Friday, May 4th, 2018 (extended)

CS 460 - Security Lab
Final Project - Reversing

Daniel Hoynoski
Net ID: hoynosk2

Ever since I was little, I have always enjoyed a good puzzle. Even if they frustrated me to the point of throwing them at the wall, my mind never left the game. Constantly idling within, these relentless puzzles took up a fight with my brain and pummeled it into submission. I was as stubborn as the puzzle was and I needed to know the answer. As I got older and embarked into the realm of Computer Science, Reverse Engineering (i.e. reversing) or the notion of bending software to your will intrigued me immensely.

My final project for Security Lab (CS 460) during the spring semester of 2018 was to build more upon my knowledge base of Reverse Engineering. I saw this as time to hack my way through a few "Crack-Me" applications or applications created for the sole intent of being reversed. I started off on my endeavor with an easy "Crack-Me" example, but then soon graduated to more advanced and time-consuming ones (e.g. "Keygenme").

I first began to familiarize myself with Olly Debugger or OllyDBG. Even though we had a few labs during the semester that had the option of utilizing OllyDBG, during these sections I mainly stuck with IDA Pro and a few Linux-specific variants. But after getting well acquainted with my new environment through trial and error, I dove into some "Crack-Mes."

The first "Crack-Me" I took a crack at (pun intended) was a simple "enter the correct password" application. During this exercise, I familiarized myself with referenced strings and intermodular calls. Because the password was not an obvious string to find quickly in the referenced strings list, I studied mostly the intermodular calls for this example. And after finding the use of a String-Compare call, I was able to find the correct solution.

For the next two "Crack-Me" examples, I chose two serial key requiring exercises or Keygenmes. In the first example, I decided to opt for an easier route of finding one solution or self-keygening, than to write the actual Keygen. I chose this method as a way to get my feet wet and get a "bronze metal" (as noted in the readme documentation). But I soon ran into an exercise that was taking hours upon hours to finish. During this one particular exercise, I was beginning to understand that reversing is not a process one can rush. But much more importantly, the requiring time of any given project is controlled in a large part by the tools a reverser has. For this example, I learned the hard way that reversing every function is sometimes a bad use of my time. Through further research and tool collection, I ended my peril, for

example, of reversing a long MD5 hashing function with a tool that told me the "Crack-Me" was using MD5 and just monitoring the output before diving deeper. Treating functions as Black Boxes can sometimes save time when predictable outputs are seen.

After completely three Crack-Me exercises, I set my sights on a game I had been wanted to claw my hands into. The game was called *Golf King*, but after the servers shut down in 2007, only the nagging dialog messages of being unable to connect to a ghost server were left. You may be asking yourself:

"Why a golf game?"

I honestly don't have an answer to this, as I utterly hate golf in real life. However, maybe it was the cartoon-ish graphics or the idea of not being able to play something that motivated me. In the end, I was determined. But until now, I never had the opportunity to invest a large amount of time into trying to revive at least a fraction of it. That all changed now.

I started off with two cracks into the game, but I won't discuss those here, because in the grand scheme of things, they do not matter (see further documentation). I set my eyes finally on reversing the encryption protocol as all the option files that came with the game were encrypted with the ".enc" file type. I spent several days thinking about how I might approach this problem. I came the conclusion that the game has to decrypt these files at some point and therefore, I can re-create this decryption algorithm in C. After loading the game into OllyDBG and looking at the intermodular calls, I noticed that it was using Microsoft's CryptoAPI, a heavily documented and assessable API. Breakpointing all these function calls allowed me the chance to reconstruct the function flow path environment and view parameter options specific to the game. After several sleepless nights and learning some Windows programming, which is a beast in itself (i.e. Windows constants, what the hell is this?), I was able to reconstruct the decryption algorithm, thereby finally giving me the ability to view all the encrypted files.

Surprisingly, because all the encryption/decryption operations are taken place within Microsoft's CryptoAPI, the API only requires a one-time use key in order to initialize the Crypto environment (i.e. the API handles all

public/private RSA keys). Therefore, I discovered I was not only able to decrypt the files, but re-encrypt them too. And after looking through all the encrypted files, I discovered some debugging options that allowed for a local game to be setup with Computer AI. SUCCESS!

Although this project has seriously exhausted me, it has also gotten me extremely excited for what I can learn next. Reverse Engineering is undoubtedly a field that interests me even more so now. And I am truly excited for what I can do next (i.e. legal stuff, of course!).

(Note: I have included all of my code in this ZIP, but it is also up on the Github too)