ENGR-E 399/599: Embedded Systems Reverse Engineering

Lecture 1: Introduction and Overview

Austin Roach
ahroach@iu.edu

January 13, 2022

Today's plan

- Course topic overview
- Course plan
- Introduction to disassemblers

What is an embedded system?

- Computing system dedicated to a specific task
- Tight coupling of software to hardware
- Often with specialized design characteristics
  - Low power
  - High reliability
  - Real-time constraints

Embedded systems examples

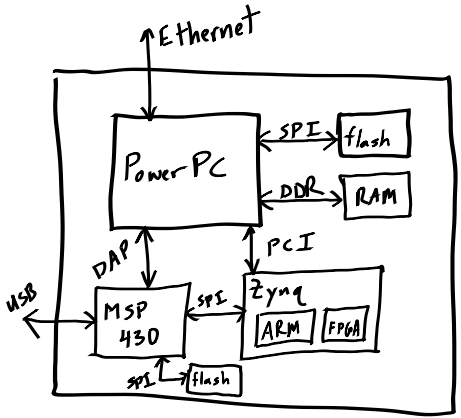
- Automotive
- Home/building automation
- Avionics
- Medical devices
- Dedicated computing/networking infrastructure
- Industrial control
- IoT

Applications to general purpose computers

- BIOS/UEFI
- Mobile operating systems (Android, iOS)
- Peripherals
- Management systems

Architectural diversity

- Diverse set of internal and external communications
- Wide variety of microcontrollers and microprocessors in use
  - AVR, MSP430, 8051, PIC, ARM, RISC-V, ...
  - x86, ARM, MIPS, PPC, SuperH, RISC-V, ...
- Functionality may be defined in hardware
  - ASICs
  - FPGAs
  - Analog circuitry or discrete logic



And all of these present in a single system!

Software diversity

- Full multi-process embedded OS
  - Linux
  - BSD
  - ...
- Multi-threaded RTOS
  - VxWorks
  - FreeRTOS
  - Mbed OS
  - ..
- 'Bare-metal' program
  - Sometimes with statically linked hardware abstraction layer

Building stuff

- Start with a requirement
- Apply available technologies to meet the requirement
- Test that the requirement was met
- Repeat to close gaps, add improvements

What is reverse engineering?

Figuring out how stuff works

- Primarily an investigative process
- Uncover technical details about the operation or composition
- **Develop a higher level understanding (why or how)**

Why would you want to reverse engineer something?

- Curiosity
- Security analysis
- Subvert content protections :-(
- Re-purpose hardware
- Interoperability
- Steal someone else's IP :-(
- IP protection

Legality

- Some reverse engineering applications have negative connotations
- Some applications are illegal
- Some applications are in an uncomfortable gray area
- EFF has a good summary of legal issues surrounding software reverse engineering
- If you are uncertain, talk to a lawyer!

Can reverse engineering be easier than forward engineering?

```
push   ebp
mov    ebp,esp
sub    esp,0x8
mov    eax,DWORD PTR [ebp+0x8]
cmp    eax,DWORD PTR [ebp+0xc]
je     804848a
mov    DWORD PTR [esp],0x80485ec
call   8048414
jmp    8048496
mov    DWORD PTR [esp],0x80485fe
call   8048414
leave
ret
```

I don't need to know all of x86. I just need to understand 9 instructions!

When is reverse engineering harder?

Forward engineering:

You make design choices according to what is familiar to you.

When is reverse engineering harder?

Forward engineering:

You make design choices according to what is familiar to you.

Reverse engineering:

You're playing on the home court of the original designer.

Name that function: Function 1

```
uint64_t FUN_001(uint64_t uParm1)
{
        uParm1 = (uParm1 & 0x5555555555555555) +
                        ((uParm1 >> 1) & 0x5555555555555555);
        uParm1 = (uParm1 & 0x3333333333333333) +
                        ((uParm1 >> 2) & 0x3333333333333333);
        uParm1 = (uParm1 & 0x0f0f0f0f0f0f0f0f) +
                        ((uParm1 >> 4) & 0x0f0f0f0f0f0f0f0f);
        uParm1 = (uParm1 & 0x00ff00ff00ff00ff) +
                        ((uParm1 >> 8) & 0x00ff00ff00ff00ff);
        uParm1 = (uParm1 & 0x0000ffff0000ffff) +
                        ((uParm1 >> 16) & 0x0000ffff0000ffff);
        uParm1 = (uParm1 & 0x00000000ffffffff) +
                        ((uParm1 >> 32) & 0x00000000ffffffff);
        return uParm1;
}
```

Name that function: Function 2

```
uint64_t FUN_002(uint64_t uParm1)
{
        uint64_t uVar1;

        uVar1 = 0;
        while (uParm1 != 0) {
                uParm1 = uParm1 & (uParm1 - 1);
                uVar1 = uVar1 + 1;
        }
        return uVar1;
}
```

Name that function: Function 3

```
uint64_t FUN_003(uint64_t uParm1)
{
        uint64_t uVar1;
        uint64_t uVar2;

        uVar2 = 0;
        for (uVar1 = 0; uVar1 < 64; uVar1++) {
                uVar2 = uVar2 + ((uParm1 >> uVar1) & 1);
        }
        return uVar2;
}
```

Name that function: Function 4

```
uint64_t FUN_004(uint64_t uParm1)
{
        uint64_t uVar1;

        uVar1 = _mm_popcnt_u64(uParm1);
        return uVar1;
}
```

Complexity of the problem

Real-world systems could have:

- Thousands of components on a PCB
- Gigabytes of stored information
- Dozens of communication interfaces

How do we find the information that we need?

Reverse-engineering as a process

1. **Define the goal**

Reverse-engineering as a process

1. **Define the goal**
2. Gather available information

Reverse-engineering as a process

1. **Define the goal**
2. Gather available information
3. Determine what access is available to you

Reverse-engineering as a process

1. **Define the goal**
2. Gather available information
3. Determine what access is available to you
4. Plan the next stage of analysis

Reverse-engineering as a process

1. **Define the goal**
2. Gather available information
3. Determine what access is available to you
4. Plan the next stage of analysis
5. Perform analysis

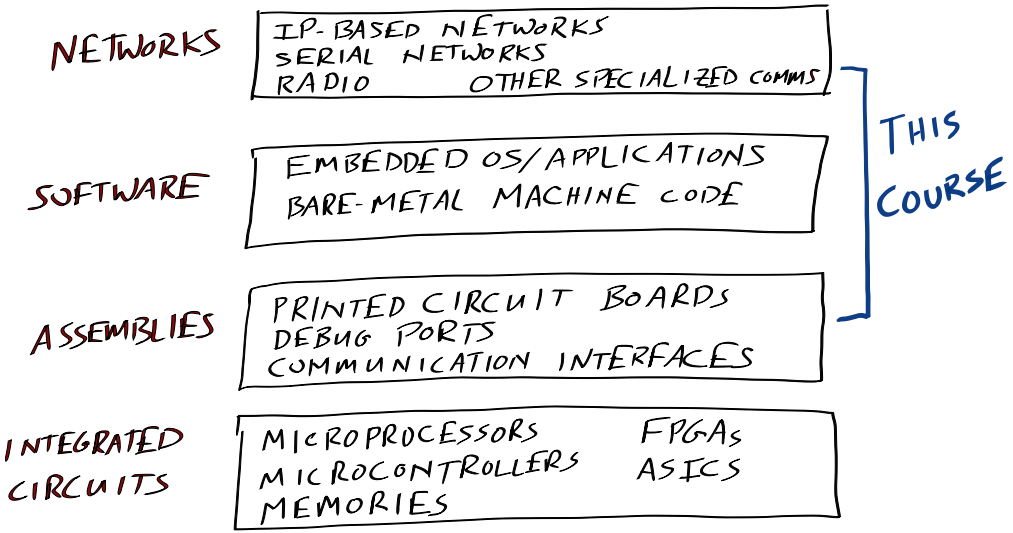
Reverse-engineering as a process

1. **Define the goal**
2. Gather available information
3. Determine what access is available to you
4. Plan the next stage of analysis
5. Perform analysis
6. Determine where you are relative to the goal

Reverse-engineering as a process

1. **Define the goal**
2. Gather available information
3. Determine what access is available to you
4. Plan the next stage of analysis
5. Perform analysis
6. Determine where you are relative to the goal
7. Return to Step 2

Course target

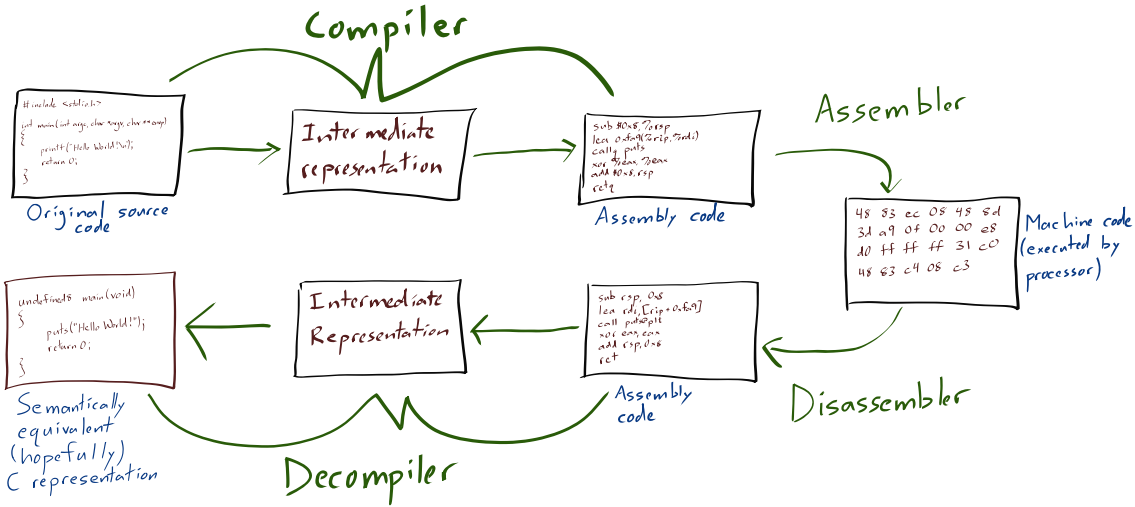


Syllabus review

https://github.iu.edu/ahroach/re_embedded_systems_2022_sp

Break

Disassemblers



Disassembler tasks

- Parse binary file structures
- Identify (or allow specification of) target architecture
- Determine (or allow specification of) entry points
- Apply disassembly algorithm of choice
- Display results
- Allow annotation
- Allow manual modification
- Other fancy features...

The world of disassemblers

- objdump/gdb/etc.
- IDA Pro
- Radare2/Rizin/Cutter
- Binary Ninja
- Ghidra

Ghidra overview

Ghidra (https://ghidra-sre.org and https://github.com/NationalSecurityAgency/ghidra):

- Open-source software reverse engineering framework
- Requires Java 11 (tested on OpenJDK)
  - Ubuntu 21.04: apt-get install openjdk-11-jdk openjdk-11-jdk-headless openjdk-11-jre openjdk-11-jre-headless
  - Windows, MacOS: https://adoptium.net/releases.html?variant=openjdk11&jvmVariant=hotspot
- Will be used for almost every project
- Documentation:
  - In-application help menu is very descriptive
  - `docs/CheatSheet.html`
  - `docs/GhidraClass/`
  - https://github.com/NationalSecurityAgency/ghidra

Ghidra demo