Can Data-Only Exploits be Detected at Runtime Using Hardware Events?

A Case Study of the *Heartbleed* Vulnerability

Gildo Torres
Clarkson University
8 Clarkson Ave
Potsdam, New York
torresg@clarkson.edu

Chen Liu
Clarkson University
8 Clarkson Ave
Potsdam, New York
cliu@clarkson.edu

ABSTRACT

In this study, we investigate the feasibility of using an anomaly-based detection scheme that utilizes information collected from hardware performance counters at runtime to detect *data-oriented* attacks in user space libraries. Using the Heartbleed vulnerability as a test case, we studied twelve different hardware events and used a Support Vector Machine (SVM) model to classify between regular and abnormal behaviors. Our results demonstrated a detection accuracy over 92% for the two-class SVM model and over 70% for the one-class SVM model. We also studied the limitations of using certain type of hardware events and discussed possible implications of their use in detection schemes. Overall, the experiments conducted suggest that *data-oriented* attacks can be more difficult to detect than *control-data* exploits, as certain events are susceptible to interference hence less reliable.

1. INTRODUCTION

Malicious software has existed for many years and continues to proliferate. There are many classes of malware, e.g., rootkits, viruses, trojans, worms, spywares, exploits, etc., each with its own goal and behavior. In an effort to address the malware problem, researchers have devised numerous prevention and detection schemes, each focused on a certain aspect of the malware life-cycle, from creation, distribution, to execution. While it is infeasible to prevent bad actors from authoring malware, schemes that target the distribution and execution of malware do exist. For example, modern App marketplaces provide an effective "whitelisting" service; digitally signed software can prevent the execution of software from unknown sources. As an additional example, vulnerability analysis tools can be used to identify and even fix vulnerabilities in software, thereby preventing the distribution of malware. Techniques such as no-execution bit (NX), address space layout randomization (ASLR) [3], and control flow integrity (CFI) [2, 19] are used to prevent the execution of malicious logic. Despite the valiant efforts, prevention techniques are not perfect as evidenced by the existence of third party marketplaces and continued presence of vulnerabilities. Detection techniques are used as an extra line of defense when prevention fails.

Here we focus on the detection of anomalous execution due to malware. Malware can be separated into three categories: self-contained malware (malware with its own code such as rootkits), parasitic malware (malware that executes new code in the context of a host process such as viruses and worms),and data-only malware (malware that can achieve its goals without introducing any new code). Previous research has focused on the first two categories where behavioral models were used to detect the presence of suspicious code such as rootkits [16] and the execution of newly introduced code through injection or control flow hijacking known as *control-data attacks* [13, 7, 6].

In this work we focus on data-only malwares that do not divert the application's control-flow and where no additional code is introduced into the system during the attack. *Data exploits*, also known as *data-oriented attacks* or *non-control-data attacks*, are a type of data-only malware that seeks to cause a piece of software to violate the confidentiality and/or integrity of the data and the rest of the system by solely manipulating the inputs [9]. These types of attacks usually result in the disclosure of sensitive information/data to the attackers or an escalation of privileges, allowing the attacker to achieve a similar level of compromise of the victim system as that achieved by common *control-data attacks* [9, 6]. SQL injections are famous examples of this type of malware and the Heartbleed vulnerability is a recent example where sensitive data is disclosed inadvertently.

To formulate the study, we assume that the malware executes in the context of a user-space program and there exists a privileged kernel module that controls the performance counting hardware used for detection, similar to that of Tang et al. [13]. Hardware-based prevention schemes such as the NX bit are preferred since they have better performance characteristics and are more difficult to bypass. However, important contextual information such as process abstractions might be lost, meaning a software layer must still exist to provide the context. The same observations hold for hardware-based detection.

We focus on one specific type of vulnerability, buffer over-reads, to study the feasibility of performing anomaly detection of data-only malware using hardware performance counters. There are two reasons for this choice. First, buffer over-reads are very similar in nature to buffer overruns, which were studied in [13]. Second, we can use the *Heartbleed Bug* (CVE-2014-0160) as a real world case study.

The main **contributions** of our work can be summarized as follows:

– We believe this work to be the first one to use low-level hard-

HASP 2016, June 18 2016, ,

© 2016 ACM. ISBN 978-1-4503-4769-3/16/06. . . $15.00

DOI: http://dx.doi.org/10.1145/2948618.2948620

ware events for the detection of *data-oriented* attacks.
– We conduct an empirical study on the statistical behavior of hardware events between malicious and regular behavior.
– We study the feasibility of using low-level hardware events to detect *data-oriented* attacks.
– We present an in-depth qualitative analysis of the hardware events behavior and demonstrate that certain events are unstable and therefore unreliable.
– We assess the proposed methodology in terms of its advantages and limitations with the study of a recent real-world vulnerability.

2. BACKGROUND

2.1 Hardware Performance Counters

Hardware performance counters (HPCs) are special hardware registers available on most modern processors. These registers can be used to determine the number of occurrences of certain types of hardware events such as: instructions retired, cache-misses suffered, and branches miss-predicted. These events are reckoned without slowing down software execution because they use dedicated hardware.

Although originally created for debugging hardware designs during development, performance tuning, or identifying bottlenecks in program execution, HPCs can also be used to collect runtime behavioral information of programs. The type and number of available events and the methodology for using these performance counters vary widely, not only across architectures, but also across processor families sharing the same Instruction Set Architecture (ISA). For example, Intel's modern processors offer more than two hundred different events that can be tracked [10]; however, only a selected few events can be monitored concurrently due to limitations in the number of available HPC registers.

2.2 Anomaly Detection

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior [4]. Much work is therefore dedicated to the modeling of expected behavior. In the context of malware detection, the advantage is it can be deployed to detect both known and unknown attacks. It can also be designed to scale with programs by carefully selecting the events to be monitored and profiled. On the downside, anomaly detection can lead to false positives or false negatives. The detection model can only approximate the behavior of the actual program, therefore errors tend to occur.

2.3 OpenSSL

OpenSSL is an open source implementation of the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS) protocols as well as a general-purpose cryptography library [14]. There exist versions of the OpenSSL library for nearly every major platform including Windows, Linux, and Mac OS. The most notable software using OpenSSL are open source web servers like Apache and nginx. The combined market share of just those two is over 66% according to Netcraft's April 2014 Web Server Survey [11]. Numerous vulnerabilities have been discovered, reported, and fixed within the OpenSSL library, as with any software project of its magnitude, one of the most serious being *Heartbleed*.

2.4 *Heartbleed* Vulnerability

Heartbleed is a serious vulnerability discovered in the OpenSSL library which was publicly disclosed in April 2014 [1]. The bug appears in the implementation of the Heartbeat Extension for the TLS and DTLS protocols. The protocol provides a way to test and keep alive secure communication links without the need to re-negotiate the connections. Under normal circumstances, a requesting party can send a Heartbeat request that contains a token T of length T_{size} and a request size of R_{size}. The server is expected to echo back the R_{size} bytes of the token. The main issue consists of a missing check between an advertised request R_{size} and the real token size T_{size}. This allows the requesting party to trick the target into sending more information (memory content) than it should. Due to the widespread deployment of the OpenSSL library, the existence of cryptographic keys in memory and the ease of exploitation, this vulnerability had a significant impact on secure web servers when it was disclosed. The vulnerability itself has been well studied and there has been a number of proof-of-concept exploit malware developed ever since, making it a great case study candidate.

3. SYSTEM ARCHITECTURE/MODEL

A high level representation of our proposed system architecture is shown in Figure 1. The anomaly detection module resides between the targeted applications and the hardware performance counters. Its presence is transparent to the targeted process and shared library.

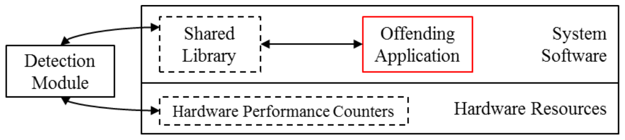

Figure 1: Simplified system architecture.

3.1 Heartbeat *Good* vs *Bad* Requests

The *Heartbleed* vulnerability consists of a mismatch between the real size of a message's token (T_{size}) and the size of the payload that is advertised/requested by the attacker (R_{size}). If the request is not malicious, then the size of the requested payload is equal to the real size of the token ($R_{size} == T_{size}$) and there is no extra data included in the reply, which we refer to as *good* requests. For the malicious cases, the requested size is larger than the real size of the token ($R_{size} > T_{size}$), where the difference between them represents the total amount of data leaked by the vulnerable system, which we refer to as *bad* requests. In a single request, the maximum size of the token being sent is 16KB and the maximum requested payload allowable is 64KB.

3.2 Detection Module

For the detection module, we employed four different classification methods:

- **Single-event/single-threshold:** Samples are individually classified as *good* or *bad* based on a simple threshold corresponding to each hardware event.
- **Multiple-events/multiple-thresholds:** First, samples are classified as *good* or *bad* based on individual event thresholds, and then a final classification is made based on the total number of events suggesting the same decision.
- **Multi-class SVM[1]:** This method trains a multi-class Support Vector Machine (SVM) model with a *training set* containing both *good* and *bad* requests and then conducts classification on a *test set* containing both *good* and *bad* samples.
- **One-class SVM:** Using a one-class Support Vector Machine model, the *training set* exclusively contains *good* Heartbeat

[1]We used libsvm library [5] for the SVM classification model and prediction.

Table 1: Recorded Hardware Events.

Event Name	Description
RET	Near return instructions retired
MISP_BR	Mispredicted branch instructions
LOAD	Load instructions retired
MISP_BR_C	Mispredicted conditional branches
STORE	Store instructions retired
MISS_ITLB	I-TLB misses
STLB_HIT	Shared TLB hits after i-TLB misses
MISS_DTLB	DTLB-misses
CALL_ID	Indirect near call instructions retired
MISS_ICACHE	I-Cache misses
CALL_D	Direct near call instructions retired
MISS_LLC	Last Level Cache misses

samples. The classification is conducted on the *test set* containing both *good* and *bad* samples.

For the work presented in this paper, the detection module was implemented at the same privilege level as the OpenSSL library (user-space). This helps reducing variability and isolating potential issues. Moving our implementation to kernel space should be straightforward although confirmation is left as future work.

4. METHODOLOGY

This section includes a description of the methodology followed in this work.

4.1 Configuration

Platform: The experiments were conducted on a machine powered by an Intel Core i7 950 (Nehalem, QuadCore, HT, 3.06GHz) processor with 8GB of memory running Ubuntu 13.04 with Linux kernel version 3.8.0. The Core i7 processor used includes 4 built-in configurable hardware counters. The vulnerable environment had OpenSSL version 1.0.1f installed, which was the last version containing the *Heartbleed* vulnerability. The *perf_events* syscall in Linux was used to configure and access the hardware counters.

Hardware Events: Given the extensive nature of distinct hardware conditions that could be recorded using HPCs [10] and the huge number of combinations that could be derived, making an exhaustive study is infeasible. Instead, we selected an initial set of 12 hardware events reported in recent literature [7, 8, 13, 15, 16, 17, 18]. Table 1 shows the hardware events considered in this study.

Sample collection: We modified the OpenSSL library to include calls to *perf_events* to configure, start, and stop the hardware performance counters at runtime. Counting was only enabled during user-space execution, excluding kernel events. A sample of the hardware event readings was collected every time the Heartbeat function was invoked. The samples were saved to a log file for later processing and classification. By following this approach, the performance impact on the normal execution of the targeted process and shared library was minimized.

Exploit: The *Heartbleed* attacks we generated use a modified version of the exploit developed by Stafford [12]. For this study, it was assumed that the host was not compromised at any level and that there was no interference with the configuration and functioning of the HPCs.

5. EVALUATION

This section includes details of the systematic experimental studies conducted. We keep the following questions in mind when designing our experiments:

– How clearly can different high-level actions, specifically buffer over-reads, be recognized using hardware events?

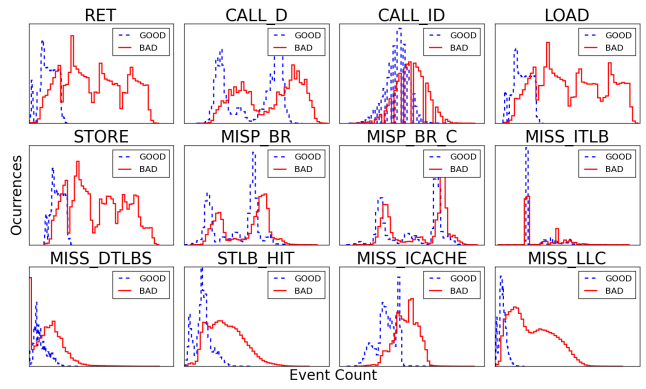

Figure 2: Distribution of occurrences for event counts of *good* and *bad* heartbeat requests. The x axis represents each individual hardware event count, while the y axis shows the number of occurrences for every count. Both axes are normalized to their individual event range.

– Which hardware events are altered the most by such changes in software behavior?
– Are the hardware events reliable?
– How precisely could these actions be detected by only monitoring a limited number of these hardware events?

5.1 Test Cases

The set of experiments we ran are as follows:

– Total of 1 million tests divided into 1000 test groups (1000×1000).
– Each test group containing 1000 Heartbeat requests.
– Different *good*/*bad* Heartbeat request distribution per group[2].
– Attack intensity distribution within groups varied from [1000 *good* requests /0 *bad* request] to [1 *good* request /999 *bad* requests].

We used the server application included within the OpenSSL toolkit as the vulnerable target and recorded the hardware events for every Heartbeat request made to the server. Because this study targeted a total of 12 different hardware events, and the experimental platform only has four hardware performance counter registers to count simultaneously, we repeated the experiments three times collecting different subsets of four events at every run.

5.2 Overhead

In a separate study, we measured the overhead added to the target function by the Linux's *perf_events* syscall when accessing the performance counters. On average, the performance counter event collection scheme added a 7% overhead to the Heartbeat function. Taking into consideration the fact that the time it takes to access the hardware counters should be uniform and independent of the process(s) being monitored, and the fact that the Heartbeat extension itself is very simple, such delay represents a reasonable overhead. Furthermore, what we considered is the worst case scenario, given the fact that a simpler and more direct manipulation of the performance counter registers could be implemented at the assembly level using the RDMSR and WRMSR instructions directly.

5.3 Hardware Events' Statistical Distribution

The conducted experiments capture the behavior of different hardware events when subjected to different *good* and *bad* requests.

[2]For all heartbeat requests the token size (T_{size}) and the requested payload's size (R_{size}) were randomly generated within the ranges previously described in Section 3.1.

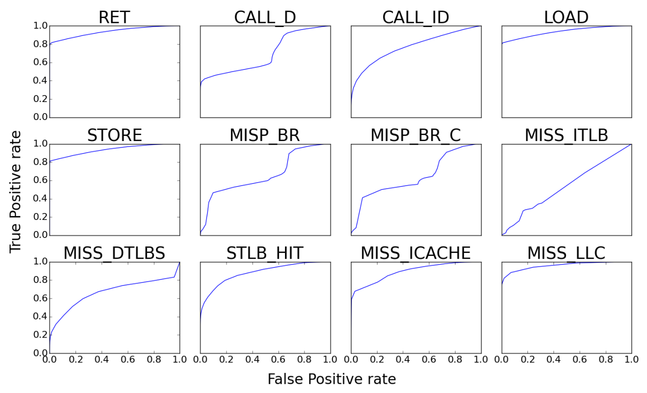

Figure 3: Receiver Operating Characteristic (ROC) classification performance of each individual event.

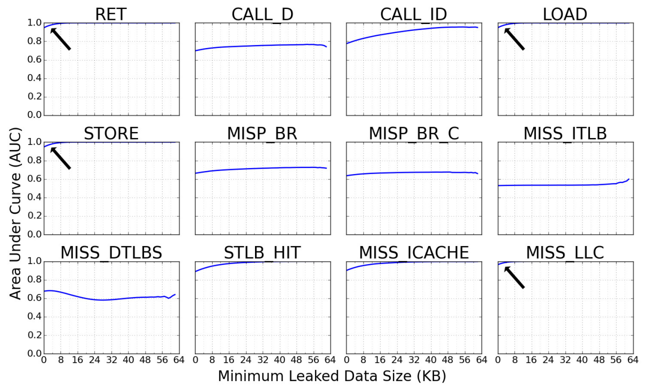

Figure 4: Area Under the Curve (AUC) of individual hardware events for leaked *gaps* between 1KB and 64KB.

Figure 2 shows the individual distribution of every event for the 1 million Heartbeat requests recorded. Each plot illustrates the contrast between the event count distributions of *good* and *bad* requests.

As we can see, all hardware events experience a certain level of overlap between *good* and *bad* requests. The degree of overlap, however, varies significantly among different events. For some cases, the overlap is almost indistinguishable, e.g., MISS_ITLB, CALL_D, and MISP_BR_C; while for others, e.g., MISS_LLC, RET, LOAD, and STORE, clear differences can be observed. These plots offer an initial insight about which events might perform better when detecting abnormal behavior for this particular *Heartbleed* test-case.

5.4 Detection Accuracy

Here we present the performance of classifier based on the individual event method and the SVM method.

5.4.1 Individual events

The Receiver Operating Characteristic (ROC) is a graphical plot used to illustrate the performance of a binary classifier system. The ROC plot for the normal vs. abnormal classification of each individual hardware event is shown in Figure 3. This curve represents the ratio between the *bad* cases classified as *bad* (True Positive) to the *good* cases misclassified as *bad* (False Positive) for different classification thresholds. Curves appearing above the diagonal represent positive detection ratios. A common measure to the specific performance of an individual ROC plot is represented by its Area Under the Curve (AUC) value.

Table 2: Detection performance of different classifiers.

Classifier	**Classification Accuracy (%) of different sets**						
	All	1KB	2KB	4KB	8KB	16KB	32KB
2-class SVM	92.80	94.02	95.38	97.01	98.75	99.98	100
1-class SVM	70.88	73.04	73.70	74.68	74.55	74.46	74.41

As expected from the event distribution plots shown in Figure 2, the events that appeared as hardly discernible show a poor performance in the ROC plots. On the other hand, the events that exhibited better-defined differences performed significantly better in the ROC plots.

We also study the sensitivity of the single-threshold classification method when *good* and *bad* requests are separated by a minimum leak data size *gap*[3] ranging between 1KB and 64KB.

As we mentioned in Section 3.1, a single malicious Heartbeat request can leak up to 64KB. Technically, a request that leaks only 1 byte of data is still considered as a malicious request. However, since the attacker only controls how much data to leak (up to 64KB), but not what specific data from the memory or if the data is OpenSSL related, it is arguable how much data must be disclosed to the attacker before it is of some use.

Figure 4 shows the Area Under the Curve (AUC, an ROC performance metric) for each individual event when we vary the *gap* size from 1KB up to 64KB. For each *gap* size represented along the *x* axis, we generated the ROC plots and obtained their corresponding AUC values, which are represented along the *y* axis.

The AUC graphs shows higher detection accuracy as the *gap* size grows for certain hardware events. At 4KB, the AUC score for the RET, LOAD, STORE, and MISS_LLC hardware events is already larger than 0.98. It is also noticeable how the classification accuracy of some events (e.g., MISS_ITLB, MISS_DTLBS, MISP_BR, and MISP_BR_C) appears to be immune to different *gap* sizes.

Overall, the single-event classification method achieved acceptable performance despite being a simple method. The performance of the multiple-events/multiple-thresholds classification method is very similar to the single-event/single-threshold, therefore its performance results are omitted here.

5.4.2 SVM Classification Performance

As described in Section 3, we crafted two implementations of the SVM algorithm, i.e., one-class SVM and two-class SVM models. Both implementations used a radial basis function (RBF) kernel [5].

Table 2 shows the classification accuracy achieved by the SVM models using information from all 12 hardware events to classify several different sets of samples. The initial set, labeled as "All", includes all samples from the data set. The remaining sets only contained samples from the *gap* subsets ranging between 1KB and 32KB. The performance is measured as the percentage of samples classified correctly for each set.

The results included in Table 2 correspond to a *training set* containing 10,000 samples and a *test set* containing the remaining samples.

Overall, the two-class SVM model showed very good performance. In the worst case, including all samples, this model achieved a 92.8% detection accuracy with a 0.99% false negative rate. On the other hand, the one-class SVM model did not prove as effective. For the best case, with a 8KB *gap* set, it achieved a detection

[3]We refer to an "X *gap*" subset as the group of samples only including *bad* samples that leaked X bytes or more ($R_{size} \geqslant T_{size} + X$ bytes), and all the *good* samples as ($R_{size} = T_{size}$)

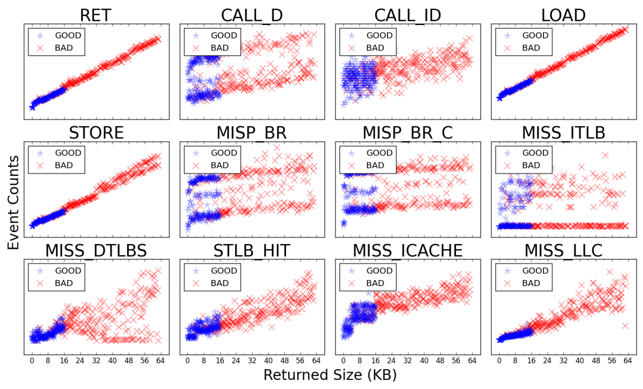

Figure 5: Hardware-event counts scattered plots. Each y axis is normalized to the range of its individual event.

accuracy of 74.55% with a 0.99% false negative rate. This is due to the fact that no *bad* samples are included in the *training set* for one-class SVM and that the overlapping between *good* and *bad* samples is significant. The results demonstrate the limitation of this technique for detecting unknown attacks.

5.5 Hardware Event Subsets

Based on their individual classification accuracy, as shown in Figures 2, 3, and 4, we identified the 6 most effective hardware events being RET, LOAD, STORE, MISS_ICACHE, STLB_HIT, and MISS_LLC. Because the processor used in the experiments has four built-in configurable performance counters, we studied which subset of four hardware events offered the best performance among all. For each individual subset, we evaluated the classification accuracy of several leaked data *gaps* between 1KB and 64KB using the two-class SVM model.

The results show that subset classification rates improved as the size of the *gap* grew larger. For each subset, we computed the average of their classification accuracy for all tested data *gaps*. Among all subsets, the classification average ranged between 96.8% with the four-event subset of [LOAD, STORE, STLB_HIT, MISS_LLC] and 97.8% with [RET, LOAD, STLB_HIT, MISS_ICACHE].

We observed that when the size of the data *gap* was 8KB or larger, all subsets achieved a detection performance higher than 98%. Although the detection accuracy varied among all subsets, the difference appears to be marginal (around 1%). On the other hand, the larger variation in the classification performance among individual subsets was observed for the smaller *gap* sizes.

6. HARDWARE EVENTS IN-DEPTH BEHAVIOR ANALYSIS

In this section, we conduct a qualitative analysis of the hardware events studied.

Figure 5 shows the relationship between the returned data size of a Heartbeat request (R_{size}) and its associated individual hardware event count. The good cases cannot request more than 16KB while bad cases can request up to 64KB. As it can be seen, a linear relationship is observed between these two parameters for hardware events such as RET, LOAD, and STORE. However, individual plots of other hardware events, e.g., CALL_ID, MISP_BR, and MISS_ITLB show a very different reality.

It might be natural to expect a consistent behavior of hardware events for the same set of operations (e.g., execution of a program or a function within a program). However, many factors affect the way in which programs are executed, e.g., hardware resources/configuration, OS internals, etc. As a consequence, the "reliability" or "stability" of hardware events may change depending on their likelihood of being affected by these factors. Henceforth, we separate the hardware events used in this study into three categories as shown in Table 3. Some events that we did not monitor during the *Heartbleed* experiments are also included (noted by $^+$), which we studied with a separate set of experiments.

Table 3: Hardware event categories

Behavioral Instructions	Instructions Retired - Loads, Stores - Indirect Calls, Returns - Direct Calls - Branches Taken$^+$ — Conditional Branches Taken$^+$
Behavioral Data	Memory Operands - Reads$^+$, Writes$^+$
Models	Mispredicted Branches - Mispredicted Conditional Branches Last Level Cache Misses - I-Cache Misses, D-Cache Misses$^+$ TLB Misses - I-TLB Misses, D-TLB Misses - Shared TLB Hits after I-TLB Miss

The hardware events contained within what we call *Behavioral - Instructions* group show the most stable behavior in Figure 5. This is somewhat expected since they are mostly deterministic events that are independent of the context and the state of the hardware. The second group, which we call *Behavioral - Data* group, includes events related to memory accesses. We evidenced a consistent relationship between these events and the instructions being executed. We observed that not only the instructions but also the list of memory operands to be processed had a direct impact on the hardware counters. Overall, this group of hardware events still experiences relative stability. Finally, we consider the events within what we call *Models* group to be platform dependent and not reliable. There is too much context needed in order to have consistent behavior and make sense of the data offered by the counters among different executions. These events are significantly affected by co-executing programs and by the size and configuration of the cache subsystem. They are also susceptible to how busy the cache subsystem is at every given moment. Furthermore, these factors change from machine to machine.

Overall, selecting the right combination of hardware events might imply a trade-off between accuracy (linked to one specific platform) and portability. If events from the last group are required, then most likely the model of the program being analyzed must be built on the same exact platform it will be executed. The first two categories are less restricted because they should remain consistent across platforms.

7. NEED FOR MORE ARCHITECTURAL SUPPORT

The results presented in this work show certain similitudes with similar detection methodologies applied to *control-flow* exploits in previous research [9, 7, 16]. However, the detection effectiveness of this general methodology varies significantly between *data-oriented* and *control-data* attacks. Detection of the *control-data* exploits heavily depends on the assumption that an attack will disturb the control-flow of the target program. However, such assumptions do not hold true for *data-oriented* exploits. In fact, preserving the normal control-flow of the target program is one of the restrictions of *data-oriented* attacks [9].

In previous research, the statistical distribution of the hardware event counts show remarkable differences between the normal execution and the execution during the attack [7, 13]. For the *Heartbleed* case study, some hardware events show clear separation between the *good* and *bad* cases for higher leakage gaps (leaked gap $\geqslant$ 8KB). However, for smaller gaps (leaked gap $<$ 8KB), the overlapping of the statistical distribution of the *good* and *bad* cases was significant, making it very hard to discern between them.

Furthermore, the study presented in Section 6 demonstra-ted that even though some events showed significant differences between normal execution and the execution under attack, their stability and reliability was somehow compromised.

We believe that the incorporation of architectural support would improve the effectiveness of the methodology studied in this work and most systems using hardware performance counters. We suggest the following as realistic features that could be included with future-generation processor architectures:

– **Increment the total number of performance counter registers:** This would allow expanding the total number of hardware events to be monitored simultaneously. It would provide software components with the possibility of deploying more advance detection mechanisms. It will also make it more difficult for an attacker to mimic an attack to have a hardware events' footprint as that of the normal execution.

– **Expand the number of architectural performance-monitoring events**[4]**:** This would increase the reliability of hardware events across different platforms. It would most likely make some events grouped in the *Models* category to be less platform dependent and therefore more reliable among different architectures.

Another aspect that would help the performance of an anomaly detection scheme is the addition of general classification hardware resources into new generation processors. We could take advantage of such built-in feature to conduct online monitoring of software behavior, based on hardware performance counters with improved speed and reliability.

8. CONCLUSION

In this work, we conducted a systematic study to determine if low-level hardware events can be effectively used to identify the occurrences of buffer over-reads attacks in user space libraries. As a test case, we studied the *Heartbleed Bug*, a serious vulnerability in the popular OpenSSL cryptographic software library.

Our experiments demonstrate that first, with certain limitations, it is possible to use information collected from hardware events to detect buffer over-read attacks in user space libraries, OpenSSL in this case.

Second, an extensive study of hardware events showed that different hardware events experienced different sensitivity to the studied attack. Hence, monitoring the correct events is key for the performance of the anomaly detection scheme.

Third, non-deterministic events, e.g., last level cache misses, showed potential for differentiating between normal and abnormal behavior. However, they also appeared to be susceptible to user and kernel space activity unrelated to the studied program, therefore making them less reliable in uncontrolled environments. In addition, we present an in-depth qualitative analysis of different hardware events and classify them into three categories according to their "reliability" and "stability". We analyze which factors contribute to such differences and discuss the limitations of certain type of events.

[4]Performance monitoring events are architectural when they behave consistently across microarchitectures [10].

Fourth, we employed a Support Vector Machine (SVM) algorithm to implement an anomaly-detection scheme. Results showed a detection accuracy over 92% for the two-class SVM model and over 70% for the one-class SVM mo-del. For our case study, the SVM algorithm proved to have limitations for detecting unknown *data-oriented* attacks.

As for future work, we plan to extend the number of hardware events and derived metrics included in our study, as well as include other vulnerabilities/exploits to further validate the feasibility of the proposed scheme. We also want to explore the use of more sophisticated classification algorithms, including current machine learning models.

9. REFERENCES

[1] The heartbleed bug. http://www.heartbleed.com.

[2] Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Trans. Inf. Syst. Secur.*, 13(1):4:1–4:40, November 2009.

[3] Eep Bhatkar, Daniel C. Duvarney, and R. Sekar. Address obfuscation: an efficient approach to combat a broad range of memory error exploits. In *In Proceedings of the 12th USENIX Security Symposium*, pages 105–120, 2003.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[5] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, May 2011.

[6] Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. Non-control-data attacks are realistic threats. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM'05, pages 12–12, Berkeley, CA, USA, 2005. USENIX Association.

[7] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. On the feasibility of online malware detection with performance counters. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 559–570, New York, NY, USA, 2013. ACM.

[8] John Demme and Simha Sethumadhavan. Rapid identification of architectural bottlenecks via precise event counting. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 353–364, NY, USA, 2011. ACM.

[9] Hong Hu, Zheng Leong Chua, Sendroiu Adrian, Prateek Saxena, and Zhenkai Liang. Automatic generation of data-oriented exploits. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 177–192, Washington, D.C., August 2015. USENIX Association.

[10] Intel. Intel 64 and ia-32 architectures software developer manual. Technical report, Intel, 2013.

[11] Netcraft. April 2014 web server survey. http://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html, April 2014.

[12] Jared Stafford. Heartbleed proof of concept. https://gist.github.com/10100394, 2014.

[13] Adrian Tang, Simha Sethumadhavan, and SalvatoreJ. Stolfo. Unsupervised anomaly-based malware detection using hardware features. In Angelos Stavrou, Herbert Bos, and Georgios Portokalidis, editors, *Research in Attacks, Intrusions and Defenses*, volume 8688 of *Lecture Notes in Computer Science*, pages 109–129. Springer International Publishing, 2014.

[14] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. http://www.openssl.org, April 2003.

[15] Gildo Torres and Chen Liu. Adaptive virtual machine management in the cloud: A performance-counter-driven approach. *Int. J. Syst. Serv.-Oriented Eng.*, 4(2):28–43, April 2014.

[16] Xueyang Wang and R. Karri. Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–7, May 2013.

[17] Lichen Weng, Chen Liu, and Jean-Luc Gaudiot. Scheduling optimization in multicore multithreaded microprocessors through dynamic modeling. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '13, pages 5:1–5:10, New York, NY, USA, 2013. ACM.

[18] Wucherl Yoo, Kevin Larson, Lee Baugh, Sangkyum Kim, and Roy H. Campbell. Adp: Automated diagnosis of performance pathologies using hardware events. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 283–294, New York, NY, USA, 2012. ACM.

[19] Chao Zhang, Tao Wei, Zhaofeng Chen, Lei Duan, Laszlo Szekeres, Stephen McCamant, Dawn Song, and Wei Zou. Practical control flow integrity and randomization for binary executables. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 559–573, Washington, DC, USA, 2013. IEEE Computer Society.