Zloader email campaign using MHTML to download and decrypt XLS

hornetsecurity.com/en/threat-research/zloader-email-campaign-using-mhtml-to-download-and-decrypt-xls/

Security Lab March 29, 2021

Summary

Zloader[1] malware (associated with the `kev` configuration tag) is spreading via malspam using MIME encapsulation of aggregate HTML documents (MHTML)[5] attachments. These MHTML files contain a Word document with VBA macros. The VBA macro code downloads and decrypts a password-protected XLS file, and after that, the XLS file decodes and executes the Zloader malware embedded within it.

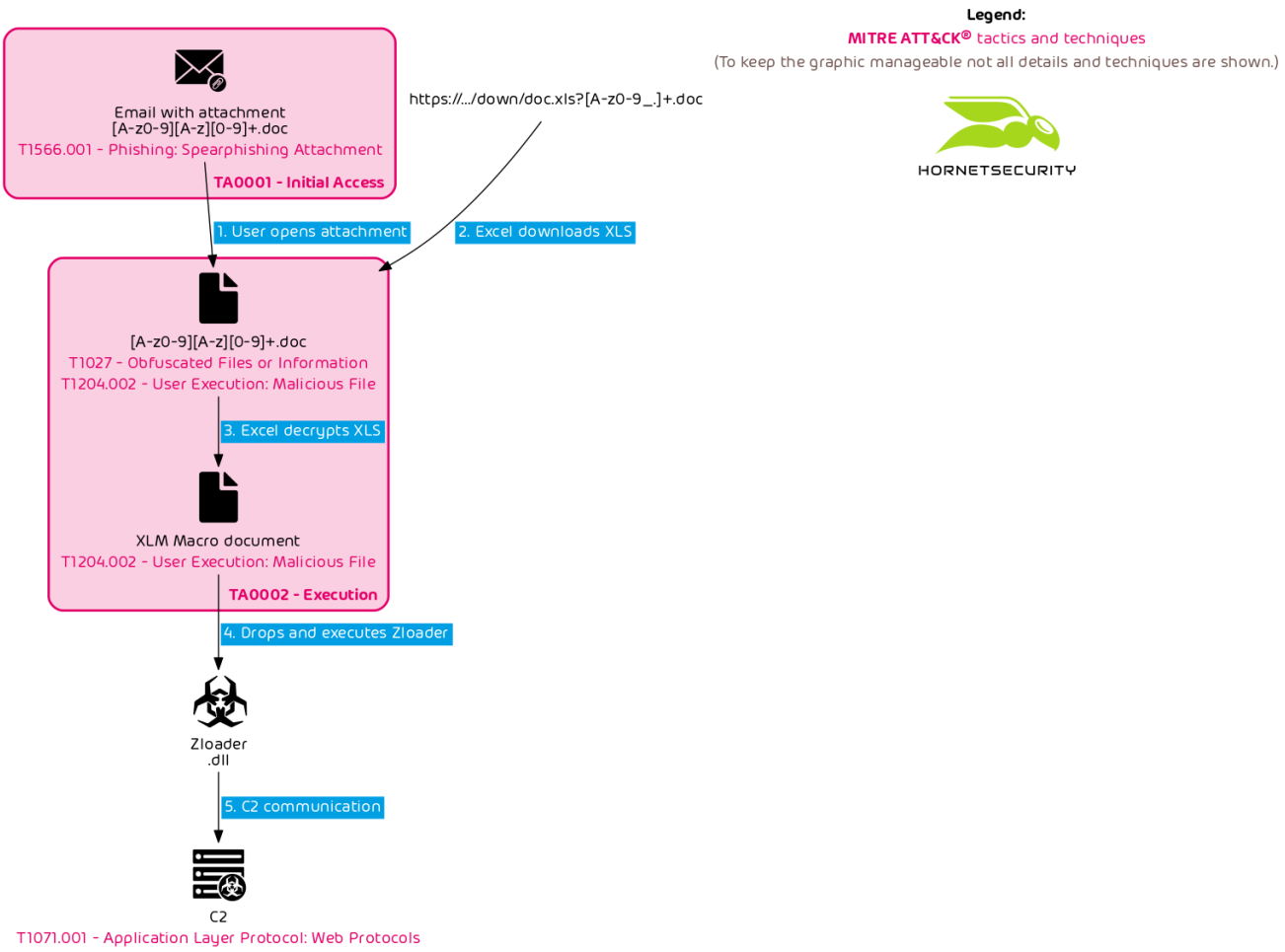
Background

In February 2020, campaigns distributing Zloader ramped up usage of XLM (also known as Excel 4.0) macros. Detection of this old spreadsheet-based by design self-modifiable macro code format by anti-virus software is far lower than detection of regular sequential not by design self-modifiable plain-text VBA macro source code. We already highlighted the abuse of XLM macros in previous reports, e.g., XLM macros used to spread QakBot[2] or BazarLoader[3]. However, as detection for XLM macro code has picked up with even Microsoft adding XLM macro support to AMSI[4], threat actors continue to evolve.

Starting in January 2021, Hornetsecurity took notice of a new Zloader campaign using MHTML attachments. MIME encapsulation of aggregate HTML documents (MHTML)[5] is a web page archive format used to combine multiple files into one. It used base64 encoding and MIME-boundaries similar to multipart MIME encoding in emails. Microsoft Word can open documents stored inside MHTML files.

Technical Analysis

The chain of infection of the Zloader MHTML campaign is as follows:



We will now outline each step of the attack chain.

Emails

The attack starts with emails.

Januar (first wave)

The first emails were designed and built like purchase invoices.

From
Subject **Invoice for purchase order is given approval**
To

Reply | Reply All | Forward | Archive | Junk | Delete | More

1/26/21, 5:11 PM +0000

Good day to you,
Payment we received from you is checked and approved by our financial dept.
Your box would be delivered to your residence within 4 business days.
Please check your details and payslip in the file attached.
Best regards!

Rollins, Inc.
2170 Piedmont Rd NE,
Atlanta, Georgia, 30324,
USA
call: +1 (404) 888-2393

1 attachment: consum_det-168726.doc 259 KB

Save

The wording and pretext changed between emails of this campaign. However, the general "invoicing" theme remained constant.

From
Subject **Bill for online order is given approval**
To

Reply | Reply All | Forward | Archive | Junk | Delete | More

1/26/21, 5:28 PM +0000

Greetings,
Your advance payment is managed and affirmed by our managers.
Your shipment would be delivered to your residence within six days.
Please look at your information and receipt in the file attached.
Regards!

RaceTrac Petroleum Inc
200 Galleria Pkwy SE Suite 900,
Atlanta, Georgia, 30339,
United States
Toll Free: (770) 431-3160

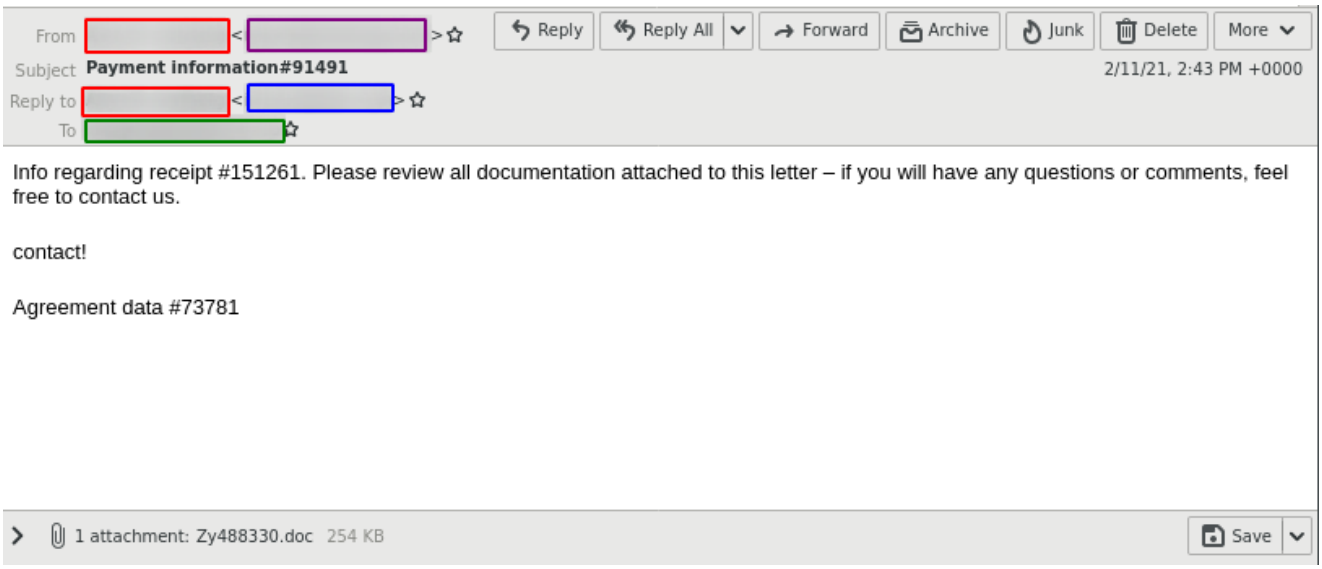
1 attachment: consum_det-832712.doc 257 KB

Save

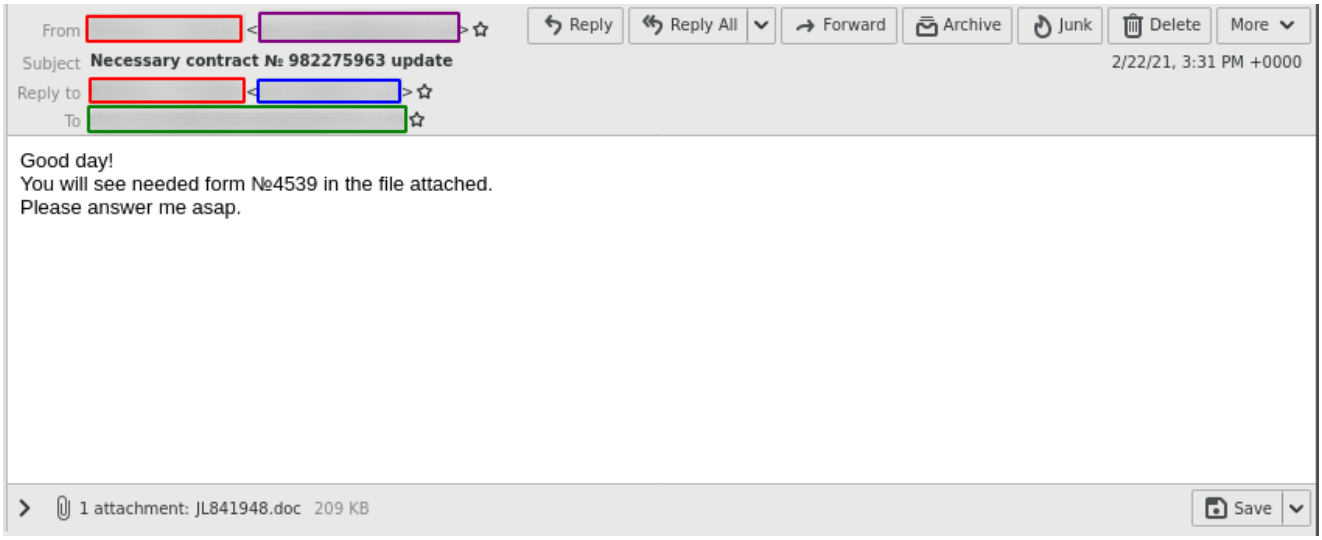
Initial email attacks were of low volume, the emails templates above have not been used much.

February

In February, contract pretexts were added to the mix of invoice pretexts.



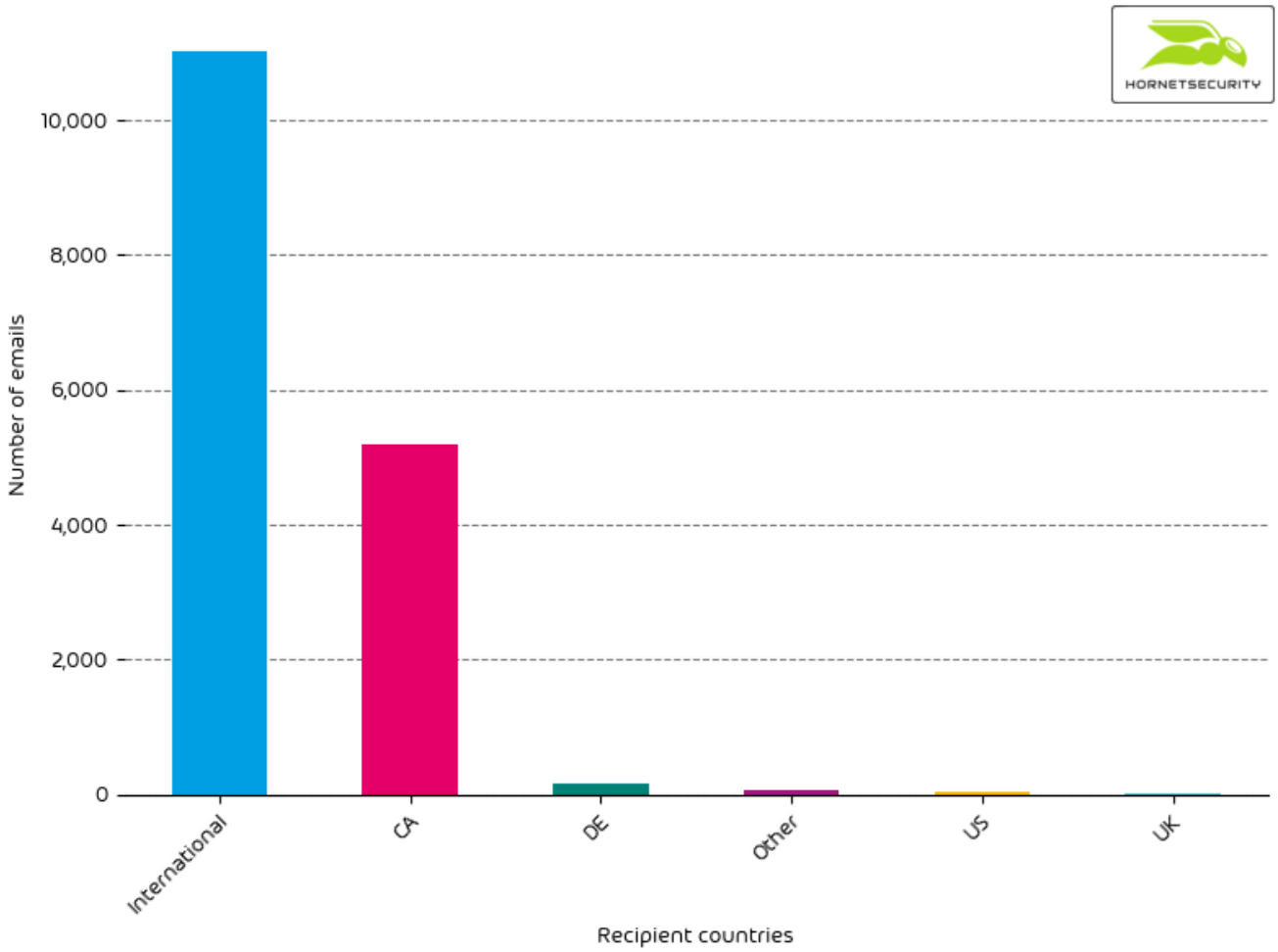
At one point, the numero sign (№) was used instead of the number sign (#).



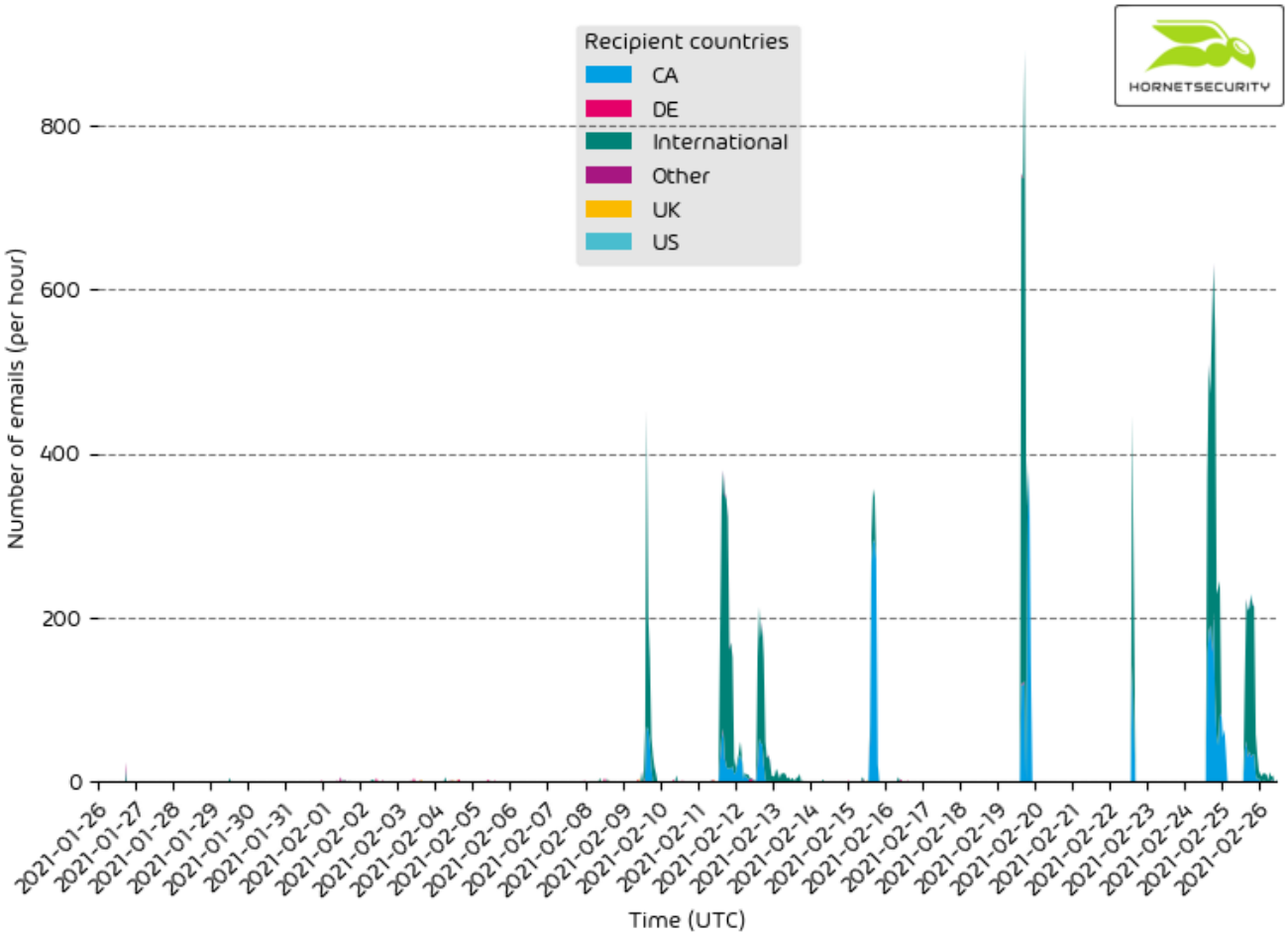
After that, with recent changes to the email template, the campaign's volume started to increase sharply.

Targets

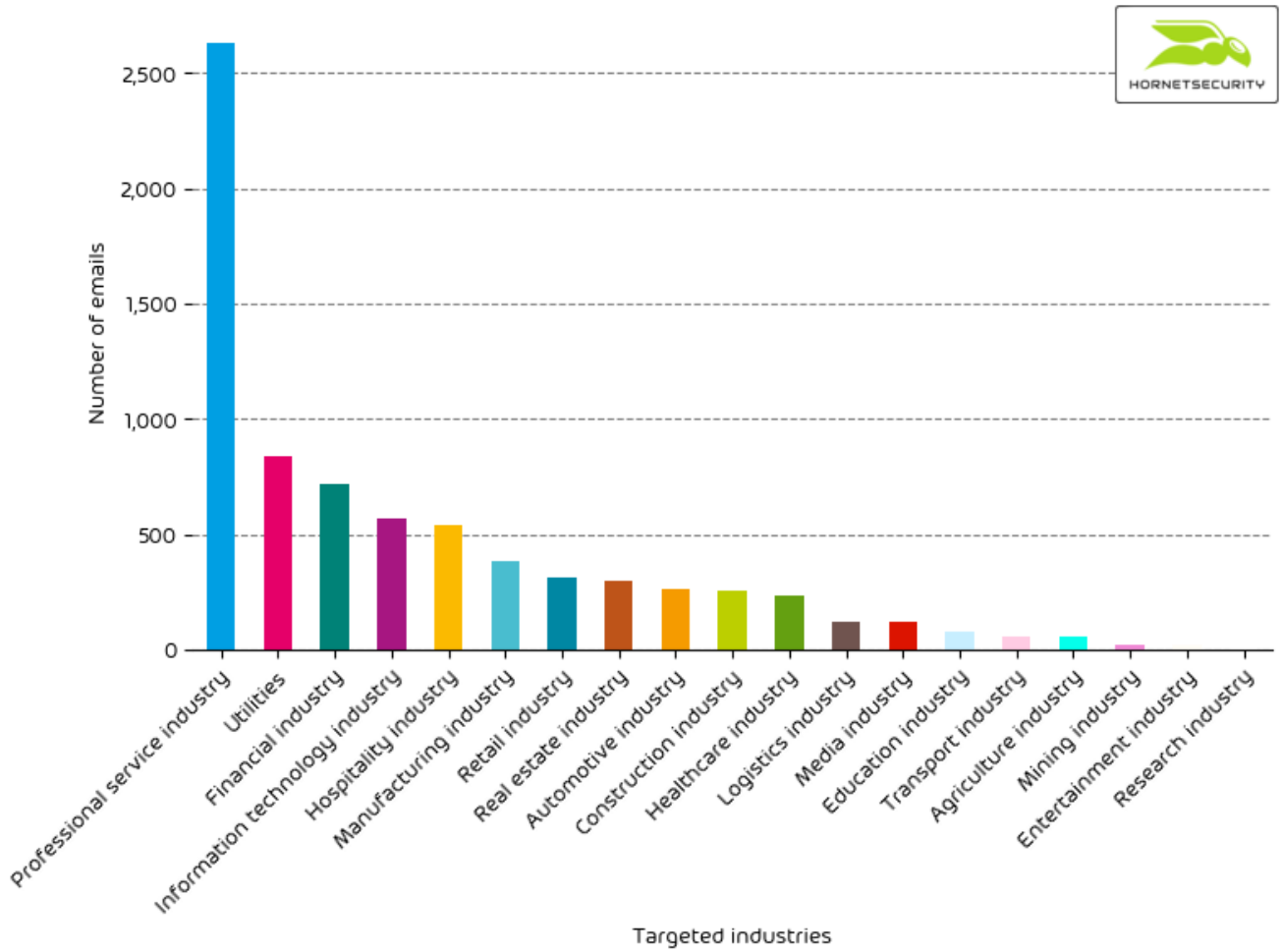
The campaign targets international, Canadian, US, and British companies, mainly English-speaking users.



However, the time histogram shows that on 2021-02-15, the majority of Zloader MHTML emails were destined for Canadian recipients.



The estimated distribution of recipients by industries would suggest a bias towards the professional services industry, i.e., consultancies, freelancers, funeral homes, law firms, etc.

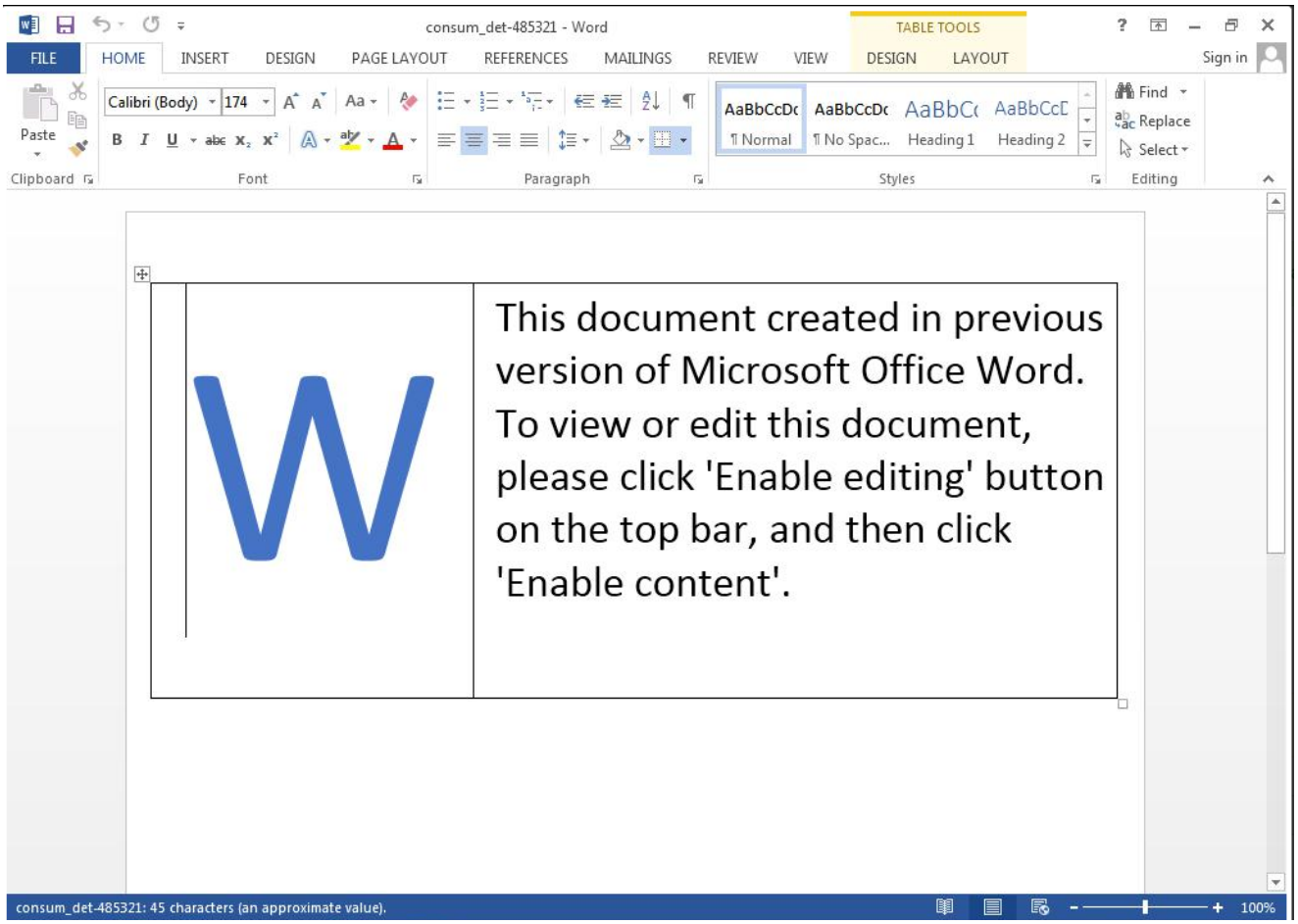


MHTML documents

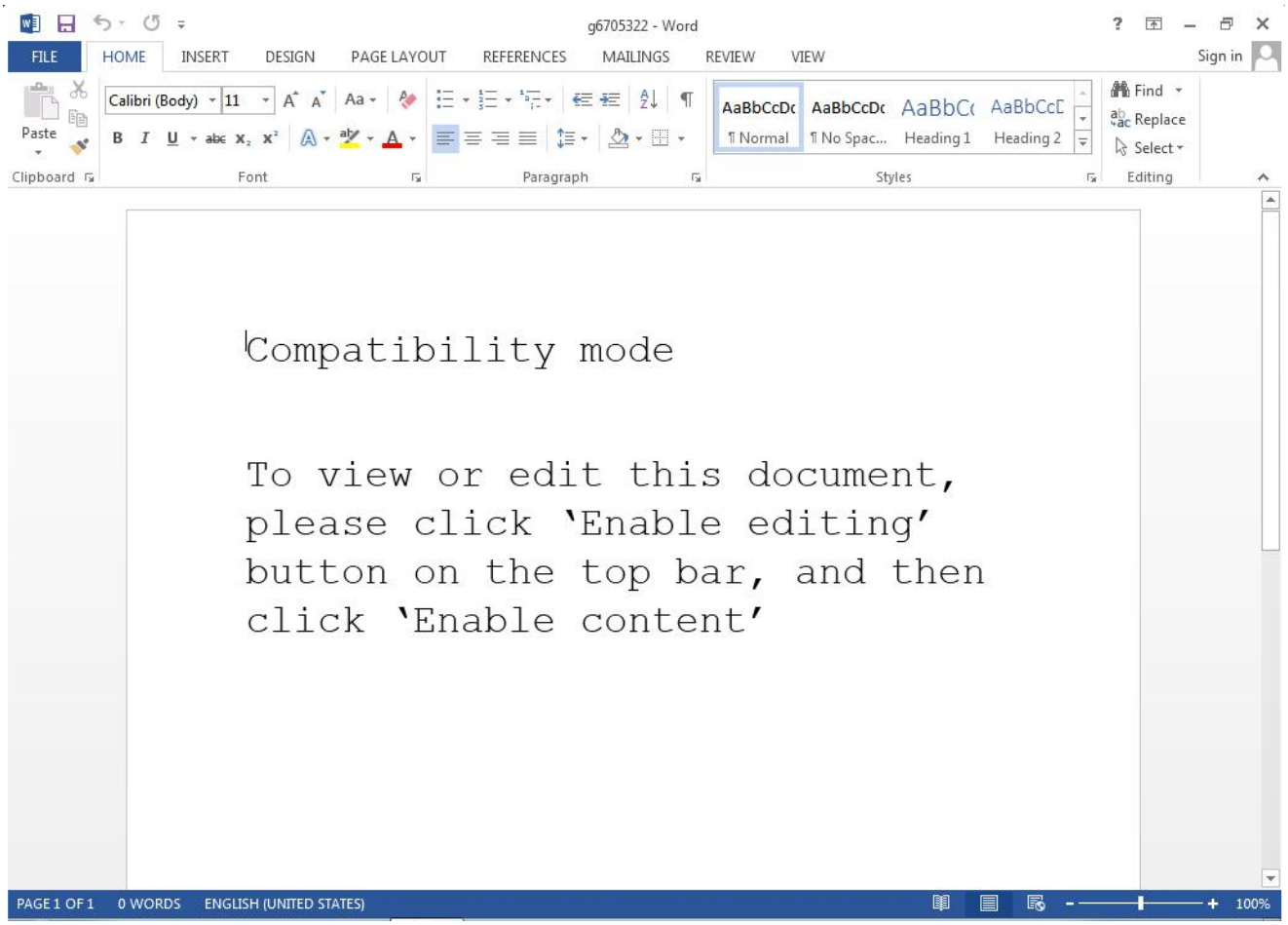
The MHTML document's extension was set to .doc, so Microsoft Word will open the documents directly.

The smaller January campaign and later February campaign MHTML document's main difference is the image instructing the user to "enable content" and "enable editing", i.e., activating macro execution.

The January campaign lure image looks as follows:



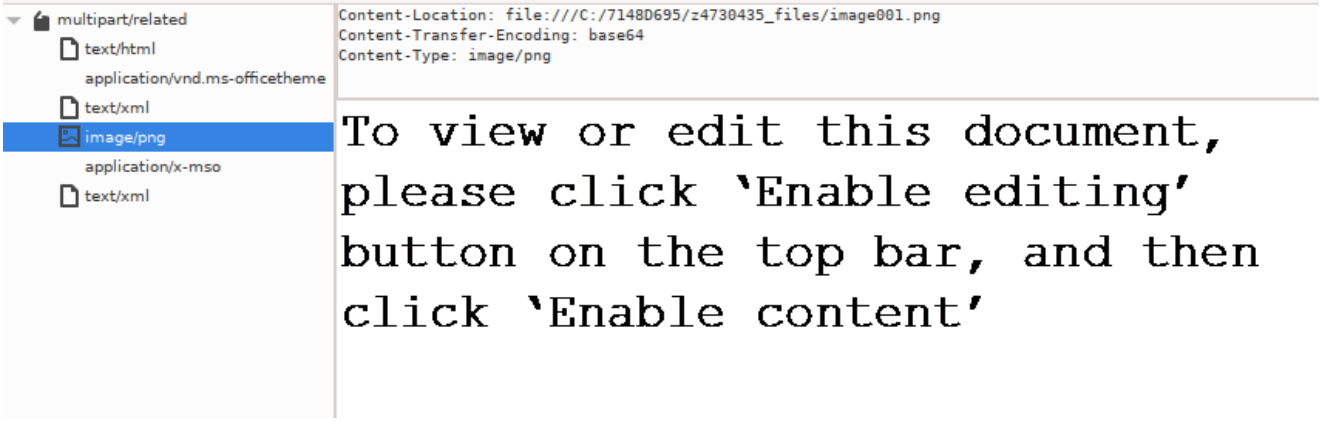
The February campaign lure image looks as follows:



The MHTML document is an ASCII document featuring multiple MIME-parts.

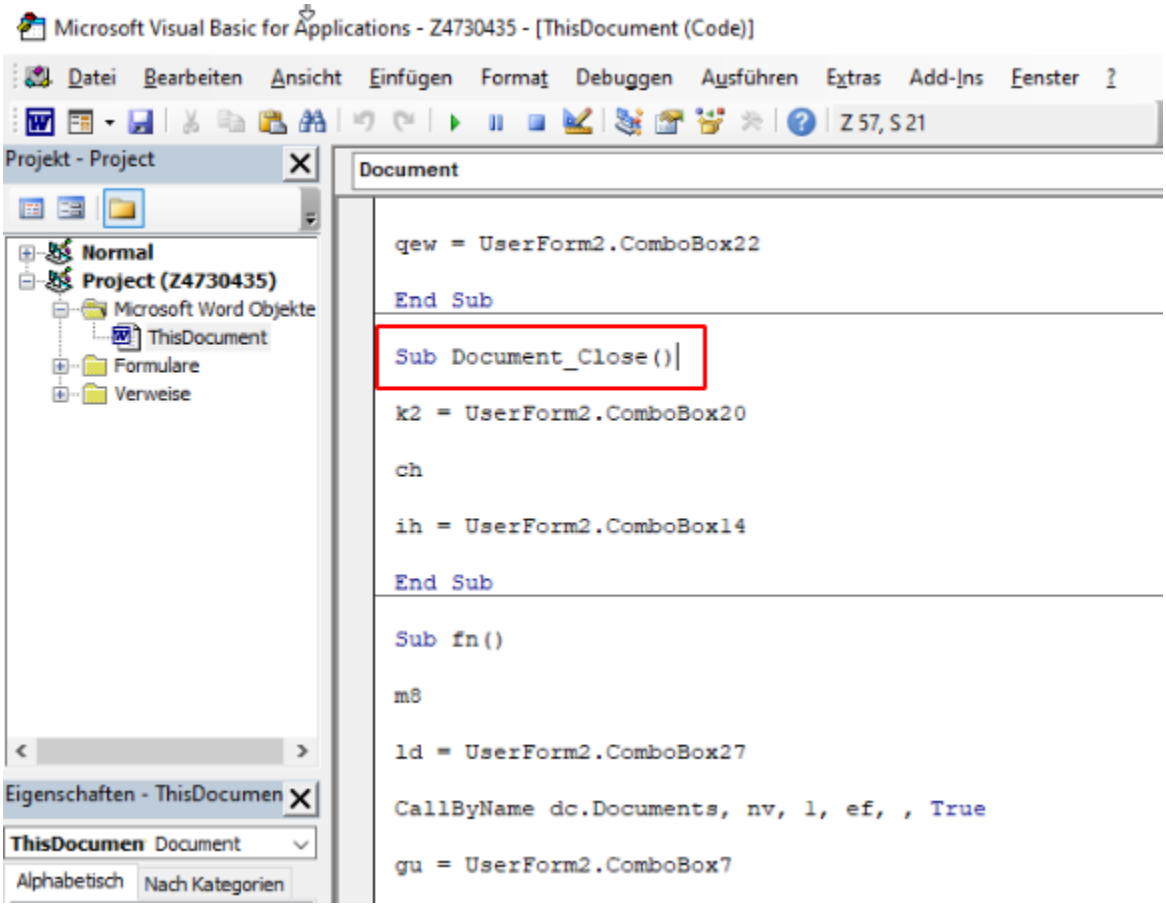
```
$ head g6705322.doc
MIME-Version: 1.0
Content-Type: multipart/related; boundary="----=_NextPart_01D7051D.33731710"

This document is a Single File Web Page, also known as a Web Archive file.  If you are seeing this message, your browser or editor doesn't support Web Archive files.
Please download a browser that supports Web Archive.

------=_NextPart_01D7051D.33731710
Content-Location: file:///C:/F11B5272/g6705322.htm
Content-Transfer-Encoding: quoted-printable
Content-Type: text/html; charset="windows-1252"
```

One MIME-part contains the lure image.

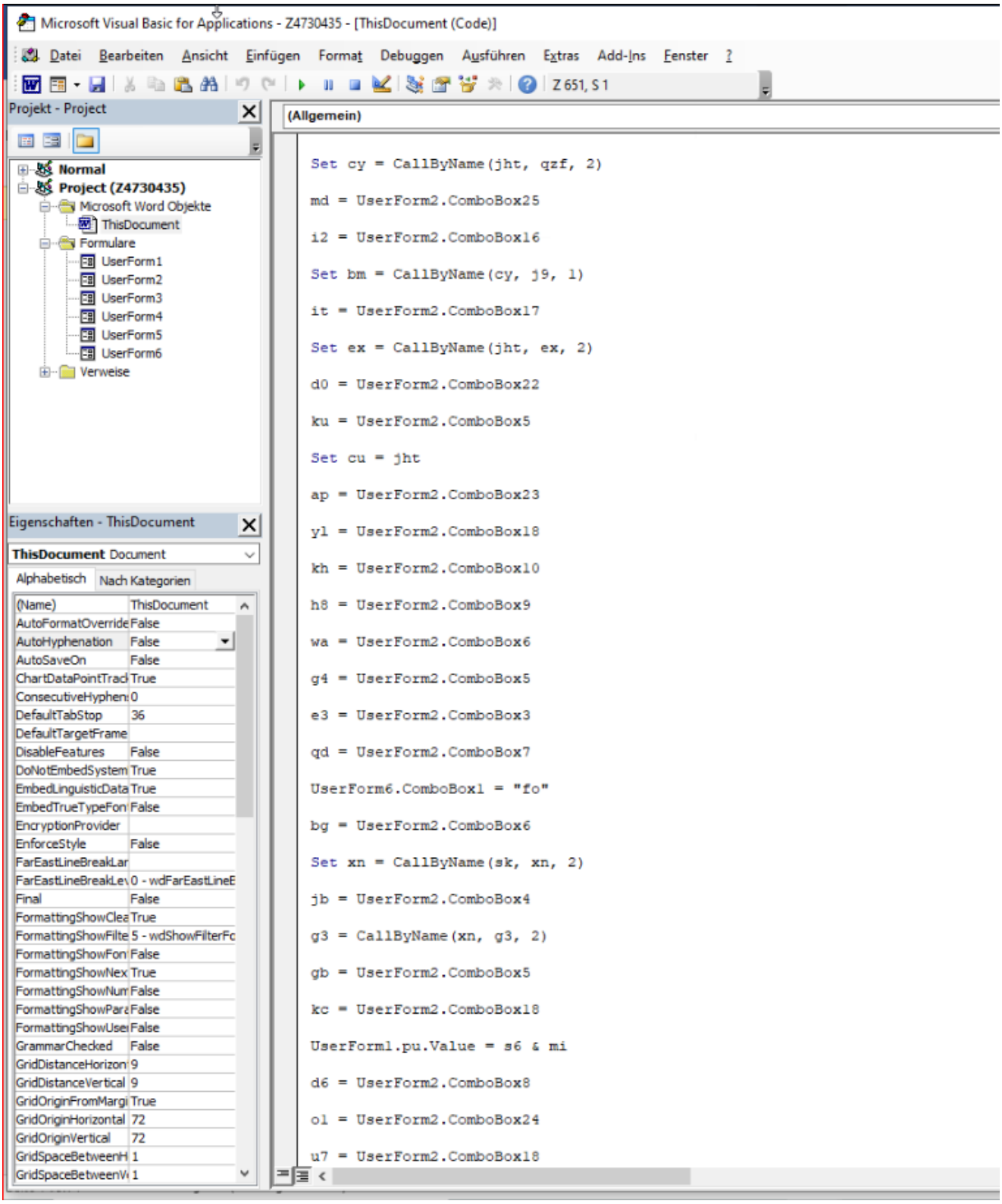


The other parts contain an `application/vnd.ms-officetheme` and an `application/x-mso` file. Which (in addition to the `text/xml` files) are used by Microsoft Word to load the embedded Word document.

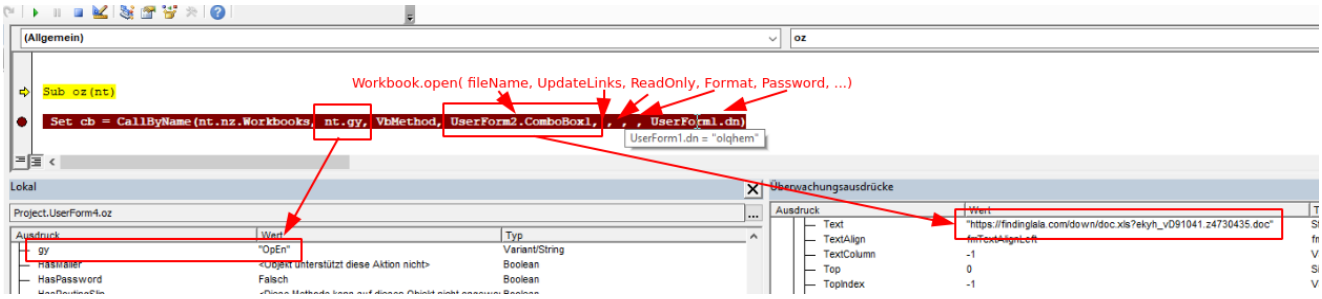
The document will automatically execute the macro code on closing the document:



The VBA code uses `UserForm` objects for obfuscation.



Within the `ComboBox` objects' initialization code in the `UserForm` objects and various other mechanisms, a download URL and a password are assembled and used within a call to the VBA function `CallByName`. This calls the `Workbook` object's `open` function with the `fileName` parameter set to the download URL, the `Password` parameter set to the assembled password, and the other optional parameters left empty.



The call will cause Word to open Excel and download the encrypted XLS file from the URL `https://findinglala[.]com/down/doc.xls?ekyh_vD91041.z4730435.doc`. Excel will use the provided password to decrypt the document.

The XLS document will use XLM macros to decode and use `rundll32.exe` to execute an embedded Zloader payload.

Zloader

Zloader is a fork of the famous Zeus banking Trojan. It is a loader that allows its operator to load additional malware onto infected devices.

Coarse dynamic analysis

The via `rundll32.exe` started Zloader process from the XLS document will spawn a suspended `msiexec.exe` process and inject code into it.

Process Tree

- **rundll32.exe** 7452 *"C:\Windows\system32\rundll32.exe" "C:\Users\Antonia\AppData\Local\Temp\c247aa8ff45f67b7bd27.dll",#1*
  - **msiexec.exe** 540 *msiexec.exe*

First, the original DLL running in `rundll32.exe` starts a `msiexec.exe` process.

NtCreateUserProcess	ProcessHandle: 0x000002a8 ThreadHandle: 0x000002ac ProcessDesiredAccess: 0x02000000 ThreadDesiredAccess: 0x02000000 ProcessFileName: ThreadName: ImagePathName: C:\Windows\SysWOW64\msiexec.exe CommandLine: msiexec.exe ProcessId: 6748	success	0x00000000

Then, `WriteProcessMemory` is used to write code into it.

WriteProcessMemory	ProcessHandle: 0x000002a8 BaseAddress: 0x00380000 Buffer: \xd4\xd4\x81\xec\x9a\x8e\x11\xec\x9d\x8e\x11\xecfq\x11\xec!\x8e\x11\xec\x99\x8e\x11\xec\xd9 \x8e\x11\xec\x99\x8e\x11\xec\x99\x8e\x11\xec\x99\x8e\x11\xec\x99\x8e\x11\xec\x99\x8e\x11\xec\x99 \x8e\x11\xec\x99\x8e\x11\xec\x99\x8e\x11\xecI\x8e\x11\xec\x97\x91\xab\xe2\x99:\x18!\xb86\x10\xa0T\xafE \x84\xf0\xfd1\x9c\xeb\xe1v\x9e\xf8\xe31\x8f\xf8\xe0\x7f\x83\xed\xaes\x89\xb9\xfcd\x82\xb9\xe7\x7f\xcc \xdd\xc1B\xcc\xf4\xe1u\x89\xb7\x83\x1c\xe6\xbd\x8e\x11\xec\x99\x8e\x11\xec\xaaT\x1f2\xee5qa\xee5qa \xee5qa\x8bSp`\xe75qa\xee5pa\xc85qa\xee5qa\xef5qaDks`\xef5qa\xcb\xe7r\x84\xee5qa\x99\x8e\x11\xec \x99\x8e\x11\xec\x99\x8e\x11\xec\x99\x8e\x11\xec\x99\x8e\x11\xec\x99\x8e\x11\xec\xc9\xcb\x11\xec \xd5\x8f\x15\xecD\xcb\xd8\xb3\x99\x8e\x11\xec\x99\x8e\x11\xecy\x8e\x13\xcd\x92\x8f\x1f\xec\x99~\x10 \xec\x99\xc2\x11\xec\x99\x8e\x11\xecIF\x11\xec\x99\x9e\x11\xec BufferLength: 0x00026000 StackPivoted: no	success	0x00000001
GetSystemTime		success	0x00000000
NtAllocateVirtualMemory	ProcessHandle: 0x000002a8 BaseAddress: 0x003b0000 RegionSize: 0x00001000 Protection: PAGE_READWRITE StackPivoted: no	success	0x00000000
WriteProcessMemory	ProcessHandle: 0x000002a8 BaseAddress: 0x003b0000 Buffer: \xbe\x00\x008\x00\xb9\x00`\x02\x00\xb8\x99\xec\x11\x8e\x83\xf9\x00t 0\x06F\xc1\xc0\x08I\xeb \xf2\xe9N\xca\xfd\xff\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 BufferLength: 0x00000042 StackPivoted: no	success	0x00000001

Eventually, the Zloader code running in `rundll32.exe` resumes the thread in the `msiexec.exe` process via `NtResumeThread` and the injected code starts running.

NtGetContextThread	ThreadHandle: 0x000002ac HollowedInstructionPointer: 0x00000000 CurrentInstructionPointer: 0x77053be0 ProcessId: 6748	success	0x00000000
VirtualProtectEx	ProcessHandle: 0x000002a8 Address: 0x003b0000 Size: 0x00000042 MemType: 0x00000000 Protection: PAGE_EXECUTE OldProtection: PAGE_READWRITE StackPivoted: no	success	0x00000001
NtSetContextThread	ThreadHandle: 0x000002ac HollowedInstructionPointer: 0x00000000 CurrentInstructionPointer: 0x003b0000	success	0x00000000
VirtualProtectEx	ProcessHandle: 0x000002a8 Address: 0x00380000 Size: 0x00026000 MemType: 0x00000000 Protection: PAGE_EXECUTE_READWRITE OldProtection: PAGE_READWRITE StackPivoted: no	success	0x00000001
NtResumeThread	ThreadHandle: 0x000002ac SuspendCount: 1 ProcessId: 6748	success	0x00000000
NtTerminateProcess	ProcessHandle: 0x00000000 ExitCode: 0x00000000	success	0x00000000

Zloader will then generate a lot of random directories in `%APPDATA%\Roaming`.

NtQueryAttributesFile	FileName: `C:\Users\Antonia\AppData\Roaming\Pubu`	failed	`OBJECT_NAME_NOT_FOUND`
CreateDirectoryW	DirectoryName: `C:\Users\Antonia\AppData\Roaming\Pubu`	success	`0x00000001`
NtQueryAttributesFile	FileName: `C:\Users\Antonia\AppData\Roaming\Pubu\sapau.dll`	failed	`OBJECT_NAME_NOT_FOUND`
NtQueryAttributesFile	FileName: `C:\Users\Antonia\AppData\Roaming\Olis`	failed	`OBJECT_NAME_NOT_FOUND`
CreateDirectoryW	DirectoryName: `C:\Users\Antonia\AppData\Roaming\Olis`	success	`0x00000001`
NtQueryAttributesFile	FileName: `C:\Users\Antonia\AppData\Roaming\Olis\taeze.se`	failed	`OBJECT_NAME_NOT_FOUND`
NtQueryAttributesFile	FileName: `C:\Users\Antonia\AppData\Roaming\Rolaa`	failed	`OBJECT_NAME_NOT_FOUND`
CreateDirectoryW	DirectoryName: `C:\Users\Antonia\AppData\Roaming\Rolaa`	success	`0x00000001`
NtQueryAttributesFile	FileName: `C:\Users\Antonia\AppData\Roaming\Rolaa\noo.vae`	failed	`OBJECT_NAME_NOT_FOUND`
NtQueryAttributesFile	FileName: `C:\Users\Antonia\AppData\Roaming\Kina`	failed	`OBJECT_NAME_NOT_FOUND`
CreateDirectoryW	DirectoryName: `C:\Users\Antonia\AppData\Roaming\Kina`	success	`0x00000001`
NtQueryAttributesFile	FileName: `C:\Users\Antonia\AppData\Roaming\Kina\zal.eta` ▪ ▪ ▪	failed	`OBJECT_NAME_NOT_FOUND`
NtQueryAttributesFile	FileName: `C:\Users\Antonia\AppData\Roaming\Nai`	failed	`OBJECT_NAME_NOT_FOUND`

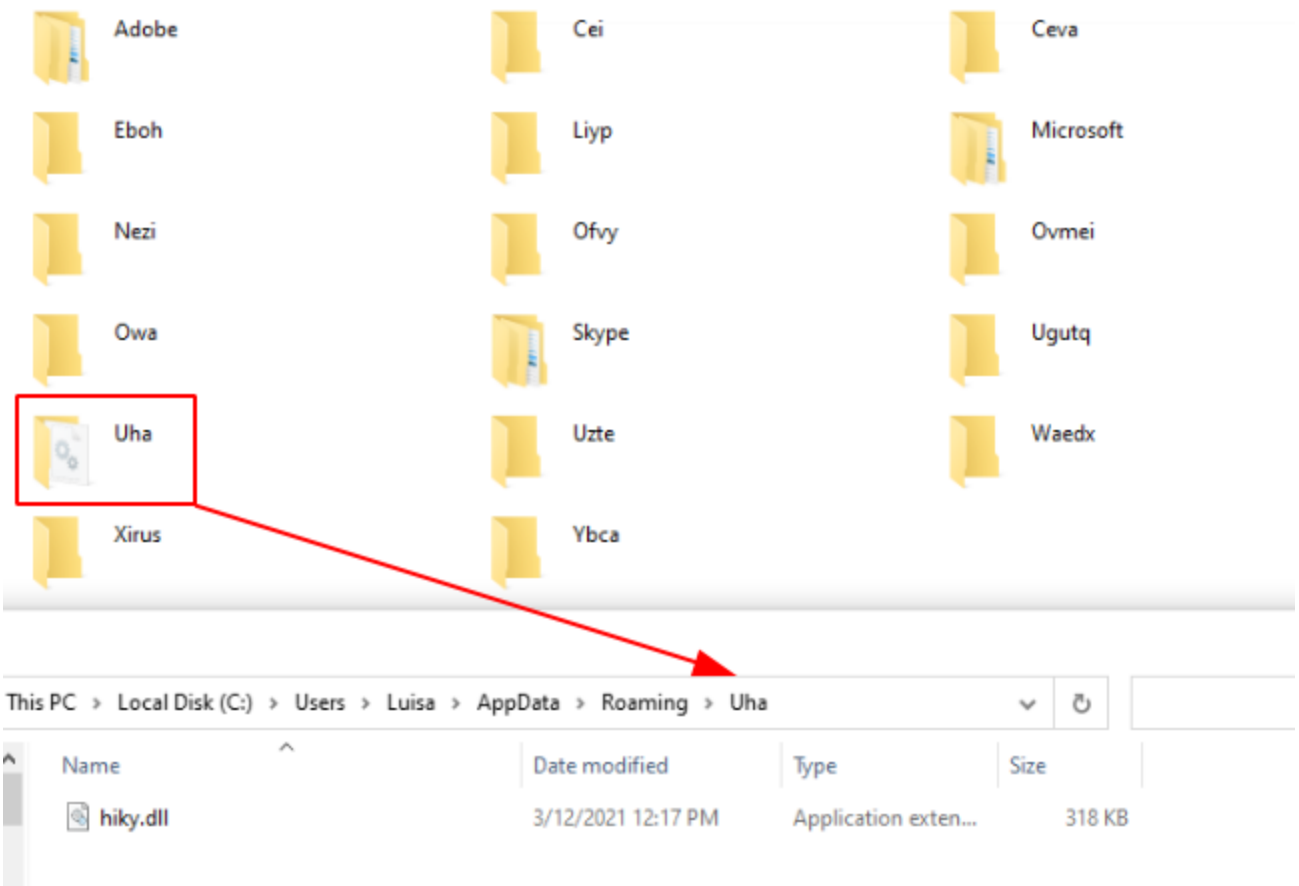
It then copies the original DLL into one of the folders. To this end, it first reads the original DLL into memory.

NtReadFile	FileHandle: `0x0000046c` HandleName: `C:\Users\Antonia\AppData\Local\Temp\c247aa8ff45f67b7bd27.dll` Initial DLL Buffer: `MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff\x00\x00\xb8\x00\x00\x00\x00\x00\x00\x00@\x00 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x10\x01\x00\x00\x0e\x1f\xba\x0e\x00\xb4 \xcd!\xb8 \x01L\xcd!This program cannot be run in DOS mode. $\x00\x00\x00\x00\x00\x00\x00f\x0fZ\|"n4/"n4/"n4/\xff \x91\xfa/&n4/\xff\x91\xe4/#n4//<\xe9/$n4//<\xeb/-n4//<\xd4/3n4//<\xd5/0n4/\xff\x91\xff/.n4/"n5 /{o4//<\xde/\xb5n4//<\xe8/#n4//<\xef/#n4//<\xea/#n4/Rich"n4/\x00\x00\x00\x00\x00\x00\x00\x00` Length: `325120`	success	`0x00000000`

In the next step, data from the DLL is being written back into a new file.

NtCreateFile	FileHandle: `0x00000470` DesiredAccess: `GENERIC_WRITE|FILE_READ_ATTRIBUTES|SYNCHRONIZE` FileName: `C:\Users\Antonia\AppData\Roaming\Pubu\sapau.dll` CreateDisposition: `FILE_OVERWRITE_IF` ShareAccess: `FILE_SHARE_READ` FileAttributes: `FILE_ATTRIBUTE_NORMAL` ExistedBefore: `no` StackPivoted: `no`	success	`0x00000000`
NtWriteFile	FileHandle: `0x00000470` HandleName: `C:\Users\Antonia\AppData\Roaming\Pubu\sapau.dll` Buffer: `MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff\x00\x00\xb8\x00\x00\x00\x00\x00\x00\x00@\x00 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x10\x01\x00\x00\x0e\x1f\xba\x0e\x00\xb4 \xcd!\xb8 \x01L\xcd!This program cannot be run in DOS mode. $\x00\x00\x00\x00\x00\x00\x00f\x0fZ\|"n4/"n4/"n4/\xff \x91\xfa/&n4/\xff\x91\xe4/#n4//<\xe9/$n4//<\xeb/-n4//<\xd4/3n4//<\xd5/0n4/\xff\x91\xff/.n4/"n5 /{o4//<\xde/\xb5n4//<\xe8/#n4//<\xef/#n4//<\xea/#n4/Rich"n4/\x00\x00\x00\x00\x00\x00\x00\x00` Length: `325120`	success	`0x00000000`

The other folders remain empty but can be used at later points in time to hold additional data.



The original DLL is deleted.

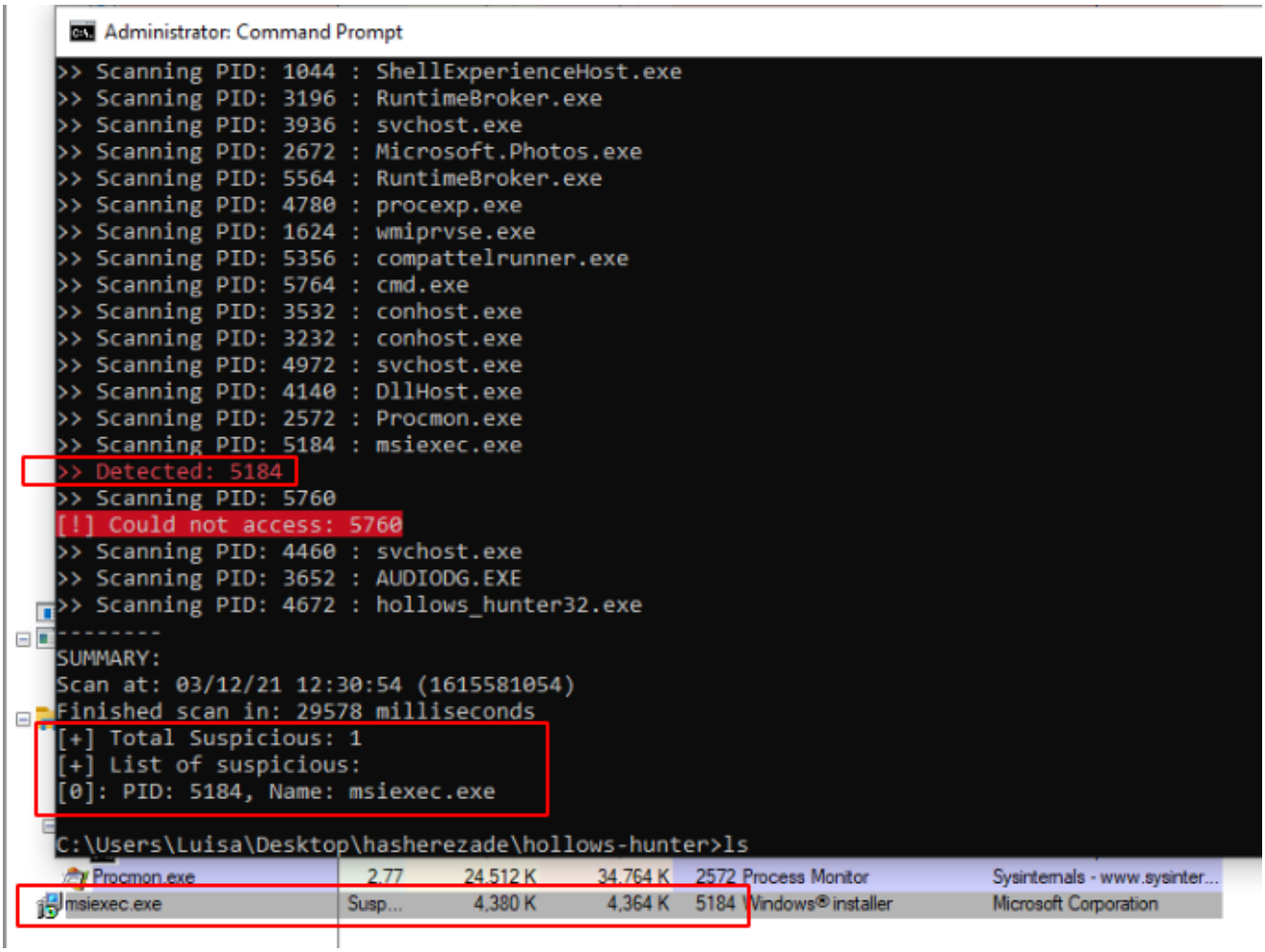
NtCreateFile	FileHandle: `0x000004d4` DesiredAccess: `GENERIC_WRITE|FILE_READ_ATTRIBUTES|DELETE|SYNCHRONIZE` FileName: `C:\Users\Antonia\AppData\Local\Temp\c247aa8ff45f67b7bd27.dll` CreateDisposition: `FILE_OVERWRITE_IF` ShareAccess: `FILE_SHARE_READ|FILE_SHARE_WRITE|FILE_SHARE_DELETE` FileAttributes: `0x00000000` ExistedBefore: `yes` StackPivoted: `no`	success	`0x00000000`
NtQueryAttributesFile	FileName: `C:\Windows\SysWOW64\urlmon.dll`	success	`0x00000000`
NtOpenFile	FileHandle: `0x000004e8` DesiredAccess: `FILE_READ_ACCESS|FILE_EXECUTE|SYNCHRONIZE` FileName: `C:\Windows\SysWOW64\urlmon.dll` ShareAccess: `FILE_SHARE_READ|FILE_SHARE_DELETE`	success	`0x00000000`
NtQueryAttributesFile	FileName: `C:\Users\Antonia\AppData\Local\Temp\c247aa8ff45f67b7bd27.dll`	failed	`OBJECT_NAME_NOT_FOUND`

Eventually, C2 communication initializes.

InternetCrackUrlA	Url: `https://vidhyashram.edu.in/post.php`	success	`0x00000001`
InternetOpenA	Agent: `Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36` AccessType: `0x00000000` ProxyName: ProxyBypass: Flags: `0x00000000`	success	`0x00cc0004`

Unpacking

The Zloader DLL injected into `msiexec.exe` can be extracted either automatically via the open-source CAPE sandbox, manually dumped by breaking on `NtResumeThread` in the original `rundll32.exe` process, then dumping the `msiexec.exe` process, or semi-automatically by using a tool such as `hollows_hunter` [5] (or PE-Sieve).



The following analysis was performed on a Zloader DLL dumped from the `msiexec.exe` process.

Obfuscation

The Zloader malware is obfuscated. It makes extensive use of junk code, i.e., adding program instructions that do not contribute to the program logic with the sole purpose of complicating analysis. Further, it often calls complicated functions to perform trivial calculations, making the code appear very complex. For example, the following is a function that performs the binary AND operation on two parameters. The code is littered with such junk code.

```
Decompile: zl_and - (ba0000.dll)

uint __cdecl zl_and(uint param_1,uint param_2)

{
  byte bVar1;
  uint uVar2;
  int iVar3;
  uint pHVar4;
  uint uVar4;

  pHVar4 = param_2 & param_1;
  uVar4 = (int)(char)param_1 ^ 0xffffffe0;
  iVar3 = uVar4 * param_2;
  uVar2 = zl_xor_with_0x647400ac(0x29e0044f);
  if (((param_1 != param_2) && (iVar3 - param_1 == 0)) && (uVar2 == param_1)) {
    bVar1 = (byte)param_2 * (char)pHVar4 * ((byte)iVar3 & (byte)uVar4);
    FUN_00ba1590((HWND)pHVar4);
    iVar3 = (int)(char)(bVar1 | (byte)param_2) * ((int)(char)bVar1 | pHVar4);
  }
  if (param_1 != param_2) {
    iVar3 = (int)(((((int)(char)iVar3 | 0x10U) + param_1) * param_1 + pHVar4) * -0x34000000 +
                  -0x30000000) >> 0x18;
  }
  DAT_00bc20d8 = iVar3;
  return pHVar4;
}
```

Chunk code (not changing defintion of pHVar4) add as obfuscation

Dynamic library, function and string resolution

Functions are dynamically resolved at runtime via a hash lookup. Instead of calling a function directly, a proxy function returning a pointer to the desired function is called. The following example shows the function with which Zloader deletes its original file. The function we named `zl_get_func` received two parameters, the first is a library ID (`0` is ntdll) and the second is a hash of the function name that should be called.

Decompile: zl_delete_CreateFileW_CloseHandle_file_Sleep_DeleteFileW_GetFileAttributesW ...

```
1
2  int zl_delete_CreateFileW_CloseHandle_file_Sleep_DeleteFileW_GetFileAttributesW(char *filepath)
3
4  {
5    code *func;
6    int return;
7    uint uVar1;
8    int i;
9
10   i = 0;
11   do {
12                        /* CreateFileW */
13     func = (code *)zl_get_func(ntdll,CreateFileW);
14     return = (*func)(filepath,0x40000000,7,0,2,0x4000000,0);
15     uVar1 = zl_obfs_value_00ba4a90(return,0);
16     if ((uVar1 & 1) == 0) {
17                        /* CloseHandle */
18       func = (code *)zl_get_func(0,CloseHandle);
19       (*func)(return);
20     }
21                        /* GetFileAttributesW */
22     func = (code *)zl_get_func(0,GetFileAttributesW);
23     return = (*func)(filepath);
24     if (return == -1) break;
25                        /* DeleteFileW */
26     func = (code *)zl_get_func(0,DeleteFileW);
27     (*func)(filepath);
28                        /* Sleep */
29     func = (code *)zl_get_func(0,Sleep);
30     (*func)(3000);
31     i = i + 1;
32     return = zl_xor_with_0x647400ac(0x647400a6);
33   } while (i != return);
34   zl_heapfree_HeapFree((int)filepath);
35   return 0;
36 }
37
```

This is standard practice in modern malware, so no suspicious imports are present in the binary. It also makes a static analysis more complicated.

Obviously, the hash calculation also uses the previous mentioned junk code obfuscation.

Decompile: zl_hash - (ba0000.dll)

```
uint __cdecl zl_hash(char *func_name,int str_len)

{
  int a;
  uint b;
  uint d;
  uint hash;
  char c;

  if (str_len == -1) {
    str_len = zl_strlen(func_name);
  }
  hash = 0;
  if ((func_name != (char *)0x0) && (0 < str_len)) {
    hash = 0;
    do {
      c = *func_name;
      a = zl_sub(0,hash << 4);
      hash = zl_sub(0,a - (uint)(byte)c);
      b = zl_xor_with_0x647400ac(0x947400ac);
      b = zl_and(~(b ^ hash),hash);
      zl_xor_with_0x647400ac(0x947400ac);
      if (b != 0) {
        d = zl_xor_with_0x647400ac(0x6b8bff53);
        hash = zl_xor(b >> 0x18,d & hash);
      }
      func_name = func_name + 1;
      str_len = str_len + -1;
    } while (str_len != 0);
  }
  return hash;
}
```

However, it can be reimplemented in Python as follows.

```
def zl_hash(func_name):
    func_name = func_name.lower()
    hash = 0
    for c in func_name:
        a = 0 - (hash << 4)
        hash = 0 - (a - ord(c))
        b = 0x647400ac ^ 0x947400ac
        b = hash & (~(b ^ hash))
        if b != 0:
            d = 0x647400ac ^ 0x6b8bff53
            hash = (b >> 0x18) ^ (d & hash)
    return hash
```

With this function calls can be de-obfuscated.

Strings are XOR encoded with a static repeating ASCII keystream.

```
00bc094b 00          ??        00h
00bc094c 00          ??        00h
00bc094d 00          ??        00h
00bc094e 00          ??        00h
00bc094f 00          ??        00h
                 DAT_00bc0950
00bc0950 62          undefined1  62h
                 s_gH?B2eA-StHTW.L5_00bc0951
00bc0951 67 48 3f    ds        "gH?B2eA-StHTW.L5"
         42 32 65
         41 2d 53 ...
                 DAT_00bc0962
00bc0962 6f          ??        6Fh    o
```

XOR key

```
25      if (out_char != 0) {
26        i = 2;
27        j = 1;
28        do {
29          if (0x5e < (ushort)(out_char - 0x20)) {
30            if (0xd < out_char) {
31              return (char *)input;
32            }
33            if ((0x2600U >> (out_char & 0x1f) & 1) == 0) {
34              return (char *)input;
35            }
36          }
37          out_char = (short)(char)PTR_DAT_00bc20ec[j % 0x11] ^ *(ushort *)((int)input + i);
38          *(ushort *)((int)output + i) = out_char;
39          uVar2 = zl_stringdecode_end(out_char,0);
40          i = i + 2;
41          j = j + 1;
42        } while ((uVar2 & 1) == 0);
43      }
```

Configuration

The Zloader configuration is RC4 encrypted with a key using only ASCII as keyspace.

Decompile: zl_startup_init_stuff - (ba0000.dll)

```
1
2  undefined4 zl_startup_init_stuff(void)
3
4  {
5    uint uVar1;
6    byte *pbVar2;
7    undefined4 uVar3;
8    byte local_17 [19];
9
10   uVar1 = FUN_00bad670();
11   if ((char)uVar1 != '\0') {
12     pbVar2 = zl_select_one_of_them(&DAT_00bc0751,local_17);
13     uVar1 = FUN_00bb8830(DAT_00bc2ca8,(char *)pbVar2);
14     if (((char)uVar1 != '\0') && (uVar3 = FUN_00ba6d60(DAT_00bc2ca8), (char)uVar3 != '\0')) {
15       FUN_00bbb180();
16       FUN_00bb6750();
17       zldr_00bbc210_WSAStartup();
18       FUN_00bb0390();
19       FUN_00bb0430();
20       zl_decrypt_config(&zl_encrypted_config,"hbfsijrsbqmqlefskqsxz");
21       uVar1 = zldr_00babcd0_GetModuleFileNameW();
22       if (((char)uVar1 != '\0') &&
23          ((uVar3 = zldr_00babb90_GetLengthSid(), (char)uVar3 != '\0' &&
24           (uVar3 = FUN_00bacf30(), (char)uVar3 != '\0')))) {
25         zldr_00bb0650_GetCurrentProcessId();
26         uVar3 = FUN_00baeeb0();
27         return CONCAT31((int3)((uint)uVar3 >> 8),1);
28       }
29     }
30   }
31   return 0;
32 }
33
```

RC4 key (from a plaintext ASCII keyspace)

The data at the location we labeled `zl_encrypted_config` can be decoded with the ASCII string handed as the second parameter to the function we named `zl_decrypt_config`. Consequently, the configuration of the Zloader sample will be revealed.

```
00bc0ca0  0b 00 1d 00 7f 00 26 00 1a 00 48 00 48 01 00 00  ......&...H.H...
00bc0cb0  6e 75 74 00 00 00 00 00 00 00 00 00 00 00 00 00  nut.............
00bc0cc0  00 00 00 00 00 30 34 2f 30 32 00 00 00 00 00 00  .....04/02......
00bc0cd0  00 00 00 00 00 00 00 00 00 00 68 74 74 70 73 3a  ..........https:
00bc0ce0  2f 2f 76 69 64 68 79 61 73 68 72 61 6d 2e 65 64  //vidhyashram.ed
00bc0cf0  75 2e 69 6e 2f 70 6f 73 74 2e 70 68 70 00 00 00  u.in/post.php...
00bc0d00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00bc0d10  00 00 00 00 00 00 00 00 00 00 00 68 74 74 70 73  ...........https
00bc0d20  3a 2f 2f 63 61 72 6d 65 74 61 2d 61 6d 70 75 68  ://carmeta-ampuh
00bc0d30  2e 63 6f 6d 2f 70 6f 73 74 2e 70 68 70 00 00 00  .com/post.php...
00bc0d40  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00bc0d50  00 00 00 00 00 00 00 00 00 00 00 00 68 74 74 70  ............http
00bc0d60  73 3a 2f 2f 62 65 73 74 61 72 74 69 63 6c 65 62  s://bestarticleb
00bc0d70  6c 6f 67 2e 63 6f 6d 2f 70 6f 73 74 2e 70 68 70  log.com/post.php
00bc0d80  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00bc0d90  00 00 00 00 00 00 00 00 00 00 00 00 00 68 ad 4e  .............h.N
```

The configuration contains a botnet name the particular sample is associated with, a campaign ID (presumably for the threat actors to keep track of infections per campaign), an RC4 key (used to encrypt and decrypt updated configuration stored in the registry) and last but not least a list of command and control URLs the malware should connect to for commands and updates.

We provide an update to the DC3-MWCP script included with the open-source CAPE sandbox that handles configuration extraction for the analyzed Zloader sample in the appendix. It helps automating the configuration extraction.

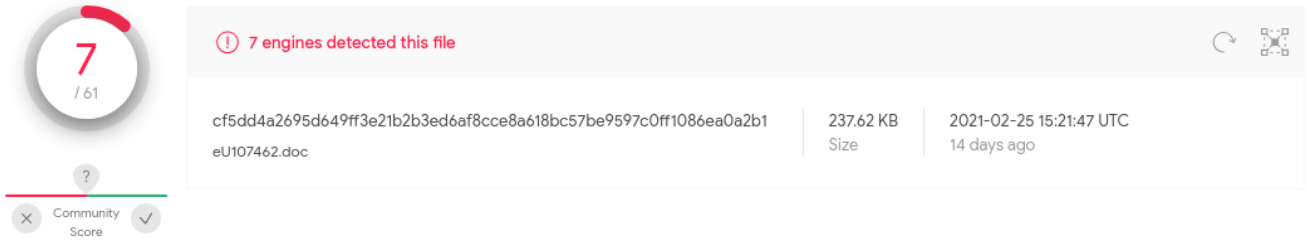
Malware objectives

The configurations of downloaded pieces of Zloader malware associates them with the `kev` botnet. The `kev` ID has been publicly observed since December 2020.

Zloader has been identified as an access vector for Ryuk and Egregor ransomware deployments. Whether the installments associated with the `kev` configuration tag are part of this or a different ransomware operation is currently unknown. However, by the direction the current threat landscape is moving, it is highly likely the malware is also used to deploy ransomware.

Conclusion and Countermeasures

Spreading the attack into multiple encoded stages (document in HMTL; payload URL in `UserForm`s; download of password-protected XLS; decoding of Zloader payload from decrypted XLS) shows that much effort was put into evading detection. Even after the campaign ran for several weeks, the initial MHTML documents still only got 7 out of 61 detections when first scanned on VirusTotal.



The unusual MHTML encoding of the initial Word document can pose problems for security software unfamiliar with this format. Its initial layer must be parsed differently from OLE/CDF/OpenXML-based Office documents and being ASCII plain-text may completely bypass some detections. For network-based protection software, it is impossible to investigate the intermediary downloaded XLS document with the Zloader payload – because it is encrypted. Another struggle is the low level of maliciousness of the initial Word document. While downloads from documents should always be deemed at least suspicious, in this case, only another Excel document was download. Some business workflows may require Word documents to download resources from web. Consequently, the observed behaviour may fly under the radar. Hence, spreading the malicious components (download; dropper; Zloader malware) over multiple stages can bypass detection for some security solutions.

Hornetsecurity's Spam Filtering Solutions and Malware Protection detects and quarantines the outlined threat. Hornetsecurity's Advanced Threat Protection extends this protection by also detecting yet unknown threats.

References

Appendix

DC3-MWCP / CAPE configuration parser

The following DC3-MWCP configuration parser is an update to CAPE's Zloader parser and can be used as a drop in replacement (additionally we opened a pull request with the upstream project):

```
# Copyright (C) 2020 Kevin O'Reilly (kevoreilly@gmail.com)
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

from mwcp.parser import Parser
import struct
import string
import pefile
import yara
import re
from Crypto.Cipher import ARC4
import logging
log = logging.getLogger(__name__)

rule_source = '''
rule Zloader
{
    meta:
        author = "kevoreilly"
        description = "Zloader Payload"
        cape_type = "Zloader Payload"
    strings:
        $rc4_init = {31 [1-3] 66 C7 8? 00 01 00 00 00 00 90 90 [0-5] 8? [5-90] 00 01
00 00 [0-15] (74|75)}
        $decrypt_conf = {e8 ?? ?? ?? ?? e8 ?? ?? ?? ?? e8 ?? ?? ?? ?? e8 ?? ?? ?? ??
68 ?? ?? ?? ?? 68 ?? ?? ?? ?? e8 ?? ?? ?? ?? 83 c4 08 e8 ?? ?? ?? ??}
    condition:
        uint16(0) == 0x5A4D and any of them
}

'''
MAX_STRING_SIZE = 32

yara_rules = yara.compile(source=rule_source)

def decrypt_rc4(key, data):
    cipher = ARC4.new(key)
    return cipher.decrypt(data)

def string_from_offset(data, offset):
    string = data[offset : offset + MAX_STRING_SIZE].split(b"\0")[0]
    return string

class Zloader(Parser):
```

```
    DESCRIPTION = 'Zloader configuration parser'
    AUTHOR = 'kevoreilly'

    def run(self):
        filebuf = self.file_object.file_data
        pe = pefile.PE(data=filebuf, fast_load=False)
        image_base = pe.OPTIONAL_HEADER.ImageBase
        matches = yara_rules.match(data=filebuf)
        if not matches:
            return
        for match in matches:
            if match.rule != "Zloader":
                continue
            for item in match.strings:
                if '$decrypt_conf' in item[1]:
                    decrypt_conf = int(item[0])+21
        va = struct.unpack("I",filebuf[decrypt_conf:decrypt_conf+4])[0]
        key = string_from_offset(filebuf, pe.get_offset_from_rva(va-image_base))
        data_offset =
pe.get_offset_from_rva(struct.unpack("I",filebuf[decrypt_conf+5:decrypt_conf+9])[0]-
image_base)
        enc_data = filebuf[data_offset:].split(b"\0\0")[0]
        raw = decrypt_rc4(key, enc_data)
        items = list(filter(None, raw.split(b'\x00\x00')))
        self.reporter.add_metadata("other", {"Botnet name":
items[1].lstrip(b'\x00')})
        self.reporter.add_metadata("other", {"Campaign ID": items[2]})
        for item in items:
            item = item.lstrip(b'\x00')
            if item.startswith(b'http'):
                self.reporter.add_metadata("address", item)
            elif len(item) == 16:
                self.reporter.add_metadata("other", {"RC4 key": item})
        return
```

Indicators of Compromise (IOCs)

Email

Subjects

- `Agreement information#?[0-9]+`
- `Agreement info#?[0-9]+`
- `Contract info#?[0-9]+`
- `Contract information#?[0-9]+`
- `Payment data#?[0-9]+`
- `Invoicing data#?[0-9]+`
- `Contract data#?[0-9]+`
- `Invoicing information#?[0-9]+`
- `Invoice data#?[0-9]+`

- `Invoicing details#?[0-9]+`
- `Payment info#?[0-9]+`
- `Invoicing info#?[0-9]+`
- `Agreement data#?[0-9]+`
- `Contract details#?[0-9]+`
- `Invoice information#?[0-9]+`
- `Invoice details#?[0-9]+`
- `Payment information#?[0-9]+`
- `Invoice info#?[0-9]+`
- `Agreement details#?[0-9]+`
- `Payment details#?[0-9]+`
- `Important agreement #?[0-9]+ update`
- `Essential contract No. #?[0-9]+ update`
- `Essential agreement No. #?[0-9]+ documentation`
- `Important contract No. #?[0-9]+ update`
- `Important contract №#?[0-9]+ update`
- `Essential agreement Number #?[0-9]+ update`
- `Necessary contract №#?[0-9]+ update`
- `Important contract № #?[0-9]+ update`
- `Important agreement No. #?[0-9]+ documentation`
- `Important agreement #?[0-9]+ documentation`
- `Necessary agreement Number #?[0-9]+ documentation`
- `Important contract № #?[0-9]+ documentation`
- `Important contract #?[0-9]+ documentation`
- `Important agreement № #?[0-9]+ documentation`
- `Essential contract #?[0-9]+ update`
- `Essential agreement No. #?[0-9]+ update`
- `Necessary agreement Number #?[0-9]+ update`
- `Necessary contract № #?[0-9]+ update`
- `Important agreement Number #?[0-9]+ update`
- `Essential contract No. #?[0-9]+ documentation`
- `Important contract #?[0-9]+ update`
- `Essential agreement #?[0-9]+ update`
- `Necessary agreement № #?[0-9]+ update`
- `Important agreement No. #?[0-9]+ update`
- `Necessary contract Number #?[0-9]+ documentation`
- `Necessary contract No. #?[0-9]+ update`
- `Necessary agreement #?[0-9]+ update`
- `Essential contract Number #?[0-9]+ documentation`
- `Essential contract № #?[0-9]+ documentation`
- `Essential agreement №#?[0-9]+ update`
- `Necessary agreement №#?[0-9]+ update`

- `Essential contract Number #?[0-9]+ update`
- `Necessary contract No. #?[0-9]+ documentation`
- `Important contract №#?[0-9]+ documentation`
- `Important agreement № #?[0-9]+ update`
- `Essential contract №#?[0-9]+ documentation`
- `Essential contract #?[0-9]+ documentation`
- `Necessary contract №#?[0-9]+ documentation`
- `Necessary agreement #?[0-9]+ documentation`
- `Important contract Number #?[0-9]+ documentation`
- `Necessary contract #?[0-9]+ update`
- `Important agreement №#?[0-9]+ update`
- `Essential contract №#?[0-9]+ update`
- `Essential agreement №#?[0-9]+ documentation`
- `Necessary contract #?[0-9]+ documentation`
- `Important agreement Number #?[0-9]+ documentation`
- `Essential agreement № #?[0-9]+ documentation`
- `Necessary agreement № #?[0-9]+ documentation`
- `Essential agreement Number #?[0-9]+ documentation`
- `Essential contract № #?[0-9]+ update`
- `Necessary contract Number #?[0-9]+ update`
- `Necessary agreement No. #?[0-9]+ update`
- `Essential agreement № #?[0-9]+ update`
- `Essential agreement #?[0-9]+ documentation`
- `Important contract No. #?[0-9]+ documentation`
- `Necessary agreement No. #?[0-9]+ documentation`
- `Important contract Number #?[0-9]+ update`
- `Necessary contract № #?[0-9]+ documentation`
- `Necessary agreement №#?[0-9]+ documentation`
- `Important agreement №#?[0-9]+ documentation`

Attachments

The following regular expressions describe the attachment names used in the campaigns:

- `([a-z]{4,8}_){1,2}[a-z]{4,8}[0-9]+.doc`
- `[A-z0-9][A-z][0-9]+.doc`

Hashes

Hashes of publicly available files:

MD5	Filename	Description
`6743ca84f7e9929c2179238e20934f57`	`nG772044.doc`	Zloader MHTML document

MD5	Filename	Description
`7a888f899a4850f02bad194bf01daaa7`	`eU107462.doc`	Zloader MHTML document
`35ee0681eb3076674e01efec565f663b`	`L1978883.doc`	Zloader MHTML document
`25e2cffc5621cab99bd0a36d234c234f`	`QG915014.doc`	Zloader MHTML document
`222cb61e1041f3e4dbdc3493572388e6`	`dY433632.doc`	Zloader MHTML document

URLs

`https://findinglala[.]com/down/doc.xls?ekyh_vD91041.z4730435.doc`

DNS

- `findinglala[.]com`
- `funkstarnews[.]com`
- `heavenlygem[.]com`
- `2tut[.]com`
- `khalilmouna[.]com`