SNIFFER: A High-Accuracy Malware Detector for Enterprise-Based Systems

Evan Chavis, Harrison Davis, Yijun Hou, Matthew Hicks,
Salessawi Ferede Yitbarek, Todd Austin, and Valeria Bertacco
Computer Science and Engineering, University of Michigan

***Abstract*—In the continual battle between malware attacks and antivirus technologies, both sides strive to deploy their techniques at always lower layers in the software system stack. The goal is to monitor and control the software executing in the levels above their own deployment, to detect attacks or to defeat defenses. Recent antivirus solutions have gone even below the software, by enlisting hardware support. However, so far, they have only mimicked classic software techniques by monitoring software clues of an attack. As a result, malware can easily defeat them by employing metamorphic manifestation patterns.**

With this work, we propose a hardware-monitoring solution, SNIFFER, which tracks malware manifestations in system-level behavior, rather than code patterns, and it thus cannot be circumvented unless malware renounces its very nature, that is, to attack. SNIFFER leverages in-hardware feature monitoring, and uses machine learning to assess whether a system shows signs of an attack. Experiments with a virtual SNIFFER implementation, which supports 13 features and tests against five common network-based malicious behaviors, show that SNIFFER detects malware nearly 100% of the time, unless the malware aggressively throttle its attack. Our experiments also highlight the need for machine-learning classifiers employing a range of diverse system features, as many of the tested malware require multiple, seemingly disconnected, features for accurate detection.

I. Introduction

Over the past few decades, malware creators and antivirus developers have been engaged in an arms race, each striving to deploy their solution at a lower layer in the software stack than the other side. The motivation behind this trend is that lower software layers have full control and visibility over upper layers. In the context of malware, this trend has led all the way down to bootkits [1], and a similar downward progression has taken place in antivirus technology.

To address the invasive nature of these attacks and protections, recent research has introduced the idea of enlisting hardware support in providing malware protection [2], [3], [4]. This idea is promising because it seems unlikely that malware creators will be able to leverage hardware support in their efforts. Unfortunately, initial proposals for incorporating hardware-level features in antivirus tools are mere extensions of their software-only counterparts, relying on code signature detection of low-level instruction analysis [2], [3], [4]. As such, many of these techniques share limitations with software antivirus in that they cannot detect sophisticated metamorphic malware, and by examining instructions they tend to be overly myopic, leading to frequent false positives.

To address these deficiencies, we turn to behavioral monitoring as a key mechanism to reliably detect next-generation malware. Specifically, we make the following contributions:

- We propose a hardware-assisted malware-detection approach based on a malware's external behavior, which is insensitive to the specific malware implementation.
- We show that our protections provide high detection accuracy (99.6%), and low false positive rates (<1% for idle systems), even when malware attempts to hide itself by curtailing its own malicious behaviors.

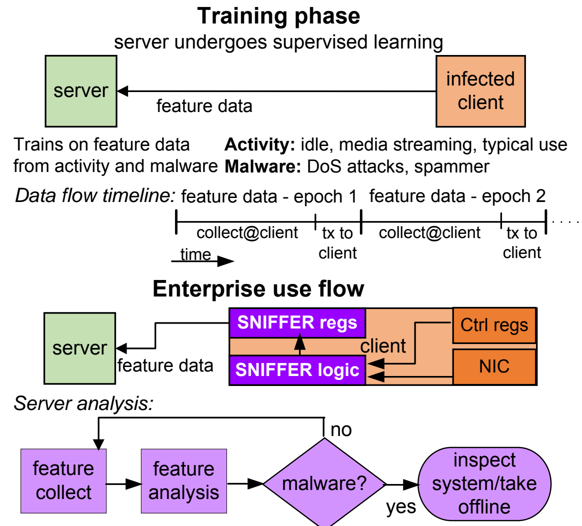

Figure 1. **SNIFFER overview.** During the training phase, SNIFFER leverages supervised machine learning (ML) algorithms to train its server-bound ML classifiers on clients under malware attack. At runtime, feature data is collected on a per-epoch basis, stored in dedicated SNIFFER secure storage at the client side, and transferred via a secure channel to the server at regular intervals for classification.

We combine these capabilities into a full system targeted at enterprise networks, where there is a network operations center including trusted servers. SNIFFER relies on continuous hardware-assisted measurements at each client, which include architecture-level (*e.g.*, superuser time) and system-level (*e.g.*, outgoing data packets) activity.

Before SNIFFER can be deployed, the monitoring servers must be trained using multiple supervised learning algorithms, up to one per malware class. During this phase the machine learning (ML) algorithms at the server end classify feature data gathered on a per-epoch basis from a client used for training. The client is operated under both no-malware conditions and while infected, for each of the malware classes to be detected.

Once the server is trained, SNIFFER can reliably detect malware in its training set. The lower portion of Figure 1 presents this execution flow. Client measurements are partitioned into equal-length epochs, during which metrics relevant for malware detection are collected into dedicated SNIFFER secure storage; the collected measurements are transferred to the server at the network operations center at regular intervals via a secure network channel. The trusted server analyzes the data based on the training dataset to determine if the incoming features show malware. In the event that the classifier identifies an infected host, it alerts a network engineer to respond.

We evaluated the effectiveness of the ML classifiers by collecting feature data on both clean and infected Windows virtual machines and then analyzed SNIFFER's detection capabilities on it. Additionally, we measured bandwidth and measurement overheads entailed by the analysis: we found that SNIFFER is

able to identify malware 99.6% of the time with a false-alarm rate of <1% for idle machines and <13% for heavily loaded machines. Moreover, we show that we are able to successfully detect metamorphic malware from varied implementations and we can always provide SNIFFER's accuracy, unless the malware throttles down its malicious activity aggressively.

II. Background, Motivation, and Related Work

Commercial malware detection. Static code analysis is fundamentally limited by the immense volume of malware codes; some commercial antivirus tools miss as much as 20% of known malware [5]. Two main reasons for these failures are that millions of previously unseen malware binaries are released daily [6] making it impractical to maintain an up-to-date signature database and that attackers employ code obfuscation and transformation tools to create numerous distinct versions of a single malware [7].

Behavioral malware detection. To achieve better detection results against malware with obfuscated code, researchers propose incorporating dynamic program information into detection through behavioral analysis. This new family of approaches provides a modest improvement in detection accuracy and shows success at preventing zero-day payloads. The key insight driving behavioral malware analysis is that there are only a handful of payloads (even if there are a myriad of ways to implement them) and by focusing on the attack behavior, detectors become robust against polymorphic and metamorphic malware. However, the key challenge for this approach is in reducing the cost of dynamic analysis, which must be run frequently. SNIFFER explores the idea of behavioral malware analysis further by pushing program monitoring completely down into hardware layers and adding more carefully selected features. Combined with moving classification of the malware behavior to a trusted computer, the SNIFFER's architecture presents security, accuracy, and performance advantages over prior work. **A key to SNIFFER's benefits is its focus on malice as opposed to anomalous behavior**, as in prior work [2], [3].

Hardware-assisted antivirus. Ozsoy, *et al.*, propose adding custom counters to processors in malware-aware processors [8] to overcome some of the shortcomings of using only existing performance counters, as was the case in previous proposals [2], [3]. malware-aware processors demonstrate that low-level features, such as program instruction mix and memory access traces are useful in identifying malware execution. While malware-aware processors show promise, several downsides of this approach are: 1) that the proposed systems still rely on software-level antivirus assistance, which has the same defect rate as normal software [9], [10], and it is vulnerable to attacks while reading the in-hardware counters and classifying the results, and 2) all software must halt during this classification period. Another work in this space [11] performs instruction signature detection using a trusted server for signature matching; to reduce network bandwidth requirements, they employ compressive sensing. While this work also performs analysis on a remote trusted server, our collected data represents high-level behavioral features, which provides significant insight into a program's intentions and reduced bandwidth requirements.

SNIFFER extends and improves the idea behind Malware-aware processors by removing all antivirus software components from the untrusted system—pushing the classification to trusted servers inside the network operations center (NOC). Besides protecting the antivirus from attack, pushing the burden of classification to servers in the NOC has the benefit of removing all run-time overhead from the monitored machines. Having a centralized management scheme also allows network engineers to select which malware they check for in the field. Finally, our solution is orthogonal to [11], in that, if SNIFFER were to greatly increase the number of features, it could employ their proposed compression to to keep in check transfer bandwidth requirements.

III. Threat Model

SNIFFER addresses the threat of malware running on networked computers. Specifically, our goal is to detect malware on computers connected to enterprise networks. Networks of this size often employ a centralized management structure referred to as a Network Operations Center (NOC). We assume that the system performing behavioral analysis within the NOC is trustworthy (including all hardware and software). We consider all software running on computers connected to the enterprise network but outside the NOC to be untrustworthy, while microprocessors (including the hardware-assisted monitoring) on these systems are trustworthy. Thus, we must assume other computers connected to the network are controlled by malware.

To summarize our threat model, we consider **trusted**: all hardware (*e.g.*, processors and network interfaces) and computers and software within the network operations center. We deem **untrusted**: all software on users' computers. This threat model presents two challenges for SNIFFER: (1) making trusted measurements on untrusted computers; and (2) making accurate and timely high-level decisions based on low-level measurements from users' computers.

IV. System Design

As illustrated in the lower portion of Figure 1, SNIFFER's approach to malware detection can be partitioned into two parts: **hardware-assisted feature acquisition** at the client computer and **behavioral classification** at a central trusted server. The system components responsible for feature acquisition and storage are trusted hardware-level components located on each of the monitored client systems within the enterprise network. The behavioral classification component is located on one of the Network Operations Center (NOC) trusted servers (see also Section III). At the end of each set of computational epochs, the feature data collected on the client's computer is transferred to the server, which in turn analyzes it and classifies the observed behavior as malware-infected or malware-free. By embedding the feature acquisition into a hardware-level component of the client machines, we ensure that the behavioral classifier on the server receives an untampered view of the client system's operation. By relocating the behavioral classification task away from the client systems, we avoid both the overhead of malware detection for these computers and also minimize the attack surface of our detection system. We detail each component of SNIFFER in the following sections.

A. Hardware-assisted feature acquisition and collection

Our system contains a protected hardware monitor that collects features that can be used individually or collectively by the behavioral classifier to detect malware. The features are representative of the manifested behavior of malware to prevent it from hiding by changing its implementation. For

instance, for a denial of service malware, there must be a burst of network packets of some type directed towards one or more victims. In this case, monitoring packet transmission rates and target addresses would be useful features to collect.

Below we provide the list of features currently collected. As we grow the range of attacks SNIFFER can protect from, we expect features to grow accordingly:

- **[superuser time]** - The time spent by the processor in user and superuser space is indicative of what privileges are needed by currently running programs. On our x86-based implementation, this measurements requires sampling the Code Segment (CS) register's Privilege Level (CPL).
- **[page faults]** - The number of page faults is collected because it provides information on the pattern of memory accesses. On x86, this measurement entails tracking the number of updates to the CPU's Control Register 2 (CR2).
- **[network packets]** - The counts of incoming and outgoing packets can detect departures from normal network activity. Our implementation assumes the use of a CPU-integrated Network Interface Controller (NIC), and the NIC provides packet capture and analysis capabilities to gather these statistics and those below.
- **[destination IPs]** - The number of distinct destination IP addresses for outgoing packets alerts the classifier of unusual concentration or dispersion of IP address destinations.
- **[source IPs]** - The number of distinct source IP addresses for incoming packets reveals potential divergences between outgoing destinations and incoming sources.
- **[ports]** - Number of ports used to send/receive packets – alerts the detector of suspicious spikes in either category.
- **[packet size]** - The average size of outgoing packets detects a device sending a large number of small or large packets, as in a synflood or an ICMP fragmentation flood.
- **[syn packets]** - The fraction of outgoing packets that are syn packets is indicative of a synflood if both the percentage and the number of outgoing packets are high.

There are several other considerations in our feature acquisition process. The key trade off lies between the thoroughness of features collected and the bandwidth required to transmit the collected measurements to the classification servers. Once hardware structures collect an epoch of features, data must be transferred to a trusted server for analysis. We assume that the system incorporates an integrated (and trusted) network interface controller (NIC) that is capable of sending feature data via a secure link (*e.g.*, via TLS) to a trusted server. Since next-generation NICs will likely incorporate TCP/IP implementations in hardware-protected components, it is reasonable to expect that this transmission can be made in a secure fashion, even from a compromised machine.

B. Behavior classification

The data collected during feature measurements is analyzed by a trusted analysis server in the NOC. The goal of this analysis is to detect systems infected with malware, while maintaining a low false alarm rate.

Using a single epoch's measured features, a machine-learning (ML) classifier is used to determine if the measured behavior is indicative of a machine infected by malware, or a malware-free system. Within our evaluation we considered K-nearest neighbors (KNN), Bayesian, SVM, and neural network classifiers. We found that, for the malware types and features we considered, KNN and Bayesian network classifiers provide

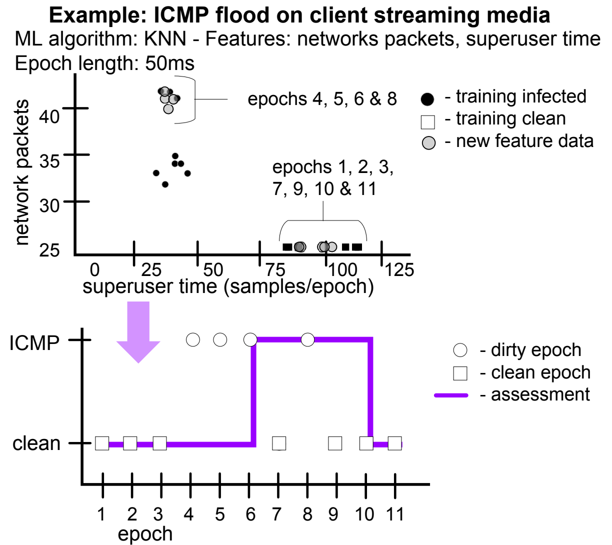

Figure 2. **Example of server classification for ICMP flood.** An ICMP flood is identified by collecting feature data on the number of network packets and time spent in superuser mode. After the initial assessment by the machine learning algorithm, KNN, SNIFFER applies a weighted moving average filter (EWMA) to stabilize the determination and reduce the incidence of false positives in flagging malware.

the best outcomes. The trained ML classifiers, one for each major class of malware, will provide an initial assessment of the client state (infected or clean) for each individual epoch. Note how, during training, SNIFFER selects the set of features and the ML classifier that works best for the specific class of malware. Then, during classification, the previously selected classifier is used to evaluate data from a run-time epoch with respect to a specific class of potential malware infection. Note also that, in general, the administrators of an enterprise would want to validate the client computers against a range of malware. Thus, SNIFFER will run multiple classifications, one for each class of malware under investigation, using a distinct classifier and feature set.

After the classifier provides an initial assessment, SNIFFER applies an exponentially weighted moving average (EWMA) filter to the outcome (clean/infected) of the current and past epochs. This filter stabilizes the assessment. It is not infrequent that a client manifests short bursts of activity resembling the activity of a malware; however, unless the activity is sustained SNIFFER will overlook it to minimize false positives. The final assessment provided to the security administrators for each execution epoch of the client is the output of the EWMA filter, represented by the thick purple line in Figure 2.

Since the actual incidence of malware is typically low, one of the key challenges of malware detection is to keep the false alarm rate low. The EWMA filter limits the false alarm rate by leveraging the fact that an epoch is less likely to be tagged malware if the previous epoch was tagged clean. The downside of using this filter is that, if the exponential average is weighted too heavily towards previous predictions, lightweight malware could exploit the weighting to avoid detection while still eventually accomplishing its goals. Balancing these two considerations requires careful tuning.

C. Training the malware detection classifiers

Our classifiers must be trained in advance of use for each class of malware that they must detect. For our experimental malware detector, we trained a variety of classifiers on both

baseline (non-malicious programs) and malware. The training then used both KNN and Gaussian Naive Bayesian candidate classifiers. KNN requires setting parameters, which were selected through cross validation. Finally, all the training data is fed to the classifier so that it learns a model, and the classifier configuration that performs best on the cross-validation data set is selected as the classifier for that particular class of malware. We envision that in a production environment the training of classifiers would be either performed in the NOC or provided by an anti-malware software vendor. The training provider could easily configure the training to detect any subset of malware needed by the user.

V. Experimental Setup

Rather than create a physical implementation of our SNIFFER system for experiments, we created an experimental setup, based on virtual machine execution, that allows us to evaluate sniffer's performance without needing to physically build the system itself. The experimental setup faithfully represents the full function of the proposed system, and it collects results representing how well it performs at detecting malware.

A. Malware testbeds

In our experimental evaluation, we employed several malware types described below. For each attack, we used a publically available implementation of the attack but in all cases we extended the attack implementation to provide tunable duty cycle and peak rate capabilities. The duty cycle specifies what fraction of time the malware is active during a fixed period of one second. For instance, if the user specifies a duty cyle of 10%, the malware is active for 100ms and then goes dormant for 900ms before becoming active again. This feature strengthens the stealth capabilities of malware, making it harder to detect. At the limit, a user can specify a duty cycle of 100% so that the malware is always active. The peak duty cycle is a user-specified metric indicating the intensity of the attack when the malware is active. The specific units depend on the type of attack the malware is performing.

- **[UDP Flood]** - A UDP flood is a denial of service (DoS) attack that makes use of the UDP protocol. It sends UDP packets to random TCP/IP ports at the target, forcing it to reply with destination-unreachable ICMP packets, thus consuming the target's resources. This malware is derived from the PyNuker UDP flood project [12], which we adapted with control for duty cycle and peak rates.
- **[SYN Flood]** - A SYN flood is similar to a UDP flood, but uses SYN (synchronization) packets to perpetrate a denial-of-service attack. Our SYN flood has the same duty cycle and peak rate control as UDP flood. We developed it as a wrapper around Nmap [13], a network scanning tool used to send packets to a target at a user-set constant rate. For duty cycles below 100%, we built the malware on Scapy [14], a Python packet manipulation program.
- **[ICMP Flood]** - ICMP floods send a large number of ping packets to a victim in order to draw ping responses. Our ICMP flood provides duty cycle and peak rate control, and it is developed in Python using Scapy [14].
- **[SMTP Spammer]** - Simple Mail Transfer Protocol (SMTP) is the standard protocol for email. Our SMTP spammer is a Python program wrapping a Windows command line utility for sending email (based on SwithMail). It has two modes of operation: sending one email per second or as many emails as possible.
- **[Slowloris]** - The Slowloris attack opens several HTTP connections to a web server and keeps them open to prevent the target from responding to other traffic. We use the Slowloris implementation that is distributed with Nmap [13].

These malicious programs are selected because they represent a variety of current and historical attacks that a compromised computer within a botnet might encounter. In our evaluation, we strive to detect these malware programs under several stealth conditions: we limit attack time by controlling the duty cycle to avoid detection, and we attack while a variety of non-malicious applications execute concurrently on the system.

To avoid compromising third party systems and malware propagation, our malware programs omit a propagation component and only execute the payload of the malicious application; it should also be noted that only IP addresses within our lab's network were attacked.

B. Feature collection

We collect all features listed in Section IV-A throughout each epoch. In our setup, epoch lengths are 50ms to strike a balance between the noisy characteristics of short epochs, where even small spikes of activity may be interpreted as malware, and the low-pass filter effects of long epochs, which average away the behavior of duty-cycle controlled malware. Specifically, we simulate a physical SNIFFER implementation by reading a virtual machine's registers via LibVMI [15], a tool to monitor low-level characteristics of a virtual machine; this includes the number of page faults and the time spent in superuser mode. The remaining features are related to the packets entering or leaving the virtual machine. We capture all packets going into or out of the virtual machine using `tcpdump` [16] on the host machine and then parse the raw data to gather the desired features.

C. Classifier setup

Once we collect all the feature data from the virtual machine, we run the selected classifier and features that SNIFFER found most suitable for the given malware during training (see Section IV-C). The ML algorithm processes several epochs at a time (called a **set**), approximately 200 in our setup, and then applies the weighted moving average filter to attain the final assessment for each epoch. If any of the epochs in the set is deemed as malware, the client is flagged as infected and the set is classified as malware-flagged, otherwise the set is clean.

VI. Experimental Evaluation

This section evaluates SNIFFER in terms of the quality of its analysis and its performance impact on the enterprise components. The metrics that we use to assess SNIFFER's malware detection capabilities are based on the relation between SNIFFER's classification and the client's situation. Specifically, we summarize these measurements with two key metrics based on counting one event per set of epochs (see Section V-C):

- **[False Alarm Rate (FAR)]** - This metric represents how often SNIFFER will flag malware on a client that is malware-free, thus forcing an unnecessary secondary inspection or downtime on the client, causing loss of trust in the system. We compute FAR as: **FalsePositives / (FalsePositives + TrueNegatives)*100** over each 50ms epoch.
- **[Missed Alarm Rate (MAR)]** - This metric represents how often the system fails to detect malware when a client is infected. MAR is derived as **FalseNegatives / (FalseNegatives + TruePositives)*100** over each 50ms epoch.

machine activity / malware detector	Targeted Malware on Idle System	Targeted Malware with Media Streaming	No Malware Typical Usage
UDP Flood	FAR=MAR=0%	FAR=MAR=0%	FAR=0%
SYN Flood	FAR=3% any rate MAR=80%@1 pkt/s, 0%@ >100 pkt/s	FAR=0 MAR=7%@10 pkt/s, 0%@ >15 pkt/s	FAR=0%
ICMP Flood	FAR=MAR=0%	FAR=13.3% any rate MAR=0%	FAR=1% any rate
SMTP Spammer	FAR=MAR=0%	FAR=MAR=0%	FAR=0%
Slowloris	FAR=MAR=0%	FAR=MAR=0%	FAR=0%

Table I. **False Alarm Rate (FAR) and Missed Alarm Rate (MAR)** reported for each malware testbed (across all 50ms epochs) for three distinct types of client executions: idle, streaming media applications, and typical workday use. In most cases, SNIFFER provides a FAR and MAR of 0.0%.

During the training phase, the classifier and features to be used for each class of malware are selected. Each class aside from Slowloris requires a K-nearest neighbors classifier, while Slowloris uses a Bayesian classifier. In terms of features, SYN Flood uses SYN packets, user time and outgoing network packets, Slowloris uses page faults and user time, all other malware uses all the features that SNIFFER measures.

A. Malware detection

SNIFFER's accuracy in detecting malware is presented in Table I, which reports FAR and MAR metrics for each of the malware testbeds and for multiple operating conditions. The "Media Streaming" column includes malware detection concurrent with the website YouTube continuously streaming video. We varied the rate of flooding for all flood malware from 1 packet/second to 1,000, while keeping the duty cycle at 100%. For the SMTP spammer we used both the 1 email/second and max rate settings. Slowloris was operating at the default parameter setup from Nmap. Note that both FAR and MAR are at 0% (indeed, 0.0%) for most combinations. SYN Flood has noticeable MAR at extremely low attack rates, but they both quickly approach 0% at rates above 100 packets/second when the client is idle, and at rates above 15 packets/second when the client is simultaneously streaming media. Finally, ICMP Flood and SYN Flood present a small FAR, one under idle and typical use conditions, and the other while streaming.

To assess the extent that non-malicious activities triggered false positive malware occurrences (*e.g.*, a non-zero FAR), we performed analysis of a machine running one hour of typical usage for three users. We did not report MAR for typical use workloads because no malware was activated, thus, we only tracked the FAR metric. As shown in Table I there were no false positives encountered except for occasional ICMP Flood alerts. These alerts were quite infrequent, and they did not form a significant contribution to the overall alarm rate.

Next, we evaluated how SNIFFER's detection performs with malware attempting to evade detection by throttling down their attacks. In duty cycle mode (see Section V-A), a malware is active for short intervals of time, followed by dormant periods. This is one of the most effective techniques against a solution like SNIFFER, which detects the behavioral manifestations of malware, since the malware detector may deem the short duty cycle of an attack as non-malicious activity.

Figure 3 reports SNIFFER's ability to detect malware as the duty cycle varies. In our experiments, we varied the duty cycle from 10% (100ms) to 30% (300ms). We analyzed SNIFFER's detection of all our flood malware while the client was operating on idle. Users can choose to tune their epoch length parameter to the minimum duty cycle they wish to be able to detect. Given these results, it is clear that SNIFFER would be

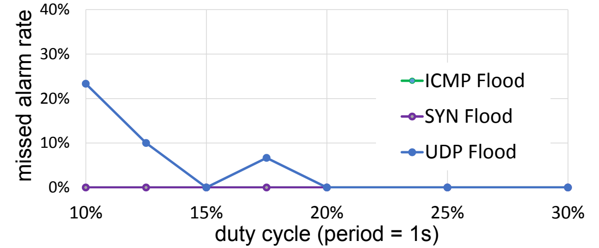

Figure 3. **Missed Alarms Rate for malware on idle clients.** The plot reports the MAR for UDP Flood, SYN Flood and ICMP Flood while varying the duty cycle from 10% to 30% when attacking from an idle client.

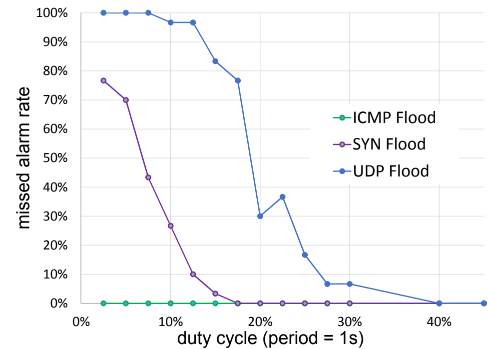

Figure 4. **Missed Alarms Rate for malware on media streaming clients.** The plot reports the MAR for UDP Flood, SYN Flood and ICMP Flood while varying the duty cycle from 0% to 45% when attacking from a client which is streaming media.

a powerful tool to identify malware, or at least force malware to limit its own effectiveness. Figure 4 plots the same critical metric, MAR, again for ICMP Flood, SYN Flood and UDP Flood. However, this time the infected client's activity is performed concurrently while the infected machine is streaming media, which generates network activity in the background. SNIFFER's detection capabilities are extremely accurate even at very low attack rates (as low as 1 packet/second) and with high incidents of non-malicious activity.

Finally, we also evaluated SNIFFER sensitivity to the malware implementation. As discussed earlier, SNIFFER attempts to be insensitive to evasion techniques that are based on code transformation. As an initial test for this, we compiled the UDP Flood (written in Python) with baseline settings and with modified compiler optimization settings, so as to generate varied implementations. Figure 5 plots the Missed Alarm Rate measured when SNIFFER detects these two versions of UDP Flood on a client streaming media. SNIFFER's MAR trend is independent of the optimization level of the malware code, and only sensitive to the malware's duty cycle. Although code optimizations fall short of the transformations possible with metamorphic malware, this result is promising.

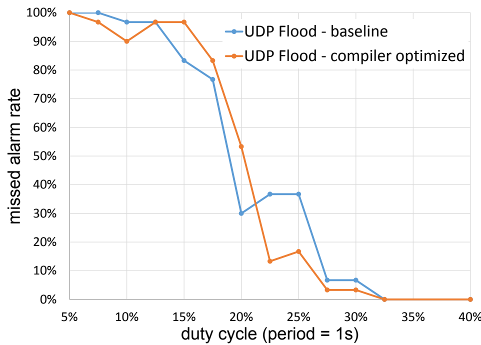

Figure 5. **Missed Alarm Rate for two different implementations of UDP Flood.** The plot reports the MAR over varying duty cycles for two versions of UDP Flood: a baseline one and one with optimized compilation.

Storage per epoch	bits
superuser time	8
user time	8
page faults	6
IN packets	8
OUT packets	8
destination IPs	4
source IPs	4
OUT packet src ports	8
OUT packet dest ports	8
IN packet src ports	8
OUT packet dest ports	8
packet size	5
syn packets	7
TOTAL storage/epoch	90

Transfer	bits
data per epoch	90
data per transfer	90*20
security hash	256
nonce	32
TOTAL TRANSFER	2,088

Table II. **Storage and transfer requirements for SNIFFER.** Storage is required for all features gathered during each epoch, and we store 20 epochs of data at a time. Each transfer must transmit all feature data augmented with a security data for a total of 2,088 bits/second.

B. Key feature analysis

Earlier, we reported which features were used for each malware type to obtain the results in Table I. For all but two malware classes, we are unable to achieve comparable results with only one feature. This result shows that it is not always possible to assess if a client is under attack with simple intuitive measurements of just one feature, but it is important to deploy the fine-grained capabilities of machine learning classifiers that can be trained with a wide range of features.

C. Performance and area overhead analysis

To evaluate SNIFFER's overhead, we first considered the amount of storage necessary at the client end to gather all the measurements for an epoch. The left part of Table II reports the number of bits of storage per epoch for each of the features we gathered. In addition, we transfer feature data to the server in batches of 20 epochs, that is, once a second. Thus we need to store measurements for 20 epochs, for a total of 1,800 bits.

The right part of the table computes how many bits are transferred during each transmission: 1,800 bits are required for 20 epochs of measurements. Moreover, we need to include a hash to protect the data from network-bound attacks. Thus, each time feature data is transferred to the server, the requirement is to transfer 2,088 bits, equivalent to 261 bytes. The network bandwidth required to support SNIFFER-related communication is thus 2,088 bits/second. These settings can be adjusted by a user based on their needs.

In a typical enterprise there are many client computers that must be continuously monitored for potential malware infections: in Figure 6 we report how the bandwidth demands on

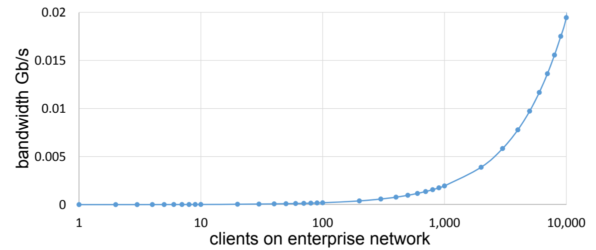

Figure 6. **Network bandwidth demands for feature transmission** as the number of clients in the enterprise network increases.

the network increases as the number of clients increases. Note that the demand is limited up to 10,000 clients, and becomes significant for larger enterprises. In those environments it is most practical to partition the network into multiple distinct domains. We expect an enterprise-class network to easily support 100,000's of machines with no excessive burden on network resources, with only a few malware analysis servers.

VII. Conclusion

In this paper, we introduced an enterprise-class antivirus analysis framework, called SNIFFER, which measures in hardware at run-time system-level features for transmission to trusted classification servers. System features are analyzed, using pre-trained machine-learning models, over an epoch of execution to determine if a machine is infected with malware. Experiments show that SNIFFER is effective at detecting malware with low false positive rates, even as attackers lower their attack rates to evade detection. We also examined the bandwidth requirements of the SNIFFER system and found that a single analysis server could easily service over 10,000 machines without undue network demands. These results show the accuracy and security benefits of combining in-hardware monitoring with behavioral-focused machine learning and suggest that the SNIFFER architecture is a promising approach to address increasingly sophisticated malware codes.

Acknowledgements. This work was supported in part by C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] P. Kleissner, "Stoned bootkit," BlackHat, 2009.
[2] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," in *Proc. ISCA*, 2013.
[3] A. Tang, S. Sethumadhavan, and S. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *Proc. RAID*, 2014.
[4] K. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in *Proc. RAID*, 2015.
[5] "Latest RAP quadrant," Virus bulletin, 2015.
[6] AV-TEST, "Malware statistics," https://www.av-test.org/en/statistics.
[7] P. Szor, *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, 2005.
[8] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *Proc. HPCA*, 2009.
[9] T. Ormandy, "Symantec/Norton Antivirus ASPack remote heap/pool memory corruption vulnerability," CVE-2016-2208, 2016.
[10] Sophos, "Tavis Ormandy finds vulnerabilities in Sophos," 2015.
[11] X. Wang, S. Chai, M. Isnardi, S. Lim, and R. Karri, "Hardware performance counter-based malware identification and detection with adaptive compressive sensing," *ACM Trans. on Code Opt.*, vol. 13, no. 1, 2016.
[12] "Pynuker," https://sourceforge.net/projects/pynuker/.
[13] "Nmap website," https://nmap.org/.
[14] "Scapy website," http://www.secdev.org/projects/scapy/.
[15] B. Payne, "Simplifying virtual machine introspection using LibVMI," Sandia National Laboratories, Tech. Rep. SAND2012-7818, Sep. 2012.
[16] "TCPDump website," http://www.tcpdump.org/.