CS 69/169: Basics of Reverse Engineering

Sergey Bratus

Please note that this is DRAFT. Although the focus and pre-requisites of the course will not change, week-by-week topics and activities will likely go though several rounds of changes, to accommodate student backgrounds and requests.

Objectives: Develop an understanding of how systems and development tool chains are built "under the hood". Learn to read compiled binaries without available source code, to recover program logic, and to modify (de)compiled binaries. Understand the challenges of reverse engineering larger programs, and of automating reverse engineering.

Required background: The course assumes that you wrote relatively complex programs in a language like C, C++, Java, or Python, and are fluent with Unix command line tools. For this reason, CS 10 and CS 50 (or equivalent experience) are required.

You will be expected to either have or quickly develop an understanding of how a language like C is compiled down to machine code, and how that machine code executes on the actual CPU. For this reason, either CS 51 or equivalent familiarity with x86 or ARM assembly is required.

Software and Hardware: Linux or MacOS preferred. We will make heavy use of the Unix shell and tools. We will use both x86 and ARM platforms; for ARM, either a Raspberry Pi or the newer M1 MacBooks would suffice. Familiarity with ARM is a plus (but not required).

We will focus and heavily rely on free and open source tools such as LLVM, GCC, Ghidra, etc.

If you decide to use Windows, you will likely have to spend more time setting up the required tools and looking at *StackOverflow.com* to get unstuck. In short, your mileage with Windows *will* vary, and it may be easier for you to run exercises in a virtual machine such as VirtualBox or Qemu. However, if you passion is to learn about Windows reverse engineering, and you are fluent with Windows development, a special set of exercises can be arranged (but let me know in advance!).

Course Topics:

Introduction:

- Fundamentals of the x86 and ARM execution model and assembly.
- Compilation tool chain: where assembly comes from. Artifacts of compiling C and C++.
- Structure of executable binaries and libraries. How dynamic linking works.
- Linker maps and linker scripts. How embedded firmware images are built.
- Writing code in assembly: how and why.

Disassembly and decompilation:

- Producing and reading disassembly.
- Recovering symbols, program structure, and algorithmic artifacts.
- Cross-references and interactive disassembly.
- Patching binaries and relocating compiled code.
- x86, x86_64, and ARM decompilation challenges.
- Cooperative and crowd-sourced disassembly.
- Algorithms of decompilation and binary diff-ing.

Embedded and "Internet of Things" systems:

- Differences in the development process, platforms, and toolchains: how an embedded developer's world is different from PCs.
- Extracting firmware. Firmware update protocols.
- Overview of embedded ARM disassembly.

Research challenges:

- Reverse engineering as a research field: history, major successes, and challenges.

Expectations:

Skills: You will be expected to develop fluency in using tools such as debuggers, disassemblers, and decompilers to dissect executable binaries, understand their operation, and modify binaries to create intended changes in their behavior.

You will be expected to recognize standard patterns of binary implementation associated with higher-level programming languages and software design abstractions, such as system calls, function calls, dynamically loadable libraries, static and dynamic linking, implementation of virtual functions and inheritance in object-oriented programming, virtual machines for execution of bytecode, just-in-time compilation, and Application Binary Interfaces (ABIs).

Evaluation: Each student will be expected to complete several (3–5) exercises in binary reverse engineering and modification of binaries, by respective designated deadlines. Each student will be expected to propose and complete a final project. Team projects will be allowed, but will require additional approval, to make sure that individual contributions are suitable and split fairly between the team members.

All exercises will be take-home. There will be no traditional in-person timed exams or quizzes. Collaboration will be permitted on some but not all exercises. Students are expected to follow the Dartmouth Academic Honor Principles.

Credit be will 60% exercises, 40% final project.

Tentative Course Schedule:

Week 1: x86 architecture execution model and assembly fundamentals. What a compiled C program looks like in the binary.

Week 2: ARM architecture execution model and assembly fundamentals. Recognizing functions, local and global variables, and standard library calls.

Week 3: Operating system binary interfaces and artifacts. ABIs and system calls. Structure of an executable binary.

Week 4: Dynamic linking and loadable libraries. Linkable and loadable binary formats, their structure and metadata. Relocation, binary patching and rewriting.

Week 5: Compilation tool chain, compiler intermediate representations and plugins, object code, linker maps and linker scripts. Creating and dissecting embedded software and firmware images.

Week 6: Object-oriented programming languages and their compilation artifacts. How C++ function overloading, virtual functions, and inheritance are compiled, and how they can be decompiled.

Week 7: Automating reversing and decompilation. Heuristics and research algorithms for symbol, structure, and data structure recovery.

Week 8: Embedded and IoT systems specifics: key differences in development process, platforms, and toolchains. Extracting firmware, overview of embedded ARM disassembly. Firmware update protocols.

Week 9: Virtual machines and their artifacts. What VM-based interpreted languages look like in the binary. Bytecode and just-in-time compilation in modern applications and operating system mechanisms, and what they look like in the binary.

Week 10: Containers and packages. What foundations of cloud infrastructures look in the binary. Final project presentation.

Readings: The course will emphasize freely available and affordable reading material, including

- "Linux Assembly Language Programming" by Bob Neveln, available new or used from Amazon (`http://a.co/9hDjmSa`);
- "Reverse Engineering for Beginners" by Dennis Yurichev, freely available online at `https://beginners.re/`;
- "Linkers and Loaders" by John Levine, manuscript freely available online at `https://www.iecc.com/linker/`, also from Amazon in printed form (`http://a.co/613nVrl`);
- Hacker and hobbyist online magazines and journals such as *Phrack.org* and *Uninformed.org*;
- ... and more to be recommended in class.

Guest lectures: The course will include guest lectures by prominent industry reverse engineers. I strongly encourage you to attend these lectures and the informal lab discussions we'll arrange with the invited speakers at the Trust Lab. As with other forms of complex craft, reverse engineering is best learned directly from the leading practitioners

These guest lectures may occur either during the regular class meetings or the X-hours.

Research and Professional Opportunities: Dartmouth's Computer Science department prides itself on sustaining the John G. Kemeny Tradition, the vision that undergraduate students can and should be involved in state-of-the-art computing research, and should be given access to the state-of-the-art technology and the leading experts in the field to facilitate this involvement.

Accordingly, students will be encouraged to read relevant academic research paper and watch presentations from leading industry conferences. I will invite Dartmouth alumni who, while at Dartmouth, got involved in state-of-the-art systems security and reverse engineering research at the Trust Lab, presented their results at highly competitive professional conferences, and got the opportunity to collaborate with the leading InfoSec industry researchers.

For those interested, I will welcome discussion of potential Senior Honors Thesis topics, and will strive to connect these thesis projects with the state-of-the-art industry research.