All Software is Open Source

AN INTRODUCTION TO

REVERSE ENGINEERING

DMITRIY BERYOZA

About me

Dmitriy Beryoza

@0xd13a on HackFest Discord/Twitter

https://www.linkedin.com/in/beryozad

- Senior Security Researcher at *Vectra AI*
- Prior to that - Researcher/Pentester with X-Force Ethical Hacking Team at *IBM Security*
- 25+ years in software design and development
- Ph.D. in Computer Science, CEH, OSCP
- *Interests*: reverse engineering, binary exploitation, secure software development, cloud and network threat detection, and CTF competitions
- Live in Ottawa, originally from Russia *(cue Russian hacker joke)*

Agenda

- What is Reverse Engineering
- Applications of RE
- What Can You Reverse and Why?
- Legal Issues
- Categories of RE
- Code Examples
- Tools
- Common Obstacles
- How to Practice
- General Strategies
- Resources

What is Reverse Engineering

Reverse Engineering (or *reversing*)- process of creating a blueprint of an object by analyzing it

- Must be done because the original design is not available or intentionally withheld
- Humans have been doing RE forever
    - Many inventions in human history have been analyzed and copied
- All natural sciences are essentially RE
    - Nature doesn't come with a manual
- You have probably done RE in your life more than once
    - The toy you pulled apart as a kid to see what's inside
    - The clock you tried to fix (unsuccessfully)

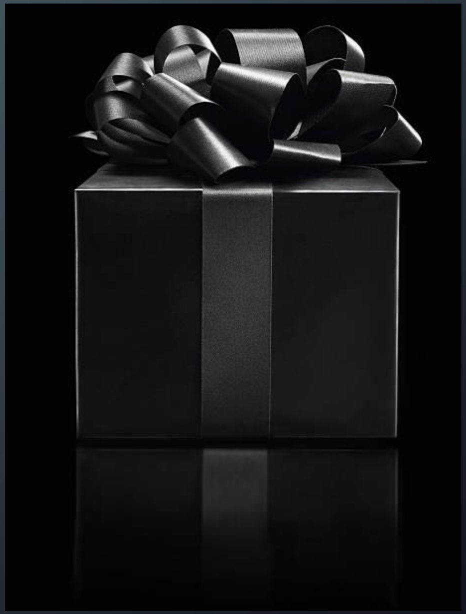
Modern Uses for RE

- Security/vulnerability research
- Pentesting
- Malware analysis
- Military/intelligence work
- Scientific research
- Commercial research/product compatibility
- Independent quality control
- Patent infringement detection
- ...and more

- We will talk about software RE
  - But that includes most hardware too - much of modern hardware is driven by software

Why Should You Do RE?

- A lot of software and hardware around us are "black boxes"
  - Documentation and architecture blueprints are scarce or withheld
- We want to be able to:
  - Analyze product for safety/quality
  - Analyze product for security/hidden backdoors
  - Recover legacy knowledge
  - Build compatible products
  - Patching behavior changes or bug fixes
- RE gives you that power
  - *"When you know assembly - all software is open source" -- Unknown*



Why Should You Do RE? (cont.)

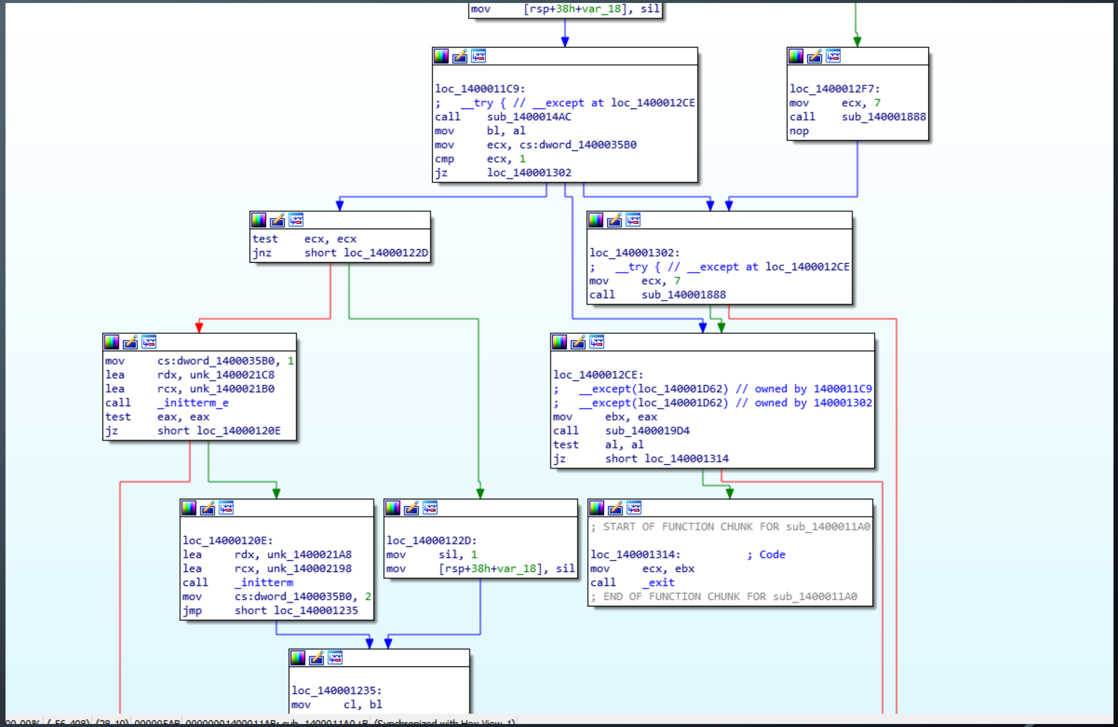
- Required skill if you are in Infosec
  - Vulnerability analysis (most CVEs require some RE)
  - Pentesting (analyzing customer software for holes)
  - Incident response (samples of ransomware, malware, phishing, C2, remote shells)

- RE can be lots of fun!
  - Gives you sense of discovery and accomplishment
  - You are overcoming challenges and discovering secrets
  - You will probably like it if you like:
    - Putting together puzzles
    - Solving crosswords
    - Geocaching
    - …

What Can You Reverse?

- Pretty much anything!

- Windows, macOS, Linux natively compiled executables
- Compiled Java, .Net, Python portable code
- iOS/Android apps
- Minified/obfuscated JavaScript and WASM
- Obfuscated PowerShell
- Obfuscated Office VBA
- Compiled automation scripts (AutoIt, AutoHotKey, etc...)
- BIOS/bootloaders
- Shellcode
- Hardware firmware



```
mov     [rsp+38h+var_18], sil
```

```
loc_1400011C9:
;   __try { // __except at loc_1400012CE
call    sub_1400014AC
mov     bl, al
mov     ecx, cs:dword_1400035B0
cmp     ecx, 1
jz      loc_140001302
```

```
loc_1400012F7:
mov     ecx, 7
call    sub_140001888
nop
```

```
test    ecx, ecx
jnz     short loc_14000122D
```

```
loc_140001302:
;   __try { // __except at loc_1400012CE
mov     ecx, 7
call    sub_140001888
```

```
mov     cs:dword_1400035B0, 1
lea     rdx, unk_1400021C8
lea     rcx, unk_1400021B0
call    _initterm_e
test    eax, eax
jz      short loc_14000120E
```

```
loc_1400012CE:
;   __except(loc_140001D62) // owned by 1400011C9
;   __except(loc_140001D62) // owned by 140001302
mov     ebx, eax
call    sub_1400019D4
test    al, al
jz      short loc_140001314
```

```
loc_14000120E:
lea     rdx, unk_1400021A8
lea     rcx, unk_140002198
call    _initterm
mov     cs:dword_1400035B0, 2
jmp     short loc_140001235
```

```
loc_14000122D:
mov     sil, 1
mov     [rsp+38h+var_18], sil
```

```
; START OF FUNCTION CHUNK FOR sub_1400011A0

loc_140001314:          ; Code
mov     ecx, ebx
call    _exit
; END OF FUNCTION CHUNK FOR sub_1400011A0
```

```
loc_140001235:
mov     cl, bl
```

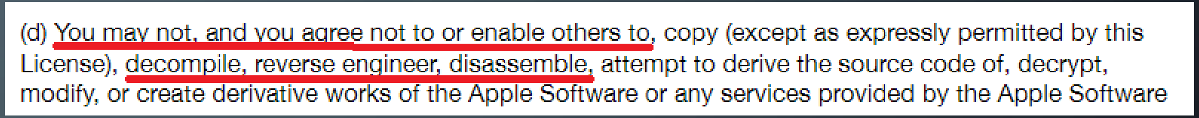
Legality of RE

Big caveat - I'm not a lawyer, and this is not legal advice

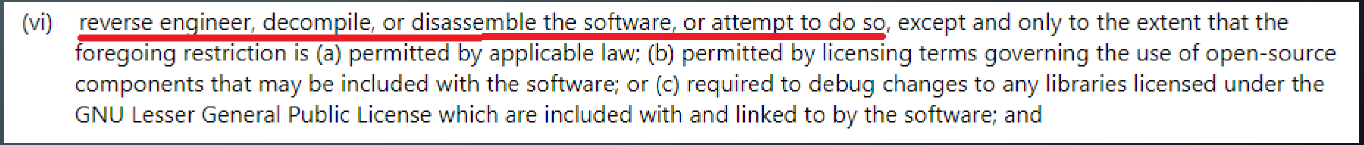
- Is RE Legal? - It depends...

- Most license agreements contain anti-RE clauses
  - Vendors want to protect their intellectual property
  - Vendors want flaws to be harder to find

- Existing laws restrict RE
  - Digital Millennium Copyright Act (DMCA)
  - Computer Fraud and Abuse Act (CFAA)
  - Copyright law
  - EU Directive 2009/24
  - ...and others


(d) You may not, and you agree not to or enable others to, copy (except as expressly permitted by this License), decompile, reverse engineer, disassemble, attempt to derive the source code of, decrypt, modify, or create derivative works of the Apple Software or any services provided by the Apple Software

Apple iOS 14 License Agreement


(vi) reverse engineer, decompile, or disassemble the software, or attempt to do so, except and only to the extent that the foregoing restriction is (a) permitted by applicable law; (b) permitted by licensing terms governing the use of open-source components that may be included with the software; or (c) required to debug changes to any libraries licensed under the GNU Lesser General Public License which are included with and linked to by the software; and

Windows 10 Retail License

Legality of RE (cont.)

- You have more protection if you own the device or software you are reversing
- The Fair Use defense in Copyright Law
    - Using copyrighted works for good-faith security research is likely fair use
- Under DMCA legal owner of the program may RE it and circumvent its protection to achieve "interoperability"
- Intent and what you do with information discovered through RE is key

- For deeper analysis see this guide by Harvard Law School & EFF: https://clinic.cyber.harvard.edu/files/2020/10/Security_Researchers_Guide-2.pdf

Legality of RE: 3 Categories

Safe to do

- RE of non-commercial code or anything you yourself produce
- RE of commercial products done in private, without publishing or taking advantage of results
- Malware analysis

Tread carefully

- Public security analysis of commercial products (bounty programs offer some protection)
- Publishing of limited reversed code/data as part of responsible vulnerability disclosure
- Publishing of tools based on RE discoveries

- Famous RE legal cases
  - Jon Lech Johansen arrested in 2000 for circumventing DVD copy protection; acquitted
  - Dmitry Sklyarov arrested after DEF CON in 2001 for circumventing Adobe protections; charges eventually dropped
- Even if what you are doing is not breaking the law, legal harassment is possible

Lawyer-up!

- Large scale disclosure of proprietary information, or profiting from it
- Building compatible or competing products

Types of RE

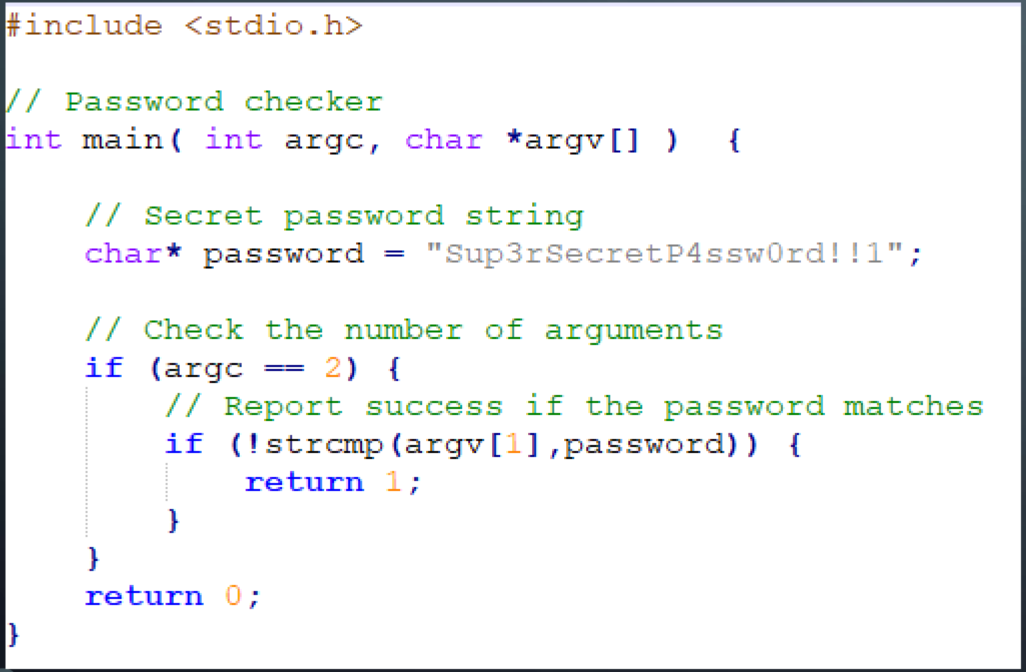
- Data/communications analysis
  - Analysis of proprietary data formats (graphics, storage, databases)
  - Analysis of proprietary network protocols
  - Requires careful experimentation and use binary editors and capture tools
- Disassembly
  - Conversion of compiled binary to human-readable machine instructions
  - Native code (x86, ARM, MIPS, ...) and p-code (JVM, .Net, Python, ...)
- Decompilation
  - Attempt to recover original code from compiled binary
  - For native code results are spotty due to code optimizations; a C-like pseudocode is usually recovered
  - P-code is higher level and results are much better

Static vs. Dynamic Analysis

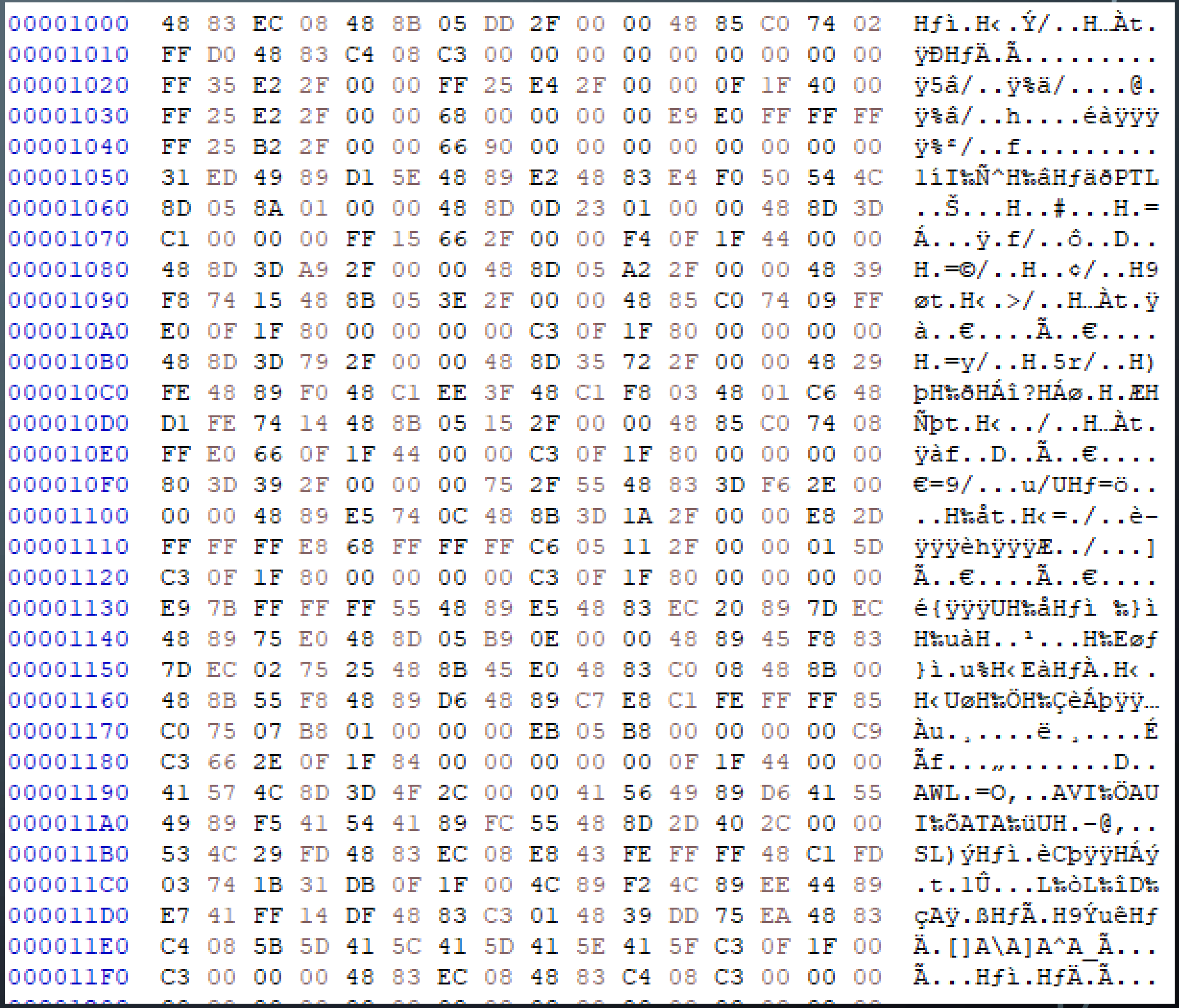
- Static involves analysis without execution
  - Helps where neither hardware nor emulators are available
  - Helps when debugging is not possible or anti-debugging measures are employed
- Dynamic is analysis though execution and debugging
  - Helps in cases of obfuscated/compressed/encrypted code and data
  - Helps confirm insight gleaned from static analysis
- Disassemblers and decompilers are often integrated with debuggers
- A combination approach often works best

Disassembly Example: C

- Consider a simple C example
- Compiled representation looks intimidating…



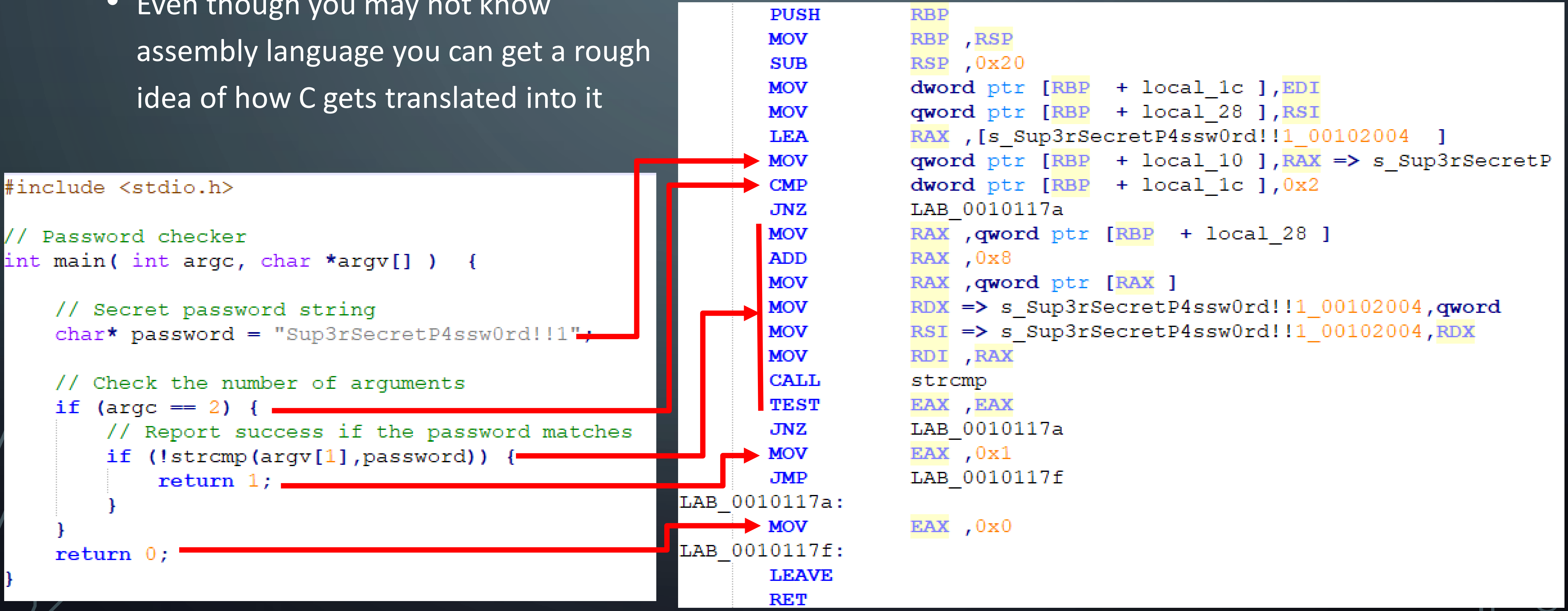
```c
#include <stdio.h>

// Password checker
int main( int argc, char *argv[] )  {

    // Secret password string
    char* password = "Sup3rSecretP4ssw0rd!!1";

    // Check the number of arguments
    if (argc == 2) {
        // Report success if the password matches
        if (!strcmp(argv[1],password)) {
            return 1;
        }
    }
    return 0;
}
```



```
00001000  48 83 EC 08 48 8B 05 DD 2F 00 00 48 85 C0 74 02  Hƒì.H‹.Ý/..H…Àt.
00001010  FF D0 48 83 C4 08 C3 00 00 00 00 00 00 00 00 00  ÿÐHƒÄ.Ã.........
00001020  FF 35 E2 2F 00 00 FF 25 E4 2F 00 00 0F 1F 40 00  ÿ5â/..ÿ%ä/....@.
00001030  FF 25 E2 2F 00 00 68 00 00 00 00 E9 E0 FF FF FF  ÿ%â/..h....éàÿÿÿ
00001040  FF 25 B2 2F 00 00 66 90 00 00 00 00 00 00 00 00  ÿ%²/..f.........
00001050  31 ED 49 89 D1 5E 48 89 E2 48 83 E4 F0 50 54 4C  1íI‰Ñ^H‰âHƒäðPTL
00001060  8D 05 8A 01 00 00 48 8D 0D 23 01 00 00 48 8D 3D  ..Š...H..#...H.=
00001070  C1 00 00 00 FF 15 66 2F 00 00 F4 0F 1F 44 00 00  Á...ÿ.f/..ô..D..
00001080  48 8D 3D A9 2F 00 00 48 8D 05 A2 2F 00 00 48 39  H.=©/..H..¢/..H9
00001090  F8 74 15 48 8B 05 3E 2F 00 00 48 85 C0 74 09 FF  øt.H‹.>/..H…Àt.ÿ
000010A0  E0 0F 1F 80 00 00 00 00 C3 0F 1F 80 00 00 00 00  à..€....Ã..€....
000010B0  48 8D 3D 79 2F 00 00 48 8D 35 72 2F 00 00 48 29  H.=y/..H.5r/..H)
000010C0  FE 48 89 F0 48 C1 EE 3F 48 C1 F8 03 48 01 C6 48  þH‰ðHÁî?HÁø.H.ÆH
000010D0  D1 FE 74 14 48 8B 05 15 2F 00 00 48 85 C0 74 08  Ñþt.H‹../..H…Àt.
000010E0  FF E0 66 0F 1F 44 00 00 C3 0F 1F 80 00 00 00 00  ÿàf..D..Ã..€....
000010F0  80 3D 39 2F 00 00 00 75 2F 55 48 83 3D F6 2E 00  €=9/...u/UHƒ=ö..
00001100  00 00 48 89 E5 74 0C 48 8B 3D 1A 2F 00 00 E8 2D  ..H‰åt.H‹=./..è-
00001110  FF FF FF E8 68 FF FF FF C6 05 11 2F 00 00 01 5D  ÿÿÿèhÿÿÿÆ../...]
00001120  C3 0F 1F 80 00 00 00 00 C3 0F 1F 80 00 00 00 00  Ã..€....Ã..€....
00001130  E9 7B FF FF FF 55 48 89 E5 48 83 EC 20 89 7D EC  é{ÿÿÿUH‰åHƒì ‰}ì
00001140  48 89 75 E0 48 8D 05 B9 0E 00 00 48 89 45 F8 83  H‰uàH..¹...H‰Eøƒ
00001150  7D EC 02 75 25 48 8B 45 E0 48 83 C0 08 48 8B 00  }ì.u%H‹EàHƒÀ.H‹.
00001160  48 8B 55 F8 48 89 D6 48 89 C7 E8 C1 FE FF FF 85  H‹UøH‰ÖH‰ÇèÁþÿÿ…
00001170  C0 75 07 B8 01 00 00 00 EB 05 B8 00 00 00 00 C9  Àu.¸....ë.¸....É
00001180  C3 66 2E 0F 1F 84 00 00 00 00 00 0F 1F 44 00 00  Ãf...„......D..
00001190  41 57 4C 8D 3D 4F 2C 00 00 41 56 49 89 D6 41 55  AWL.=O,..AVI‰ÖAU
000011A0  49 89 F5 41 54 41 89 FC 55 48 8D 2D 40 2C 00 00  I‰õATA‰üUH.-@,..
000011B0  53 4C 29 FD 48 83 EC 08 E8 43 FE FF FF 48 C1 FD  SL)ýHƒì.èCþÿÿHÁý
000011C0  03 74 1B 31 DB 0F 1F 00 4C 89 F2 4C 89 EE 44 89  .t.1Û...L‰òL‰îD‰
000011D0  E7 41 FF 14 DF 48 83 C3 01 48 39 DD 75 EA 48 83  çAÿ.ßHƒÃ.H9ÝuêHƒ
000011E0  C4 08 5B 5D 41 5C 41 5D 41 5E 41 5F C3 0F 1F 00  Ä.[]A\A]A^A_Ã...
000011F0  C3 00 00 00 48 83 EC 08 48 83 C4 08 C3 00 00 00  Ã...Hƒì.HƒÄ.Ã...
```

Disassembly Example: C (cont.)

- But if we open the executable in a disassembler things look much clearer
- Even though you may not know assembly language you can get a rough idea of how C gets translated into it

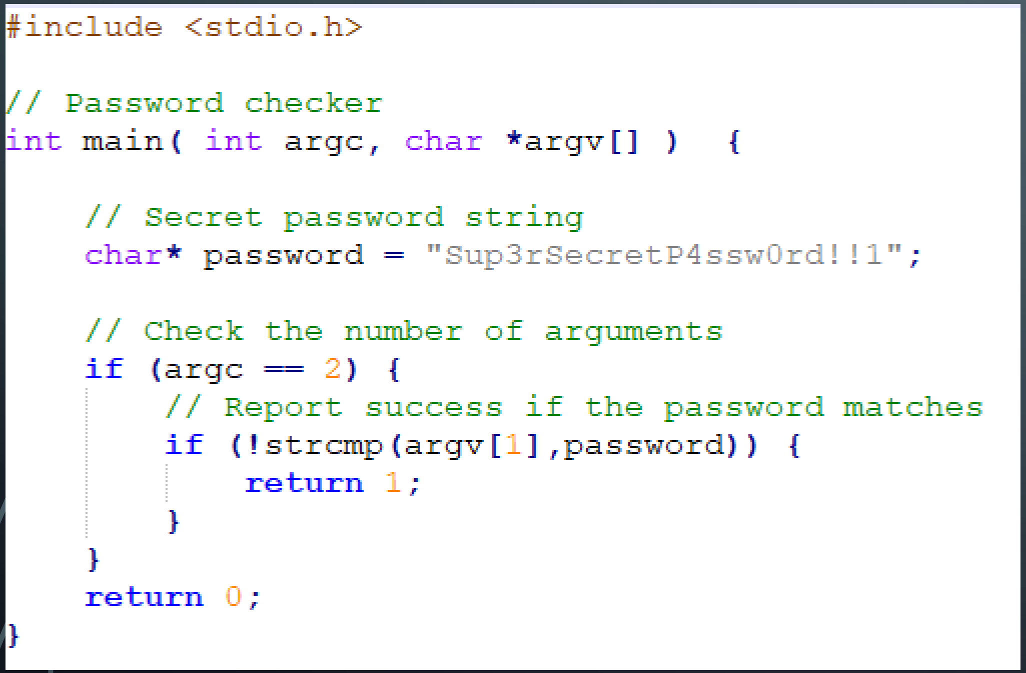


```c
#include <stdio.h>

// Password checker
int main( int argc, char *argv[] )  {

    // Secret password string
    char* password = "Sup3rSecretP4ssw0rd!!1";

    // Check the number of arguments
    if (argc == 2) {
        // Report success if the password matches
        if (!strcmp(argv[1],password)) {
            return 1;
        }
    }
    return 0;
}
```

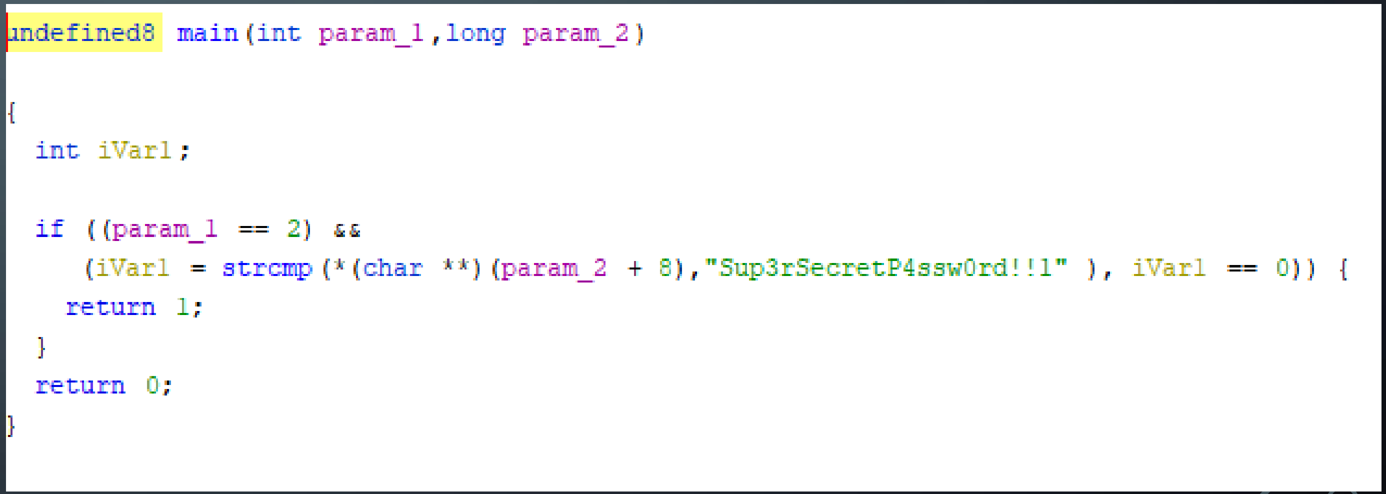
```
       PUSH        RBP
       MOV         RBP ,RSP
       SUB         RSP ,0x20
       MOV         dword ptr [RBP  + local_1c ],EDI
       MOV         qword ptr [RBP  + local_28 ],RSI
       LEA         RAX ,[s_Sup3rSecretP4ssw0rd!!1_00102004  ]
       MOV         qword ptr [RBP  + local_10 ],RAX => s_Sup3rSecretP
       CMP         dword ptr [RBP  + local_1c ],0x2
       JNZ         LAB_0010117a
       MOV         RAX ,qword ptr [RBP  + local_28 ]
       ADD         RAX ,0x8
       MOV         RAX ,qword ptr [RAX ]
       MOV         RDX => s_Sup3rSecretP4ssw0rd!!1_00102004,qword
       MOV         RSI => s_Sup3rSecretP4ssw0rd!!1_00102004,RDX
       MOV         RDI ,RAX
       CALL        strcmp
       TEST        EAX ,EAX
       JNZ         LAB_0010117a
       MOV         EAX ,0x1
       JMP         LAB_0010117f
LAB_0010117a:
       MOV         EAX ,0x0
LAB_0010117f:
       LEAVE
       RET
```

Decompilation Example: C

- With a decompiler we can get an even better picture of the code
- Quality of decompilation varies and depends on complexity of an application

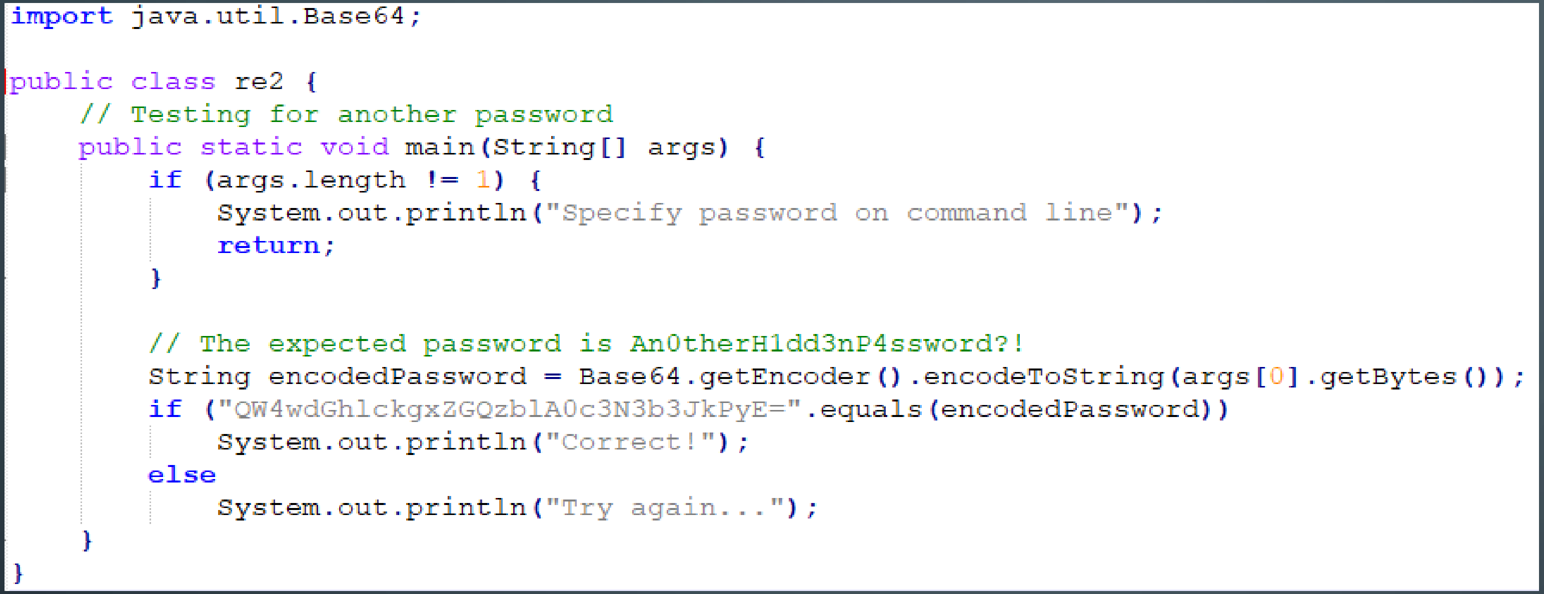


```c
#include <stdio.h>

// Password checker
int main( int argc, char *argv[] )  {

    // Secret password string
    char* password = "Sup3rSecretP4ssw0rd!!1";

    // Check the number of arguments
    if (argc == 2) {
        // Report success if the password matches
        if (!strcmp(argv[1],password)) {
            return 1;
        }
    }
    return 0;
}
```



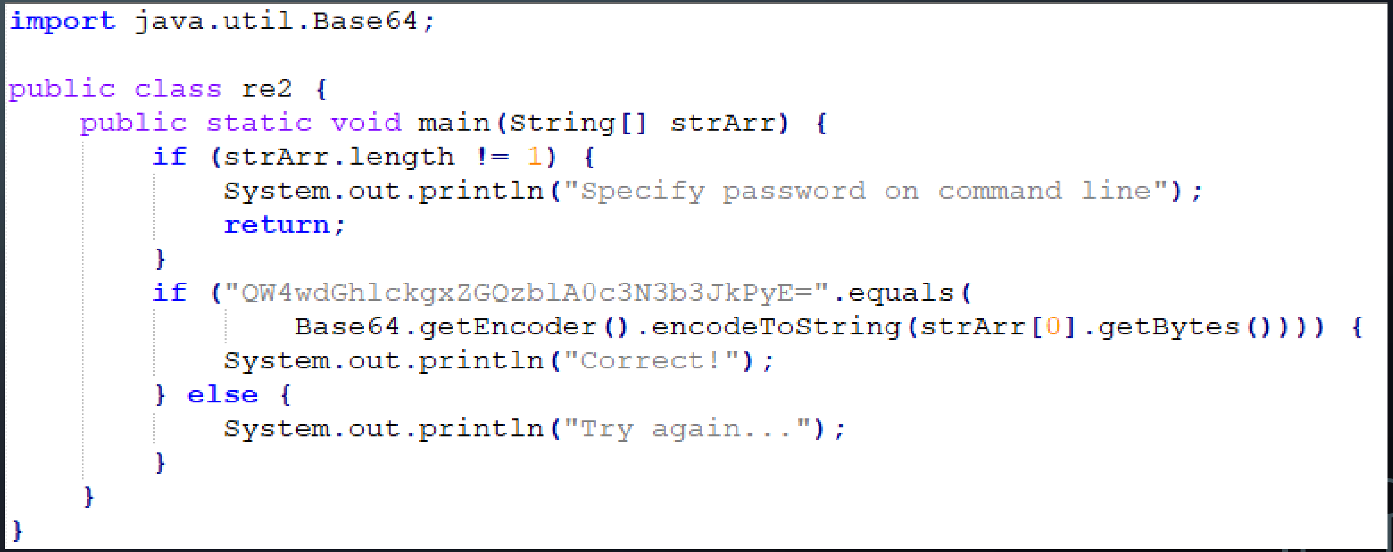
```c
undefined8 main(int param_1,long param_2)

{
  int iVar1;

  if ((param_1 == 2) &&
     (iVar1 = strcmp(*(char **)(param_2 + 8),"Sup3rSecretP4ssw0rd!!1" ), iVar1 == 0)) {
    return 1;
  }
  return 0;
}
```

Decompilation Example: Java



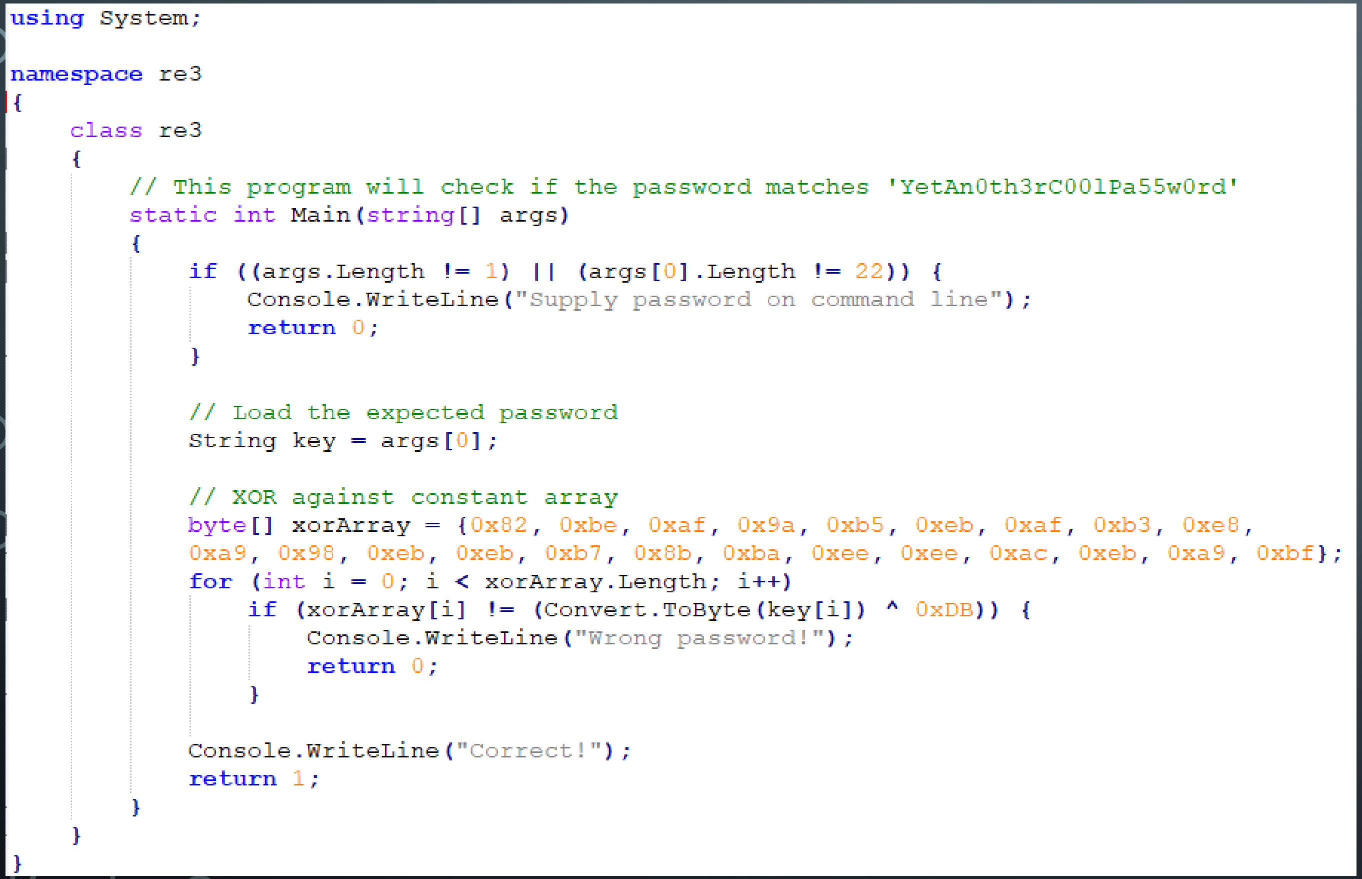
```java
import java.util.Base64;

public class re2 {
    // Testing for another password
    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Specify password on command line");
            return;
        }

        // The expected password is An0therH1dd3nP4ssword?!
        String encodedPassword = Base64.getEncoder().encodeToString(args[0].getBytes());
        if ("QW4wdGhlckgxZGQzblA0c3N3b3JkPyE=".equals(encodedPassword))
            System.out.println("Correct!");
        else
            System.out.println("Try again...");
    }
}
```

- Quality of decompilation of p-code language is much better

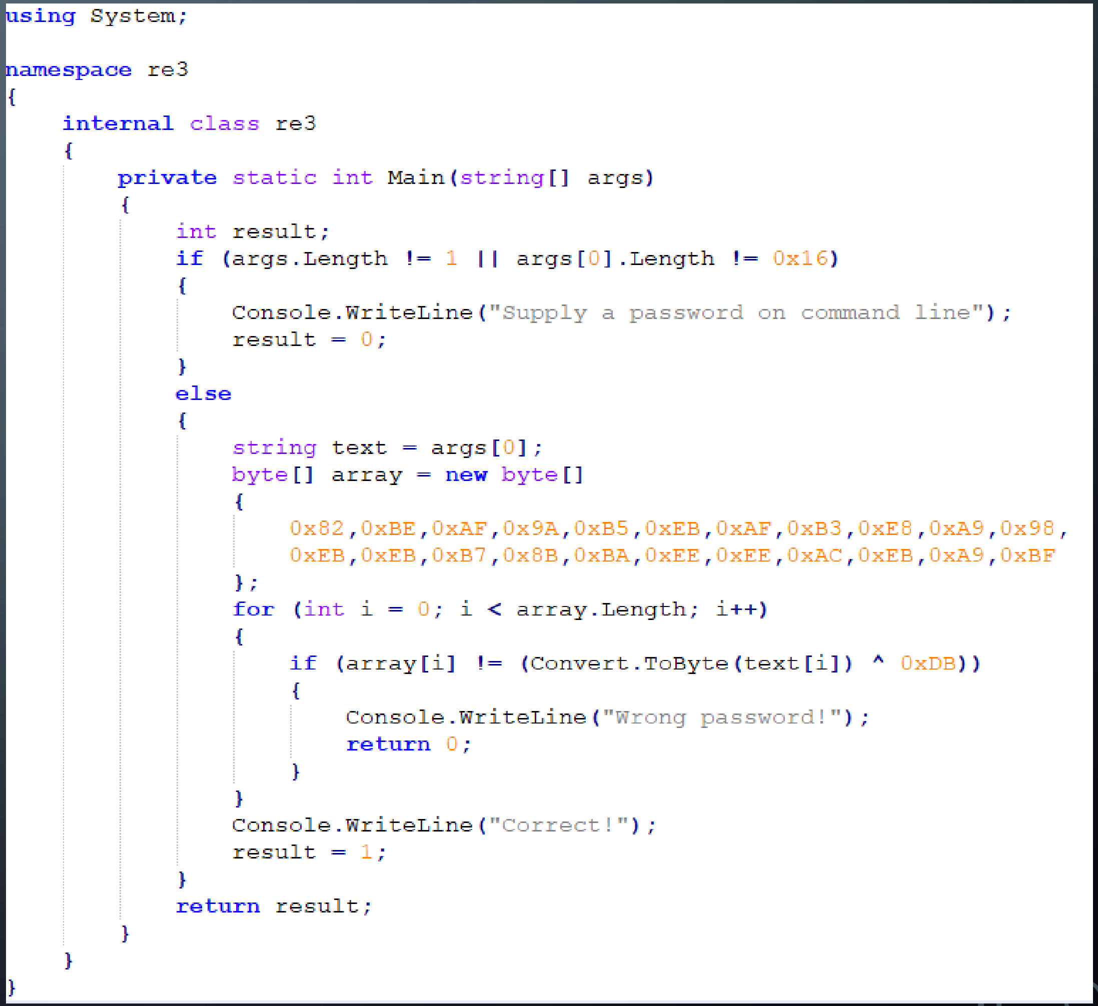


```java
import java.util.Base64;

public class re2 {
    public static void main(String[] strArr) {
        if (strArr.length != 1) {
            System.out.println("Specify password on command line");
            return;
        }
        if ("QW4wdGhlckgxZGQzblA0c3N3b3JkPyE=".equals(
                Base64.getEncoder().encodeToString(strArr[0].getBytes()))) {
            System.out.println("Correct!");
        } else {
            System.out.println("Try again...");
        }
    }
}
```

Decompilation Example: C#/.Net



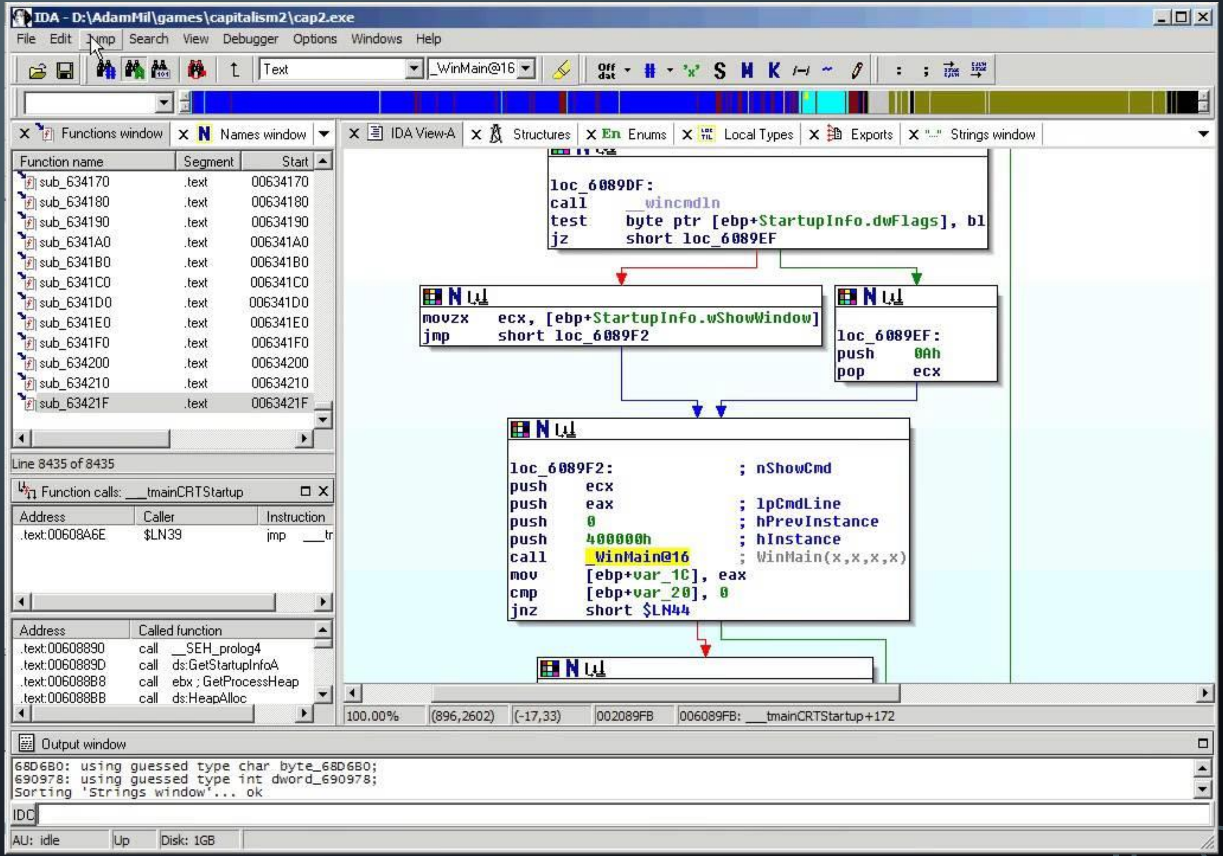
```
using System;

namespace re3
{
    class re3
    {
        // This program will check if the password matches 'YetAn0th3rC00lPa55w0rd'
        static int Main(string[] args)
        {
            if ((args.Length != 1) || (args[0].Length != 22)) {
                Console.WriteLine("Supply password on command line");
                return 0;
            }

            // Load the expected password
            String key = args[0];

            // XOR against constant array
            byte[] xorArray = {0x82, 0xbe, 0xaf, 0x9a, 0xb5, 0xeb, 0xaf, 0xb3, 0xe8,
            0xa9, 0x98, 0xeb, 0xeb, 0xb7, 0x8b, 0xba, 0xee, 0xee, 0xac, 0xeb, 0xa9, 0xbf};
            for (int i = 0; i < xorArray.Length; i++)
                if (xorArray[i] != (Convert.ToByte(key[i]) ^ 0xDB)) {
                    Console.WriteLine("Wrong password!");
                    return 0;
                }

            Console.WriteLine("Correct!");
            return 1;
        }
    }
}
```



```
using System;

namespace re3
{
    internal class re3
    {
        private static int Main(string[] args)
        {
            int result;
            if (args.Length != 1 || args[0].Length != 0x16)
            {
                Console.WriteLine("Supply a password on command line");
                result = 0;
            }
            else
            {
                string text = args[0];
                byte[] array = new byte[]
                {
                    0x82,0xBE,0xAF,0x9A,0xB5,0xEB,0xAF,0xB3,0xE8,0xA9,0x98,
                    0xEB,0xEB,0xB7,0x8B,0xBA,0xEE,0xEE,0xAC,0xEB,0xA9,0xBF
                };
                for (int i = 0; i < array.Length; i++)
                {
                    if (array[i] != (Convert.ToByte(text[i]) ^ 0xDB))
                    {
                        Console.WriteLine("Wrong password!");
                        return 0;
                    }
                }
                Console.WriteLine("Correct!");
                result = 1;
            }
            return result;
        }
    }
}
```

Tools: IDA Pro

- The ultimate RE tool
- Tons of support for many processors and environments
- Disassembler + Debugger
- +Decompiler add-on (Hex-Rays)
- Plugins, scripting, …
- Expensive - US$2K-4K
- …but there is a Free and a Home (US$365) version



Function name	Segment	Start
sub_634170	.text	00634170
sub_634180	.text	00634180
sub_634190	.text	00634190
sub_6341A0	.text	006341A0
sub_6341B0	.text	006341B0
sub_6341C0	.text	006341C0
sub_6341D0	.text	006341D0
sub_6341E0	.text	006341E0
sub_6341F0	.text	006341F0
sub_634200	.text	00634200
sub_634210	.text	00634210
sub_63421F	.text	0063421F

Line 8435 of 8435

Function calls: ___tmainCRTStartup

Address	Caller	Instruction
.text:00608A6E	$LN39	jmp ___tr

Address	Called function
.text:00608890	call __SEH_prolog4
.text:0060889D	call ds:GetStartupInfoA
.text:006088B8	call ebx ; GetProcessHeap
.text:006088BB	call ds:HeapAlloc

```
loc_6089DF:
call    __wincmdln
test    byte ptr [ebp+StartupInfo.dwFlags], bl
jz      short loc_6089EF
```

```
movzx   ecx, [ebp+StartupInfo.wShowWindow]
jmp     short loc_6089F2
```

```
loc_6089EF:
push    0Ah
pop     ecx
```

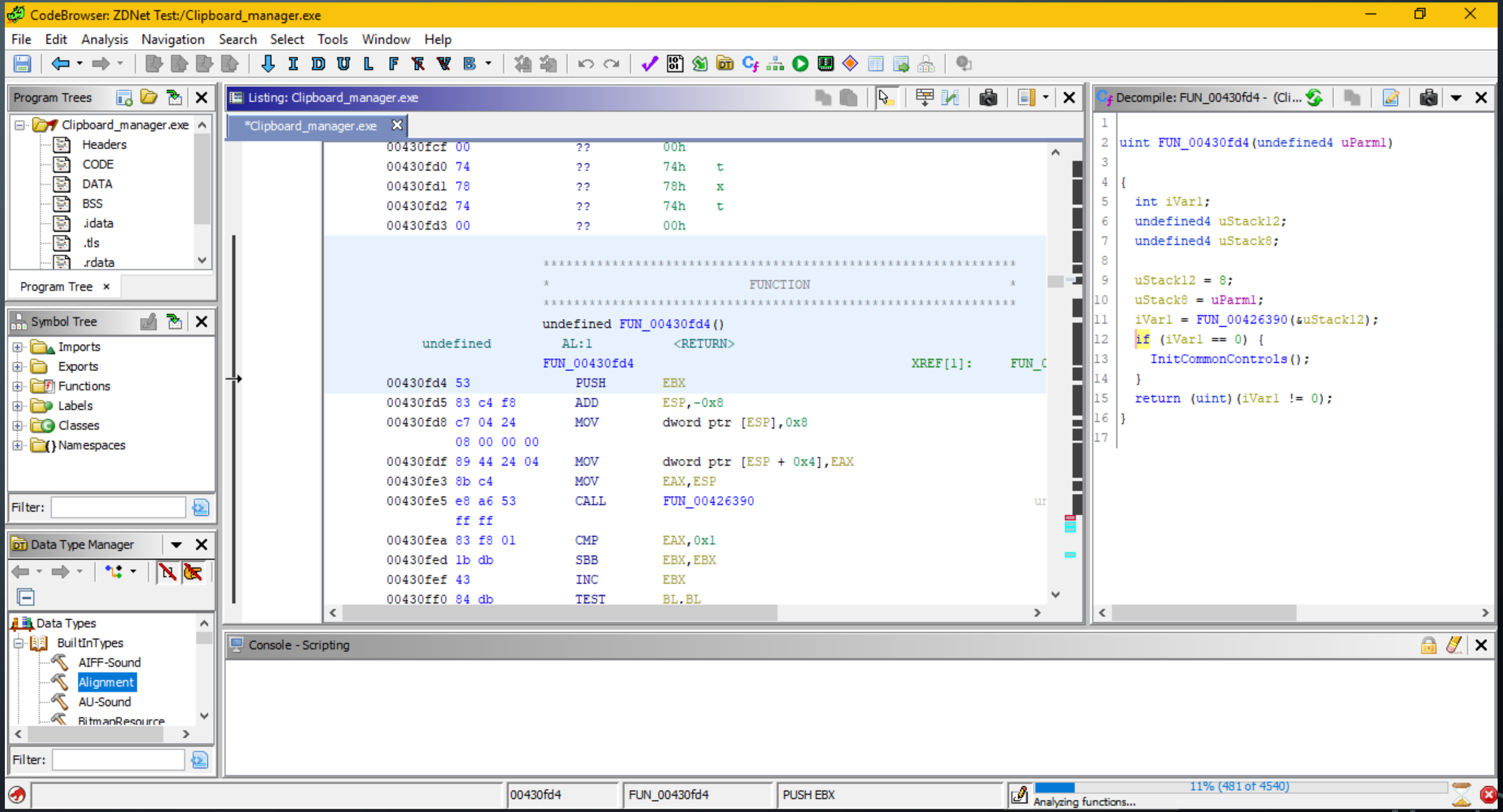
```
loc_6089F2:                     ; nShowCmd
push    ecx
push    eax                     ; lpCmdLine
push    0                       ; hPrevInstance
push    400000h                 ; hInstance
call    _WinMain@16             ; WinMain(x,x,x,x)
mov     [ebp+var_1C], eax
cmp     [ebp+var_20], 0
jnz     short $LN44
```

```
100.00%  (896,2602)  (-17,33)  002089FB  006089FB: ___tmainCRTStartup+172
```

```
68D6B0: using guessed type char byte_68D6B0;
690978: using guessed type int dword_690978;
Sorting 'Strings window'... ok
```

Tools: Ghidra

- Open-sourced by NSA (!)
- Many supported architectures
- Disassembler + Decompiler
- No debugger (some efforts under way)
- Plugins, scripting, …

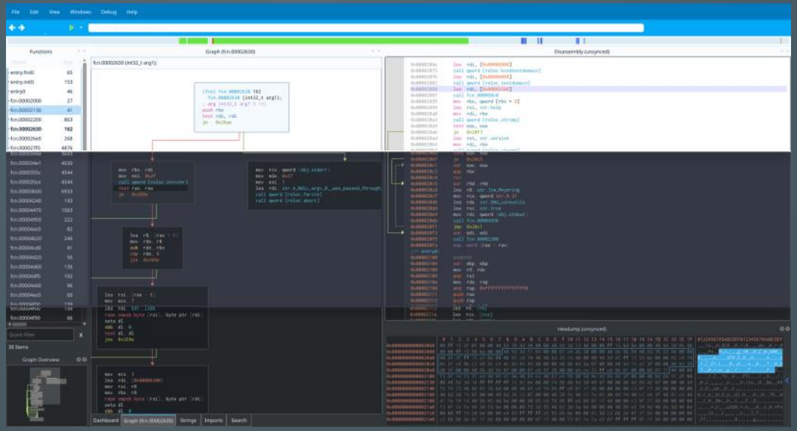


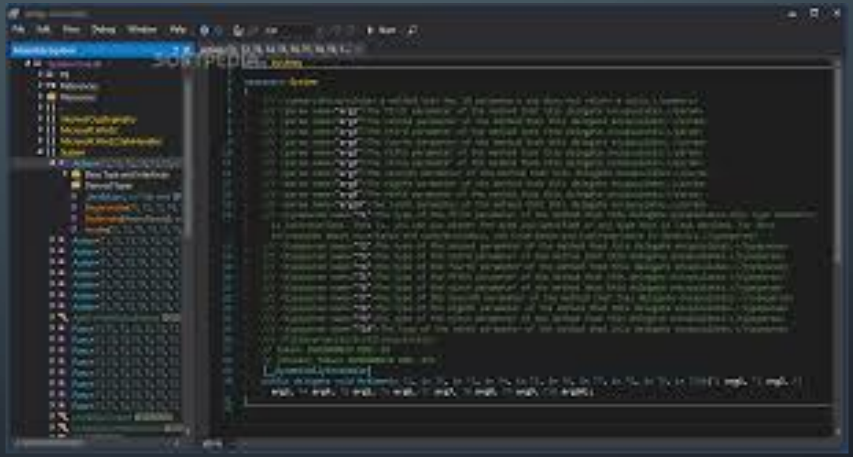
```
00430fcf 00             ??          00h
00430fd0 74             ??          74h    t
00430fd1 78             ??          78h    x
00430fd2 74             ??          74h    t
00430fd3 00             ??          00h

                *************************************************************
                *                          FUNCTION                         *
                *************************************************************
                undefined FUN_00430fd4()
     undefined       AL:1          <RETURN>
                FUN_00430fd4                                  XREF[1]:     FUN_
00430fd4 53             PUSH        EBX
00430fd5 83 c4 f8       ADD         ESP,-0x8
00430fd8 c7 04 24       MOV         dword ptr [ESP],0x8
         08 00 00 00
00430fdf 89 44 24 04    MOV         dword ptr [ESP + 0x4],EAX
00430fe3 8b c4          MOV         EAX,ESP
00430fe5 e8 a6 53       CALL        FUN_00426390
         ff ff
00430fea 83 f8 01       CMP         EAX,0x1
00430fed 1b db          SBB         EBX,EBX
00430fef 43             INC         EBX
00430ff0 84 db          TEST        BL.BL
```

```

uint FUN_00430fd4(undefined4 uParm1)

{
  int iVar1;
  undefined4 uStack12;
  undefined4 uStack8;

  uStack12 = 8;
  uStack8 = uParm1;
  iVar1 = FUN_00426390(&uStack12);
  if (iVar1 == 0) {
    InitCommonControls();
  }
  return (uint)(iVar1 != 0);
}

```

More Tools

- Complete frameworks
  - Radare2 (+ Cutter GUI)
  - Binary Ninja
- Decompilers
  - retdec
  - Snowman
- Debuggers
  - OllyDbg
  - EDB
  - x64dbg





- .Net
  - dnSpy
  - ILSpy
- Java
  - JD-GUI
  - CFR
- Python
  - uncompyle6
- WASM
  - wasmdec

- Tools are too numerous to list them all

Obstacles: Information Lost in Compilation

- As part of compilation and optimizations code is made more difficult to reverse
- Lost:
  - Code comments
  - Meaningful names of functions and variables
  - Structure of data
  - Objects (e.g. C++)
- Variables move between stack and registers
- Execution flow constructs obscured (loops, conditional statements, exception handling, ...)
- Function inlining and embedding of libraries explodes body of code

Obstacles: Counter-reversing Measures

- Software publishers know about RE and are not happy about it
- Many countermeasures are deployed to complicate analysis

- Obfuscation
  - Deliberate name obfuscation for scripting and p-code languages
  - Code minification
- Compression
  - Code and data compressed with "packers" (e.g. UPX)
- Encryption
  - Code and data decrypted at run time
  - Multiple nested levels and algorithms can be used

Obstacles: Counter-reversing Measures (cont.)

- Anti-disassembly
  - Jumps into the middle of instruction
  - False branches
  - Self-modifying code

- Anti-decompilation
  - Program flow intentionally obscured
    - Jump to register
    - Jumps through return
    - Call as a jump
  - Useless/dead code insertion
  - Extreme optimization
  - VM implementation (e.g. OISCs - Subleq, M/o/Vfuscator)

Obstacles: Counter-reversing Measures (cont.)

- Anti-debugging
  - Debugger detection
  - VM detection
  - Sporadic insertion of breakpoints
  - Use of debug handlers (e.g. PTRACE) for execution
  - Signal-based execution
  - Alarms and time-dependent code

...and this is by no means an exhaustive list

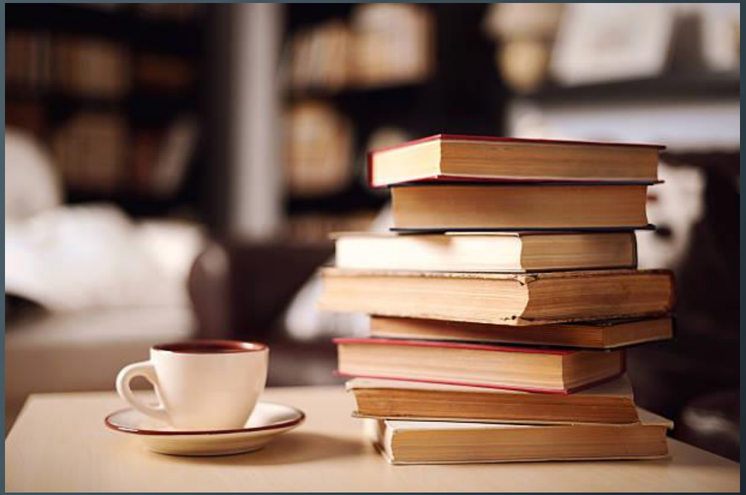
Obstacles: Counter-reversing Measures (cont.)

- It's an arms race
  - Good debuggers and VMs can hide their presence to a degree
  - Disassemblers/decompilers can overcome some counter-measures
  - Deobfuscation/decompression tools are available

- Expect these challenges in your RE work
- They can all be overcome with adequate techniques and tools
- There is nothing out there that can't be analyzed and reversed, it just a question of time and effort

Practicing: L0 - Starting Out

- The only way to get good at RE is to study and practice
- Required skills
  - Beginner ability to write in a programming language
    - Most languages are fine, but C/C++ an asset
  - Basic knowledge of target computer architecture
    - Processor registers, machine language instructions, memory organization
  - General idea of compiler operation
    - It helps to understand how high-level language is translated into low-level (because you will be doing the reverse)
  - Attention to detail and patience
- You can learn as you go, but need basics to enjoy it and to make good progress



Practicing: L1 - Do-it-yourself Code Samples

- Write and compile a few primitive programs that use different aspects of the programming language
- Open compiled programs in a few decompilers and disassemblers
  - Study different tool features
  - Look at how compiled representation maps to original code
  - What code compiler adds to the executable (setup/teardown, function prologue/epilogue, library code)?
  - Try debugging
- This way you are taking one of the variables out of the equation, you know how the end result of your efforts should look like

Practicing: L2 - Crack-me's

- These are challenges designed by someone else
- Typically small applications that are hiding a secret ("flag")
- Go at your own pace

- Great resource - We Chall (http://www.wechall.net/)
- Directory of challenge sites (60+)
- Global scoreboard to gamify experience

- Shout-out to RingZero Online CTF

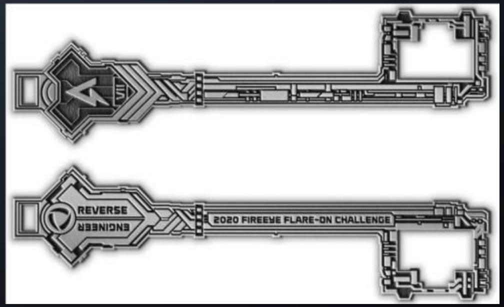
Practicing: L3 - CTFs

- Online and in-person competitions
- All levels (from beginner to pro)
- Occur all over the world regularly
- RE challenges are present in almost every CTF
- Typically 48-hour events
- Adds a time pressure to the experience (puts learning in over-drive! :))

- Recommended resource - CTF Time (https://ctftime.org/)
- A directory of online (and sometimes offline) CTFs
- Global scoreboard

Practicing: L4 - Flare-On

- Annual competition created by FireEye (http://flare-on.com/)
- The toughest RE competition in the world *(as far as I know - please let me know if you know of something better)*
- ~12 challenges of increasing difficulty
- 40 days to complete them
- Windows, macOS, Linux, scripting, mobile, embedded, esoteric…
- Many challenges are based on real malware samples or malware techniques
- Finalists get a prize



General RE Strategies

- Make it an enjoyable experience
  - Requires a lot of time, patience and attention to detail
  - Make sure it's something you are motivated to do and enjoy
    - Solving a puzzle, finding a secret, trying to discover a bug...

- Keep your eyes on the prize
  - Fully understanding even an average application is unrealistic due to size
    - Simple application like Windows Notepad has ~13K machine instructions
  - Even if you had fully commented original source code it would take days and weeks
  - Focus on the information are looking for (algorithm, piece of data) and expand your understanding around that

General RE Strategies (cont.)

- Combine static and dynamic analysis
  - Actively use debugger for analysis - analyzing by hand is often too labor-intensive
  - Make the application do the heavy lifting (e.g. to bypass encryption and obfuscation algorithms)
  - Bypass problematic code, jump around
  - Make static analysis simpler by scripting some of the reversing work (e.g. decoding/decrypting data)

- Scripting for the win
  - Reversing requires a lot of processing
  - Take advantage of scripting to automate tasks
  - Use scripting available within tools

General RE Strategies (cont.)

- Look for the signs
  - Magic numbers
  - Common algorithms (hashing, encryption, ...)
  - Systems calls and library functions
  - There are tools and plugins to help
- Document, document, document
  - A lot of info is lost in compilation
  - Body of code is usually very large, and details are easy to forget
  - As you analyze the code:
    - Add names to functions and variables
    - Restore data structures and objects
    - Comment important code blocks
  - Rinse and repeat

Resources

- Lots of blogs, articles, writeups online
  - Google is your friend

- Conferences
  - REcon (held in Montreal and Brussels) https://recon.cx/

- Getting help online
  - https://www.reddit.com/r/ReverseEngineering/
  - https://reverseengineering.stackexchange.com/

Books

- Eilam, E. "Reversing: Secrets of Reverse Engineering"
- Yurichev, D. "Reverse Engineering for Beginners"
- Sikorski, M. "Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software"
- Dang, B. "Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation"
- Eagle, C. "The IDA Pro Book, 2nd Edition: The Unofficial Guide to the World's Most Popular Disassembler"

Wrap-up

- A lot of software and hardware around us are "black boxes"
- RE can help you look inside and understand how they work
- You can help make them more secure, build compatible solutions, and modify them in new ways based on RE knowledge
- Plenty of tools are available to help reverse any product out there
- As you learn more about it, I'm sure you will enjoy RE as a fun and intellectually-rewarding activity!

Q&A

@0xd13a on HackFest Discord/Twitter

https://www.linkedin.com/in/beryozad

Slides can be found at https://github.com/0xd13a/presentations

Thank you!