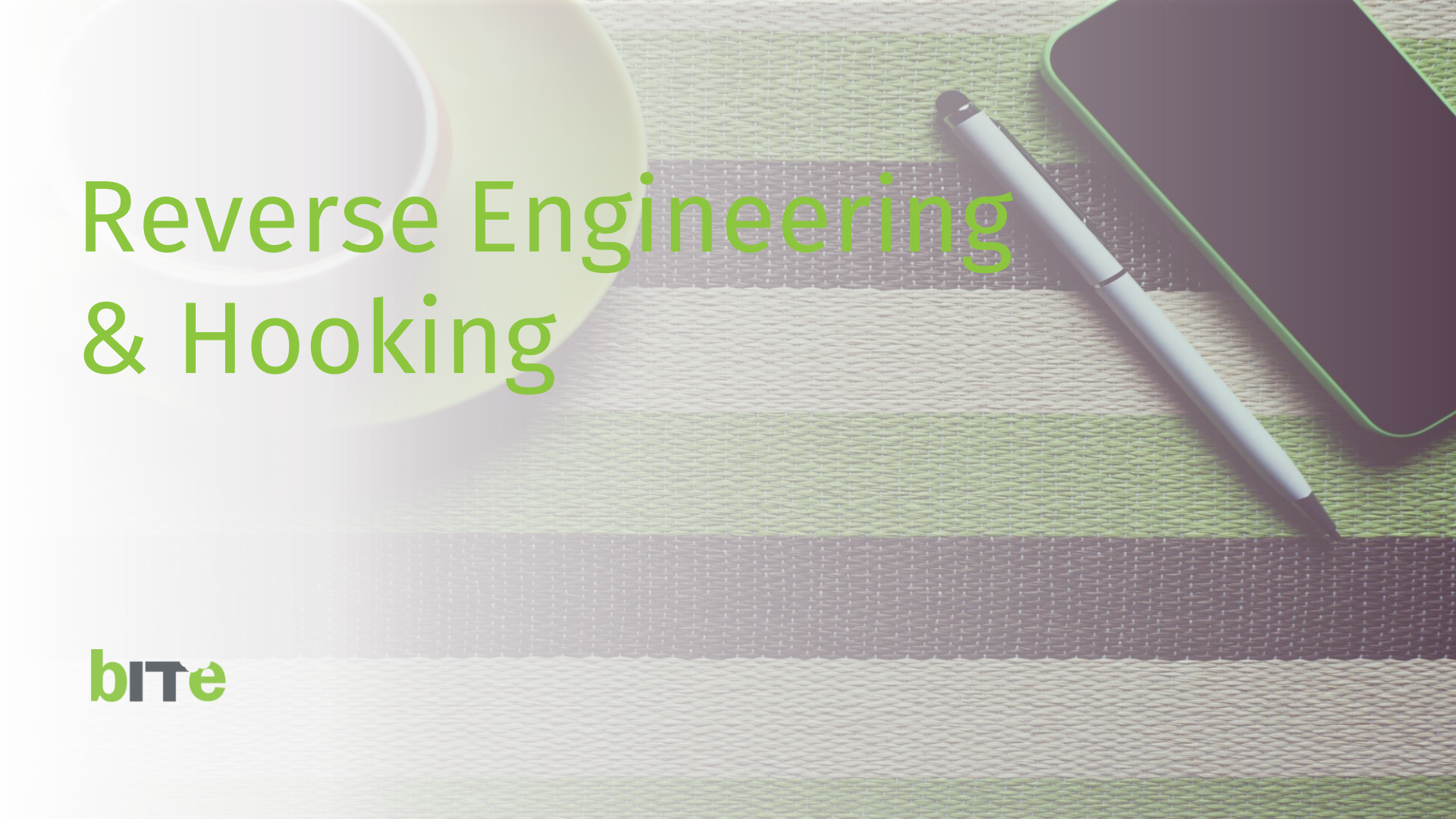
Reverse Engineering & Hooking



Summary

- Interlude
- Introduction
  - The Reverse Engineering Process
  - Memory Registers
  - Essential Instructions
  - Call Stack & Stack Frame
- Debugging
  - Patching Memory
- DLL Injection
  - Calling Functions
  - Signature Scanning
  - Detour|Mid|End Hook
- Tools
- Live Example

Interlude

- Why did I learn C++ and Reverse Engineering?

- Prerequisites
  - C++
  - CPU Architecture (registers)

The Reverse Engineering Process

- Static Analysis
  - Disassembly
- Dynamic Analysis
  - Tracing behavior

Memory Registers

32 bit registers	Type	Description
EAX	Primary accumulator	Used for input/output & most arithmetic instructions
EBX	Base register	Used for indexed adressing
ECX	Count register	Used for storing loop count in iterative operations
EDX	Data register	Used for input/output & most arithmetic instructions (large data)
ESI	Source Index	Used for copy operations
EDI	Destination Index	Used for copy operations
EBP	Base pointer	Used in order to create the stackframe for a function that is the space that is used for arguments and local variables
ESP	Stack pointer	Responsible for keeping track of the top of the stack in memory
EIP	Instruction pointer	Tells the CPU to go in memory to fetch the 'next' instruction

Essential Instructions

- ISA (Instruction Set Architecture)
  - Examples: x86 Intel

ISA	Type	
ADD <DEST>, <SOURCE>	Arithmetic	ADD | SUB | INC | DEC | MUL | DIV
MOV <DEST>, <SOURCE>	Memory	
CMP <arg1>, <arg2>	Comparison	UPDATE FLAGS -> E/RFALGS (CF, OF, SF, ZF, AF, PF)
JMP <DEST>	Control flow	JMP | CALL (to POINTER)

Call Stack & Stack Frame

- Prologue
- Epilogue

```
example_function:
    push    EBP
    mov     EBP, ESP        PROLOGUE
    sub     ESP, 0x08
    ..
    mov     ESP, EBP
    pop     EBP             EPILOGUE
    ret
```

```
sample_function_call:
    0x401000    push 3
    0x401002    push 2
    0x401004    push 1
    0x401006    call function
    0x40100A    add ESP, 0xC
```

- Arguments
  - Each 'push' decrements value of ESP

Stack	
3	HIGHER
2	
1	
0x40100A	
(OLD) EBP	
Local variables	
	LOWER

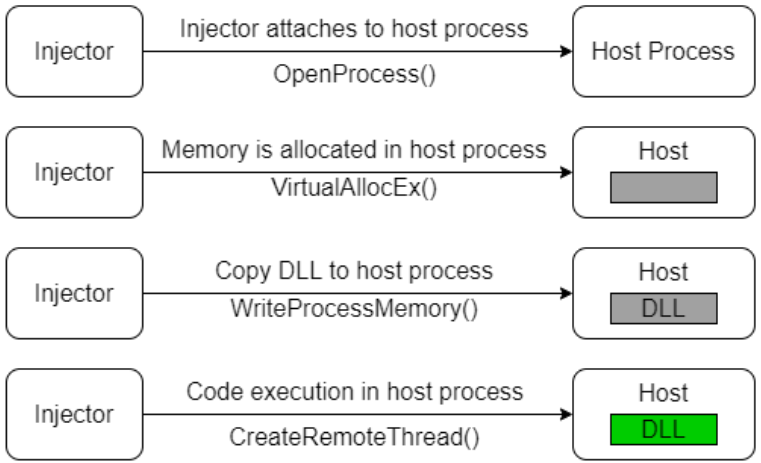
Debugging

- Dynamic Patching (Runtime)
  - Change memory instructions to a running application
  - What is nopping?
    - I.E. No Operation instruction (0x90)
- Static Patching
  - Create a 'patched' application

- Demo - SlnCrackMe

DLL Injection

- Injecting a DLL into a running application
  - Extends functionality
    - Logging
    - Optimizing



- Demo - Injector

Calling Functions

- How to call functions from main app?
  - Keep in mind 'calling conventions'

Keyword	Stack cleanup	Parameter Passing
__cdecl	Caller	Pushes parameters on the stack, in reverse order
__clrcall	N/A	Load parameters onto CLR expressions stack, in order
! __stdcall	Callee	Pushes parameters on the stack, in reverse order
! _fastcall	Callee	Stored in registers, then pushed on the stack
! __thiscall	Callee	Pushes on the stack, this pointer stored in ECX
__vectorcall	Callee	Stored in registers, then pushed on thack, in reverse order

- Demo - SlnCallingFunctions

Signature Scanning

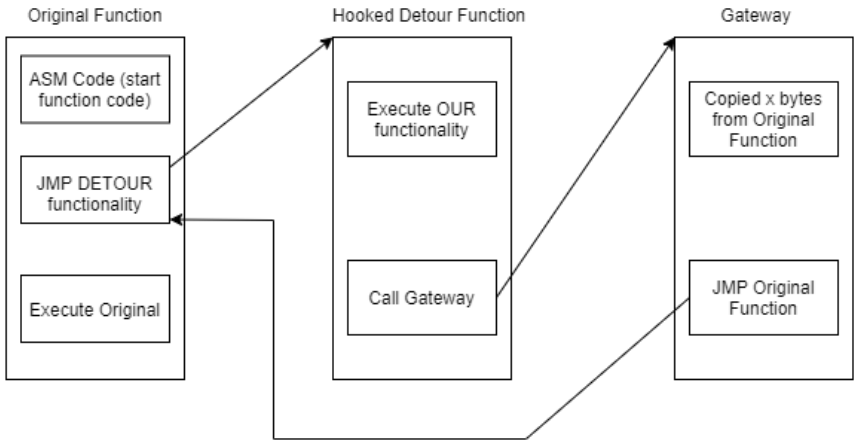
- Each time a programs compiles, addresses are randomized
- With 'signature scanning' you can find specific address back with a certain instruction set
  - Drawbacks?
    - If 'original code' is changed at that location, the signature differs

- Other method?
  - Calculate RVA (Relative Virtual Address)
    - Offset between 'Virtual Address' and 'ImageBase'

- Demo – (signature scan will be shown later)

Detour Hook

- Detour Hook (trampoline of codecave)
- Main use?
  - Argument interception

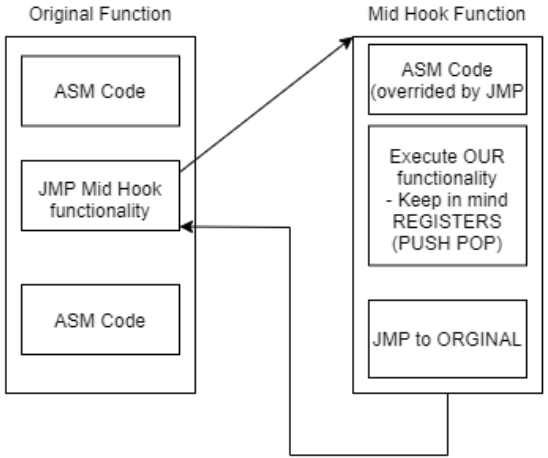


- Demo - SlnCalculator

Mid Hook

- What if I need to change a certain specific memory space?

- Main use?
  - Local variable interception
  - Replace existing logic



- Demo - SlnHeavyLogic

End Hook

- The same as a mid hook function but we don't know where the ending of a function is
  - Manually search for it and apply same logic as 'Mid Hook'

Tools

- Visual Studio
- Cheat Engine
- IDA PRO
- x32dbg (scylla)

Live Example

Questions?