Reverse Engineering as an Education Tool in Computer Science

Ivan Klimek, Marián Keltika and František Jakab
Computer Networks Laboratory, Technical University Košice, Slovakia
(ivan.klimek, marian.keltika, frantisek.jakab)@tuke.sk

***Abstract* - The concept of Reverse Engineering is used in many fields of IT every day, to name just a few: legacy compatibility, binary code patching, malware analysis, network protocols analysis, debugging or even rapid prototyping. Despite its broad use, reverse engineering is not actively taught as a part of computer science courses. This paper tries to provide a survey of some of the real life usage scenarios of reverse engineering, analyzes what skills and ways of thinking are developed by reverse engineering and provides examples how reverse engineering could be taught by practical problem solving, introducing creative thinking models and strategies. We focus on the importance of reverse engineering as a tool to ignite the self-motivation of students and systematically build their logical thinking capabilities and analytical skills.**

I. INTRODUCTION

Since the beginning of time the natural way of thinking was to "reverse engineer" the nature; flame, tools, Newton's apple, nuclear fusion all of them are just examples of understanding an external process, not necessarily deep understanding but enough to be able to use it for our own purpose. For thousands of years man did not understand the chemical process of flame and we still don't know if gravitation has its own particle or not. We look at these processes as black-boxes, describe experimentally determined properties and just use them. Such way of thinking is called top-down - it starts from a complex system and tries to decompose it into sub-parts [1, 2, 3, 4]. This way reverse engineering allows to focus only on the important detail through abstracting all - for the given task - unimportant details, for example use the fire to cook without caring about how it works. Interestingly, the whole education system takes the opposite approach - "forward engineering" - builds from scratch, layer upon layer of adding complexity, which is also called a bottom-up approach. We don't aim to replace the current system, we just want to provide a different point of view that is normally not taught and as we will show can be helpful in various situations where it can solve normally unsolvable problems or greatly reduce the complexity of the solution.[1]

II. REVERSE ENGINEERING

The practical aspects of Reverse Engineering (RE) were already studied deeply in the thesis *Software reverse engineering education* by T. Cipresso (San José State University) [5]. Cipresso divides RE into two categories: Software development related and Security related as shown on Figures 2 and 3. He also states that one of the strongest motivations why teach RE is Legacy software maintenance and proves his point by stating that about 70% of the source code in the entire world is still written in COBOL: *"Since it's cost-prohibitive to rip and replace billions of lines of legacy code, the only reasonable alternative has been to maintain and evolve the code, often with the help of concepts found in software reverse engineering."* Figure 1 illustrates the process of maintaining legacy software systems.

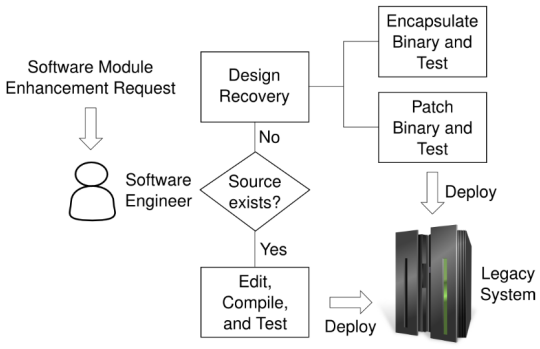

Figure 1. The process of maintaining legacy software systems

Development related software reverse engineering scenarios include:

- **Achieving Interoperability with Proprietary Software:** Develop applications or device drivers that inter-operate (use) proprietary libraries in operating systems or applications.
- **Verification that Implementation Matches Design:** Verify that code produced during the forward development process matches the envisioned design by reversing the code back into an abstract design.
- **Evaluating Software Quality and Robustness:** Ensure the quality of software before purchasing it by performing heuristic analysis of the binaries to check for certain instruction sequences that appear in poor quality code.
- **Legacy Software Maintenance, Re-engineering, and Evolution:** Recover the design of legacy software modules when source is not available to make possible the maintenance, evolution, and reuse of the modules.

[1] For the purposes of this paper we will use the term "Reverse Engineering" (RE) as a synonym for Software Reverse Engineering (SRE) and Reverse Code Engineering (RCE)

978-1-4577-0052-1/11/$26.00 ©2011 IEEE

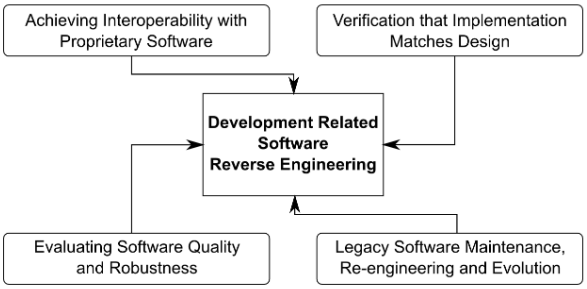

Figure 2. Development related software reverse engineering scenarios

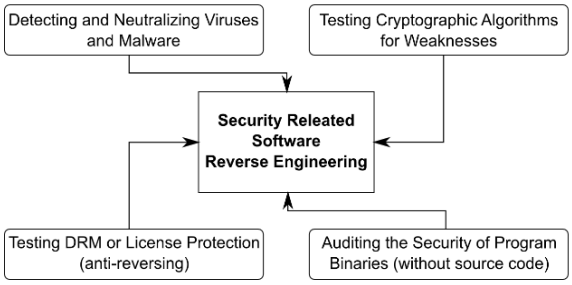

Figure 3. Security related software reverse engineering scenarios

Security related software reverse engineering scenarios include:

- **Detecting and Neutralizing Viruses and Malware**: Detect, analyze, or neutralize (clean) malware, viruses, spyware, and adware.
- **Testing Cryptographic Algorithms for Weaknesses:** Test the level of data security provided by a given cryptographic algorithm by analyzing it for weaknesses.
- **Testing DRM or License Protection (anti-reversing):** Protect software and media digital-rights through application and testing of anti-reversing techniques.
- **Auditing the Security of Program Binaries**: Audit a program for security vulnerabilities without access to the source code by scanning instruction sequences for potential exploits.

III. Reverse Engineering in Education

Some universities already tried integrating Reverse Engineering techniques with traditional computer science courses; one example is the University of Missouri-Rolla. The results were very encouraging, 77% students indicated that introducing reverse engineering methodology reinforced concepts taught during lectures. Furthermore, 82% wanted it to be blended in future courses, especially those that dealt with design [6].

Both mentioned approaches (San José State University and University of Missouri-Rolla) are trying to integrate reverse engineering into their coursework by focusing only on its practical usage completely disregarding its deeper pedagogical value which we are going to explore in the following text.

Motivation comes in two forms, intrinsic and extrinsic [7]. We are going to cover the intrinsic motivation also called: self-motivation, which is the ability of oneself to push and continue on a path even when obstacles are encountered without influence from other people. Clearly self-motivation is a critical trait of a successful engineer and is necessary to be able to solve problems independently. We argue that self-motivation is to some extent a learned trait, learned by experience.

As a matter of fact we are not the first to formulate this idea as a famous quote by Admiral William Halsey proves: *"There are no great men, only great challenges that ordinary men are forced by circumstances to meet."*

Imagine being repeatedly forced to overcome all expectations. Overcome your own limits and do what you never thought possible. By the repeated exposure to this practice build self-confidence and boost ego. Ego boosting is naturally addictive and thus a form of self-motivation to overcome challenges [7]. Reverse engineering is perfect for ego boosting because results can be obtained fast and without vast prior knowledge.

To prove this statement we could look at why RE is so popular amongst certain engineers. An example of a group that reverse engineers software for enjoyment is CORE which stands for "Challenge of Reverse Engineering" [8]. Another example could be the recent high-profile hacking attacks of LulzSec, who draw their name from the neologism "Lulz", (from LOLs), "laughing out loud", which often signifies laughter at the victim of a prank, and "Sec," short for "Security" [9]. The same factors that motivate these engineers to take the extreme risks of high-profile hacking/cracking could and should be used in a controlled manner to motivate students.

Classical learning approach of forward engineering could be then compared to reverse engineering as learning for success to learning through success.

This logic can be simply demonstrated on an example RE task, an application binary is provided and the students are asked to change the behavior of the application without access to the neither source code nor documentation. The only technique applied is the execution flow control using a debugger[2]. A very basic knowledge of the Assembler programming language is required so that the students can monitor and control the chain of execution[3]. Students need to explore the behavior of the application, use analytical skills to identify which part of it is important to focus on, find the

[2] For example using OllyDbg which is an x86 debugger that emphasizes on binary code analysis, which is useful when source code is not available.

[3] Basic knowledge of jumps, calls and returns should suffice.

corresponding code and trace the execution flow until they realize the exact instructions defining the application behavior that they have been asked to modify. Using a systematic experimental approach they modify the localized set of instructions and monitor the resulting behavior until the given goal is accomplished. The students just created their first binary patch, or we could say their first software "crack"[4]. We suppose the broad majority of students although came into contact with software "cracks" never thought they would be able to create one on the first lesson of a course.

The same scenario can be varied depending on the level of prior knowledge of the Assembler language for example the execution flow could be diverted by understanding the logic comparisons and changing the "cmp" instructions accordingly. Or a "key generator" could be reverse-engineered based on the understanding of the key verification process.

The subject of reverse engineering is not limited to dissecting application binaries, network protocols are an interesting target too. We could look at them as on input/output of an application and by understanding them, understand the application. Transport Layer Security (TLS) resp. Secure Socket Layer (SSL) are the most commonly known cryptographic protocols that provide communication security over the Internet. They are used in applications such as web browsing, electronic mail, Internet faxing, instant messaging and voice-over-IP (VoIP). The students could be asked to reverse engineer an SSL-protected protocol using a network proxy[5]. Analyze it and either simulates a Man-in-the-Middle attack or a client that could use the network provided service without the original software. Again, analytical thinking would be required to manage the "complexity" of the protocol, focus at one layer of complexity at a time and work their way down through it. With only limited prior knowledge the students would accomplish to "hack" a protocol protected with the same mechanism that protects most of the Internet within a single lesson. Of course the goal is just to open the door for the students, so that they realize they are able to accomplish more than they thought. Further exploitation of the protocol could be given as homework; the most creative student could be rewarded.

As already mentioned, the top-down approach employed by reverse engineering is extremely valuable for example also for rapid prototyping. Let's say the students would be given the task of prototyping a system too complex to develop from scratch, further they would be given only a very limited time to fulfill the assignment. The only way how they could manage would be to look for existing solutions like libraries or whole applications which they could combine to create a system with the desired functionality. To make the educational point clearer they would not need to make it actually run, but just understand the API or functional parts and be able to combine them using pseudo-code. The simplest and most elegant solution should be rewarded. This way the students would realize that it is possible to accomplish far more than they thought in a limited time frame by dissecting a complex system into sub-parts and reusing existing solutions.

[4] Software cracking is the modification of software to remove or disable features which are considered undesirable by the person cracking the software.

[5] For example the Fiddler debugging proxy could be used to analyze and simulate protocol behaviour without the need to write a single line of code.

IV. CONCLUSIONS

If we define hacking as the art of creative problem solving [10], reverse engineering is the art of problem dissecting and thus a critical part of any efficient problem solving process. In this paper we presented ideas how reverse engineering can be used to ignite student self-motivation, systematically build their logical thinking capabilities and analytical skills. With the exponential growth of the amount of information necessary to solve IT-related problems or to compete in the global workforce market, it is crucial to understand that there are only two options where the education system can be headed. Either try to compete with population rich countries like India or China using the traditional bottom-up approach focusing on skills and information that keep changing at a growing rate or try to augment the education system with techniques like reverse engineering to focus on teaching how to think creatively instead. We argue that with the right way of thinking and a systematical approach, all problems are solvable. Reverse engineering is the art that teaches these skills. Due to its importance we sincerely hope that reverse engineering will be integrated into computer science education either as a standalone course or as a part of existing courses.

ACKNOWLEDGMENT

This work was supported by the Slovak Cultural and Educational Grant Agency of Ministry of Education of Slovak Republic (KEGA) under the contract No. 3/7245/09 (60%). This work is also the result of the project implementation Development of the Center of Information and Communication Technologies for Knowledge Systems (project number: 26220120030) supported by the Research & Development Operational Program funded by the ERDF (40%).

REFERENCES

[1] Niklaus Wirth, "Program development by stepwise refinement," *Communications of the ACM*, 14(4):221–227, April 1971.

[2] Gerald M. Weinberg. *The Psychology of Computer Programming.* New York: Dorset House, 1998.

[3] Eldad Eilam. *Reversing: Secrets of Reverse Engineering.* Indianapolis: Wiley, 2005.

[4] Mark Stamp. *Information Security Principles and Practice.* New Jersey: Wiley, 2006.

[5] Cipresso, Teodoro, *Software Reverse Engineering Education.* SJSU Master's Thesis, 2009.

[6] Muhammad Raza Ali, "Why teach reverse engineering?," *ACM SIGSOFT Software Engineering Notes*, v.30 n.4, July 2005.

[7] Bénabou, R. and Tirole, J, "Intrinsic and Extrinsic Motivation," *Review of Economic Studies*, 70: 489–520, 2003.

[8] Wikipedia (2011), "Reverse engineering," available: http://en.wikipedia.org/wiki/Reverse_engineering [accessed 15 Sep 2011].

[9] Wikipedia (2011), "LulzSec," available: http://en.wikipedia.org/wiki/LulzSec [accessed 15 Sep 2011].

[10] Erickson, Jon. *Hacking the Art of Exploitation.* 2nd ed. San Francisco, Calif: No Starch, 2008.