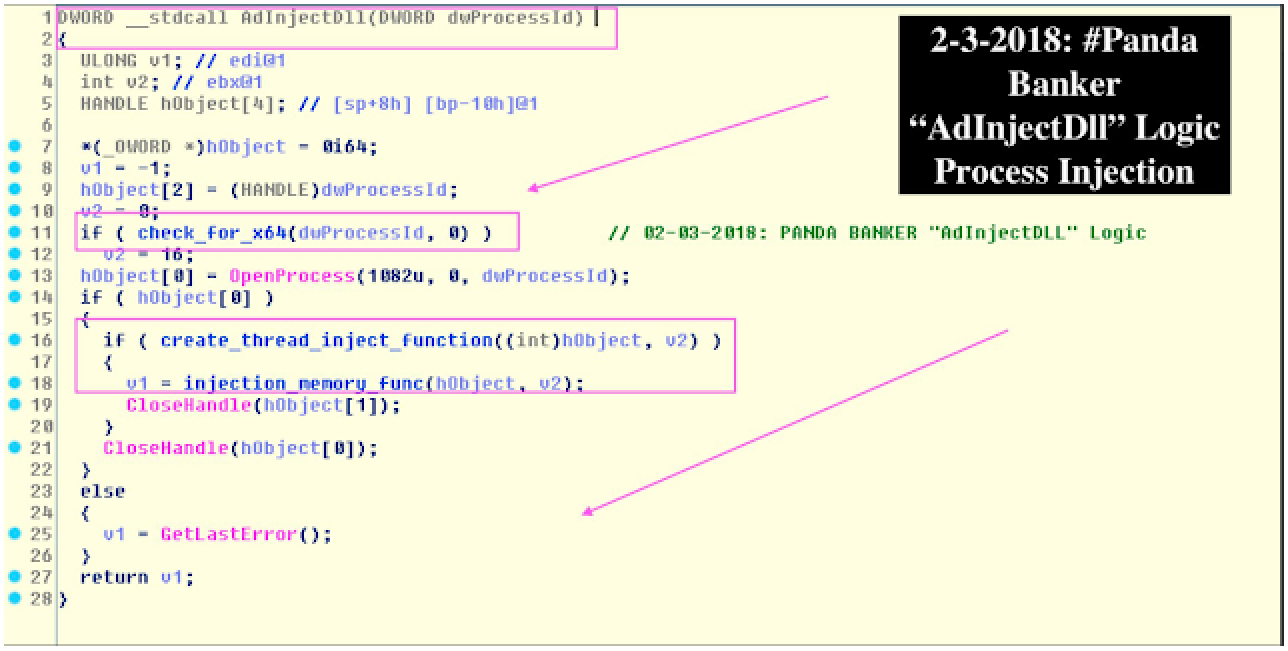
Let's Learn: Dissect Panda Banking Malware's "libinject" Process Injection Module

vkremez.com/2018/01/lets-learn-dissect-panda-banking.html

Goal: Unpack and dissect the Panda banking malware injection DLL module titled "libinject.dll."
Source:
Panda Loader (MD5: 2548a068f7849490c56b63288a8ae5c2)
Panda Loader (unpacked) (MD5: adab9c2b1d897d6a157b82d59f9c2306)
Panda "libinject" (MD5: 47dcbc79f98ff4501619eb5d25da03bd)



```
DWORD __stdcall AdInjectDll(DWORD dwProcessId)
{
  ULONG v1; // edi@1
  int v2; // ebx@1
  HANDLE hObject[4]; // [sp+8h] [bp-10h]@1

  *(_QWORD *)hObject = 0i64;
  v1 = -1;
  hObject[2] = (HANDLE)dwProcessId;
  v2 = 0;
  if ( check_for_x64(dwProcessId, 0) )                    // 02-03-2018: PANDA BANKER "AdInjectDLL" Logic
    v2 = 16;
  hObject[0] = OpenProcess(1082u, 0, dwProcessId);
  if ( hObject[0] )
  {
    if ( create_thread_inject_function((int)hObject, v2) )
    {
      v1 = injection_memory_func(hObject, v2);
      CloseHandle(hObject[1]);
    }
    CloseHandle(hObject[0]);
  }
  else
  {
    v1 = GetLastError();
  }
  return v1;
}
```

2-3-2018: #Panda Banker "AdInjectDll" Logic Process Injection

Background:
While analyzing one of the latest Panda malware spam campaings identified by @JAMESWT, I decided to investigate the binary deeper to see some interesting and/or undisclosed ways the malware interacts with the victim environment. Immediately what stood out to me is Panda's DLL inject module due its compatibility with 32-bit (x86) and 64-bit (x64) architecture.

> #zeus #panda #italy
> /5.196.121.163/connection.jpg
> (thanks to @SettiDavide89)
> — JAMESWT (@JAMESWT_MHT) January 9, 2018

By and large, the Panda banker malware leverages the following Windows NTDLL and kernel32 for process injection:
NtUnmapViewOfSection
ZwWow64ReadVirtualMemory64

ZwGetContextThread
ZwSetContextThread
ZwWow64QueryInformationProcess64
NtProtectVirtualMemory
RtlExitUserThread
NtCreateSection
CreateRemoteThread
WriteProcessMemory
ResumeThread

Analysis:

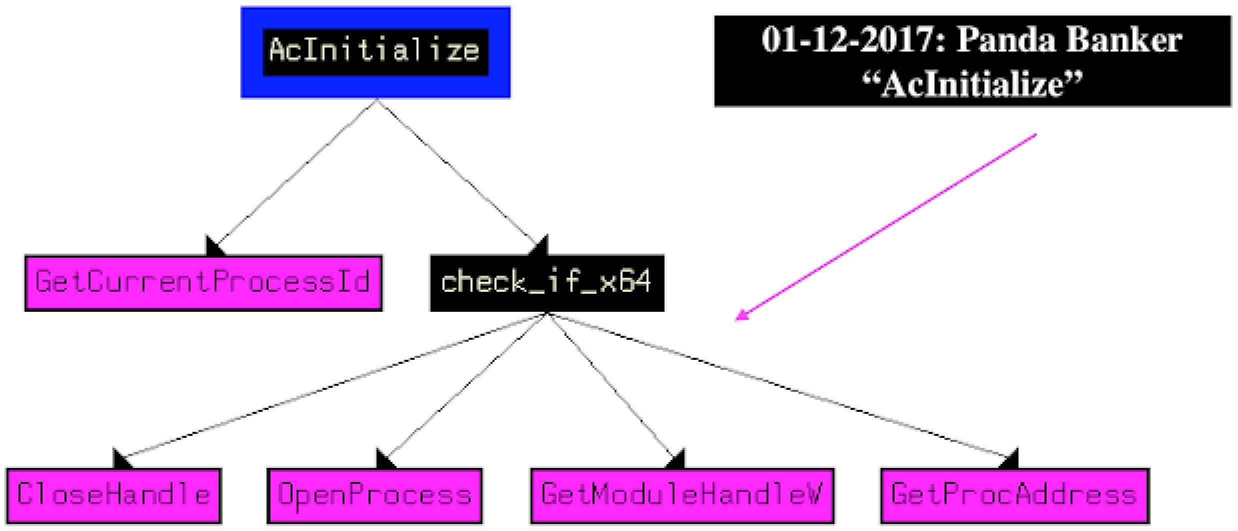
Panda Banker injection module outline
I. Export functions
II. AcInitialize
III. AdInjectDll
IV. Yara Rule

I. Export functions

The Panda export ordinal functions are as follows:

- **AcInitialize**: size_t function type that initializes the structures necessary for the injection export function.
- **AdInjectDll**: DWORD function type that performs the process injection with the argument with the desired process ID (PID) as an argument of the DWRD type.

II. AcInitialize



The functions contain check x64 process check via IsWOW64 returning an integer value.
The pseudocoded C++ function is as follows:
The main AcInitialize module check_x64:

```
  v7 = 0;
  v2 = (FARPROC)dword_10004380;
```

```
if ( dword_10004380
  || (v3 = GetModuleHandleW(L"KERNEL32.DLL"),
     v2 = GetProcAddress(v3, "IsWow64Process"),
     (dword_10004380 = (int)v2) != 0) )
{
  if ( dwProcessId )
  {
    v4 = OpenProcess(1024u, 0, dwProcessId);
    v2 = (FARPROC)dword_10004380;
  }
  else
  {
    v4 = (HANDLE)a2;
  }
  if ( v4 )
  {
    v5 = ((int (__stdcall *)(HANDLE, int *))v2)(v4, &v7);
    v7 = v5 != 0 ? v7 : 0;
    if ( dwProcessId )
      CloseHandle(v4);
  }
}
```

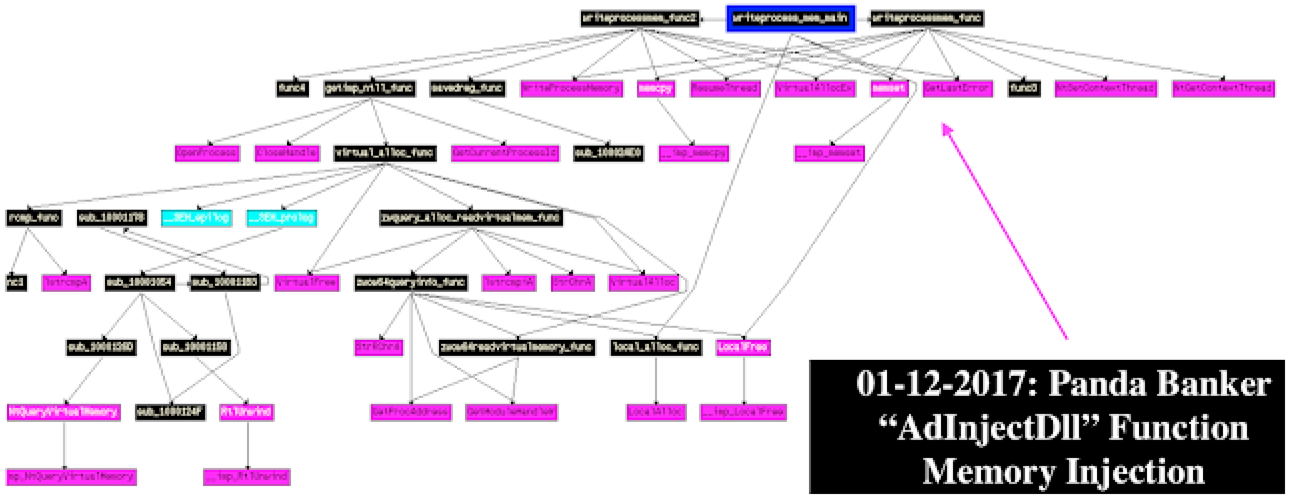
III. AdInjectDll main

The main AdInjectDll sets both createthreadfunction and injectfunction functions, pseudocoded as follows:

```
DWORD __stdcall AdInjectDll(DWORD dwProcessId)
{
  ULONG v1;
  int v2;
  HANDLE hObject[4];
  *(_OWORD *)hObject = 0i64;
  v1 = -1;
  hObject[2] = (HANDLE)dwProcessId;
  v2 = 0;
  if ( check_if_x64(dwProcessId, 0) )
    v2 = 16;
  hObject[0] = OpenProcess(1082u, 0, dwProcessId);
  if ( hObject[0] )
  {
    if ( createthreadfunction((int)hObject, v2) )
    {
      v1 = injectfunction(hObject, v2);
      CloseHandle(hObject[1]);
    }
    CloseHandle(hObject[0]);
  }
  else
  {
    v1 = GetLastError();
  }
  return v1;
}
```

A. createthreadfunction

Creates thread either via CreateRemoteThread or NtCreateThreadEx (or both)



B. injectfunction

The malware createa a section via NtCreate and calls NtMapViewOfSection to unmap the payload in memory.

One of the notable Panda features is its compatibility with x32/x64 architectures is achieved by using **IsWow64Process** (definition of OS architecture).

ZwWow64QueryInformationProcess64-ZwWow64ReadVirtualMemory64 are used for searching NTDLL in PEB, then for searching API addresses required for work of injecting DLL module (x32/x64) which is being located in AP svchost by using NtCreateSection-NtMapViewOfSection-NtUnmapViewOfSection ResumeThread-Sleep-SuspendThread are used for unmapping and injecting the payload into the main thread.

IV. Yara Rule

rule crime_win32_64_panda_libinject_dll_module {

meta:

description = "Panda Banker Iinject DLL modile"

author = "@VK_Intel"

```
reference = "Detects Panda Banker libinject.dll"
date = "2018-01-10"
hash = "75db065b70c6bce9117e46a6201d870e580d07b7c3ee6d2ddab34df0b5dff51f"
strings:
$lib = "libinject.dll" fullword ascii

$export1 = "AdInjectDll" fullword ascii
$export2 = "AcInitialize" fullword ascii

$import0 = "ZwWow64QueryInformationProcess64" fullword ascii
$import1 = "ZwWow64ReadVirtualMemory64" fullword ascii
$import2 = "NtCreateSection" fullword ascii
$import3 = "NtUnmapViewOfSection" fullword ascii
$import4 = "NtGetContextThread" fullword ascii
$import5 = "NtSetContextThread" fullword ascii
$import6 = "WriteProcessMemory" fullword ascii
$import7 = "ResumeThread" fullword ascii
$import8 = "CreateRemoteThread" fullword ascii
$import9 = "VirtualAllocEx" fullword ascii

condition:
all of ($export*) and one of $lib and all of ($import*)
}
```