Shellcode

Alison Buben

COSC 480 – 001

480 Reaction Paper

Bibliographic Citations:

http://www.phrack.org/issues.html?id=14&issue=49
http://portal.acm.org/citation.cfm?id=1920305
http://portal.acm.org/citation.cfm?id=1653725
http://portal.acm.org/citation.cfm?doid=1920261.1920310
http://www.phrack.org/issues.html?issue=56&id=8#article
MLA Format in Appendix

Brief Summaries of Citations:

Smashing the Stack for Fun and Profit [4]

This is the quintessential article on shellcode. There are so many valuable parts in this article, but overall, I would have to say that it gives an amazing tutorial on how buffer overflows work in regard to writing shellcode. Many of my newer articles do not go into detail about the basics because this article has already explained everything so well. There are fundamental terms defined here including: a buffer, static and dynamic variables, memory, and stacks. There are also several examples using C and Assembly.

Comprehensive Shellcode Detection Using Runtime Heuristics [5]

An interesting feature about this paper is that a few students in the class actually used this article to form their discussion questions. In this journal, there is a brief explanation on shellcode, and then it goes on to talk about other ways shellcode can be used. The valuable part of this article is all the ways the authors showed how to detect shellcode. Polymorphic shellcode was discussed here and how the authors' network detector was going to work. Information about computer architecture and examples using DLLs was also mentioned.

English Shellcode [3]

This article was very helpful for my presentation, as it gave me terms such as NOP and Sled and explained ways that attackers would help get their shellcode up and running. It also talked about vulnerabilities and code-injection attacks. A valuable part of this article was identifying the attacker's goal and other information about the program counter. Metasploit was mentioned here as well as character encoding. The article mainly focused on making “English Shellcode”[3] or making shellcode look like english, and has a nice tutorial and explanation.

Heap Taichi: exploiting memory allocation granularity in heap-spraying attacks [1]

Although the previous article did quickly mention other ways attackers could exploit a system besides buffer overflows, I wanted an article that discussed another way to help have a better knowledge about shellcode attacks. This article mentioned NOP sleds again, and talked more about detection of shellcode in respect to heap spraying. This article was valuable because it talked about new attacks and shellcode obfuscation. It showed how a new attack might not be detected and other ways shellcode is detected.

SMASHING C++ VPTRS [2]

Since my other web source was not very current, I wanted another article from the web that showed some of the newer information about shellcode. This article was great because it showed how shellcode could be written in C++. There are many examples of shellcode being written in C++ and some more information on buffer overflows. An important part of this article was how GCC was used to show the dump of assembler code.

The summarized discussion that occurred:

After my presentation was over, Dr. Shubra had several questions. First, he wanted to know: “what is the use of a buffer in an OS”? I explained what a buffer was, and how it was the method attackers used to get their shellcode into the target system. I also gave an example that a buffer could be a form in a website, however that did not seem to be the answer he wanted and he later explained why. Next, I was asked to explain what an IDS was, and I replied with Intrusion Detection System. Perhaps I needed to have that in my powerpoint. Another question was “where [would] this fit into the Computer Science Course?” Initially I suggested either COSC 316 because it is an introduction to security course and then COSC 362 because many of the tools used are found in Linux. This brought up the question: “Should assembly be required?” Although knowing how to write assembly is not required to understand the concepts covered in

my presentation, knowledge of registers, stacks, and other related concepts is needed. The last question before student research questions were asked was more of a suggestion. I was to talk to Dr. Shumba and Dr. Oblitey about what other topics the computer courses should add.

When I was reading over all the questions my fellow students had asked, I found that they were easily grouped into three categories. These were: detection, prevention and miscellaneous. While I may not answer all of the questions, I intend to deal with each one, starting with the miscellaneous group first.

One of my sources, "Comprehensive shellcode detection using runtime heuristics", discussed the term "shellcode metamorphism"[5], which another student wanted to know more about. The article said: "mutating the actual instructions of the shellcode before launching the attack"[5]. Another question was: "Do current smart phones still have vulnerability to shellcode? If so, how were they dealt with? If not, how should it be dealt with in the future?" Since shellcode exploits a vulnerability and there is most likely no system without vulnerabilities, I would have to say that smart phones are most likely very vulnerable to shellcode attacks. However, this can be mitigated by trying to make smartphones more secure. Lastly in the miscellaneous group, regarding the question "since shellcode is written in assembly or a low level C language, it limits itself to a unstable environment, is this favorable?" I would say that although any limitation is not favorable, this does not mean people cannot write destructive and dangerous programs.

The next group of questions all deal with detection in some way, and this is where some of my journal papers really helped. First, there is the question "how is it possible to detect self-modifying shellcode?" The article "Heap Taichi: exploiting memory allocation granularity in heap-spraying attacks" describes different ways this can be done. Next, "are there any

commercially acceptable IDS currently available that can detect polymorphic shellcode?" is a difficult question to answer. Since the student asks about commercial IDS which I cannot test out since I would need to probably pay for them, I would only have to believe that something must exist.

The articles "Comprehensive shellcode detection using runtime heuristics" and "Heap Taichi: exploiting memory allocation granularity in heap-spraying attacks" discuss ways the groups designed to detect shellcode, so detection is being worked on. The same article also would lend good answers to the next two questions that deal with " how do you detect polymorphic shellcodes?" and "how is shellcode usually detected in a network channel"? One of my sources, "Comprehensive shellcode detection using runtime heuristics", talked specifiably about a student's question "can using a virtual machine so at runtime detection of shellcode attacks be 100% if so will overhead of the system be worth it to stop attacks that are unknown." I would like to add that no detection rate can ever be 100%. However, some detection methods may be able to get very close to 100%. When considering overhead, it depends on how critical it is to detect problems. If detection is a must, then the closer to 100% the better, with lesser consideration for the overhead.

The third area in which students had research questions was the idea of prevention. I had a question that started out with "the term hackers is almost synonymous with searches for shellcode, are there any ways to shut down remote shellcode attacks or set up your PC so that shellcode cannot be run unless approved by the user". Because nothing can ever be 100% secure, I would just recommend to try and be as secure as possible, which involves layered security protections and reducing your vulnerabilities as much as possible. The next question asks "if something as wide spread as msword can be targeted by shellcode injections, how can

we prevent them?" In addition to trying to be as secure as possible, this problem might be dealt with by applying patches to correct the vulnerability in msword. Another question asks "is a buffer overflow the most common way that shellcode is executed? If so, what measures can be taken to prevent this?" Although in this current age I cannot say for sure if buffer overflows are the most common, but they are well known and used massively. Also, I again suggest that people try to be as secure as possible. The last research question from a student is: "when talking about remote shellcode what would be my best action to take to prevent me from suffering an attack like this?" Here I would just like to state how important it is to try and practice good security procedures, have a good layered security plan and try to reduce vulnerabilities.

My reaction to my research:

When I was given the chance to do a presentation on a computer science topic, I knew exactly what I wanted to do. Several topics in security had my interest, and I took this opportunity to learn even more about one specific topic that interested me. This was a great topic, as it helped me become more knowledgeable in several aspects of computer security. I really loved my topic, and was very happy to talk about security concepts.

How I will use what I learned:

Especially for my field of Information Assurance, I found the topic very helpful in gaining more knowledge towards my future. Now I know more details of how attacks exploit vulnerabilities and how I need to protect against these vulnerabilities. I intend to become even better at detection and prevention and learn more about shellcode. While I am learning more about shellcode, I would also be increasing my skill with various other security concepts. These can include debuggers and assemblers, since those are extremely powerful tools.

For a research project, since I know that I want to get a Master's Degree in computer

engineering, I would definitely do something dealing with hardware, perhaps “how to detect malicious code on hardware”. This could include shellcode, or some exploit on the hardware level.

Glossary:

Buffer: “contiguous block of computer memory that holds multiple instances of the same data type” [4]

Dynamic Variables: “[variables] allocated at run time on the stack.” [4]

GCC - GNU Compiler Collection

IDS: Intrustion Detection System.

NASM – netwide assembler

NOP: “Non-operation” [3]

Sled: NOP bytes that lead to Shellcode

Static Variables: “[variables] allocated at load time on the data segment” [4]

Appendix:

1. Ding, Yu, Tao Wei, TieLei Wang, Zhenkai Liang, and Wei Zou. "Heap Taichi." *ACM Digital Library*. 201. Web. 2 Feb. 2011. <http://portal.acm.org/citation.cfm?doid=1920261.1920310>.

2. Rix. "SMASHING C++ VPTRS" *Phrack Magazine*. Phrack, 2000. Web. 2 Feb. 2011. <http://www.phrack.org/issues.html?issue=56&id=8#article>.

3. Mason, Joshua, Sam Small, Fabian Monrose, and Greg MacManus. "English Shellcode." *ACM Digital Library*. 2009. Web. 2 Feb. 2011. <http://portal.acm.org/citation.cfm?id=1653725>.

4. One, Aleph. "Smashing The Stack For Fun And Profit." *Phrack Magazine*. Phrack, 1996. Web. 2 Feb. 2011. <http://www.phrack.org/issues.html?id=14&issue=49>.

5. Polychronakis, Michalis, Kostas Anagnostakis, and Evangelos Markatos. "Comprehensive Shellcode Detection Using Runtime Heuristics." *ACM Digital Library*. 2010. Web. 2 Feb. 2011. <http://portal.acm.org/citation.cfm?id=1920305>.