MALWARE EARLY-STAGE DETECTION USING MACHINE LEARNING ON HARDWARE PERFORMANCE COUNTERS

by

ANCHAL RAHEJA

THESIS

Submitted in partial fulfillment of the requirements

for the degree of Master of Science in Computer Science at

The University of Texas at Arlington

AUGUST, 2018

Arlington,Texas

Supervising Committee:

Dr.Jiang Ming, Supervising Professor
Dr. David Kung
Dr. Jeff Y. Lei

ACKNOWLEDGEMENTS

I would like to thank Dr. Ming for his timely guidance and motivation. His insights for this research were valuable. I would also like to thank Dr. David Kung and Dr. Jeff Y. Lei for taking time out from their schedule and attending my dissertation and providing valuable feedback. Thank you Lakshay for helping me with my research. I would like to thank my colleagues for supporting me directly or indirectly. Last but not the least, I would like to thank my parents, my family and my friends for encouraging me and supporting me throughout my research.

July 23, 2018

ABSTRACT

MALWARE EARLY-STAGE DETECTION USING
MACHINE LEARNING ON HARDWARE
PERFORMANCE COUNTERS

Anchal Raheja, MS

The University of Texas at Arlington, 2018

Supervising Professor: Dr. Jiang Ming

Systems affected by Malware in the past 10 years has risen from 29 million to 780 million, which tells us it's a rapidly growing threat. Viruses, ransomware, worms, backdoors, botnets etc. all come under malware. Ransomware alone is predicted to cost $11.5 billion in 2019. As the downtime and financial damages are rising the researchers are finding new ways to tackle this threat. However, the usual approach is prone to high false positive rate or delayed detection rate.

This research explores a dynamic approach for early-stage malware detection by modeling it's behavior using hardware performance counters with low overhead. The analysis begins on a bare-metal machine running malware which is profiled for hardware calls using Intel VTune before it infects the system. By using this system design, I am able to generate models from data extracted using hardware performance counters and use it to train the system using machine learning techniques from known malware samples collected from VirusTotal and Hybrid Analysis.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

ABSTRACT iv

LIST OF ILLUSTRATIONS vi

LIST OF TABLES vii

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 BACKGROUND 4

CHAPTER 3 RELATED WORKS 8

CHAPTER 4 SYSTEM DESIGN 12

CHAPTER 5 EXPERIMENTAL SETUP AND EVALUATION 19

CHAPTER 6 IMPLEMENTATION 25

CHAPTER 7 PERFORMANCE AND LIMITATIONS 28

CHAPTER 8 CONCLUSION 31

REFERENCES 33

BIOGRAPHICAL INFORMATION 35

LIST OF ILLUSTRATIONS

Figure 4-1 Intel VTune Sample Run 14

Figure 4-2 Multi Stage Malware 18

Figure 5-1 VirusTotal Result Page 22

Figure 5-2 Hybrid Analysis Result Page 23

LIST OF TABLES

Table 4-1 Hardware Events 15

Table 5-1 Family Data 23

Table 7-1 System Performance 28

CHAPTER 1

INTRODUCTION

Viruses, ransomware, worms, Trojans, and bots are all part of types of malicious software's known as malware. Malware is created for the purpose of generating money illegally by compromising sensitive information, stealing credentials stored in the system and destroying files in victim's computer. They equipped to attack various platforms such as computers, mobiles, routers, smart televisions etc. Variety of malware is used in cyber espionage, hacktivism, and cyber warfare. Attacks using malware can be done in several ways such as via file downloads from malicious websites, through email containing benign looking file or embedding malware in Microsoft word, pdf documents. Once a system gets compromised, it is connected with other infected systems. The infected network of systems is known as the botnet. Common types of malware consist of viruses which are programmed to travel autonomously making its way to victim's computer being attached to files downloaded from websites or peer-to-peer file sharing. It may exist on a system but will not trigger its attack process until an infected file is clicked by the user. Another category or malware spreading automatically is worms. Its behavior differs from the virus in a way that instead of attaching itself to multiple objects on the system, after execution it looks for another system on the network to infect. Most worms don't even require human interaction for spreading themselves.

Trojans being part of malware family differ from viruses and worms. The major distinction that separates Trojans from viruses and worms is that they are not self-replicating and rely on the connectivity of the internet to the device. Backdoor Trojans provide remote administration to victim system over the internet to the attacker without the user knowing about it. It often includes keyloggers, recording every key press from the victim in pursuit of finding passwords and other confidential data reported to the

attacker. In spring 2017 Mac users were affected by Proton Remote Access Trojan(RAT) having root capabilities to execute bash command, getting passwords, taking screenshots/webcam shots etc[1].

In recent times, we have seen hybrid threats that combine the functionality of virus, worms, and Trojans providing criminals with ease of use in carrying out various attacks. Once the system is connected to botnet it can be used criminals to use the whole network of infected systems to perform target attacks on specific organizations, sending spam messages. Mirai botnet made Global headlines by taking down much of the internet for a day by attacking single organization with 100,000 malicious endpoints and peaked attack bandwidth of 1. 2Tbps[2]. Another major player in the malware family is ransomware. It infects the system and limits access to files of the user on the system until the ransom is paid. It is done by encrypting the victim's file and locking the victim's desktop. It is considered as one of the key threats to a system security. Recently WannaCry, a type of ransomware, infected over 200,000 systems in 150 countries, causing billions in damages[3].

Considering the recent attacks carried out, coming up with solutions to protect users from such attacks have become very important. To devise a solution and build protection, we need the fundamental understanding of how malware functions and infects the systems. Most techniques for malware defense include static and ineffective ways to tackle the threat. That is done by performing basic analysis and updating the malware definitions as they encounter malware. The purpose of this paper is to use a dynamic approach in exploring the working of malware and its effects on infected systems.

Now, detection of malware exploitation can be achieved using anomaly-based detection techniques due to developments in hardware performance counters. Analyzing malware on basis of code can reveal static properties such as hashes, packer signature,

functions etc. This approach is done on malware without execution on binary files and by disintegrating the malware using a disassembler. On the other hand, the anomaly-based analysis is done by monitoring system activity and classifying it as either normal or anomalous. The classification is done on the basis of recorded micro-architectural events such as cache hit, miss etc. during the execution of the malicious file. The execution is done in a sandbox environment, that helps to get the system back in the original clean state while analyzing the execution and recording of the events. Anomaly-based detection has the upper hand to signature-based detection as it is capable of detecting zero-day malware. Many behavior-based and signature-based detection techniques fail to detect certain advanced malware due to looking for specific behavior such as modifying file system or precise changes in the file signature.

In this work, my analysis is based on anomaly-based detection using micro-architectural events to model the behavior of malicious files as compared to benign files. In this procedure, a realistic sandbox environment is generated on which the malware is executed and its events are recorded using the hardware performance counter. The malware is executed with admin privileges giving it access to file system, network activity etc. System state is preserved using Deep Freeze [4] which saves the clean state and allows the execution of malware and its interaction with the system. Data is from hardware performance counters is collected using Intel Vtune Amplifier [5] during the execution. Vtune Amplifier is configured to measure specific events due to their high discriminative power.

In this research, I have analyzed clean 7024 and 698 malware samples consisting of viruses, ransomware, Trojans, rootkits, and worms. Proposed detection technique was able to achieve the rate of 99.13%. The suggested system should be used with existing signature-based detection systems to greatly improve system security.

CHAPTER 2

BACKGROUND

Like Malware in recent years has evolved drastically with the development of new legitimate tools giving attackers newfound capabilities. Starting from viruses to ransomware, most of the attacks have gone “clickless” requiring no or little human interaction with the malicious files.

Exploits are bypassing user interaction by execution of remote exploits or exploiting Remote Desktop Protocol. Benign tools such as PsExec have given attackers great capabilities allowing them to execute attacks remotely, without even dropping files on disk.

Viruses use a vast number of techniques to infect the system and further spread the infection on different systems. For example, it attaches itself to all the existing files which are hidden to the user, by creating hidden shortcuts, and as soon as user attaches removable drive it copies itself on the drive too, which leads to further infection when connected to another clean system. After the virus infection, usually, the user is unable to access command prompt to further avoid removal of virus. Malware establishes persistence in systems usually through registry keys. Malware will modify registry keys to ensure they can launch itself even after system reboots, hide themselves with existing legitimate user processes. Detection of such attacks can be audited using Autoruns [6] tool from Microsoft. Worms, on the other hand, leave very little or no trace on the system disk but usually, the infection can be detected by an increase in network activity even during system idle stage. Ransomware after communicating with their command and control servers, to establish the connection with the attackers, commonly encrypt the user data and change the desktop wallpaper. The wallpaper lists out the user files have been

encrypted and guide the user through the ransom process, to pay and gain access to own files.

System Persistence: After malware execution through Intel Vtune, persistence in the system is established through changes in registry hive. Malware usually makes API calls through regedit.exe using the command line to modify registry values and adding registry keys. Top categories targeted by attackers are HKCR and HKLM. HKCR short for **HKEY_CLASSES_ROOT** Is a registry hive which contains data related to installed programs such as video players etc. Registry values and subkeys are attacked in this to execute malicious calls hiding under benign processes. The other target hive is HKLM or HKEY_LOCAL_MACHINE. This hive includes information related to the system such as sensitive security settings. By modifying this hive malware establish persistence even after system reboots.

Variation in User Files: Viruses which come under file infector viruses as the name suggests infects executable files with intent to cause permanent damage and make them usable. It injects itself into the code or worse overwrites it. Typically, it infects files with .exe extensions so as when the file is executed or accessed the virus further spreads. Multiple file shortcuts of the same file are usual observations after the attack. Ransomware families such as WannaCry, CryptoLocker, Crysis attack the user files such as documents, pictures etc. It starts encrypting the user files after calculating the encryption key either at the system or getting it from command and control servers. Generally, ransomware use Microsoft cryptographic API for encrypting victim's files. After the process is complete files can be seen with. wnry, .wncry, .wncrypt extension after the attack is complete.

Detection: For detecting the malicious activities after the execution there are several key areas to observe and tools help in further detecting the region and flow of attacks. To

audit startup processes, registry hives Autoruns for windows comes into play. It reports on registry hive, installed drivers, startup folder, auto start utilities etc. VirusTotal feedback on those reports help in deciding the file nature. The machines were also equipped with Crowdstrike Falcon which has endpoint detection and response (EDR) capabilities which helps in understanding the flow of malware executed down to API calls to observe changes made on the system. Ransomware makes createDesktop() API calls to establish the system has been compromised by listing out the process to pay ransom as wallpaper.

Hardware Features: Ongoing studies have shown promising detections of malware programs using hardware execution patterns. These features are easier to capture and analyze as compared to high-level features such as OS observables. Other than showing promising results events have low overheads to capture as they are captured straight from hardware performance counters situated in modern processors freeing us from the need of additional hardware. As observed in some studies [7] micro-architectural features are very noisy of benign programs as compared to malware. Using hardware features are highly desirable as they are harder to evade and modify for the attacker. Considering an example if the malware uses a combination of wireless adapter and GPS unit to transmit the location, no matter how much the attacker changes the execution order of instruction wireless adapter and GPS unit will be interrupted to fetch metrics for malware. Software-based detection techniques might miss these calls as a slight change in code will lead to change in signature or hash of the malware, but the hardware-based calls will remain identical. To evade hardware-based detection the attacker has to induce for example cache behaviors, branch misprediction etc. which is arguably tough to achieve and raises the bar for attackers [8].

Metrics: Intel Vtune [5] measures 100+ hardware events per 1ms for each file executed. For analysis, feature selection is done on basis of Fisher Score to use features with high discriminative power such as cache hit, a cache miss and branch misprediction etc., which enables larger separation between benign and malicious processes.

Hardware Calls with VTune: VTune is a powerful software performance analysis profiler available for 32 and 64 bit x86 systems. It is also capable of hardware sampling on Intel-manufactured chips. For hardware sampling, it uses the on-chip performance monitoring unit on an Intel processor. It has both GUI and command line interface. Custom analysis type can be built in an analysis type tab. For this paper, I am recording 19 different parameters.1ms is used as a sampling interval for hardware calls.

Using this paper, I describe various procedures under which the malware is executed to compromise the system and ability to detect on basis of hardware performance counter.

CHAPTER 3

RELATED WORKS

Malware attacks have troubled users and many other organizations for many years and are increasing day by day. In response to this behavior, defenders have created and maintained various techniques such as developing antivirus with signature-based technique to detect malware and researching better ways for automated malware analysis systems. Static malware analysis involves notable challenges due to commonly used polymorphism, packing, and other unclear techniques. However, to eliminate the ill results of malware scanning using static analysis, malware analysts focus on dynamic analysis approach to scan and detect malicious executables. In this process, a heavily instrumented and well-maintained environment called as 'sandbox' is loaded with a malicious code and executables to monitors its behavior at varying levels such as I/O activities, System calls, and registry values. Mobile application marketplace, search engines and many antivirus companies along with the security research communities rely on dynamic code analysis. Evasive malware often uses various artifacts of the target systems to alter its behavior give the fact that system emulators and virtual machines are commonly used platforms for developing sandboxes. But, the Slightly misconfigured sandbox can produce undesired results and increase the number of false positives. In such cases, lower-level properties of the execution environment can be used to identify the presence of a virtual environment. These include emulator intricacies that can be noticed at runtime using small code snippets, timing properties of certain virtualized instructions and cache side effects [9].

An unfolding approach to detect malware is to examine and build indicators in hardware. This technique uses information available in the hardware to detect malware. It has been observed that hardware malware scanning approach is promising for two major

benefits. First and most important, hardware systems protect vulnerable software using powerful and simple hardware implementations that reduce the bug defect density in contrast to software malware solutions that aims to protect vulnerable software with equally vulnerable software like any software antivirus. Second, it is difficult to achieve evasion in a system that employs hardware features considering the fact that an intruder may evade any of the defensive techniques. An important instinct behind this technique is that the degree of control over the lower-level hardware features varies from the software features. As a result, it is difficult to persuade branch misprediction or cache misses than to alter file names or system calls over the period while exploiting the target system.

Existing malware analysis and detection techniques can be categorized as follows: malware detection approach and the malware features that are targeted. Detection approaches are further classified as anomaly-based detection and misuse-based detection. Unlike misuse-based detection, anomaly-based detection identifies known as well as novel attacks. However, there is a wide range of features that can be used for detection. Earlier, till 2013, OS and application level features such as system calls and network traffic were considered to be noticeable. Since then, lower-level features which are closer to hardware such as microarchitectural events have been used for malware detection. The feasibility and limitations of anomaly-based malware detection using both architectural and low-level microarchitectural features available from HPCs can be observed and examined. During exploitation in the anomaly-based detection of malware, the malware alters the original program flow to carry out an abnormal code in the context of the victim program. Execution of such unexpected code causes disturbance to the dynamic execution attribute of the program. These interruptions can form the basis of detecting malware exploits if they are noticeable.

In [7] this research, baseline characteristics of common vulnerable programs such as the Internet Explorer 8 and Adobe PDF Reader 9 are modeled considering these two as most attacked programs and examined if such interruptions do exist.

A multi-stage malware infection/exploit process, as the name suggests, involves various stages. Typically, the multi-stage attack occurs when your browser visits the malicious site and executes a malicious program code that launches another stage in which Java code runs which downloads and executes the final payload which is nothing but the malware. These stages are further explained in detail.

Eliciting the vulnerabilities: Step 1 includes identification of the exploit. The adversary identifies the exploit and passes it to the victim to target a particular vulnerability that is known to the adversary. The vulnerability is commonly a memory corruption bug. The exploit is generally sent to the victim from a browser window or as an attachment in an email. When the victim downloads this payload and accesses the exploit, two malicious subprograms load into the memory of the vulnerable program. These subprograms can be identified as the code reuse ROP and shellcodes.

Code Reuse Shellcode (ROP): Data Execution Prevention is commonly used in the modern processor to limit the execution of the code from the web pages. However, this Data Execution Prevention can be manipulated by the program itself. Thus, this ROP shellcode reuses instructions in the original program and evades Data Execution Prevention. It calls a function which disables Data Execution Prevention utility for the web page that contains the code. The shellcode is generally small code stub that is used for downloading the payload in memory to perpetuate the secrecy. Stage 2 revolves around the payload. The payload is generally a keylogger, backdoor or any system exploration program. Once it is downloaded, stage1 shellcode runs the payload using strategic DLL

injection. Such injection is known as a secret library injection techniques that do not require any physical files to compromise the system.

However, The Stage1 shellcode and Stage2 payload are dissimilar in design, its operations and functioning and size, majorly due to the operational restrictions on the Stage1 shellcode. Exploit authors try to use as less memory as possible to guarantee that the program does not accidentally overwrite their malicious code in the memory. This code needs to be small, platform independent and faster in execution in order to achieve a good probability of success. Therefore, it is written in assembly language and uses very limited memory addressing style. These restrictions cut downs adversary's ability to write large shellcodes. On the other hand, the payload can be of any size and does not have all these restrictions. There are a large number malware exploits that are executed from memory. They may maintain disk and process acquiring the information by making sure that no new processes are being initiated as well as no files are being written to the disk. This can lead to an ease of evasion of many of the file-based malware detection techniques.

CHAPTER 4

SYSTEM DESIGN

In this chapter, I outline the approach on detecting malware attacks at an early stage. The system proposed in this research aims to enhance the detection rate during a malicious activity. Viruses, Trojans, ransomware each belong to separate malware families, having specialized capabilities and attack vectors. A major concern during malware execution was to maintain a clean environment for each run while gathering metrics. To build the testing environment I have used DeepFreeze [4] which freezes the environment state and still renders the capability to read/write files on disk during the session. Using that I could go back to original testing state after reboot even after successful malware execution.

Due to the rise in highly sophisticated malware the need for creating a realistic testing environment arises, which has been explained further in this paper. After describing the testing environment design, I lay out the process used for capturing metrics from hardware performance counters and how metrics help in analysis and detection of malware using HPC events.

Testing Environment: Malware has evolved drastically in recent years and is using advanced detection techniques even before infecting the system. environment aware malware is able to distinguish between an artificial environment and a "real-user" environment with an accuracy of 92.86% [9]. Various system artifacts such as the number of processes running, web history, number of files existing on desktop help malware to gauge whether the system is a testing sandbox or a real user environment. Malware alters its behavior as soon as they establish that the execution is being done on an analysis sandbox environment. While creating the realistic environment for testing and

avoiding evasion strategies, emphasis should also be on the creation of naturalistic usage history.

To boost the prospects of avoiding fingerprinting techniques, best would be to address this issue by creating real user data with real usage habits over a period. This helps in building usage history, patterns of file location and naming conventions which represents itself as a real user. In order to tackle this environment was created by using the system as a regular system for daily work. This helped in creating a trail of usage history with real processes and files in the environment.

HPC Metrics: For gathering metrics from hardware performance counter Intel Vtune is being used. It is capable of code profiling including stack sampling and hardware event sampling. Tune uses chip performance monitoring using which requires Intel processors. Under measuring hardware performance, it generates three critical metrics [5]:

- CPU Utilization
- Memory Access
- FPU Utilization

For collecting samples from on-chip monitoring hardware statistical sampling is done. Overhead is very low for sampling due to the on-chip collector.

Vtune has various analysis types focusing on the algorithm, compute-intensive application, microarchitecture and platform analysis. Hardware performance counter is able to gather a huge variety of metrics per sampling interval. Instead of using pre-built analysis types, I am using custom analysis in which features with high discriminative power and having sizable separation are collected. Figure 1 shows the collection results from custom analysis by executing a binary file.

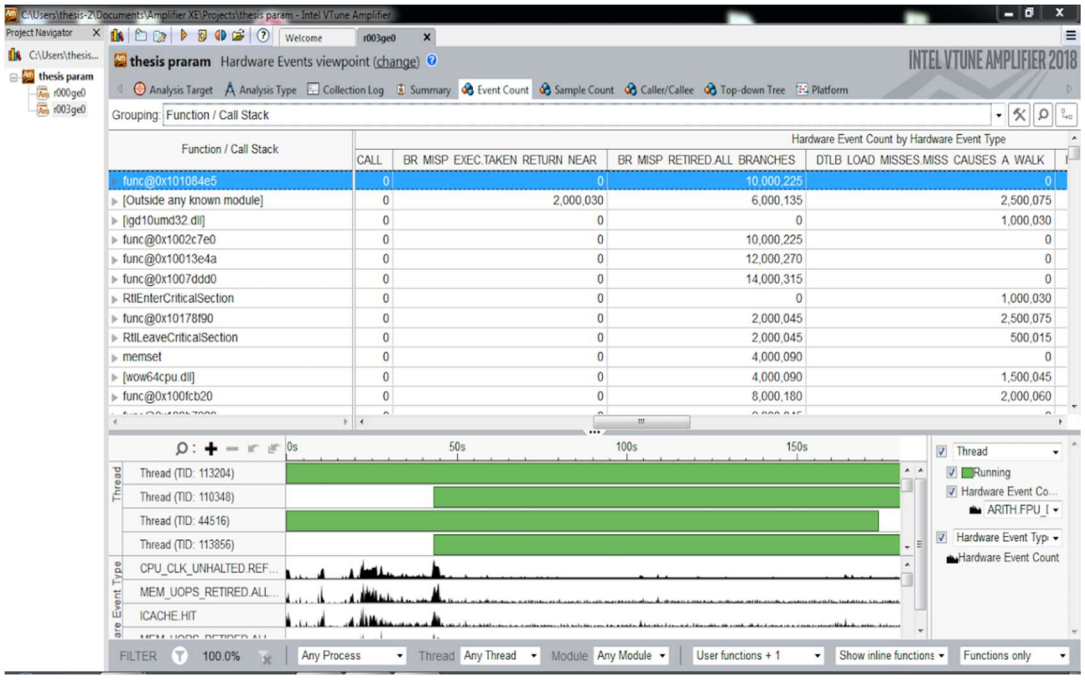

Figure 4-1 Intel VTune Sample Run

The result shows multiple columns from the collection of results based on selected high discriminative hardware events out of thousands of events. Here we can observe Hardware event count by hardware event type which helps in separating malicious files from benign ones.

Sampling: For detecting malicious behaviors I am measuring 20 events captured from Hardware Performance Counters. These events consist of both architectural and microarchitectural events. The events listed below with description are key events with the highest discriminative power which helps in the detection of malicious behavior stage which separates it from benign behavior.

Table 4-1 Hardware Events

SNo.	Hardware Event Type	Description
1.	ARITH.FPU_DIV	This event counts the number of the divide operations executed.
2.	BR_INST_EXEC.ALL_DIRECT_NEAR_CALL	Speculative and retired direct near calls.
3.	BR_INST_RETIRED.ALL_BRANCHES_PS	All (macro) branch instructions retired. (Precise Event - PEBS).
4.	BR_INST_RETIRED.NEAR_CALL_PS	Direct and indirect near call instructions retired. (Precise Event - PEBS).
5.	BR_INST_RETIRED.NEAR_RETURN_PS	Return instructions retired. (Precise Event - PEBS).
6.	BR_MISP_EXEC.ALL_CONDITIONAL	Speculative and retired mispredicted macro conditional branches.
7.	BR_MISP_EXEC.ALL_DIRECT_NEAR_CALL	Speculative and retired mispredicted direct near calls.
8.	BR_MISP_EXEC.TAKEN_RETURN_NEAR	Taken speculative and retired mispredicted indirect branches with return mnemonic.
9.	BR_MISP_RETIRED.ALL_BRANCHES	All mispredicted macro branch instructions retired.
10	DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	Load misses in all DTLB levels that cause page walks.

11	DTLB_STORE_MISSES.MISS_CAUSES_A_WALK	Store misses in all DTLB levels that cause page walks.
12	ICACHE.HIT	Number of Instruction Cache, Streaming Buffer and Victim Cache Reads. both cacheable and noncacheable, including UC fetches.
13	INST_RETIRED.PREC_DIST	Instructions retired. (Precise Event - PEBS).
14	ITLB_MISSES.MISS_CAUSES_A_WALK	Misses at all ITLB levels that cause page walks.
15	ITLB_MISSES.STLB_HIT	Operations that miss the first ITLB level but hit the second and do not cause any page walks.
16	L1D.REPLACEMENT	This event counts L1D data line replacements. Replacements occur when a new line is brought into the cache, causing eviction of a line loaded earlier.
17	LONGEST_LAT_CACHE.MISS	Core-originated cacheable demand requests missed LLC.
18	LONGEST_LAT_CACHE.REFERENCE	Core-originated cacheable demand requests that refer to LLC.
19	MEM_UOPS_RETIRED.ALL_LOADS_PS	This event counts the number of load uops retired (Precise Event)
20	MEM_UOPS_RETIRED.ALL_STORES_PS	This event counts the number of store uops retired. (Precise Event - PEBS)

Event Analysis: Data provided from Intel VTune represents architectural and microarchitectural events. For analysis, supervised machine learning models like linear

regression and support vector machine SVM were applied to analyze features with the highest degree of separation of the given 18 features/events that can help further to detect deviations that can occur because of malware exploitation.

Collected data were randomly divided into two parts, 70 is to 30, where 70 % of the data was selected for training purpose and the remaining 30% data was selected for prediction analysis. Out of linear kernel and non-linear kernels, linear kernel of support vector machines was used as the dimensionality of data is two. Every event was individually predicted using the trained model and the test dataset. Accuracies were checked for the individual predicted values to determine the best events that can be used for early malware detection.

Later Logistic Regression was implemented where the dataset was again randomly divided into two parts, 70 is to 30, where 70 % of the data was selected for training and the remaining 30% for testing. The overall accurately predicted percentage of logistic regression was compared to the overall accurately predicted percentage of SVM to decide which model can be best used for this purpose.

Malware Behavior: In this section, I am going to go through the process of how multi-stage malware works. Understanding different stages of malware help in better detection of malware in the earliest stage possible. The infection process is as follows [7]-

1.Entry: The first part of the attack consists of entering the system, either in form of an email attachment or drive-by download. Usually, memory corruption bug is the vulnerability, which is being exploited through the means of execution of this file. After the exploitation, ROP and Stage 1 Shellcodes are loaded into the memory of the vulnerable program and in turn transfers the control to ROP shellcode.

2.**ROP:** Return-oriented programming is an advanced version of smashing the stack attack, which facilitates the writing of attack code on the stack. Because of Data

Execution prevention, the attacker cannot execute malicious instructions. To circumvent this the return address is manipulated. The redirection facilitates the execution of shellcode in the next stage.

3. **Shellcode**: It is a small piece of code which is used in exploitation. The size of shellcodes is typically less than 400 bytes. These are carefully crafted instructions whose sole purpose is to download the payload in memory secretly.

4. **Payload:** This is the main part of malware which the attacker wants to execute on the system to perform malicious actions. It consists of backdoors, keyloggers, rootkits, screen scrapers, logic bombs etc. Since the payload in the address space of another process, it is executed through the process of DLL injection by forcing it to load dynamic-link library.

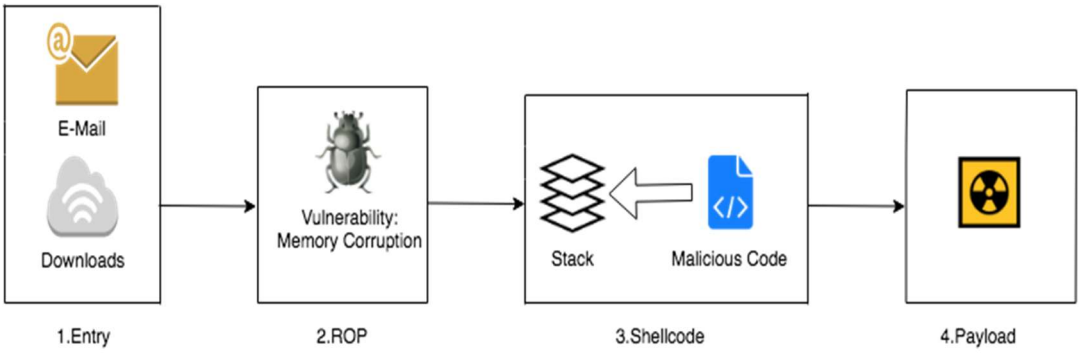

Figure 4-2 Multi Stage Malware

After these stages, the systems if fully infected. Shellcode is supposed to be small and portable in size so as to not to overwrite the exploit code in memory. Whereas the payload is free from these constraints hence can be of greater size. These exploits completely reside in memory and are executed from there. File-based malware detection techniques rely on new files written on disks or new processes created, hence this malware are able to evade those using these strategies.

CHAPTER 5

EXPERIMENTAL SETUP AND EVALUATION

Due to the execution of malware during fetching metrics it is important to be able to protect user files and be able to rollback user file system to original state without the possibility of data loss, file encryption and residue malware traces, therefore the baseline clean systems are protected by DeepFreeze [4]. DeepFreeze is a kernel level tool used to redirect information being written on the hard disk or the partition to protect the hard drive integrity. Since it is installed on the real user system this prevents in avoiding malware evading execution due to sandboxing. After the system restart redirected information is no longer available, that helps in restoring the system back to its clean state at the disk sector level. Since all the information is being redirected temporarily the malware is given full virtual control of the machine, which enables them to perform delete, modify, add, encrypt etc., to the file system and even making the system unusable in case of ransomware temporarily. The system is restored to original state once it is rebooted removing all traces of malware execution which happened before the reboot.

For my analysis, I am using four machines running Windows 7 (64 bit). Each of the machines has DeepFreeze installed on them over the unique user file system created to prevent malware from identifying them as a sandbox for testing to prevent anti-evasion features built-in malware. Every individual machine is configured to generate new fake MAC address after reboot to help in avoidance of being identified as a testing machine. All the network traffic routed was inspected with Snort [10]. Snort is a free and open source intrusion prevention system and network intrusion detection system. It has real-time traffic analysis and packet logging feature with protocol analysis. During testing, it was used in sniffer mode for just monitoring purpose. It allowed malware to talk to internet and their command and control servers to launch the attack but it also helped to

cut off the internet connections as soon as the spike was observed on the network for further spread of malware outside the system.

Each of system has setup up with real user file system consisting of valid files in My Documents folder, images and video files at desktop and downloads folder. The browser has valid user browsing history with saved username and passwords, cookies and bookmarks of top 10 websites curated from Alexa. The system also has a set of basic software installed for media files, music playback, and document related works. Each of the samples was executed for a specific amount of time, which was enough to model malicious behavior using the hardware performance counter of each sample. All the metrics fetched during the execution were sent to a remote system used to maintain metrics for processing. After this cycle is done the system is rebooted, hence restored back to its original clean state with new MAC address to avoid any impediment in any sampling thereafter. Each execution and analysis of malware as well as the benign file was done according to established practices for malware experiments to achieve transparency, correctness, and safety for collecting and using malware datasets [11]. Four guidelines mentioned for designing prudent experiments are as follows.

1. **Correct Datasets**: This involves checking whether the samples have legitimate software also known as good ware in the dataset and removing it from the dataset. This was done by checking each sample with two different public sample submission websites – VirusTotal and Hybrid Analysis. Datasets were also balanced to best of effort to avoid domination of single malware family.
2. **Transparency**: Each of the malware executed and analyzed was categorized into the family relates to. This was done using info available on public sample submission domains. System specification and network activity used in the analysis are also mentioned in this study to induce transparency.

3. **Realism**: Emphasis in this section is on behavioral characteristics of malware and its relevance to the real world. To accommodate this only samples discovered in the last two years were used. Considerate internet connectivity was provided for completion of malicious execution such as communication with command and control servers.
4. **Safety**: This includes deploying and describing containment policies. Certain malware ten to generate high traffic from infected devices. Strict containment rules were followed to avoid harm to other systems during the execution and analysis of samples.

After establishing the system to be used for analysis and rules for experiments, next critical phase was to create a dataset consisting of malware such as viruses, Trojans, worms, and ransomware belonging to different families so as to incorporate variety in the analysis. The dataset also needs to have a set of benign files which in no way should have any malicious traces so baseline characteristics are well defined. For this, I have used two web applications VirusTotal [12] and HybridAnalysis [13]. VirusTotal is an online application which aggregates many antivirus products and acts as an online scan engine. For dynamic analysis of malware, VirusTotal uses Cuckoo sandbox [14] for execution of the file. After analysis, it displays hash and scores from various antivirus tools. Hybrid Analysis is also a web application like VirusTotal which uses VxStream Sandbox. It saves fine-grained memory dump snapshots of the process during runtime. Building the dataset involved following steps

- **File Collection**: Files were collected by browsing the internet and downloading random files and software from it. Files with extensions having a doc, pdf, exe, xls, zip were downloaded. Also, specially formed queries on Google were done

using Google to display results containing file downloads. Additionally, samples were also downloaded from VirusTotal and Hybrid Analysis.

- **Classification**: Files collected were classified into malicious and non-malicious categories.SHA-256 hash was calculated for each file and using respective API's the hash was checked on VirusTotal and Hybrid Analysis. Besides hash, each file was uploaded to both the websites to get the detailed analysis. The file was categorized as malicious if five or more engines on each website detected the file as malicious. The file was considered benign if none of the engines on either website returned any negative results.

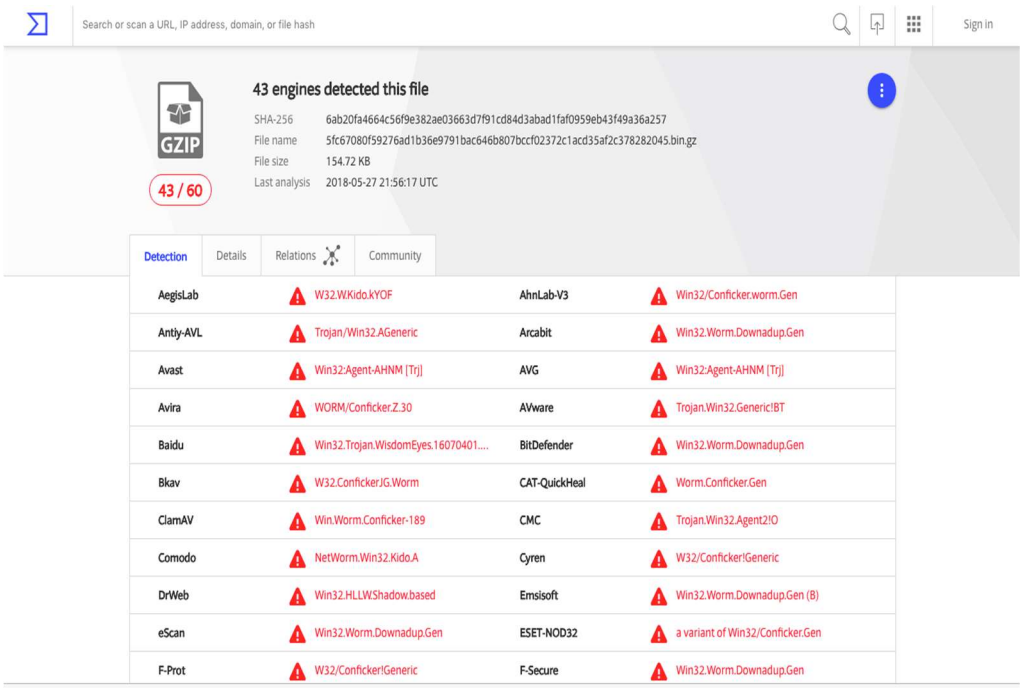

Figure 5-1 VirusTotal Result Page

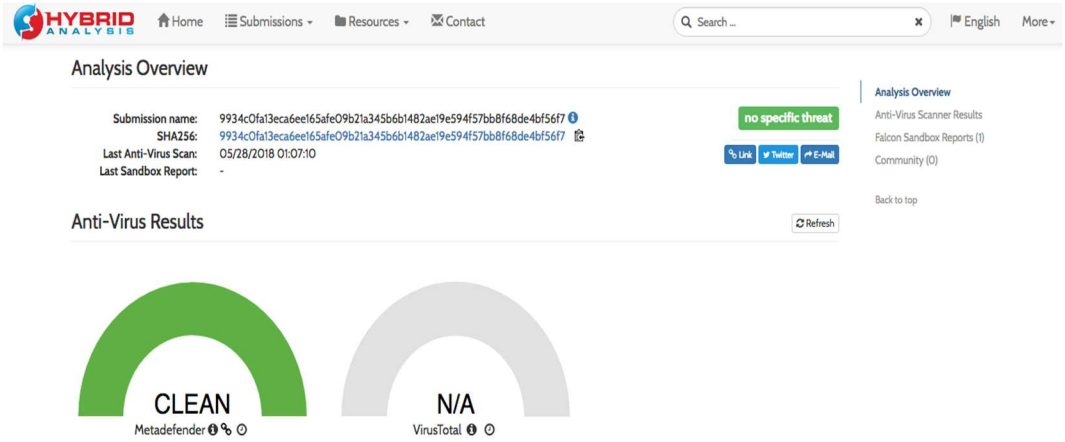

Figure 5-2 Hybrid Analysis Result Page

- **Family**: If the file is found to be malicious on VirusTotal and Hybrid Analysis, the family name is displayed for the malware. It analyses the code and behavior of the file to establish signatures related to a specific family.
- **Type**: Out of all samples categorized as malicious further analysis was done using both the websites. Malicious files were categorized as virus, Trojans, and worms.
- **Timestamp**: Files and hashes submitted on VirusTotal also show the first submission date. If the malicious file was first submitted more than two years ago, it was removed from the dataset.

Table 5-1 Family Data

Type	Count
Worm	78
Ransomware	308
Rootkit	13
Trojan	240
Browser Hijacker	31
Botnet	28
Total	**698**

The plan of this approach is to model the malware exploit process using calls recorded from hardware performance counters. Hardware calls recorded during the malicious process execution is be used in modeling the malware behavior and is used to separate benign behavior. For profiling, I have built a custom filter to record specific events Using hardware performance counters. In the custom filter, I am recording 20 different characteristics such as cache hit, mispredicted branch instructions retired, speculative and retired direct near calls etc. which are queried every 1ms. After the result is collected in Intel Vtune the metrics are then extracted to CSV file's file is named as the same name of the file being analyzed, this helps in better tracking of data. Later all CSVs are used in support vector machine as well as logistic regression for analysis. The challenge encountered here was the values fetched directly from hardware performance counter made detecting anomalies undetectable. The values recorded in different file runs had a high variance which impeded the performance of support vector machines and logistic regression. To tackle this, issue the values of each character is normalized with both malicious and benign file. This helps in data integrity which means data is consistent and confined in particular range of values and also the better building of support vector machine model for classification as well as detection of early stage of malware.

CHAPTER 6

IMPLEMENTATION

In My aim in this chapter is to describe the execution of the proposed system developed for malware analysis and detection on windows operating the system using hardware performance counters. The principal factor in choosing Windows operating system for this research is because windows remains a top target for attacks accounting for 77.22% [15] of the recorded malware attacks. Executable extension .exe accounts for 26.27% [15] of the file malware extensions which contributes to another major reason for selecting windows as a platform for building the proposed system. The malware analysis and detection system focus on detecting malware in early stages using various architectural and microarchitectural features captured from built-in detectors in hardware. Further, I explain how the systems were created for malware analysis and how the calls are captured from hardware performance counters. In the last section, I explain how the data is pre-processed and analysis system was built after the metrics are gathered.

Creating User Environment: Setting up the user environment was the first crucial stage of this study. Since malware has evolved enough to a stage where detecting a sandbox vs real user environment is a trivial task for them [16]. Attackers have upgraded the attack methods to look for legitimate file paths, file names, registry key entries, types of file and organization of file before even launching the attack. To evade these detection techniques this real user environment was created for 30 days on each system. The environment created is further explained in following sections emphasizing characteristics used to evade sandbox detection by malware.

File extensions: Studying various attacks done by viruses, Trojans, ransomware, it is usual that attacked files are those which have either been created by the user or imported into the system by the user. Malware frequently attacks image files having

extension such as .png , .jpg, .jpeg etc. In documents the target for files having extensions .txt, .docs, .doc, .pdf , .csv , .xlsx are compromised. For media files there are various extensions ranging from .mp3, .dct , .wav , .flac for audio files to extensions such as .mp4, .mov, .wmv , .avi , .flv for videos. Malware also targets keys and licenses having an extension such as. pem, .crt , .csr, .cer. These files are records for software licenses. In daily usage of the system, various documents were created and downloaded from legitimate sources. Assorted music files were downloaded from legitimate music stores and stored on the disk with all metadata intact. For creating videos and images files were captured from webcam and screenshots were taken using the system. Additionally, files were copied from mobile phone and placed on the disk.

File Paths: Malware before launching the attack and compromising the systems also looks for valid user paths and folder names. Locations are not only limited to user documents in My Documents folder but also registry paths are checked. If all the files are stored in one location, it has been observed that attacks are not launched due to the system being identified as a sandbox. Files saved by the system usually go into default directory My Documents and categorized into subfolder depending upon the file type. Files downloaded from the internet land in the Downloads folder. Other than that, files also reside on the desktop. Keeping all these factors in minds files were organized in all three locations My Documents, Downloads, and Desktop. Subfolders were also created in those directories to make it more real user like environment. Apart from file structure malware also look for registry key entries. Registry keys were added in HKEY_CURRENT_USER\Software and HKEY_LOCAL_MACHINE\Software to simulate real user environment.

Measuring Hardware Calls: Hardware calls are made during any execution on the system. I am using Intel Vtune to query these calls that are generated during the

execution of malicious as well as benign executions. Intel Vtune uses hardware performance counters on-chip to query these metrics and fetch it every 1ms. These hardware metrics consist of both architectural and microarchitectural events generated during running the files. These data are collected at very low overheads using hardware performance counters situated on the chip. As compared to high level features these features are easy to capture and harder for attackers to evade. It is effortless for the attacker to modify the system calls and file names to evade but to induce cache hit or a cache miss in a detailed way in accordance with time is the strenuous procedure. Intel Vtune has various built-in analysis types. In this research, I have built a custom filter which measures 20 architectural and microarchitectural calls to hardware while execution. For granularity of data, CPU sampling is done every 1ms and Uncore sampling is set at 10ms. Once the sample is analyzed using VTune, the results are shown in Vtune result format. The metrics are then exported to remote machine swiftly so as to avoid results being destroyed by the malware especially in case of ransomware. After the successful transfer of data, the system is rebooted. Since the systems are in “frozen” state by DeepFreeze, meaning all the system calls, file activities etc. are virtual changes and system gets restored to its original state at the disk sector level. After reboot system gets back in its original state and new MAC address is assigned to avoid being detected as a sandbox for testing malware.

Data Analysis: After the metrics are sent over to the remote system, the metrics are then added to the central database with correct labeling for processing. Since the variance was high in values, the data required pre-processing before feeding it to Support Vector Machine for supervised training and analysis. The values are normalized and then fed into the model for analysis. The same database is used for analysis using logistic regression.

CHAPTER 7

PERFORMANCE AND LIMITATIONS

This study proposes a malware analysis and detection system using hardware performance counters, an approach which is different from the usual and arduous for attackers to evade. Actual and latest malware were used in real life environment for testing and quality results were obtained. But due to evolving times and attackers constantly upgrading their attack vectors, I further explain a few limitations and strategies which can possibly be used to evade the malware analysis and detection system. Nonetheless, these are harder to implement and are based on a few assumptions. These evasion strategies [17] require redesigning of shellcode which is hard and raises the bar for the attackers. Table 1 shows the overall performance of the proposed system.

Table 7-1 System Performance

Type	**Count**
Worm	78
Ransomware	308
Rootkit	13
Trojan	240
Browser Hijacker	31
Botnet	28
Benign Executions	7024
Total Accuracy	**99.13%**

Stalling Code: This technique involves the malware to perform useless CPU cycles in order to disguise itself to look like a benign program. This exploits the fact that the analysis system spends a certain amount of time to look for metrics from hardware performance counters and then building models for analysis. The malware can stay dormant at that time and not create and malicious activities. During the stalling time malware, can also look for the variables to identify the system as being an analysis environment and not even execute at all. But this approach will have a low success rate as the proposed system is not using sandbox rather using real user-like environment for analysis. Another drawback of this approach can be detected by software based antivirus software's as the malware will reside in the system for quite some time.

Instruction Addition: This approach is complicated and requires full knowledge of the system used for analysis to evade detection. Full knowledge of system comprises of knowing all the features of user environment such as which operating system on which processor it is running. Also, since hardware performance counters are able to measure thousands of event the attacker should also know which all architectural and micro-architectural events are being systematically monitored out of the total available. Other than that, the exploit writer should also have knowledge of the analysis model being used to separate the malicious behavior from a benign one to detect malware in early stages. Assuming all this, the attacker can insert "NOP" instructions which are no operation instruction [18], before the shellcode code adequately so that the behavior exhibited by shellcode matches to the behavior exhibited by the benign process. These NOP instructions should be able to modify the behaviors of all the events being monitored to achieve evasion from the system. If the NOP instructions are inserted more than required that may raise the numbers for certain events and may make it easier for the detections system to detect the malicious intent.

Metamorphic Malware: Instead of adding instructions, the exploit builder could replace the already existing instructions inside the malware with the equal variants of those to perform the same functions. This behavior is common in metamorphic malware. This also assumes the attacker having full knowledge of the system and measuring techniques and specifics. To build this the attacker must profile each event from the equivalent code generated. Arguably this evasion strategy is extremely difficult to achieve.

CHAPTER 8

CONCLUSION

In this study, I proposed a setup for analysis and early detection of malware by measuring architectural and microarchitectural events fetched using hardware performance counters. The system of capable of analyzing and detecting the malware behavior with low overhead cost using HPC and feeding the data to a supervised learning model for detection. The system shows that it is possible to detect deviations in behavior for malware using small perturbations captured using hardware performance counters with low overheads. Due to anomaly-based supervised learning of models, this system is also capable of detecting zero-day malware. The tests were conducted using the latest viruses, worms, ransomware, Trojans etc. Fetching of events was done using Intel Vtune which is used to profile hardware events using counters already built inside with low overhead cost. Data was fed into support vector machine for supervised learning and analysis after that.

In previous chapters, I have also described working for multi-stage malware and how the malware is being upgraded to evade sandboxes to avoid detection. Chapters also mention techniques used to establish the system is not a sandbox but a real user. Lastly, I have also discussed the limitations of this system and how the attackers can avoid detection using stalling of code, adding NOP instructions or changing the existing instruction to equivalent code. All of these techniques require full knowledge of the system and events being monitored, which is surely challenging for an attacker to model and take evasion measures.

Lastly, this approach can be accompanied by existing software-based malware detectors for better and early stage detection of malware.

REFERENCES

[1]. FILIP TRUTA. *Own a Mac? You Could Be Infected With Proton RAT and Not Even Know It.* Bitdefender - 2017.

[2]. NICKY WOOLF. *DDoS attack that disrupted internet was largest of its kind in history, experts say.* The Guardian-2016.

[3]. *Cyber-attack hits 200,000 in at least 150 countries: Europol.* Reuters-2017.

[4]. DeepFreeze: Instant Restore Software.

[5]. Intel VTune: Performance Profiler.

[6]. Microsoft Autoruns: System Configuration Utility.

[7]. ADRIAN TANG SIMHA SETHUMADHAVAN SALVATORE STOLFO. *Unsupervised Anomaly-based Malware Detection using Hardware Feature. 2014.*

[8]. JOHN DEMME MATTHEW MAYCOCK JARED SCHMITZ ADRIAN TANG ADAM WAKSMAN SIMHA SETHUMADHAVAN SALVATORE STOLFO. *On the Feasibility of Online Malware Detection with Performance Counter.* ISCA 2013.

[9]. NAJMEH MIRAMIRKHANI, MAHATHI PRIYA APPINI, NICK NIKIFORAKIS, MICHALIS POLYCHRONAKIS. Spotless Sandboxes: Evading Malware Analysis Systems using Wear-and-Tear Artifacts. IEEE 2017.

[10]. Snort: Network Intrusion Detection and Prevention System.

[11]. CHRISTIAN ROSSOW, CHRISTIAN J. DIETRICH, CHRIS GRIER, CHRISTIAN KREIBICH, VERN PAXSON, NORBERT POHLMANN, HERBERT BOS, MAARTEN VAN STEEN. *Prudent Practices for Designing Malware Experiments: Status Quo and Outlook.* IEEE-2012.

[12]. VirusTotal: Sample Provider and analyzer.

[13]. HybridAnalysis: Sample Provider and analyzer.

[14]. Cuckoo Sandbox: An automated sandbox.

[15]. Security Report Report, Av-Test [July 2017].

[16]. DILSHAN KERAGALA. *Detecting Malware and Sandbox Evasion.*SANS-2016.

[17]. 5 Ways Advanced Malware Evades the Sandbox, Secureworks-2015.

[18]. Defeating Sleeping Malware, JoeSecurity-2012.

BIOGRAPHICAL INFORMATION

This report belongs to Anchal Raheja. Anchal has obtained a Bachelor's degree in Information Technology and Master of Science degree in Computer Science and he has successfully defended his master's thesis in Information Security under the supervision of Dr. Jiang Ming.

Anchal is pursuing his career in field of Information Security and Cyber Threat Intelligence. He has worked on number of projects in field of Information Security involving cloud service, endpoint protection, root cause analysis, application security and various platforms such as CrowdStrike, AppSpider, Wireshark, Kali Linux and IDS/IPS, etc. He was appointed as Graduate Teaching Assistant for Fall 2017.

He developed his expertise in Cyber Security by winning various CTF's and participating in bug bounty programs, which improved his understanding and polished his thought process.

Anchal is pursuing his career as Operation Security Engineer in S&P 500 company after completing an internship as application security intern.