Details of a Malicious Code Analysis Course

Ann Sobel
sobelae@miamioh.edu

Michael Gentile
gentilm5@miamioh.edu

Miami University
Oxford, OH 45056

Abstract – Miami University has committed to the goal of increasing cybersecurity education. To this end, the department of Computer Science and Software Engineering is considering an offering of a cybersecurity minor which covers the accreditation criteria of a National Security Agency CAE cyber operations fundamentals focus area. A trial offering of a new course satisfying the mandatory topic of reverse engineering is outlined here. The main component of the learning objectives of this new course is the hands-on experience of using disassembly tools to identify malicious code. Future work includes the assessment of this laboratory experience both in the tools used and their effectiveness when performing static analysis of assembly code to identify potential nefarious acts.

Keywords

Undergraduate Education, Cybersecurity, Laboratory Exercises, Accreditation

1. INTRODUCTION

Miami University intends to increase its cybersecurity learning opportunities. As with most computer science departments, we have experienced an explosion in the number of undergraduate majors which has led to a constant need for additional staff. An increase in our cybersecurity course offerings was a clear conflict with our staffing considerations so we took the approach of minimizing the changes necessary to not only increase our course offerings but to offer a cybersecurity program that matched the accreditation criteria of the National Security Agency CAE cyber operations fundamentals focus area [1]. We were able to minimally change the existing core courses of architecture, operating systems, and discrete mathematics so that a modification of our existing network security elective course and only two new courses were need to define the minor. One of the new courses on cellular and mobile technologies will be offered by our computer engineering department and the other course on reverse engineering that will be offered by our department is discussed here.

2. MALICIOUS CODE ANALYSIS COURSE

This course was created to satisfy the mandatory requirements of reverse engineering with the addition of the required legal & ethical issues. All aspects of this course were patterned off of the CAE cyber operations fundamentals. The objectives and rationales come from the description of the KU reverse engineering.

> A critical skill within the cybersecurity field is understanding software that is of unknown origin or software that has source code that is unavailable to assess whether malicious code exists. Students will be able to use tools to perform mostly static and limited dynamic analysis of software in an attempt to understand its functionality, both expected and abnormal.

The major learning objectives/outcomes that must be defined for every course at Miami University and which are used to assess the teaching effectiveness of each course offering follow. Students will be able to…

1. Explain basic static and dynamic malware analysis.
2. Analyze assembly code of software and demonstrate the ability to trace assembly code to probable language-specific code.
3. Use existing tools such as IDAPro and OllyDbg to analyze object code.
4. Demonstrate the ability to identify malicious errors.
5. Explain basic classification of known malware strategies.

The text book chosen, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software [2], was a factor in the course topics chosen and when those topics were delivered. While the text book contains a practical approach to malware code analysis, it assumes the reader has a rather extensive background in Windows file structure, compiling, programming language theory, assembly language instructions and executable file formats. This caused a number of weeks to be spent on programming language grammars, lexical analysis, generation of assembly code, packaging of that code in an executable, and the expected format of an executable given that our audience was a CS major undergraduate student.

3. TOOLS IN THE LABORATORY

3.1 Student Laboratory Basics

Initial consideration of the course laboratory had us weigh whether we wanted to use Windows, Linux, or have the students use their own personal computers. The later was discounted given we didn't want to corrupt their own environments since they needed them for other course work. Given the textbook's focus on Windows, we chose that direction. Ultimately, our virtual machine was isolated from the internet, thereby isolating potential harm, and every student was granted administrator access. A major downside to this approach was that anyone at any time could render the environment unusable.

3.2 Virtual Machine Setup

The main goals of the analysis environment are that the system should be contained, usable and independent. It shall be focused on a single analysis machine which is both the infected machine and the analysis machine. The system should be contained such that the malicious code should not have the ability to spread or scan through the network. The system should possess a graphical user interface to users in such a manner that multiple users can work on the system at once while doing non-computationally expensive tasks. Finally, the system needs to be independent of the environment around it such that a failure of this system caused by the malicious code should not cause other systems within the cluster to experience issues.

The ways in which these goals can be achieved is multi-faceted and depends strongly on the system architecture where these ideas are being presented. Although, there exists one solution that achieves all of these goals and has very subtle drawbacks to an educational environment: native virtualization. Native virtualization through tools such as VMWare or VirtualBox allow for the creation of incredibly isolated systems that achieve all the usability of a native machine with a minor performance fee. These machines can be isolated on the network such that only a specific port can be exposed to the outside world and small long-term storage drives are the only thing exposed to the environment. The system's state can easily be stored in a file and restored after mistakes were made with the malicious code. Overall, this architecture will be the best for most educational purposes

The largest issue with this architectural scheme is that some modern malware does not even run on machines that it believes to be virtualized. It has become incredibly easy to detect virtualized hardware. Malware producers are not naive to the notion that researchers do not want to run their code on native machines. Therefore, some more sophisticated malware cannot be analyzed in these environments. Although, most malware that would be analyzed in a single semester course would not venture into this level of complexity and could be easy to avoid as an instructor with some foresight.

A more difficult decision to make would be the operating system to choose for the system. Providing students with patched versions of the Windows

operating system can severely limit the effectiveness of the malware itself and truly can stop the malware from doing the things that it normally does. Obviously, this can severely limit the dynamic analysis of the malware. Further, most of the malware that you will find for textbooks will be designed for Windows XP, Vista or 7. This does not offer a large difference to anything but the case of Windows XP. Windows XP is no longer supported officially by Microsoft. Therefore, most malware can run (relatively) unobstructed in this ecosystem. This operating system choice bars the case of massive malware breaches such as WannaCry. In all cases, these systems will all work. With a strong system of isolation there should be no worry of running an unpatched version of any of these systems.

Access to the system both by the users and to access files from the web are both necessary functions of usability for the system. The recommended course of action to allow users to access the system would be to utilize remote desktop. Remote desktop allows for the user to access the system completely away from it and allow concurrent connections. For file system access, it is not recommended to expose more of the drive to the host environment than is necessary. Therefore, using a network file system would be the recommended system. This should be configured such that there is a web interface that is accessible via the intranet and is whitelisted for upload and download out of the containing environment.

3.3 Tools and Their Expected Usage

The only tools that the student should need on their personal computer would be a remote desktop client. The tools that should be taught in this course should be IDA Pro and OllyDBG. Other tools that would be useful would be PEiD, Procmon, VirusTotal and Wireshark. These other tools are typically rather simple and do not need taught. Although, it would be sufficient for a cursory review in class to go over their simple purpose.

The biggest tool in reverse engineering in general would commonly be IDA Pro. IDA Pro exists primarily as a static analysis tool to analyze executables in various different formats to gain knowledge about what they are

doing and what they are accessing. IDA has the ability to also do dynamic analysis through various additional installations. Within the static analysis realm, IDA does a large amount of the leg work by breaking down the program into different functions and decoding the executable into its imports, data and machine code. It then allows a graphical interpretation of the machine code such that it can be illustrated how the software works and how it is tied together. The tool allows for the reverser to allow them to comment on various locations within the code, replace variable locations with names and view strings of the program in plain text. This allows the program to become closer to human readable and understandable, even with a somewhat minimal understanding of assembly. The tool can be used as a sort of multitool of static analysis. It can also do things like list imported functions, exported functions, view loaded libraries, show all the names used in the files and get a logical view of code structures.

IDA Pro should likely be the largest feature tool taught within the course. It reduces the time spent on other items dramatically if the tool is used properly. The largest cons of the use of this tool is that it has a steep learning curve and it can be very easy to become lost in the details. The tool should be taught with very simple examples in class and students should have adequate time with simple problems in the tool before being exposed to more difficult, realistic malicious code.

OllyDbg is the IDA Pro of the dynamic analysis realm. OllyDbg gives the standard tools one would expect from any modern debugger in the realm of computer science. It allows for step through, step over functions and sometimes reverts. It gives a view of different variables and allows an in depth view of the program stack. It would be the ideal way of stepping through a program and trying to elicit expected behavior such that you can monitor exactly what it is doing and how to achieve it. It further allows the modification of this source code so that other operations can be explored. This tool should be introduced within the course and should be taught at a similar time IDA has been introduced. It allows students to do proofs of concept on their work and see if

they can truly predict the way that the program is going to act from static analysis to dynamic analysis.

WireShark, simply, is a tool that is used to monitor networks. Most malware now contacts some networking imports that allow the malware to tell the writers or infectors of the malware that a machine has been infected and to begin doing what they want with the machine, whether it be remote execution or some sort of spyware. It would be useful to know what that malicious code is telling the network or seeing if it is trying to branch out. Therefore, we would use a tool like WireShark to monitor these things. WireShark can be setup before the malware is executed. When the malware is executed, WireShark captures the packets that have been sent and received and allows the analyst to view what is occurring in the network. This can give great hints as to what the malware is trying to accomplish with very little effort.

PEiD, VirusTotal and Procmon are two very simple tools. PEiD allows for packed malware to be detected and then decodes it into its unpacked state. This software is not perfect, although it is a very good step before manually attempting to unpack malware. Procmon, on the other hand, is another dynamic analysis tool that allows for the analyst to view all the processes that are created and killed during the execution of the code currently under review. This becomes particularly useful for programs that alter the behavior of Windows. Finally, VirusTotal is more of a real-world tool where a program has been already identified as malicious and matches known virus signatures. It checks against a combination of various anti-virus datasets. With these datasets, it indicates whether the file indicated is known to be malicious and gives a name and type to this malware. This gives a major head start to the reverse engineering process. It's an obvious first step when dealing with an unknown file.

4. LESSONS LEARNED

It can't be emphasized enough to start really early performing the cybersecurity laboratory setup. We needed IT administration permission to

establish the virtual machine environment, and have the installation of the various analysis tools. There were instances when tools didn't work as advertised sometimes due to the version of the operating system they required; a clear problem when your IT support policy is to install the latest and greatest.

Those in academia have certainly experienced student reluctance to embrace new operating system commands, new forms of communication, and file transportation so these activities needed to be documented and frequently revisited. Our biggest issue was that the virtual machine was not part of our usual domain and access required students to provide namespace identification when logging in. File transfers required the use of Netdisk which led to domain and two factor authentication issues which weren't insurmountable but they weren't natural either.

Obtaining and using infected files was a headache. We had to resort to the very methods that someone would use to hide their corrupted files from the unsuspecting public. We most often used compression for file packaging to obfuscate infected files from virus detection software on a PC but it wasn't a given that we would be successful which lead to hiccups in material delivery. Even what might appear as simple DLL linkage added to our headaches.

Several times during this term we have pondered whether a Windows environment was the best choice. Using this environment limited our compiler choice and which system string commands were readily available, such as grep, sed and awk. Having a powerful set of string manipulation and identification tools highlights the essence of what disassembly analysis attempts to do. The course began with an exercise meant to motivate the underlying problem for which reverse engineering is necessary; namely, the identification of substrings in flu DNA for possible sources of mutations for predicting possible flu strains in the upcoming year. The solution consisted of only Linux-based commands and on a single command line. On the flip side, IDAPro may have a Linux version but it doesn't come close to the functionality of the Windows version. Lastly, if the target of the laboratories are to simulate work-related experiences, then one shouldn't consider using Linux.

Malware analysis itself takes a lot of prior knowledge in order to thoroughly understand what is happening while using these tools. In academia, it is not a common a occurrence for platform specific operating system intricacies to be taught to the degree in which would be sufficient for this course. Things like the registry, tasks and file structures are all things that need a large amount of review before the course can actually start to focus on malware. It must be a large consideration of what the student has been taught in their operating system course before the student can adequately understand what is happening. Many of these topics may need revisited or expanded upon for students to truly understand the course material.

Finally, the nature of malware makes it incredibly difficult to pinpoint exactly how malware will interact with different versions of the system. Off of every operating system patch there exists a chance that there will not be consistency among how you believe the malware to behave (when writing the assignment) and how the students will experience the behavior of the malware. Therefore, it is important to keep the system constant and that the assignments are checked for accuracy close to their assignment time. This will help to ensure consistency among malware behavior as well as assignment relevance.

5. SUMMARY

Miami University has begun an effort to increase cybersecurity education. The Computer Science and Software Engineering department has allowed the offering of reverse engineering course that covers the accreditation requirements of the National Security Agency CAE cyber operations fundamentals focus area [1]. It is the author's desire to receive approval to create a cybersecurity minor that can be accredited under this program. This particular course, Malicious Code Analysis, focuses on teaching reverse engineering using mostly hands on laboratory exercises. The laboratory setup, tool selection, and tool usage were detailed here so that other educators who wish to adopt such a course might benefit from our experience.

REFERENCES

[1] National Security Agency (NSA) Centers for Academic Excellence (CAE), "Academic Requirements for Designation as a CAE in Cyber Operations Fundamental", Online at https://www.nsa.gov/Resources/Students-Educators/centers-academic-excellence/cae-co-fundamental/requirements/

[2] Sikorski, Michael and Andrew Honig, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, 2012, ISBN:978-1-59327-290-6.