**CSI 2334 F19 Project**

Section 02

Final Report

By Sam Hobbs & Jojo Cordova

Cynthia Fry

Due December 5, 2019

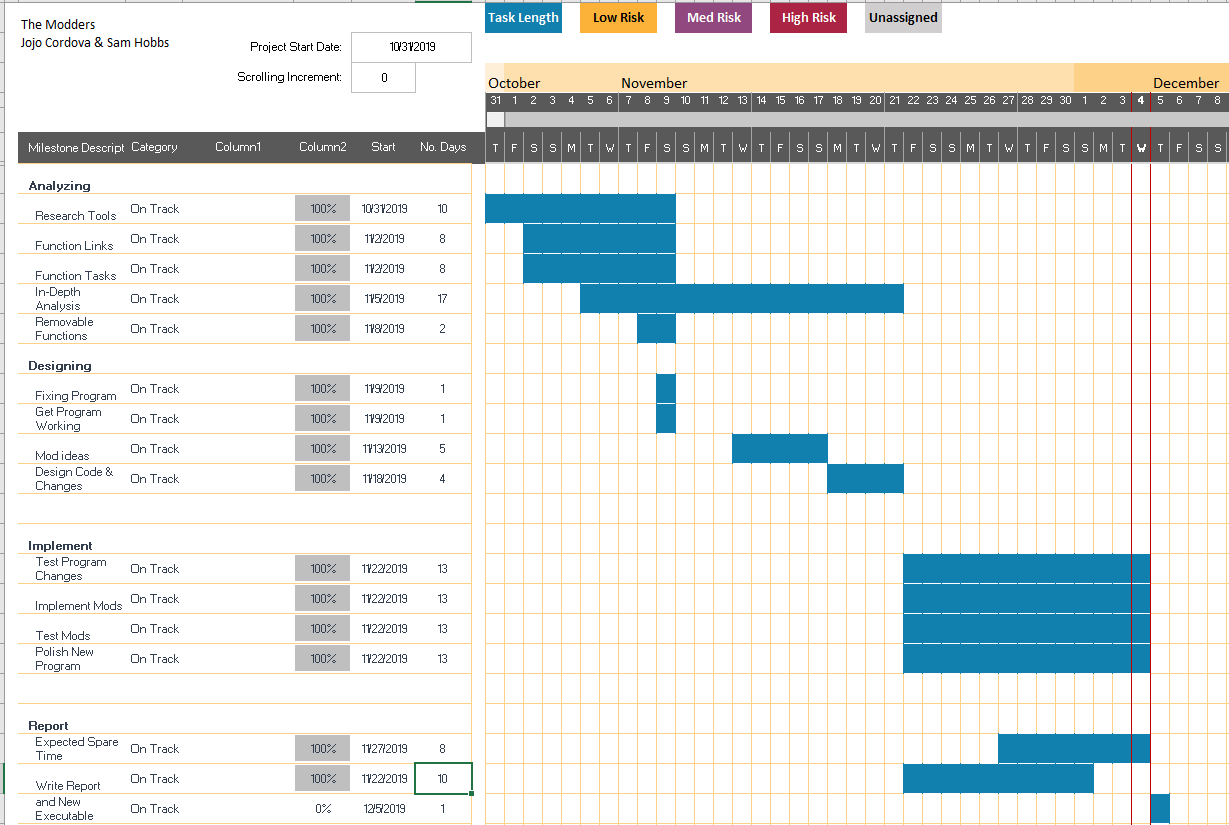
Statement of Work

Our team has been tasked with investigating the provided file “Tana1983.2334” and establishing the nature of the file. We investigated tools such as Ghidra, Snowman, OllyDBG, and ODA (Online Disassembler) that can disassemble and/or decompile the executable in question. This final report provides an analysis of the tools used during the investigation as well as their own advantages and disadvantages in the reverse engineering process. Our investigation revealed the game to be a game of “Snake” that attempts to open 1000 windows on pressing a key during the game loop. Modifications were implemented to disable the call to the console window loop and allow the game to run normally. No other malicious function was found. The file was then modified to implement our own changes to the gameplay, such as increasing the speed of the game upon increasing score and making the horizontal and vertical snake speeds consistent. Our final report, executable, and presentation will be submitted and presented on December 5th at 2:00pm.

Literature Review

Because of the necessity of computer systems in handling everyday tasks (such as financial transactions, message handling, etc.), they have become massive targets of malicious third parties. The need for specialists in cybersecurity is now larger than ever, and the necessity for skills in reverse engineering is growing at a rapid rate. In this case, reverse engineering encompasses the act of analyzing a program’s assembly code in an attempt to learn a files functionality, and more importantly how it works. Decompilers and disassemblers are examples of reverse engineering software as they take apart the files into assembly or a higher level language. One such decompiler is NSA’s Ghidra, a free and open source software allowing programmers to reverse engineer code. Released in 2019 by the NSA to help form a new generation of programmers, it is notable in that it can work with several different architectures and simplifies the seemingly monumental task of reverse engineering. These capabilities make it comparable to IDA Pro, often regarded as the industry standard in reverse engineering. Thus, Ghidra has the potential to allow aspiring computer scientists to understand the inner workings of executables and other files by easing into disassembly with this free software.

Project Plan



Findings

Our first step was to establish a schedule to determine the nature of the executable, quarantine the malicious code if needed, and repair the file if it was indeed a game. Extra time was allocated for any hiccups that weren’t originally foreseen, along with time for personal modifications. To analyze the original executable, JoJo first used the VirtualBox tool to run the file safely in a closed off virtual machine setting. After a few trial runs of the executable, it was established to be a game of “Snake” that would trick the user into pressing a key to trigger its negative effects. At this time, the only information to be gathered regarding its malicious effects were that it would continuously open console windows with the message “YOU MADE A MISTAKE” if the original window was not closed quickly.

We first approached the task of locating the malicious code by using ODA to look at the code’s general structure. This tool was not very helpful with our search for the code in question, but it did help us establish a basis of how to interact with files in a disassembler. Shortly after, Ghidra was then used to view the code in a different light. Ghidra would end up becoming our workhorse for this project in regards to modifying and searching for code. Upon learning how to patch instructions in Ghidra, JoJo replaced the call to “mostImportant” with a call to “funkyFunc2,” which only contains a return statement. This would later be replaced with NOP instructions for efficiency and safety, as it is unknown if “funkyFunc2” or any other seemingly innocent functions unknowingly affected the program. With the threat removed, Sam spent some time parsing the code and relayed information to JoJo regarding some new findings. With these findings, JoJo and Sam began to design and begin the process of implementing a few modifications for the game.

For the first mod, Sam noted that the game controls the speed with a call to the “Sleep” kernel. The data in the kernel is used to control the games refresh rate and therefore its speed, using the ECX register. Sam implemented the speed increase mod by moving several commands to make room for SUB ECX, dword ptr[rbp + -0x8], which decreases the sleep cycle by the quantity of score and thus speeding up the game.

For the second mod, Sam rewrote the procedure that checks the direction of the snake by exchanging the originals snakes face (arrows in this case) with the ‘0’ character to give it the feeling of a wide mouth. This left enough room to implement instructions that would make the speed of the snake more consistent, no matter the directions it turns.

A minor revision was implemented to exchange the walls with the characters ‘J’ and ‘O’ instead of the standard W. Along with this, an Easter egg was placed into the game by JoJo using a repurposed version of the original malicious functions. The repurposed versions contain the standard instructions up until “sidRich” which now opens only 4 windows before saying “Just kidding!” to the screen. These screens can only be seen if the player loses with a score less than 6.

Most of the revisions and their explicit details are explained on Sam’s GitHub page (username is Fireman89, link at the bottom).

Comparison of Tools

The three tools were efficiently used along with some tinkering with two others. VirtualBox, ODA, and Ghidra were our most used pieces of software while OllyDBG and Snowman were used for at most an hour each before being tossed. From our understanding, Ghidra proved the most advantageous for our task but that does not discount the pros of the other decompilers.

VirtualBox allowed us to see the executables natural function in a controlled setting without putting the main computer (in this case JoJo’s machine) in harms way. It was very advantageous in this regard as it gave us a jumpstart in the nature of the program along with a clear path to look for in regards to how the program would act. In this project we only used one virtual machine due to the use of others being deemed unnecessary after the file had been established.

ODA is an online disassembler (but not a decompiler) that displays a webpage of the file’s assembly instructions being derived from the higher level language. Its main advantage for us was allowing an overall view of the assembly instructions at play along with the naming and slight understanding of the higher level functions in play. In the case of this disassembler however, that was all that it could be used for as to our understanding there was no way to patch instructions. For future reference, those looking to reverse engineer should look to Ghidra or other decompiler software, but ODA is still useful for those who are barely beginning their understanding of reverse engineering.

The main tool, Ghidra, is a disassembler and decompiler that allowed us to directly change the assembly functions called in the executable. Other features include the ability to have several windows open at once, from assembly and decompiled code all the way down to a hex editor. Viewing the code from several different angles side by side boosted our understanding of the program at a fascinating rate. Its main advantage however, is the sheer amount of free reign that it gives the user in regards to patching instructions on the fly and editing specific bytes of hex. Although in some cases it did lead to issues in regards to using too much object code or too little initially, even a novice could quickly learn the necessity of keeping track of object code and how to work in new instructions correctly. Initially, we believed that the cumbersome nature of changing each individual instruction down to the last byte was a disadvantage, but after attempting the same tasks with Snowman and OllyDBG we sternly believe that Ghidra was the most efficient.

In regards to Snowman and OllyDBG, it was very much a matter of access when we searched for our main decompiler. Although Snowman was initially a refreshing look compared to ODA, we quickly realized that it had similar problems to ODA. The main issue was how difficult it was to understand what was being displayed to the screen, as the text was simply disassembly with no clear labels or reason to how the functions worked like Ghidra had. In the case of OllyDBG, we simply could not manage to get the software to work with any files we could place into it. On top of these issues, we also couldn’t understand how to use the software in general as even opening a file was a hassle along with its lack of response.

Lessons Learned

The main takeaway from this project is the reality of malicious executables. If someone really wanted to hide a tiny piece of code to either hinder or disable a machine, it would not take much for someone to truly disable several users or at the least annoy them due to their lack of knowledge regarding disassembly or safely handling unknown files. An example would be if our initial approach was to simply run the file on a random machine. Without any knowledge of the file, it could do as little as leave a random text file or as terrifying as wiping an entire machine. Aside from this, reverse engineering can also become a pain staking process. Although the act of replace the malicious code was no issue, searching for it proved to be much harder than anticipated. It was also far more difficult to put changes of our own into the program than it was to simply fix it. In addition, while Tana1983.exe only opened windows to annoy the user, other malicious software could do things far harder to detect or far more harmful. Nevertheless, we highly recommend using virtual machines for curious users to test unknown files, and disassemblers for those who wish to understand how their files work. For those who wish to learn, we recommend VirtualBox for testing and Ghidra for decompiling.

Summary / Conclusions

In conclusion, we determined the program to be a game of snake with a few lines of malicious code that opens one thousand console windows. A patch was implemented to bring the game into working condition along with several minor mods to add a little spice to the classic game. Our process can be streamlined to a simple set of steps which are as follows:

1. Test the file using a virtual machine to keep it quarantined.

2. If the file is established as safe after extensive testing, stop. Otherwise, either delete the file or use decompilers to search for the location of the malicious instructions.

3. Using the knowledge of said files actions and how they are triggered, parse for instructions related to that trigger. Ex: Pressing a key causes several windows to open. Search for instructions regarding keyboard hits.

4. After searching and replacing instructions, test in the virtual machine again. Repeat 4 and 3 until file is presumed safe.

After all is said and done, we can simply say that computer security is a tough game to those who don’t understand the nature of files. Users who stay ignorant may be susceptible to files such as Tana in the future and should learn safe and efficient methods if they intend to make steps into the computer science field, hobby or otherwise.

Appendices:

Citations:

Rohleder, Roman. *Hands-On Ghidra: A Tutorial about the Software Reverse Engineer Framework*. 2019. [*SPRO'19*](https://www.sigsac.org/ccs/CCS2019/) *Proceedings of the 3rd ACM Workshop on Software Protection.* p.77-8. New York, USA: ACM

Asghar, Muhammad Rizwan and Andrew Luxton-Reilly. Teaching Cyber Security Using Competitive Software Obfuscation and Reverse Engineering Activities*.* 2018. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. p.179 – 84. Baltimore, Maryland.

*Links to Project Resources:*

<https://github.com/Fireman89/CS-Project> (Sam’s GitHub page, contains all major revisions)

<https://github.com/charger716/jazzHands> (Jojo’s GitHub page)

<https://onlinedisassembler.com/odaweb/AJhUQzTk> (Link to online disassembler w/Tana1983.2334)

*READme.md on Fireman89 GitHub page*

Tana 1983.2334 --Original Version, 5 NOP commands replacing call to malicious code at address 00401613

Tana 1983(1.1).exe --First revision by Jojo -Changed malicious call at 00401613 to opcode E81F020000 (calling funkyfunc, which does nothing)

Tana 1983(1.2).exe --Minor, experimental revisions by Sam -Changed instruction at 004015f8 from Sleep(100) (B9640000) to Sleep(50) (B9320000), doubling game cycle speed

Tana 1983(JJ).exe --Initial Upload by Jojo on 11/26 (og w/malicious code)

Tana 1983(1.3).exe --jedi1 now consists of entirely NOP commands (to use for our own purposes), called at 00401613

Tana 1983(1.3.2).exe --Now speeds up on increasing score--moved commands from adressi 00401606 to 00401617, allowing room for SUB ECX,dword ptr[rbp+ -0x8], decreasing sleep cycle (default 100 in this version) by current score. 2 free bytes right after call

Tana 1983(1.4.6).exe --Rewrote direction change check (where it reads arrow input, now 00401610-61d) to have snake always have '0' as head, allow room for check slowing down speed (more sleep) for vertical directions (from 004016a1-b2)

Tana 1983(1.4.7).exe --'0' is now the default head of the snake

*Timeline*

**Tuesday October 29, 2019 –** Tana1983.2334 is released.

**Thursday October 31, 2019 –** JoJo and Sam join together and create The Modders project group.

**Saturday November 2, 2019 –** First meeting, first report is delivered. Basic roles are established, first viewing of the code using *ODA – The Online Disassembler*.

**Monday November 4, 2019 –** *VirtualBox* is used to safely run the file in a controlled state and figure out its contents. It’s now confirmed that the executable is a game of “Snake” that when ran seems to be normal, but if any key is pressed then 3 messages will be displayed before 1000 windows will be opened. Each of these windows displays the message “YOU MADE A MISTAKE”.

**Tuesday November 5, 2019 –** Meeting with Cindy Fry. Second meeting in order to find where the malicious code is located. Location was not found but data strings were.

**Saturday November 9, 2019 –** *Ghidra* is first used. Malicious code is located.

**Sunday November 10, 2019 –** Malicious code is quarantined.

**Tuesday November 12, 2019 –** Second report is delivered.

**Wednesday November 13, 2019 –** Third meeting, mods are discussed along with roles.

**Monday November 18, 2019 –** Fourth meeting, basic mods are decided. Minor ones include changing game end messages and changing game visuals. Major includes changing game speed to increase as pellets are consumed, and change vertical speed to match the visuals of horizontal speed.

**Thursday November 21, 2019 –** Fifth meeting, roles for final section are delegated.

**Friday November 22, 2019 –** Work on final mods and final report begins.

**Monday November 25, 2019 –** Sixth meeting (over the phone), roles are re-delegated. Work on first major mod begins.

**Wednesday November 27, 2019 –** Sam completes the main mod: changing base game mechanics. The speed of the game increases as the pellets are consumed. Final presentation materials are gathered.

**Sunday December 1, 2019** – Sam completes second mod: changing base game mechanics. The snake’s head is changed from an arrow depending on direction to always a ‘0’, reduced vertical speed of snake to be physically closer to horizontal speed

**Monday December 2, 2019** – Sam begins work on the final report; seventh meeting between team members

**Wednesday December 4, 2019** – Easter egg and minor wall mods are implemented by JoJo. Final presentation materials are completed. Sam and JoJo finish the final report.