Lec 5: Trusting Trust

CS492E: Introduction to Software Security

Sang Kil Cha
Why Binary?
Binary vs. Source Code

Given both binary and source code of a program, which one do you need to analyze if you want to know the program is safe to run?

1. Source code
2. Binary code
Source Code is Not Always Available

- Malware
- Commercial software
- Etc.

What about open-sourced programs?
Fun Fact

Security experts often analyze binaries even though they possess source code.

Why?
Key Question

You are given the entire source code of an app, can you find all possible vulnerabilities in the app by analyzing its source code?
The answer is NO!

Ken Thompson
Reflections on Trusting Trust
CACM 1984
Trusting Trust
Trust Boundary

User program

Kernel

H/W

Syscall
Trust Boundary
Software Security = Distrusting S/W

- You cannot trust code that you did not totally create yourself
- No amount of source-level verification or scrutiny will protect you from using untrusted code!
Stage 1: Self-Reproducing Program (a.k.a. Quine)

```c
char s[ ] = {
    '\n',
    '0',
    '\n',
    '}',
    ';
    ';
    '\n',
    '\n',
    '/',
    ';
    '\n',
    (213 lines deleted)
0
};

/*
 * The string s is a
 * representation of the body
 * of this program from '0'
 * to the end.
 */

main( )
{
    int i;

    printf("char\ts[ ] = {\n");  
    for(i=0; s[i]; i++)
        printf("\t%d, \n", s[i]);
    printf("%s", s);
}
```
Stage 2: C Compiler in C

```c
... c = next( );
if(c != '\\')
    return(c);

... c = next( );
if(c != '\\')
    return(c);

... c = next( );
if(c == 'n')
    return('n');

... c = next( );
if(c == 'n')
    return('n');
```

```c
... c = next( );
if(c == 'v')
    return('v');

... c = next( );
if(c == 'v')
    return('v');
```

```c
... c = next( );
if(c == 'n')
    return('n');

... c = next( );
if(c == 'n')
    return('n');
```

```c
... c = next( );
if(c == 'n')
    return('n');
```

```c
... c = next( );
if(c == 'n')
    return('n');
```

```c
... c = next( );
if(c == 'n')
    return('n');
```

```c
... c = next( );
if(c == 'n')
    return('n');
```

```c
... c = next( );
if(c == 'n')
    return('n');
```

```c
... c = next( );
if(c == 'n')
    return('n');
```
Stage 3: Trojan Horse

```c
void compile(char *s) {
    // ...
}

void compile(char *s) {
    if(match(s, "login pattern")) {
        compile("login backdoor");
        return;
    }
    // ...
}
```
void compile(char *s)
{
   // ...
}

void compile(char *s)
{
    if(match(s, "login pattern")) {
        compile("login backdoor");
        return;
    }
    if(match(s, "compiler pattern")) {
        compile("insert the backdoor");
        return;
    }
    // ...
}
Self-Replicating Backdoor

This technique applies to any program-handling program such as an assembler, a loader, or hardware microcode, etc.
To What Extend Should We Trust?

Compiler -> Program

Compiler -> Kernel

Kernel -> CPU
What You See Is Not What You Execute*

```
#include <stdio.h>
int main (void)
{
    printf( "hi\n" );
}
```

* 2007 Gogul Balakrishnan, PhD Thesis
Binary Code Analysis is Essential

This is what we execute

Binary Code

01010101010111110101010101010101000100100100011111110101111101010100101010010010100111011010100000010101011000001000001011
Reverse Engineering

Semantics
Reverse Engineering

Read and analyze binaries and understand their semantics
Example Kernel Vulnerability

groups_per_flex = 1 << sbi->s_log_groups_per_flex;
if (groups_per_flex == 0) return 1;
flex_group_count = ... / groups_per_flex;

When overflow?
Example Kernel Vulnerability

```
groups_per_flex = 1 << sbi->s_log_groups_per_flex;
if (groups_per_flex == 0) return 1;
flex_group_count = ... / groups_per_flex;
```

On x86, $1 << 36$ is equivalent to $1 << 4 = 16$
On PPC, $1 << 36$ is 0
Binary Analysis
= Software Security
Binary Analysis is Difficult

Compile

- Idea
- Source Code
- Intermediate Representation
- Assembly Code
- Binary Code

Reverse Engineering
Why Difficult?

• Requires manual effort

• There’s no program abstraction in binary code
4C 8B 47 08  mov r8,qword ptr [rdi+8]
BA 02 00 00 00  mov edx,2
48 8B 4F 20  mov rcx,qword ptr [rdi+20h]
45 0F B7 08  movzx r9d,word ptr [r8]
E8 54 16 00 00  call 00000001400026BC
48 8B 74 24 38  mov rsi,qword ptr [rsp+38h]
8B C3  mov eax,ebx
48 8B 5C 24 30  mov rbx,qword ptr [rsp+30h]
48 83 C4 20  add rsp,20h
5F  pop rdi
C3  ret
48 8B C4  mov rax,rsp
48 89 58 08  mov qword ptr [rax+8],rbx
48 89 68 10  mov qword ptr [rax+10h],rbp
48 89 70 18  mov qword ptr [rax+18h],rsi
48 89 78 20  mov qword ptr [rax+20h],rdi
41 54  push r12
41 56  push r14
41 57  push r15
48 83 EC 40  sub rsp,40h
48 8B 9C 24 90 00  mov rbx,qword ptr [rsp+00000000000000090h]

Types?
Functions?
Variables?
...
Conclusion

• Binary analysis is necessary for software security.
• Binary analysis is difficult, but we will learn how to do it throughout this course.
• More advanced topics for binary analysis and software security
  – IS561: Binary Code Analysis and Secure Software Systems
Questions?