



CRACK.ME UP!!

AN INTRODUCTION TO

BINARY REVERSE ENGINEERING

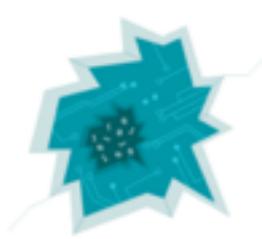
André Baptista

[@0xACB](https://twitter.com/0xACB)



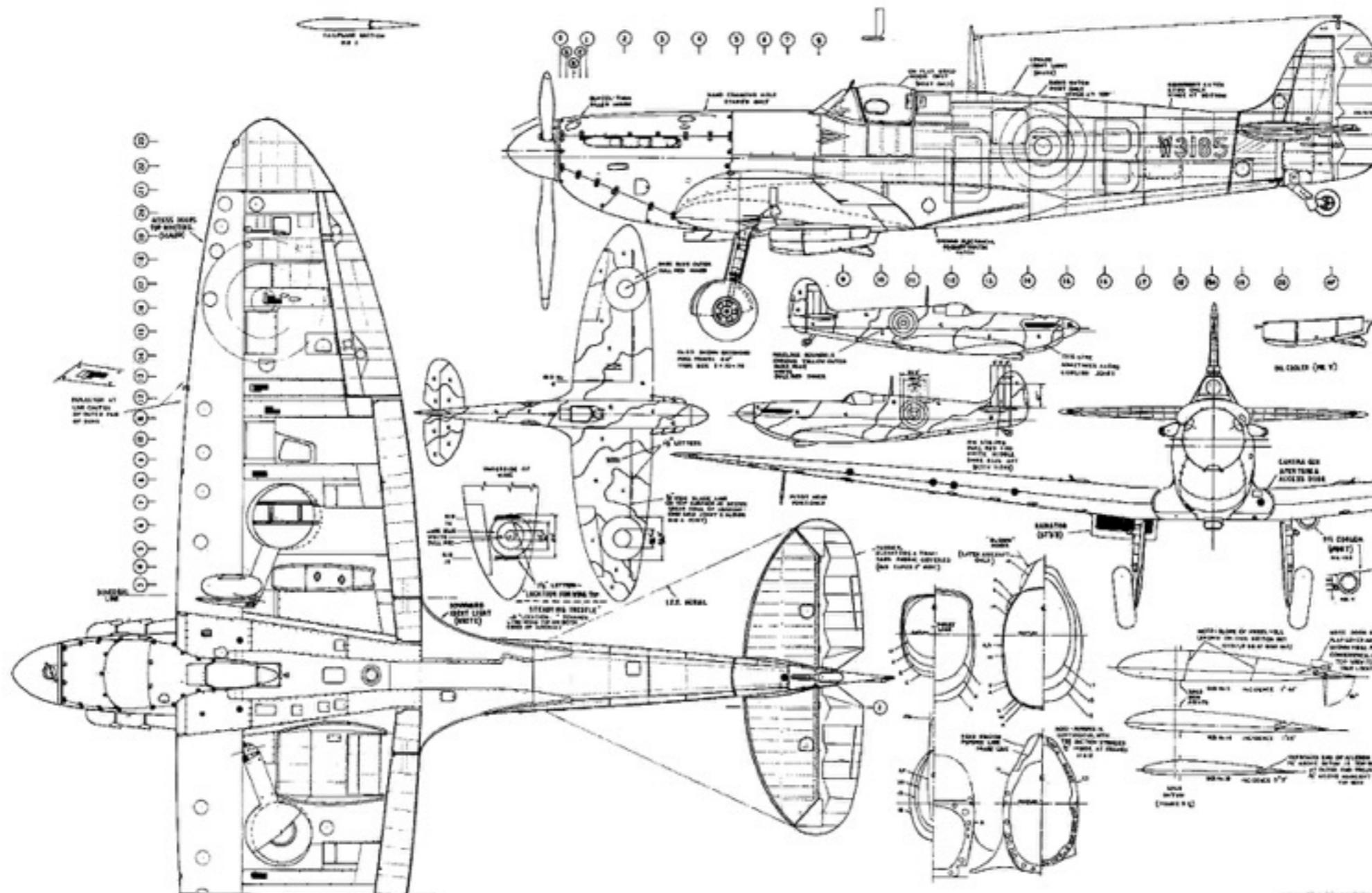
Reverse Engineering

- Uncovering the hidden behaviour of a given technology, system, program, protocol or device, by analysing the structure and operation of its components
- Extracting knowledge about any unknown engineering invention



0xOPOSEC Meetup

History

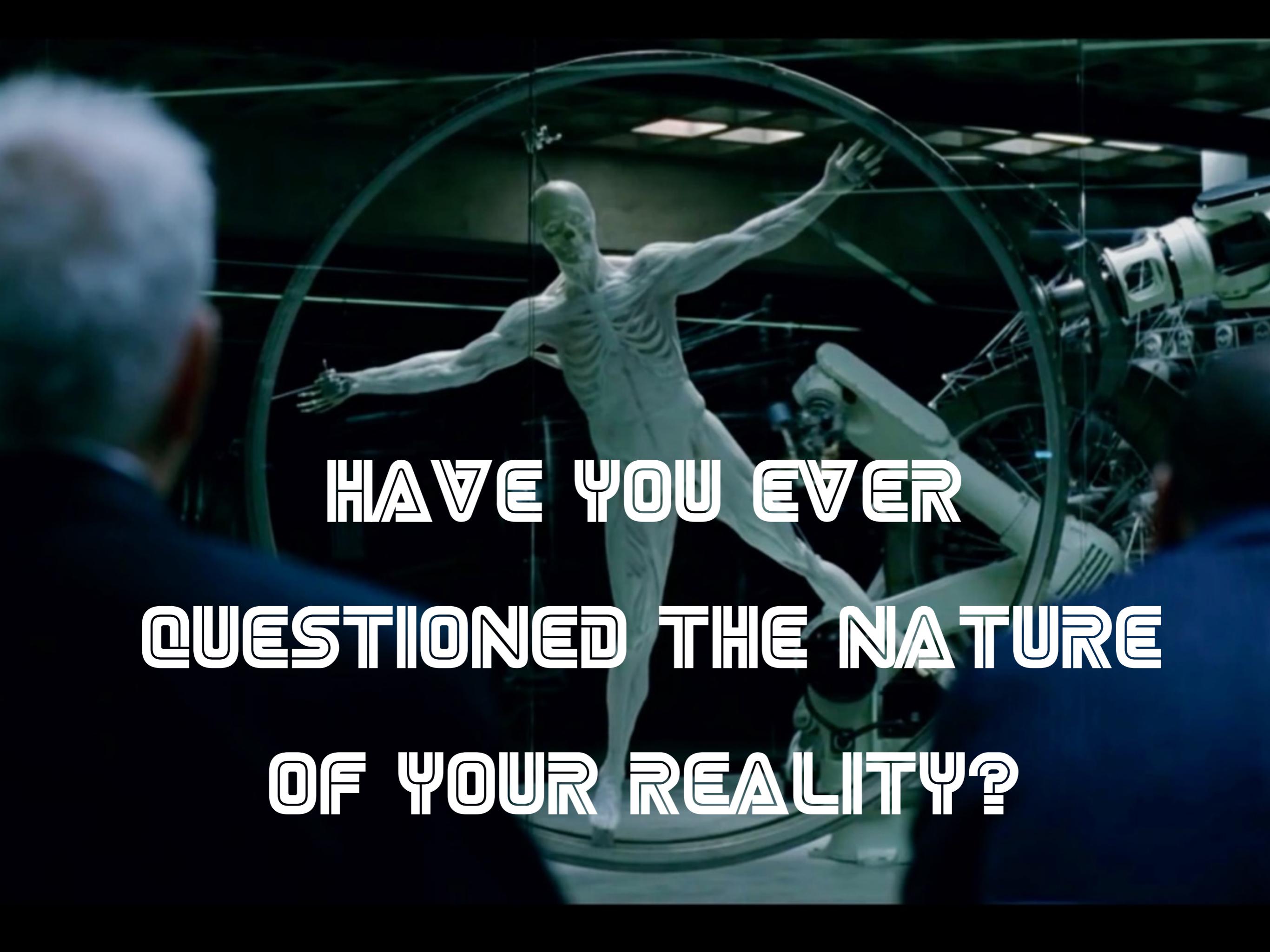




Reverse Engineering

History

- RE was used to copy inventions made by other countries or business competitors
- Frequently used in the WW2 and Cold War:
 - *Jerry can*
 - *Panzerschreck*

A white, skeletal human figure with a translucent, ethereal quality is floating in a dark, metallic space station corridor. The figure's arms are outstretched, and its bones are clearly visible against the dark background. The corridor is filled with various pipes, cables, and structural elements of the space station.

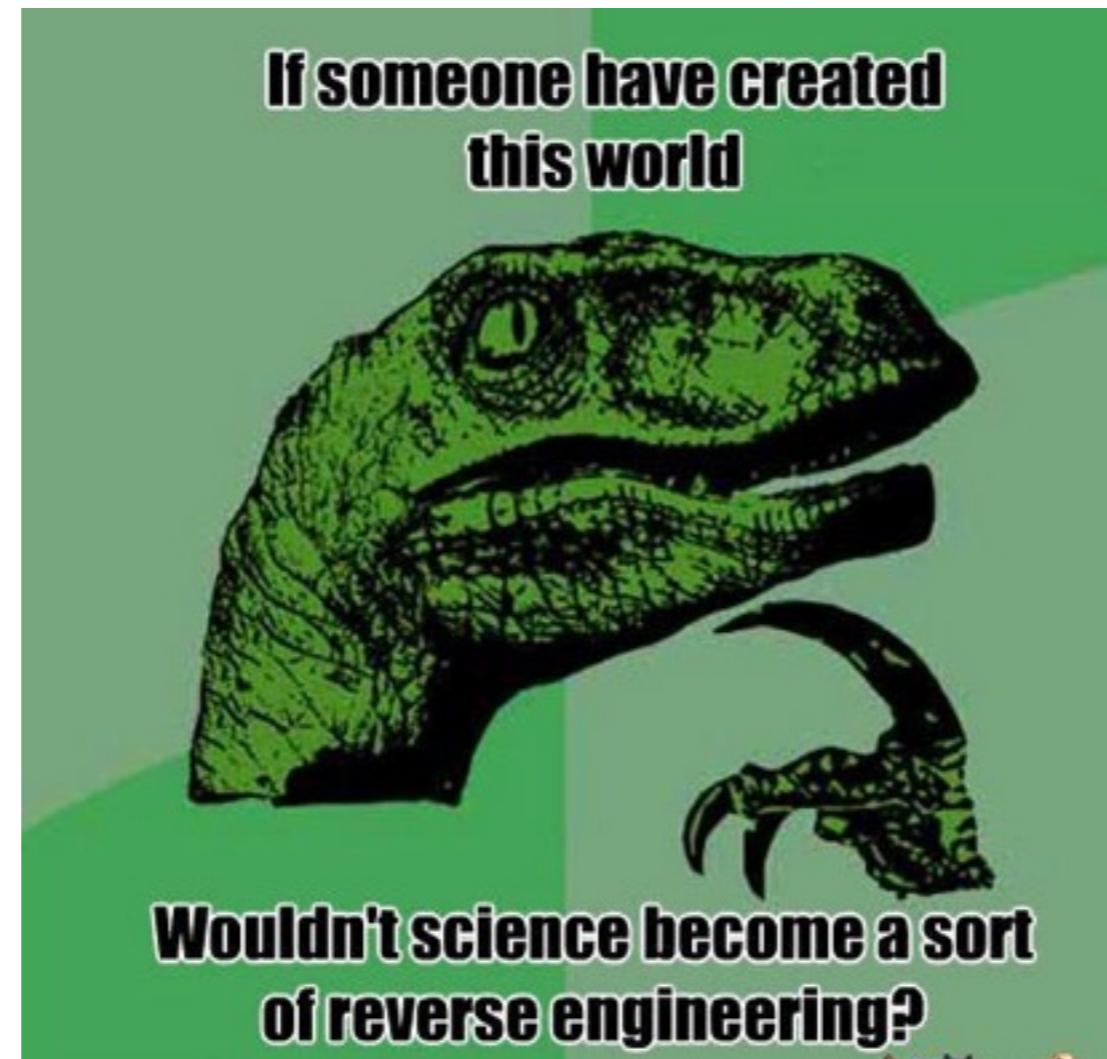
HAVE YOU EVER
QUESTIONED THE NATURE
OF YOUR REALITY?



OxOPOSEC Meetup

Reverse Engineering

"Is biology reverse engineering?"





OxOPOSEC Meetup

Binary Reverse Engineering

- It's the process of getting knowledge about compiled software, in order to understand how it works and how it was originally implemented.



Binary Reverse Engineering

- It's the process of extracting knowledge about compiled software in order to understand what it does and how it was originally implemented.

**WITHOUT THE
SOURCE CODE**



Binary Reverse Engineering

Motivation

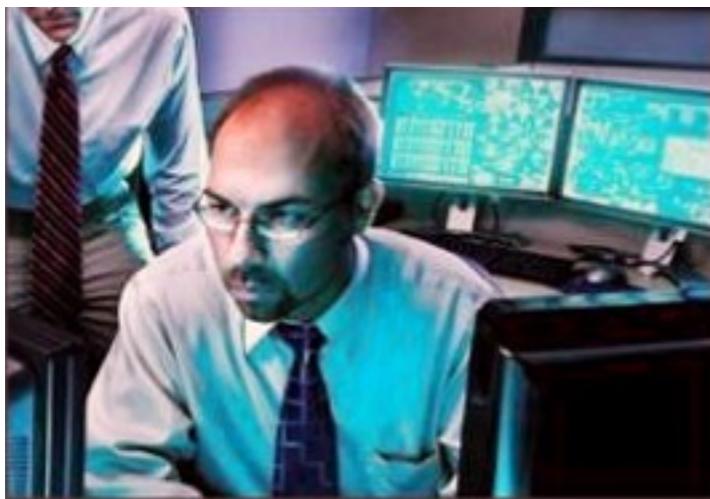
- Software and hardware cracking
- Malware analysis - botnet clients, spyware, ransomware
- Finding bugs in compiled software
- Creating or improving docs
- Interpreting unknown protocols
- Academic purposes
- Industrial or military espionage
- Software interoperability



OxOPOSEC Meetup

Who knows how to do this stuff?

- Hackers in general
- Some intelligence agencies
- Antivirus companies
- Students and curious people





Wanna Cry?

The screenshot shows a Windows application window titled "Wana Decrypt0r 2.0". The main message is "Ooops, your files have been encrypted!" in large white text on a red background. Below it, a large red padlock icon is displayed. The window is divided into several sections:

- What Happened to My Computer?**: Your important files are encrypted. Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.
- Payment will be raised on**: 1/4/1970 01:00:00
Time Left: 00:00:00:00
- Your files will be lost on**: 1/8/1970 01:00:00
Time Left: 00:00:00:00
- Can I Recover My Files?**: Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time. You can decrypt some of your files for free. Try now by clicking <Decrypt>. But if you want to decrypt all your files, you need to pay. You only have 3 days to submit the payment. After that the price will be doubled. Also, if you don't pay in 7 days, you won't be able to recover your files forever. We will have free events for users who are so poor that they couldn't pay in 6 months.
- How Do I Pay?**: Payment is accepted in Bitcoin only. For more information, click <About bitcoin>. Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>. And send the correct amount to the address specified in this window. After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am CEST.
- Send \$600 worth of bitcoin to this address:** 13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94 [Copy](#)
- Contact Us**
- Check Payment**
- Decrypt**

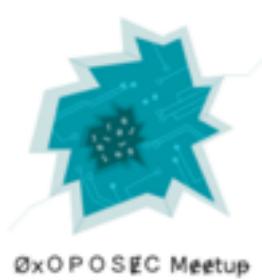


The kill switch

iuquerfsodp9ifjaposdfjhgosurijfaewrwegwea[dot]com

```
qmemcpy(&szUrl, sinkholeddomain, 0x39u);      // previously unregistered domain, now sinkholed
v8 = 0;
v9 = 0;
v10 = 0;
v11 = 0;
v12 = 0;
v13 = 0;
v14 = 0;
v4 = InternetOpenA(0, 1u, 0, 0, 0);
v5 = InternetOpenUrlA(v4, &szUrl, 0, 0, 0x84000000, 0); // do HTTP request to previously unregistered domain
if ( v5 )                                         // if request successful quit
{
    InternetCloseHandle(v4);
    InternetCloseHandle(v5);
    result = 0;
}
else                                                 // if request fails, execute payload
{
    InternetCloseHandle(v4);
    InternetCloseHandle(0);
    detonate();
    result = 0;
}
return result;
}
```

I



Then...

Neel Mehta

@neelmehta
Security researcher, Google.
Joined June 2009

TWEETS 31 **FOLLOWERS** 6,213 **LIKES** 3

Tweets Tweets & replies

Neel Mehta @neelmehta · May 15
9c7c7149387a1c79679a87dd1ba755bc @ 0x402560, 0x40F598
ac21c8ad899727137c4b94458d7aa8d8 @ 0x10004ba0, 0x10012AA4
#WannaCryptAttribution

22 235 306

Neel Mehta @neelmehta · Apr 3
Investigating Chrysaor, Android #malware used in targeted attacks:



IDA View-A Pseudocode-A Hex View-1 Structures Enums Imports Exports

```
16 u_short *v15; // [sp+18h] [bp+4h]@2
17
18 v1 = (int *)a1;
19 v2 = *(DWORD *)a1;
20 LOBYTE(v2) = 1;
21 v3 = (_BYTE *)(a1 + 5);
22 *(DWORD *)a1 = v2;
23 *(v3 - 1) = 3;
24 *v3 = 1;
25 sub_408130(a1 + 6, 32);
26 v4 = time(0);
27 *(DWORD *)(a1 + 6) = htonl(v4);
28 *(BYTE *)(a1 + 38) = 0;
29 v5 = (u_short *)(a1 + 39);
30 v6 = 0;
31 v7 = 6 * (rand() % 5 + 2);
32 if ( v7 > 0 )
33 {
34     v15 = (u_short *)(a1 + 41);
35     while ( 1 )
36     {
37         v8 = rand() % 0x4Bu;
38         v9 = 0;
39         v14 = v8;
40         if ( v6 > 0 )
41             break;
42 LABEL_8:
43         if ( v8 == -1 )
44             goto LABEL_10;
45         *v15 = htons(*(&table_03_00_04_00 + v8));
46 LABEL_11:
47         ++v6;
48         ++v15;
49         if ( v6 >= v7 )
50             goto LABEL_12;
51     }
52     v10 = *(&table_03_00_04_00 + v8);
53     v11 = v5 + 1;
54     while ( *v11 != v10 )
```

000025C1 sub_402560:34

3e6de9e2baacf930949647c399818e7a2caea2626df6a468407854aaa515eed9

RD __stdcall sub_402A40(_DWORD, char);
RD __cdecl sub_408130(_DWORD, _DWORD);

Wanna Cry



Occurrences of binary: 03...

```
16 _WORD *v15; // [sp+18h] [bp+4h]@2
17
18 v1 = (int *)a1;
19 v2 = *(_DWORD *)a1;
20 LOBYTE(v2) = 1;
21 v3 = (_BYTE *)(a1 + 5)
22 *(DWORD *)a1 = v2;
23 *(v3 - 1) = 3;
24 *v3 = 1;
25 sub_100016B0(a1 + 6, 32);
26 v4 = time(0);
27 *(DWORD *)(a1 + 6) = htonl(v4, (unsigned __int64)v4 >> 32, 4);
28 *(BYTE *)(a1 + 38) = 0;
29 v5 = (_WORD *)(a1 + 39);
30 v6 = 0;
31 v7 = 6 * (rand() % 5 + 2);
32 if ( v7 > 0 )
33 {
34     v15 = (_WORD *)(a1 + 41);
35     while ( 1 )
36     {
37         v8 = rand() % 0x4Bu;
38         v9 = 0;
39         v14 = v8;
40         if ( v6 > 0 )
41             break;
42 LABEL_8:
43         if ( v8 == -1 )
44             goto LABEL_10;
45         LOWORD(v8) = table_03_00_04_00[v8];
46         *v15 = htons(v8);
47 LABEL_11:
48         ++v6;
49         ++v15;
50         if ( v6 >= v7 )
51             goto LABEL_12;
52     }
53     v10 = table_03_00_04_00[v8];
54     v11 = v5 + 1;

00004BF7 sub_10004BA0:30
766d7d591b9ec1204518723a1e5940fd6ac777f606ed64e731fd91b0b4c3d9fc
WORD __cdecl sub_10004A30(_DWORD, _DWORD, char);
WORD __cdecl sub_10004D00(_DWORD);
```

Symantec.Contopee / Lazarus Group



North Korea?





Binary Reverse Engineering

Formats of compiled software

- **ELF** (Linux & UNIX like)
- **Mach-O** (OSX)
- **PE** (Windows)
- **Class** (Java bytecode)
- **DEX** (Android - Dalvik bytecode)
- **PYC** (Python bytecode)
- ...



REQUIRED SKILLS



Skills

- **Debugging** (GDB, WinDbg, OllyDbg)
- **Assembly** (x86, x64, ARM, MIPS and many others)
- **Programming** (C, C++, Java, Python, Ruby, etc)
- **Software architecture**
- **Logic, math, crypto, protocols, networks**
- **Don't giving up**



Awesome tools

- **Disassemblers**
- **Debuggers**
- **Decompilers**
- **Patchers**



Disassemblers

These programs translate machine code to assembly.

```
main:
080483e4    push    ebp
080483e5    mov     ebp, esp
080483e7    sub     esp, 0x18
080483ea    and     esp, 0xfffffffff0
080483ed    mov     eax, 0x0
080483f2    add     eax, 0xf
080483f5    add     eax, 0xf
080483f8    shr     eax, 0x4
080483fb    shl     eax, 0x4
080483fe    sub     esp, eax
08048400    mov     dword [ss:esp+0x18+var_18], 0x8048528 ; argument "format" for method j_printf
08048407    call    j_printf
0804840c    mov     dword [ss:esp+0x18+var_18], 0x8048541 ; argument "format" for method j_printf
08048413    call    j_printf
08048418    lea     eax, dword [ss:ebp+var_4]
0804841b    mov     dword [ss:esp+0x18+var_14], eax
0804841f    mov     dword [ss:esp+0x18+var_18], 0x804854c ; argument "format" for method j_scanf
08048426    call    j_scanf
0804842b    cmp     dword [ss:ebp+var_4], 0x149a
08048432    je      0x8048442

08048434    mov     dword [ss:esp+0x18+var_18], 0x804854f ; argument "format" for method j_printf
0804843b    call    j_printf
08048440    jmp     0x804844e

08048442    mov     dword [ss:esp+0x18+var_18], 0x8048562 ; "Password OK :)\n", argument "format"
08048449    call    j_printf

0804844e    mov     eax, 0x0
08048453    leave
08048454    ret
; endp
08048455    nop
```



Debuggers

These programs are used to test other programs.

Debuggers allow us to inspect memory and CPU registers, modify of variables in runtime, set breakpoints and call functions outside the program flow.

In reverse engineering they are widely used for *dynamic analysis*.

```
[-----registers-----]
RAX: 0x1
RBX: 0x0
RCX: 0x0
RDX: 0x1
RSI: 0x0
RDI: 0x1999999999999999
RBP: 0x7fffffff3d8 -> 0x0
RSP: 0x7fffffff3b0 -> 0x7fffffff4b8 -> 0x7fffffff83e ("/home/user/ctf/d-ctf/e300")
RIP: 0x555555554a9e (mov    rax,QWORD PTR [rbp-0x20])
R8 : 0x7ffff7dd4068 -> 0x7ffff7dd0d40 -> 0x7ffff7b9320e -> 0x2e2e00544d470043 ('C')
R9 : 0x7fffffff859 -> 0x4e47004141414100 ('')
R10: 0x1
R11: 0x0
R12: 0x5555555548c0 (xor    ebp,ebp)
R13: 0x7fffffff4b0 -> 0x3
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0x555555554a96:    movzx  edx,WORD PTR [rbp-0x2]
0x555555554a9a:    cmp    eax,edx
0x555555554a9c:    jne    0x555555554ab3
=> 0x555555554a9e:   mov    rax,QWORD PTR [rbp-0x20]
0x555555554aa2:    add    rax,0x10
0x555555554aa6:    mov    rax,QWORD PTR [rax]
0x555555554aa9:    mov    rdi,rax
0x555555554aac:    call   0x5555555549ef
[-----stack-----]
0000| 0x7fffffff3b0 -> 0x7fffffff4b8 -> 0x7fffffff83e ("/home/user/ctf/d-ctf/e300")
0008| 0x7fffffff3b0 -> 0x3555548c0
0016| 0x7fffffff3c0 -> 0x56116a4f
0024| 0x7fffffff3c8 -> 0x10000000000000
0032| 0x7fffffff3d0 -> 0x0
0040| 0x7fffffff3d8 -> 0x7ffff7a36ec5 (<__libc_start_main+245>:      mov    edi,eax)
0048| 0x7fffffff3e0 -> 0x0
0056| 0x7fffffff3e8 -> 0x7fffffff4b8 -> 0x7fffffff83e ("/home/user/ctf/d-ctf/e300")
[-----]
Legend: code, data, rodata, value
0x0000555555554a9e in ?? ()
gdb-peda$
```



Decompilers

These programs try to achieve the *near-impossible* task of translating compiled software to the original source code.

Sometimes, the generated code is enough to perform reversing tasks.



Hex-rays

```
signed __int64 __fastcall sub_400962(__int64 a1)
{
    signed int v2; // [sp+14h] [bp-Ch]@6
    int v3; // [sp+18h] [bp-8h]@1
    int i; // [sp+1Ch] [bp-4h]@1

    *(_BYTE *)(strlen((const char *)a1) - 1 + a1) = 0;
    v3 = 1;
    for ( i = 0; i <= dword_600F30; ++i )
    {
        if ( !*(_BYTE *)(i + a1) )
        {
            v3 = i;
            break;
        }
    }
    v2 = 0;
    if ( dword_600F30 - 1 < v3 )
    {
        while ( 1 )
        {
            i *= v3 + 1 / dword_600F30;
            if ( i * i * v3 < i )
                break;
            v2 ^= v3 * i;
        }
    }
    return sub_400616(a1, v3, v2);
}
```



Patchers

Patchers can change machine code in order to modify the software behaviour. Hex editors can also be used for patching but there are better tools for patching assembly instructions.

The screenshot shows a debugger interface with assembly code and a patch dialog box. The assembly code is as follows:

```
000000000400760    mov    eax, 0x601087 ; XREF=EntryPoint_2+1
000000000400765    push   rbp
000000000400766    sub    rax, 0x601080
00000000040076c    cmp    rax, 0xe
000000000400770    mov    rbp, rax
000000000400773
000000000400775
00000000040077a
00000000040077d
00000000040077f
000000000400780
000000000400785
000000000400787
000000000400790
000000000400791
000000000400792    nop    dword [ds:rax]
000000000400796    nop    word [cs:rax+rax]
```

A patch dialog box is open over the assembly code, containing the instruction `cmp rax, 0x0`. The dialog includes a dropdown for "CPU mode: Generic" and a button labeled "Assemble and Go Next". A green annotation `=sub_400760+19,` points to the instruction at address 0x40076c. A yellow bar at the bottom indicates the "B E G I N N I N G O F P R O C E D U R E".



Badass tools

- IDA Pro - <https://www.hex-rays.com/products/ida>
- Radare 2 - <http://rada.re>
- Hopper Disassembler - <http://www.hopperapp.com>
- binary.ninja - <https://binary.ninja>
- ODA - <http://www.onlinedisassembler.com>
- OllyDbg - <http://www.ollydbg.de>
- Linux tools: objdump, ltrace, strace, readelf, gdb
- GDB with steroids: [PwnDBG](#), [GEF](#), [PEDA](#), [GDB Init](#)
- Apktool - <https://ibotpeaches.github.io/Apktool>
- Packer inspector (PE) - <https://www.packerinspector.com>



Decompilers

- IDA Pro - <https://www.hex-rays.com/products/ida> (x86, x64, ARM, MIPS, etc)
- Hopper Disassembler - <http://www.hopperapp.com> (x86, x64, ARM)
- Retargetable Decompiler (AVG) - <https://retdec.com> (x86, ARM, MIPS, Power PC)
- JADX - <https://github.com/skylot/jadx> (DEX2Java)
- JetBrains dotPeek - <https://www.jetbrains.com/decompiler> (.NET)
- ILSpy - <http://ilspy.net> (.NET)
- uncompyle2/6 - <https://github.com/wibiti/uncompyle2> | <https://github.com/rocky/python-uncompyle6> (Python bytecode)



Static Analysis

- **Do not** execute the program
- Read the spooky assembly/decompiled code
- Inspect flow charts
- Take notes... Lots of them. Use a whiteboard if possible!
- Translate procedures into the programming language of your choice
- It's a pain in the ass to reverse obfuscated or very complex programs

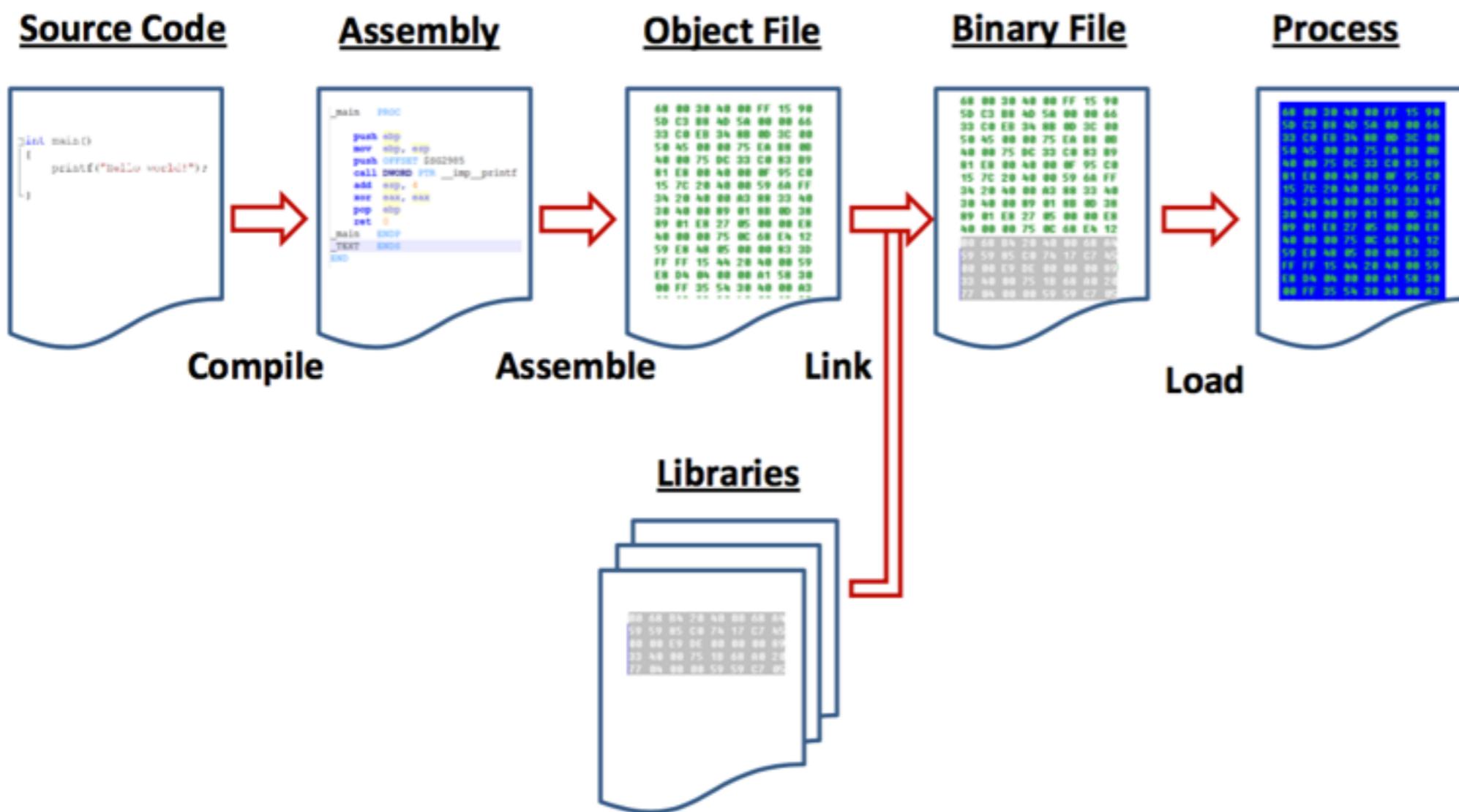


Dynamic Analysis

- **Execute** the program
- Inspect the program behaviour
- Use a debugger to inspect specific states of execution, understand the values of the CPU registers, memory segments (*stack*, *heap*, *libs*), returned values and function call arguments
- It's difficult to achieve if any anti-debugging protections exist (Some even deliberately crash common RE tools)

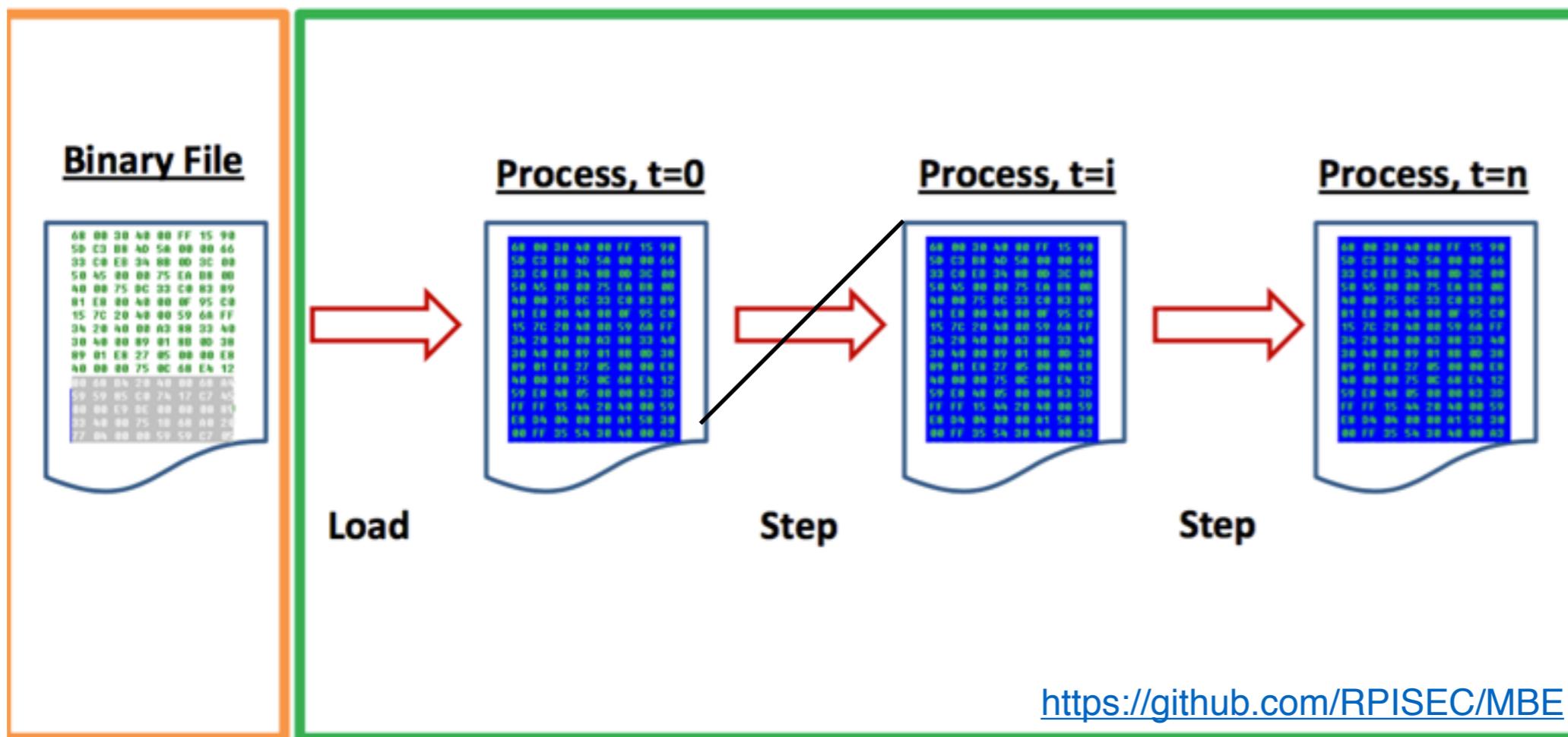


Binary RE





Binary RE



Static

Dynamic

Demo 1

Static vs Dynamic analysis

<https://goo.gl/XXoPCV>



Cracking

- This demo was a very simple cracking example
- The real stuff involves much more complex tasks (static analysis, dynamic analysis, concolic analysis, taint analysis)
- E.g. If you want to create a keygen you need to fully understand the serial number validation algorithm
- You need to be a patching ninja to remove anti-debugging protections (usually triggered in runtime)



Cracking Games

How they do it?

- In the good old days: to crack a game you just needed to patch code to bypass PC-CDROM identification checks
- Then, virtual drive tools became a thing. But games started to be compiled with additional protections: Anti-debugging, obfuscation and virtual emulators detection (DAEMON Tools/ Generic SafeDisc Emulator).

Demo 2

Let's crack a game for fun and pr...

...for educational purposes



Capture The Flag

- Reverse engineering is one of the main categories in [Security CTFs](#)
- Contestants are typically challenged to solve cracking problems
- The simplest case is just like the first demo. Find the correct input!



Cracking

Advanced techniques

```
for (i=0;i<10;i++) {  
    if (input[i] != password[i]) {  
        puts("Wrong!");  
        return;  
    }  
}  
puts("Correct!");
```

What's possibly wrong? 🤔



Cracking

Advanced techniques

- **Timing attacks**
 - When a char is correct: one more cycle is executed, i.e. more instructions
 - It's possible to launch a timing attack, char by char
 - The attack complexity is reduced from 256^{length} to $256 \times length$
 - **How can we prevent this kind of attacks? Constant time algorithms** (very nice research area)
 - Tools for local binary timing attacks: [Pin tool](#), GDB scripting



Cracking

Advanced techniques

- **Solvers**

- Serial number validation algorithms are usually composed by complex verifications, whose components are for e.g. the values of certain indexes of the serial number.

E.g. $sn[17] == sn[21] \oplus sn[34] - sn[5] \bmod (sn[14] * sn[43]^2)$

- These verifications can be translated to systems of equations, that can be easily solved by powerful magic:
[Z3 Theorem Prover](#), [Sage](#), [Maple](#), [Matlab](#)
- Z3 supports both arithmetic and bitwise operators, and custom functions as well.



Cracking

Advanced techniques - Z3

```
from z3 import *

s = Solver()

x = BitVec("x", 32)
y = BitVec("y", 32)
z = BitVec("z", 32)
a = BitVec("a", 32)
b = BitVec("b", 32)
c = BitVec("c", 32)

def shiftRight(y, c):
    return y >> c

s.add(x != 0)
s.add(x == a ^ b * z * shiftRight(y, c))

while (s.check() == sat):
    print(s.model())
    s.add(x != s.model()[x], y != s.model()[y])
|
```

Python script

```
[z = 0,
 b = 1,
 a = 33587201,
 y = 173155,
 x = 33587201,
 c = 17]
[z = 0,
 b = 1,
 a = 34635777,
 y = 173159,
 x = 34635777,
 c = 17]
[z = 0,
 b = 1,
 a = 34766849,
 y = 173158,
 x = 34766849,
 c = 17]
[z = 0,
 b = 1,
 a = 33718273,
 y = 238694,
 x = 33718273,
 c = 17]
```

Solutions



What about the future?



Predicting the RE future using **Naive Mayes**



Let's get to the powerful stuff



CG
CYBER



CG

CG
CYBER



HEARTBLEED



DARPA CGC

- A very important mark in the history of infosec
- It was the “first-ever all-machine hacking tournament”
- These machines were able to automatically find and patch vulnerabilities in binaries
- The [Mechanical Phish](#) project, from the Shellphish CTF team, was able to identify vulnerabilities using both **fuzzing** and **symbolic execution** techniques.
It's open source 😎



DARPA CGC

Mechanical Phish - Driller

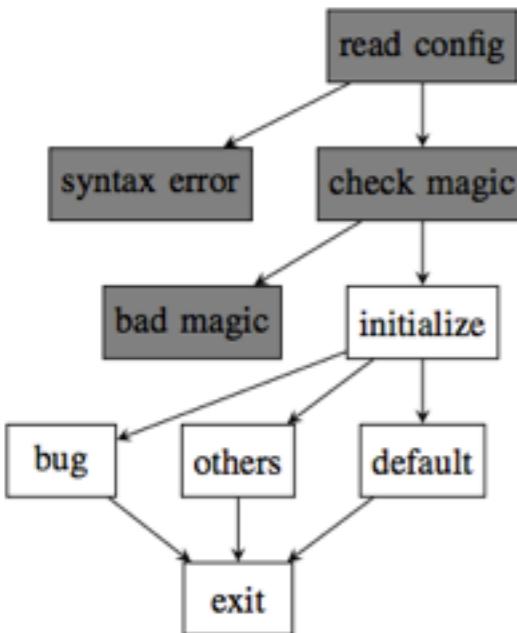


Fig. 1. The nodes initially found by the fuzzer.

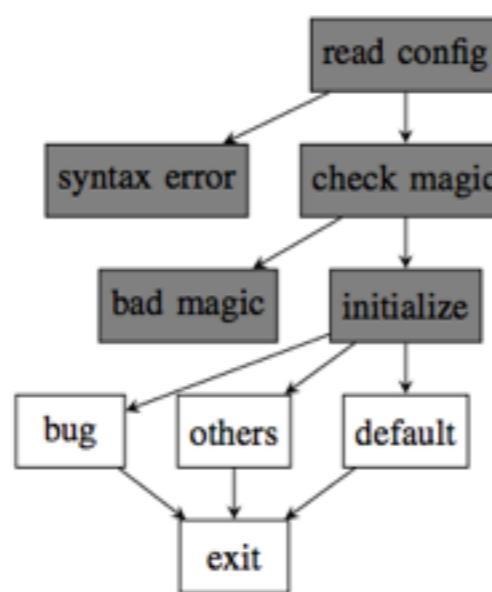


Fig. 2. The nodes found by the first invocation of concolic execution.

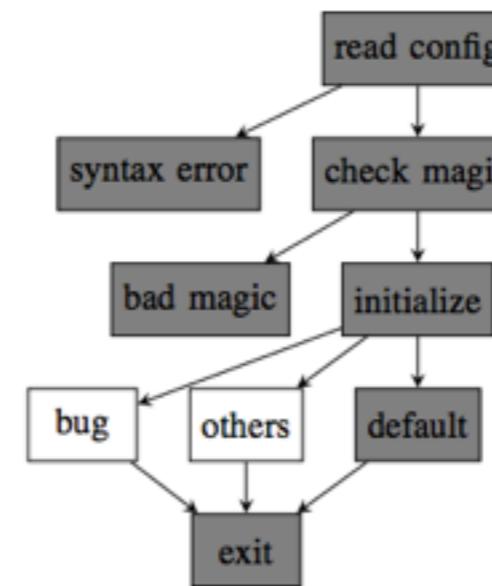


Fig. 3. The nodes found by the fuzzer, supplemented with the result of the first Driller run.

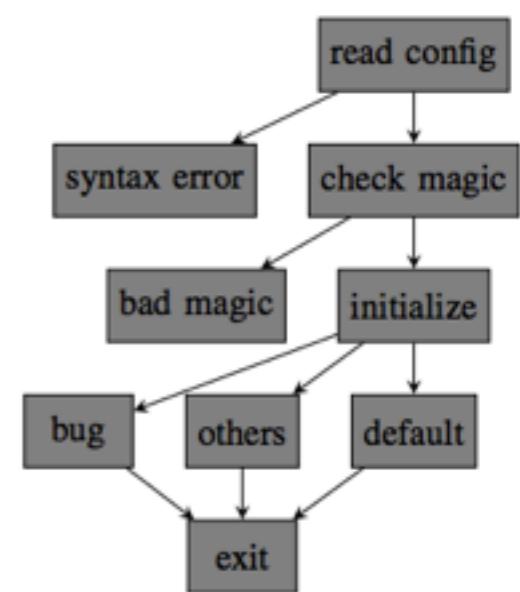


Fig. 4. The nodes found by the second invocation of concolic execution.



DARPA CGC

Mechanical Phish - ANGR

- [ANGR](#) is a very powerful binary analysis framework. It was implemented mostly by the Shellphish team and is one of the main components of Driller
- It's one of the most powerful open source software solutions to perform reversing/cracking tasks
- We can easily accomplish *control-flow* analysis, i.e., realize the damn conditions that make the program reach a specific state of execution
- First, it translates the binary in [VEX Intermediate Representation](#). Then, simulates instructions in a simulation engine -> symbolic execution: [SimuVEX](#)
- Finally, they use a custom Z3 wrapper. It is called [claripy](#): “*a abstracted constraint-solving wrapper*”



Demo 3 - Angr

<https://goo.gl/42T4mi>



Manticore

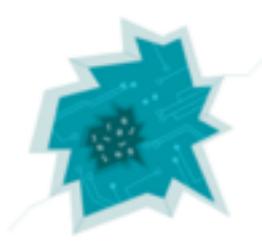
- **Input Generation:** Manticore automatically generates inputs that trigger unique code paths
- **Crash Discovery:** Manticore discovers inputs that crash programs via memory safety violations
- **Execution Tracing:** Manticore records an instruction-level trace of execution for each generated input
- **Programmatic Interface:** Manticore exposes programmatic access to its analysis engine via a Python API



Ponce

IDA plugin contest - 2016

- **Taint analysis:** this mode is used to easily track “where” a user input occurs inside a program and observe all the propagations related with the given input
- **Symbolic analysis:** in this mode, the plugin maintains a symbolic state of registers and memory at each step in a binary’s execution path, allowing the user to solve user-controlled conditions to do manually guided execution



IDA - crackme_xor.idb (crackme_xor.exe) C:\Users\Administrator\Desktop\crackme_xor.idb

File Edit Jump Search View Debugger Options Windows Help

Local Win32 debugger

Library function Data Regular function Unexplored Instruction External symbol

Functions window IDA View-A Hex View-1 Structures Enums Imports Exports

Function name

- sub_401005
- _main
- sub_401020
- _main_0
- __get_printf_count_outpu
- __printf_s_l
- __set_printf_count_outpu
- _printf
- _printf_s
- __tmainCRTStartup
- _fast_error_exit
- start
- _invoke_watson(ushort const
- _call_reportfault
- sub_401577
- _invalid_parameter
- _invalid_parameter_noinfo
- _invoke_watson
- sub_4016C4

Line 4 of 584

Graph overview

Debug application setup: win32

Application: C:\Users\Administrator\Desktop\crackme_xor.exe
Input file: C:\Users\Administrator\Desktop\crackme_xor.exe
Directory: C:\Users\Administrator\Desktop
Parameters: aaaaaa
Hostname: Port: 23946
Password:
 Save network settings as default

OK Cancel Help

var_4= dword ptr -4
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
push ebp
mov ebp, esp

cmp [ebp+var_4], 0
jnz short loc_4010D3

00000490 00401090: _main_0 (Synchronized with Hex View-1)

Output window

```
7D850000: loaded C:\Windows\syswow64\KernelBase.dll
PDBSRC: loading symbols for 'C:\Users\Administrator\Desktop\crackme_xor.exe'...
PDB: using DIA dll "C:\Program Files (x86)\Common Files\Microsoft Shared\VC\msdia90.dll"
PDB: DIA interface version 9.0
Debugger: process has exited (exit code -1)
```

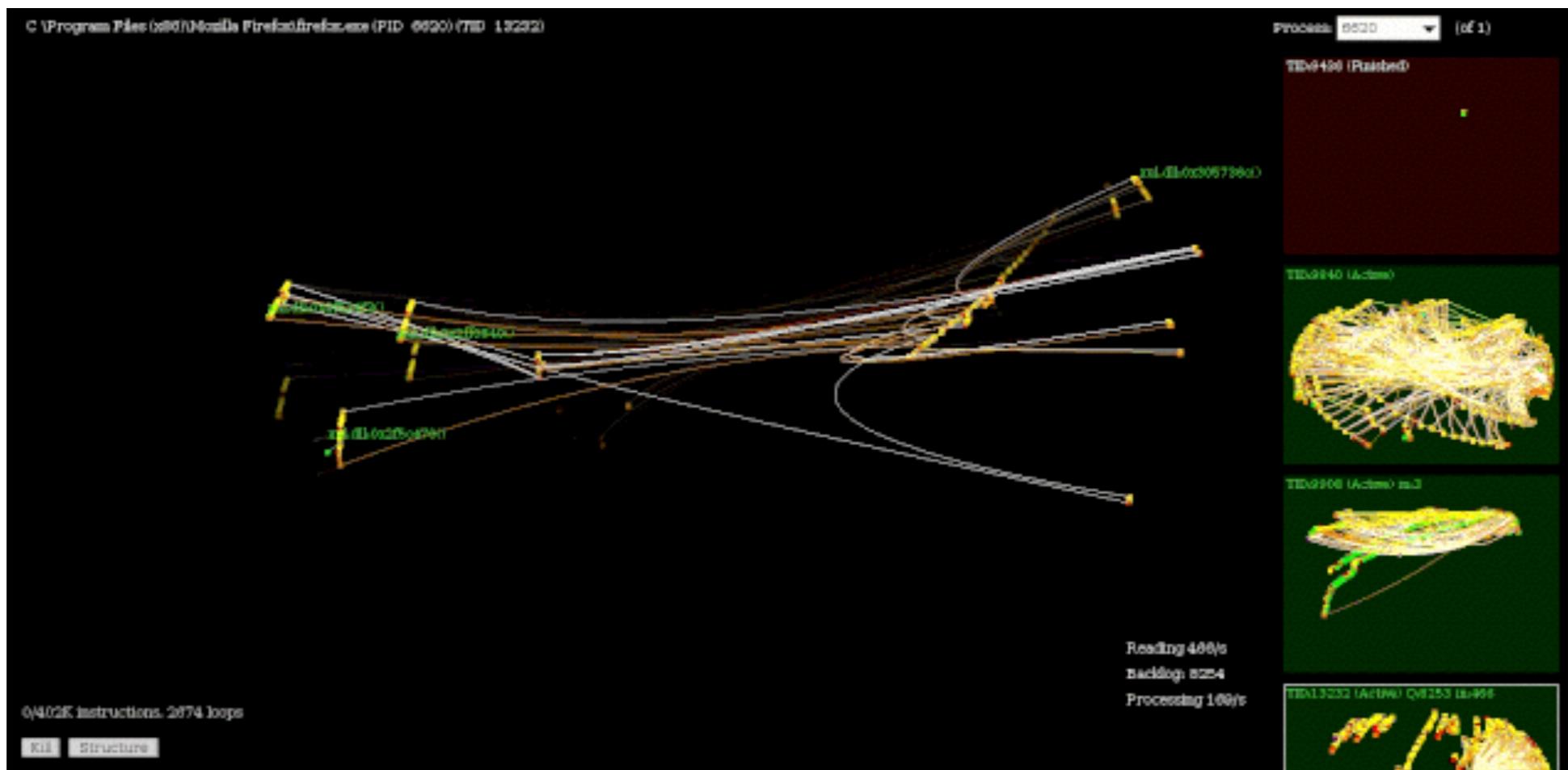
Python

AU: idle Down Disk: 139MB



RGAT

An instruction trace visualisation tool





Useful links to fry your brain

(Over 1337 °C)

- **Chill**

- <https://github.com/RPISEC/MBE> (lectures 2 and 3)
- Reddit
 - <https://reddit.com/r/reverseengineering>
 - <https://reddit.com/r/netsec>

- **Practice**

- <http://reversing.kr>
- <https://ringzer0team.com>
- <http://crackmes.de>
- <https://ctftime.org> (Read CTF writeups and try to solve available challenges)
- **Play CTFs:** alone or create/join a team



THE END



Security Through Obscurity

André Baptista
[@0xACB](https://twitter.com/0xACB)