Erik Sommer
12/14/2011
SE-4930

Article Review 1 – Obfuscation: The Hidden Malware

For the first article to review, I picked one from the previous issue of IEEE's *Security and Privacy* magazine. Entitled "Obfuscation: The Hidden Malware", authors Philip O'Kane, Sakir Sezer, and Kieran McLaughlin go in depth on how malware is hidden and the common techniques used to detect it. The article itself is located at this URL: http://doi.ieeecomputersociety.org/10.1109/MSP.2011.98

Malware is a huge issue, and one that costs companies millions of dollars. According to the article, the classical technique for detecting malware involved looking for signatures. Malware was static code, and as such, had a definite signature or pattern of behavior. Anti-malware companies would build up a database of malware signatures as more was discovered. Their malware scanner would then connect to the database and compare files against the signatures. If there was a match, that file would be flagged as malware.

For quite a while, that technique worked. But just as we're at an arms race in the real world, the cyber world is constantly going back and forth. Malware writers eventually found ways around the scanners that look for signatures. While there are many different techniques that can be employed, the article focuses on 3 of the largest: packers, polymorphism, and metamorphism. Packers do exactly what they sound like; they pack the data. Originally developed to minimize data and bandwidth usage, they are now used by malware writers to obfuscate their code. The packed, or compressed, data is essentially in an encrypted form. The only way to see what the actual contents of the file are is to unpack it. In order to unpack it, one must know the algorithm used to pack it in the first place. Some scanning tools take advantage of this fact and looks for signatures that the actual packing tool leaves. However, with so many different types of packing available, it's difficult to build a database of signatures for all of them. One small change in the code causes large changes in the packed file, which generates an entirely different signature.

While packing is an effective means of obfuscating code, some malware writers take it a step further with polymorphism. Using a polymorphic engine, a piece of malware can change the code that it contains, creating a new signature every time it runs. The main steps for the engine are as follows:

1. Gain entry – The polymorphic engine gains control of the CPU
2. Execute transfer function – The polymorphic engine uses the polymorphic key to convert the malware code into executable opcodes.
3. Load executable – The executable is written into memory
4. Run – The executable malware is ran
5. Generate new key – A new key is generated for the engine

6. Execute inverse transfer function – The polymorphic engine essentially encrypts the malware code with a new key, creating a different signature

With the malware code changing on every execute, it makes it difficult for scanners to pick up. However, this type of malware still executes the same opcodes on the CPU, making it vulnerable to scanners who look for signatures at that level. Therefore, malware writers have come up with a new method of obfuscation.

The final method of obfuscation described in the article is called metamorphism. This is by far the most interesting and advanced method discussed. Metamorphic engines actually change the opcodes of the malware itself. Much like polymorphic engines, metamorphic engines follow a 5 step process:

1. Disassembly – A disassembler decodes the opcodes
2. Shrinking – A shrinker compresses the disassembled code to prevent code growth
3. Permutation – A permuator reorders instructions by randomizing their order and connecting them with jumps. It also does register and opcodes substitution.
4. Expansion – An expander takes a single opcodes and expands it into several instructions. It may insert nops or other frivolous instructions.
5. Assembly – The program is assembled and prepared to execute

All of this happens every time the program is run. This level of complexity does create larger files. In fact, the article states that 90% of the code in malware that implements a metamorphic engine is for the engine itself. With so much changing on each execution, however, the amount of signatures that a piece of malware can have are limitless. Smarter anti-malware scanners will modify for high CPU usage used when these malware programs are executing, but they are still very difficult to detect. The article concludes with some possible solutions, mostly based on policies and restricting and controlling access.

Through reading this article, I realized how much some malware writers are willing to do to hide their code. A program that essentially rewrites itself is a very scary thought, and they are able to do it with KBs of code. It seems like malware writers are pushing the envelope with new techniques and exploits to take the most advantage of the hardware we have today. I do wonder where all the funding for this level of malware comes from, however. Probably it's an underground organization where writers sell their "product" for others to use. But now that we have code that can re-write itself, what's the next step? Malware that searches for new vulnerabilities and adapts to them on its own? Maybe that's already out there. Reading this article has shown me that malware isn't just a kid's game anymore. It's serious business and at a more complex level than I previously thought. I only hope that the malware scanners can keep up with the writers.