Machine Learning-Based Detection of Ransomware Using SDN

Greg Cusack
University of Colorado Boulder
gregory.cusack@colorado.edu

Oliver Michel
University of Colorado Boulder
oliver.michel@colorado.edu

Eric Keller
University of Colorado Boulder
eric.keller@colorado.edu

ABSTRACT

The growth of malware poses a major threat to internet users, governments, and businesses around the world. One of the major types of malware, ransomware, encrypts a user's sensitive information and only returns the original files to the user after a ransom is paid. As malware developers shift the delivery of their product from HTTP to HTTPS to protect themselves from payload inspection, we can no longer rely on deep packet inspection to extract features for malware identification. Toward this goal, we propose a solution leveraging a recent trend in networking hardware, that is programmable forwarding engines (PFEs). PFEs allow collection of per-packet, network monitoring data at high rates. We use this data to monitor the network traffic between an infected computer and the command and control (C&C) server. We extract high-level flow features from this traffic and use this data for ransomware classification. We write a stream processor and use a random forest, binary classifier to utilizes these rich flow records in fingerprinting malicious, network activity without the requirement of deep packet inspection. Our classification model achieves a detection rate in excess of 0.86, while maintaining a false negative rate under 0.11. Our results suggest that a flow-based fingerprinting method is feasible and accurate enough to catch ransomware before encryption.

CCS CONCEPTS

• Security and privacy → Malware and its mitigation; **• Networks → Network monitoring**; **• Computing methodologies** → *Classification and regression trees*;

KEYWORDS

ransomware; malware; software-defined networking; machine learning; stream processing; programmable forwarding engines

ACM Reference Format:
Greg Cusack, Oliver Michel, and Eric Keller. 2018. Machine Learning-Based Detection of Ransomware Using SDN. In *SDN-NFV Sec'18: 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, March 19–21, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3180465.3180467

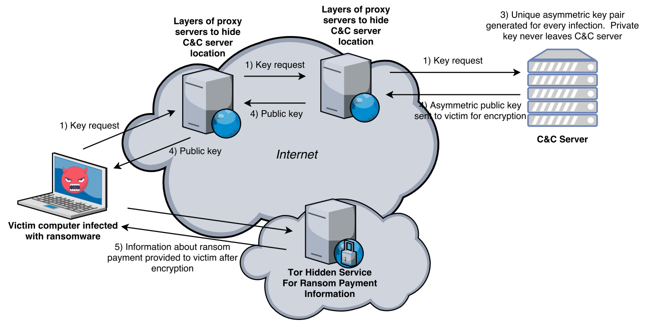

Layers of proxy servers to hide C&C server location
Layers of proxy servers to hide C&C server location
3) Unique asymmetric key pair generated for every infection. Private key never leaves C&C server
1) Key request
1) Key request
4) Public key
4) Asymmetric public key sent to victim for encryption
C&C Server
1) Key request
4) Public key
Internet
Victim computer infected with ransomware
5) Information about ransom payment provided to victim after encryption
Tor Hidden Service For Ransom Payment Information

Figure 1: Operation of typical ransomware encryption key retrieval process [5].

1 INTRODUCTION

In recent years, the prevalence of malware, has increased dramatically. In fact, ransomware has grown into one of the most prominent strains of cybercrime. We're seeing more cases of ransomware in 2017 than we have ever seen before due to its ability to autonomously propagate across the network [6]. Clearly, ransomware mitigation techniques need to be designed in order to prevent successful attacks of malware. Luckily, there has been some work in the detection and mitigation of malware [4, 5, 7]. However, these studies focus on ransomware identification delivered through HTTP. Unfortunately, malware delivery is shifting heavily to HTTPS as 37% of all malware now utilizes HTTPS as of June, 2017 [9]. We need a longer term approach that utilizes network features only available in TLS traffic. Furthermore, the work in [4] sacrifices the wellbeing of one computer in order to identify malicious servers sending and controlling malware. In this paper, we leverage advances in SDN to address the ransomware problem. Specifically, we utilize the emergence of PFEs (e.g. P4 switches), write a stream processor, and implement machine learning to identify and intercept ransomware before it enters a network.

Ransomware is a software virus that holds a victim's files at ransom. Access to the files is not returned until a ransom is paid. There are two main types of ransomware in circulation today, crypto and locker-based ransomware. Crypto ransomware encrypts the files on a victim's computer and will only provide the decryption key for the files if a ransom is paid. On the other hand, locker ransomware leaves the victim's computer files intact but locks the user out of his or her computer, only returning access once a ransom is paid. Unfortunately, detecting various types of ransomware is an arduous task. Developing a long term solution to ransomware detection has proven difficult since ransomware developers are constantly updating their product to circumvent new detection techniques. Furthermore, many flavors of ransomware are delivered via botnets [7], and as the IoT sector grows rapidly, the number of avenues for infection are increasing dramatically. We have also seen the emergence of Ransomware as a Service (RaaS), where a novice cybercriminal can pay a service and easily customize his or her own ransomware and have it spread to many computers around the world [15]. Despite the growing number of ransomware cases, the underlying method for how the two methods operate are similar. They both require communication with a C&C server in order to carry out an attack. This communication between the infected computer and the C&C server is what we exploit in our classifier.

Figure 1 shows the communication between the infected computer and the C&C server. In order to encrypt the victim's files, the victim requests an encryption key from the C&C server through multiple layers of proxies. The C&C server generates a new asymmetric key pair, keeps the private key, and returns the public key to the victim to encrypt its files. After encryption, a Tor hidden service communicates a method for paying the ransom. By analyzing the traffic flowing between the victim's computer and the proxies residing in the greater Internet, we're able to develop a classification model that identifies the encryption key retrieval process.

Previous work has shown that even if the victim has received the initial infection through a phishing email, for example, if the C&C server cannot deliver the encryption key, the malware cannot carry out the attack [5]. As a result, we look at the network traffic between the victim's computer and the C&C server in hopes that we can identify malicious communication, and prevent the delivery of the encryption key.

In order to accurately monitor all traffic going into and out of the potential victim, we leverage the recent emergence of programmable forwarding engines (PFEs). PFEs utilize switch hardware and dynamic memory caches to achieve high packet processing speeds while simultaneously providing rich flow records. These PFE-generated flow records, provide per-packet information and allow us to extract flow features for ransomware classification at line rate in an accurate and scalable manner.

The rest of the paper is broken up into sections as follows. Section 2 discusses and outlines previous work in the field of ransomware detection and PFEs. Section 3 dives into our system architecture, starting with a description of the framework we used for our stream processor and finishing with a motive for random forest. We describe the implementation of our design in Section 4 and analyze the results of the classifier in Section 5. Finally, Section 6 summarizes our application and discusses future work.

2 RELATED WORK

Two areas of related work help us in designing our ransomware detection application. Ransomware detection has been a large area of study in recent years; however, many of these solutions fall short as ransomware developers adjust their malware delivery methods. We also look at the emergence of PFEs, the programmable hardware we leverage for rapid per-packet, flow processing.

2.1 Ransomware Detection

One method of ransomware detection used machine learning to identify and classify various types of ransomware during the ransomware installation phase on target hosts. The authors mainly relied on Windows API calls, file system operations, registry operations, etc. to classify malware. Their ransomware classifier, EldeRAN, was compared to various other machine learning algorithms such as SVM and Näive-Bayes and produced a much higher true positive rate and a lower false positive rate [13]. However, EldeRAN requires the infection of a system in order to learn ransomware behavior.

Another group of researchers used an SDN approach to ransomware identification by utilizing deep packet inspection to track the packet lengths of HTTP POST messages [4]. Once ransomware was identified, the command and control server IP addresses were identified and blocked. However, this technique results in a relatively high false positive rate (up to 4.95%), leaving their method open to a base rate fallacy issue and falsely blocking valid servers.

In fact, most malware and ransomware detection methods that look at traffic traces, like the one above, are payload-based [4, 5, 16]. These network-based approaches to ransomware detection all share the same, previously described problem of relying on DPI, and therefore, are useless for fingerprinting on encrypted traffic.

2.2 Recent Hardware Trends and PFEs

In recent years, we have seen the development of a few high rate stream processing systems, which utilize switch hardware to generate network information-rich flows [3, 8, 10, 14]. PFEs allow commodity networking equipment to support the scalable generation of rich flow records. The recent trend of PFEs and the accompanying efforts to make programming them more accessible has enhanced the use and development of PFEs [2].

PFEs allow us to process network data at high rates of speed, while still extracting vital, per-packet flow information. The growth of PFEs and rich flow generation systems, provide us with the data and speed necessary for network, flow-based ransomware classification.

3 SYSTEM ARCHITECTURE

Our system's architecture is broken into two main parts, stream processing and classification. The stream processor reads from a PCAP, runs and manages a custom flow table, and extracts flow features for our classifier. The classifier takes in the extracted features and trains a model to identify ransomware.

3.1 Stream Processing

In order to process rich flow records, we utilize RaftLib's stream processing library to build high-performance, parallel, analytics applications [1]. Each kernel we wrote using RaftLib runs a step in the flow processing chain. We link multiple of our kernels to group incoming packets into their respective flow records based on each packet's 5-tuple. The 5-tuple, which consists of the packet's protocol, source IP, source port, destination IP, and destination port, serves as the flow record's key. The kernel-based approach allows us to utilize RaftLib's parallelization feature. Since we read in network traffic from PCAP files, we use a custom flow table and implement it as a kernel running in parallel with the other kernels. We simulate the generation of rich flow records and use the RaftLib framework to write a parallelized, stream processor for flow feature

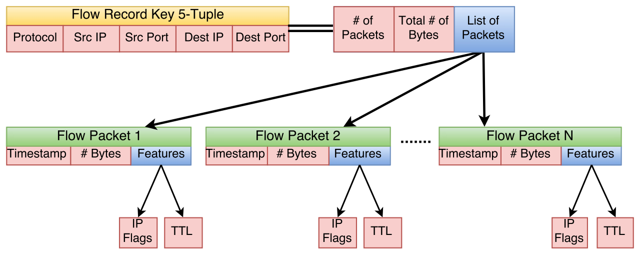


Figure 2: Compact and per packet flow records created in a hierarchical manner. The 5-tuple serves as the key for matching packets in the same flow.

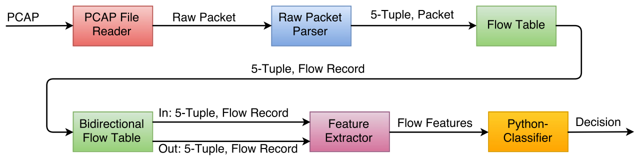


Figure 3: All boxes except the Python-classifier are kernels we wrote for stream processing. We built the kernels to convert a PCAP to a set of flow records for feature extraction. Each kernel executes one step in the flow processing system.

extraction at line rate. These extracted features are then used for ransomware classification.

3.2 Classification

We implement a random forest classifier in Python due to the random forest's low computational training cost and its use of bagging to reduce variance and overfitting. A random forest classifier is an ensemble algorithm, which utilizes a collection of decision trees to vote and predict the class of the input data. Each decision tree is created from a random subset of the feature set. Each decision tree is generated using the gini impurity metric, which measures the probability of mislabeling a randomly chosen element from the training set if the element was labeled based solely on the distribution of the binary labels in the set [11].

Three of the main tuning metrics for a random forest classifier include the number of decision trees in the forest, the depth of each decision tree, and the maximum number of features that can be included in each decision tree. The number of trees in the forest dictate the performance and variance of the classifier. A larger number of trees results in higher classification accuracy and lower variance but increases the computational cost of the classifier. The depth of each tree has a similar cost-benefit situation. As the depth of each tree increases, the induced bias in the classifier decreases; however, the added depth comes with a computational penalty.

The last main metric we used for tuning our random forest classifier is the maximum number of features that can be included in each decision tree. The maximum number of features is used to determine the best split when creating a decision tree. Once again, increasing the number of features increases performance but comes at a computational cost. In the next section, we discuss our implemented application starting with our stream processor and finishing with the ransomware classifier.

4 IMPLEMENTATION

4.1 Flow Records and Processing Kernels

We wrote five kernels on top of the RaftLib framework for processing network data and creating compact and rich flow records. Figure 2 shows the structure of our flow record. The 5-tuple serves as a key for each flow, which links to the number of packets and bytes in the flow along with a reference to specific packet features. The packet features include the packet timestamp and the number of bytes in the packet. Each flow packet also contains a link to the packet's IP flags and time to live (TTL). We utilize the data in these flow records to extract features for our ransomware classifier.

Figure 3 shows the kernels we wrote for flow generation and feature extraction. Normally, the per packet, flow records seen in Figure 2 would be generated in PFE hardware, but since we are reading from a PCAP, we wrote three kernels to simulate the rich, flow record generation process. The initial PCAP file reading kernel reads in a PCAP and outputs a raw packet, which is immediately read in and processed by the raw packet parser. The raw packet parser extracts the 5-tuple from the packet and sends the 5-tuple along with the packet features as a key-value pair to the flow table kernel. We wrote a custom flow table to do most of the packet processing and memory management. The flow table stores a map of flow records, where the key is the 5-tuple and the value is a list of packets that are members of the flow represented by the 5-tuple. When a new incoming 5-tuple and packet arrive at the input of the flow table kernel, the kernel looks for the arriving 5-tuple in its stored flow table. If the 5-tuple is found, the incoming packet features are appended to the list of packets corresponding to the packet's 5-tuple key. If the incoming packet's key is not found, then a new entry in the flow table is created.

Unfortunately, flows are direction dependent. In a client's communication with a server, two flows are extracted. One flow corresponds to the client-to-server communication, and the other flow correlates with the server-to-client communication. In order to look at traffic burst patterns and extract other features requiring knowledge of corresponding flows in opposite directions, we wrote a bidirectional flow table kernel. Similar to the preceding flow table kernel, the bidirectional flow table manages a list of flows. However, flow records are matched with each other when an incoming flow record's source IP and source port match another flow record's destination IP and destination port and vice versa. If a match is found, the two flows are exported out of the bidirectional flow table to the next kernel. If a flow match is not found, the incoming flow is added to the bidirectional flow table and waits for a match.

After two flows are matched, they are exported to the feature extraction kernel. The feature extraction kernel takes in both flow records and performs calculations using the features as seen in Figure 2. Our classifier makes use of two main types of flow features, direction independent and direction dependent. Direction independent flow features are features that do not require the knowledge of the corresponding flow traveling in the opposite direction. Flow independent features include flow duration, packet interarrival times, total number of packets and their respective lengths, and the number of unique packet lengths.

Direction dependent features are flow features that rely on knowing the features of the flow traveling in the opposite direction on the same connection. Direction dependent features include burst lengths, the ratio of outgoing to incoming packets, and the ratio of outgoing to incoming bytes. Burst lengths are defined as a sequence of outgoing packets which contain no two adjacent incoming packets. The feature extraction kernel calculates the two classes of flow features and passes them to the Python-based classifier.

4.2 Ransomware Classifier

As mentioned in Section 3.2, we tune our random forest using three main parameters: the number of trees in the forest, the depth of each tree, and the number of features used in each tree. Since the end goal is to run our classifier at line rate, we want as many trees as possible without adding significant overhead. As a result, we use 40 trees in the forest, and set the depth of each tree to 15. It should be noted that increasing the number of trees and the depth of each tree has diminishing returns. We tested numerous combinations of total decision trees and decision tree depth and found that increasing the number and depth of trees from 40 and 15 respectively resulted in minimal classification accuracy gains. Finally, due partly to convention and mainly to the high computational cost of decision tree feature splitting, we set our maximum features parameter to the square root of the total number of features in our dataset. This reduction in features greatly improves the learning time of the tree without a noticeable loss in classification performance.

5 RESULTS

In this section, we present the composition of our dataset and the metrics that define success for our classifier. We also investigate the performance of our classifier in identifying ransomware as a whole. We then move on to discuss how well our classifier can identify a specific type of crypto ransomware.

5.1 Data Collection

We collect over 100MB of ransomware traffic traces from *malware-traffic-analysis.net*, resulting in 265 unique bidirectional ransomware-related flows. We collect another 100MB of network traffic that is malware free (clean) to use as a baseline. The clean data consists of flows corresponding to web browsing, file streaming, and file downloading. When analyzing the ransomware traffic, we analyze the traffic to and from the infected machine in communication with the C&C server. We combine both the ransomware and clean traffic and feed it into our stream processor to extract features for the classifier.

5.2 Success Metrics

We next discuss our success metrics, which help us determine whether or not we have produced a strong classifier. For our first success metric, we look at the recall of our classifier. The recall deals with the classifier's false negative rate. In the future, we plan to implement our system in a real-world setting to catch ransomware before it encrypts a user's computer. To do so, we need to ensure that our false negative rate is as low as possible to prevent misclassifying ransomware as clean traffic.

We next look at the false positive rate of the classifier in determining its success. The false positive rate describes how often clean traffic is misclassified as ransomware. The false positive rate also needs to be as low as possible to prevent the unwarranted blocking of clean traffic. Furthermore, a high false positive rate results in a base rate fallacy issue, which quickly results in a massive number of falsely identified ransomware traffic.

To measure the classifier's success, we also look at the F1 score. The F1 score is a weighted average of the recall and precision scores and provides an idea of the balance between the false negative and false positive rates.

5.3 Feature Selection

We select our features based on the nature of the victim computer's communication with the C&C server. Since communication with the C&C server runs through multiple layers of proxy servers, we expect a higher than normal traffic latency. We extract this increased latency by measuring packet interarrival times. Furthermore, we also expect more incoming than outgoing traffic from the victim computer due to the downloading of the initial infection, the encryption key retrieval process, and the payment method notification from the Tor hidden service. We collect data to test this expectation by extracting the inflow to outflow packet ratios and burst lengths, where a burst length is the number of incoming packets before two adjacent outgoing packets are registered. The combination of interarrival times, packet ratios, and burst lengths can help distinguish a clean download from a malicious download through proxy servers.

5.4 Initial Classification Model

We first tune our stream processor to extract 28 unique features from our collected network traffic. These features are fed into the classifier, which first ensures the data contains the same number of malicious flows as clean flows in order to prevent classification bias. The data is then split into two, unequal sets. One set consists of 70% of the data and is used for training and the other set holds the remaining 30% of traffic and is used for testing the learned model. A 10-fold cross validation (CV) is performed on our data splitting to ensure our splitting model is unbiased. The confusion matrix in Figure 4 shows the results of our classifier using 28 different features. Even with a smaller set of traffic data, ~200MB, we are able to achieve a respectable recall of 0.89, a precision of 0.83, and an F1 score of 0.87. If we take a look at the corresponding ROC curve in Figure 7a, the area under the curve is 0.935, showing promise for successful ransomware detection. Furthermore, the average of the 10-fold CV score for our model is 0.87, indicating that we can expect similar accuracy results on other datasets.

5.5 Feature Reduction

Feature reduction is a key method used in machine learning to increase classification accuracy while simultaneously reducing the computational cost of the model. In order to reduce the number of feature in our model, we identify the top eight most influential features in classifying ransomware traffic, as seen in Figure 5. The

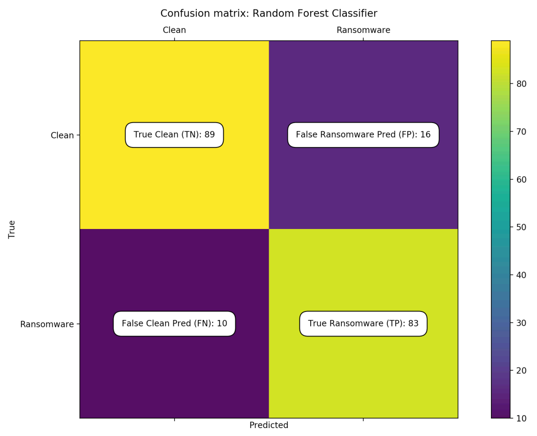


Figure 4: The confusion matrix of our 28-feature random forest classifier shows a recall of 0.89, a precision of 0.83, and an F1 score of 0.86.

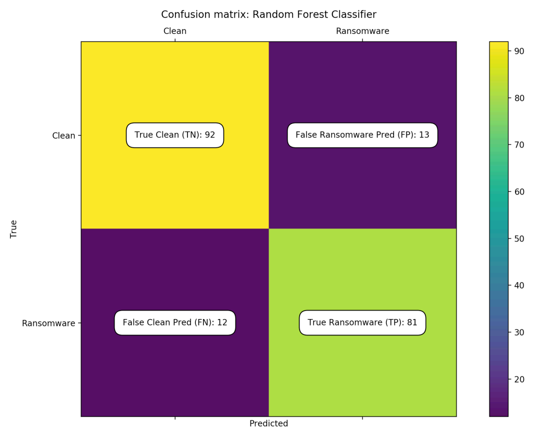


Figure 6: The confusion matrix of our 8-feature classifier shows similar results to that of our 28-feature classifier with a recall of 0.87, precision of 0.86, and F1 score of 0.87.

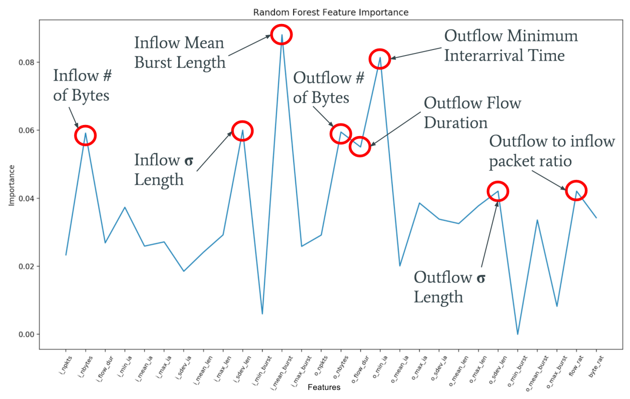
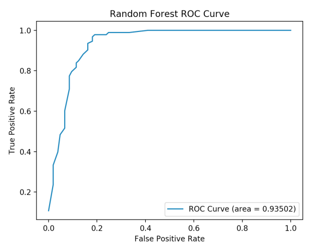
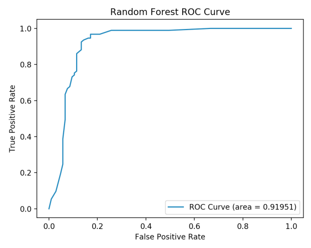


Figure 5: The plot above shows the weights of each of the 28 features in classifying ransomware traffic. The top 8 most important features are circled in red and labeled. We use these 8 features to train a new classifier.



(a) 28-Feature ROC Curve. AUC: 0.93



(b) 8-Feature ROC Curve. AUC: 0.92

Figure 7: Comparison of ROC Curves for the 28-feature and 8-feature classifiers

eight features are made up of mostly inflow and outflow length and interarrival time metrics. These eight features, which are circled in red and labeled in Figure 5 are used to develop a new random forest model for ransomware classification.

After training a model using only the inflow and outflow number of bytes, inflow and outflow standard deviation of packet lengths, inflow mean burst length, outflow minimal interarrival time, and the outflow to inflow packet ratio, we test our model and produce similar results to our classifier using 28 features. The confusion matrix of our 8-feature classifier can be seen in Figure 6. It is clear when comparing Figures 4 and 6 that the reduction in features has little impact on the classification accuracy. The 8-feature model has a slightly lower recall score at 0.87 but produces a higher precision and F1 scores of 0.86 and 0.87, respectively. However, Figure 7b shows a slightly smaller AUC for the 8-feature ROC indicating that the 8-feature classifier performs about 1.4% worse than the 28-feature model. This slight performance loss will be worth the computational savings when running classification at line rate.

5.6 Cerber Ransomware Detection

After running a classifier to detect all types of ransomware communication with a C&C server, we looked into specifically classifying Crypto-based Cerber ransomware, a ransomware which infected over 150,000 users in 2016 [12]. Cerber is a RaaS-type ransomware, which allows any nontechnical adversary to create and distribute their own ransomware. We chose to classify Cerber specifically due to its large infection footprint and its availability to anybody who wants to deploy ransomware.

We extract Cerber's eight most important network features, which include the mean and maximum burst lengths of the inflow stream, and create a random forest model for predicting Cerber ransomware. While we use a smaller sample size than in our previous tests, we are able to achieve a false negative rate of 0.0% and a false positive rate of 12.5%. Figure 8 shows the confusion matrix of the classifier. Furthermore, the ROC curve also attains a high AUC of ~0.987.

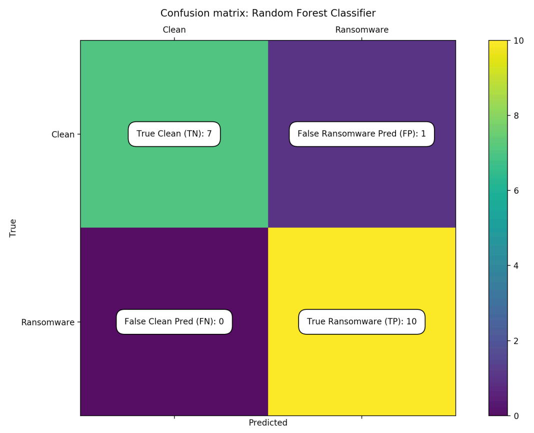


Figure 8: The confusion matrix of the Cerber classifier shows zero false negatives with a 12.5% false positive rate and an F1 score of 0.94. The initial findings are promising as we move forward in collecting more ransomware traffic.

It should be noted that our 10-fold CV score average comes in at 0.905, indicating that as we use the Cerber classifier on more network traffic, we are likely to see a slight rise in false negatives and false positives.

While we use a small sample size for classifying Cerber traffic, the results indicate that our machine learning approach may have more success in classifying specific types of ransomware rather than ransomware as a whole. While the underlying method for distributing and launching ransomware is similar, the individual traffic shapes likely differ slightly across ransomware flavors based on the ransomware developer. We leave this investigation to future work.

6 CONCLUSION & FUTURE WORK

In this paper we present a method for detecting ransomware via its network traffic signature. We utilize the high processing rate of new hardware-based flow generators in combination with RaftLib's high performance and parallel framework to process rich flow records, extract flow features, and classify ransomware. Since malware communication is moving towards HTTPS for delivery and control, we only utilize the unencrypted features of HTTPS traffic for model creation. We write a stream processor using five kernels to process rich flow records and extract high-level flow features for use in our random forest classifier. When monitoring the communication between the infected machine and the C&C server, we are able to significantly reduce our initial feature set and achieve a detection accuracy rate of almost 87%, while maintaining a strong false negative rate close to 10%.

In the future, in order to classify traffic at line rate, we plan to write our classifier in C++ as a stream processing kernel. By speeding up the classification step, we'll be able to identify ransomware before the infected computer receives the encryption key from the C&C server. Furthermore, some ransomware variants utilize UDP to communicate with their C&C server. Our current implementation does not take the UDP protocol into consideration. Adding UDP traffic features into our machine learning model will likely improve our classification accuracy and detect ransomware we would otherwise not catch. Finally, our next step is to run our own ransomware sandboxes in a controlled environment and collect ransomware traffic ourselves. Running our own malware in a controlled environment will allow us to collect more ransomware traffic; therefore, providing a more fine-grained feature set for our random forest classifier.

7 ACKNOWLEDGEMENTS

This work was supported in part by the NSF grants 1652698 (CAREER) and 1406192 (SaTC), and by the NSF and VMWare grant 1700527 (SDI-CSCS).

REFERENCES

[1] Jonathan C Beard, Peng Li, and Roger D Chamberlain. 2017. RaftLib: A C++ template library for high performance stream parallel processing. *The International Journal of High Performance Computing Applications* 31, 5 (2017), 391–404. https://doi.org/10.1177/1094342016672542
[2] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890
[3] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. ACM, New York, NY, USA, 99–110. https://doi.org/10.1145/2486001.2486011
[4] Krzysztof Cabaj, Marcin Gregorczyk, and Wojciech Mazurczyk. 2016. Software-Defined Networking-based Crypto Ransomware Detection Using HTTP Traffic Characteristics. *CoRR* abs/1611.08294 (2016). arXiv:1611.08294 http://arxiv.org/abs/1611.08294
[5] Krzysztof Cabaj and Wojciech Mazurczyk. 2016. Using Software-Defined Networking for Ransomware Mitigation: the Case of CryptoWall. *CoRR* abs/1608.06673 (2016). arXiv:1608.06673 http://arxiv.org/abs/1608.06673
[6] Europol. 2017. Internet Organised Crime Assessment 2016 IOCTA. (2017). https://www.europol.europa.eu/activities-services/main-reports/internet-organised-crime-threat-assessment-iocta-2017
[7] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection.. In *USENIX security symposium*, Vol. 5. 139–154.
[8] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. FlowRadar: A Better NetFlow for Data Centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 311–324. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/li-yuliang
[9] Arna Magnúsardóttir. 2017. Malware is Moving Heavily to HTTPS. (2017). https://www.cyren.com/blog/articles/over-one-third-of-malware-uses-https
[10] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 85–98.
[11] Yanjun Qi. 2012. Random forest for bioinformatics. In *Ensemble machine learning*. Springer, 307–323.
[12] Barkly Research. 2017. Cerber Ransomware: Everything You Need to Know. (2017). https://blog.barkly.com/cerber-ransomware-statistics-2017
[13] Daniele Sgandurra, Luis Muñoz-González, Rabih Mohsen, and Emil C. Lupu. 2016. Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection. *CoRR* abs/1609.03020 (2016). arXiv:1609.03020 http://arxiv.org/abs/1609.03020
[14] John Sonchack, Adam J. Aviv, Eric Keller, and Jonathon M. Smith. 2017. TurboFlow: Accelerating Flow Collection on Commodity Switches. (2017).
[15] Hilary Tuttle. 2016. Ransomware attacks pose growing threat. *Risk Management* 63, 4 (2016), 4.
[16] Ting-Fang Yen and Michael K Reiter. 2008. Traffic aggregation for malware detection. *Lecture Notes in Computer Science* 5137 (2008), 207–227.