SOFTWARE ENGINEERS' PREPAREDNESS FOR SECURITY THREATS: AN EXAMINATION OF THE UNIVERSITY OF VIRGINIA'S CYBERSECURITY EDUCATION

A Research Paper submitted to the Department of Computer Science
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

By

Jason Tufano

August 6, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

ADVISOR
Jim Cohoon, Department of Computer Science

Software Engineers' Preparedness for Security Threats

An Examination of the University of Virginia's Cybersecurity Education

Jason Tufano
University of Virginia
jt2ef@virginia.edu

ABSTRACT

This technical report examines the cybersecurity curriculum of the University of Virginia, using a cybersecurity capture the flag event as an indicator of the effectiveness of the education overall. The report discusses six categories of security challenges presented in the capture the flag event: reverse engineering, binary exploitation, cryptography, web exploitation, forensics, and miscellaneous. For each category, the report assesses how prepared University of Virginia software engineers are to take on a specific challenge of that variety, as well general problems from the category. This analysis considers students who have taken cybersecurity electives, as well as students who have only completed required courses. Finally, the report combines these six analyses to make an overall recommendation for the future of the University of Virginia's cybersecurity curriculum for computer science students.

1 Introduction

As software becomes more and more complicated, so too does the challenge of keeping that software secure. Cybersecurity is a rapidly changing field, with constant advancements in security met with newfound vulnerabilities in programs. As time goes on, more and more technical knowledge and skills are necessary to keep products secure. To that end, cybersecurity has quickly become essential knowledge for all software engineers.

Although complete knowledge of cybersecurity topics is unrealistic for all programmers, due to its importance, engineers should at least have a cursory knowledge of major cybersecurity topics. As it stands, solving some security challenges would require a skillset foreign to many computer science (CS) students. To determine whether University of Virginia (UVA) CS graduates are prepared to face the multitude of security challenges present in modern software development, tasks from a cybersecurity capture the flag (CTF) event will be evaluated. A CTF event involves competitors using cybersecurity knowledge to find "flags", which are usually a short string of characters that prove a challenge was completed. This report assesses each task in terms of how UVA required CS courses prepare a competitor, as well how UVA cybersecurity electives prepare a competitor.

2 ImaginaryCTF 2021

From July 23 to July 27, ImaginaryCTF held their 2021 CTF event; over 1,000 competitors, made up of teams and individuals alike, took on the various cybersecurity challenges presented. It was a jeopardy style CTF event, where each "flag" to be captured is contained within its own distinct challenge. The event consisted of over forty challenges spread across six categories: reverse engineering, binary exploitation, cryptography, web exploitation, forensics, and miscellaneous.

The event was designed with both beginners and more experienced challengers in mind, making it a perfect testing ground for cybersecurity skills and knowledge. The following sections focus on each specific category from the CTF, and assess how well-equipped UVA software engineers are to tackle challenges of the category. To that end, each section will contain both a general analysis of the category as well as a specific challenge from the ImaginaryCTF event.

2.1 Reverse Engineering

Reverse engineering is the process of figuring out how an executable file functions without access to the source code. This process is usually done by viewing the assembly code of the file, which is difficult to interpret, but at least human readable. Some tools called *decompilers* go one step further to convert the executable into possible source code, which is often more intelligible than the assembly language code. Challenges in this category usually involve determining what input to the program will achieve a desired output (in the case of a CTF event, what input will cause the program to output the flag).

In one such challenge from the ImaginaryCTF event, the executable file given was disguised as a binary exploitation challenge, which is described in the next section. At first glance, the program seemed to never run the desired section of code, which is usually a hallmark of binary exploitation problems. However, upon further examination, there is a call to the desired section of code hidden within what looks like a failure function. After this realization, the problem becomes standard reverse engineering affair, determining what input will cause the code to execute as desired. Ethan Oh [2]

provides a full explanation of the problem, showing exactly how the program works.

This challenge takes a bit of digging to find the key sections of code, but once the correct sections are found, all UVA CS graduates should be capable of solving this challenge given enough time. The only necessary knowledge to solve this problem is an understanding of what information is contained in an executable file and the ability to understand assembly code, both of which are covered in required CS courses. In general, reverse engineering is well within the scope of the core CS curriculum, because understanding assembly code is the primary skill needed in solving challenges of this kind. Cybersecurity electives at UVA, especially Defense Against the Dark Arts, may help with challenges of this category, but contain no specific necessary information for solving reverse engineering challenges.

2.2 Binary Exploitation

Binary exploitation is similar to reverse engineering. The challenger has access to an executable file but not to the source code. The challenger must figure out a way to make the executable behave in a desired way. The difference between binary exploitation and reverse engineering is that binary exploitation involves making use of vulnerabilities in the code that allow it to behave in unanticipated ways. One example of this is known as buffer overflow, which is explained below using a challenge from the ImaginaryCTF event.

The binary exploitation challenge in question required running part of the code that, under normal use, would never be accessed. Accessing the desired part of the code requires the buffer overflow technique, which essentially involves providing too much input for the program to store, causing it to overwrite other important information. In this challenge, there is a key piece of data that, when overwritten correctly, gives access to a remote machine. Once this access is obtained, the flag is easily found using the terminal. Github user s-3ntinel [3] provides a full explanation of how to overwrite the data to solve this challenge.

There is no guarantee that a UVA CS graduate would be able to solve this problem. All of the knowledge required for this challenge is taught in core classes, but without exposure to the buffer overflow technique itself, successfully completing this challenge may be difficult. While vulnerabilities such as buffer overflows are not discussed in-depth, if at all, in the core UVA CS curriculum, a student who has taken cybersecurity classes should be well aware of this type of exploit. The Introduction to Cybersecurity class discusses buffer overflows, and in the Defense Against the Dark Arts course, the technique is often utilized.

2.3 Cryptography

Cryptography challenges are all about finding ways to break encryption. The goal of these challenges is not to find ways to defeat strong encryption schemes, but rather to find an error with a specific implementation of an encryption algorithm that makes it vulnerable to attacks. Completing these challenges requires an understanding of how various encryption methods work, as well as knowledge of how slight changes to the algorithm can completely nullify any security it aims to provide.

In the ImaginaryCTF competition, there was a challenge involving a variation of a commonly used encryption technique called Diffie-Helman key exchange (DHKE). The error with its implementation is that in the challenge the DHKE is used for message encryption, which is not the purpose of the algorithm. The DHKE is meant to determine a shared secret between two users, such that the users can use the shared secret to encrypt future messages. In the challenge, however, the DHKE is directly used to encrypt messages. Even worse, an attacker can access a plaintext and an encrypted version of the same message, which provides all the information needed to break the encryption. Once again, Github user s-3ntinel [4] provides a more detailed explanation of how to break the encryption in this challenge.

Encryption algorithms are seldom discussed in require UVA CS courses. Students who did not take cybersecurity classes may not know about the DHKE or other encryption techniques, and therefore may struggle with this challenge. The DHKE is discussed in the Introduction to Cybersecurity and Network Security courses, giving students who have taken either of those courses an advantage in completing this challenge.

2.4 Web Exploitation

Web exploitation challenges involve insecurities present in code used for websites. Two common exploits are cross site scripting (XSS) and SQL injection. XSS attacks rely on making a website execute JavaScript code for other users, usually with the intent of gaining access to those users' data. SQL injection, on the other hand, relies on tampering with a query to a database, often to access accounts of other users. SQL is a language used to communicate with a database, and many websites include user input in SQL queries, for example when a user fills in username and password fields to log in. If this input is not properly sanitized, the user can provide malicious input to access data they should not be able to.

In the ImaginaryCTF competition, there was a SQL injection challenge where the user input was partially sanitized. The program would sanitize the input by removing certain keywords from the input. However, the removal algorithm only did one pass over the input, meaning if it had a keyword left over after another instance of that keyword was removed, the leftover keyword would never be removed. For example, one keyword in SQL is "OR". If the algorithm tried to remove "OR" from "OORR", it would remove the middle "O" and "R", resulting in the first "O" and last "R" to come together to form "OR", which is still a keyword. Using this technique allows the SQL query to be tampered with. Combining this with another technique involving the SQL substring command allows us to determine usernames and passwords one character at a time. The password of the only user is the flag for this challenge.

Github user s-3ntinel [5] once again provides a wonderful write-up describing how to overcome this challenge.

The main UVA CS curriculum does not cover much in terms of web-based attacks. Using information from only the required courses would make solving this challenge extremely difficult, if not impossible. SQL injection is discussed in Database Systems, Introduction to Cybersecurity, and Network Security, and the information provided in any one of those classes is enough to make headway on this problem. In particular, the knowledge of SQL commands and processing web input described in Database Systems was helpful when tackling this challenge.

2.5 Forensics

Forensics challenges involve discovering information left behind in files, or sometimes, in entire systems. Forensics is more about discovering information than about keeping software secure, but knowledge of digital forensics can be important as it helps software engineers know the potential implications of having certain information stored in their products. For example, if a password is stored in plaintext somewhere on a system, someone skilled at digital forensic analysis might be able to find that password if they had access to the system.

In the ImaginaryCTF competition, the challenge provides a supposed PDF file for the competitor to examine. Using the terminal command *file* reveals that the file is an executable, not a PDF. When run as an executable, the file outputs the first part of the flag. Looking plaintext in the file reveals a fake flag, but also a second part of the real flag. Searching the file for any additional embedded files reveals that there is a zip file in addition to the main executable. Correctly unzipping this embedded zip reveals the third and final part of the flag. Competitor Adragos [1], a member of the CTF team WreckTheLine, provides a detailed writeup on solving this challenge.

Digital forensics is a topic that is not discussed in most computer science curriculum. While the Introduction to Cybersecurity course does discuss forensic analysis, the type of analysis and tools that are discussed are not helpful in solving this particular challenge. Most CS graduates would likely be able to find at least one part of the flag, but to find all three parts would require extensive searching or prior forensics experience. Cybersecurity courses, at least for this specific challenge, would not provide much of an advantage.

2.6 Miscellaneous

Miscellaneous challenges are, as one would expect, challenges that do not fit cleanly into any of the previous five categories. Many challenges in this category are not entirely cybersecurity related and are instead general CS challenges. As such, examination of this category is less relevant to assessing engineers' software security knowledge; nevertheless, a miscellaneous challenge from the ImaginaryCTF competition will be discussed to determine how cybersecurity education helped with completing it.

The challenge this section examines involves communicating with a network socket to solve math problems. The challenge attempts to disrupt some solution strategies by sometimes sending code that will crash the competitor's program (and potentially the computer) if run. With some basic knowledge of how to communicate with a socket and how to parse input, this challenge is relatively simple. Once again, Github user s-3ntinel [6] provides a full description of the challenge, as well as a working solution.

This problem could certainly be solved by any UVA CS graduate. The curriculum of required courses is more than enough to complete this challenge. One advantage, however, is provided by the Network Security class: knowledge of how to communicate with network sockets programmatically. This knowledge is not too difficult to acquire for any CS student, but already knowing it helped speed up the completion of this challenge.

3 Recommendations for UVA Cybersecurity

Overall, and perhaps unsurprisingly, cybersecurity electives provided a major advantage in solving the CTF challenges. Many challenges could be solved using only knowledge from required UVA CS courses, but it would take much more time and effort to do so. Various cybersecurity electives were especially helpful for certain problems, such as UVA's Defense Against the Dark Arts, Network Security, and Database Systems courses. However, in general, the Introduction to Cybersecurity course was more than enough to make quick headway on these challenges. With the general information provided in that course, searching for additional resources becomes easier, making the challenges much more manageable.

Thus, it may be worth considering including the Introduction to Cybersecurity course, or at least some of its topics, in the required CS curriculum. The information provided in the course is perfect for software engineers to have in mind when developing secure products. Then again, the current required curriculum provides most of the necessary knowledge to solve cybersecurity problems, and simply lacks cybersecurity terms and contexts that help with security design. More research may be needed to determine the usefulness of cybersecurity knowledge to professional software engineers.

4 Conclusion

Cybersecurity is a continuously growing field, and therefore software engineers have a responsibility to have at least some knowledge of the topic. To that end, this report attempted to analyze the readiness of UVA CS graduates to face cybersecurity challenges. Using a single CTF event as a metric for whether more cybersecurity topics should be included in CS core curriculum, however, is certainly insufficient. This report instead argues that the analysis of the CTF event provides ample reason to continue

research. Specifically, studying how much professional software developers value cybersecurity knowledge would provide insight on how important the topics are at an undergraduate level. Ensuring that new software engineers are ready to face problems in the real world of software development is essential to guarantee the quality and security of future software products.

REFERENCES

[1] Adragos. 2021. Chimaera. (July 2021). Retrieved August 5, 2021 from https://ctftime.org/writeup/29578

[2] Ethan Oh. 2021. It's Not Pwn, I Swear! (August, 2021). Retrieved August 5, 2021 from https://github.com/babaiserror/ctf/tree/main/%5B210723-27%5D%20ImaginaryCTF%202021/It's%20Not%20Pwn%2C%20I%20Swear!

[3] s-3ntinel. 2021. Stackoverflow. (July, 2021). Retrieved August 5, 2021 from https://s-3ntinel.github.io/ctfs/imaginary_ctf2021/pwn/stackoverflow/stackoverflow.html

[4] s-3ntinel. 2021. Lines. (July, 2021). Retrieved August 5, 2021 from https://s-3ntinel.github.io/ctfs/imaginary_ctf2021/crypto/lines/lines.html

[5] s-3ntinel. 2021. Awkward_Bypass. (July, 2021). Retrieved August 5, 2021 from https://s-3ntinel.github.io/ctfs/imaginary_ctf2021/web/awkward_bypass/awkward_bypass.html

[6] s-3ntinel. 2021. Imaginary. (July, 2021). Retrieved August 5, 2021 from https://s-3ntinel.github.io/ctfs/imaginary_ctf2021/misc/imaginary/imaginary.html