Book Review: Gray Hat Python

Review by Ryan Linn, CISSP, MCSE, GPEN

“Gray Hat Python” by Justin Seitz, one of the latest releases from publisher, No Starch Press, focuses on using the Python programming language for reverse engineering. This book is subtitled “Python Programming for Hackers and Reverse Engineers” which is fitting as Justin is a member of Immunity Security, makers of the Canvas penetration testing platform and the Immunity Debugger. The foreword by Dave Aitel, Immunity's CEO, is an excellent introduction to why the content of this book is important. It focuses on the short time span that is required from discovery of a bug to exploit, and the necessity for flexible, fast, and collaborative vulnerability discovery and exploit development. Dave does an excellent job in setting the tone for why the information in the book is relevant and what the drive is for these types of tools in the industry.

Download 2 Free Chapters Below

del.icio.us

Discuss in Forums {mos_smf_discuss:Book Reviews}

Download Chapter 2: "Debuggers and Debugger Design" and Chapter 4: "PyDbg: A Pure Python Windows Debugger"

The introduction by Justin lays out why he wrote the book, and how to approach the content it contains. He focuses on how there is very little centralized information on using Python for hacking, and, while it is extremely common to use it in that capacity, the goal of this book is to organize some of this information to get others started using Python as a hacking and reversing platform. Definitely read the introduction before tackling this book, as there are recommendations for how to approach the chapters based on one’s skills in reversing and Python.

The content of “Gray Hat Python” begins by setting up the development environment that will be used throughout the book. The book leads the reader through setting up the initial Python libraries, an IDE, and pulling down additional required libraries. The second half of chapter one lays out how to convert C data structures into Python using ctypes, which is a critical piece of information for working through the rest of the book. The setup from chapter one along with the source from http://www.nostarch.com/ghpython.htm should give the reader all the tools needed for the rest of the book.

Chapter 2, “Debuggers and Debugger Design,” goes through the theory behind debuggers and gives some background into what registers and breakpoints are and how they are used. Of all the books I’ve either read or reviewed, this is one of my favorite explanations of how registers are used and the purpose of each register. Mr. Seitz went into great detail and did a fantastic job explaining some of the nuances of debuggers that many other references have a hard time explaining. The explanation of stacks is good, and there is also an introduction for hard and soft breakpoints and when each should be used.

The training wheels come off for Chapter 3, “Building a Windows Debugger.” Seitz guides the reader through the evolution of a debugger, starting with building the framework that is required to perform the basic task of launching an application within the debugger. Plenty of scaffolding is laid out with an expert explanation of what the various Windows data structures are used for and finally, by the end of the first exercise, launching an application from within the freshly built debugger. From there, the debugger evolves new functions such as the ability to attach to a currently running application and retrieving registers for all of the running threads. Chapter 3 finishes by completing a very basic Python debugger by incorporating the ability to set and catch hardware and software breakpoints, rounding out both Chapter 3 and the reader's introduction to debugger interaction with Windows.

Chapter 3 is a massive amount of knowledge and background and a great reference chapter for folks who are getting started. The code for most of the pieces is on the website. So while there is a tremendous amount of code, there isn't a requirement that all of it gets typed. Walking from one of the earlier code snippets through the final product will be good practice in setting up these fundamentals for folks that haven't worked much with Python or who want a more robust understanding of what's going on with the debugging process.

Now that the reader has some of the groundwork for programatically interacting with Windows processes, Chapter 4, “PYDBG: A Pure Python Windows Debugger,” incorporates PyDbg, the Python debugger part of the PaiMei Reverse Engineering Framework. The first activity works to extend a basic breakpoint handler to modify the arguments for the printf function in a sample application. The result is printing out random numbers in the application every time the breakpoint is encountered in the scripted debugger. While not completely practical, this is a very visible and straightforward introduction to extending a breakpoint handler. This relatively short chapter finishes out with an extremely useful debugging script which will hook dangerous functions within a test binary and allow information about why these functions caused a crash within the application by presenting snapshot data back to the user after the crash. This is something that would typically require significant work through an interactive debugging process.

The chapter on PyDbg was short, sweet, and practical. There’s a footnote in the last page of Chapter 4 that proposes trying the application against WarFTPD 1.65, a binary which has a lot of documentation regarding the stack overflow vulnerability. There is also discussion of how to fuzz this binary later in the book using Sulley, where some of these previous topics in the book will be tied together.

With a foundation laid for programmatic debugging, an additional tool is exposed in Chapter 5, “Immunity Debugger: The Best of Both Worlds.” As Seitz is part of the Immunity Debugger team, there is probably no better person to provide the introduction to the tool. Seitz walks the reader through a very brief introduction to the tool, and then immediately dives into the types of hooks that Immunity Debugger contains. Once the different types of hooks are enumerated, Seitz digs deep into the Python debugging-fu and demonstrates scripts to find exploit-friendly instructions using the Immunity Debugger python library. He then leverages the same library to do bad-character detection once an exploit and payload string has been developed. The chapter goes on further to discuss how to leverage Immunity Debugger to locate and generate shellcode to disable DEP within an exploit and to defeat anti-debugger techniques in malware.

Chapter 5 was a good introduction to Immunity Debugger. With the power of the Immunity Debugger, and the expert level of the author with this product, I do wish that this chapter was longer and had some more examples. Although the techniques in this chapter were great, I would have loved to have seen some more examples of the tools that the Immunity staff uses in their exploit creation and reverse engineering processes. The anti-debugger techniques were a great addition though, and a good practical example of the power of the debugger.

Chapter 6, “Hooking,” spends time on the initial concepts, adding in pieces of code to alter process execution in order to gain insight into what is happening within the process. The chapter starts off with soft hooking by creating a handler for INT3 breakpoints (which should be used sparingly to gather information which would be difficult to gather otherwise). The example used in this chapter is accessing the query string Firefox is submitting via SSL by directly looking at the memory every time a specific function is called. Implementing this soft hook with PyDbg every time a URL is submitted via Firefox, the query string is logged to a file. This information would require a complex application setup to obtain otherwise, but with relatively little code and some research, the data is readily accessible.

The chapter continues to discuss hard hooking with Immunity Debugger. Hard hooking gives the executing code a brief detour to run custom assembly code and then return to normal process execution without interruption unlike soft hooking. Seitz walks the reader through creating a hook that will log heap allocates and frees, a useful application for understanding how an application is allocating heap space. The code is fairly intense, but it is explained well and if you have a strong understanding of assembly language and are a professional exploit developer then this portion of the chapter will probably be incredibly useful. Unfortunately I am not skilled enough in reverse engineering or exploit development to understand when I would utilize this type of activity, but if I encounter a situation where I need to in the future, the second half of this chapter will be a good reference.

More advanced techniques are uncovered in chapter 7, “DLL and Code Injection.” This chapter digs into creating remote threads in programs after injecting remote code, allowing for custom code to run hidden within unexpected running applications. The techniques are well expressed with good examples of how these types of techniques could be used following the examples through to creating a backdoor in a process using Python and shellcode generated through Metasploit. Discussions of alternative data streams and the differences between DLL and code injection are very informative, and while this is a difficult subject to understand, Seitz presents the material in manner which was easy to follow. My only complaint is that there wasn't more of a discussion of the possibilities of DLL injection from a hacker/reverse engineer standpoint, but the shellcode injection discussion more than made up for it.

Seitz focuses on finding locations to run arbitrary code in “Fuzzing,” Chapter 8. The chapter begins with a run-down of a number of different types of vulnerabilities with quality explanations of each type of vulnerability. Once the vulnerability types have been enumerated, the chapter develops into an exercise in building a fuzzer in Python. Using PyDbg, Seitz walks the reader from a base structure that will open a file through a file fuzzer and then determine if a crash has occurred in the application that opens the mutated file. There is a good explanation of what features make up a basic fuzzer and how to fuzz within a single Python application using threads to monitor fuzzed file handler and log crashes. To understand what to do with the information discovered from the fuzzer and to understand where to head after this chapter will require some more advanced knowledge of exploitation, but with that background this chapter is a

useful background to understanding automated fuzzing techniques. The “Fuzzing” chapter would also be a great reference to get started on a custom fuzzer if special logic is needed that current fuzzers don't facilitate.

With a good understanding of how to build a basic fuzzer, Seitz pushes on to show the reader how to use a full-featured Python fuzzer called “Sulley” in the chapter by the same name. The “Sulley” chapter begins with a basic rundown of what Sulley is, how to install it, and the basic building blocks for creating fuzzing test cases. Seitz doesn't dally and gets right down to attacking WarFTPD and creating test cases for fuzzing FTP with an explanation of what each FTP command does, and then setting up the monitoring and networking necessary to track the attacks and log the successes. Details are explained about how to view progress on the web and sample output is provided to show what types of information can be gathered. This is a great chapter. Many fuzzing programs have a lot of background that is necessary to get started, and Seitz does a good job of explaining how to get up and running with Sulley. The explanation of how to setup test cases is good, and if you have a quality understanding of other network protocols, then this sort of setup should be easy to replicate.

Chapter 10, “Fuzzing Windows Drivers,” is not for the faint of heart. This chapter, as the title suggests, involves fuzzing driver functions which can lead to anything from discovering interesting privilege escalation bugs to repeatedly crashing your system. The chapter eases into the discussion by taking an application and trapping IOCTL functions, and then replacing some of the data being passed to the driver with random data to try to get crashes. This first step lets the reader get a brief glimpse of what is going on using Immunity Debugger and hooking techniques to modify the data.

Once the user-mode fuzzer is explained, Seitz walks the reader through a more direct route, fuzzing drivers so that user-mode applications aren't required. Seitz brings driverlib into the picture to demonstrate how to discover the inner workings of a sample driver to discover device names and IOCTL codes. He does this to show that a stand-alone fuzzer can be used to directly attack the driver. The examples are well explained, but this type of attack is sophisticated and definitely a step above the content in the rest of the book. This type of attack is recommended only for a VM as it is almost guaranteed to crash the system. While exploit developers are using driver fuzzing, it has gotten relatively little press. This chapter would definitely be a great reference chapter for the expert exploit developer, but is probably a bit over the head for the non-experts.

Chapter 11 takes a turn and moves to an application that is primarily used for static analysis, IDA Pro.

“IDAPython – Scripting IDA Pro” takes the reader through scripting utility functions for IDA Pro for both static analysis and the integrated debugger. Seitz helps the reader get all the pieces installed, before talking about some of the built in functions for IDAPython. Seitz presents interesting scripts to demonstrate the power of IDAPython for reverse engineering functions including ensuring potential dangerous functions are found and have breakpoints set for debugging and determining stack size of a function during binary analysis. By the end of the chapter, a descent sampling of the potential of IDAPython had been revealed. If there are tasks that are repeated while reverse engineering using IDA Pro, this chapter should provide a framework for incorporating those tasks into useable scripts.

The final chapter in the book, Chapter 12, “PyEmu – The Scriptable Emulator,” is one of the most intense in the book. Seitz leads the reader on a trek through a Python IA32 emulator named PyEmu which will allow a reverse engineer to interact intimately with a sample process and manipulate the way the application runs. One of the most interesting examples is using the PEPyEmu class of PyEmu in order to programatically unpack a UPX packed binary. This type of activity is extremely useful as it allows this commonly used packing function to be unpacked into the original binary, so that it can be statically analyzed without having to risk potentially infecting the computer doing the analysis. This chapter is very intense, and the UPX example is a great real-world application of the concepts. Once again, advanced reverse engineers and exploit developers will probably find a lot of use in this chapter, but it is well over the head of most.

Overall Gray Hat Python was a good read, but it wasn't what I expected. The book presented fresh topics and quality information on programmatic reverse engineering techniques with the first half of the book appealing more toward novice to intermediate reverse engineers, and the second half of the book with more advanced topics such as fuzzing and scripting IA32 emulators. As mentioned above, this book isn't for the faint of heart. After the first few chapters, many advanced skills are needed. Seitz will help the reader discover when he or she has found an EIP overwrite, but doesn't provide much guidance from there and discusses driver fuzzing but doesn't really have the next step. That seems to be the trend throughout the book, where small tastes of new techniques can be sampled, but these techniques never seem to come together for a culminating feeling of accomplishment.

What appears to be missing is an overall driving goal. An example is WarFTPD, which is covered in a few locations throughout the book. A great addition would be a start-to-finish approach on the WarFTPD binary by performing a number of the techniques demonstrated in the book to develop a working exploit, touching on the reverse engineering, fuzzing, and the exploit development capabilities of Python. For someone who is getting started in reverse engineering, then the first few chapters have quality explanations of reverse engineering and hacking topics with the rest of the chapters providing some great references later on. For a seasoned professional, the book provides some introduction to new techniques with the last 3 chapters holding the most value.

Ryan Linn, CISSP, MCSE, GPEN - Ryan is currently an Information Security Engineer at SAS Institute. Employed in the computer industry since 1997, he has held positions ranging from web developer to Unix Systems Programmer at a large university to his current position in Information Security. Ryan has been responsible for working with large scale deployments of various flavors of *nix, high availability web and database clusters, as well as for application programming in high availability environments. In the past few years, Ryan has incorporated Windows security into his responsibilities, and is now part of the team responsible for information security globally in one of the largest privately held software companies in the world.