Defence Research and Development Canada
Recherche et développement pour la défense Canada

DRDC | RDDC
technologysciencetechnologie

Software reverse engineering

A basic methodology

Yvan Lavoie
Alain Dessureault
Martin Salois
DRDC – Valcartier Research Centre

Defence Research and Development Canada
Reference Document
DRDC-RDDC-2018-D078
August 2018

Canada

IMPORTANT INFORMATIVE STATEMENTS

This document was reviewed for Controlled Goods by Defence Research and Development Canada (DRDC) using the Schedule to the *Defence Production Act*.

Endorsement statement: This publication has been published by the Editorial Office of Defence Research and Development Canada, an agency of the Department of National Defence of Canada. Inquiries can be sent to: Publications.DRDC-RDDC@drdc-rddc.gc.ca.

Abstract

Contrary to traditional forward engineering, software reverse engineering is a time-consuming task that not every programmer can perform after a few hours of training. Only a dedicated few top-level analysts can become proficient reverse engineers. To help managers sift through potential candidates, DRDC is often asked for an overview of the process we follow. This Reference Document presents a basic, high-level methodology that highlights this process' broad strokes.

Significance to defence and security

DRDC has been developing tools and techniques to accelerate software reverse engineering for over fifteen years. Within the current Platform-to-Assembly Secured System (PASS) 05aa project, a lot of that expertise was required. As such, clients and international collaborators have asked for training on this process. As a first step, they usually ask for a high-level overview of the process, presented in this Reference Document.

Résumé

Contrairement à l'ingénierie traditionnelle descendante. la rétro-ingénierie logicielle est une tâche très longue qui ne peut être accomplie par n'importe quel programmeur après quelques heures de formation. Seul un nombre restreint d'analystes de haut niveau et très motivés peuvent devenir de bons rétro-ingénieurs. Pour aider les gestionnaires à filtrer les candidats potentiels, RDDC se fait souvent demander un aperçu du processus que nous suivons. Ce document de référence présente à haut niveau une méthodologie de base pour illustrer les grandes lignes de ce processus.

Importance pour la défense et la sécurité

RDDC développe des outils et des techniques pour accélérer la rétro-ingénierie logicielle depuis plus de quinze ans. Dans le cadre du projet actuel Platform-to-Assembly Secured System (PASS), projet 05aa, cette expertise a été grandement requise. À cet effet, nos clients et nos partenaires internationaux ont demandé de la formation sur ce processus. À titre de première étape, ils demandent généralement un survol à haut niveau de ce processus, présenté dans ce document de référence.

Table of contents

List of figures

This page intentionally left blank.

1 Introduction

Tinkerers have been taking machines apart to understand and change them for as long as machines have been around. Software and electronics are no different, but understanding software is a complex task at the best of time. In May 2018, a non-trivial program such as Google Chrome [1] had close to 19 million lines of code, mostly written in C++, and took close to an estimated 6 000 person-years to create!

Using a conservative ratio of five to one to convert from C++ to assembly means that a reverse engineer would have close to 95 million lines of assembly to analyse to understand the whole program. This is clearly unmanageable. However, there are situations where reverse engineering is the only option, such as:

- Understanding what a malware did on your network and what information was extracted;
- Making sure that a piece of software is secure enough for the task at hand;
- Fixing legacy software for which source code has been lost; and
- Verifying that the cryptography is correctly implemented and that keys are correctly managed. In such cases, the reverse engineer must split the problem into bite-size chunks. This is more difficult than it sounds as the amount of information and strategies available is nearly endless.

This Reference Document aims to start you on this journey by presenting a basic methodology to analyze and document assembly. We propose an approach to pinpoint the parts of interest based on a given project's goals, such as finding vulnerabilities.

This is not a complete tutorial, as this would require a book. This is a high-level overview of the process, without much technical detail. Therefore, the reader should already be familiar with the following concepts:

- Good working knowledge of at least one low-level language (C/C++/Assembly); and
- Familiarity with program analysis terminology (a glossary provides some basic help).

2 Forward engineering versus reverse engineering

Figure 1 shows the basic difference between *forward* and *reverse software engineering*. *Forward engineering* is traditionally known simply as *engineering*, but we use the term to differentiate with reverse engineering.

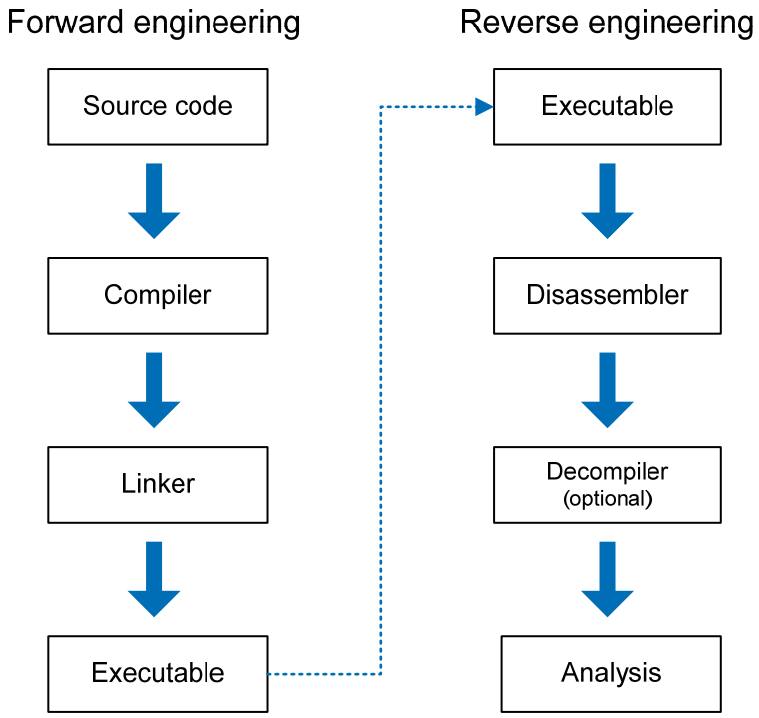

Figure 1: *Forward engineering versus reverse engineering.*

In forward engineering, a developer writes source code in a higher-level language (e.g., C/C++, Java, Python), then compiles and links it using their tool of choice (e.g., Visual Studio, gcc) to produce an executable file (e.g., .exe, .dll) that contains only zeroes and ones–the binary file. This is a lossy process as a lot of information used by the linker and the compiler is discarded because they are not necessary for the executable to run, such as variable and function names, high-level structures (e.g., for/while/do/switch loops) and data structures.

In reverse engineering, as the name implies, the process is reversed. A tool, such as IDA Pro [2], is used to produce assembly code back from the binary executable. For various reasons, this is not a perfect process and many errors can occur. Optionally, in some cases, a decompiler (internal [3] or external to IDA Pro) can be used to produce a higher-level abstraction closer to C/C++, but this is an even more imperfect process because of the information lost in translation. It is the job of the analyst to retrieve much of the information on loops and data structures and to figure out enough of the software's behaviour to fulfill the goal (e.g., finding vulnerabilities).

3 Document! document! document!

Good note taking is an essential quality for a reverse engineer. Dealing in assembly and hexadecimal numbers, it is very easy to get lost, especially after a long weekend, but also just because the incremental difficulty involved.

Document where you have been in the code, how many layers of cryptography/protection you encountered, and what worked and, perhaps more importantly, did not work. Document your structures as you figure them out. List addresses of interest. Make to-do lists. Cut and paste the output of various test runs you performed, and so on so forth. More detail on each of these in Chapter 4.

To keep track of documentation, we have tried many different tools over the years, such as wikis, Microsoft Word, Rational Rose UML designer, and good old pen and paper. Ultimately, we seem to have settled on Microsoft OneNote as the best tool for the job, with some assist from Microsoft Visio for the graphical parts. OneNote provides a free canvas where you can drop text, images, web pages, PDFs and almost anything else anywhere you want or link to external resources elsewhere on the computer or on the internet. It also provides basic editing capabilities so that you can start on your final report as you go. DRDC is working on integrating OneNote with IDA Pro so that one can easily navigate back and forth.

The documentation uses algorithms, C-like structure definition and comments, graphs, diagrams and other models to accelerate and enhance the understanding and to communicate this understanding to colleagues without them having to redo all the steps.

One can never have too many notes and it is the only way to pick up the work after a while without starting over from scratch every time. It will save your life down the road.

4 The methodology

This methodology uses a bottom-up approach to recover high-level structures and transform them into human-readable representations.

A preliminary and very important step for any reverse engineering task is often forgotten: goal setting. It is very important to clearly write down what the goal is. Reverse engineering is time consuming and very little result can be achieved without a clear objective. A clear goal tells the analyst what part of the problem to concentrate on and what parts can be delayed or simply abandoned.

This is a summary of the high-level steps we follow:

1. Reconnaissance
2. Preliminary analysis
3. Analysis and architecture reconstruction
4. Debugging and testing
5. Optional: Exploiting vulnerabilities

Each step is further refined in the following subsections. This is not a by-the-number instruction set. Don't hesitate to come back to a previous step if and when you find new information that needs to be refined in a previous step.

4.1 Reconnaissance

General tip: Always start from the known, and then push into the unknown. This sounds cliché, but it's very important. Going too quickly into the unknown is the surest way to get lost and waste a week doing nothing useful.

First, identify everything you can find about the program and the platform it runs on. Identify the processor (e.g., Intel x86, ARM). Try to identify the build environment (e.g., compiler, libraries). If it uses known quantities, this may help accelerate the analysis later on.

Get your hands on any documentation you can find. Skim the user guide to identify external components that the software interacts with as well as relevant information about the functionalities implemented by the software (e.g., the encryption it uses, the functionalities provided). If you are not familiar with the CPU, skim its documentation to find out about its architecture (e.g., 8/16/32/64 bit, endianness, virtual or physical memory, protection levels, registers, side effects of each instruction).

Scan patent application related to the software or its platform. Such applications contain a treasure trove of useful information on technological choices, many of which the manufacturer would rather protect but must divulge to obtain a patent.

Try to get the software development kit that goes with the program or its platform. This will contain information on external functions, all their arguments and their roles.

4.2 Preliminary analysis

Now it's time to fire IDA Pro and open up the binary. Since this is not a tutorial on IDA Pro, we will skip the setup and dive right into the next step.

Start by looking for things in the following order:

- Imported functions: Unless it's firmware, most programs will import external libraries. These have standardized names and arguments that must be public for compatibility. Go back to reconnaissance and find the documentation of those libraries.
- Strings of interest: Binaries are often loaded with meaningful strings, many of which should not remain in production code such as debug strings and unit test messages. Such strings also often point to interesting parts of the code. Examples of such strings are: password, SSID, IP address, error messages, and usage messages.
- Constants of interest: This is particularly interesting if the program you are analyzing performs cryptographic or mathematical functions. For example, cryptography uses very large prime numbers. IDA Pro tells you where those numbers are used so you can get at those functions directly.
- Error codes: Usually small constants that are used to report errors or status. If you are analyzing a web server, for example, look for the usual HTTP error codes, such as 404 (URL not found) and 200 (OK) to point you to the code that is handling requests.

Within this part of the process, also look for known protocol telltale signs and redo those steps. For example, TCP/IP requires specific data structures, error codes and error messages. Finding those within the code will get you to the part of the program that handles network communications. If you find a protocol you are unfamiliar, go back to reconnaissance and read a little.

Performing this step, a big picture begins to form that should allow you to divide the program into large chunks. For example, this part handles HTTP requests, this other part handles encryption, and this last part is the graphical user interface that maybe we don't really care about so we can ignore it.

Again, feel free to skip a step or go back to wherever an interesting piece of information takes you, but always remember the goal!

4.3 Analysis and architecture reconstruction

At this point, the reverse engineer usually has enough information to start reconstructing high-level structures by analyzing the actual behaviour of the program. This is where it gets into the nitty-gritty of the whole process. So, again, remember the goal! It is very important to concentrate only on the parts of interest. Fully reversing an average program can take months, if not years. Focus!

First, build a high-level view of all the device and external libraries the software interacts with. A generic example is shown in Figure 2.

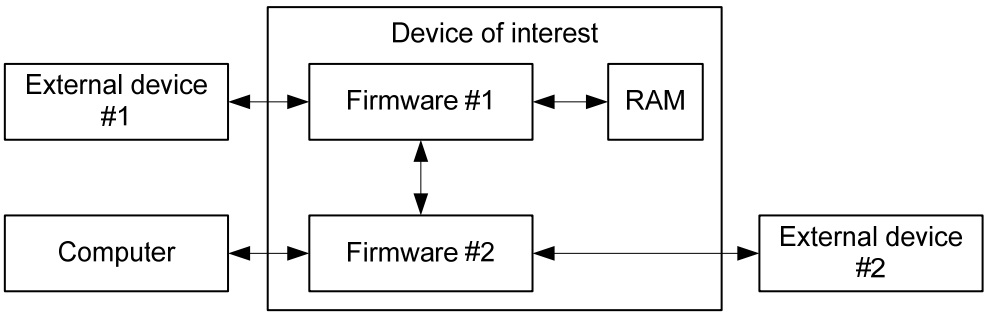

Figure 2: *Generic high-level view.*

From this generic view, build a few use cases of interest. This helps focus on the precise goal.

Start with a point of interest identified in the preliminary analysis.

Look at the control flow graph. Start on smaller leaf functions and guess their use by looking at the library they call and/or by analyzing smaller ones. If you have a guess as to their purpose, rename the function accordingly. Move on to bigger functions, using the renamed functions for further guessing.

Start looking at how the data flows between lines and functions. Describe how data is propagated and processed in the execution path. If you see variables being referenced with an offset, this is a good indication that you are looking at a data structure of some sort. Look through the documentation to see if it corresponds to a known entity. If so, rename it. If not, note what is stored or read at each offset you encounter to slowly build the whole structure.

Now look at the control flow between basic blocks and functions. This tells you how data is used and is processed by the program. Start building a state machine of your use case to describe how the program moves from one state to the next as it processes this data. Once you understand function calls and their flow, build a sequence diagram of your use case.

Repeat these steps for all the use cases and build new cases as required until you get a clear-enough picture of how the parts of interest behave.

4.4 Debugging and testing

By now, you should have a good understanding of the overall static behaviour of the parts of interest and a good guess on the dynamic behaviour it should have. If it's possible, you can spend some time refining this understanding using dynamic analysis. For example, test out your theories about what an input does to the output and how it flows through the program using a debugger.

Further refine your notes with this new knowledge and go back to previous steps if it creates new insights.

If you are looking for vulnerabilities, this is where to introduce fuzzing techniques into the mix to see if carefully crafted inputs can induce controllable crashes. It is also possible that you have discovered potential vulnerabilities already in your own analysis. Make sure that the crashes are repeatable and start trying to figure out if they are exploitable.

4.5 Optional: Exploiting vulnerabilities

Now it's time for the fun part: exploitation! In other words, this is where you verify if the vulnerabilities you have found really are threats to your software. Writing such an exploit can be easy or difficult, but good information abound (some pointers to training are included in the conclusion).

Chances are you will discover many vulnerabilities, especially in software that is not designed with security in mind (most firmware!) The second-to-last step is to rank those vulnerabilities according to their likeliness and severity. You can do that as part of a risk assessment process, such as DRDC's own Risk-based Cyber Mission Assurance Process [4]. The final step is to propose and implement mitigations for the top vulnerabilities while remaining within time and budget constraints.

5 Conclusion

Mastering software reverse engineering takes years and there is no substitute for hands-on experience yet. This Reference Document presents a very high-level view of the process we (mostly) follow at DRDC. The goal is simply to get you started on your long journey.

If you require more details, there are quite a few resources available. Specialized training is available from various reputable organizations (e.g., SANS [5]), conferences (e.g., [6–8]), free online tutorials (e.g., Briggs [9]) and books (e.g., [10, 11]). There are also various plugins available for IDA Pro that help automate or semi-automate many of these tasks [2, 12]. Finally, DRDC has various tools, techniques and plugins available to government agencies and will remain active in this area for the foreseeable future.

References

[1] Chromium (Google Chrome) (online), Google, https://www.openhub.net/p/chrome. (Access Date: July 2018).

[2] IDA Pro (online), Hex-Rays, https://www.hex-rays.com/products/ida/index.shtml. (Access Date: July 2018).

[3] Hex-Rays Decompiler (online), https://www.hex-rays.com/products/decompiler/index.shtml. (Access Date: July 2018).

[4] Rhéaume, F. (2018), Overview of the Risk-based Cyber Mission Assurance Process, In *Proceedings of Systems Concepts and Integration (SCI) Panel SCI-300 Specialists' Meeting on* "*Cyber Physical Security of Defense Systems*" *(NATO STO-MP-SCI-300)*, University of Florida, Florida, United States, May 2018.

[5] FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques (online), SANS, https://www.sans.org/course/reverse-engineering-malware-malware-analysis-tools-techniques. (Access Date: July 2018).

[6] Black Hat (online), http://www.blackhat.com. (Access Date: July 2018).

[7] DEF CON (online), https://www.defcon.org. (Access Date: July 2018).

[8] RECON (online), https://recon.cx. (Access Date: July 2018).

[9] Briggs, M. (2011), Introduction To Reverse Engineering Software (online), Open Security Training, http://www.opensecuritytraining.info/IntroductionToReverseEngineering.html. (Access Date: July 2018).

[10] Eagle, C. (2011), The IDA Pro Book, 2nd ed. No Starch Press, https://www.amazon.ca/IDA-Pro-Book-2nd-ebook/dp/B005EI84TM. (Access Date: July 2018).

[11] Sikorski, M. and Honig, A. (2012), Practical Malware Analysis: A Hands-On Guide to Dissecting Malicious Software, 1st ed. No Starch Press, https://www.amazon.ca/Practical-Malware-Analysis-Hands-Dissecting/dp/1593272901. (Access Date: July 2018).

[12] Thawt, O. (A list of IDA Plugins (online), https://github.com/onethawt/idaplugins-list. (Access Date: July 2018).

Glossary

Basic block	A block of assembly that is always executed in the same order from its first to its last line. Any jump or landing point creates a new block.
Call graph	A graph, usually directed, showing functions and their caller/callee relationships.
Control flow graph	A graph showing the flow of execution, usually between basic blocks, but sometimes between functions or libraries.
Fuzzing	A technique that throws millions and millions of carefully crafted inputs at a program hoping to find controllable crashes.
Sequence diagram	A Unified Modelling Language diagram that shows a sequence of events implementing functionality.
State diagram	A Unified Modelling Language diagram that shows transitions between objects in the program. Typically classes, objects can be any other chunk of the program or the system, such as libraries or different parts of the hardware.

DOCUMENT CONTROL DATA

(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g., Centre sponsoring a contractor's report, or tasking agency, are entered in Section 8.)

DRDC – Valcartier Research Centre
Defence Research and Development Canada
2459 route de la Bravoure
Quebec (Quebec) G3J 1X5
Canada

2a. SECURITY MARKING (Overall security marking of the document including special supplemental markings if applicable.)

CAN UNCLASSIFIED

2b. CONTROLLED GOODS

NON-CONTROLLED GOODS
DMC A

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Software reverse engineering: A basic methodology

4. AUTHORS (last name, followed by initials – ranks, titles, etc., not to be used)

Lavoie, Y.; Dessureault, A.; Salois, M.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
August 2018	15	12

7. DESCRIPTIVE NOTES (The category of the document, e.g., technical report, technical note or memorandum. If appropriate, enter the type of report, e.g., interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Reference Document

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

DRDC – Valcartier Research Centre
Defence Research and Development Canada
2459 route de la Bravoure
Quebec (Quebec) G3J 1X5
Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
05aa	

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC-RDDC-2018-D078	

11a. FUTURE DISTRIBUTION (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Public release

11b. FUTURE DISTRIBUTION OUTSIDE CANADA (Any limitations on further dissemination of the document, other than those imposed by security classification.)

12. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

Contrary to traditional forward engineering, software reverse engineering is a time-consuming task that not every programmer can perform after a few hours of training. Only a dedicated few top-level analysts can become proficient reverse engineers. To help managers sift through potential candidates, DRDC is often asked for an overview of the process we follow. This Reference Document presents a basic, high-level methodology that highlights this process' broad strokes.

Contrairement à l'ingénierie traditionnelle descendante, la rétro-ingénierie logicielle est une tâche très longue qui ne peut être accomplie par n'importe quel programmeur après quelques heures de formation. Seul un nombre restreint d'analystes de haut niveau et très motivés peuvent devenir de bons rétro-ingénieurs. Pour aider les gestionnaires à filtrer les candidats potentiels, RDDC se fait souvent demander un aperçu du processus que nous suivons. Ce document de référence présente à haut niveau une méthodologie de base pour illustrer les grandes lignes de ce processus.

13. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g., Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Software reverse engineering; Cyber Security; Methodology