Automated Forensics Analysis

Grace Harper

Junior Thesis
College of Computing
Georgia Tech
Spring 2019

Acknowledgements
Thank you to the graduate students for teaching me how to think.

Abstract

RAIN [2, 4] is a system which has the ability to record and replay system call events at the process level as well as perform dynamic information flow tracking (DIFT) on-demand during replay. This project's goal is to utilize RAIN to more efficiently determine the entrypoint of an attack as well as its effects on the victim's system. Specifically, this project attempts to use the low-level information recorded by RAIN to determine potential entry points of the attack.

1. Background and Related Work

In RAIN every system call, along with its arguments, are recorded so DIFT can be performed. Additionally RAIN creates an action history graph (AHG), which is a provenance graph representing data flow via system calls between all relevant files, processes, and network connections. The motivation behind RAIN is the ability to accurately and efficiently analyze Advanced Persistent Threats (APTs). Because APTs are designed to go undetected in a system due to their low-and-slow nature, they can be especially pernicious. RAIN currently allows users, after an attack has been detected, to replay any portion of the attack from its stages. The main advantage to using RAIN for analyzing APTs is its ability to accurately trace which information (i.e., file contents) within the system that the attacker modified or leaked. Another advantage is its low overhead during recording, it can be used in a production environment (e.g., industry or government). Consider that after the Equifax hack, Equifax needed to determine exactly which customers' information was divulged [7]. One of the drawbacks of analyzing APTs on RAIN is the overwhelming amount of data that could be recorded. Once the inception of the attack has been located, tainting analysis frameworks such as DYTAN, a generic dynamic taint analysis framework, which has already proven itself efficient on the Firefox browser [5], can be employed for further analysis. However, since RAIN is designed to run for years at a time, the amount of system calls and nondeterminism could be overwhelming to sift through to (1) find the attack's root cause and (2) find the attacks effects on the system. The user must have an exact idea of when the initial infiltration of the attack occurred if the entire attack is to be replayed and analyzed [3]. It is not always reasonable to assume that the user will have knowledge of the location of the attack's root cause or what it has affected within the system. It is important to note that because DIFT is fine-grained, it has a large overhead time. Because of this and the large scale nature of the recorded data, using DIFT to not only track, but to locate the attack by naively tainting all recorded data on the system is infeasible. Therefore, different methods of determining the starting point of the attack are necessary. Previous research in APT detection has been done through systems like Dorothy's Intrusion Detection Model [8] , USTAT [9], and HOLMES [1]. In order to attempt anomaly detection, we began with an older but simpler model that did not require any foreknowledge of security, Dorothy's Intrusion Detection Model [8]. This model was developed in 1980. The goal here was to see if information generated with no knowledge of the system could provide value as anomaly detection software. When that model proved insufficient, we turned to a slightly more modern model, USTAT [9]. Published in 1993, this paper's detection model uses prior knowledge of security, matching audit log data onto generic and simple APT patterns. The goal here was to see what advantages prior knowledge, as opposed to pure maths alone, could give to a system. After seeing mild improvement, we began work on

recreating HOLMES[1] , a highly sophisticated APT detection model for provenance graphs. Published in 2018, HOLMES looks at MITRE's ATT&CK framework [6] which defines several behavioral patterns called Tactics, Techniques, and Procedures (TTPs). Using these TTPs as building blocks, HOLMES develops a generic model for APTs then looks for TTPs that form a pattern closely matching that of an APT's stages and behaviors. The goal here was to see if the complicated patterns used by HOLMES could be simplified and used for anomaly detection on RAIN.
In total, in attempt to find a fast and accurate detection system to build off of, we tried three different detection models, where each successor was more sophisticated than the previous but also required more information processing.

2. Motivating APT Scenarios

RAIN's developers intend for the record and replay system to be useful for on-host APT activities. For this to be possible, discovering where in the system an attack begins is crucial. We wanted to consider common attacks often seen in industry, specifically those tested by the DARPA Transparent Computing Group. Despite the fact that these attacks were shorter than a few days, they easily produced millions of logs. This meant there was a large amount of data to analyze in order to pinpoint where the attacks began and what they effected. Since our user's machine was running RAIN, every part of the attack could be reconstructed and DIFT could be performed on the recorded system calls to better analyze the attack. However, in order to know where to begin tainting, the data had to be filtered down to a manageable amount. The challenge here for us was to significantly reduce the search space.

3. Technical Plan

For this project, there were three methods tested with RAIN to see if any reduction in search space for the attack could be made: Approach 1, Frequency, modelled after Dorothy's Intrusion Detection Model [11], Approach 2, Sensitive Files, modelled after USTAT [9], and Approach 3, APT Patters, modelled after HOLMES[1]. While Approach 1 and 2 were completed and analyzed, Approach 3 is still currently being tested.

3.1 Approach 1: Frequency

The first approach was naive. It did not assume any prior knowledge about the system nor about current understandings of security. It focused only on looking for anomalies in frequency of regular system calls. The program assumed that a malicious attack would result in a sudden spike in system call frequency beyond some normal threshold.
The threshold was developed as such: in small intervals of about 50 minutes, the number of system calls per second were recorded over time, then plotted. The threshold was the number of system calls per second one standard deviation away from the mean. When run on an actual dataset, any averages that went above the threshold were considered potential threats, creating an alert.

3.2 Approach 2: Sensitive Files

The second approach was to implement part of USTAT [9]. USTAT has 12 predefined patterns (STDs) generally associated with an APT. If a system's actions match every part of a pattern chronologically, USTAT generates an alert.
Our initial attempt was to recreate one of USTAT's escalation privilege detection patterns. However, this required determining when a user gains root privileges without explicitly calling root.
It was confirmed that RAIN's CDM records did not differentiate consistently between the privilege levels of a current user. This discovery rendered the majority of USTAT's patterns, which focused predominantly on privilege escalation, useless since the patterns required being able to determine which user had which privileges at any given point in time.
Many of the remaining USTAT patterns were already obsolete for our purposes because they dealt either with SunOS-specific vulnerabilities or UNIX components that are no longer popularly used, such as the mail server.
However, some of USTATs patterns were still viable, specifically STD 11 which concerns itself with sensitive files. The author of USTAT precompiled a list of sensitive files and directories, categorizing them by read/write sensitivity. After modifying it to include sensitive files in Ubuntu 12, it was this sensitive directory/file STD that we implemented. If any file or directory was written to or read by a program without the necessary privileges, that timestamp was recorded as an alert.

3.3 Approach 3: HOLMES

While there have been complications preventing Approach 3 from being tested, the goal is clear. We want to utilize HOLMES' sophisticated APT patterns to detect the Engagement attacks in RAIN on a Neo4j provenance graph. If these prove useful in significantly reducing the graph without removing any part of the attack, we can begin using fine-grain analysis on the remaining graph to find and follow the attack.

3.4. Data Used

Currently, RAIN is being tested by the DARPA Transparent Computing Project. All data used in the development and testing of this project came from these DARPA Engagements
There are four total Engagements in which RAIN has been tested so far.
All graphs and results in this paper for Approaches 1 and 2 are from testing on Engagement 4. Approach 3 will be tested on Engagement 3.

3.4.1 Approach 1 and 2

Unlike the original Intrusion Detection Model by Dorothy [11] and the USTAT system for Unix by Ilgun [2], both Approach 1 and Approach 2, though designed to be utilized in real time, were tested only on static data. During each Engagement, RAIN recorded everything in the kernel. This information was then used to generate Common Data Model, CDM, files in bulk after the completion of an attack. These CDM files were a json-formatted collection of logs in chronological order. Each log was either a system call event, such as a read, or kernel object,

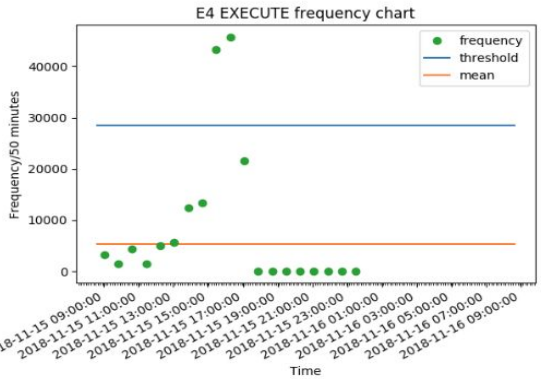
such as a file object. For both Approach 1 and 2, early versions of the software read in the CDM data line by line, immediately processing the results of each line before reading another. This was done in an attempt to simulate real time performance. However, because this was slow and ultimately did not help address whether the methods being tested would yield accurate results, later versions of Approach 1 loaded all CDM data into the PostgreSQL database directly before manipulating. Later versions of Approach 2 used a hybrid method, explained in more detail in the Approach 2 section. However, both approaches can be modified to begin taking in streaming data.

3.4.2 Approach 3

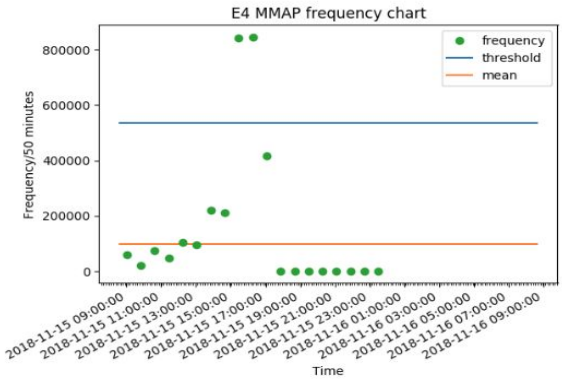
All graphs and results in this paper for Approach 3 are from testing the models on Engagement 3 data. Unlike the other two Approaches, Approach 3 was not designed to detect the attack in real time, but rather to prune a provenance graph for the reconstruction of a previously detected attack. Instead of reading in a CDM data file, Approach 3 used a Neo4j provenance graph where objects such as processes, networks, and files became nodes and events causing information flow, such as writes or mmaps, became directed edges. Engagement 3 was used because it was the only dataset for which there was a labelled graph available, allowing for more precise analysis of the performance. Because of the unique nature of Approach 3, it seemed fitting to use this earlier data set despite Approaches 1 and 2 being tested on Engagement 4.

4. Results

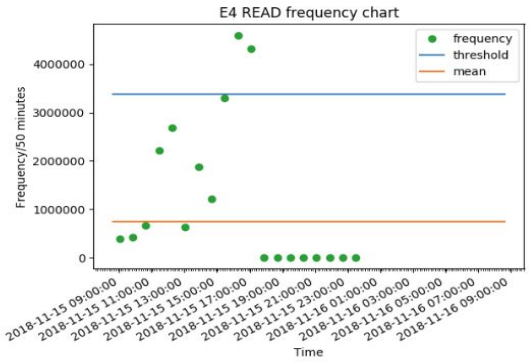
4.1 Approach 1



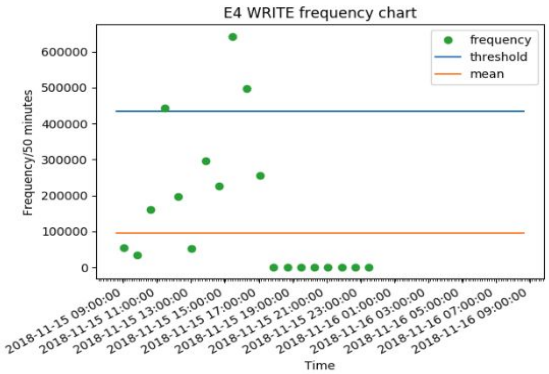
False Positive	0
False Negative	3
True Positive	2
True Negative	14



False Positive	0
False Negative	3
True Positive	2
True Negative	14



False Positive	0
False Negative	3
True Positive	2
True Negative	14

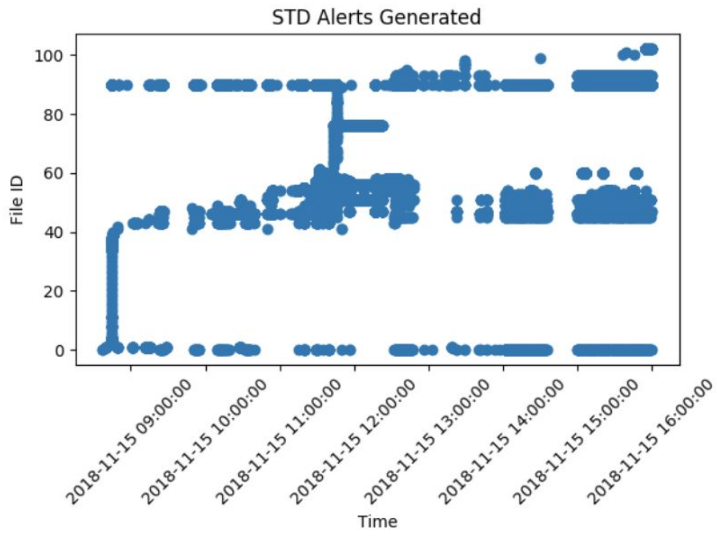


False Positive	0
False Negative	2
True Positive	3
True Negative	14

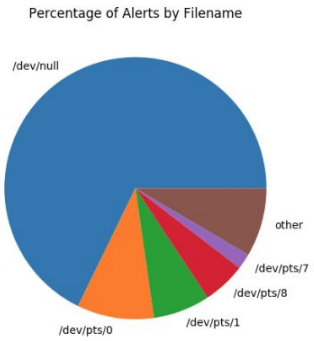
Approach 1 measured the frequency of READ, WRITE, MMAP, and EXECUTE system calls. Multiple threshold levels and intervals were tried, but all yielded similar results. For the graphs displayed below, the threshold is one standard deviation above the average. The interval used is ~50 minute intervals, which allowed for the most precise graphs that could still be generated in a reasonable period of time, less than two days.
For the first round of attacks, the detection system performed hourly checks and generated an alert if the frequency for that hour was above the determined threshold. On the surface, this detection system seemed to work fairly well. Generating 9 alerts over the course of one day for 4 attacks seemed reasonable. However after a comparison with the ground truth, it was found that no anomalies in the graph correlated accurately with a significant part of the attacks. Furthermore, almost no background noise from benign programs was being generated. Had it been, the threshold would have increased, potentially high enough to completely wipe out all alerts. Studying further, it became clear that system call frequency was not correlated well with an attack. This led us to consider our next solution.

4.2 Approach 2

STD 11, unlike Dorothy's Intrusion Detection system, utilized security knowledge so it was expected to perform better. RAIN's Engagement 4 included the Pine Metasploit backdoor attack and the Micro SSH APT. Both worked by creating/opening a file /dev/glx_alsa_675. Since /dev/ was listed as a "sensitive-write" directory, we used STD 11, an STD focused on sensitive file reads and writes, to catch the attack. We knew then, going in, that this model would definitely catch the attack.



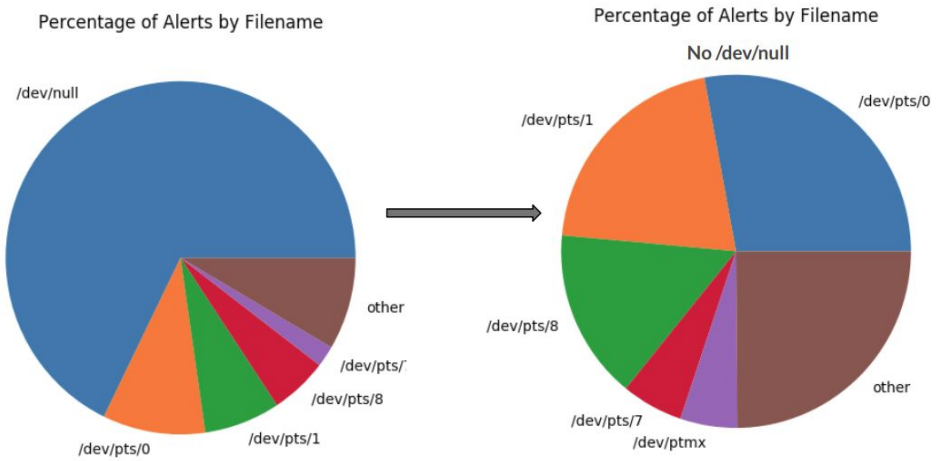
STD Alerts with /dev/null



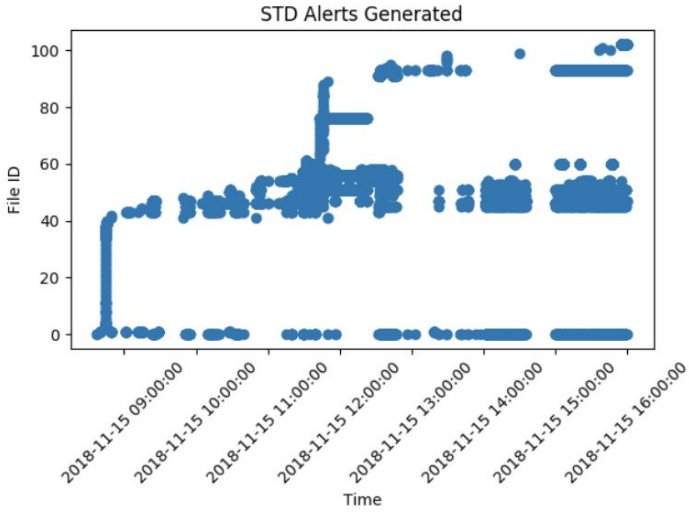
False Positive	467018
False Negative	0
True Positive	83203
True Negative	0

However, the results were insufficient. Our first testing on Engagement 3, figure 2, showed USTAT generated alerts not only during an attack but during the majority of the time the

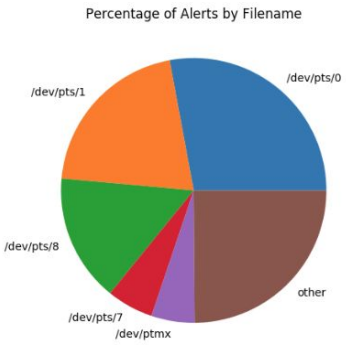
computer was turned on. This made STD 11 impractical because of the vast number of false positives.



Further inspection revealed that /dev/null was causing a majority of the false positives. Because /dev/null is essentially the garbage can of the OS, we felt comfortable whitelisting it. We did so and reran the program. This drastically reduced the number of alerts generated.



STD Alerts without /dev/null



False Positive	84332
False Negative	0
True Positive	92874
True Negative	0

However, the reduction in false positives was did not render the system practical as it still stood true that alerts were generated more often than not, making it difficult for a user to sort out false positives from true positives. It become clear that the simplistic approach taken by our modified USTAT was insufficient for accurate anomaly detection.

4.3 Approach 3

As discussed earlier this third approach is still in progress. While several metrics were generated from the provenance graph, there were major setbacks with the graph. The first setback came much earlier when it was found that the data loaded into the graph was without timestamps due to an error. This meant the metrics we originally wanted to record were infeasible. New metrics had to be thought up and tested. Then, recently, it was discovered that the graph itself did not appear to accurately reflect the generated CDM data. At this point, all results gotten were deemed invalid and dismissed. Because these queries require a decent amount of time to run and these issues were discovered recently, there are no concrete results from Approach 3. However, it is in progress.

5. Discussion

This semester, we explored traditional methods of anomaly detection by imitating papers such as Dorothy's Intrusion Detection System[8] and USTAT [9] on the new data generated by RAIN. The traditional methods were created for less-dense information audit records available on most UNIX systems. From the work done this semester, it has become clear that historical methods of anomaly detection are insufficient for modern needs. Besides being impractically slow, they did not yield significant results that could be useful in real-life scenarios.

Detection software for RAIN must be developed at the intersection of security and mathematics, not one alone. Record and Replay systems make it valuable for researchers to be able to detect anomalies in aggregated historical data. For RAIN, the focus should be on designing more comprehensive data models that can be utilized to weed out the background noise of a modern day system. We anticipate the methodology of HOLMES will prove more useful than its predecessors.

6. Future Steps

We anticipate being able to use APT pattern matching, similar to that done in HOLMES[11] to begin weeding out benign information from the data generated by RAIN. If the search space is sufficiently reduced by matching these APT patterns, we can begin using DIFT. If not, we will try other methods to reduce the benign noise generated by RAIN.

References

1. S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan, "HOLMES: Real-time APT Detection through Correlation of Suspicious Information Flows," arXiv preprint arXiv:1810.01594, 2018.
2. Yang Ji, Sangho Lee, Evan Downing, Weiren Wang, Mattia Fazzini, Taesoo Kim, Alessandro Orso, and Wenke Lee: Rain: Refinable Attack Investigation with On-demand Inter-Process Information Flow Tracking. in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. Dallas, USA, 2017, pp. 377–390.
3. Y. Ji, S. Lee, M. Fazzini, J. Allen, E. Downing, T. Kim, A. Orso, and W. Lee, "Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking," in 27th USENIX Security Symposium (USENIX Security 18). Baltimore, USA, 2018, pp. 1705–1722.
4. Y. Jang, S. P. Chung, B. D. Payne, and W. Lee. Gyrus: A framework for user-intent monitoring of text-based networked applications. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, February 2014.
5. J. Clause, W. Li, and A. Orso. Dytan: A Generic Dynamic Taint Analysis Framework. In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2007), pages 196–206, London, UK, July 2007.
6. Adversarial tactics, techniques, and common knowledge. https://attack.mitre.org/wiki/Main_Page
7. Equifax Hack PR Website. https://www.equifaxsecurity2017.co
8. Dorothy E. Denning. 1987. An Intrusion-Detection Model. IEEE Trans. Softw. Eng. 13, 2 (February 1987), 222-232. DOI=http://dx.doi.org/10.1109/TSE.1987.232894
9. Ilgun, Koral. (1993). USTAT: a real-time intrusion detection system for UNIX. 16-28.10.1109/RISP.1993.287646.