfuscated Files or Information	Wishful Woodchuck uses multi-byte XOR obfuscation of strings in its binary and log file
Defense Evasion	T1027	Obfuscated Files or Information	Wishful Woodchuck dynamically resolves Windows API functions using names XORed with 0x55
Defense Evasion	T1036.005	Masquerading: Match Legitimate Name or Location	Wishful Woodchuck uses a filename (`lmm32.dll`) similar to a legitimate Windows signed binary (`imm32.dll`)

Functionality

Overview

Three versions of the keylogger DLL and a dropper executable have been analysed:

- `lmm32.dll` is dropped by an executable dropper (`msvrt.exe`) and is a 32-bit Windows DLL that provides its functionality using exported functions.
- `hlpapi.dll` is a 32-bit Windows DLL which does not export any functions.
- `ManualMap_Injector.exe` is a 64-bit Windows DLL which does not export any functions.

Analysing these files shows the behaviour is very similar across them all and, unless otherwise specified, the following analysis is based on `hlpapi.dll`.

Mutex

In each analysed sample, Wishful Woodchuck creates a mutex and uses as its name the account username from which it is running, to ensure that only one instance runs at a time.

Dropper

As described in the previous section, one variant of Wishful Woodchuck consists of a dropper that contains a DLL with exported functions.

`msvrt.exe` is a Windows executable file which drops a DLL containing the main keylogging functionality to a statically configured filename. If executed with no arguments, it will drop the file and call its `SetHook` export, but it also accepts the following arguments:

- `-k` : the DLL export `SetHook` will be called.
  - This function implements the keylogger functionality as described in the 'Keylogger' section of this report.
- `-w` : the DLL export `SetWinCheck` will be called.
  - This function will write to the logfile with the header information and the details on the current process, with no keylogging occurring.

The dropped DLL is stored as a resource in the dropper with the name '`BINARY`' and ID `145`. Although the log file produced by the keylogger and the strings inside the binaries are encoded, the resource containing the keylogger binary is in plaintext.

Keylogger

Keystroke messages are hooked using the Windows API call `SetWindowsHookExW` and are logged to a file `msimm.dat` on disk. This log file is obfuscated using a static multi-byte rotating XOR key, as described in the 'Defence Evasion' section of this report, and the plaintext consists of UTF-16 strings. If the log file already exists, future runs of the keylogger will be appended to the same log file, starting from the `Start` string. An example deobfuscated log file is shown in Figure 1.

KSL0

TVer=21.0

Start

[u]:DESKTOP-RBMHRT6\user

[26.05.2021 14:29:30.471][h]:656738 [pid]:5480 [pn]:notepad++.exe [t]:new 1 - Notepad++

[26.05.2021 14:29:30.471][h]:656738 [pid]:5480 [pn]:notepad++.exe [t]:new 1 - Notepad++
1234...

[26.05.2021 15:37:33.601][h]:525034 [pid]:2460 [pn]:explorer.exe [t]:Windows
sys<Up><Up><Down><Down><Down><Down><Down><Down><Down><Down><Up><Up><Enter>imm

[26.05.2021 15:52:03.856][h]:65796 [pid]:2460 [pn]:explorer.exe [t]:Program Manager
<!LCtrl>c<#LCtrl>

Figure 1: Example keylogger log file

The log file starts with a header consisting of a fixed magic string (`KSL0`), a version number (`TVer=21.0`), the word `Start` to indicate the beginning of the key log entries and a line indicating the username of the current user. The format string for logging this information contains both `Start` and `[u]:` indicating the user will only be printed once at the start of each keylogging session.

As shown in Figure 1, key log entries consist of a single line containing several tab-delimited metadata fields followed by a sequence of captured keystrokes. Each line is prefixed with a millisecond-accurate timestamp and is terminated with LF – however note that captured whitespace, including new line characters, will be logged directly into the file.

Metadata fields are presented in the following format. Field identifiers are described in Table 1.

Data structure

```
[26.05.2021 14:29:30.471] \t [h]:656738 \t [pid]:5480 \t [pn]:notepad++.exe \t [t]:new 1 - Notepad++ \t 1234…
```

Timestamp	Window Handle	Process ID	Process Name	Title	Keystrokes

Figure 2: Log entry field structure

Field	Description
KSL0	Fixed string to indicate start of log file
TVer	Description
Start	Indicates beginning of keystroke message logging
[u]	The current user
[h]	Handle to the foreground window
[pid]	Process ID associated with currently focused window
[pn]	Process name associated with currently focused window
[t]	Title of currently focused window

Table 1: Log file metadata descriptions

There are certain non-text keystrokes which are represented in the log file with descriptive labels. A complete list of these labels is included in the appendix of this report.

Defence evasion

Wishful Woodchuck employs several methods to evade detection.

Import resolution

Windows API functions are dynamically resolved by looking up the function name in the relevant DLL export tables. These function names are obfuscated with single-byte XOR of 0x55 to prevent static detection. The strings for the module names from which these functions are imported are also under XOR 0x55 obfuscation.

String obfuscation

The strings required for the running of the binary, including the strings to be written to the log file, are stored in obfuscated form where hardcoded string values are multi-byte XORed with a hardcoded key.

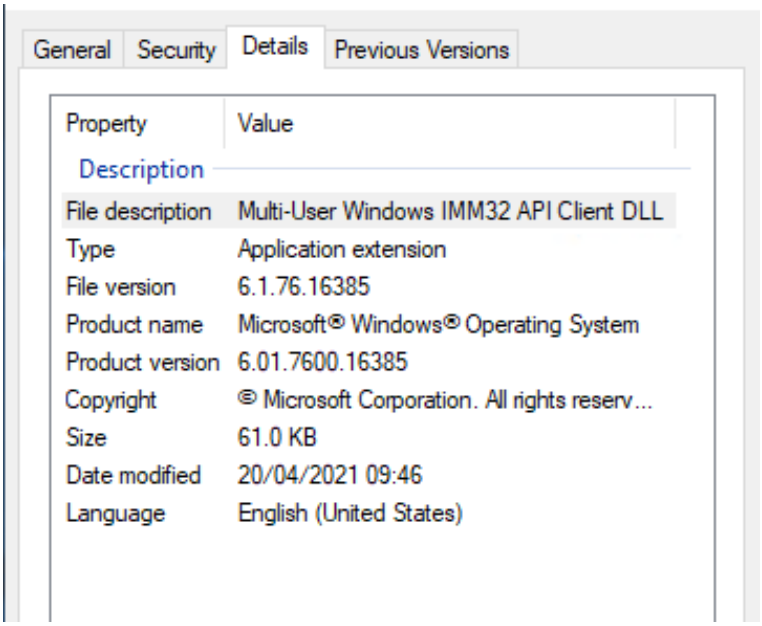
The binaries analysed all contain the same 100-byte-long key which is repeated to produce a keystream. Each string is obfuscated using a specific seed value which is used as a starting index into this stream, allowing each string to be obfuscated using a unique stream of key bytes. This stream is then XORed with the hardcoded encoded strings.

For the strings in the dropper binary, this seed value is hard coded. The same method is used for obfuscating the log file, where the seed value corresponds to the file size of the Wishful Woodchuck DLL that generated the log file.

The seed values for the strings in the keylogger binaries are also hard coded, however in this case the keystream initial index is calculated by first subtracting 2 from the hard coded values. Example scripts to implement both versions of this obfuscation are included in the appendix of this report.

DLL naming

In one analysed sample, the keylogger file is named `lmm32.dll`. As `imm32.dll` is the name of a legitimate signed Windows binary, this name has likely been selected to attempt to blend with normal operating system behaviour. Additionally, the description for this file is `Multi-User Windows IMM32 API CIient DLL` (note the capital `I` in `CIent`) which is similar to the description for the legitimate version, with one character mistyped. When viewed in the properties window, this is hard to spot (as shown below), and is only noticeable when the font is changed. This may stop immediate [human] recognition that an unrecognised file is running but it would not stop detection by automated methods.



Communications

Wishful Woodchuck contains no network communications functionality.

Conclusion

This keylogger is basic in its implementation, and although there are some defence evasion techniques included, they are simple. There is no mechanism for this to persist, so it will only execute while the host is running. Captured keystrokes are written to disk in an encoded format in the location from which the file was run. Retro-hunting found many variants of this keylogger, which reuse indicators such as the hardcoded multi-byte XOR key.

The `TVer` string in the log file is a known way that the Turla threat group reference versioning for their malware. Encoding Windows API function names with XOR 0x55 is also observed across numerous previously analysed Turla binaries. These observations would both suggest that this keylogger was written by Turla.

Detection

Indicators of compromise

Type	Description	Values
File	Log file dropped to disk by keylogger DLL to same location from which it runs.	`msimm.dat`
Timestamp	Multiple versions of the keylogger are compiled with the same timestamp.	`2015-03-31 19:48:33`

Rules and signatures

Description	Detects binaries containing the hardcoded XOR key from the keylogger binaries.
Precision	No false positives seen from Virus Total retro-hunts.
Rule type	YARA

```
rule wishfulwoodchuck_hardcoded_xorkey
{
    meta:
        author = "NCSC"
        description = "Finds binaries containing the hardcoded XOR key
from the keylogger binaries"


    strings:
        $xorkey = {0A 19 59 2D 6C 59 6F FA 8B 6F 9B FF 37 9B BD 7B 59 4B
7B DD 0F 64 91 C7 D6 9C 6F 7B 9C 01 9C 91 79 C7 C8 C9 DF E1 FA FF 04 08
59 E6 64 6D 37 9B 38 81 2D 81 65 7D 66 9A}

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

Description	Code to compare keystroke to certain values, mapping to non-character keys, such as up, down etc.
Precision	No false positives seen from Virus Total retro-hunts.
Rule type	YARA

```
rule wishfulwoodchuck_CheckKeystrokeValue
{
    meta:
        author = "NCSC"
        description = "Keystroke comparison code from binary"


      strings:
            // mov     eax, [ebp+1504h+var_1520]
            // mov     eax, [eax]
            // cmp     eax, 0A1h
            // jz      short loc_10002A89
            // cmp     eax, 0A0h
            // jz      short loc_10002A89
            // cmp     eax, 0A3h
            // jz      short loc_10002A89
            // cmp     eax, 0A2h

            $1 = {8B ?? ?? 8B 00 3D A1 00 00 00 74 ?? 3D A0 00 00 00 74
?? 3D A3 00 00 00 74 ?? 3D A2 00 00 00}

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

National Cyber Security Centre
a part of GCHQ

Description	Detects obfuscation occurring before the log file is written to disk.
Precision	No false positives seen from Virus Total retro-hunts.
Rule type	YARA

```
rule wishfulwoodchuck_obfuscation
{
    meta:
        author = "NCSC"
        description = "Obfuscation code from binary"


      strings:
            $1 = {B8 1F 85 EB 51 F7 E3 C1 EA 05 6B D2 64 8B C3 2B C2 0F
BE 90 ?? ?? ?? ?? 0F BE 04 39 33 D0 88 14 39 43 89 ?? ?? ?? ?? ?? 41 EB
??}
        // mov     eax, 51EB851Fh
        // mul     ebx
        // shr     edx, 5
        // imul    edx, 64h ; 'd'
        // mov     eax, ebx
        // sub     eax, edx
        // movsx   edx, keylist[eax]
        // movsx   eax, byte ptr [ecx+edi]
        // xor     edx, eax
        // mov     [ecx+edi], dl
        // inc     ebx
        // mov     dword_1000FCE4, ebx
        // inc     ecx
        // jmp     short loc_100013D0

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

Description	Unique file description
Precision	No false positives seen from Virus Total retro-hunts.
Rule type	YARA

```
rule wishfulwoodchuck_file_description
{
    meta:
        author = "NCSC"
        description = "Unique file description"


      strings:
            $description = "Multi-User Windows IMM32 API CIient DLL" wide
      condition:
            uint16(0) == 0x5A4D and
            uint32(uint32(0x3c)) == 0x00004550 and
            all of them
}
```

Description	Encoded log file header, will not change as long as the key doesn't change.
Precision	No false positives seen from Virus Total retro-hunts.
Rule type	YARA

```
rule wishfulwoodchuck_logfile_encodedheader
{
    meta:
        author = "NCSC"
        description = "Encoded log file header"


    strings:
        $header = {41 19 0A 2D 20 59 5F FA 81 6F CF FF 61 9B D8 7B 2B 4B
46}
    condition:
        all of them
}
```

Appendix

Log file decryption script

```
import os, sys
xor_key = [0x0A, 0x19, 0x59, 0x2D, 0x6C, 0x59, 0x6F, 0xFA, 0x8B, 0x6F,
0x9B, 0xFF, 0x37, 0x9B, 0xBD, 0x7B, 0x59, 0x4B, 0x7B, 0xDD, 0x0F, 0x64,
0x91, 0xC7, 0xD6, 0x9C, 0x6F, 0x7B, 0x9C, 0x01, 0x9C, 0x91, 0x79, 0xC7,
0xC8, 0xC9, 0xDF, 0xE1, 0xFA, 0xFF, 0x04, 0x08, 0x59, 0xE6, 0x64, 0x6D,
0x37, 0x9B, 0x38, 0x81, 0x2D, 0x81, 0x65, 0x7D, 0x66, 0x9A, 0x6F, 0xBD,
0x65, 0x59, 0x4B, 0x2D, 0x1A, 0x63, 0x59, 0x7B, 0x65, 0x59, 0x59, 0x0B,
0x4E, 0x85, 0x8C, 0x91, 0x88, 0x59, 0x0C, 0x01, 0x4E, 0x3A, 0x0D, 0x58,
0x38, 0x16, 0x91, 0x57, 0x7E, 0x68, 0x6A, 0x55, 0x42, 0x55, 0x5D, 0xC5,
0x9E, 0x4E, 0x17, 0x3B, 0x0F, 0x42]

def decode_function(buffer, seed):
    plaintext = b""
    for i in range(0, len(buffer)):
        key_index = (seed + i) % 100
        plaintext += bytes([buffer[i] ^ xor_key[key_index]])
    return plaintext

def decode_and_output_file(logfile, dllsize, outputfile):
    with open(logfile, "rb") as infile:
        plaintext = decode_function(infile.read(), int(dllsize))
    with open(outputfile, "ab") as outfile:
        outfile.write(plaintext)

def decode_and_output_bytes(byte_string, seed):
    plaintext = decode_function(bytearray.fromhex(byte_string),
int(seed))
    print(plaintext.decode('utf-16-le'))

# Usage: python3 decode.py <mode> <value> <seed> [<outfile>]
# Args:
#      mode: “1” to decode logfile, “2” to decode string
#      value: Logfile path for mode “1”, encoded string for mode “2”
#      seed: Length of DLL for mode “1”, seed for mode “2”
#      outfile: Path to write plaintext log file to, mode “1” only

if __name__ == "__main__":
    # Mode - 1 for FILE, 2 for BYTES
    mode = int(sys.argv[1])
    # file to decode OR byte string to decode
    valtodecode = sys.argv[2]
    # DLL length or string seed value
    seed = sys.argv[3]

    if mode == 1:
        # Output file (logfile decode only)
        outfile = sys.argv[4]
        decode_and_output_file(valtodecode, seed, outfile)

    elif mode == 2:
        decode_and_output_bytes(valtodecode, seed)
    else:
        sys.exit(1)
```

Keylogger binary strings decryption script

```
import sys

xor_key = [0x0A, 0x19, 0x59, 0x2D, 0x6C, 0x59, 0x6F, 0xFA, 0x8B, 0x6F,
0x9B, 0xFF, 0x37, 0x9B, 0xBD, 0x7B, 0x59, 0x4B, 0x7B, 0xDD, 0x0F, 0x64,
0x91, 0xC7, 0xD6, 0x9C, 0x6F, 0x7B, 0x9C, 0x01, 0x9C, 0x91, 0x79, 0xC7,
0xC8, 0xC9, 0xDF, 0xE1, 0xFA, 0xFF, 0x04, 0x08, 0x59, 0xE6, 0x64, 0x6D,
0x37, 0x9B, 0x38, 0x81, 0x2D, 0x81, 0x65, 0x7D, 0x66, 0x9A, 0x6F, 0xBD,
0x65, 0x59, 0x4B, 0x2D, 0x1A, 0x63, 0x59, 0x7B, 0x65, 0x59, 0x59, 0x0B,
0x4E, 0x85, 0x8C, 0x91, 0x88, 0x59, 0x0C, 0x01, 0x4E, 0x3A, 0x0D, 0x58,
0x38, 0x16, 0x91, 0x57, 0x7E, 0x68, 0x6A, 0x55, 0x42, 0x55, 0x5D, 0xC5,
0x9E, 0x4E, 0x17, 0x3B, 0x0F, 0x42]

def xor(inbyte, keybyte):
    print(chr(inbyte ^ keybyte))

def decode(encoded, seed):
    value = -2
    counter = 0
    while counter < len(encoded):
        xor(int(encoded[counter], 16),  xor_key[(seed+value)%100])
        counter+=1
        value +=1

# Usage: python3 decode.py <stringbytes> <seed>
# Args:
#      stringbytes: The bytes to decode in format “0x12 0x34 0x56 0x78”
#      seed: seed as defined per string in the keylogger binary

if __name__ == "__main__":
    inlist = (sys.argv[1]).split(' ')
    seed = int(sys.argv[2])
    decode(inlist, seed)
```

National Cyber Security Centre
a part of GCHQ

List of log file values for non-text keystrokes

<#RShift> <#LShift> <#RCtrl> <#LCtrl> <!RShift> <!LShift> <!RCtrl> <!LCtrl> <PageUp> <PageDown> <NumLock> <Down> <Up> <Right> <Left> <Del> <Print> <End> <Insert> <CapsLock> <Enter> <Backspace> <Esc> <Tab> - + [ ] \ ; / ` ' , .	<r/> <r*> <r-> <r+> <r1> <r2> <r3> <r4> <r5> <r6> <r7> <r8> <r9> <r0> <r.> <F1> <F2> <F3> <F4> <F5> <F6> <F7> <F8> <F9> <F10> <F11> <F12>

National Cyber Security Centre
a part of GCHQ

Additional files found through retro-hunts

Filename	lMM64.dll
Description	Keylogger DLL for x64 architecture
Size	62464 bytes
Type	Windows Executable (PE) x64
MD5	59b57bdabee2ce1fb566de51dd92ec94
SHA-1	9eb3c79dc361022a9d6ce3e2aa4962f240baf6f2
SHA-256	b7b5d28be983c774ef83a8960a68134732a79818c572e8800cea6428f27fb114
Compile time	2015-03-31 19:48:33

Filename	Manualmap_injector.vmp.exe
Description	Keylogger DLL for x64 architecture
Size	1299968 bytes
Type	Windows Executable (PE) x64
MD5	cd141c202737af15cac2612e0659aeb1
SHA-1	d5a7aaab836dcc539a3224bbb43a7f71f1aed37c
SHA-256	d76fe44316171dbed42265be0af798ce21ac917b42b3b5d01372e67781e579b3
Compile time	2015-03-31 19:48:33

Filename	N/A
Description	Keylogger DLL for x86 architecture
Size	94208 bytes
Type	Windows Executable (PE) x64
MD5	1d9fbd02954d4be1dfb0ff2fd27a6500
SHA-1	32ea80d2fc3c8db986d0eaa16fbfc8127b1a4936
SHA-256	05e045490c80c4464f3c6fb6c0c48bf040a0c482d6792e9e11471f1566e96ec6
Compile time	2015-03-31 19:48:33

Filename	WEXTRACT.EXE
Description	Keylogger executable for x86 architecture
Size	37892 bytes
Type	Windows Executable (PE) x86
MD5	792c4348c6a2d6f055b374beded31379
SHA-1	4c83a37489ff370523166b08159885fa245db83c
SHA-256	c4a1cd6916646aa502413d42e6e7441c6e7268926484f19d9acbf5113fc52fc8
Compile time	2009-02-02 13:06:31

Filename	WEXTRACT.EXE
Description	Keylogger executable for x86 architecture
Size	110592 bytes
Type	Windows Executable (PE) x86
MD5	a864cb7c29991475187968ad0ac5cd54
SHA-1	ec52ebda5c82084e797c275e76f5002b89b74fc4
SHA-256	e302a4cafd2d92fa99a79fd45cfc43b015f8ce444528fb78f32a88a874a52779
Compile time	2006-07-11 05:45:21

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.