I. INTRODUCTION

As the number of mobile users, digital applications, and data networks have increased over the past decade, so have the opportunities for exploitation. The modern world now depends on critical systems, networks, and data repositories that are not as secure as they should be. Network outages, hacking, computer viruses, and similar incidents affect our lives in ways that range from inconvenient to life-threatening. The Equifax Hack of 2017 unquestionably demonstrates the disastrous consequences of cyber attacks for people and organizations. In fact, the hack has been marked as the worst data breach in US history, having compromised the Social Security numbers of half the US population. Not only is personal information at risk, hacking can present financial consequences as well. In 2013, Amazon Services temporarily went down, losing $66,240 dollars a minute. That's approximately $2 million for the 30 minutes the services were offline.

The ongoing competition between security software engineers and hackers has propelled both increasingly sophisticated security systems and malware. One particular type of malware, called a rootkit, is notoriously good at hiding itself from security software even as it simultaneously extracts the victim's data. This summer, Dr. Karen Karavanic at Portland State University allowed me to work on her research project to develop a new type of security monitoring tool. The project, called Extensible, Performance-Aware SMM Runtime Integrity Measurement (EPA-RIMM), is focused on "detecting rootkits that compromise operating system kernels and hypervisors." It is a tool that utilizes a special feature of a computer processor called System Management Mode. My 9-week internship produced a simulator to determine the most efficient and scalable scheduling algorithm for security monitoring workloads.

II. BACKGROUND

A. Operating System Security Layers

There are multiple layers to a computer system, commonly called the operating system (OS) security layers or privilege rings. The highest and least secure layer contains the user applications and utilities, while the Kernel is the lowest and most secure layer. Below the Kernel is System Management Mode (SMM), a key component of EPA-RIMM. More specifically, "SMM is a special x86 processor mode that privileged software such as kernels or hypervisors cannot access or interrupt." SMM was originally created solely for mundane tasks, such as power throttling, hardware emulation, and system health checks. Recently, security experts have proposed its adoption as a security tool due to its due to hardware protections and high privilege. While these benefits are desirable, the time in SMM required to run security checks can degrade OS and application performance.

B. Previous Research

A previous study on this use of SMM at George Mason University is called SPECTRE. SPECTRE can "introspect a live operating system without relying on any underlying software, and provides a fast, transparent, secure framework for malware detection with a small Trusted
However, the previously mentioned undesired effects on an OS disrupt the user's experience enough to render SPECTRE impractical as a security tool. A paper presented in 2013 by my mentor, Prof. Karen Karavanic, and Ph.D. student Brian Delgado demonstrates the exact "performance implications of [SMM]." such as delays and inaccurate time accounting at the kernel and application levels. Karavanic and Delgado's current research works to reduce those performance implications.

One method of countering these implications is reducing the amount of time spent in SMM with each System Management Interrupt (SMI). Essentially, large security inspections are broken down into smaller tasks that must be scheduled into bounded SMM intervals. The appropriate scheduling of these tasks helps eliminate those performance implications, thus requiring a unique type of scheduling algorithm. To facilitate the design of the most efficient algorithm, a simulator was implemented to study and compare different types of algorithms.

III. METHODS

A. Learning Python

My initial assignment was to revise, edit and document the existing EPA-RIMM simulator. I spent the first half of my internship learning Python and general concepts of a queue, list, stack, and range. As my skills increased, I was asked to write and document a new simulator from scratch. I learned Python with the help Prof. Karavanic, other graduate students working on the project, and tutoring myself through Codeacademy.com and the official Python Tutorial. To practice, I used codingbat.com and practicepython.org.

My first taste of meaningful problem-solving and coding came with the assignment to build a Queue that would be implemented in the existing simulator. A Queue is a common abstract data structure that uses First-In-First-Out (FIFO), and whose interface consists solely of enqueue/append and dequeue/pop operations. My experience creating the queue helped me when I wrote my simulator.

B. Simulator & FIFO Scheduling Algorithm

The simulator is supposed to simulate the processing of security monitoring workloads which have been divided into smaller jobs. In EPA-RIMM, jobs are called "tasks" and groups of tasks are referred to as "bins." The simulator itself consists of three main components: A task generator, a queue, and a scheduling algorithm. The task generator creates tasks with three parameters: an increasing task ID, a priority of three, and a randomized cost. The randomized cost assigned to each task carries a random value between 1 and 1000. Those tasks are then appended to a queue, from which they are removed according to the implemented scheduling algorithm.

After I completed the simulator, Karavanic proposed that I test it with a simple algorithm that relies on FIFO queue. I conducted a study on the performance of that simple algorithm to produce a baseline against which we could compare other algorithms. The FIFO queue algorithm places tasks into a bin until the next task will not fit, then closes and releases that bin and starts over. Other parameters consist of the maximum cost per bin, the maximum number of bins, and the maximum number of tasks. The maximum cost per bin is a predetermined number specified by the user. It served as a limit to the number of tasks that could be placed in each bin in terms of cost. In the study, the parameter of maximum cost per bin was given values of 3000, 4000, and
5000. The number of bins was run with values of both five and ten, while the number of tasks remained constant at 100.

IV. RESULTS AND FUTURE WORK

My study confirmed that the FIFO scheduling algorithm is inefficient and also established a baseline for future algorithm performance studies. The algorithm is inefficient due to its simplicity because with each bin completed, much empty space remains that could be filled by other smaller tasks farther along the queue. As seen in the chart to the right, between about six to 13 percent of tasks are wasted with each bin creation, depending on the maximum number of bins (five or ten, shown in blue and orange respectively). The ideal algorithm would not waste any bin space.

In my research, I tested the performance of a FIFO scheduling algorithm. The next step is to extend my simulator to accommodate other types of scheduling algorithms, thus permitting the comparison of multiple algorithms. Other algorithms, for example, could select randomly, or according to the priority of the task. Comparing multiple algorithms is required to accomplish the larger goal of determining the most efficient and scalable scheduling algorithm for security monitoring workloads.

V. ACKNOWLEDGEMENTS

I would like to thank my mentor Prof. Karen Karavanic for providing me the opportunity to do research and guiding me in my project at Portland State University. I would also like to thank the other members of the EPA-RIMM project: Brian Delgado, John Fastabend, Cody Shepherd, Tejaswini Vibhute, and fellow summer interns Stephen Rivera and Ozzy Sanchez-Aldana. My research was supported by a Distributed Research Experiences for Undergraduates (DREU) award from the Computing Research Association - Women committee.

VI. REFERENCES