Getting Started with Windows Malware Development

rad98

whoami

- Senior Security Engineer
- Passionate malware developer
- Black Mass Volume 1 Contributor

If you give a man a fish, you feed him for a day. If you teach a man to fish, you feed him for a lifetime.

Step 1: Learning to Program

- Malware is **JUST** Malicious Software
  - Often we want to blend in as legitimate software
  - Write our code in the same languages used for those
- Many programming languages
  - Different use cases for each
  - Identify your goal and pick the most suitable language to achieve that goal

Common Languages

Implant :

- ➔ C - *C, a Software Engineering Approach*
- ➔ C++ - *Effective Modern C++*
- ➔ Golang - *Go by Example*
- ➔ Rust - *The Rust Programming Language*

Backend:

- ➔ Pretty much anything that works, works.
  - ◆ Golang/Rust/C++/Python

- ➔ Language wars are pointless, use what works (these are just examples)
- ➔ Be comfortable with reading and writing multiple languages

Step 2: Develop a basic understanding of Assembly

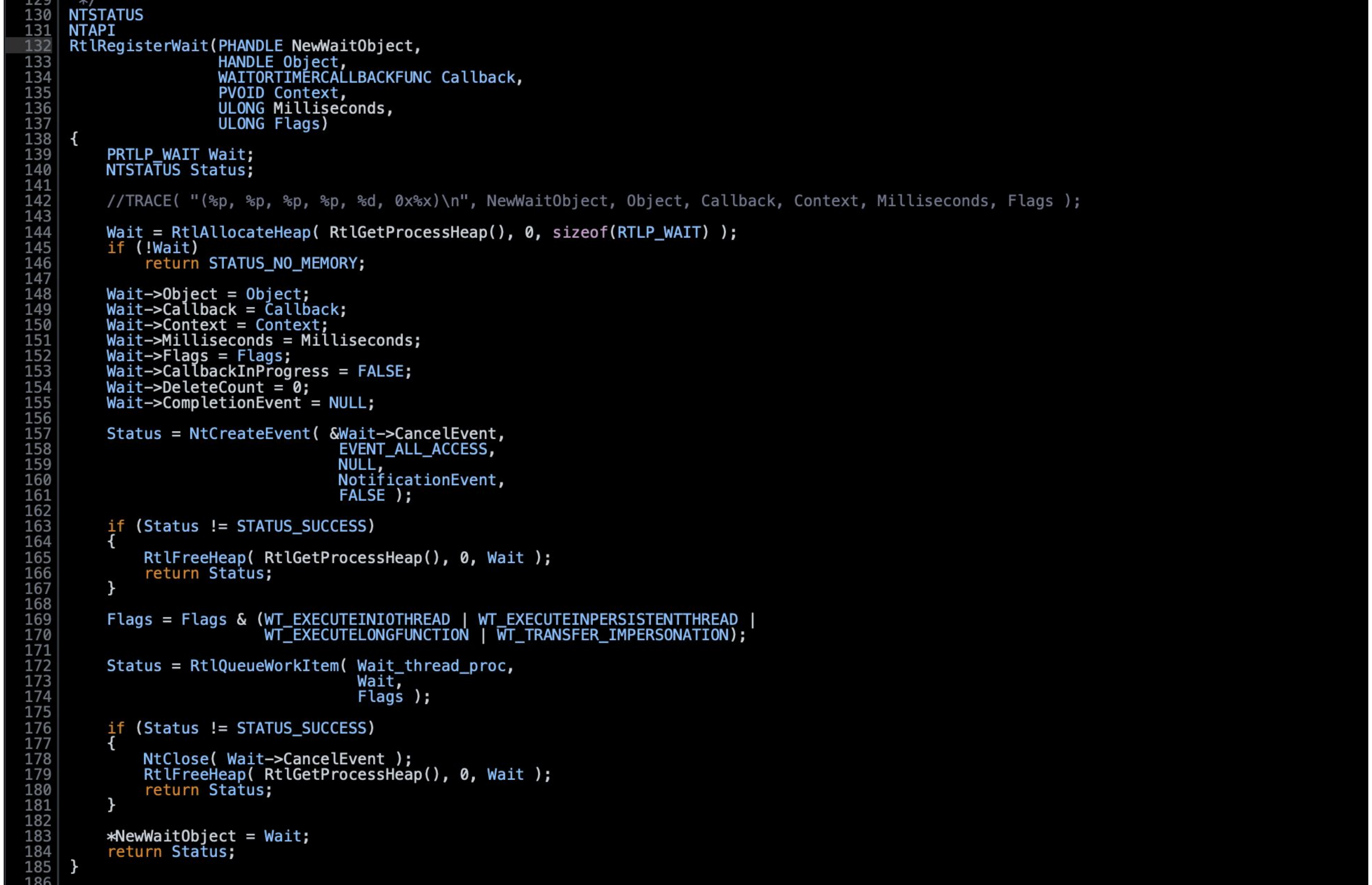
- *Computer Systems: A Programmer's Perspective*
- Arch1001 OST2
- Compiler Explorer (https://godbolt.org/)
  - Understand what your code looks like in assembly
- Example use case for indirect syscalls and callstack spoofing: https://github.com/HavocFramework/Havoc/tree/dev/payloads/Demon/Source/Asm
- Get used to debugging with WinDbg Preview/x64dbg (Debuggers 1011 OST2)

- Be comfortable with reading **both** Intel and AT&T instructions
- Be comfortable with debugging code with and without symbols/source code

Step 3: Understand Windows Internals Fundamentals

- ➔ First 3 Chapters of Windows Internals 7th Edition, Part 1
- ➔ Understand system architecture
  - ◆ Memory
  - ◆ Threading
  - ◆ Processes
- ➔ Read ReactOS source to further correspond your understanding
- ➔ Follow along with exercises in the book
- ➔ Reverse engineer system components which are interesting to you

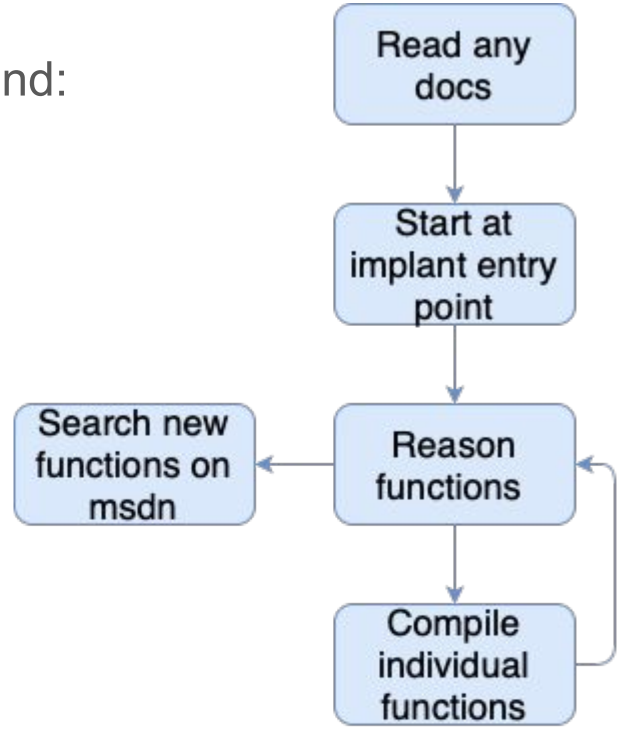
- ➔ Understand how to answer Windows internal specific questions
  - ◆ Where to look?



```
 */
NTSTATUS
NTAPI
RtlRegisterWait(PHANDLE NewWaitObject,
                HANDLE Object,
                WAITORTIMERCALLBACKFUNC Callback,
                PVOID Context,
                ULONG Milliseconds,
                ULONG Flags)
{
    PRTLP_WAIT Wait;
    NTSTATUS Status;

    //TRACE( "(%p, %p, %p, %p, %d, 0x%x)\n", NewWaitObject, Object, Callback, Context, Milliseconds, Flags );

    Wait = RtlAllocateHeap( RtlGetProcessHeap(), 0, sizeof(RTLP_WAIT) );
    if (!Wait)
        return STATUS_NO_MEMORY;

    Wait->Object = Object;
    Wait->Callback = Callback;
    Wait->Context = Context;
    Wait->Milliseconds = Milliseconds;
    Wait->Flags = Flags;
    Wait->CallbackInProgress = FALSE;
    Wait->DeleteCount = 0;
    Wait->CompletionEvent = NULL;

    Status = NtCreateEvent( &Wait->CancelEvent,
                             EVENT_ALL_ACCESS,
                             NULL,
                             NotificationEvent,
                             FALSE );

    if (Status != STATUS_SUCCESS)
    {
        RtlFreeHeap( RtlGetProcessHeap(), 0, Wait );
        return Status;
    }

    Flags = Flags & (WT_EXECUTEINIOTHREAD | WT_EXECUTEINPERSISTENTTHREAD |
                     WT_EXECUTELONGFUNCTION | WT_TRANSFER_IMPERSONATION);

    Status = RtlQueueWorkItem( Wait_thread_proc,
                               Wait,
                               Flags );

    if (Status != STATUS_SUCCESS)
    {
        NtClose( Wait->CancelEvent );
        RtlFreeHeap( RtlGetProcessHeap(), 0, Wait );
        return Status;
    }

    *NewWaitObject = Wait;
    return Status;
}
```

Step 4: Read leaked malware source codes

- ➔ Malware - Start from the entry point and understand:
  - ◆ WHAT is being performed?
  - ◆ WHY is it being performed?
  - ◆ Are there other ways to perform the same behaviour?
    - ● (look at ReactOS/reverse the API(s) being used)
- ➔ Never seen the API used?
  - ◆ site:https://learn.microsoft.com "FunctionName"
  - ◆ RTFM!!


Read any docs
Start at implant entry point
Reason functions
Search new functions on msdn
Compile individual functions

Good Starting Points

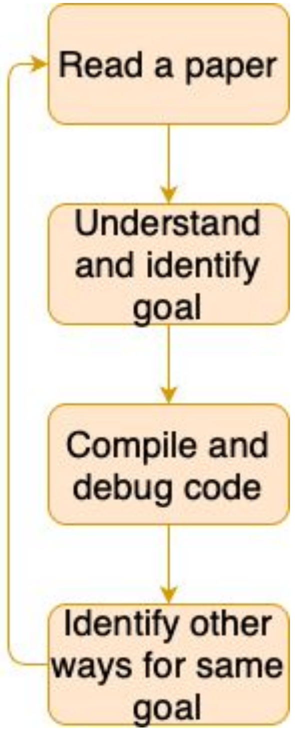
- ➔ https://github.com/vxunderground/MalwareSourceCode/tree/main/Win32
- ➔ Start from the entrypoint and then follow code paths
  - Zeus
  - m0yv
  - Carbanak
- ➔ No need to reinvent the wheel, if it was done 5 years ago and worked!
  - But it may not be the best way, so always keep it in mind

- ➔ Be comfortable reading various malware source code and reasoning with the code

Step 5: Resources

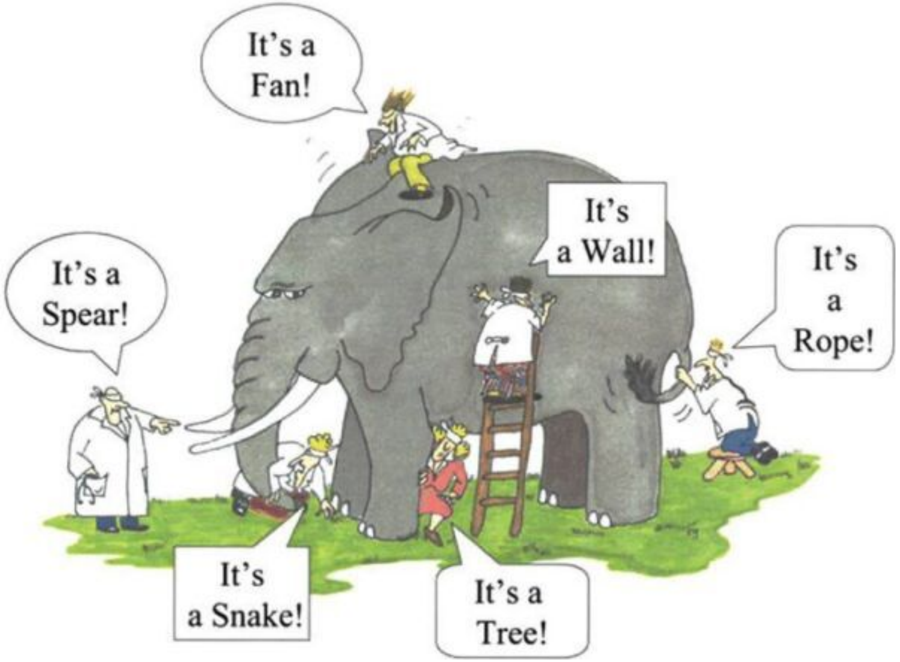
- ➔ https://www.vx-underground.org/windows.html
  - ◆ Curated list of good quality papers

- ➔ https://modexp.wordpress.com/
- ➔ https://www.x86matthew.com/
- ➔ https://pre.empt.blog/
- ➔ https://github.com/stephenfewer/ReflectiveDLLInjection
- ➔ https://github.com/rapid7/metasploit-payloads/tree/master/c/meterpreter
- ➔ https://www.youtube.com/@OALABS


Read a paper
Understand and identify goal
Compile and debug code
Identify other ways for same goal

Step 6: The Map is Not The Territory

- ➔ Maps represent reality but are often not reality
- ➔ e.g. MSDN represents reality
  - ◆ But what is real is often different
- ➔ Don’t blindly trust the map (documentation)
- ➔ Explore the terrain by seeing what is actually happening
  - ◆ Reverse engineer the necessary components to answer the questions you ask



Step 7: Ask better questions

- ➔ Ask yourself and others better questions and you’ll get better answers
- ➔ Understand what you are trying to solve, and the end goal

“Whoever best describes the problem is the one most likely to solve it”

- Dan Roam

Any Questions?