Ero Carrera & Peter Silberman

STATE OF MALWARE: FAMILY TIES

Who are we?

- Ero Carrera
  - Researcher at VirusTotal / Zynamics GmbH

- Peter Silberman
  - Researcher/Engineer at MANDIANT

Terms and Definitions

- Mass Malware (MM) – malware written for distribution across the internet targeting hundreds of thousands or millions of computers.
- Targeted Malware – malware written specifically for a target attack. Seen on very few networks.

Background: Zynamics

Zynamics GmbH develops advanced analysis and research tools in the computer security arena.

BinNavi and **BinDiif**, two of its flagship products focus on binary analysis while **VxClass** is an automated environment for the analysis and classification of executable code, with an emphasis on malware

- We will use VxClass' results to attempt to correlate the samples we collected for this talk
- Samples of malware were obtained through VirusTotal's VTMIS (VirusTotal Malware Intelligence Service)

Background: MANDIANT

MANDIANT is a company of consultants, authors, instructors and security experts. We work with the Fortune 500, the defense industrial base and the banks of the world to secure their networks and combat cyber-crime. We have testified in court and helped bring many of these criminals to justice.

- MANDIANT has collected and analyzed over 300 unique APT samples, including seven of the *Fortune 50* and many other fortune 500, defense and financial sectors.
- **Bottom Line:** APT is everywhere you wish you were ☺

Malware Families

- Malware has been classified into related clusters
  - Referred to as families
- Allows for:
  - tracking of authorship
  - correlating information
  - Identifying new variants

Mass Malware Families

- Major families covered in our study: Sinowal, Mebroot, Conficker/Downadup, Waledac, WSnPoem/Zeus, Bredolab, Srizbi, Rustock, Poisonivy, zbot, Bobax/Kraken, Pandex, Koobface, Cutwail, Nuwar/Peacomm, RlsLoup, Tedroo, Xarverster
- Features of these families: many…

Targeted Malware

- aka APT (oh god….)
- Targeted Malware is clustered into families
- Families indicate:
  - Capabilities
    - Malware
    - Attackers
    - Authors
  - Remediation output effort
    - Likelihood of successful remediation

Hypothesis

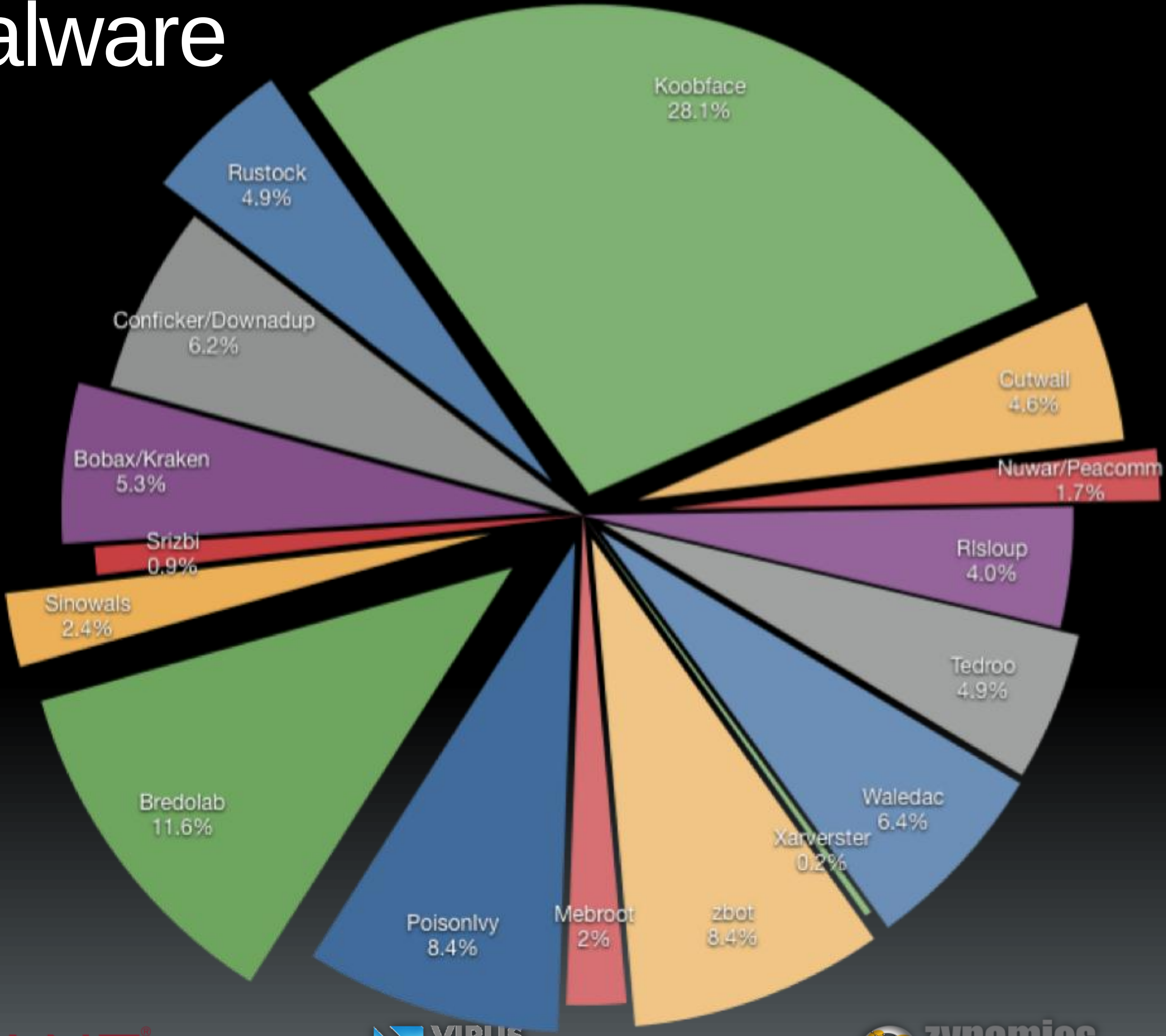
- We have a hypothesis about the relationships of:
  - Mass malware
  - Rootkits
  - Targeted malware

Mass Malware

Mass Malware

- We collected samples from many of the major families of malware
- We attempted to obtain clues of code-reuse among families
- The results are negative with a high probability (we haven’t checked every single little function). There is no large-scale code sharing

The Malware

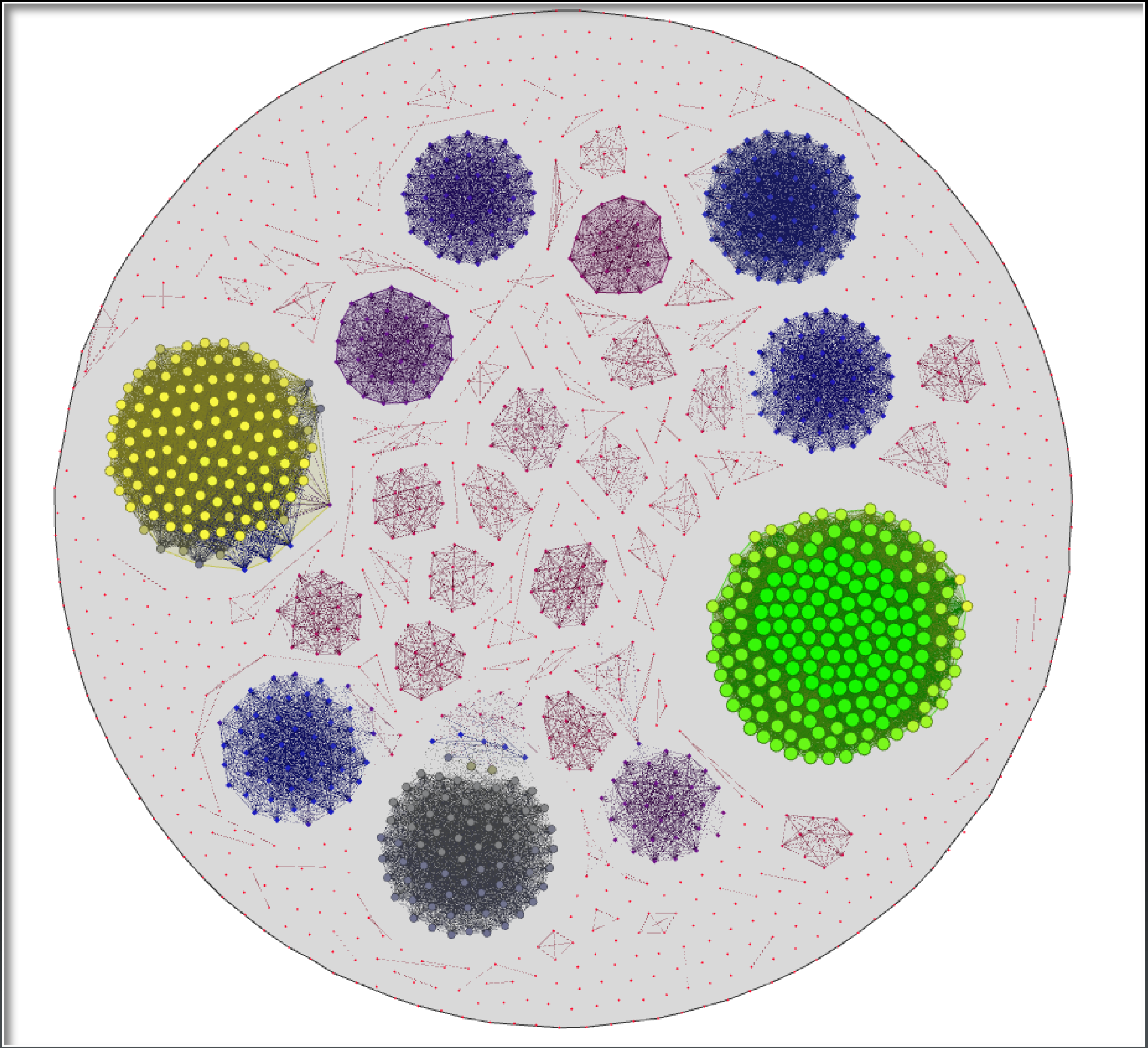

Koobface 28.1%
Cutwail 4.6%
Nuwar/Peacomm 1.7%
Rlsloup 4.0%
Tedroo 4.9%
Waledac 6.4%
Xarverster 0.2%
zbot 8.4%
Mebroot 2%
PoisonIvy 8.4%
Bredolab 11.6%
Sinowals 2.4%
Srizbi 0.9%
Bobax/Kraken 5.3%
Conficker/Downadup 6.2%
Rustock 4.9%

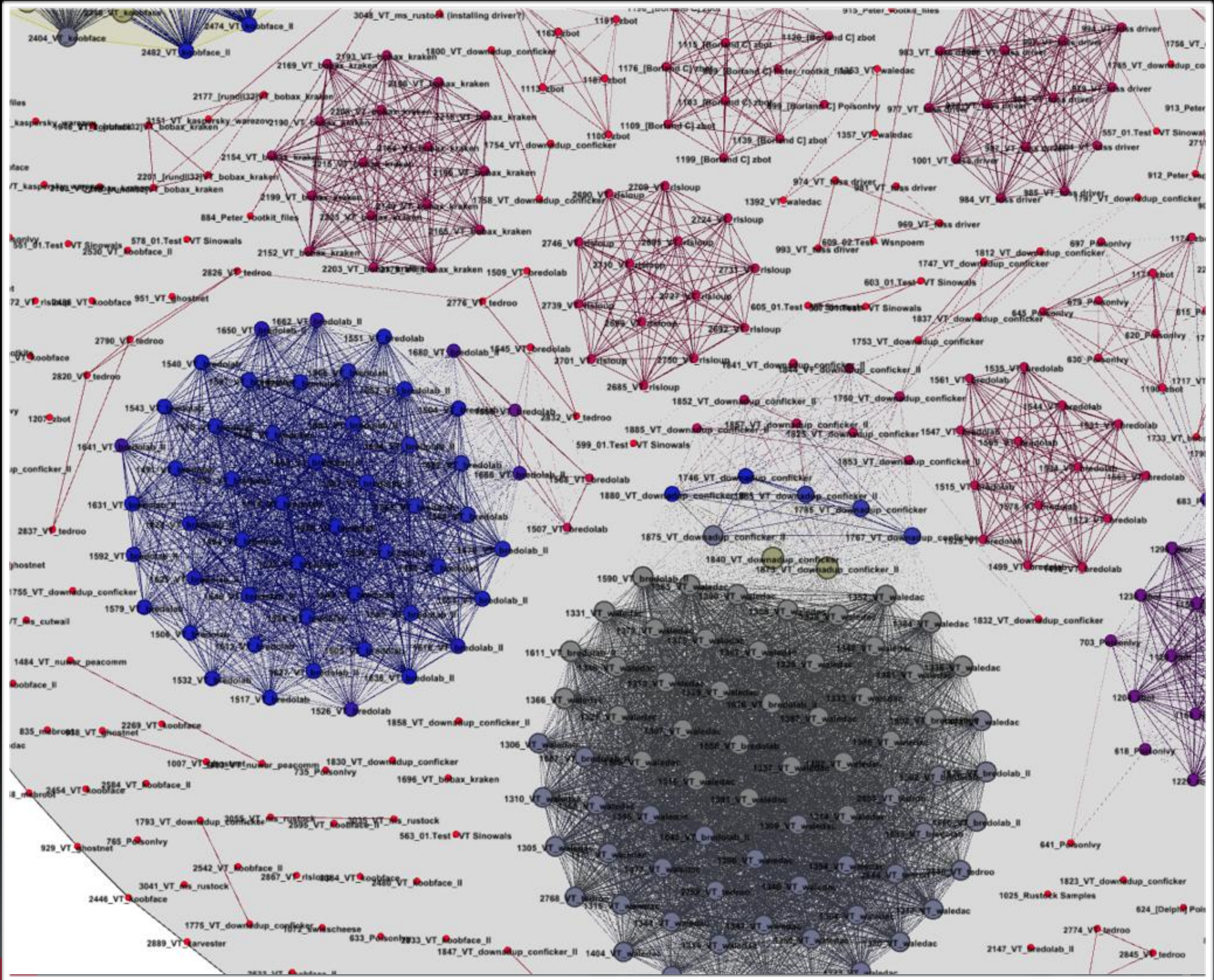
Movie time!

Results

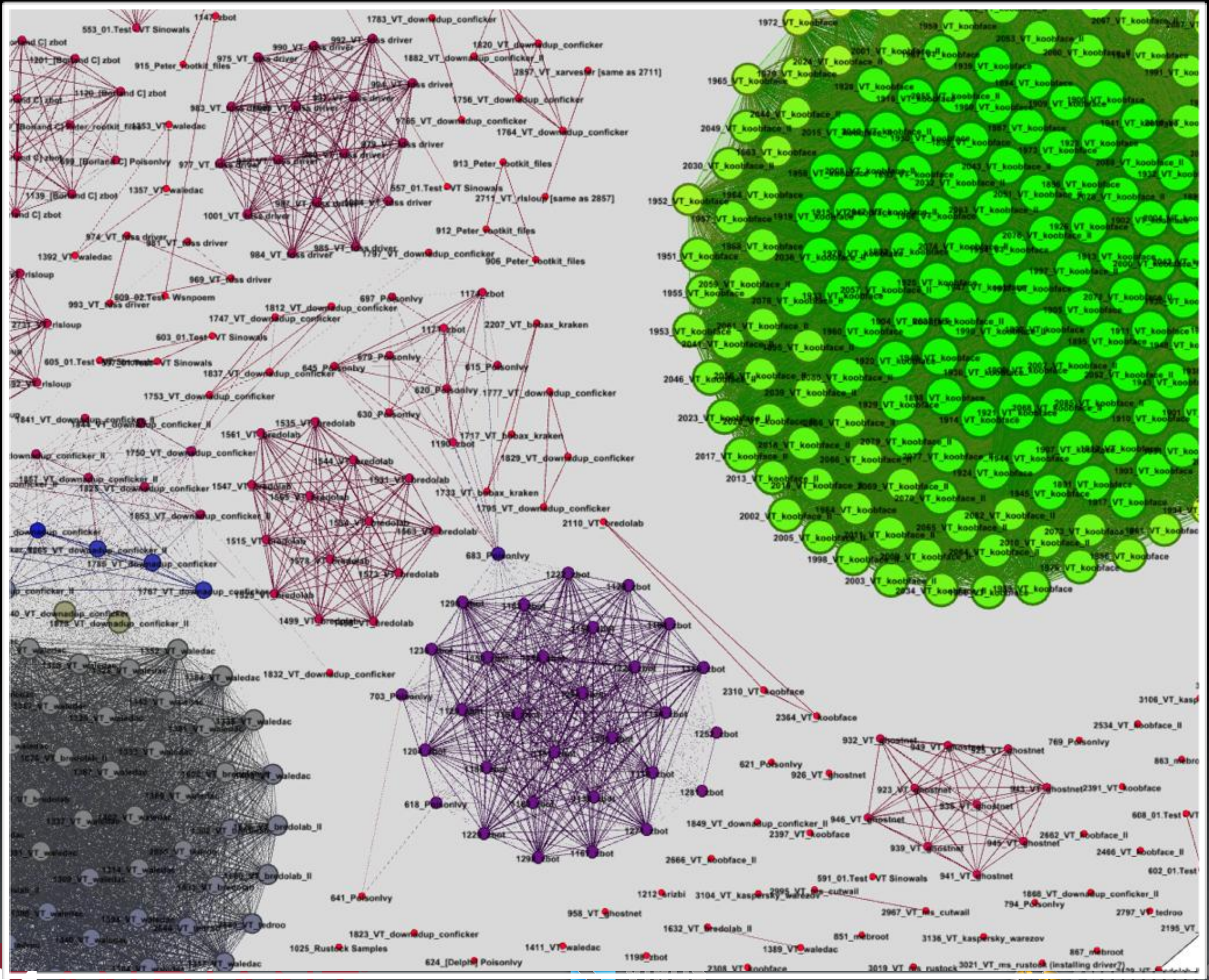
How do you feel about colorful diagrams?!

- A few hundred pieces of malware classified
- The cut-off threshold was set to 0.6 (60% similarity or more)
- Strong intra-family relations are obvious

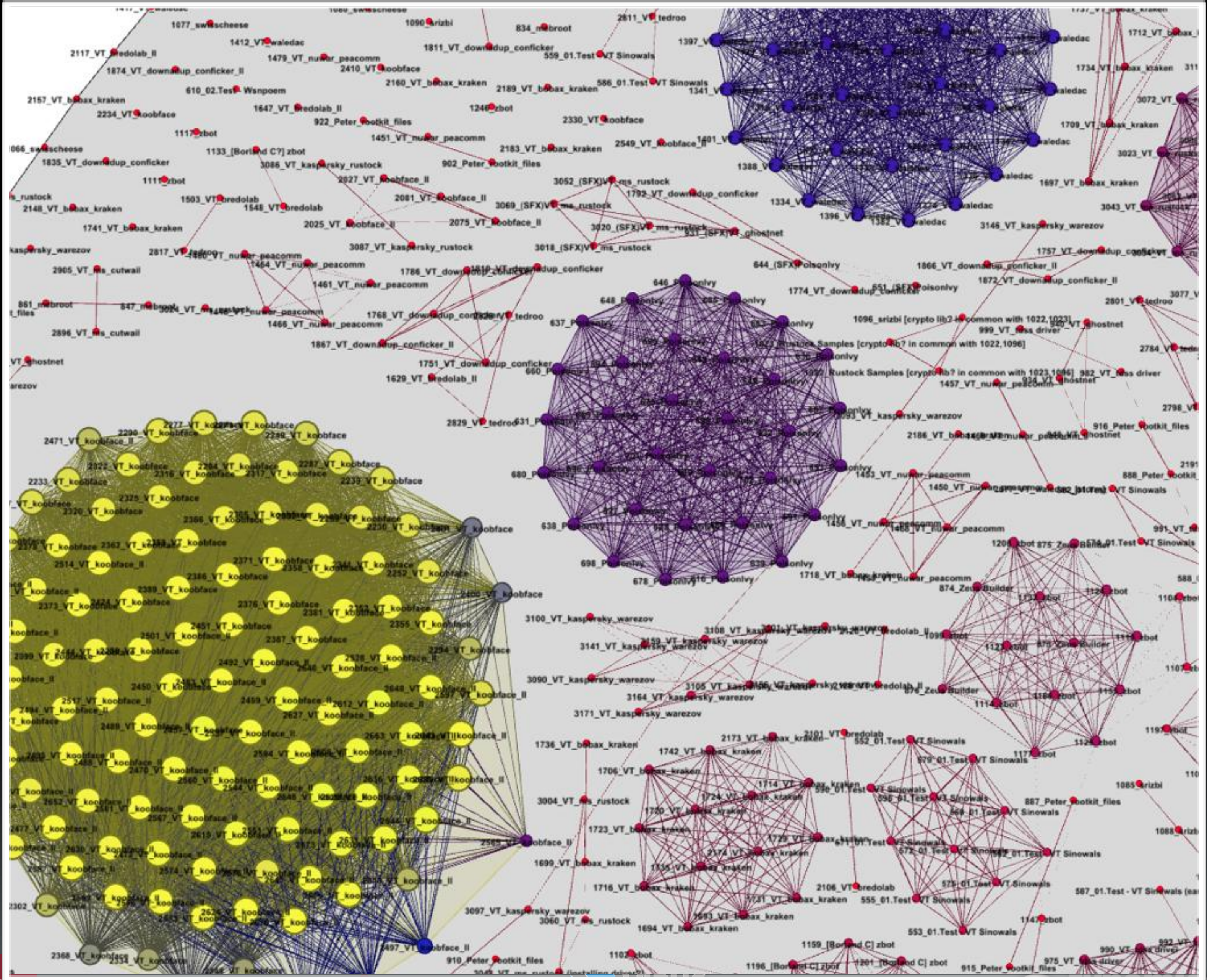




3048_VT_ms_rustock (installing driver?)
2404_VT_koobface
2474_VT_koobface_II
2482_VT_koobface_II
2169_VT_bobax_kraken
2193_VT_bobax_kraken
1800_VT_downadup_conficker
2177_[rundll32]VT_bobax_kraken
2151_VT_kaspersky_warezov
2154_VT_bobax_kraken
2201_[rundll32]VT_bobax_kraken
2199_VT_bobax_kraken
884_Peter_rootkit_files
2152_VT_bobax_kraken
2203_VT_bobax_kraken
1754_VT_downadup_conficker
1758_VT_downadup_conficker
1162_zbot
1161_zbot
1113_zbot
1187_zbot
1100_zbot
1115_[Borland C] zbot
1120_[Borland C] zbot
1176_[Borland C] zbot
1109_[Borland C] zbot
1199_[Borland C] zbot
1139_[Borland C] zbot
1357_VT_waledac
1392_VT_waledac
983_VT_fuss driver
913_Peter
557_01.Test VT Sinowals
1001_VT_fuss driver
984_VT_fuss driver
912_Peter
1797_VT_downadup_conficker
969_VT_fuss driver
993_VT_fuss driver
609_02.Test VT Wsnpoem
1812_VT_downadup_conficker
1747_VT_downadup_conficker
603_01.Test VT Sinowals
605_01.Test VT Sinowals
679_Poisonlvy
697_Poisonlvy
645_Poisonlvy
620_Poisonlvy
630_Poisonlvy
1837_VT_downadup_conficker
1753_VT_downadup_conficker
2709_VT_risloup
2690_VT_risloup
2724_VT_risloup
2746_VT_risloup
2731_VT_risloup
2739_VT_risloup
2692_VT_risloup
2701_VT_risloup
2750_VT_risloup
2685_VT_risloup
578_01.Test VT Sinowals
581_01.Test VT Sinowals
2826_VT_tedroo
2776_VT_tedroo
951_VT_ghostnet
1509_VT_bredolab
2790_VT_tedroo
2820_VT_tedroo
1207_zbot
2837_VT_tedroo
1755_VT_downadup_conficker
1484_VT_nuwar_peacomm
835_mebroot
2269_VT_koobface
1662_VT_bredolab_II
1650_VT_bredolab
1551_VT_bredolab
1680_VT_bredolab
1545_VT_bredolab
1540_VT_bredolab
1543_VT_bredolab
1641_VT_bredolab_II
1631_VT_bredolab
1592_VT_bredolab
1579_VT_bredolab
1506_VT_bredolab
1532_VT_bredolab
1517_VT_bredolab
1526_VT_bredolab
1507_VT_bredolab
1858_VT_downadup_conficker_II
2832_VT_tedroo
1852_VT_downadup_conficker_II
1885_VT_downadup_conficker
599_01.Test VT Sinowals
1746_VT_downadup_conficker
1880_VT_downadup_conficker
1875_VT_downadup_conficker_II
1840_VT_downadup_conficker
1873_VT_downadup_conficker_II
1767_VT_downadup_conficker
1785_VT_downadup_conficker
1853_VT_downadup_conficker_II
1750_VT_downadup_conficker
1561_VT_bredolab
1535_VT_bredolab
1515_VT_bredolab
1499_VT_bredolab
1832_VT_downadup_conficker
703_Poisonlvy
1590_VT_bredolab_II
1331_VT_waledac
1611_VT_bredolab
1366_VT_waledac
1306_VT_waledac
1310_VT_waledac
1305_VT_waledac
2768_VT_tedroo
1404_VT_waledac
1352_VT_waledac
1384_VT_waledac
1830_VT_downadup_conficker
735_Poisonlvy
1696_VT_bobax_kraken
1007_VT_nuwar_peacomm
2454_VT_koobface
2584_VT_koobface_II
1793_VT_downadup_conficker
3035_VT_ms_rustock
563_01.Test VT Sinowals
929_VT_ghostnet
765_Poisonlvy
2542_VT_koobface_II
3041_VT_ms_rustock
2446_VT_koobface
2867_VT_risloup
2460_VT_koobface_II
1775_VT_downadup_conficker
2889_VT_harvester
633_Poisonlvy
1847_VT_downadup_conficker_II
641_Poisonlvy
1823_VT_downadup_conficker
1025_Rustock Samples
2774_VT_tedroo
2147_VT_bredolab_II
2845_VT_tedroo
618_Poisonlvy



MANDIANT VIRUS TOTAL zynamics www.zynamics.com



No code sharing... at all?

- There were some commonalities
- We found obvious similarities:
  - Malware written in the same language (Delphi)
  - Malware used common libraries (BZip2, OpenSSL, SFX installer code)
  - Same packer

Common functionality

- Does no common code mean no similar functionality?
  - No, identical functionality could be implemented with a different syntax (obfuscated)
  - Let’s look at one case across many families: Code Injection

Code Injection

- The general idea:
  - Do a OpenProces() on the target process
  - Allocate memory in the remote process: *VirtualAllocEx()*
  - Write data into the allocated memory: *WriteProcessMemory()*
  - Use *CreateRemoteThread()* to start a new thread executing the injected code
  - Wait until the remote thread terminates: *WaitForSingleObject()*

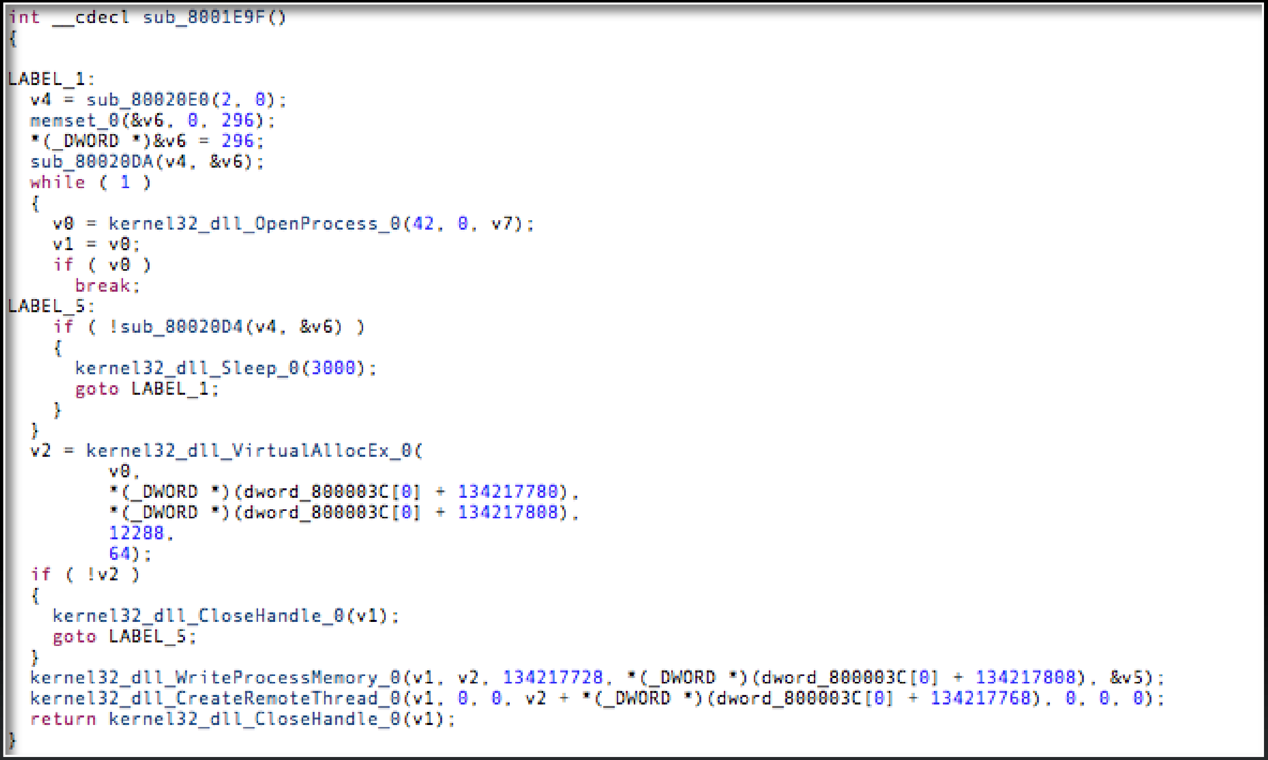
Tracking the functionality

- Fortunately the same tool we used to classify and cluster kept information about all functions in all analyzed executable code (in this case the table had close to one million entries)
- Query all executables making use of the Windows APIs:
  - *CreateRemoteThread() VirtualAllocEx() WriteProcessMemory() ZwOpenProces()*

Inject-capable Malware

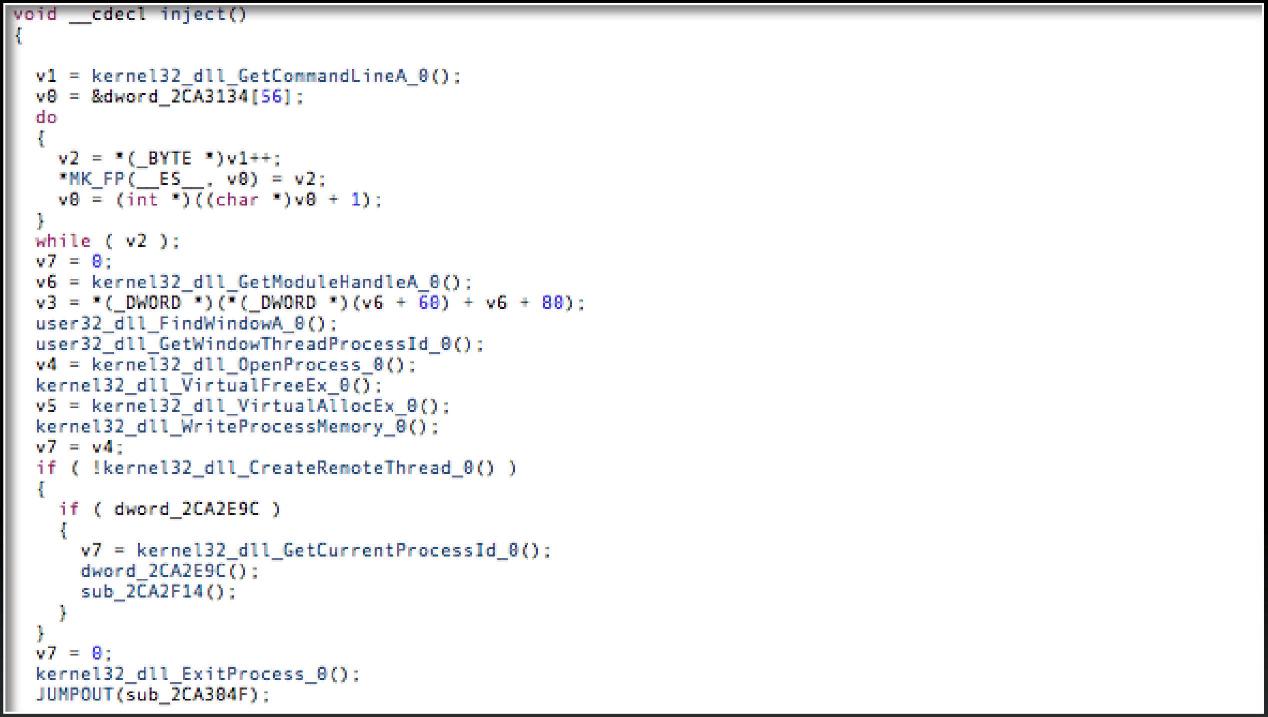
- Samples from these families were found to use those common code-injection APIs:
  - Zbot
  - Cutwail
  - Kraken/Bobax
  - Srizbi
  - Bredolab
  - Conficker
  - Targeted Malware (A LOT)

Cutwail



```
int __cdecl sub_8001E9F()
{

LABEL_1:
  v4 = sub_80020E0(2, 0);
  memset_0(&v6, 0, 296);
  *(_DWORD *)&v6 = 296;
  sub_80020DA(v4, &v6);
  while ( 1 )
  {
    v0 = kernel32_dll_OpenProcess_0(42, 0, v7);
    v1 = v0;
    if ( v0 )
      break;
LABEL_5:
    if ( !sub_80020D4(v4, &v6) )
    {
      kernel32_dll_Sleep_0(3000);
      goto LABEL_1;
    }
  }
  v2 = kernel32_dll_VirtualAllocEx_0(
         v0,
         *(_DWORD *)(dword_800003C[0] + 134217780),
         *(_DWORD *)(dword_800003C[0] + 134217800),
         12288,
         64);
  if ( !v2 )
  {
    kernel32_dll_CloseHandle_0(v1);
    goto LABEL_5;
  }
  kernel32_dll_WriteProcessMemory_0(v1, v2, 134217728, *(_DWORD *)(dword_800003C[0] + 134217800), &v5);
  kernel32_dll_CreateRemoteThread_0(v1, 0, 0, v2 + *(_DWORD *)(dword_800003C[0] + 134217768), 0, 0, 0);
  return kernel32_dll_CloseHandle_0(v1);
}
```

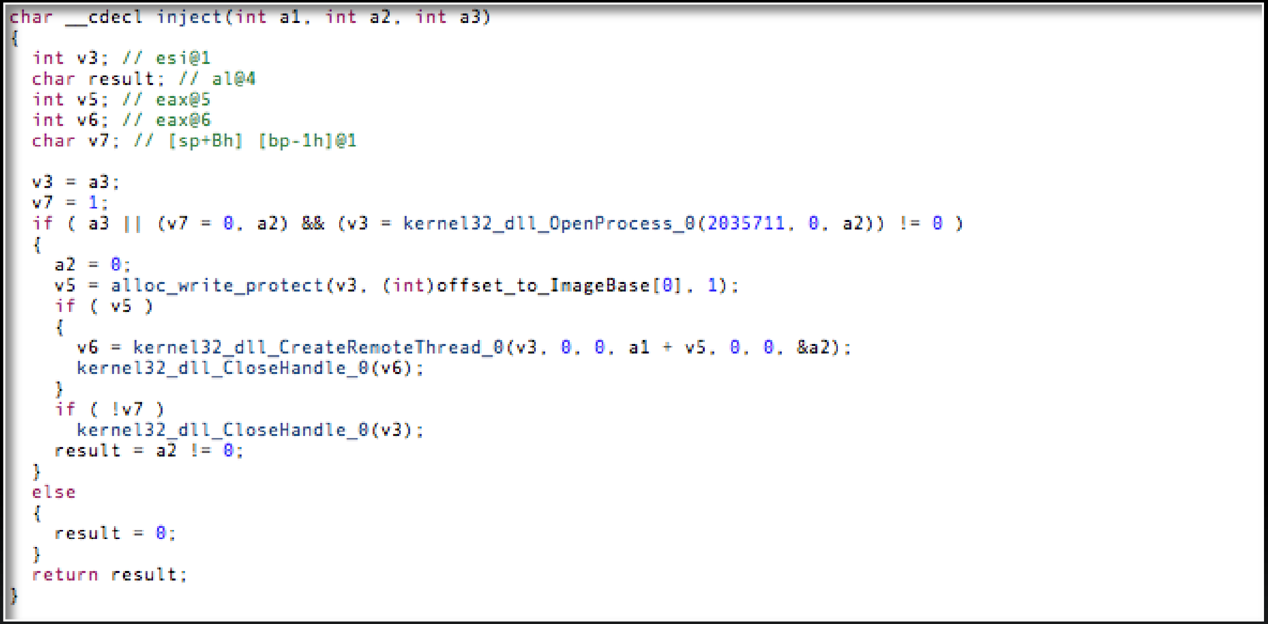
Kraken/Bobax



```
void __cdecl inject()
{

  v1 = kernel32_dll_GetCommandLineA_0();
  v0 = &dword_2CA3134[56];
  do
  {
    v2 = *(_BYTE *)v1++;
    *MK_FP(__ES__, v0) = v2;
    v0 = (int *)((char *)v0 + 1);
  }
  while ( v2 );
  v7 = 0;
  v6 = kernel32_dll_GetModuleHandleA_0();
  v3 = *(_DWORD *)(*(_DWORD *)(v6 + 60) + v6 + 80);
  user32_dll_FindWindowA_0();
  user32_dll_GetWindowThreadProcessId_0();
  v4 = kernel32_dll_OpenProcess_0();
  kernel32_dll_VirtualFreeEx_0();
  v5 = kernel32_dll_VirtualAllocEx_0();
  kernel32_dll_WriteProcessMemory_0();
  v7 = v4;
  if ( !kernel32_dll_CreateRemoteThread_0() )
  {
    if ( dword_2CA2E9C )
    {
      v7 = kernel32_dll_GetCurrentProcessId_0();
      dword_2CA2E9C();
      sub_2CA2F14();
    }
  }
  v7 = 0;
  kernel32_dll_ExitProcess_0();
  JUMPOUT(sub_2CA384F);
```

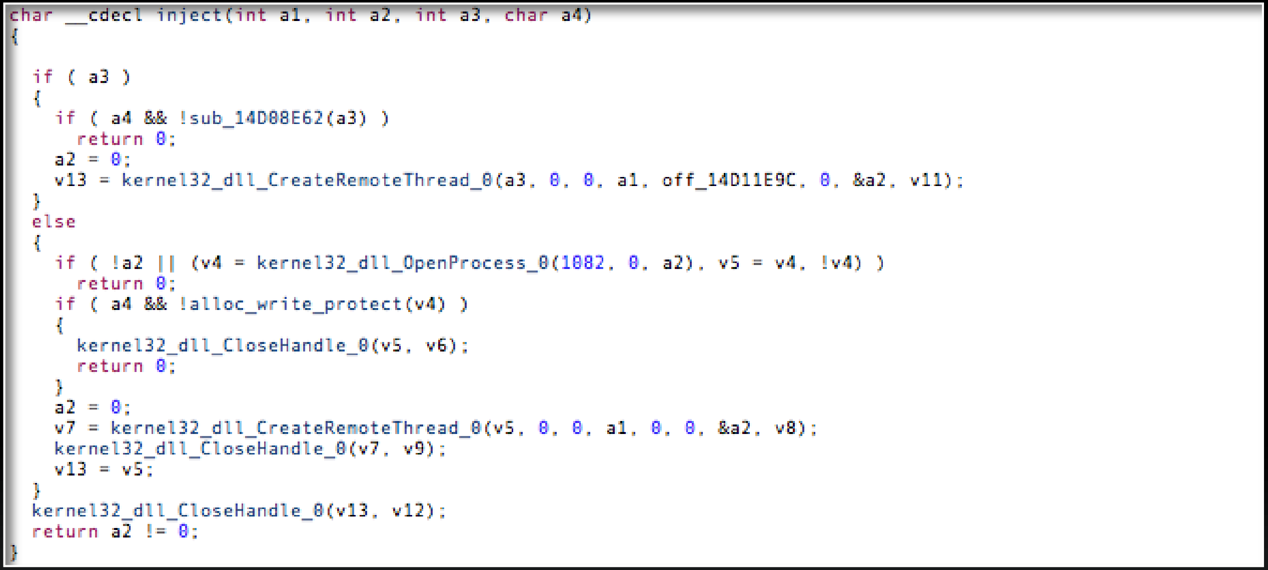
MANDIANT® VIRUS TOTAL zynamics www.zynamics.com

Zbot



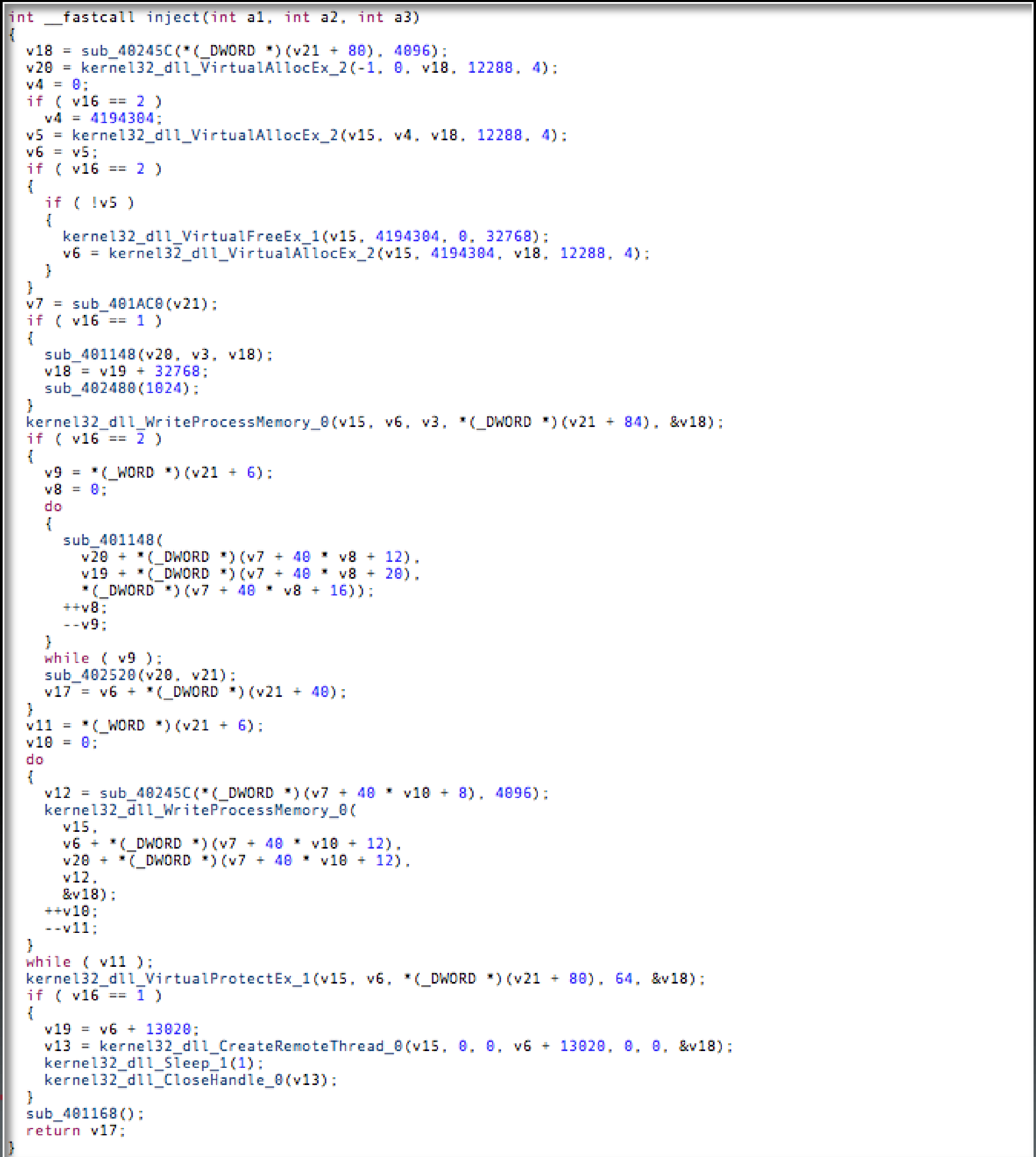
```
char __cdecl inject(int a1, int a2, int a3)
{
  int v3; // esi@1
  char result; // al@4
  int v5; // eax@5
  int v6; // eax@6
  char v7; // [sp+Bh] [bp-1h]@1

  v3 = a3;
  v7 = 1;
  if ( a3 || (v7 = 0, a2) && (v3 = kernel32_dll_OpenProcess_0(2035711, 0, a2)) != 0 )
  {
    a2 = 0;
    v5 = alloc_write_protect(v3, (int)offset_to_ImageBase[0], 1);
    if ( v5 )
    {
      v6 = kernel32_dll_CreateRemoteThread_0(v3, 0, 0, a1 + v5, 0, 0, &a2);
      kernel32_dll_CloseHandle_0(v6);
    }
    if ( !v7 )
      kernel32_dll_CloseHandle_0(v3);
    result = a2 != 0;
  }
  else
  {
    result = 0;
  }
  return result;
}
```

Zbot (2)



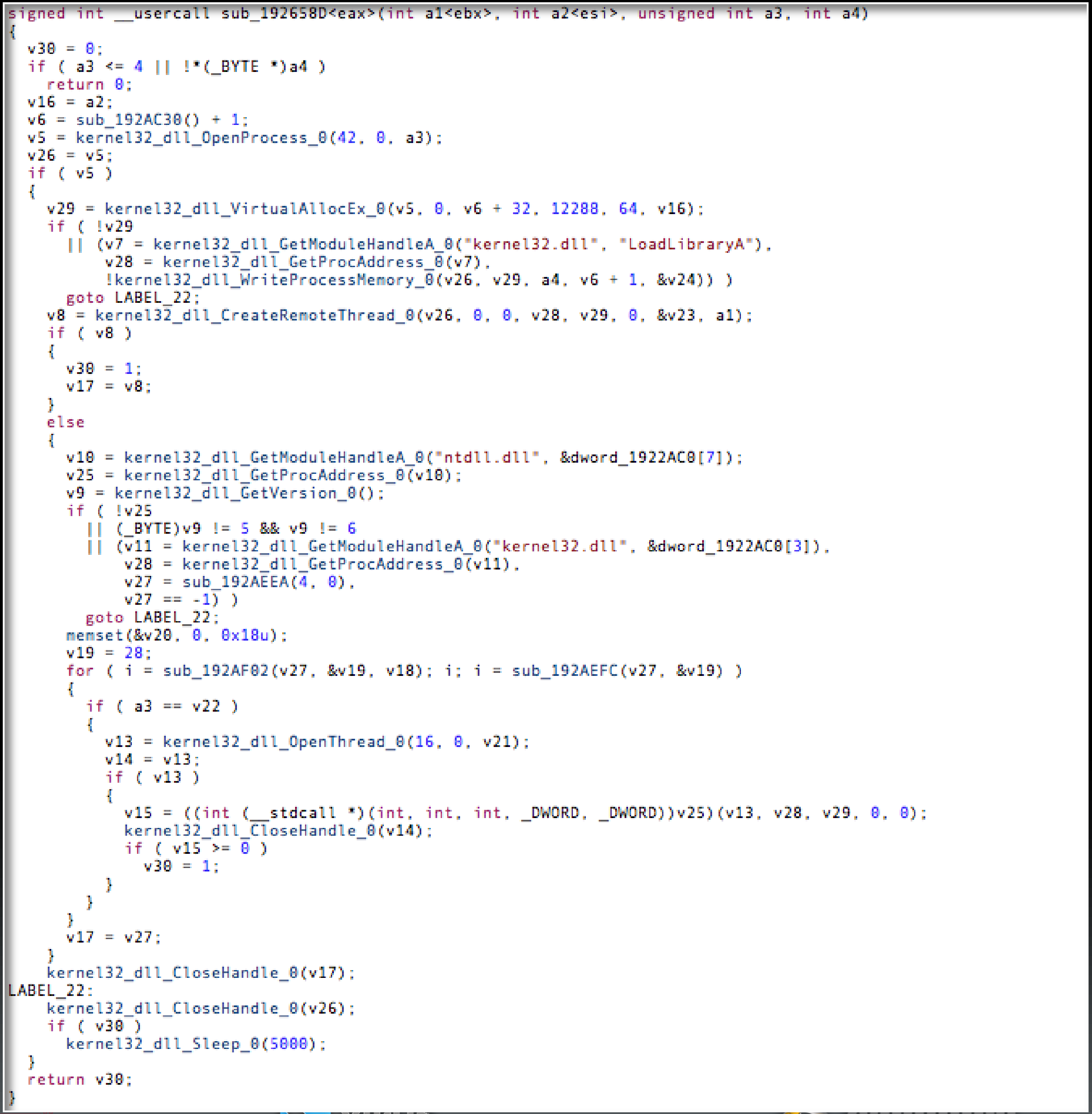
```
char __cdecl inject(int a1, int a2, int a3, char a4)
{

  if ( a3 )
  {
    if ( a4 && !sub_14D88E62(a3) )
      return 0;
    a2 = 0;
    v13 = kernel32_dll_CreateRemoteThread_0(a3, 0, 0, a1, off_14D11E9C, 0, &a2, v11);
  }
  else
  {
    if ( !a2 || (v4 = kernel32_dll_OpenProcess_0(1082, 0, a2), v5 = v4, !v4) )
      return 0;
    if ( a4 && !alloc_write_protect(v4) )
    {
      kernel32_dll_CloseHandle_0(v5, v6);
      return 0;
    }
    a2 = 0;
    v7 = kernel32_dll_CreateRemoteThread_0(v5, 0, 0, a1, 0, 0, &a2, v8);
    kernel32_dll_CloseHandle_0(v7, v9);
    v13 = v5;
  }
  kernel32_dll_CloseHandle_0(v13, v12);
  return a2 != 0;
}
```

Bredolab



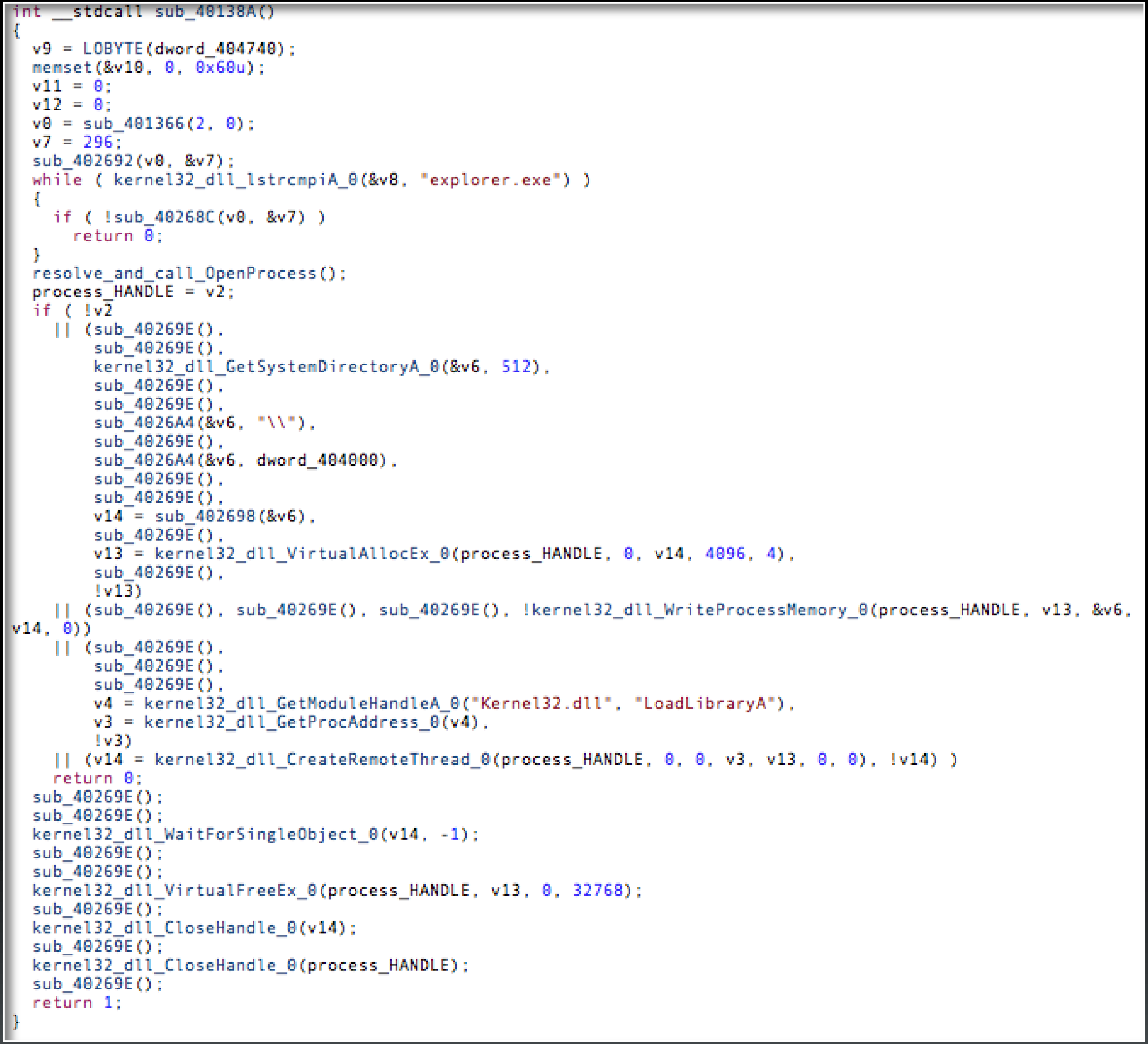
```
int __fastcall inject(int a1, int a2, int a3)
{
  v18 = sub_40245C(*(_DWORD *)(v21 + 80), 4096);
  v20 = kernel32_dll_VirtualAllocEx_2(-1, 0, v18, 12288, 4);
  v4 = 0;
  if ( v16 == 2 )
    v4 = 4194304;
  v5 = kernel32_dll_VirtualAllocEx_2(v15, v4, v18, 12288, 4);
  v6 = v5;
  if ( v16 == 2 )
  {
    if ( !v5 )
    {
      kernel32_dll_VirtualFreeEx_1(v15, 4194304, 0, 32768);
      v6 = kernel32_dll_VirtualAllocEx_2(v15, 4194304, v18, 12288, 4);
    }
  }
  v7 = sub_401AC0(v21);
  if ( v16 == 1 )
  {
    sub_401148(v20, v3, v18);
    v18 = v19 + 32768;
    sub_402480(1024);
  }
  kernel32_dll_WriteProcessMemory_0(v15, v6, v3, *(_DWORD *)(v21 + 84), &v18);
  if ( v16 == 2 )
  {
    v9 = *(_WORD *)(v21 + 6);
    v8 = 0;
    do
    {
      sub_401148(
        v20 + *(_DWORD *)(v7 + 40 * v8 + 12),
        v19 + *(_DWORD *)(v7 + 40 * v8 + 20),
        *(_DWORD *)(v7 + 40 * v8 + 16));
      ++v8;
      --v9;
    }
    while ( v9 );
    sub_402520(v20, v21);
    v17 = v6 + *(_DWORD *)(v21 + 40);
  }
  v11 = *(_WORD *)(v21 + 6);
  v10 = 0;
  do
  {
    v12 = sub_40245C(*(_DWORD *)(v7 + 40 * v10 + 8), 4096);
    kernel32_dll_WriteProcessMemory_0(
      v15,
      v6 + *(_DWORD *)(v7 + 40 * v10 + 12),
      v20 + *(_DWORD *)(v7 + 40 * v10 + 12),
      v12,
      &v18);
    ++v10;
    --v11;
  }
  while ( v11 );
  kernel32_dll_VirtualProtectEx_1(v15, v6, *(_DWORD *)(v21 + 80), 64, &v18);
  if ( v16 == 1 )
  {
    v19 = v6 + 13020;
    v13 = kernel32_dll_CreateRemoteThread_0(v15, 0, 0, v6 + 13020, 0, 0, &v18);
    kernel32_dll_Sleep_1(1);
    kernel32_dll_CloseHandle_0(v13);
  }
  sub_401168();
  return v17;
}
```

Conficker



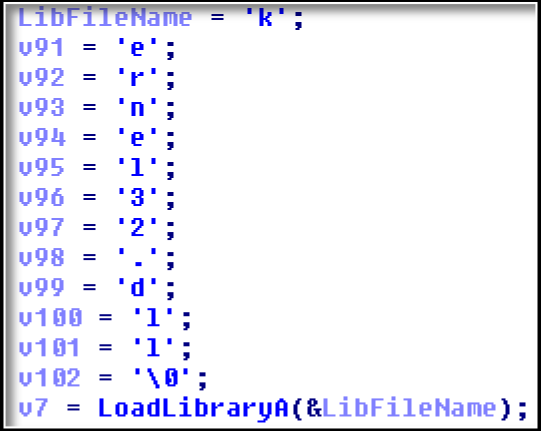
```
signed int __usercall sub_192658D<eax>(int a1<ebx>, int a2<esi>, unsigned int a3, int a4)
{
  v30 = 0;
  if ( a3 <= 4 || !*(_BYTE *)a4 )
    return 0;
  v16 = a2;
  v6 = sub_192AC30() + 1;
  v5 = kernel32_dll_OpenProcess_0(42, 0, a3);
  v26 = v5;
  if ( v5 )
  {
    v29 = kernel32_dll_VirtualAllocEx_0(v5, 0, v6 + 32, 12288, 64, v16);
    if ( !v29
      || (v7 = kernel32_dll_GetModuleHandleA_0("kernel32.dll", "LoadLibraryA"),
          v28 = kernel32_dll_GetProcAddress_0(v7),
          !kernel32_dll_WriteProcessMemory_0(v26, v29, a4, v6 + 1, &v24)) )
      goto LABEL_22;
    v8 = kernel32_dll_CreateRemoteThread_0(v26, 0, 0, v28, v29, 0, &v23, a1);
    if ( v8 )
    {
      v30 = 1;
      v17 = v8;
    }
    else
    {
      v10 = kernel32_dll_GetModuleHandleA_0("ntdll.dll", &dword_1922AC0[7]);
      v25 = kernel32_dll_GetProcAddress_0(v10);
      v9 = kernel32_dll_GetVersion_0();
      if ( !v25
        || (_BYTE)v9 != 5 && v9 != 6
        || (v11 = kernel32_dll_GetModuleHandleA_0("kernel32.dll", &dword_1922AC0[3]),
            v28 = kernel32_dll_GetProcAddress_0(v11),
            v27 = sub_192AEEA(4, 0),
            v27 == -1) )
        goto LABEL_22;
      memset(&v20, 0, 0x18u);
      v19 = 28;
      for ( i = sub_192AF02(v27, &v19, v18); i; i = sub_192AEFC(v27, &v19) )
      {
        if ( a3 == v22 )
        {
          v13 = kernel32_dll_OpenThread_0(16, 0, v21);
          v14 = v13;
          if ( v13 )
          {
            v15 = ((int (__stdcall *)(int, int, int, _DWORD, _DWORD))v25)(v13, v28, v29, 0, 0);
            kernel32_dll_CloseHandle_0(v14);
            if ( v15 >= 0 )
              v30 = 1;
          }
        }
      }
      v17 = v27;
    }
    kernel32_dll_CloseHandle_0(v17);
LABEL_22:
    kernel32_dll_CloseHandle_0(v26);
    if ( v30 )
      kernel32_dll_Sleep_0(5000);
  }
  return v30;
}
```

Srizbi

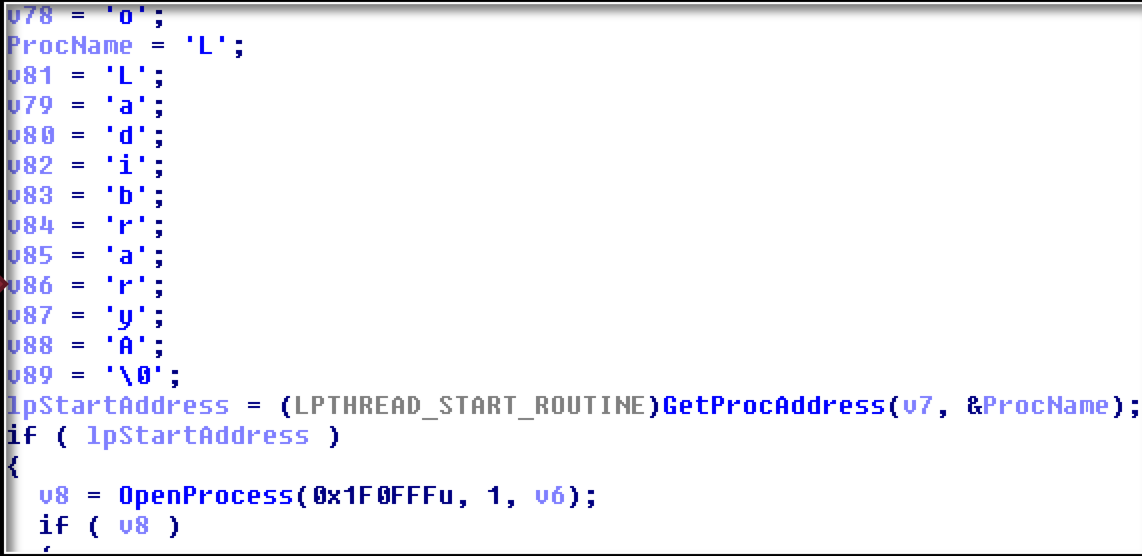


```
int __stdcall sub_40138A()
{
  v9 = LOBYTE(dword_404740);
  memset(&v10, 0, 0x60u);
  v11 = 0;
  v12 = 0;
  v0 = sub_401366(2, 0);
  v7 = 296;
  sub_402692(v0, &v7);
  while ( kernel32_dll_lstrcmpiA_0(&v8, "explorer.exe") )
  {
    if ( !sub_40268C(v0, &v7) )
      return 0;
  }
  resolve_and_call_OpenProcess();
  process_HANDLE = v2;
  if ( !v2
    || (sub_40269E(),
        sub_40269E(),
        kernel32_dll_GetSystemDirectoryA_0(&v6, 512),
        sub_40269E(),
        sub_40269E(),
        sub_4026A4(&v6, "\\"),
        sub_40269E(),
        sub_4026A4(&v6, dword_404000),
        sub_40269E(),
        sub_40269E(),
        v14 = sub_402698(&v6),
        sub_40269E(),
        v13 = kernel32_dll_VirtualAllocEx_0(process_HANDLE, 0, v14, 4096, 4),
        sub_40269E(),
        !v13)
    || (sub_40269E(), sub_40269E(), sub_40269E(), !kernel32_dll_WriteProcessMemory_0(process_HANDLE, v13, &v6,
v14, 0))
    || (sub_40269E(),
        sub_40269E(),
        sub_40269E(),
        v4 = kernel32_dll_GetModuleHandleA_0("Kernel32.dll", "LoadLibraryA"),
        v3 = kernel32_dll_GetProcAddress_0(v4),
        !v3)
    || (v14 = kernel32_dll_CreateRemoteThread_0(process_HANDLE, 0, 0, v3, v13, 0, 0), !v14) )
    return 0;
  sub_40269E();
  sub_40269E();
  kernel32_dll_WaitForSingleObject_0(v14, -1);
  sub_40269E();
  sub_40269E();
  kernel32_dll_VirtualFreeEx_0(process_HANDLE, v13, 0, 32768);
  sub_40269E();
  kernel32_dll_CloseHandle_0(v14);
  sub_40269E();
  kernel32_dll_CloseHandle_0(process_HANDLE);
  sub_40269E();
  return 1;
}
```

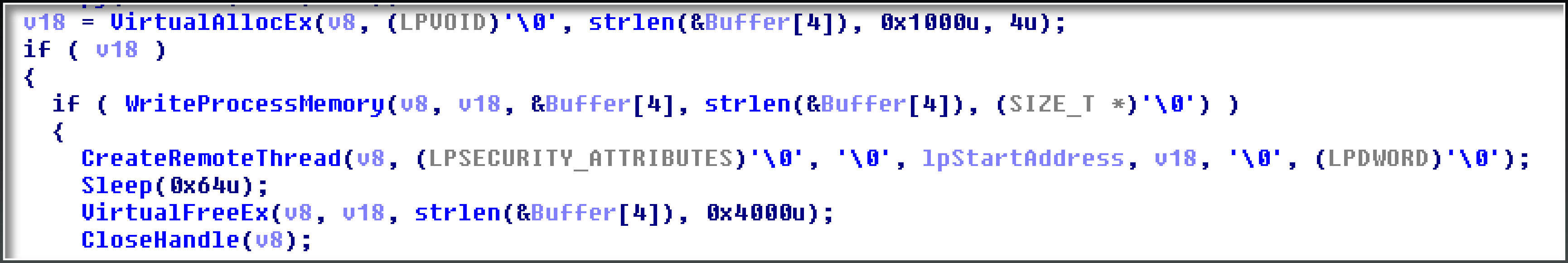
Targeted Malware



```
LibFileName = 'k';
v91 = 'e';
v92 = 'r';
v93 = 'n';
v94 = 'e';
v95 = 'l';
v96 = '3';
v97 = '2';
v98 = '.';
v99 = 'd';
v100 = 'l';
v101 = 'l';
v102 = '\0';
v7 = LoadLibraryA(&LibFileName);
```



```
v78 = 'o';
ProcName = 'L';
v81 = 'L';
v79 = 'a';
v80 = 'd';
v82 = 'i';
v83 = 'b';
v84 = 'r';
v85 = 'a';
v86 = 'r';
v87 = 'y';
v88 = 'A';
v89 = '\0';
lpStartAddress = (LPTHREAD_START_ROUTINE)GetProcAddress(v7, &ProcName);
if ( lpStartAddress )
{
  v8 = OpenProcess(0x1F0FFFu, 1, v6);
  if ( v8 )
```



```
v18 = VirtualAllocEx(v8, (LPVOID)'\0', strlen(&Buffer[4]), 0x1000u, 4u);
if ( v18 )
{
  if ( WriteProcessMemory(v8, v18, &Buffer[4], strlen(&Buffer[4]), (SIZE_T *)'\0') )
  {
    CreateRemoteThread(v8, (LPSECURITY_ATTRIBUTES)'\0', '\0', lpStartAddress, v18, '\0', (LPDWORD)'\0');
    Sleep(0x64u);
    VirtualFreeEx(v8, v18, strlen(&Buffer[4]), 0x4000u);
    CloseHandle(v8);
```

Implementations of Functionality

- As we have seen there are many ways of implementing a nearly identical functionality
- Differences come from:
  - Source-code
  - Compilers
    - This can be overcome

Rootkits

The stuff dreams and nightmares are made of

Rootkits

- Unique results
- *Theory: Rootkits would have high levels of shared code because kernel code is complex and tiresome to re-write.*
- Answer: Sort of

Rootkits

- Compared:
  - targeted malware
  - rootkits from rootkit.com
  - Mass rootkits
- Very little similarity
- This can be explain:
  - Kernel code is hard to re-use a lot of modifications have to occur
  - Rootkit.com projects are dated
  - Copying and pasting code from one project to another is hard to do without modifications

Rootkits

- Targeted Rootkits still accomplish same goals as public ones
  - Modification of the SSDT
  - Hiding system resources
  - Hiding network traffic

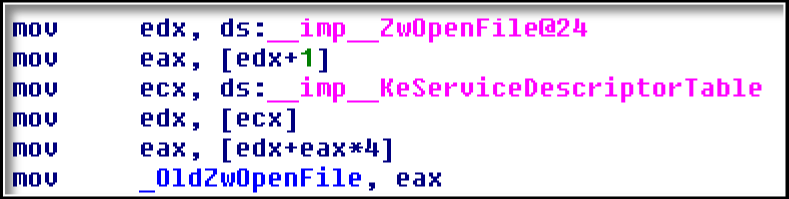
Rootkits

- Case Studies:
  - Similarities between targeted and mass malware
  - “borrowing” of source code
  - Avoiding detection

Rookits: Case Studies

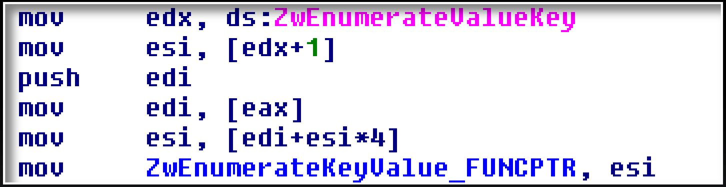
FUNCTION RETRIEVAL

- Circa 2001



```
mov     edx, ds:__imp__ZwOpenFile@24
mov     eax, [edx+1]
mov     ecx, ds:__imp__KeServiceDescriptorTable
mov     edx, [ecx]
mov     eax, [edx+eax*4]
mov     _OldZwOpenFile, eax
```

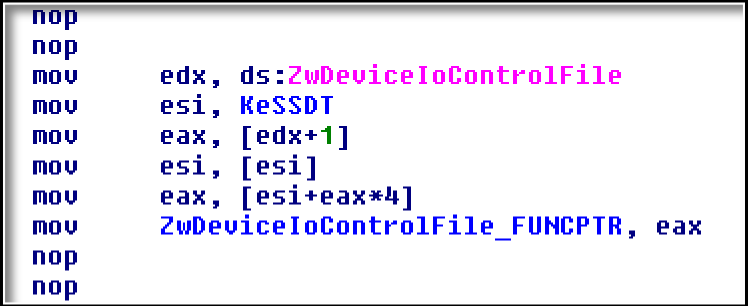
- Circa 2009



```
mov     edx, ds:ZwEnumerateValueKey
mov     esi, [edx+1]
push    edi
mov     edi, [eax]
mov     esi, [edi+esi*4]
mov     ZwEnumerateKeyValue_FUNCPTR, esi
```

FUNCTION RETRIEVAL

- Circa 2010

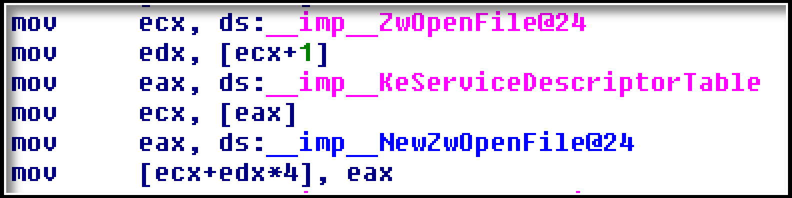


```
nop
nop
mov     edx, ds:ZwDeviceIoControlFile
mov     esi, KeSSDT
mov     eax, [edx+1]
mov     esi, [esi]
mov     eax, [esi+eax*4]
mov     ZwDeviceIoControlFile_FUNCPTR, eax
nop
nop
```

Rookits: Case Studies

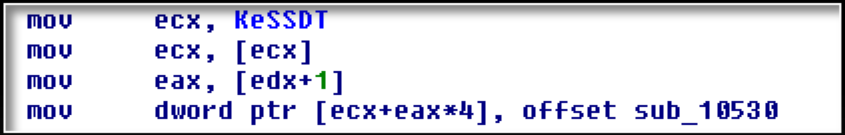
HOOK INSTALLATION

- Circa 2001



```
mov     ecx, ds:__imp__ZwOpenFile@24
mov     edx, [ecx+1]
mov     eax, ds:__imp__KeServiceDescriptorTable
mov     ecx, [eax]
mov     eax, ds:__imp__NewZwOpenFile@24
mov     [ecx+edx*4], eax
```

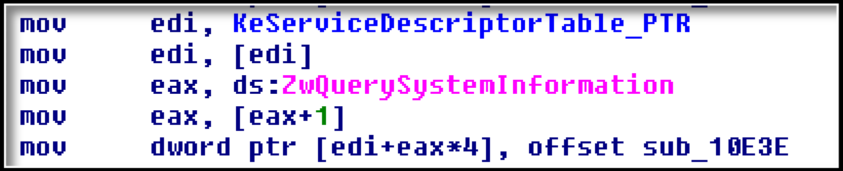
- Circa 2009



```
mov     ecx, KeSSDT
mov     ecx, [ecx]
mov     eax, [edx+1]
mov     dword ptr [ecx+eax*4], offset sub_10530
```

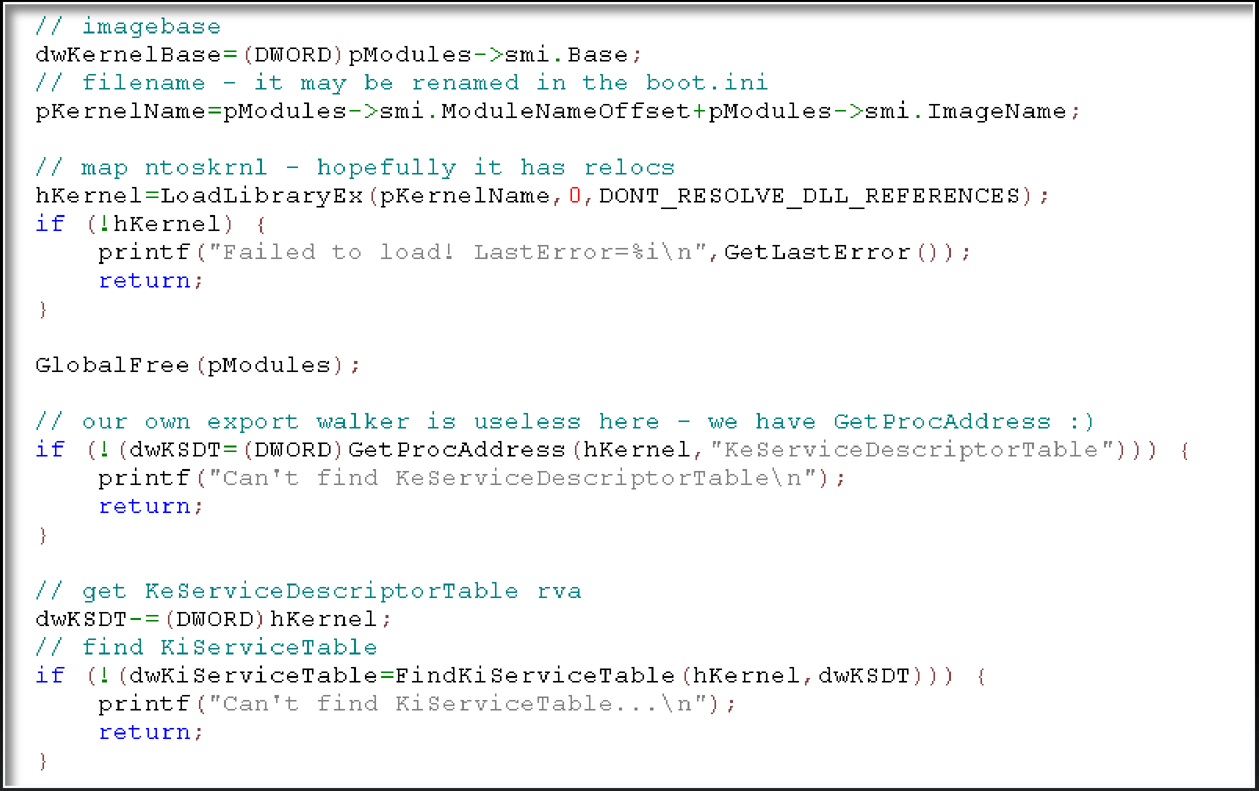
HOOK INSTALLATION

- Circa 2010



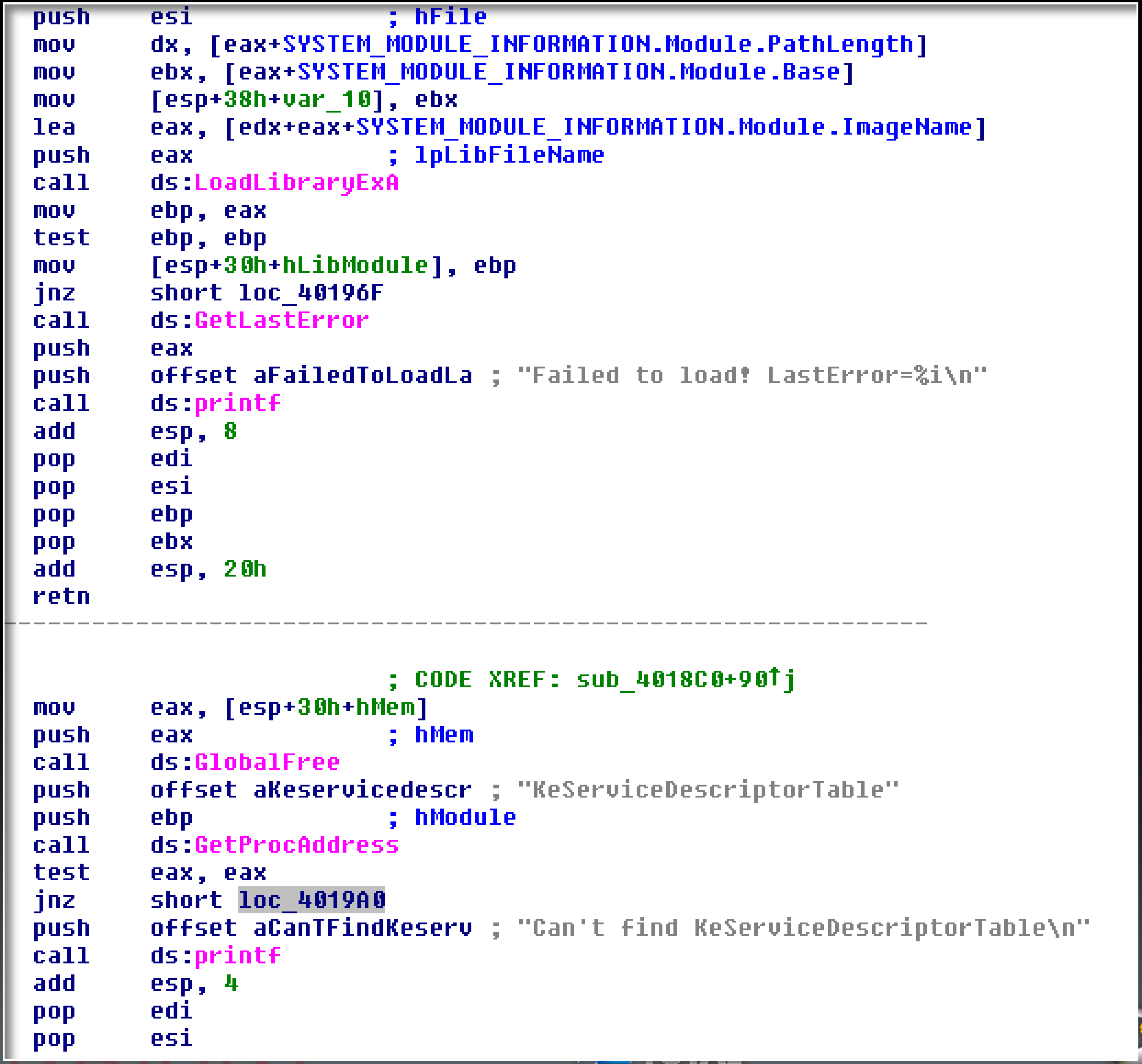
```
mov     edi, KeServiceDescriptorTable_PTR
mov     edi, [edi]
mov     eax, ds:ZwQuerySystemInformation
mov     eax, [eax+1]
mov     dword ptr [edi+eax*4], offset sub_10E3E
```

Rootkit: Case Studies



```
// imagebase
dwKernelBase=(DWORD)pModules->smi.Base;
// filename - it may be renamed in the boot.ini
pKernelName=pModules->smi.ModuleNameOffset+pModules->smi.ImageName;

// map ntoskrnl - hopefully it has relocs
hKernel=LoadLibraryEx(pKernelName,0,DONT_RESOLVE_DLL_REFERENCES);
if (!hKernel) {
    printf("Failed to load! LastError=%i\n",GetLastError());
    return;
}

GlobalFree(pModules);

// our own export walker is useless here - we have GetProcAddress :)
if (!(dwKSDT=(DWORD)GetProcAddress(hKernel,"KeServiceDescriptorTable"))) {
    printf("Can't find KeServiceDescriptorTable\n");
    return;
}

// get KeServiceDescriptorTable rva
dwKSDT-=(DWORD)hKernel;
// find KiServiceTable
if (!(dwKiServiceTable=FindKiServiceTable(hKernel,dwKSDT))) {
    printf("Can't find KiServiceTable...\n");
    return;
}
```

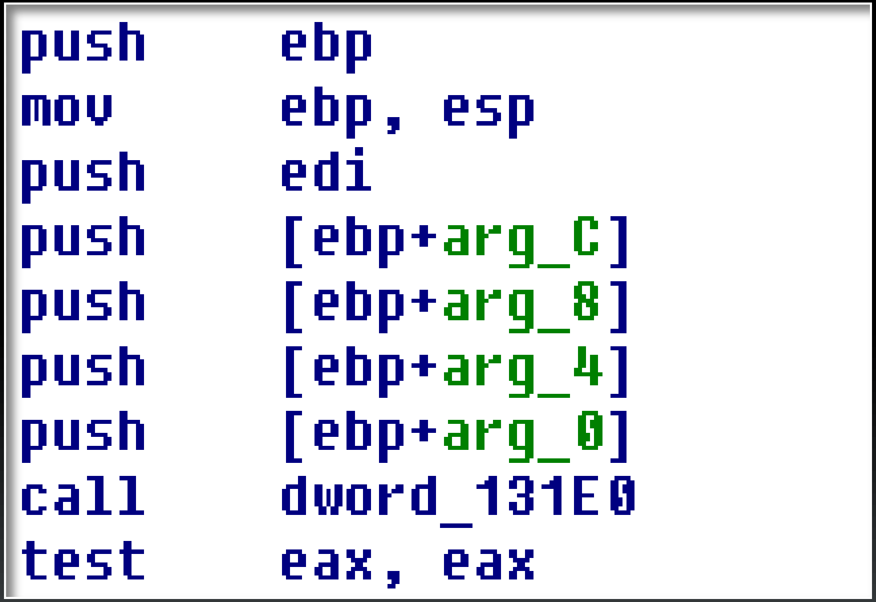
Rootkits: Case Studies



```
push    esi             ; hFile
mov     dx, [eax+SYSTEM_MODULE_INFORMATION.Module.PathLength]
mov     ebx, [eax+SYSTEM_MODULE_INFORMATION.Module.Base]
mov     [esp+38h+var_10], ebx
lea     eax, [edx+eax+SYSTEM_MODULE_INFORMATION.Module.ImageName]
push    eax             ; lpLibFileName
call    ds:LoadLibraryExA
mov     ebp, eax
test    ebp, ebp
mov     [esp+30h+hLibModule], ebp
jnz     short loc_40196F
call    ds:GetLastError
push    eax
push    offset aFailedToLoadLa ; "Failed to load! LastError=%i\n"
call    ds:printf
add     esp, 8
pop     edi
pop     esi
pop     ebp
pop     ebx
add     esp, 20h
retn
---------------------------------------------------------------

                        ; CODE XREF: sub_4018C0+90↑j
mov     eax, [esp+30h+hMem]
push    eax             ; hMem
call    ds:GlobalFree
push    offset aKeservicedescr ; "KeServiceDescriptorTable"
push    ebp             ; hModule
call    ds:GetProcAddress
test    eax, eax
jnz     short loc_4019A0
push    offset aCanTFindKeserv ; "Can't find KeServiceDescriptorTable\n"
call    ds:printf
add     esp, 4
pop     edi
pop     esi
```

Rootkits: Case Study

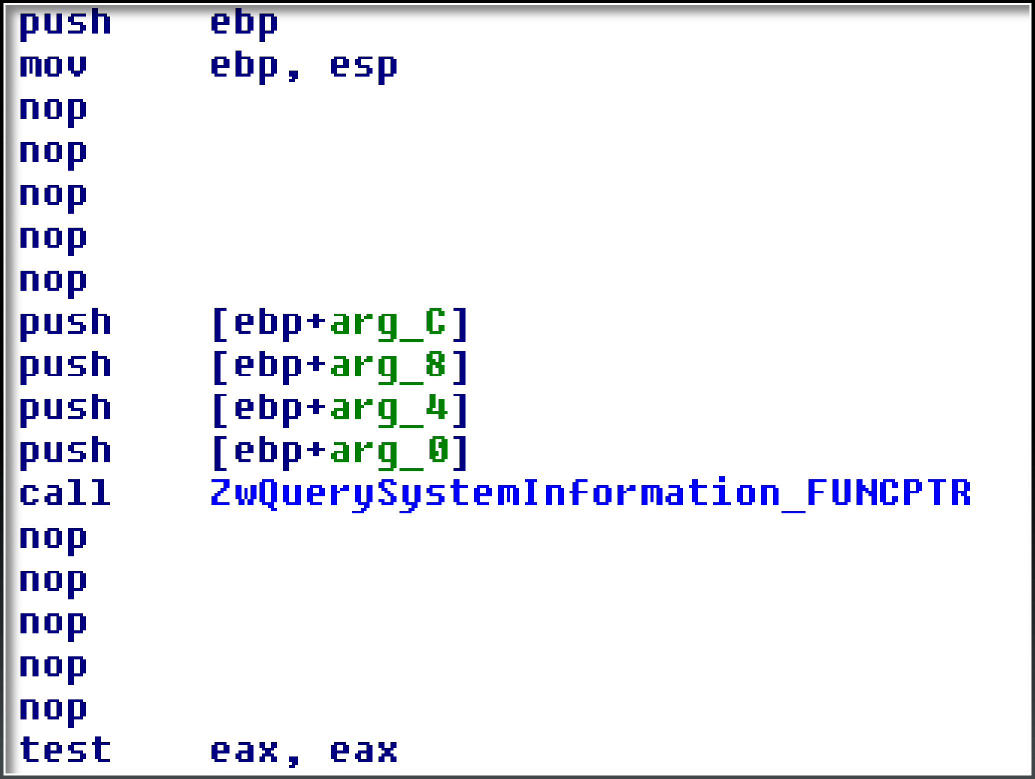
- Variant A:
  - ZwQuerySystemInformation hook handler



```
push    ebp
mov     ebp, esp
push    edi
push    [ebp+arg_C]
push    [ebp+arg_8]
push    [ebp+arg_4]
push    [ebp+arg_0]
call    dword_131E0
test    eax, eax
```

Rootkit: Case Study

- Variant B:
  - ZwQuerySystemInformation hook handler



```
push    ebp
mov     ebp, esp
nop
nop
nop
nop
nop
push    [ebp+arg_C]
push    [ebp+arg_8]
push    [ebp+arg_4]
push    [ebp+arg_0]
call    ZwQuerySystemInformation_FUNCPTR
nop
nop
nop
nop
nop
test    eax, eax
```

Targeted Malware

When I say A-P-T you say … HO!

Targeted Malware

- Targeted malware is manually classified by analysts
  - When more than a few samples have the same characteristics they get put in a family
- MANDIANT tracks over 20 families
- *The family names for the white paper and presentation have been obfuscated*

Targeted Malware

- Tracking families is very important for Incident Response
- Each family has different capabilities, and levels of sophistications
  - Remediation effort
  - IP loss
  - Exfiltration methods

Targeted Malware

- **Theory:** Samples will not belong to more than one family. Samples will not match mass malware families?

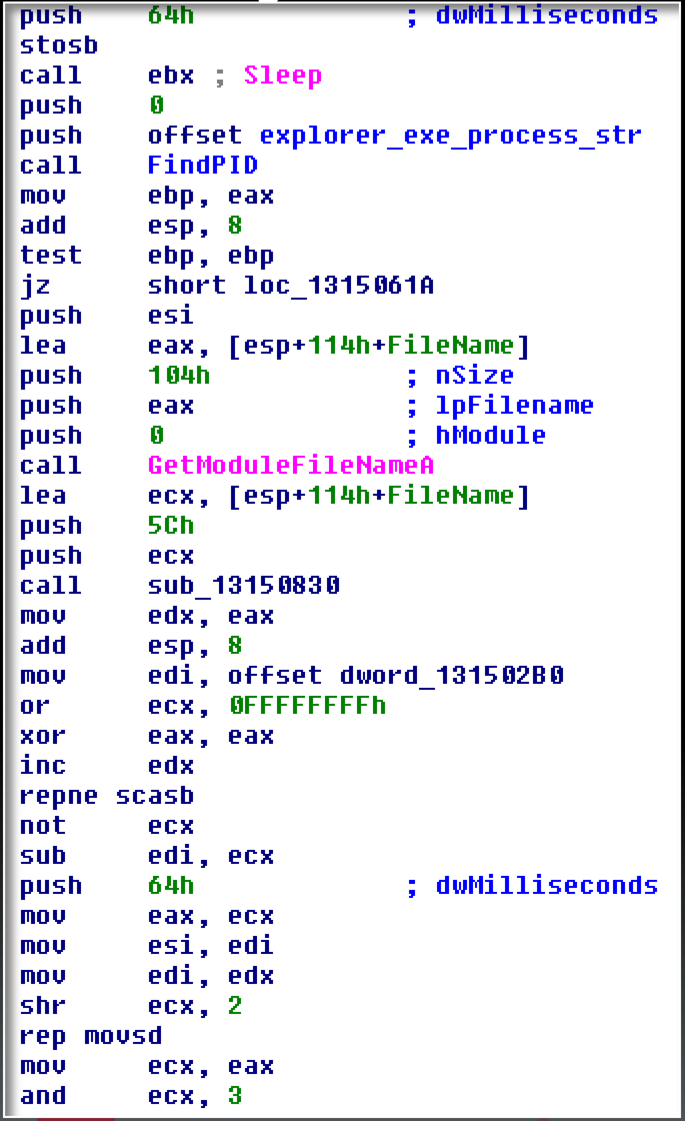
Targeted Malware

- Results:
  - No samples shared enough traits to be considered a member of two families
  - No samples shared enough traits to be considered part of a mass malware families
  - Samples shared feature implementations across families

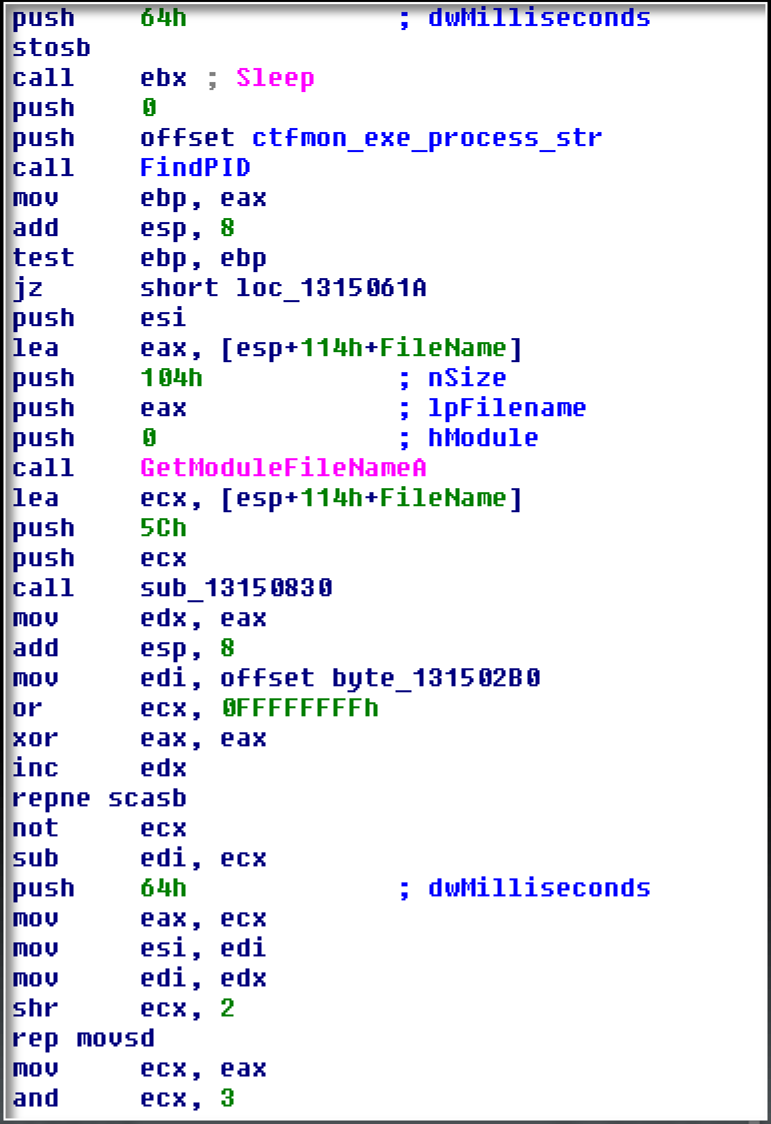
Targeted Malware

- Feature Implementations:
  - Two families (DDD, MMM) had samples with *very* similar implementations of backdoor droppers.
  - Two families (FFF, AAA) had samples with the similar implementations for:
    - Installing/Executing services
    - Removing service
    - *These were all exported functions*
  - It is our belief that:
    - DDD, MMM written by one group
    - FFF, AAA written by one group
  - That’s four families with two different authors

Family: DDD, MMM



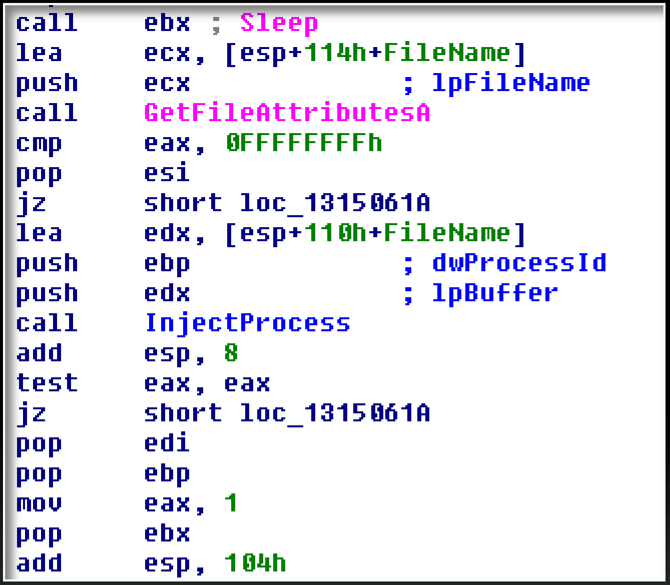
```
push    64h                 ; dwMilliseconds
stosb
call    ebx ; Sleep
push    0
push    offset explorer_exe_process_str
call    FindPID
mov     ebp, eax
add     esp, 8
test    ebp, ebp
jz      short loc_1315061A
push    esi
lea     eax, [esp+114h+FileName]
push    104h                ; nSize
push    eax                 ; lpFilename
push    0                   ; hModule
call    GetModuleFileNameA
lea     ecx, [esp+114h+FileName]
push    5Ch
push    ecx
call    sub_13150830
mov     edx, eax
add     esp, 8
mov     edi, offset dword_131502B0
or      ecx, 0FFFFFFFFh
xor     eax, eax
inc     edx
repne scasb
not     ecx
sub     edi, ecx
push    64h                 ; dwMilliseconds
mov     eax, ecx
mov     esi, edi
mov     edi, edx
shr     ecx, 2
rep movsd
mov     ecx, eax
and     ecx, 3
```



```
push    64h                 ; dwMilliseconds
stosb
call    ebx ; Sleep
push    0
push    offset ctfmon_exe_process_str
call    FindPID
mov     ebp, eax
add     esp, 8
test    ebp, ebp
jz      short loc_1315061A
push    esi
lea     eax, [esp+114h+FileName]
push    104h                ; nSize
push    eax                 ; lpFilename
push    0                   ; hModule
call    GetModuleFileNameA
lea     ecx, [esp+114h+FileName]
push    5Ch
push    ecx
call    sub_13150830
mov     edx, eax
add     esp, 8
mov     edi, offset byte_131502B0
or      ecx, 0FFFFFFFFh
xor     eax, eax
inc     edx
repne scasb
not     ecx
sub     edi, ecx
push    64h                 ; dwMilliseconds
mov     eax, ecx
mov     esi, edi
mov     edi, edx
shr     ecx, 2
rep movsd
mov     ecx, eax
and     ecx, 3
```

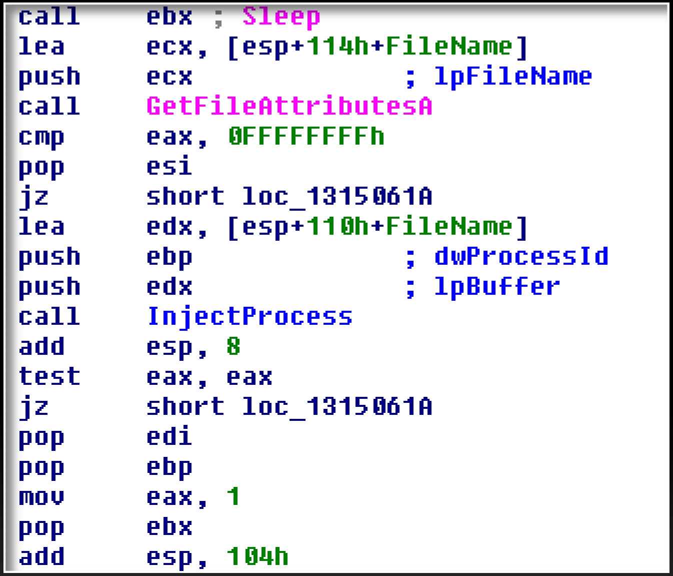
Family: DDD, MMM

INJECTION CALL



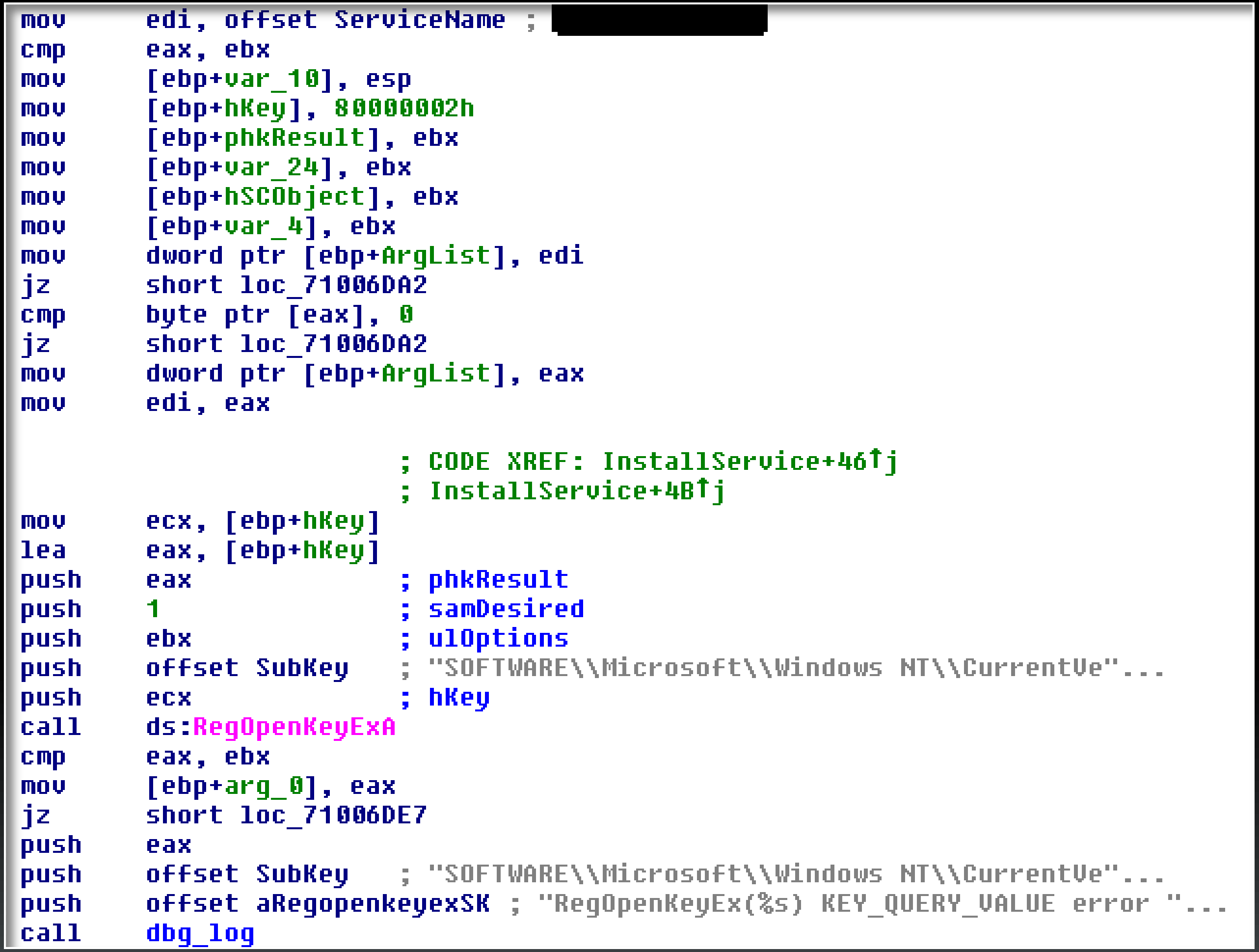
```
call    ebx ; Sleep
lea     ecx, [esp+114h+FileName]
push    ecx             ; lpFileName
call    GetFileAttributesA
cmp     eax, 0FFFFFFFFh
pop     esi
jz      short loc_1315061A
lea     edx, [esp+110h+FileName]
push    ebp             ; dwProcessId
push    edx             ; lpBuffer
call    InjectProcess
add     esp, 8
test    eax, eax
jz      short loc_1315061A
pop     edi
pop     ebp
mov     eax, 1
pop     ebx
add     esp, 104h
```

INJECTION CALL



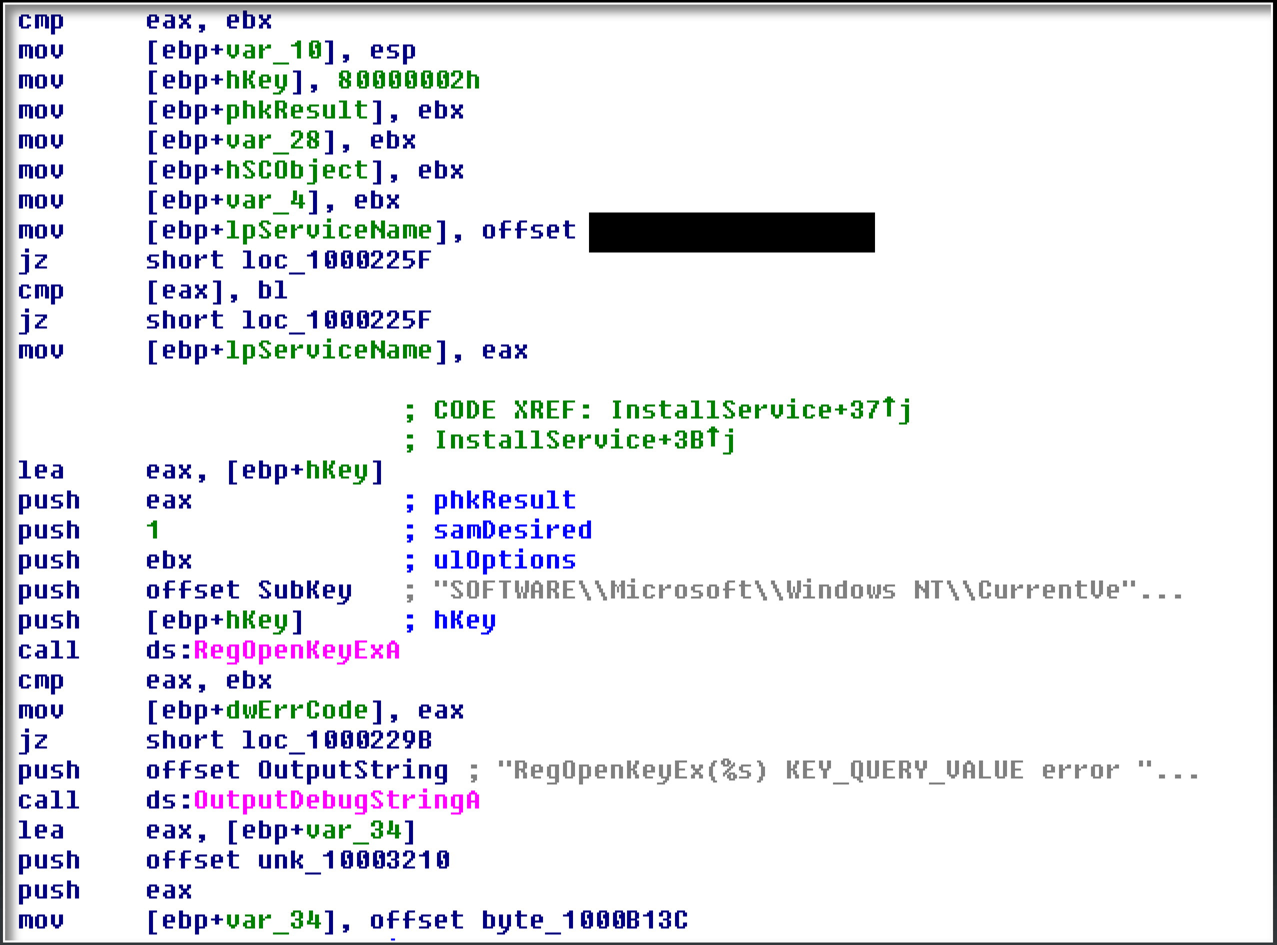
```
call    ebx ; Sleep
lea     ecx, [esp+114h+FileName]
push    ecx             ; lpFileName
call    GetFileAttributesA
cmp     eax, 0FFFFFFFFh
pop     esi
jz      short loc_1315061A
lea     edx, [esp+110h+FileName]
push    ebp             ; dwProcessId
push    edx             ; lpBuffer
call    InjectProcess
add     esp, 8
test    eax, eax
jz      short loc_1315061A
pop     edi
pop     ebp
mov     eax, 1
pop     ebx
add     esp, 104h
```

Family: AAA



```
mov     edi, offset ServiceName ; [redacted]
cmp     eax, ebx
mov     [ebp+var_10], esp
mov     [ebp+hKey], 80000002h
mov     [ebp+phkResult], ebx
mov     [ebp+var_24], ebx
mov     [ebp+hSCObject], ebx
mov     [ebp+var_4], ebx
mov     dword ptr [ebp+ArgList], edi
jz      short loc_71006DA2
cmp     byte ptr [eax], 0
jz      short loc_71006DA2
mov     dword ptr [ebp+ArgList], eax
mov     edi, eax

                        ; CODE XREF: InstallService+46↑j
                        ; InstallService+4B↑j
mov     ecx, [ebp+hKey]
lea     eax, [ebp+hKey]
push    eax             ; phkResult
push    1               ; samDesired
push    ebx             ; ulOptions
push    offset SubKey   ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
push    ecx             ; hKey
call    ds:RegOpenKeyExA
cmp     eax, ebx
mov     [ebp+arg_0], eax
jz      short loc_71006DE7
push    eax
push    offset SubKey   ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
push    offset aRegopenkeyexSK ; "RegOpenKeyEx(%s) KEY_QUERY_VALUE error "...
call    dbg_log
```

Family: FFF



```
cmp     eax, ebx
mov     [ebp+var_10], esp
mov     [ebp+hKey], 80000002h
mov     [ebp+phkResult], ebx
mov     [ebp+var_28], ebx
mov     [ebp+hSCObject], ebx
mov     [ebp+var_4], ebx
mov     [ebp+lpServiceName], offset [redacted]
jz      short loc_1000225F
cmp     [eax], bl
jz      short loc_1000225F
mov     [ebp+lpServiceName], eax

                        ; CODE XREF: InstallService+37↑j
                        ; InstallService+3B↑j
lea     eax, [ebp+hKey]
push    eax             ; phkResult
push    1               ; samDesired
push    ebx             ; ulOptions
push    offset SubKey   ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
push    [ebp+hKey]      ; hKey
call    ds:RegOpenKeyExA
cmp     eax, ebx
mov     [ebp+dwErrCode], eax
jz      short loc_1000229B
push    offset OutputString ; "RegOpenKeyEx(%s) KEY_QUERY_VALUE error "...
call    ds:OutputDebugStringA
lea     eax, [ebp+var_34]
push    offset unk_10003210
push    eax
mov     [ebp+var_34], offset byte_1000B13C
```

Targeted Malware

- Results were verified by other researchers examining network traffic
  - Network traffic linked up multiple families to single groups of author(s)
    - Confirmed our beliefs

Future Research

- Matching feature implementations
- Comparing exploit kits
- More analysis to prove relationships in binary that we are already aware of
- Scaling and fine tuning algorithms
  - malware-universe graph

Conclusion

- No unknown ties between mass malware families and targeted malware
- No large code reuse between the families analyzed
  - believe us, we looked hard...
  - ... other than standard libraries, that is
- Targeting implementation/capabilities may make for interesting identification techniques

Questions? I know you have at least one?

Thanks

- We hope you've enjoyed a wide look into the malware universe ... stay tuned...

- ero.carrera@{virustotal,zynamics}.com
  - http://www.virustotal.com
  - http://www.zynamics.com

- peter.silberman@mandiant.com
  - http://blog.mandiant.com
  - http://www.mandiant.com