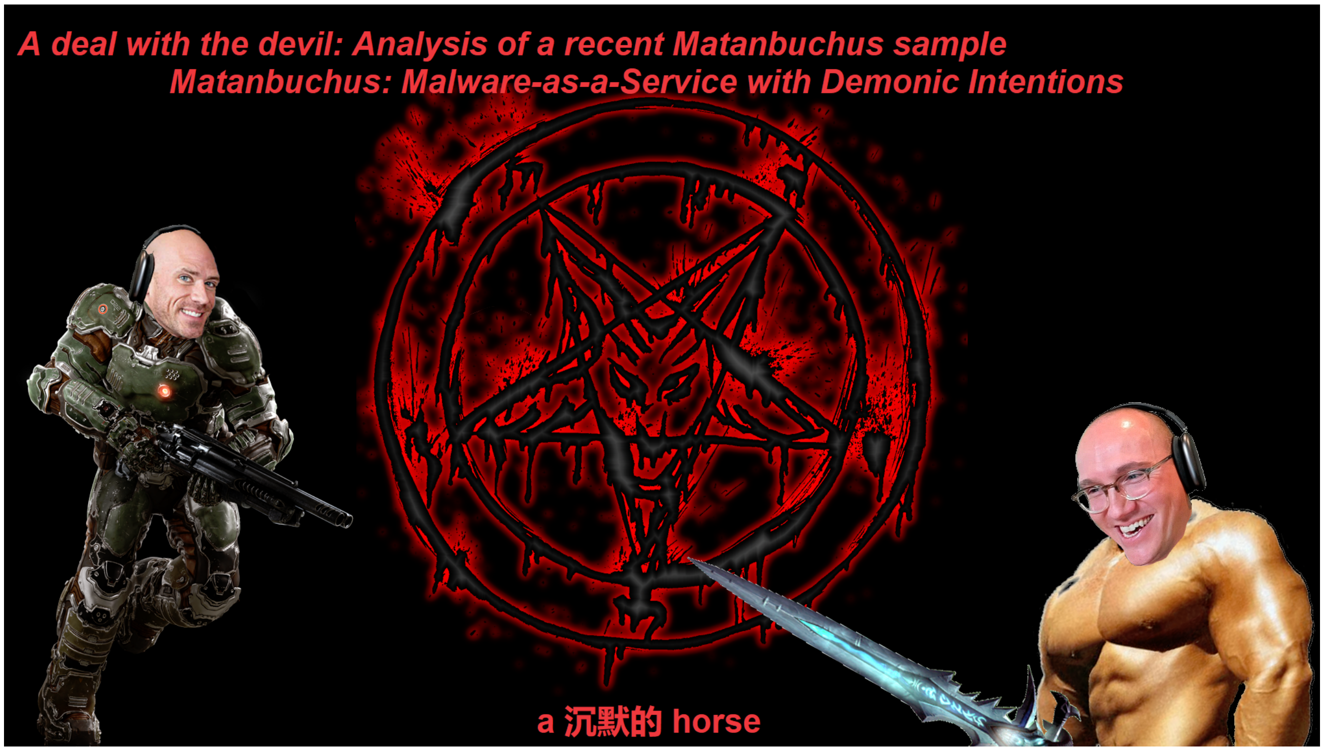
Matanbuchus Triage Notes

research.openanalysis.net/matanbuchus/loader/yara/triage/dumpulator/emulation/2022/06/19/matanbuchus-triage.html

OALABS Research

June 19, 2022

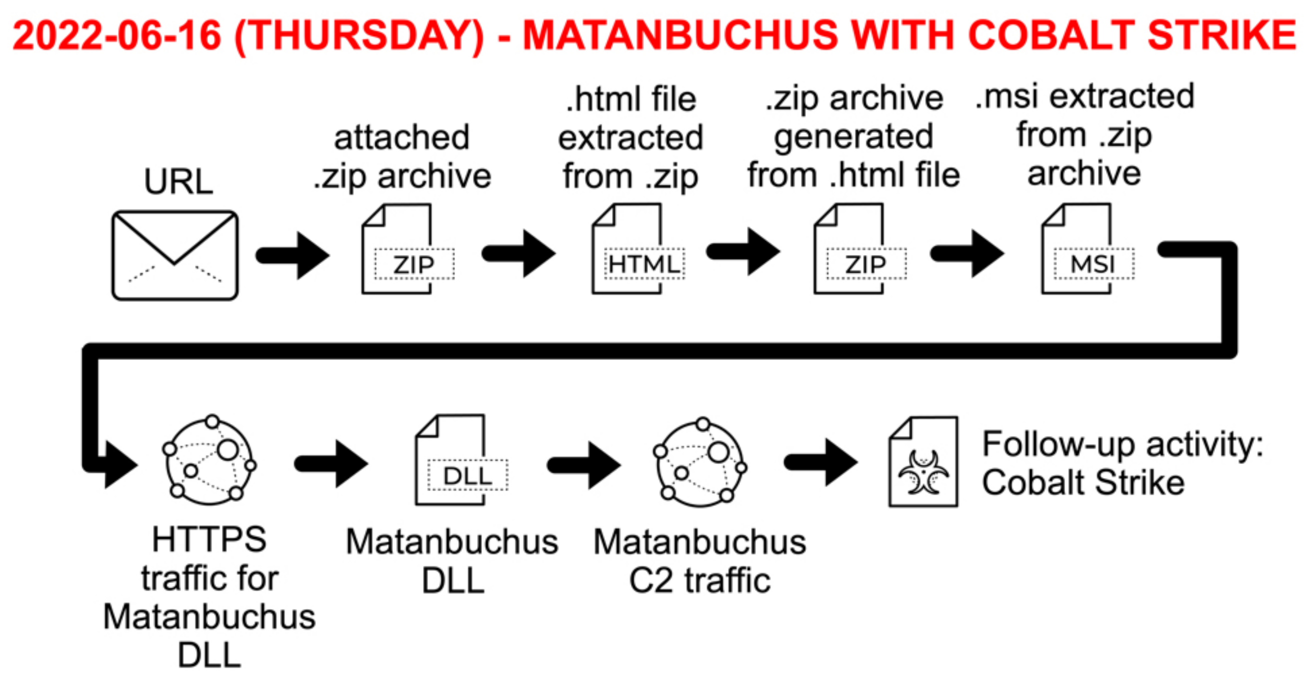

A deal with the devil: Analysis of a recent Matanbuchus sample
Matanbuchus: Malware-as-a-Service with Demonic Intentions
a 沉默的 horse

Overview

Matanbuchus is a malware downloader that has been observed as the final loading stage in multiple phishing campaigns. There are two components, a Matanbuchus loader intented to load the Matanbuchus downloader. Once the downloader is executed it makes an HTTP request to the C2 with some victim info, and downloads the final payload.

According to DCSO CyTec...

> Matanbuchus is the name given to a Malware-as-a-Service sold on Russian-speaking cybercriminal forums. Starting at a rental price of $2,500, the malware consists of an obfuscated two-stage loader which has been deployed in conjunction with Qakbot and Cobalt Strike payloads.


2022-06-16 (THURSDAY) - MATANBUCHUS WITH COBALT STRIKE
URL
attached .zip archive
ZIP
.html file extracted from .zip
HTML
.zip archive generated from .html file
ZIP
.msi extracted from .zip archive
MSI
HTTPS traffic for Matanbuchus DLL
DLL
Matanbuchus DLL
Matanbuchus C2 traffic
Follow-up activity: Cobalt Strike

References

Analysis

- The DLL has an export called `HackCheck` that uses OutputDebugStringA to print `start dll HackCheck`
- The sample uses API hashing with FNV1a hash algo to resolve API calls

Yara Rule

This yara rule is very brittle and needs lots of testing/refining

```
rule matanbuchus {
    meta:
        description = "Identifies matanbuchus"

   strings:
       // recursive fnv1 hash
       // 69 C2 93 01 00 01                      imul    eax, edx, 1000193h
       // 50                                     push    eax
       // B9 01 00 00 00                         mov     ecx, 1
       // C1 E1 00                               shl     ecx, 0
       $x1 = { 69 c2 93 01 00 01 50 b9 01 00 00 00 c1 e1 00 }

      //  string decryption
      //  0F BE 1C 01                            movsx   ebx, byte ptr [ecx+eax]
      //  33 DE                                  xor     ebx, esi
      //  6A 00                                  push    0
      //  6A 01                                  push    1
        $x2 = { 0f be 1c 01 33 de 6a 00 6a 01 }


   condition:
       all of ($x*)
}
```

Yara Rule Revised

```
rule matanbuchus {
    meta:
        description = "Identifies matanbuchus"

   strings:
       // fnv1 hash
       // 69 C2 93 01 00 01                       imul    eax, edx, 1000193h
       //
       // 69 C0 93 01 00 01                       imul    eax, 1000193h
       $x1 = { 69 ?? 93 01 00 01 }

      //pe loader
      $x2 = { B8 4D 5A 00 00 }
      $x3 = { 81 38 50 45 00 00 }

      //InternetCloseHandle
      $x5 = { 66 E9 DD 4D }

      //InternetOpenUrlA
      $x6 = { BC 8B CF F4 }

      //InternetCheckConnectionA
      $x7 = { 3F 82 58 52 }

      //InternetReadFile
      $x8 = { 66 E9 DD 4D }

   condition:
       all of ($x*)
}
```

String Decryption

Simple Decryption

Some strings are built as stack strings then copied into a buffer and returned from a function. The returned buffer is then decrypted directly using a simple XOR decryption routine where the first byte is the key.

```
import struct

data = b'jl8|tt8Py{s[p}{s'
key = struct.pack('<I',0x796C6B18)
data = key[1:] + data
out = []


for i in data:
    out.append(i ^ key[0])

bytes(out)
```

```
b'start dll HackCheck'

def unhex(hex_string):
    import binascii
    if type(hex_string) == str:
        return binascii.unhexlify(hex_string.encode('utf-8'))
    else:
        return binascii.unhexlify(hex_string)

def tohex(data):
    import binascii
    if type(data) == str:
        return binascii.hexlify(data.encode('utf-8'))
    else:
        return binascii.hexlify(data)
```

Complex Decryption

Other strings are also build from stack strings in a function and returned in a buffer, but these strings are decrypted with a second function call to a decryption function. To handle these more complex strings, we will use dumpulator

```
from dumpulator import Dumpulator
import struct
import re
import pefile

file_data = open('/tmp/mat.bin','rb').read()
pe = pefile.PE(data=file_data)

dp = Dumpulator("/tmp/mat.dmp", quiet=True)
fn_get_string = 0x708641C0
ss_start = 0x708631B2
ss_end = 0x708632AA

dp.start(ss_start, end=ss_end)

ss = dp.read(dp.regs.ebp-0x50, 0x48)
```

```
ss = bytes(ss)[:0x3e]

buff = dp.allocate(0x3e)
dp.write(buff, ss)
fn_dec_str = 0x7086F3D0

dp.call(fn_dec_str, [buff, 0x3e, 0x0, 0x7ebbfa3, 0x1010101])
dp.read(buff,0x3e)
```

```
Failed to read module data
```

```
bytearray(b'https://telemetrysystemcollection.com/m8YYdu/mCQ2U9/auth.aspx\x00')
```

Dumpulator Notes

The DLL uses thread safe functions that require a call into `EnterCriticalSection` and `LeaveCriticalSection`. Because our dump was taken before the DLL was initialized the critical section object was not initialized and this causes Dumpulator to crash.

First we tried just calling the initialization function in the DLL to setup this object (also tried calling the wrapper functions for `InitializeCriticalSectionEx`) but these all led to crashes (we could implement some syscalls and try again but we are lazy!

Our solution is to just patch out the `EnterCriticalSection` and `LeaveCriticalSection` calls in the loaded `ntdll` using a simple return 0.

```
33 C0     xor eax,eax
C2 04 00  ret 4
```

Dumpulator Patching Out Functions

```
dp = Dumpulator("/tmp/mat.dmp", quiet=True)

ntdll_start = 0x77a10000


patch_bytes = b'\x33\xC0\xC2\x04\x00'

ptr_RtlLeaveCriticalSection = 0x77A5FFF0
ptr_RtlEnterCriticalSection = 0x77A5FDF0

get_str_fn = 0x70861000

#tohex(dp.read(ntdll_init_crit_sec,10))

dp.write(ptr_RtlLeaveCriticalSection, patch_bytes)
dp.write(ptr_RtlEnterCriticalSection, patch_bytes)



dp.call(get_str_fn)
```

```
Failed to read module data
unmapped read from 8[4], cip = 77a6693a
error: Invalid memory read (UC_ERR_READ_UNMAPPED), cip = 77a6693a

134217727
```

Dumpulator Implementing Syscalls to Load DLL

Our patching method sort of worked but there is other initializations stuff that was causing more crashes. We are going to try and just implement the syscalls that we need to actually emulate the full DLL load routine and see if that works better.

```
from dumpulator import Dumpulator, syscall
from dumpulator.native import *

@syscall
def ZwQueryVolumeInformationFile(dp: Dumpulator,
                                 FileHandle: HANDLE,
                                 IoStatusBlock: P(IO_STATUS_BLOCK),
                                 FsInformation: PVOID,
                                 Length: ULONG,
                                 FsInformationClass: FSINFOCLASS
                                 ):
    return STATUS_SUCCESS



dp = Dumpulator("/tmp/mat2.dmp", quiet=True)

dll_base_addr = 0x71950000

dp.start(dp.regs.eip, end=dp.read_ptr(dp.regs.esp))


get_str_fn = 0x71951000
dp.read_str(dp.call(get_str_fn))

Failed to read module data

'https://telemetrysystemcollection.com/m8YYdu/mCQ2U9/auth.aspx'
```

Finding Encrypted Stack Strings

```
egg = rb'(\xC6\x45..){4}'

egg = rb'\x55\x8b\xec\x83\xec\x08\x33\xc0\x88\x45\xff'

stack_string_offsets = []
for m in re.finditer(egg, file_data):
    fn_offset = m.start()
    fn_addr = pe.get_rva_from_offset(fn_offset) + dll_base_addr
    tmp_str = dp.read_str(dp.call(fn_addr))
    print(tmp_str)
```

```
https://telemetrysystemcollection.com/m8YYdu/mCQ2U9/auth.aspx
https://collectiontelemetrysystem.com/m8YYdu/mCQ2U9/auth.aspx
DllRegisterServer
http://collectiontelemetrysystem.com/m8YYdu/mCQ2U9/home.aspx
http://telemetrysystemcollection.com/m8YYdu/mCQ2U9/home.aspx
https://telemetrysystemcollection.com/m8YYdu/mCQ2U9/home.aspx
```

Revising Our Config Extractor

Now that we have a base config extractor we need to test this across multiple samples/versions of the malware to make sure it is robust enough.

Our initial sample was `f8cc2cf36e193774f13c9c5f23ab777496dcd7ca588f4f73b45a7a5ffa96145e`.

We have collected the following samples to triage:

- `b9b399dbb5d901c16d97b7c30cc182736cd83a7c53313194a1798d61f9c7501e`
- `c41f7b7ec0909d5c21107ddebc2fe84dbc137f701b42943c1a5e69f5d50e05ab`
- `b4e7710488c2b7aaa71688b8bd546410af07a215c2e835e8dfbe24887186bd4f`
- `60f030597c75f9df0f7a494cb5432b600d41775cfe5cf13006c1448fa3a68d8d`
- `58a673023bbc7f2726e3b7ac917a43d9306692dc87b638ee21b98705a3262ccd`
- `4b87f95c4477fc66c58b8e16a74f9c47217913cb4a78dc69f27a364a099acd90`
- `4eb85a5532b98cbc4a6db1697cf46b9e2b7e28e89d6bbfc137b36c0736cd80e2`

More samples from Rattle (these work with our existing tools)

- `67a9e8599ab71865a97e75dae9be438c24d015a93e6a12fb5b450ec558528290`
- `075c904c5e18cd49f7d48f0fd1fc67d0921d51c7effb9233a3c92fbfa4f218ed`
- `60f030597c75f9df0f7a494cb5432b600d41775cfe5cf13006c1448fa3a68d8d`

These samples don't work with our first version of the config extractor:

- `e58b9bbb7bcdf3e901453b7b9c9e514fed1e53565e3280353dccc77cde26a98e.bin`
- `b9b399dbb5d901c16d97b7c30cc182736cd83a7c53313194a1798d61f9c7501e.bin`

New Config Extractor - Static String Decryption Notes

```
def unhex(hex_string):
    import binascii
    if type(hex_string) == str:
        return binascii.unhexlify(hex_string.encode('utf-8'))
    else:
        return binascii.unhexlify(hex_string)

def tohex(data):
    import binascii
    if type(data) == str:
        return binascii.hexlify(data.encode('utf-8'))
    else:
        return binascii.hexlify(data)
```

```
import re
import struct
import pefile

FILE_PATH =
'/tmp/samples/e58b9bbb7bcdf3e901453b7b9c9e514fed1e53565e3280353dccc77cde26a98e.bin'
FILE_PATH = '/tmp/samples/orig.bin'
FILE_PATH = '/tmp/samples/bd_rony.bin'
file_data = open(FILE_PATH,'rb').read()



def xor_decrypt(data, key):
    out = []
    for i in range(len(data)):
        out.append(data[i] ^ key[i%len(key)])
    return bytes(out)


def is_ascii(data):
    return re.match(B"^[\s!-~]+\0*$", data) is not None


def filter(data):
    return re.match(rb'[\f\v\t]', data) is not None


stack_strings = []

#string_egg = rb'(\xC6\x45..){2,}'

string_egg = rb'(?P<a>(?:\xC6\x85.{5}){0,})(?P<b>(?:\xC6\x45..){1,})'

for m in re.finditer(string_egg, file_data):
    match_data = m.group(0)
    #stack_strings.append({"data":match_data.replace(b'\xC6\x45',b'')[1::2],
"match":match_data})
    stack_strings.append({"data":m['a'][6::7] + m['b'][3::4], "match":match_data})




keys = []

key_byte_len_egg = rb'\x68(....)\x68(....)\x6a\x00\x6a(.)'

for m in re.finditer(key_byte_len_egg, file_data):
     keys.append( {'key':m.group(2) + m.group(1), 'length':ord(m.group(3))})


key_dw_len_egg = rb'\x68(....)\x68(....)\x6a\x00\x68(....)'
```

```
for m in re.finditer(key_dw_len_egg, file_data):
     keys.append( {'key':m.group(2) + m.group(1),
'length':struct.unpack('<I',m.group(3))[0]})


for sj in stack_strings:
    s = sj.get('data')
    str_len = len(s)
    if str_len < 5:
        continue
    #print(f"\n\n{str_len}")
    flag_found = False
    #print('\n')
    #print(f'{tohex(sj.get("match"))}')
    tmp_strings = []
    for k in keys:
        if k.get('length') == str_len:
            #print(f"found key: {tohex(k.get('key'))}")
            out = xor_decrypt(s, k.get('key'))
            if is_ascii(out) and not filter(out):
                tmp_strings.append(out)
                flag_found = True
    if not flag_found:
        tmp_xoe_dec = xor_decrypt(s[1:-1], bytes([s[0]]))
        if is_ascii(tmp_xoe_dec) and not filter(tmp_xoe_dec):
            print(tmp_xoe_dec)

    else:
        #print(f"\n{tmp_strings}")
        spec_char_egg = rb'[^A-Za-z0-9\s\./\\\%]{1}'
        best_match =  min( (len(re.findall(spec_char_egg, s)),s) for s in tmp_strings
)[1]
        print(best_match)
```

```
b'Content-Length: \x00'
b'C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe \x00'
b'collectiontelemetrysystem.com\x00'
b'DllRegisterServer\x00'
b'097f5m\x00'
b'Running dll in memory #3 (DllInstall(Unstall))\x00'
b'runas\x00'
b'.exe\x00'
b'timeout /t 3 && move /Y \x00'
b'Running dll in memory #3 (DllInstall(Install))\x00'
b'.exe\x00'
b'.exe\x00'
b'TiC7\x00'
b'.nls\x00'
b'Run PS in memory\x00'
b'Admin\x00'
b'%LOCALAPPDATA%\\\x00'
b'DllInstall\x00'
b'cmd.exe /c \x00'
b'collectiontelemetrysystem.com\x00'
b'Not in domain\x00'
b'regsvr32.exe \x00'
b'%PROCESSOR_REVISION%\\\x00'
b'%APPDATA%\\\x00'
b'41.4.0\x00'
b'8QN04\x00'
b'64 Bit\x00'
b'8S2x\x00'
b'Starting the exe with parameters\x00'
b'C:\\Windows\\System32\\cmd.exe /c \x00'
b'cmd.exe /c \x00'
b'High start exe\x00'
b'Running exe\x00'
b'User-Agent: \x00'
b'%PROCESSOR_REVISION%\\\x00'
b'Content-Type: application/x-www-form-urlencoded\x00'
b'%APPDATA%\\\x00'
b'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/\x00'
b'\\explorer.exe\x00'
b'MemLoadDllMain || MemLoadExe\x00'
b'Run CMD in memory\x00'
b'3CEk\x00'
b'3m7x\x00'
b'.nls\x00'
b'%APPDATA%\\\x00'
b'NCST \x00'
b'BCha\x00'
b"/c ECHO 'You must restart the program to resolve a critical error!' && start \x00"
b'7eriel\x00'
b'%APPDATA%\\\x00'
b'.nls\x00'
b'%PROCESSOR_REVISION%\\\x00'
```

```
b"\\'z{VIS\rA6\rb\x00"
b'%LOCALAPPDATA%\\\x00'
b'IsWow64Process\x00'
b'rundll32.exe \x00'
b'%PROCESSOR_REVISION%\\\x00'
b'kernel32\x00'
b'%PROCESSOR_REVISION%\\\x00'
b'.nls\x00'
b'User\x00'
b'%PROCESSOR_REVISION%\\\x00'
b'.nls\x00'
b'\r\n\r\n\x00'
b'\rXOGONSEzBER%\x00'
b'%PROCESSOR_REVISION%\\\x00'
b'\nost: \x00'
b'MemLoadShellCode\x00'
b'Q6X6\x00'
b'timeout /t 3 && del \x00'
b'cmd.exe\x00'
b'%02X-%02X-%02X-%02X-%02X-%02X\x00'
b'Running dll in memory #2 (DllRegisterServer)\x00'
b'Crypt update & Bots upgrade\x00'
b'timeout /t 3 && del \x00'
b'3fe11\x00'
b'%PROCESSOR_ARCHITECTURE%\x00'
b'%PROCESSOR_ARCHITECTURE%\x00'
b'NSeyDX\x00'
b'%APPDATA%\\\x00'
b'4nes\x00'
b'jpofxs\x00'
b'Not in domain\x00'
b'DllInstall\x00'
b' && rd /s /q \x00'
b' && regsvr32.exe -e "\x00'
b'32 Bit\x00'
b'%USERDOMAIN%\x00'
```

Final Config Decryptor

```
import re
import struct
import pefile


def xor_decrypt(data, key):
    out = []
    for i in range(len(data)):
        out.append(data[i] ^ key[i%len(key)])
    return bytes(out)


def is_ascii(data):
    return re.match(B"^[\s!-~]+\0*$", data) is not None


def get_strings(file_path):
    file_data = open(file_path,'rb').read()

    stack_strings = []

    #string_egg = rb'(\xC6\x45..){2,}'

    string_egg = rb'(?P<a>(?:\xC6\x85.{5}){0,})(?P<b>(?:\xC6\x45..){3,})'

    for m in re.finditer(string_egg, file_data):
        match_data = m.group(0)
        #stack_strings.append({"data":match_data.replace(b'\xC6\x45',b'')[1::2],
"match":match_data})
        stack_strings.append({"data":m['a'][6::7] + m['b'][3::4],
"match":match_data})

    keys = []

    key_byte_len_egg = rb'\x68(....)\x68(....)\x6a\x00\x6a(.)'

    for m in re.finditer(key_byte_len_egg, file_data):
         keys.append( {'key':m.group(2) + m.group(1), 'length':ord(m.group(3))})


    key_dw_len_egg = rb'\x68(....)\x68(....)\x6a\x00\x68(....)'

    for m in re.finditer(key_dw_len_egg, file_data):
         keys.append( {'key':m.group(2) + m.group(1),
'length':struct.unpack('<I',m.group(3))[0]})

    out_strings = []

    for sj in stack_strings:
        s = sj.get('data')
        str_len = len(s)
        flag_found = False
        tmp_out = []
        for k in keys:
            if k.get('length') == str_len:
                out = xor_decrypt(s, k.get('key'))
                if is_ascii(out):
                    tmp_out.append(out)

        if len(tmp_out) == 0:
            out = xor_decrypt(s[1:-1], bytes([s[0]]))
            if is_ascii(out):
                out_strings.append(out)
        elif len(tmp_out) == 1:
            out_strings.append(tmp_out[0])
        else:
            # This is a hack
            # We have strings that will give valid ascii for multiple keys
            # So far these have only been dll names
            for out in tmp_out:
                if out[:-3].lower() == b'dll':
                    out_strings.append(out)
                    break
    return list(set(out_strings))
```

Test All Samples

```
get_strings(FILE_PATH)

import os

for filename in os.listdir('/tmp/samples'):
    if filename.endswith(".bin"):
        file_path = os.path.join('/tmp/samples', filename)
        print(f"\n{file_path}")
        print(get_strings(file_path))
```

```
/tmp/samples/55d329a13da236bec15c4c67686b809a2fbf5f6c9625b62d900ac22a7b729ba9.bin
[b'VirtualAlloc', b'Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0\x00',
b'rundll32.exe\x00', b'9npSEGB3kg9suo3Yit\x00', b'kernel32.dll', b'https://azure-
dbupdate.at/vuUwUx/FyNRoM/index.php\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'LoadLibraryA',
b'IPHLPAPI.DLL\x00', b'https://azure-dbupdate.at/vuUwUx/FyNRoM/auth.php\x00',
b'https://azure-updatedb.at/vuUwUx/FyNRoM/index.php\x00', b'VirtualFree',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%']

/tmp/samples/4b87f95c4477fc66c58b8e16a74f9c47217913cb4a78dc69f27a364a099acd90.bin
[b'VirtualAlloc', b'Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0\x00',
b'rundll32.exe\x00', b'9npSEGB3kg9suo3Yit\x00', b'kernel32.dll', b'https://azure-
dbupdate.at/vuUwUx/FyNRoM/index.php\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'LoadLibraryA',
b'IPHLPAPI.DLL\x00', b'https://azure-dbupdate.at/vuUwUx/FyNRoM/auth.php\x00',
b'https://azure-updatedb.at/vuUwUx/FyNRoM/index.php\x00', b'VirtualFree',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%']

/tmp/samples/0bdf1060b85ad55e73393eb0b59c1d226e091da4f4dcce65dacba5e9a1fd76a7.bin
[b'VirtualAlloc', b'start dll HackCheck',
b'http://collectiontelemetrysystem.com/m8YYdu/mCQ2U9/home.aspx\x00',
b'https://telemetrysystemcollection.com/m8YYdu/mCQ2U9/auth.aspx\x00',
b'rundll32.exe\x00', b'kernel32.dll', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'Netapi32.dll\x00',
b'LoadLibraryA', b'http://telemetrysystemcollection.com/m8YYdu/mCQ2U9/home.aspx\x00',
b'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; Win64; x64; Trident/8.0;
.NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729;
Microsoft Outlook 16.0.5197; ms-office; MSOffice 16)\x00', b'IPHLPAPI.DLL\x00',
b'Wkscli.dll\x00',
b'https://telemetrysystemcollection.com/m8YYdu/mCQ2U9/home.aspx\x00', b'VirtualFree',
b'https://collectiontelemetrysystem.com/m8YYdu/mCQ2U9/auth.aspx\x00',
b'%PROCESSOR_LEVEL%']

/tmp/samples/3cae2ce9b2d7040292f1661af63dc28e778027c46f78d8be3b1d43f4b6c2b046.bin
[b'VirtualAlloc', b'Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0\x00',
b'rundll32.exe\x00', b'9npSEGB3kg9suo3Yit\x00', b'kernel32.dll', b'https://azure-
dbupdate.at/vuUwUx/FyNRoM/index.php\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'LoadLibraryA',
b'IPHLPAPI.DLL\x00', b'https://azure-dbupdate.at/vuUwUx/FyNRoM/auth.php\x00',
b'https://azure-updatedb.at/vuUwUx/FyNRoM/index.php\x00', b'VirtualFree',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%']

/tmp/samples/b4e7710488c2b7aaa71688b8bd546410af07a215c2e835e8dfbe24887186bd4f.bin
[b'VirtualAlloc', b'Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0\x00',
b'rundll32.exe\x00', b'9npSEGB3kg9suo3Yit\x00', b'kernel32.dll', b'https://azure-
dbupdate.at/vuUwUx/FyNRoM/index.php\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'LoadLibraryA',
b'IPHLPAPI.DLL\x00', b'https://azure-dbupdate.at/vuUwUx/FyNRoM/auth.php\x00',
b'https://azure-updatedb.at/vuUwUx/FyNRoM/index.php\x00', b'VirtualFree',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%']

/tmp/samples/orig.bin
[b'VirtualAlloc', b'start dll HackCheck',
b'http://collectiontelemetrysystem.com/m8YYdu/mCQ2U9/home.aspx\x00',
b'https://telemetrysystemcollection.com/m8YYdu/mCQ2U9/auth.aspx\x00',
b'rundll32.exe\x00', b'kernel32.dll', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'Netapi32.dll\x00',
b'LoadLibraryA', b'http://telemetrysystemcollection.com/m8YYdu/mCQ2U9/home.aspx\x00',
b'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; Win64; x64; Trident/8.0;
.NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729;
Microsoft Outlook 16.0.5197; ms-office; MSOffice 16)\x00', b'IPHLPAPI.DLL\x00',
b'Wkscli.dll\x00',
b'https://telemetrysystemcollection.com/m8YYdu/mCQ2U9/home.aspx\x00', b'VirtualFree',
b'https://collectiontelemetrysystem.com/m8YYdu/mCQ2U9/auth.aspx\x00',
b'%PROCESSOR_LEVEL%']

/tmp/samples/2f36c571f20b2b2c2058d4db574a6d53b148450356bf529d72aefc19505c912e.bin
[b'VirtualAlloc', b'Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0\x00',
b'rundll32.exe\x00', b'9npSEGB3kg9suo3Yit\x00', b'kernel32.dll', b'https://azure-
dbupdate.at/vuUwUx/FyNRoM/index.php\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'LoadLibraryA',
b'IPHLPAPI.DLL\x00', b'https://azure-dbupdate.at/vuUwUx/FyNRoM/auth.php\x00',
b'https://azure-updatedb.at/vuUwUx/FyNRoM/index.php\x00', b'VirtualFree',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%']

/tmp/samples/4eb85a5532b98cbc4a6db1697cf46b9e2b7e28e89d6bbfc137b36c0736cd80e2.bin
[b'rundll32.exe\x00', b'ztCYGAuJ\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'https://windowsdriverupdate.at/V.asp\x00', b'DllRegisterServer\x00', b'Microsoft
Office/16.0 (Windows NT 10.0; Microsoft Outlook 16.0.13127; Pro)\x00',
b'Shell32.dll\x00', b'Wininet.dll\x00', b'USER32.dll\x00',
b'https://windowsdriverupdate.at/ZBEr.asp\x00', b'Dll Uinstall\x00',
b'%PROCESSOR_LEVEL%\x00', b'https://driverwindowsupdate.at/V.asp\x00',
b'UnregisterServer\x00', b'IPHLPAPI.DLL\x00', b'%PROCESSOR_REVISION%\\\x00',
b'regsvr32.exe\x00', b'"C:\\Windows\\system32\\schtasks.exe" /Create /SC MINUTE /MO 1
/TN \x00']

/tmp/samples/10d5483faf9a4e0fbc17556164f47f7014650797b7d501289b269515a0853b64.bin
[b'VirtualAlloc', b'Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0\x00',
b'rundll32.exe\x00', b'9npSEGB3kg9suo3Yit\x00', b'kernel32.dll', b'https://azure-
dbupdate.at/vuUwUx/FyNRoM/index.php\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'LoadLibraryA',
b'IPHLPAPI.DLL\x00', b'https://azure-dbupdate.at/vuUwUx/FyNRoM/auth.php\x00',
b'https://azure-updatedb.at/vuUwUx/FyNRoM/index.php\x00', b'VirtualFree',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%']

/tmp/samples/58a673023bbc7f2726e3b7ac917a43d9306692dc87b638ee21b98705a3262ccd.bin
```

```
[b'VirtualAlloc', b'Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0\x00',
b'rundll32.exe\x00', b'9npSEGB3kg9suo3Yit\x00', b'kernel32.dll', b'https://azure-
dbupdate.at/vuUwUx/FyNRoM/index.php\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'LoadLibraryA',
b'IPHLPAPI.DLL\x00', b'https://azure-dbupdate.at/vuUwUx/FyNRoM/auth.php\x00',
b'https://azure-updatedb.at/vuUwUx/FyNRoM/index.php\x00', b'VirtualFree',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%']

/tmp/samples/b9b399dbb5d901c16d97b7c30cc182736cd83a7c53313194a1798d61f9c7501e.bin
[]

/tmp/samples/fa6500946210334d397d612d5ee9b11456316e25672bc60c1267bbdb002af9c7.bin
[b'VirtualAlloc', b'Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0\x00',
b'rundll32.exe\x00', b'9npSEGB3kg9suo3Yit\x00', b'kernel32.dll', b'https://azure-
dbupdate.at/vuUwUx/FyNRoM/index.php\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'LoadLibraryA',
b'IPHLPAPI.DLL\x00', b'https://azure-dbupdate.at/vuUwUx/FyNRoM/auth.php\x00',
b'https://azure-updatedb.at/vuUwUx/FyNRoM/index.php\x00', b'VirtualFree',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%']

/tmp/samples/60f030597c75f9df0f7a494cb5432b600d41775cfe5cf13006c1448fa3a68d8d.bin
[b'VirtualAlloc', b'start dll HackCheck',
b'http://collectiontelemetrysystem.com/m8YYdu/mCQ2U9/home.aspx\x00',
b'https://telemetrysystemcollection.com/m8YYdu/mCQ2U9/auth.aspx\x00',
b'rundll32.exe\x00', b'kernel32.dll', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'Netapi32.dll\x00',
b'LoadLibraryA', b'http://telemetrysystemcollection.com/m8YYdu/mCQ2U9/home.aspx\x00',
b'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; Win64; x64; Trident/8.0;
.NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729;
Microsoft Outlook 16.0.5197; ms-office; MSOffice 16)\x00', b'IPHLPAPI.DLL\x00',
b'Wkscli.dll\x00',
b'https://telemetrysystemcollection.com/m8YYdu/mCQ2U9/home.aspx\x00', b'VirtualFree',
b'https://collectiontelemetrysystem.com/m8YYdu/mCQ2U9/auth.aspx\x00',
b'%PROCESSOR_LEVEL%']

/tmp/samples/e58b9bbb7bcdf3e901453b7b9c9e514fed1e53565e3280353dccc77cde26a98e.bin
[b'C:\\Windows\\System32\\schtasks.exe\x00', b'rundll32.exe\x00', b' /TR
"%windir%\\system32\\regsvr32.exe -e \x00', b'%ProgramData%\\\x00',
b'Shlwapi.dll\x00', b'WS2_32.dll\x00', b'DllRegisterServer\x00',
b'%PROGRAMFILES%\\Opera\\Opera.exe\x00', b'Shell32.dll\x00', b'Wininet.dll\x00',
b'USER32.dll\x00', b'Dll Uinstall\x00', b'%COMPUTERNAME%\x00',
b'https://manageintel.com/RKyiihqXQiyE/xukYadevoVow/BhJM.xml\x00',
b'UnregisterServer\x00', b'IPHLPAPI.DLL\x00',
b'https://manageintel.com/RKyiihqXQiyE/xukYadevoVow/QXms.xml\x00', b'.ocx\x00',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%\\\x00', b'%PROCESSOR_REVISION%\x00']

/tmp/samples/a3c896e23c86e47bcb77096e743010546cd7699e0189344d11b9c642b89deef1.bin
[b'VirtualAlloc', b'Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0\x00',
b'rundll32.exe\x00', b'9npSEGB3kg9suo3Yit\x00', b'kernel32.dll', b'https://azure-
dbupdate.at/vuUwUx/FyNRoM/index.php\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'LoadLibraryA',
b'IPHLPAPI.DLL\x00', b'https://azure-dbupdate.at/vuUwUx/FyNRoM/auth.php\x00',
b'https://azure-updatedb.at/vuUwUx/FyNRoM/index.php\x00', b'VirtualFree',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%']

/tmp/samples/f27821dddb17b6c8d59fb2ada1e90eac8d561476e5af3a6be064177683b0eee9.bin
[b'VirtualAlloc', b'Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0\x00',
b'rundll32.exe\x00', b'9npSEGB3kg9suo3Yit\x00', b'kernel32.dll', b'https://azure-
dbupdate.at/vuUwUx/FyNRoM/index.php\x00', b'Shlwapi.dll\x00', b'WS2_32.dll\x00',
b'DllRegisterServer\x00', b'loaddll32.exe\x00', b'Shell32.dll\x00',
b'Wininet.dll\x00', b'USER32.dll\x00', b'GetProcAddress', b'LoadLibraryA',
b'IPHLPAPI.DLL\x00', b'https://azure-dbupdate.at/vuUwUx/FyNRoM/auth.php\x00',
b'https://azure-updatedb.at/vuUwUx/FyNRoM/index.php\x00', b'VirtualFree',
b'regsvr32.exe\x00', b'%PROCESSOR_LEVEL%']
```

Sample Using ADV String Obfuscation

A newer sample uses some type of ADV string obfuscation `b9b399dbb5d901c16d97b7c30cc182736cd83a7c53313194a1798d61f9c7501e`. We can probably use our old dumpulator tricks for this.

The start of the `.text` section contains a vtable with all of the string decryption functions.

```
from dumpulator import Dumpulator, syscall
from dumpulator.native import *

@syscall
def ZwQueryVolumeInformationFile(dp: Dumpulator,
                                 FileHandle: HANDLE,
                                 IoStatusBlock: P(IO_STATUS_BLOCK),
                                 FsInformation: PVOID,
                                 Length: ULONG,
                                 FsInformationClass: FSINFOCLASS
                                 ):
    return STATUS_SUCCESS



dp = Dumpulator("/tmp/b9b.dmp", quiet=True)

dp.start(dp.regs.eip, end=dp.read_ptr(dp.regs.esp))

functs =
[0x74040976,0x740409A4,0x740409BA,0x74040998,0x7404098C,0x7403F910,0x7403F91C,0x7403F9


for fn in functs:
    dp.call(fn,[])


str_tbl_start = 0x7407FDB4
str_tbl_end = 0x7407FE7C

str_tbl_start = 0x7407E000
str_tbl_end = 0x74080224

for ptr in range(str_tbl_start,str_tbl_end,4):
    try:
        ss = dp.read_str(dp.read_ptr(ptr))

        if len(ss) > 4:
            print(ss)
    except:
        continue
```

```
Failed to read module data
C:\Users\IEUser\Desktop\DLLLoader32_82D6.exe
"C:\Users\IEUser\Desktop\DLLLoader32_82D6.exe"
e03ed
Uninstall
3fe11
Running exe
Starting the exe with parameters
Run CMD in memory
Run PS in memory
Running dll in memory #3 (DllInstall(Unstall))
Running dll in memory #3 (DllInstall(Install))
Regsvr32 & Execute
MemLoadDllMain || MemLoadExe
zNETjp
5deb9c
Run EXE with admin rights
TAMfm
RunDll32 & Execute
Crypt update & Bots upgrade
Running dll in memory #2 (DllRegisterServer)
tbesqn
```

Different Sample From Same Family

This is clearly a different sample based on the decrypted strings, but it seems to be part of the matanbuchus family... maybe this is a payload instead of a loader? The sample matches analysis from this blog: Introduction of a PE file extractor for various situations>

We need to figure out why these samples are so different...

Taking a Closer Look At Obfuscated Samples

Obfuscated sample

> `b9b399dbb5d901c16d97b7c30cc182736cd83a7c53313194a1798d61f9c7501e` - does match yara (2021-11-12 11:47:44 UTC)

Rony

> `bd68ecd681b844232f050c21c1ea914590351ef64e889d8ef37ea63bd9e2a2ec` - doesn't match yara (2022-06-14 10:30:32 UTC)

These samples appear to be the same (they are likely the "payload" portion of Matanbuchus) but the earlier sample uses obfuscated string encryption, while the newer sample uses the simpler stack based string encryption.

Fixing our Yara Rule to Match the Stack Based String Encryption Payload (as well as loaders)

This was a simple fix! The payload does not contain the murmur hash code, once we removed that the yara rule matched all samples.

Let's Take a Look At the Obfusacated Strings

We know that the one sample we have that is a "payload" will likely have some of the same strings as the obfuscated sample so let's pull these our first as a reference.

```
b'Content-Length: \x00'
b'C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe \x00'
b'collectiontelemetrysystem.com\x00'
b'DllRegisterServer\x00'
b'097f5m\x00'
b'Running dll in memory #3 (DllInstall(Unstall))\x00'
b'runas\x00'
b'.exe\x00'
b'timeout /t 3 && move /Y \x00'
b'Running dll in memory #3 (DllInstall(Install))\x00'
b'.exe\x00'
b'.exe\x00'
b'TiC7\x00'
b'.nls\x00'
b'Run PS in memory\x00'
b'Admin\x00'
b'%LOCALAPPDATA%\\\x00'
b'DllInstall\x00'
b'cmd.exe /c \x00'
b'collectiontelemetrysystem.com\x00'
b'Not in domain\x00'
b'regsvr32.exe \x00'
b'%PROCESSOR_REVISION%\\\x00'
b'%APPDATA%\\\x00'
b'41.4.0\x00'
b'8QN04\x00'
b'64 Bit\x00'
b'8S2x\x00'
b'Starting the exe with parameters\x00'
b'C:\\Windows\\System32\\cmd.exe /c \x00'
b'cmd.exe /c \x00'
b'High start exe\x00'
b'Running exe\x00'
b'User-Agent: \x00'
b'%PROCESSOR_REVISION%\\\x00'
b'Content-Type: application/x-www-form-urlencoded\x00'
b'%APPDATA%\\\x00'
b'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/\x00'
b'\\explorer.exe\x00'
b'MemLoadDllMain || MemLoadExe\x00'
b'Run CMD in memory\x00'
b'3CEk\x00'
b'3m7x\x00'
b'.nls\x00'
b'%APPDATA%\\\x00'
b'NCST \x00'
b'BCha\x00'
b"/c ECHO 'You must restart the program to resolve a critical error!' && start \x00"
b'7eriel\x00'
b'%APPDATA%\\\x00'
b'.nls\x00'
b'%PROCESSOR_REVISION%\\\x00'
```

```
b"\\'z{VIS\rA6\rb\x00"
b'%LOCALAPPDATA%\\\x00'
b'IsWow64Process\x00'
b'rundll32.exe \x00'
b'%PROCESSOR_REVISION%\\\x00'
b'kernel32\x00'
b'%PROCESSOR_REVISION%\\\x00'
b'.nls\x00'
b'User\x00'
b'%PROCESSOR_REVISION%\\\x00'
b'.nls\x00'
b'\r\n\r\n\x00'
b'\rXOGONSEzBER%\x00'
b'%PROCESSOR_REVISION%\\\x00'
b'\nost: \x00'
b'MemLoadShellCode\x00'
b'Q6X6\x00'
b'timeout /t 3 && del \x00'
b'cmd.exe\x00'
b'%02X-%02X-%02X-%02X-%02X-%02X\x00'
b'Running dll in memory #2 (DllRegisterServer)\x00'
b'Crypt update & Bots upgrade\x00'
b'timeout /t 3 && del \x00'
b'3fe11\x00'
b'%PROCESSOR_ARCHITECTURE%\x00'
b'%PROCESSOR_ARCHITECTURE%\x00'
b'NSeyDX\x00'
b'%APPDATA%\\\x00'
b'4nes\x00'
b'jpofxs\x00'
b'Not in domain\x00'
b'DllInstall\x00'
b' && rd /s /q \x00'
b' && regsvr32.exe -e "\x00'
b'32 Bit\x00'
b'%USERDOMAIN%\x00'
```

```
from dumpulator import Dumpulator, syscall
from dumpulator.native import *
import pefile


@syscall
def ZwQueryVolumeInformationFile(dp: Dumpulator,
                                 FileHandle: HANDLE,
                                 IoStatusBlock: P(IO_STATUS_BLOCK),
                                 FsInformation: PVOID,
                                 Length: ULONG,
                                 FsInformationClass: FSINFOCLASS
                                 ):
    return STATUS_SUCCESS



dp = Dumpulator("/tmp/b9b.dmp", quiet=True)

dp.start(dp.regs.eip, end=dp.read_ptr(dp.regs.esp))

functs =
[0x74040976,0x740409A4,0x740409BA,0x74040998,0x7404098C,0x7403F910,0x7403F91C,0x7403F9


functs = [0x7403FC98]

for fn in functs:
    dp.call(fn,[])


data_start = 0x7407E000
data_end = 0x74080228

for ptr in range(data_start,data_end,4):
    try:
        ss = dp.read_str(dp.read_ptr(ptr))

        if len(ss) >= 4:
            print(ss)
    except:
        continue


print("\n\n\n ===\n")

print(dp.read_str(dp.read_ptr(0x7407FDFC )))
```

```
Failed to read module data
C:\Users\IEUser\Desktop\DLLLoader32_82D6.exe
"C:\Users\IEUser\Desktop\DLLLoader32_82D6.exe"
Q6X6
zkC7
rJqU
e03ed
3CEk
Uninstall
3fe11
Running exe
au5o
Starting the exe with parameters
3m7x
Run CMD in memory
Run PS in memory
Running dll in memory #3 (DllInstall(Unstall))
Running dll in memory #3 (DllInstall(Install))
Regsvr32 & Execute
MemLoadDllMain || MemLoadExe
b2tb
hszA
zNETjp
5deb9c
DS2x
Run EXE with admin rights
TAMfm
RunDll32 & Execute
wgjv
Crypt update & Bots upgrade
f1da
nX8y
Running dll in memory #2 (DllRegisterServer)
tbesqn
```

===

Running exe

```
FILE_PATH =
'/tmp/samples/b9b399dbb5d901c16d97b7c30cc182736cd83a7c53313194a1798d61f9c7501e.bin'
file_data = open(FILE_PATH,'rb').read()



def xor_decrypt(data, key):
    out = []
    for i in range(len(data)):
        out.append(data[i] ^ key[i%len(key)])
    return bytes(out)


def is_ascii(data):
    return re.match(B"^[\s!-~]+\0*$", data) is not None




# .text:74049A7D C6 45 B8 0D                            mov     byte ptr [ebp-48h],
13
# .text:74049A81 C7 45 E0 6E 60 69 23                   mov     dword ptr [ebp-20h],
2369606Eh
# .text:74049A88 C7 45 E4 68 75 68 2D                   mov     dword ptr [ebp-1Ch],
2D687568h
# .text:74049A8F C7 45 E8 22 6E 2D 00                   mov     dword ptr [ebp-18h],
2D6E22h
# .text:74049A96 C7 45 EC DA 4E 1E 00                   mov     dword ptr [ebp-14h],
1E4EDAh

test_data =
unhex('895db4c645b80dc745e06e606923c745e46875682dc745e8226e2d00c745ecda4e1e0052')

stack_strings = []

string_egg = rb'(?P<a>(?:\xC6\x45..){1})(?P<b>(?:\xC7\x45.....){2,})'


for m in re.finditer(string_egg, file_data):
    match_data = m.group(0)
    #print(tohex(match_data))
    key = m['a'][3]
    raw_data = m['b']
    str_data = b''
    for i in range(0,len(raw_data),7):
        str_data += raw_data[i:i+7][-4:]
    print(xor_decrypt(str_data, bytes([key])))
```

```
b'Shell32.dllR'
b'Matanbuc'
b' HTTP/1.'
b'User-Agent: '
b'DllInsta'
b'DllInsta'
b'cmd.exe /c \x05\xdfK\x1b\x05'
b' && rd /s /q'
b'cmd.exe /c \r\xd7C\x13\r'

# .text:7404406B C7 45 AC 71 5D 48 5D                    mov     dword ptr [ebp-54h],
5D485D71h
# .text:74044072 C7 45 B0 52 5E 49 5F                    mov     dword ptr [ebp-50h],
5F495E52h
# .text:74044079 C7 45 B4 54 49 4F 0A                    mov     dword ptr [ebp-4Ch],
0A4F4954h
# .text:74044080 66 C7 45 B8 0A 0A                       mov     word ptr [ebp-48h],
0A0Ah



# .text:74048CF5 C6 45 B8 31                             mov     byte ptr [ebp-48h],
31h ; '1'
# .text:74048CF9 C7 45 E0 75 5D 5D 78                    mov     dword ptr [ebp-20h],
785D5D75h
# .text:74048D00 C7 45 E4 5F 42 45 50                    mov     dword ptr [ebp-1Ch],
5045425Fh
# .text:74048D07 66 C7 45 E8 5D 5D                       mov     word ptr [ebp-18h],
5D5Dh
# .text:74048D0D 88 5D EA                                mov     [ebp-16h], bl
# .text:74048D10 C7 45 EC DA 4E 1E 00                    mov     dword ptr [ebp-14h],
1E4EDAh



# .text:740454A9 30 14 08                                xor     [eax+ecx], dl
# .text:740454AC 41                                      inc     ecx
# .text:740454AD 83 F9 1D                                cmp     ecx, 1Dh

  File "<ipython-input-82-89ec58536910>", line 18
    80 b0 d8 ff 07 74 a8 40 83 f8 06
       ^
SyntaxError: invalid syntax
```

```
from dumpulator import Dumpulator, syscall
from dumpulator.native import *
import pefile


@syscall
def ZwQueryVolumeInformationFile(dp: Dumpulator,
                                 FileHandle: HANDLE,
                                 IoStatusBlock: P(IO_STATUS_BLOCK),
                                 FsInformation: PVOID,
                                 Length: ULONG,
                                 FsInformationClass: FSINFOCLASS
                                 ):
    return STATUS_SUCCESS



dp = Dumpulator("/tmp/b9b.dmp", quiet=True)

dp.start(dp.regs.eip, end=dp.read_ptr(dp.regs.esp))



start_addr = 0x74044390
end_addr = 0x740443C1

dp.start(start_addr, end=end_addr)

dp.read(dp.regs.ebx, 14)

Failed to read module data

bytearray(b'193.56.146.60\\')
```

String Decryption Recap

With these older samples, there are 3 different string decryption methods used

- Stack strings build with DWORDs that are decrypted using a single byte XOR, the byte is the first byte pushed onto the stack string (this is the same method used for small strings in the new samples)
- Global strings that are generated using constructors which have some light obfuscation. To deal with these we simply emulate all of the constructors and scrape the global strings from the `.data` section of the PE.
- The third and most complex method relies on two calls to functions used to build the encrypted string and then a simple single-byte XOR to decrypt the string.

Complex Third String Decryption Method

```
c6 45 c4 53 57 6a 08

740458C7

.text:740436A5 BE DA 4E 1E 00                        mov     esi, 1E4EDAh
.text:740436AA 89 9D 28 FF FF FF                     mov     [ebp+var_D8], ebx
.text:740436B0 89 9D 2C FF FF FF                     mov     [ebp+var_D4], ebx

BE DA ?? ?? ?? 89 ?? ?? ?? ?? ?? 89

.text:740480FF 89 5D B0                              mov     [ebp-50h], ebx
.text:74048102 8B D3                                 mov     edx, ebx
.text:74048104 89 5D B4                              mov     [ebp-4Ch], ebx
.text:74048107 89 5D B8                              mov     [ebp-48h], ebx
.text:7404810A C6 45 BC 1A                           mov     byte ptr [ebp-44h],
1Ah

89 ?? ?? 8b ?? 89 ?? ?? 89 ?? ?? c6 45

.text:740431A2 72 F7                                 jb      short loc_7404319B
.text:740431A4 88 59 0C                              mov     [ecx+12], bl

.text:740438E2 72 F7                                 jb      short loc_740438DB
.text:740438E4 88 59 0B                              mov     [ecx+0Bh], bl
```

```
from dumpulator import Dumpulator, syscall
from dumpulator.native import *
import pefile




@syscall
def ZwQueryVolumeInformationFile(dp: Dumpulator,
                                 FileHandle: HANDLE,
                                 IoStatusBlock: P(IO_STATUS_BLOCK),
                                 FsInformation: PVOID,
                                 Length: ULONG,
                                 FsInformationClass: FSINFOCLASS
                                 ):
    return STATUS_SUCCESS


def emulate(start_addr, end_addr, ret_reg, str_len):
    dp = Dumpulator("/tmp/b9b.dmp", quiet=True)
    dp.start(dp.regs.eip, end=dp.read_ptr(dp.regs.esp))
    dp.start(start_addr, end=end_addr)
    return dp.read(dp.regs.__getitem__(ret_reg), str_len)


pe = pefile.PE(data=file_data)

em_egg = rb'\xBE\xDA...\x89.....\x89.+?(?=\x72\xf7\x88)'


for m in re.finditer(em_egg, file_data):
    start_offset = m.start()
    end_offset = m.end() + 2
    start_addr = pe.get_rva_from_offset(start_offset) + 0x74030000
    end_addr =  pe.get_rva_from_offset(end_offset) + 0x74030000
    str_len = file_data[end_offset + 2]
    if file_data[end_offset + 1] == 0x5f:
        reg_name = 'edi'
    elif file_data[end_offset + 1] == 0x59:
        reg_name = 'ecx'
    print(f"Testing: {hex(start_addr)}")
    try:
        print(emulate(start_addr, end_addr, reg_name, str_len ))
    except:
        print("fail")
        continue
```

```
Testing: 0x74042f48
Failed to read module data
bytearray(b'IPHLPAPI.DLL')
Testing: 0x740430ce
Failed to read module data
bytearray(b'IPHLPAPI.DLL')
Testing: 0x740436a5
Failed to read module data
bytearray(b'\xfb\xcc\xf5\xfc\xd5i\xd2\t\xf7nz')
Testing: 0x74043814
Failed to read module data
bytearray(b'\x98\xe7\xe1\x13-\x0fo\xc0hUY')
Testing: 0x74043991
Failed to read module data
fail
```