

Windows Internals ***Crash Course***

Duncan Ogilvie

About me

- Creator and maintainer of [x64dbg](#)
- Love doing open source 🦾
- Used to develop obfuscation
- Currently a security researcher

Outline

- Goal
- Windows internals
- Process monitor

Process

Container to separate applications from each other.

- `_EPROCESS`
- Threads
- Handles
- Memory
 - Modules

Process creation (kernel)

- Initialize address space ⇐
 - Map `KUSER_SHARED_DATA`
 - Map the executable
 - Map `ntdll.dll`

Portable Executable (PE)

- Sections
- Imports
- Exports
- Relocations
- AddressOfEntryPoint
- Subsystem
- ...

Demo

Process creation (kernel)

- Initialize address space
 - Map `KUSER_SHARED_DATA`
 - Map the executable
 - Map `ntdll.dll`
 - Allocate PEB ←

Process Environment Block

- Small memory range
- Storage for process-specific information
 - Environment variables
 - Command line
 - Working directory
 - Module list
 - Heap pointer

Process creation (kernel)

- Initialize address space
 - Map `KUSER_SHARED_DATA`
 - Map the executable
 - Map `ntdll.dll`
 - Allocate PEB
- Create initial thread ⇐
 - Allocate stack
 - Allocate TEB
 - `ntdll.LdrInitializeThunk`

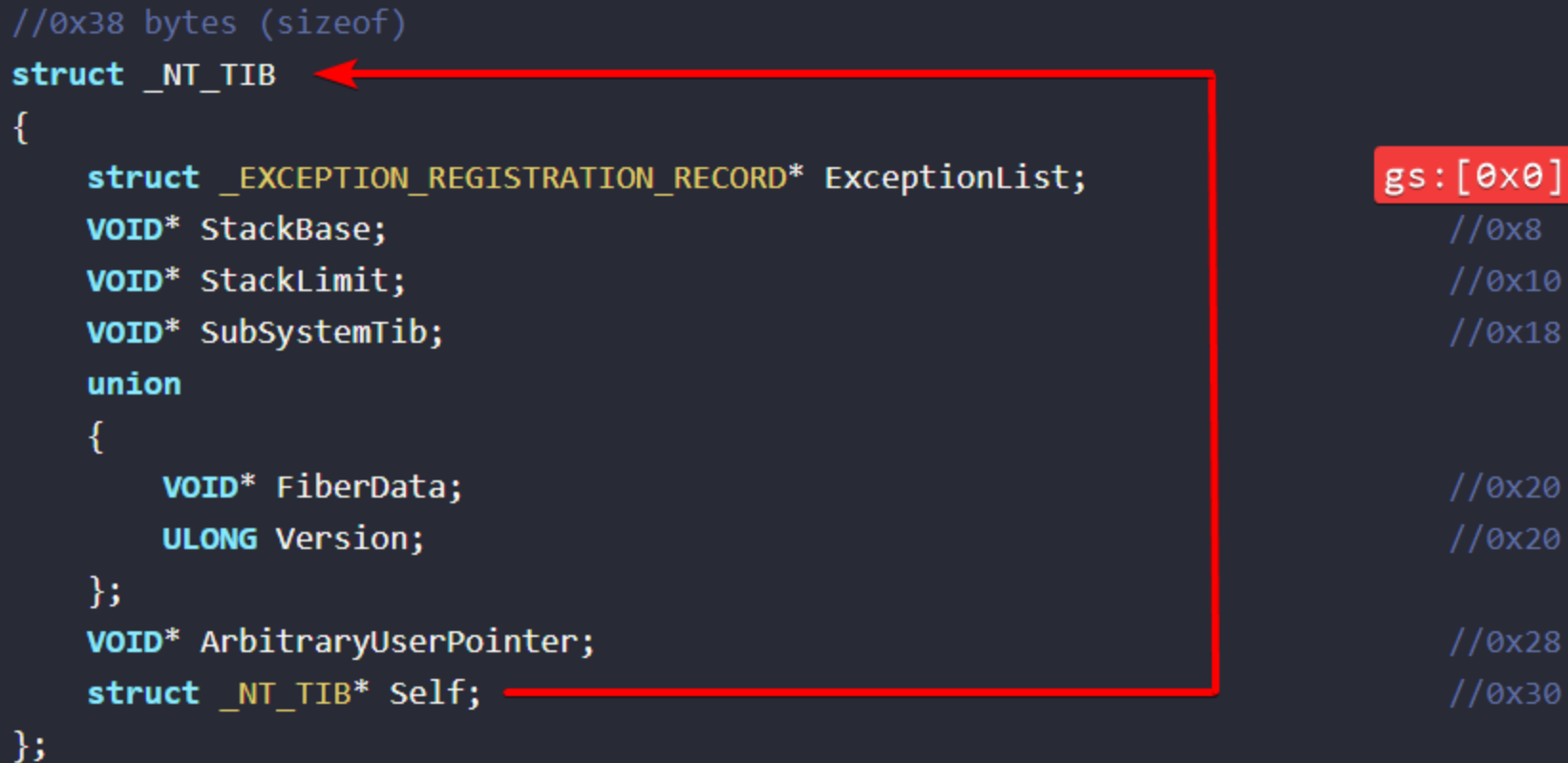
Thread Environment Block

- Small memory range
- Storage for thread-specific information
 - Thread ID
 - Stack range
 - GetLastError
 - TLS: Thread Local Storage
- `gs:[X] = [IA32_KERNEL_GS_BASE + X]`

Thread Environment Block

```
//0x38 bytes (sizeof)
struct _NT_TIB ←
{
    struct _EXCEPTION_REGISTRATION_RECORD* ExceptionList;
    VOID* StackBase;
    VOID* StackLimit;
    VOID* SubSystemTib;
    union
    {
        VOID* FiberData;
        ULONG Version;
    };
    VOID* ArbitraryUserPointer;
    struct _NT_TIB* Self;
};
```

gs:[0x0] //0x8
//0x10
//0x18
//0x20
//0x20
//0x28
//0x30



Calling conventions

```
BOOL WINAPI DllMain(  
    HINSTANCE hinstDLL, // RCX  
    DWORD fdwReason,    // RDX  
    LPVOID lpReserved   // R8  
);
```

- Parameters: RCX, RDX, R8, R9
- Volatile vs Non-volatile
- Documentation (used registers)

DllMain/TLS Callbacks

- Notification of thread start/end
- Used for initialization
- DLLs vs Executables
- Loader lock (DllMain)

Demo!

Summary

- Process creation
- PE format
- TEB/PEB
- Calling conventions
- TLS callbacks

Questions?

Debuggers

```
CreateProcess(..., DEBUG_ONLY_THIS_PROCESS, ...);

while(true) {
    DEBUG_EVENT e;
    WaitForDebugEvent(&e, -1);
    DWORD continueStatus = DBG_EXCEPTION_NOT_HANDLED;
    switch(e.dwDebugEventCode) {
        case CREATE_PROCESS_DEBUG_EVENT: ...
        case LOAD_DLL_DEBUG_EVENT: ...
        case EXCEPTION_DEBUG_EVENT: ...
        ...
    }
    ContinueDebugEvent(e.dwProcessId, e.dwThreadId,
        continueStatus);
}
```

Process creation (kernel)

- Initialize address space
 - Map `KUSER_SHARED_DATA`
 - Map the executable
 - Map `ntdll.dll`
 - Allocate PEB
- Create initial thread
 - Allocate stack
 - Allocate TEB
 - `ntdll.LdrInitializeThunk` ⇐

LdrInitializeThunk

```
void LdrInitializeThunk(  
    PCONTEXT ContextRecord,  
    PVOID Parameter /* ntdll base */  
);
```

- Load imported DLLs
- Loader lock
- TLS callbacks/DllMain
- ZwContinue -> RtlUserThreadStart

ZwContinue

```
NTSTATUS ZwContinue(  
    PCONTEXT ContextRecord,  
    BOOLEAN TestAlert  
);
```

Continues execution of the current thread with a different context.

Demo!

RtlUserThreadStart

```
void RtlUserThreadStart(  
    PTHREAD_START_ROUTINE Function,  
    PVOID Parameter  
);
```

- ntdll.RtlUserThreadStart
- kernel32.BaseThreadInitThunk
- OptionalHeader.AddressOfEntryPoint
 - mainCRTStartup(PEB) -> main(argc, argv)

Demo!

Summary

- Process creation
- Executable loading
- Kernelmode → Usermode

Questions?

Syscalls

```
NTSTATUS ZwSuspendThread(HANDLE ThreadHandle, PULONG PreviousSuspendCount);
```

```
ntdll.ZwSuspendThread:  
    mov r10, rcx    ; rcx is cluttered by syscall  
    mov eax, 0x1BC ; syscall number  
    test byte ptr ds:[KUSER_SHARED_DATA.SystemCall], 0x1  
    jne @legacy  
    syscall  
    ret  
@legacy:  
    int 0x2E  
    ret
```

Callbacks

- LdrInitializeThunk
- KiUserExceptionDispatcher
- KiUserCallbackDispatcher
- ...

Initialization: PspInitializeSystemDlls

KiUserExceptionDispatcher

```
void __usercall KiUserExceptionDispatcher(  
    PEXCEPTION_RECORD ExceptionRecord,  
    PCONTEXT ContextRecord  
);
```

- Executed on segfaults/interrupts
- Handles the exception in user-mode

Demo!

Summary

- Process initialization
- Syscalls
- Callbacks

Questions?

Process Monitor

- Simple GUI
- Shows process events
- Filter driver
- Notification callbacks

Demo!

Thanks

- JustMagic
- Brit
- xenocidewiki
- herrcore
- Fiske
- Karl
- Can

Questions?