Final Year Project Plan

Abstract

This project aims to utilize the tools provided by the Capstone Engine project, being a state of the art disassembly framework for binary analysis and reverse engineering, in order to produce a final deliverable of a Disassembler capable of, as a bare minimum, disassembling target binary executables. In addition to this, critical features of the software include file parsing (in order to identify each assembly instruction and what follows once the target binary has been disassembled), traversal, heuristics, data export, and visualization/control flow diagrams. The disassembler should not be ignorant towards non-executable data such as jump tables, alignment bytes etc. Additionally, the disassembler should also be able to deal with obfuscation mechanisms implemented by compilers. This potentially includes junk insertion.

The general plan is to implement a disassembler at an early phase which is capable of disassembling executables on all platforms and in every low level programming language, working on any operating system. This is a key feature of using the capstone library and Using Java for this as per the project outline therefore seems to make absolute sense. This is also sensible for myself, having done most of my previous programming in java. Once the basic disassembler has been implemented, the bulk of the work will be in using heuristics to eliminate the issue of obfuscated binaries as well as forming assembly control flow graphs by parsing the assembly produced from the capstone engine. The control flow graphs will probably be done by abstracting the control flow into a table of some sort initially and then implementing the means to visualize the control flow. Using heuristics to eliminate obfuscation will take some careful planning and extensive reading on how to actually go about this.

I chose this project because it's an opportunity to get some depth of knowledge in an area I have already briefly looked at in a couple of previous modules, being disassembly. Assembly code personally scares me, and even more so binaries. I am hoping that this project will give me a satisfactory learning experience, as it is genuinely interesting...It will in every sense be my biggest academic and practical computer science challenge ever, which I'm now really looking forward to.

Timeline

5th October 2017

Complete the Overview of Capstone early deliverable. This is a nice start to the project as it is an opportunity to research and really understand the capstone engine and its workings.

12th October 2017

Complete the report section on disassembly. This is important and will require a lot of reading to understand disassembly in more depth before diving into the project.

25th October 2017

Have completed all early deliverables, being the report on disassembly, machine code semantics and an overview of capstone. This will set the base for completing the initial disassembler and help to affirm adequate knowledge, via the required learning experience, to continue the development. Begin the initial version of the disassembler.

Friday 1st December 2017

Produce the initial disassembler without GUI implementation. Have full implementation of at least a tabular representation of the assembly control flow, as it is a critical feature. It can the be abstracted from to provide some graphical representation

Monday 4th – Friday 8th December 2017

Attend the interim review viva, having the initial version of the project ready to be described/presented.

By 10th February 2018

Hopefully have had implemented resistance against binary obfuscation into the disassembler as well as code discovery heuristics. This will be critical.

March 1st 2018

Confidently complete the Evaluation of Disassembler on Set Binaries report, Architecture and Design Patterns report, and the Legal Issues in Reverse Engineering Report

Friday 22nd March 2018

Fully implement the GUI and have ready a flawless disassembly program ready for completion between 1st march to this date, and work on any positive UX features / some added functionality if there is time.

Friday 23rd March 2018

Submit the final deliverable piece of software and report. The report will have concatenated all previously stated report-affiliated deliverables and more, as well as coming across as a professional report.

Bibliography

Benjamin Schwarz, Saumya Debray, Gregory Andrews
"Disassembly of Executable Code Revisited"
Department of Computer Science University of Arizona
This paper was useful for understanding two commonly used disassembly algorithms as well as a 'hybrid' algorithm, which can detect any incorrect disassembly. More than anything, reading this paper gave some basic understanding requied of some disassembly algorithms – the Linear sweep and Recursive traversal.
https://www2.cs.arizona.edu/~debray/Publications/disasm.pdf

Capstone Engine
"**Capstone-Engine"** overview and tutorials on capstone
The capstone engine's website really clearly outlines the specification of the Capstone disassembly framework, as it should. It is essential reading for this project, as it is centric. Understanding how capstone works is key and learning of the library's bindings to key languages is and has been important.
http://www.capstone-engine.org/documentation.html

C. Linn, S. Debray
"Obfuscation of executable code to improve resistance to static disassembly"
ACM Conference on Computer and Communications Security (CCS 2003)
This paper was easily the most useful paper for me. The paper aims to describe methods by which to increase the difficulty of statistically disassembling a program – one of the key problems that this project's disassembler aims to eliminate. By looking at methods defined in this paper, it makes it simpler to understand and therefore thwart obfuscation techniques. As far an introductory paper on reverse engineering/disassembly goes, this one was pitched at a perfect beginner's level and gradually eased into complexity. It, like "Disassembly of Executable code revisited", gave an introduction to disassembly algorithms and in a really clear way. The paper also outlines the issues surrounding disassembly and how it has gotten easier to reverse engineer programs in recent times.
https://www2.cs.arizona.edu/solar/papers/CCS2003.pdf

Susanta Nanda, Wei Li Lap-Chung Lam Tzi-cker Chiueh
"**BIRD: Binary Interpretation using Runtime Disassembly**"
Department of Computer Science University of Arizona
This paper wasn't so useful, but seeing another view of a disassembly algorithm/extensions to existing algorithms was insightful.
https://www.researchgate.net/profile/Lap_Lam/publication/4231624_BIRD_binary_interpretation_using_runtime_disassembly/links/00b49518db7561ed95000000/BIRD-binary-interpretation-using-runtime-disassembly.pdf

Risk assessment

I feel that there are two real risks associated with my program – one being errors in programming. Given that there are no servers running/need for network communications and no other programs/engines running in conjunction with the disassembler, mistakes in any code that I write will be a problem, which may cause program crashes or produce (potential quietly) unwanted results. What is useful about program crashes though is console error dumps which should easily be available upon crashes. Because of this, test driven development will be key in the development process as well as extensive defined tests of early and final deliverable programs. The program will also make use of the capstone framework's available libraries as well as other Java libraries, for example a graphical user interface etc. It is not necessarily guaranteed that external libraries will work bug-free.

The other main risk with the program is a general risk which applies to all disassemblers: no single disassembler can be flawless. Disassemblers are and quite possibly cannot be one hundred percent accurate in their results. Given that the disassembler will work on so many different platforms and the fact that disassembling on different platforms may have no single solution, this presents an obvious problem of inconsistency in results and multiple disassembly results, which can affect control flow graphs produced and any other data extracted from disassembly.

The risks of imperfect disassembly are further increased after any custom code discovery heuristics or heuristics for detecting obfuscation are devised and implemented. Perhaps my propositions of heuristics may entail incorrect disassembly or affect the result of previously thought-to-be correct disassembly results once implemented. This can cause a lot of confusion. Mitigations involve extremely carefully planned heuristics, with a lot of emphasis on theoretical solutions to ensure that they do their job as intended without affecting any other outputs of the software negatively.

A final risk which comes to mind is the testing of the disassembler on various different platforms. The capstone engine specifies compatibility with the majority of systems. Perhaps this specification may turn out to cause problems. It is a problem that I can only discover after an early version of the disassembler is produced.