IMPROVING SYSTEMS DEPENDABILITY THROUGH PRIVATE DISTRIBUTED COMPUTATION AND ANALYSIS OF ANOMALIES

by

Rufaida Ahmed

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

December 2021

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Rufaida Ahmed**

has been approved by the following supervisory committee members:

Robert Preston Riekenberg Ricci ,	Chair(s)	**11/23/2021** Date Approved
Jacobus Eramsus Van Der Merwe ,	Member	**11/23/2021** Date Approved
Vivek Srikumar ,	Member	Date Approved
Ryan Stutsman ,	Member	**11/23/2021** Date Approved
Kuang-Ching Wang ,	Member	Date Approved

by **Mary Hall** , Chair/Dean of

the Department/College/School of **Computing**

and by **David B.Keida** , Dean of The Graduate School.

ABSTRACT

If you survey computer systems' users, most of them will agree that one of the most important system properties is dependability. The dependability of a system expresses the users' trust in that system and how confident they are relying on these systems. It conveys the extent of the user's confidence that it will operate as users expect. In general, dependability includes inter-dependent systems properties such as availability, safety, security, and resilience.

In this dissertation, I look closely at different complicated computer systems. The goal is to enhance the systems' trustworthiness and eventually improve their overall dependability.

First, I look at the edge computing paradigm, specifically content distribution networks. Although CDNs are widely used and highly beneficial, they require end-users to share their secrets with the CDN to maximize benefits. This imposes a serious security threat. To tackle this issue, I present Harpocrates. It is a framework that utilizes Intel's SGX technology to maintain the privacy of sensitive secrets while still allowing the end-users to leverage the CDN's capabilities.

Second, I look at complex system logs and attempt to model the system's normal behavior using invariant mining. This study aims to answer two questions: Are automated anomaly detection tools, specifically invariant mining useful in datacenters? And; does the anomalous behavior persist over time? To answer these questions, I study the logs from one year of operations coming from the Cloudlab testbed. The results contained valuable insights indicating that although some of the invariants persisted throughout the year, the detectors must be re-trained systematically to capture new invariants.

Finally, after studying different anomaly detection tools, I found that they do not specifically label malicious anomalies. They either disregard the malicious class, or they consider all anomalies malicious. These approaches pose a problem for system admins who need to differentiate between benign anomalies and malicious requests. To solve this problem,

I designed Deep-Sec. It is an anomaly detection framework that introduces a novel fine-grained scoring system that enables system admins to distinguish between benign and malicious anomalies and enhances accuracy on long sessions driven by human behavior rather than program structure.

For my parents and husband, for their belief in me, endless support and sacrifices. For my son Ali, who endured my absence during this work, I hope this will inspire you to achieve your own greatness. And for my siblings and friends for their constant encouragement, even when I was away.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This dissertation is based upon work supported by the National Science Foundation under Grant Numbers CNS-1419199, CNS-1314945, 1743363 and 1801446. and a fellowship from the University of Utah School of Computing.

Most importantly, I thank my advisor and committee chair Professor Robert Ricci for his support and direction during this work. His expertise was invaluable in formulating the research questions and methodology. I also thank all my committee members for their helpful input and review of my work.

I thank the Flux team members and the Cloudlab administrators for access to the data and help interpreting it; in particular, I thank Mike Hibler, Gary Wong, David Johnson, and Aleksander Maricq for manually labeling sessions. I also thank Vivek Srikumar for his help selecting and interpreting ML techniques, and to the anonymous MLCS reviewers whose comments helped to improve the dissertation. And I thank Sachin Kumar Singh and Mugahed Izzeldin for their valuable input and help with the data processing .

Finally, I thank Guineng Zheng, who was tremendously helpful with understanding Deeplog implementations and finding datasets.

CHAPTER 1

INTRODUCTION

Looking at the complicated structure of modern computer systems, especially data centers and cloud environments, and the exponential growth in the number of users depending on these systems, you will quickly realize the significant need for robust, trustworthy, and dependable systems. In 2017, the CDN provider Cloudflare [19] had the famous Cloudbleed incident [36], where a bug caused a leak in private data from one website's sessions into other website's sessions. This may have led to personal information being leaked, and it may have even leaked website's private encryption keys. With these keys, an attacker could fully impersonate a website, authenticate himself to other websites, and decrypt it's traffic. In 2015, the famous open-source CMS WordPress [81] was compromised due to a vulnerability that enabled attackers to carry Cross-site Scripting (XSS) attacks. This attack enables adversaries to execute any code they want or reveal any information on the target website. It is also very hard to model the correct or expected behavior of these systems. This behavior modeling is crucial because it helps us identify system error and anomalies and protect the systems from malicious adversaries. These incidents and many more prove that some of the systems that users rely on for critical services have serious design flaws.

In this dissertation, I considered different domains where I aim to improve systems' dependability.

I look at three specific aspects that are important to ensure the trustworthiness of systems. First, in Chapter 2, I look at the privacy of CDN networks. Content Distribution Networks (CDNs) offer websites and web services the ability to host content on servers near the edge of the network, close to users. Benefits of this arrangement include low latency, scalability, and resistance to Denial-of-Service attacks. The current emergence of pushing active computation to the edge in CDNs offers a wealth of new opportunities

for web services to become faster and more scalable. However, it also increases the exposure to new security threats such as the leakage of secret materials that are accessed by this function. To mitigate this issue, I present a framework called Harpocrates. This framework allows active code to be pushed from an origin web server out to workers at the edge of a CDN without compromising the privacy of the secrets. Harpocrates makes use of Intel's SGX technology to keep data private and presents an environment similar to the JavaScript WebWorker API to simplify the process of code that can run on either origin servers or the CDN. To demonstrate the capabilities of Harpocrates, I implemented four useful applications: The first application allows CDNs to maintain protection against DDOS attacks while keeping the origin server services available to genuine users, the CDN performs a security check at the edge by checking the secure login cookies generated by the origin server. The second application improves on the first application by adding a new security measure handled within the CDN by checking the users' password at the edge without compromising any sensitive information. The Third application allows the CDN to perform data aggregation from multiple sources at the edge to provide improved latency and scalability to the users. The fourth application is an improvement of keyless SSL[20], it allows CDNs to serve HTTPS requests without compromising the origin server's private keys.

In Chapter 3, I look at complex system logs and try to understand the correct and anomalous behavior of systems using anomaly detection. The set of anomalies seen, however, can change over time: as the system evolves, is put to different uses, and encounters different workloads, both its typical behavior and the anomalies that it encounters can change as well. This naturally raises two questions: how effective is automated anomaly detection in this setting, and how much does anomalous behavior change over time? I examine these questions for a dataset taken from a system that manages the lifecycle of servers in datacenters. I look at logs from one year of operation of a datacenter of about 500 servers. Applying state-of-the-art techniques for finding anomalous events, I find that there are a core set of anomaly patterns that persist over the entire period studied, but that in to track the evolution of the system, I must re-train the detector periodically. Working with the administrators of this system, I find that, despite these changes in patterns, they still contain actionable insights.

In Chapter 4, I attempt to tie system logs to security incidents in the system. I achieve this by designing an anomaly detection framework, i.e., Deep-Sec. This framework improves on DeepLog [24], rather than performing just binary classification of the data into normal or anomalous, it divides log data into three classes: normal sessions, benign anomalies, and malicious anomalies. Deep-Sec achieve this by introducing a novel identifier called anomaly-score that can help distinguish between the three classes of anomalies especially the two anomalous classes. To evaluate the performance of Deep-Sec I have collected, manually labeled, and released a dataset from a real datacenter environment that has the three classes of data. To the best of our knowledge, no such dataset exists. This dataset is specifically interesting because it reflects a real human behavior within a website over a period of four months, and thus it is much harder to find a pattern in the data compared to datasets that represent a code execution or an automated process. The performance of Deep-Sec was evaluated quantitively by computing its accuracy and showing that its performance is acceptable and qualitatively by examining the types of attacks discovered and how useful can it be for system administrators. Furthermore, I measure the predictive ability of Deep-Sec to evaluate how early in a session can Deep-Sec attach the correct label to a session. This is done to measure the feasibility of implementing a future version of Deep-Sec that can block malicious requests online. Tests has shown that Deep-Sec cannot only work with regular logs deployed in systems but also expose new malicious behaviors not found by other tools, and hence it can be considered a complementary tool with other intrusion detections systems or any vulnerability detection tool.

Combined, these studies aim to address the aforementioned reliability issues. By deploying Harpocrates, users will no longer worry about incidents like Cloudbleed, because they are no longer relying on the typical security measures used by the CDNs to protect their sensitive information. And by using Deep-Sec as a complementary tool to other vulnerabilities detection tools, system admins will have peace of mind knowing that they will be alerted in case of malicious attacks or a compromised vulnerability like the XSS attack rather than not being able to prioritize system anomalies and knowing what is urgent and what is not.

CHAPTER 2

HARPOCRATES: GIVING OUT YOUR SECRETS AND KEEPING THEM TOO

Content Distribution Networks (CDNs) offer websites and web services the ability to host content on servers that are near the edge of the network, close to users. Benefits of this arrangement include low latency, scalability, and resistance to Denial of Service attacks. Traditionally, CDNs have hosted primarily *static content*, but increasingly, there is an interest in pushing *active computation* to the edge as well. This active computation, which is similar in style to the "serverless" computing becoming popular in clouds, offers a wealth of new opportunities for web services to become faster and more scalable. With this opportunity, however, comes a much greater exposure to security threats. One is leakage of secret materials (such as keys, identities, etc.) that are accessed by these functions. Another is the possibility that sensitive calculations are not executed faithfully in the CDN; e.g. a modified version of the customer's code is run.

In this chapter, we present the design of Harpocrates, a framework that allows active code to be pushed from an origin webserver out to workers at the edge of a CDN. Harpocrates makes use of Intel's SGX technology to keep data private, and presents an environment similar to the JavaScript `WebWorker` API to simplify the process of code that can run on either origin servers or the CDN. We use Harpocrates to design a number of interesting services, including a service that generates and checks secure cookies within the CDN, and a framework that protects against denial-of-service attacks in a way that is customized to a specific website. We show that the framework performs well enough to be deployable in practice.

2.1 Introduction

Content distribution networks (CDNs) are widely used to enhance user experience, reliability, and security through providing a system of distributed servers and networks

that can deliver internet-based services to the user from locations at or near the network edge. Many CDNs [1, 35, 54] work by "fronting" for an origin webserver. In this configuration, DNS lookups for the origin server resolve to IP addresses belonging to the CDN rather than the origin server itself. The CDN then acts as a reverse proxy or "man in the middle," processing all requests for the website and deciding whether to serve pages out of its cache, relay requests to the origin server, apply DDoS prevention methods, etc.

While this model is simple for the origin server, and provides strong protection against DDoS by hiding the true IP address of the origin, it leads to a number of security problems such as leakage of tenant data [36] and sharing of private keys between the content provider and the third-party hosting entity [15]. It has also, historically, been limited to static content, leaving the actual application code running on origin servers or in the cloud, where the latency benefits of caches at the edge are not realized.

Computing, however, is starting to become available at the edge. Cloudflare JavaScript Workers [73] are one example of "edge-computing" available in a CDN. The basic principle behind Cloudflare Workers is to allow origin servers to provide JavaScript code which will be run within the Cloudflare CDN itself, reducing latency for dynamically-generated content, and offering the origin server a simple way to scale. The high configurability offered with Cloudflare workers turns Cloudflare from just a CDN service into an edge computing platform.

Another example of this type of edge computing is AWS Lambda@Edge [2]. This service also allows the client to push code to the edge, allowing it to be closer to the end user to minimize latency. This code will typically be triggered by events from the Amazon CloudFront CDN. After pushing the code to the edge, AWS will take responsibility of code management duties such as replication, scaling and routing.

This arrangement increases security and privacy concerns for the CDN's clients. Generally, CDN clients want to keep most of their code secured and protected against both the CDN and possible malicious entities access, and using such a facility requires the client to distribute both code and potentially sensitive data throughout the CDN's network.

In this chapter, we present Harpocrates[1], a system that allows origin servers to dis-

[1]Harpocrates was the ancient Greek god of secrets and confidentiality

tribute sensitive code and secret information throughout the world without having to worry about it being leaked. This enables secure computation at the edge, and will allow for faster, more responsive, and more dynamic web services. Harpocrates makes use of Intel's Software Guard Extensions (SGX). It provides an abstraction based on the JavaScript Worker model, which is widely used on both the client and server side (Node.js) of web applications; this enables ease of offloading to the CDN from either direction.

We claim that Harpocrates enables multiple interesting applications that can help improve CDNs by providing storage for more critical information on the CDN without comprising the privacy and security of data and code. To demonstrate the power of the framework, we designed four such applications. The first two use cases help keep the origin webserver available to legitimate users during Distributed Denial-of-Service (DDoS) attacks. They check secure login cookies set by the origin server to identify users who have legitimately logged in to the service; they use a secret provided by the origin server to establish the authenticity of the login cookies, and use Harpocrates to prevent compromise of that secret. The second application goes beyond simple checking of an existing cookie by adding an additional authentication step, handled entirely in the CDN and using the user's own password for the website. Our third use case allows aggregation of data from multiple sources to occur at the edge of the network, in the CDN, for scalability and latency reasons. The fourth is an adaptation of the Keyless SSL [20] protocol to allow the CDN to serve HTTPS requests for the origin without requiring access to the latter's private keys.

The rest of this chapter is organized as follows: We begin by giving background on the technologies that we use in Section 2.2 and covering the related work. Our threat model, in particular the details of the materials that we seek to keep secret, is described in Section 2.3. The design of Harpocrates is covered in Section 4.4, followed by a description of several applications that it in enables in Section 2.5. Section 2.6 has some brief notes on the implementation, and Section 2.7 provides an evaluation of the overheads associated with using SGX in this setting and demonstrating the benefit to website performance of moving active code to the edge. We conclude and discuss future work in Section 4.6.

2.2 Background and Related Work

In this section, we provide background on the technologies that we use and cover related efforts.

2.2.1 Intel SGX

Intel Software Guard Extensions (Intel SGX) is a set of extensions to the Intel architecture that are designed to provide integrity and confidentiality for application running in ring 3 from parties running in the privileged ring, including the operating system, BIOS, Virtual Machine Manager, etc. This allows users to run trusted code on untrusted servers. It gives users a high level of confidence that private data will be as secure in a Cloud or Edge Cloud as it would be running in a local trusted environment.

An SGX application consists of two parts: an untrusted part and a trusted part. The trusted part is also referred to as an "enclave". An enclave is an execution container instantiated by an untrusted application. After being instantiated, the untrusted part can issue an Enclave Call (ECALL) to switch into the enclave and start the trusted execution. Similarly, the code inside enclave can switch to the untrusted world by making an Out Call (OCALL). An ECALL is a call made into an interface function within the enclave, triggering the enclave to execute a piece of code after necessary security checking, while and OCALL allows code in the enclave to make use of outside services like system calls or other functions that requires privileged permissions [41].

Harpocrates mainly uses two features of SGX. One is its secure execution environment: data belonging to an enclave is presented as plaintext within the CPU, but this data is encrypted with integrity protection applied when it is flushed from the cache to DRAM. Because SGX flushes the CPU cache and other memory mappings when switching in or out of an enclave, not even a privileged party, e.g. the kernel, can see the plaintext of the enclave's memory: it cannot see the previous contents of the CPU's cache, and can only see the encrypted copy of the content in the memory [2]. Any modification to the encrypted content in physical memory will lead to general protection fault when the trusted entity get scheduled to run again.

[2]Recently, Spectre-like attacks have been discovered against SGX [16]. Like all other SGX-enabled applications, Harpocrates' security will depend on the development of effective countermeasures against these attacks.

The other SGX feature Harpocrates uses is remote attestation. Because most trusted code requires some set of secret data to operate on, there must be a way to securely communicate that data to the enclave. Furthermore, it is critical that the party sending this secret data have a way to trust that it is indeed talking to specific, known, trusted code, and that this code is running in a real SGX enclave (and not, say, an emulation of SGX that will allow the attacker to access the data). SGX provides these facilities through a feature called "remote attestation" that enables an enclave to "attest" the identity and integrity of the code to a remote party. The Harpocrates design uses this mechanism to transfer secrets from the origin server into the enclave, and to give the origin server confidence that its functions are being faithfully executed.

2.2.2 TLS and Keyless SSL

The traditional TLS protocol is primarily designed to secure end-to-end communication. This was appropriate for older styles of web service when communication was only between two parties: the client (browser) and the webserver. However, when the connection is terminated in an intermediate node (such as a "reverse proxy" CDN) the security guarantees of these protocols no longer apply. In the CDN world, in order to provide HTTPS service, CDN providers that "front" for their clients domains needs to impersonate the original content provider by having the private key at hand, which brings with it security problems inherent in sharing a private key with a third party. To mitigate the problems brought by key sharing, Keyless SSL [20] was introduced by Cloudflare. Keyless SSL exploits the fact that private key is only used once in each TLS handshake to split the whole TLS handshake geographically, with most of the handshake work happening at the Cloudflare's edge. The private key itself remains on the origin server, which is contacted once per SSL/TLS handshake to identify the origin server. The security property of Keyless SSL has attracted significant attention and has been successfully applied in the wild [67]. However, Keyless SSL suffers from significant performance degradation and limited scalability due to the extra round trip from the CDN to Key Server in each handshake [78]. Researchers have sought alternative solutions to this problem and one of the main directions is using SGX to enable the private-key-holding sever to run in or near the CDN [78], eliminating the additional round trip.

2.2.3 Related Work

There is significant other work on exploiting SGX capabilities to enable trusted computation in the Cloud and Edge. For example, Haven [8], SCONE [3] and Graphene-SGX [71] make attempts to run entire containers in a shielded environment. mbTLS [51], ShieldBox [70] and EndBox [34] use SGX to protect middleboxes. This work makes all execution and data within the enclave trusted, and essentially only uses the third party infrastructure as a computation support. However, we argue that to make full use of the benefits conferred by a CDN provider, a tenant needs to expose some information to the CDN so that it can help the tenant optimize their performance and security, e.g. caching and DDoS prevention. Harpocrates targets a different point in the design space in which we do allow the CDN to see less-critical "derived secrets" while avoiding catastrophic incidents in which "master secrets" are leaked.

Bhargavan et. al [9] explore a means to optimize Keyless SSL itself, but does not consider computation need at the edge. mcTLS [52], Blindbox [62], and Splitbox [4] explore different ways to keep secrets from untrusted parties, but they emphasize the protection of content instead of of private key protection. STYX [78] does explore the space of key management, using SGX to enhance key distribution. Harpocrates differs from it in that we focus on pushing execution of code from the origin server to the edge, a broader use case than SSL key management.

2.3 Threat Model

The first element of our model is that the origin server has a "master secret" from which other secrets are generated, and that protecting this secret is higher priority than protecting the values that are derived from it. This situation is common in modern web services. For example, a master secret is often used to generate cryptographically secure cookies. Theft of an individual cookie is undesirable, as it allows the thief to impersonate a user for the duration of the cookie's validity or until the theft is discovered and the affected user can have her cookie re-generated. Theft of the master secret, on the other hand, is catastrophic, as it allows the thief to *generate* correct cookies, and therefore impersonate *any* user, and it can only be remedied by regenerating *all* users' cookies. Similar situations arise with TLS/HTTPS: leaking the key for a specific session is problematic, but leaking the private

key used to authenticate the server is much worse. Our goal in Harpocrates is to protect the master secret, as it is not always possible to protect derived secrets and still have the CDN do its job.

Second, we assume that the CDN needs to be able to see requests and responses "in plaintext" to do its job; without this, it cannot know whether to serve responses from its cache, whether to send requests to the origin server, whether to dispatch them to a secure function, etc. All of these decisions are based on information in the HTTP request, such as the request URL and headers (particularly cookies). This is why we only seek to protect master secrets: for example, if a secure function sends a cookie to a client, that cookie will be seen by the CDN in the next request the client makes. We do assume that the connection between the client and the CDN is, or can be made, over HTTPS; techniques such as Cloudflare's Keyless SSL [20] make this possible while keeping the origin's private key secret (though they may leak session keys).

Third, we assume that the CDN may leak any information that it sees in plaintext; this includes the contents of HTTPS connections that are decrypted by the CDN through mechanisms like Keyless SSL. This may be due to a bug, such as in the case of CloudBleed [36], in which Cloudflare inadvertently leaked memory belonging to one origin in bytes of connections for other origins' clients. It could also be due to a malicious insider at the CDN or a malicious third party that has compromised the CDN.

Finally, we assume the correctness of Intel's SGX and remote attestation protocols. While there do exist attacks against SGX [55], we assume that mitigations are available or that they are closed in future revisions of SGX.

2.4 Design

The fundamental element of security in Harpocrates comes from leveraging Intel SGX. SGX supports code secrecy by putting sensitive code and data into CPU-hardened protected regions called enclaves. Intel also provides a remote attestation mechanism which can prove to others that they are really communicating with the specific, known, code in a real SGX enclave, and not an impostor. As part of this process, the remote party can provide data (master secrets, in our design) that can only be decrypted inside the enclave.

Harpocrates, following the SGX Developer Guide [41], splits code into two parts: *un-*

trusted code, which runs outside of an SGX enclave, and `trusted` code that runs inside of it. All entry points are in the untrusted code: the CDN delivers requests that it receives from clients to untrusted functions, and expects to receive a reply from an untrusted function as well. Once invoked, an untrusted function may choose to call a trusted one: for example, to perform operations involving secret data.

We offer an API based on the JavaScript `WebWorker`, an abstraction that allows a caller to start a task by *posting* an event and registering a function to receive the *completion* of that event. These two calls map well to the ECall and OCall interface offered by SGX. An ECall is made from untrusted code, and invokes code inside of an enclave; this is analogous to posting an event. An OCall is made by trusted code inside an enclave, and calls the untrusted code registered to receive the completion. Another advantage of adopting the `WebWorker` design is that it is frequently used in server-side JavaScript in the Node.js environment, and thus facilitates moving code from an origin server to a CDN. It is also used in Cloudflare's `Workers`.

There are some differences between our design and the "normal" `WebWorker`. We require the user to supply a function in the trusted code that will be called on initialization of the enclave: typically, this code will perform remote attestation with the origin server to securely transfer a secret(s) into the enclave. In addition, the untrusted code cannot call arbitrary functions when it posts an event: it can only call functions that the customer has defined for the enclave and exported to be available for ECalls. Untrusted code may still register arbitrary functions to receive completions.

Figure 2.1 shows a typical CDN operating in "reverse proxy" mode. Note that though we depict the CDN as a simple "stack" of servers, in practice, it is typically a set of servers distributed around the world, and clients are directed to the closest one through DNS resolution or IP anycast. The basic goal of the CDN is to use the path represented by the solid arrows for as many requests as possible: these are the requests for which the CDN, sitting near the user at the edge of the network, can respond out of its cache. The dotted lines are essentially a fallback: when the content is dynamic or not yet cached, the CDN must forward the request to the origin server, and relay the response back to the client.

Figure 2.2 shows the data flow within the CDN in Harpocrates. Requests enter on the left from clients (marked #1 in the diagram) and reach the *director*. This director examines

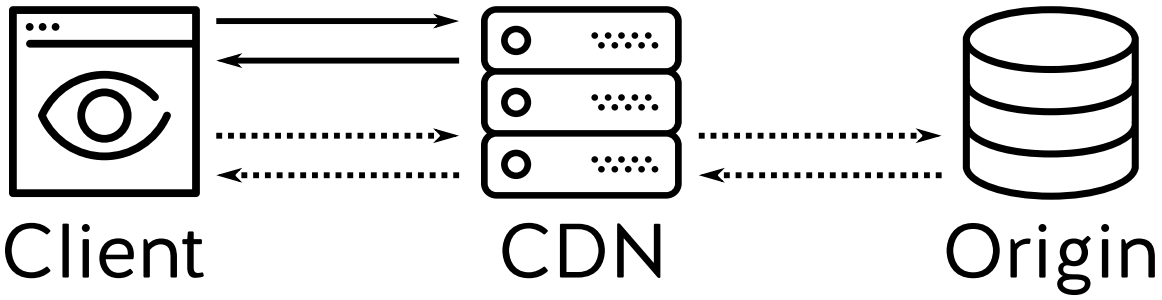

Figure 2.1. Typical "reverse proxy" CDN. Solid lines represent the CDN serving content out of its cache. Dotted lines represent the CDN forwarding traffic to the origin, and forwarding back the results.

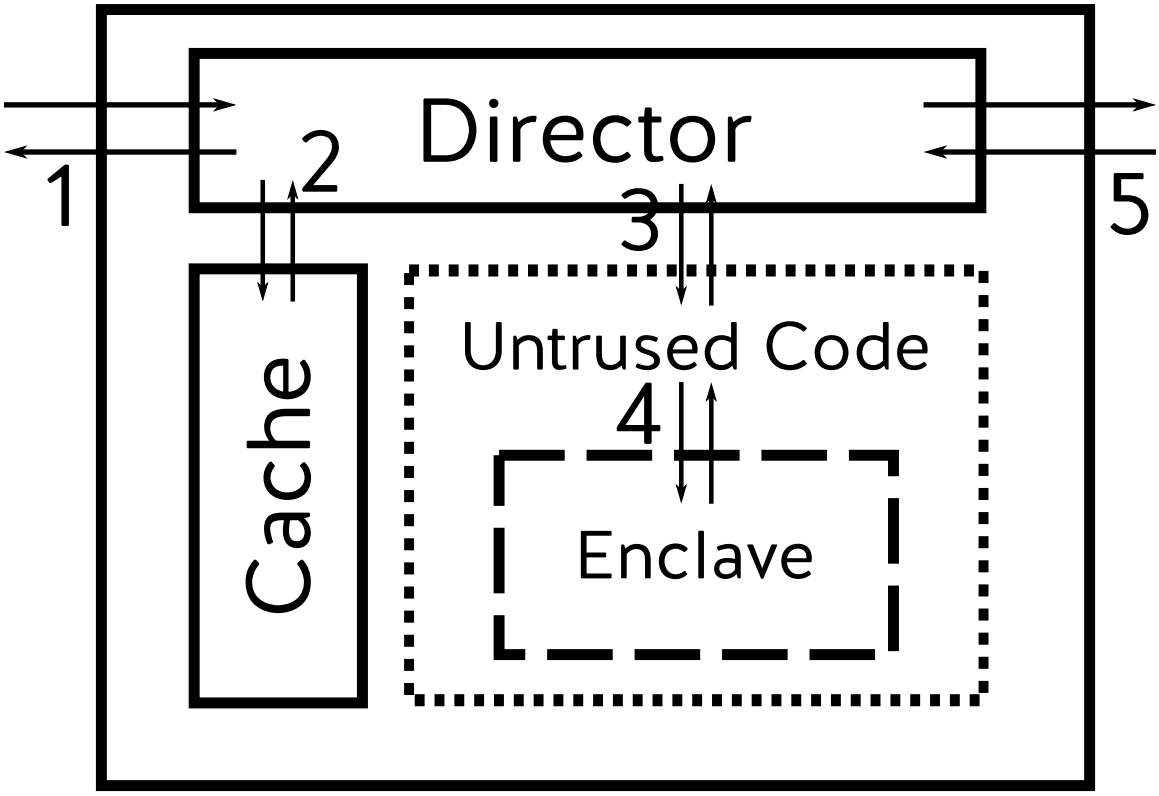

Figure 2.2. The design of Harpocrates. The client is to the left, and the origin server is to the right.

the HTTP request (acting as the endpoint of the TLS session for HTTPS connections), and selects one of several options. First, for static content, it may serve the content directly out of the CDN's cache (#2). Second, if the request URL belongs to an endpoint registered by the origin server's code, it forwards the request to the *untrusted* part of the extension's code (#3). This is the portion of the code supplied by the user that does not run inside of an SGX enclave and does not handle master secrets. In some extensions, no secret data may be needed, and this function returns data to the director (which passes it back to the client) without calling into an enclave at all; this looks similar to Cloudflare's Worker model. For computations on secret data, the extension calls into the enclave (#4) by posting a SecureWorker event. Computation on the secret data—such as secure cookie generation—occurs in the enclave, which returns the data to the untrusted code, and from there, back to the director and client. Finally, the director still has the option of sending requests that cannot be handed by either the cache or local code back to the origin server (#5).

The CDN is responsible for deciding when and where to bring up new instances of the customer's function, along with the associated enclave. Each enclave must have an initialization function; typically, this function will contact the origin server, use remote attestation to authenticate itself, and be given the master secret through this secure channel. While this bootstrapping procedure does put some additional load on the origin server, it only needs to happen once for each host in the CDN network, which is much lower than the total number of end clients.

An important point that developer should keep in mind is the fact that secrets and sensitive information should not be statically compiled into the enclave, and must only be securely transferred to the enclave at runtime during remote attestation. Code and data used to initialize an enclave are, by design, unencrypted in SGX. Any secret loaded before attestation can possibly be inspected by the unauthorized CDN or a malicious party.

2.5 Example Applications

Using a content delivery network (CDN) to host scripts, files, and even sensitive information that are frequently accessed can improve the overall performance of the origin website and conserve bandwidth. But unfortunately, using CDNs also comes with a risk. If an adversary gains access to these files he can inject arbitrary malicious content to change

or even replace those files.

According to surveys [15], it was reported that 76% of all organizations that use third-party hosting services such as CDNs share at least one of their private keys with this entity. Sharing such crucial data is risky, because the origin site has to trust not only the CDN's software, but also the CDN's system administrators, all those who have physical access to the provider's hardware, and any law-enfrocement body that might have authority to access the origin site's replicated data in the provider's physical location.

We now describe four applications that could be built using Harpocrates, the benefit they would have to users and origin sites, and the way that they fit into our security model.

2.5.1 Secure Cookie DoS Prevention

The most basic service that reverse proxy CDNs offer to their customers—after caching—is the ability to withstand denial of service attacks. Because end hosts—including attackers—see the CDN's IP address, rather than the origin's true location, all traffic, benign and malicious, passes through the CDN. The CDN aims to have enough capacity, in terms of bandwidth and servers, to withstand any DoS attacks that may be directed at its customers.

For static (cachable) content, the attack becomes against the CDN's resources and is up to the CDN to handle appropriately. For dynamic content, however, the problem is trickier: normally, requests for this content would be forwarded to the origin server, which would customize the response for the session, user, etc. Forwarding *all* requests for dynamic would clearly expose the origin server to the DoS attack, but the origin would often like to allow some class of requests—such as those coming from logged-in users—to be forwarded. Offloading these decisions to the CDN is problematic, as in order to make them securely, the CDN will need some way of identifying valid requests through, e.g. a cookie set in the request, and checking this cookie requires a secret.

In Harpocrates, this tension is resolved by allowing the origin to write a function that checks login cookies. These cookies are generated by the origin server on successful login using a widely-used method of applying a cryptographic hash to a concatenation of the username, a nonce, an expiration date, and a master secret. These cookies are quick to generate and check; all information other than the secret is sent in plaintext in the request, so all that must be done to check the validity of the cookie is to concatenate the secret

and check that the hash of this string matches the hash in the request. This master secret can be generated on the origin server, and shared with the enclaves running in the CDN. The untrusted code supplied by the origin to the CDN can parse the request, extract the necessary fields from headers, etc., and pass the appropriate fields to the trusted code to test. The trusted code, with access to the master secret, simply returns success or failure depending on whether the cookie validates, and the untrusted code takes the appropriate action to block or forward the request. The path of an individual request is shown in Figure 2.3.

2.5.2 Site-Specific DoS Authentication

The procedure above works so long as the adversary does not have access to a legitimate, valid cookie. If he does have one, such as by compromising a legitimate client, he can distribute this cookie to all attackers and pass the security check, allowing attack traffic to reach the origin server. Our next scheme protects against this case. When there is some indication that a particular cookie might be involved in an attack (e.g. evidenced by seeing the same cookie in many requests), we can enter a more cautious mode in which possessing a login cookie is not enough: one must also have a cookie that is tied to the user's IP address to prevent the same login cookie from being used by an entire DDoS botnet.

Login cookies are not typically tied to a particular IP address, because users are often on DHCP, behind NAT, move between networks, move between mobile and WiFi networks, etc. When one wants to verify that the user behind a particular IP address is a person, rather than a bot, a typical way to do so is is through a CAPTCHA [76]. In many cases, today's reverse-proxy CDNs issue CAPTCHA tests to clients coming from IP addresses that they consider suspicious. A major problem with this arrangement is that this CAPTCHA test is essentially conducted "outside" the website; e.g. it provides a confusing user experience by serving up a page from the CDN rather than the origin website, asks the user to input information that they are not used to providing the website, etc.

Our scheme works much more cooperatively with the original website, as shown in Figure 2.4. When the regular login cookie checking function at the CDN decides that a

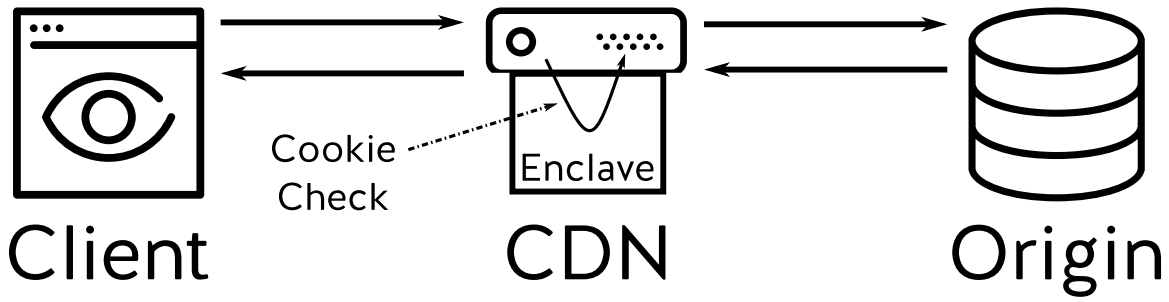

Figure 2.3. Simple checking of a login cookie in an enclave in the CDN.

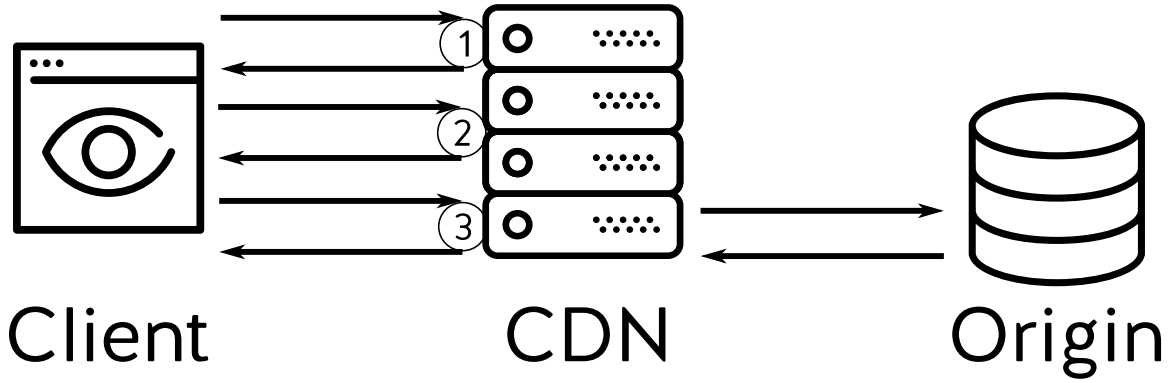

Figure 2.4. Exchanges when the enhanced protection mode is enabled. For visual simplicity, individual calls into the enclave are not shown.

login cookie might be being abused, it serves the client a page—clearly coming from the origin website, but served from the CDN—requesting that the user re-enter their password. This is shown as interaction #1 in the figure: the request contains a valid login cookie, but not an IP-binding cookie. The login cookie is checked as in Figure 2.3, and if it is correct, the worker in the CDN returns a page asking the user to re-enter their password, instead of forwarding the request to the origin server.

The user login database is shared from the origin to the enclaves: enclaves can easily contain tens of megabytes of memory, sufficient to hold login credentials for hundreds of thousands of users. The user's response is received by the origin's function on the CDN, where the password is checked inside the enclave (here, the password database is the master secret), and if successful, a new secure cookie, similar to the one used for normal logins is generated. This cookie, however, is tied to a particular client IP address. Thus, any bots attempting to use it from other IP addresses can be quickly and easily rejected within the CDN; requests coming from the correct IP address can be forwarded to the origin. If the user does roam to a different IP address, they will simply have the slight inconvenience of having to enter their password again. This is shown as interaction #2 in the figure: here, the request contains the user's password, the enclave checks that password, and, if the password is correct, the worker returns the IP-bound cookie and a redirect to the page the user was originally trying to access. In interaction #3, the request now contains a correct IP-bound cookie, and the function within the CDN checks this cookie and forwards the request to the origin.

Note that that this new authentication procedure takes place within the CDN: the origin server sees no additional load from the requests to re-enter passwords, whether successful or not. This solution therefore scales with the CDN, and this additional level of protection comes without additional load on the origin. The end user only sees the additional password dialog once (unless she or he changes IP address while the attack is ongoing).

2.5.3 Content Aggregation at the Edge

One of the documented use cases for Cloudflare's workers is aggregating information from multiple points within the CDN [18]. In this configuration, the worker grabs infor-

mation from multiple sources and creates a unified page to serve back to the user. This information can come from third-party APIs such as weather, mapping, stock, and social media services.

In many cases, such services are accessed by having an "API token" that belongs, in this case, to the origin server, and which is used to authenticate to the API. We treat this as a "master secret." For some APIs—particularly, those dealing with financial or personal information—the information being handled is very sensitive, and disclosure of the key can be very dangerous. For example, consider a bank which has a relationship with a credit reporting agency, which it uses to display credit scores to the bank's customers: displaying this information within the bank's website is a valuable service, but the bank must take great care to ensure that the key that it uses to authenticate itself to the credit agency is not compromised.

In Harpocrates, the origin's API keys can be kept within the enclave; this allows the origin to benefit from the latency and scaling advantages of aggregating information at the edge, without needing to worry about compromise of the keys it uses to gather that information.

2.5.4 Keyless SSL

Keyless SSL [20] is a design that allows a CDN to serve HTTPS connections with the origin's SSL/TLS certificate without needing to have access to the origin's private key. It does so by having a small server that runs (in the original design) on the origin, which keeps the private key and is contacted during TLS session establishment to perform the cryptographic operations that authenticate the server. Others have shown that the server holding the private key can be run within an SGX enclave, and that there are substantial latency and scalability benefits to doing so [78]. Harpocrates can be used as an alternate way to implement the keyless server.

2.6 Implementation

To implement Harpocrates, we use an existing package called SecureWorker [47] that allows JavaScript workers to run inside the trusted environment provided by an Intel SGX enclave. The execution environment outside of the enclave (for untrusted code)

is provided by Node.js, a popular engine for server-side JavaScript. Node.js is much too large and has too many dependencies to be practical for running inside of an SGX enclave, so SecureWorker uses the Duktape [25] JavaScript engine to provides a familiar `Worker`-like environment inside the enclave. Duktape is designed to be embedded in other environments, and is therefore very lightweight and has few dependencies. SecureWorker allows users to write isometric code and use the same code on client, server, and inside enclaves, it presents the secure workers as just another, secure and trusted, component in the JavaScript-based architecture. The SecureWorker package provides a rich API in which we can start a new worker, terminate it, receive and send messages between the trusted and untrusted part of the system, and even perform remote attestation services.

The original SecureWorker package provided wrappers for a subset of the JavaScript webcrypto API [77]; however, this subset was small, and only covered a few symmetric key ciphers and has functions provided by early releases of the Intel SGX SDK. Because we expect cryptography to be the main use case for trusted code, we extended these wrappers to cover public key cryptography and a larger set of ciphers and hashes from the OpenSSL library.

For the sake of this prototype, we focused on the design aspect of the system and did not include the full remote attestation in our implementation. The full remote attestation process requires a license from Intel, a signed certificate from a recognized certificate authority, and a registered service provider ID [74]; this does not add to the proof-of-concept value of our implementation.

2.7 Evaluation

We evaluate the feasibility of our approach by showing that it does not cause undue overheads on worker functions; showing the latency benefits that are to be gained from moving these workers to the edge; and showing the utility of the system by presenting a detailed case study showing how to build a website-specific DDoS defense mechanism.

Our evaluation was done in the CloudLab [69] testbed. All hardware SGX evaluation was done using an Intel `i5-6260U` processor.

2.7.1 Microbenchmarks

In general, small functions that do not use large amounts of memory execute at a similar speed within an SGX enclave with outside of ones [11]. We found this to hold for the types of functions needed for many fingerprinting and secure cookie schemes: we ran a SHA-256 hash of 20 bytes of data using the JavaScript `js-sha256` library. Outside of the enclave, this function was run in Node.js; inside, it was run using the Duktape JavaScript engine. In both cases, the actual hash implementation is native code. Outside of the enclave, we found that it took, on average, 0.0017ms (averaged over 10,000 repetitions) and 0.0029ms (averaged over 100,000 repetitions), while the same hash, run entirely within an enclave took only 0.0041ms on average (averaged over 100,000 repetitions). We found that the gap between the execution speed outside and inside the enclave gets smaller, in relative terms, as we increase the input message size. From this, we can conclude that for many small functions, running inside SGX will not, itself, introduce overheads that are significant in comparison to network RTTs.

There is, however, another source of overhead that we must consider: crossing the trust boundary to enter and exit an enclave *does* have a noticeable performance effect. We assume that workers implemented in our system will have two such boundary crossings: one ECALL made when the untrusted code posts an event that causes execution of the trusted code, and one OCALL when the trusted code posts results back. Under this assumption, the effect is still within reasonable range compared to wide-area network delays: when we re-factor the function above to put the testing loop in the untrusted code and the hash itself in the trusted code, the average time is 0.9ms. This is significant, but still well under the typical network round trip time between a client and origin server.

We do note that initialization of the enclave adds additional latency: in our experiments, this amounted to an extra 16ms, and would be much higher with fully implemented remote attestation. This is a one-time cost per CDN server, and CDNs have two choices for how to handle it. One would be to implement SGX enclave initialization on-demand and to shut down an enclave after some period of inactivity. This would keep overheads low by preventing the CDN from having to maintain many enclaves, but would result in occasional higher latencies seen by clients (e.g. the first to use the origin from a particular geographical region). Another would be to proactively and predictively bring up enclaves;

this could reduce the occasional latency spikes, but would require more sophisticated workload prediction and could mean more resources spent maintaining inactive enclaves.

2.7.2 End-to-End Benchmark

For the deployed topology configuration, we emulated link latencies for the user-to-CDN and CDN-to-origin links. In [10], the authors found that a typical round trip time between a client and the Cloudflare CDN was 16ms; we approximate this in the emulated topology using a 10ms (one-way) delay between the end user and the CDN. (We do not model the time it takes for the client to find the closest CDN server.) For the link between the CDN and the origin server, we used a 30ms (one-way) delay, approximating an inter-continental link of about 9,000 km.

The function we used for this evaluation is a simple one which calculates a secure one-time user cookie for the end user. The primary computation done by this function is the hash described above. The request made for this experiment is minimal, as is the response from the server.

From the client to the origin, passing through the CDN, the time to retrieve the result averaged 118ms. (This includes the time to set up the TCP connection, send the request, compute the secure cookie, etc.) When the function is moved to the CDN, the time is only 38ms, a reduction of almost 2/3. The exact benefit in practice will, of course depend on how close the CDN is to the end user (the closer, the higher the benefit), how close the origin server is to the CDN (the farther, the greater the benefit), and the amount of time the function takes to execute (long functions may dominate network RTT).

2.7.3 Case Study

To illustrate how Harpocrates can be used to implement one of the applications from Section 2.5, we show snippets of code from our implementation of the cookie-checking DOS protection.

Listing 2.1 gives the general sense of what the untrusted portion of the code for this function does. Lines 1 and 2 set up the SecureWorker package, loading in the binary (`enclave.so`) and Javescript (`cookies.js`) that implement the trusted portion of the cookie checker; recall that the binary run inside of the enclave is the lightweight Duktape JavaScript engine. Lines 4–10 are the function that will receive the completion event from the un-

Listing 2.1. Partial code listing for untrusted portion of cookie checking function

```
var SecureWorker = require('./lib/real.js');
const worker = new SecureWorker('enclave.so', 'cookies.js');

worker.onMessage(function (message) {
    if (message.result) {
        // Forward request to origin
    } else {
        // Drop request
    }
});
// Parse username, nonce, expiration, and cookie from request
worker.postMessage({name: username, n: nonce, exp: expiration, c: cookie});
```

trusted code; we omit the body of this function, which simply forwards the request to the client or drops it depending on the message sent by the trusted code. Line 13 is where the message is posted to the secure worker; here, the username, nonce (to prevent replays), and expiration are available in plaintext in the request, as is the cookie.

Listing 2.2 shows the trusted code that runs in the SGX enclave in response to the `postMessage` from the untrusted code. Line 2 performs the concatenation of the data passed in with the secret In a full implementation, this secret would be obtained (just once) from the origin on enclave initialization via SGX's remote attestation; our prototype does not implement remote attestation. Line 3 hashes this concatenated value, and line 4 checks this concatenation against the cookie sent by the client and sends a completion event to the untrusted code via `postMessage`.

Listing 2.2. Partial code listing for trusted portion of cookie checking function

```
SecureWorker.onMessage(function (message) {
    var hashstring = message.name.concat(message.n,message.exp,SECRET);
    var hashvalue = crypto.subtle.digest({name: 'SHA-256'}, hashstring);
    SecureWorker.postMessage({result: hashvalue == meassage.c});
});
```

2.8 Conclusion

In this chapter, I have presented Harpocrates, a system for moving computation from origin webservers to nodes in a CDN network, which typically reside at or near the network edge. Using SGX enclaves, our system allows the origin webserver to place "master secrets" in these enclaves, giving them strong assurance that these secrets will not be leaked even if the CDN needs to be able to "see" data derived from them in order to do its job. By using JavaScript and an API similar to the WebWorker API, we make it easy for the CDN's customer to move computation from the server to the CDN. We demonstrate that the overheads inherent in this arrangement are not high, and that there is significant benefit to end users in the origin being able to offload secure computations to the CDN servers near them.

Harpocrates is designed for applications with small amounts of trusted code and small master secrets. This is due to the limited enclave size provided by SGX and the costs associated with crossing the enclave boundary. These have performance implications for bigger applications, due to page swapping or the need to build multiple enclaves with secure communication channels between them. We look to other work that has focused on running larger, more intensive applications in SGX [3, 56] for ways to alleviate this limitation in the future.

CHAPTER 3

A YEAR OF AUTOMATED ANOMALY DETECTION IN A DATACENTER

Anomaly detection based on Machine Learning can be a powerful tool for understanding the behavior of large, complex computer systems in the wild. The set of anomalies seen, however, can change over time: as the system evolves, is put to different uses, and encounters different workloads, both its 'typical' behavior and the anomalies that it encounters can change as well. This naturally raises two questions: how effective is automated anomaly detection in this setting, and how much does anomalous behavior change over time?

In this chapter, we examine these question for a dataset taken from a system that manages the lifecycle of servers in datacenters. We look at logs from one year of operation of a datacenter of about 500 servers. Applying state-of-the art techniques for finding anomalous events, we find that there are a 'core' set of anomaly patterns that persist over the entire period studied, but that in to track the evolution of the system, we must re-train the detector periodically. Working with the administrators of this system, we find that, despite these changes in patterns, they still contain actionable insights.

3.1 Introduction

A key task for administrators of large computer facilities is understanding the steady-state operation of their facilities and reacting to any anomalies that might occur. The sequences of events that actually take place in "normal" operation may or may not align with administrators' intuition about the behavior of the facility and its users; having a full understanding is important to effective system administration. Exceptions to normal sequences may indicate problems with the facility's hardware, software, or configuration and may require administrator attention. Such exceptions could also signal new uses or emergent behaviors that administrators should be aware of.

Understanding "normal" and anomalous behavior is not always straightforward. Events from these types of systems are typically collected in logfiles [24], and simply looking for "errors" in these logfiles is not always informative [13]. Some "errors" may be benign: they might correspond to ways in which the facility is used that were not anticipated by software writers, or they may represent transient states that the system is able to recover from itself. On the flip-side, some sequences that are not explicitly flagged in logs as "errors" may be quite worrying, such as increased frequency of certain operations or cessation of others. The examination of logfiles for anomalies and errors is thus a ripe area for machine learning and data mining [14, 24, 50].

In this chapter, we apply the technique of anomaly detection by *invariant mining* [40, 45] to the administration of CloudLab [26], a facility used by thousands of researchers and educators in computer science. CloudLab collects extensive logfiles regarding the provisioning of the servers under its control; as we lay out in more detail in Section 4.3, the dataset used for this chapter covers a year of operation of 583 servers, comprising a total of 15,018,235 log entries. Invariant mining looks at the relationships between frequencies of entries in these logfiles, finding patterns that describe typical operation ("invariants") and log sequences that "violate" those invariants and are thus anomalous. Our goal is to look at the following questions to find whether invariant mining is a useful technique to aid administrators in this setting:

1. Does invariant mining successfully create discriminators capable of distinguishing "normal" behavior from anomalous behavior?

2. Do the invariants found provide information that is interpretable by the administrators of these systems?

3. Do the set of invariants change over time, and if so, how much?

3.2 Related Work

The high risk posed by compromised systems, anomalies, and security threats has led to substantial interest in analyzing system logs to debug system failures and perform root cause analysis.

Moreover, Machine Learning (ML) and data mining techniques have been used widely to monitor large scale systems for the purpose of anomaly detection and system diagnosis. Several statistical and machine-learning models have been proposed to analyze the behavior of systems and detect failures or problem by deeply analyzing systems logs and other sources of data. In this section, we discuss the most closely related efforts in the area of anomaly detection and log file analysis.

3.2.1 Anomaly Detection

There are some generic methods that use system logs for anomaly detection. Typically, this is done by using a log parser to parse the unstructured log entries into a structured form [23] that can be analyzed and modeled by different machine learning techniques. These machine learning techniques are divided into *supervised* and *unsupervised* methods.

For supervised methods, labels are required to complete the analysis and perform anomaly detection. The simplest labels for this use would be "normal" and "anomalous". In practice, however, it is usually hard to obtain labeled data in log files: system logs commonly contain hundreds of thousands to millions of entries, making manual labeling by administrators too time-consuming. Additionally, because anomalies are, by definition, rare, it is not practical to use small subsets of system logs for training, since this would risk including too few, or even zero, anomalies.

Related work in the area of anomaly detection in systems goes back several decades. For example, Bates et al. [7] proposed an event definition language that allow programs to generate logs with deep semantics information, such as hierarchical relationships between events. However, this approach requires access to the source code. Some more recent methods [14] perform anomaly detection on log files without requiring hand-crafted features or pre-processing of data. These work on raw text data and output a score for each log entry, which enables the systems administrator to classify the log entry as either anomalous or normal. Baseman [6] proposes a framework that performs anomaly detection by combining graph analysis, relational learning and kernel density estimation. Moreover, it presents a novel event block detection algorithm that extracts related syslog messages from the log files. The proposed methods analyze individual messages rather than event blocks which limit the application scope. Furthermore, Baseman [5] introduced Interpretable and

Interactive Classifier-Adjusted Density Estimation with Temporal components (iCADET). This framework utilizes random forest classifiers to explain the labeling of certain points as likely anomalous. This technique is more suitable for smaller scale data.

There are also some open source solutions for log files inspection and anomaly detection. For example, Project Scorpio [1] connects to streaming sources and uses unsupervised machine learning methods to generate a prediction of anomalous log entries.

3.2.2 Logfile Analysis

Other research efforts have specifically targeted logfile analysis for anomaly detection. For example, DeepLog [24] proposes a deep neural network model utilizing Long Short-Term Memory (LSTM). This model allows DeepLog to train a model *unsupervised* based on the log pattern and report an anomaly when log patterns deviate from the expected result by the trained model. Several other approaches based in machine learning have been proposed for different systems. Many of these are rule-based approaches, which limits them to specific application and requires domain knowledge. For example, M. Cinque [17], performs a change in the logging mechanism itself, which requires both effort and domain knowledge to implement the change to the logging system first. Other kinds of tool rely on comparing anomalous logs against normal ones, such as [50]. A limitation of such tools such tools is the fact that it is hard to detect new kind of anomalies that the model has not been exposed to before. Because our goal in this work is to study how anomalies change over time, it is important that we be able to find anomalies that were not seen during earlier periods.

Furthermore, some methods were developed to reduce the size of the log files and thus reducing the effort needed for analysis. For example, LSTM, which have been recently used for log analysis purposes in data centers. T. Yang and V. Agrawal. [85] introduced a framework that highlights the messages it deems to be the most important text in the failed log messages, making it less tedious for the human operator or even automated software to analyze the cause behind the failures.

Invariant mining [45] is a general approach that does not rely on the nature of the data or require any significant domain knowledge and unlike rule/keyword based log

[1] `https://github.com/AICoE/log-anomaly-detector`

analysis tools the rules are easier to update when components are upgraded or changed as they usually tend to do. Our work builds on this work, which we give an overview of in Section 3.4.1. Lou et al. [45] applied invariant mining to two case studies, Hadoop and CloudDB (a structured data storage service developed by Microsoft). The testing environment was setup specifically for the purposes of this research. In contrast, our our work uses real-life data from a time span of one year, giving us the opportunity to gain a better understanding of the nature of anomalies and the benefits of using invariant mining to detect anomalies in real datacenter systems.

3.3 Dataset

In this section, we describe the dataset. We talk about CloudLab[2], the facility our logfiles come from. In addition to describing the contents of the logs themselves, we also cover the process we used to prepare the data for analysis. The dataset used for this chapter is available with DOI `https://doi.org/10.5281/zenodo.4073861`.

3.3.1 CloudLab

CloudLab [26] is a facility that serves the computer systems research community. It operates as an environment in which researchers can build their own clouds: it provisions resources at a "bare metal" level, enabling its users to see, control, and modify portions of the cloud software stack including virtualization, networking, and storage. It has approximately 5,00 users around the world who have, to date, run 150,000 experiments on it. CloudLab has three main clusters; the data that we use for this chapter comes from its cluster at the University of Utah [68].

We chose CloudLab for this study because we have access to both its logfiles, which are collected centrally, and its administrators, who can help us interpret our results and evaluate their utility. While CloudLab is a unique facility in terms of the specific features it offers to users, its basic functionality of managing the provisioning of servers, interaction with users via a web interface, etc. has much in common with other facilities and should lead to generalizable results.

In this chapter, we focus on CloudLab's *node booting* process: the automated process of

[2] `https://cloudlab.us/`

booting servers into various operating systems for user experiments, utility tasks (such as re-imaging local hard drives), and general system administration. Though a conceptually simple task, this process involves interactions between firmware and BIOS on the servers themselves, standard network protocols such as DHCP and TFTP, and a number of services that CloudLab runs to track server state [53] and inform servers what their next actions should be. There is substantial emergent complexity in this system and large amounts of parallelism that are difficult to control. As a result, failures to boot are not uncommon, and the CloudLab code includes many measures to detect and automatically recover from common failure modes. Because of the way CloudLab allocates resources, it is common for a server to be part of several experiments in a single day in sequence, and thus to go through this boot process every few hours. It is also common for some servers to be allocated to an individual experiment for long periods of time, meaning that they may not reboot for a period of days or weeks.

3.3.2 Data Collection

CloudLab log data is collected, processed and stored using the ELK (Elasticsearch[3], Logstash[4], Kibana[5]) stack. In our configuration, the ElasticSearch cluster is composed of five data nodes and one client node that also serves the Kibana frontend. As is common with the ELK stack, we have Filebeat[6] aggregate and forward logs from the main CloudLab servers to be processed by Logstash and stored in the ElasticSearch cluster.

The logfiles that we collect come from a mix of standard server software, such as Apache, ISC DHCPD, and `tftpd`; and custom software that has been developed for CloudLab and other related testbeds [31, 79]. Overall, we collect on average 350,000 logs entries per hour (though only a subset of those is used in this analysis.)

---

[3] `https://github.com/elastic/elasticsearch`

[4] `https://github.com/elastic/logstash`

[5] `https://github.com/elastic/kibana`

[6] `https://github.com/elastic/beats`

3.3.3 Parsing and Cleaning

To be used for data mining and machine learning, log messages must be individually identified and parameters, etc. parsed out; the relatively *unstructured* text found in the logfiles must be converted into *structured* data. For invariant mining in particular, each log message must be assigned a corresponding event ID, (also called a *log key*) that indicates the message type. These event IDs are matched to specific patterns, where the pattern represents the constant parts of the message and the variable parameters that the message contains. To get this information, we process each message against a set of Grok [28] patterns. While this log parsing method is sometimes automated [23, 39, 86], our initial attempts to use these automated systems did not produce satisfactory results; thus, we created the Grok patterns by hand to ensure accuracy and to further explore our data. Lists of patterns are consumed by a script to automatically generate a LogStash configuration file to process messages, and we *version* these patterns: each entry in Elasticsearch is tagged with the version number of the pattern set, so that when we add or change patterns, we can re-parse all stored log entries.

An interesting aspect of processing logfiles is that sometimes *mapping* is required between different identifiers for the same entity. One way this manifests in the CloudLab data is that in some logfiles, machines are identified by their "node ID", the primary identifier CloudLab uses to track its resources. In others this information is not available. For example, in DHCP logs, initial requests are identified only by their MAC address. As part of our parsing process in Logstash, we use mapping tables to augment records with *all* identifiers for the node to make it easier to relate entries with each other.

With the data processed and stored using our ELK stack, the data must be collected and formed into data files before applying invariant mining. Data files were created using a script that generated ElasticSearch queries based on selected node type, node range, date range and log types. Each entry from the resulting query had its message and event ID written to an output file along with its assigned session ID. The session ID is formed from the node ID and date to delineate chunks of log entries into *sessions*, where each session represents a 24 hour period for a particular node.

To provide clean datasets, some data had to be excluded because of inconsistencies or errors. As a result of abnormal node ID formats and deformed log messages, some

log entries were not correctly matched with a pattern and any such entry was excluded from generated data sets. Additionally, each message has two timestamps; one from the machine time which contains the message and another assigned by Logbeat at its collection time. In some cases, the two timestamps differed significantly, with Logbeat retrieving the log months after the machine timestamp. Such occurrences had to be excluded from datasets to ensure that the date used to form session IDs were accurate.

3.3.4 Resulting Dataset

For the purposes of this chapter of the dissertation, the dataset was formed from logfiles of all CloudLab nodes of the types `m400` and `m510`, and was gathered from January 1 to December 31 of 2019. The resulting dataset contains over 15 million log entries for those 583 nodes and forms 51,375 sessions.

The dataset we use for this chapter is formed from four specific logfiles, each of which has its own set of message patterns. All of these logfiles record events related to the process of *provisioning* and *booting* nodes. Typically, a reboot of a node is initiated by the CloudLab server in response to a user starting or ending an experiment, though users can reboot nodes themselves either intentionally or as a side effect of a kernel crash on the node.

- `reboot` is a log that contains the system's high-level "intent" with respect to rebooting nodes; that is, when a node is intentionally rebooted, an entry is created in this logfile.

- `stated` reports the status of an internal state machine used in some CloudLab processes [53]. Each state (such as `BOOTING`) has a set of expected successor states (such as `DHCP`, `RELOADING`, etc.) and some states have timeouts associated with them. CloudLab uses this state machine to detect and attempt to recover from certain kinds of faults.

- `bootinfo` is a CloudLab-specific daemon that is used to inform nodes of what they should boot next (eg. boot into a special memory-based filesystem used for re-imaging, boot from a partition on the disk, etc.) The first-stage bootloader contacts this service, so it provides information that a node has reached a certain point in the boot process and gives context regarding what the node is booting.

- `dhcpd` records DHCP events from the server's perspective. Because nodes contact the DHCP sever at multiple points during the boot process (from the PXE ROM, OS initialization, etc.), this provides fairly fine-grained information regarding nodes' progress through the boot process.

To parse these log files, we used 48 unique log patterns with `bootinfo` and `stated` having the most unique patterns, with 25 and 15 respectively.

3.4 Analysis Methodology

We took the dataset described in Section 4.3 and applied invariant mining [45] to find what constitutes "normal" behavior for the CloudLab provisioning process, and to find deviations from this normal. In addition to mining invariants for specific time periods, we also develop a method for examining how they change over time so that we can understand if the steady-state behavior of the facility changes or not.

As mentioned before, manual inspection of log files is infeasible due to the system's large scale and high complexity. Moreover, the software that manages this system is updated frequently, which makes it impractical to rely on rule-based log analysis solutions. Since invariant mining does not utilize constant rules, does not require labels for training, and does not depend on the domain knowledge of system admins, it is more appropriate for use with regularly-revised, large-scale systems.

3.4.1 Invariant Mining

The idea behind invariant mining [45] is that what we consider to be normal behavior can be learned by mining the log files to discover the inherent linear characteristics of the program workflow. Any log entry that does not match the workflow will be considered anomalous. This method can be used to automatically define rules for anomalies and thus automatically detect them. The linear invariants reflect the properties of execution path and so a violation of an invariant can often reflect the physical meaning of the system problem which makes it a superior diagnostic tool for human operators.

The input that we provide to the invariant miner is a set of sessions (described in Section 3.3.3), with each session containing a count of how many times each log key occurred during the session. The miner looks for sets of keys that typically occur with

linear relations and outputs these ratios. For example, the miner might discover that each message indicating that a server has begun rebooting is typically paired with a message indicating a successful boot. Or, it might find that a message indicating that a server has begun PXE booting typically matches with two DHCP requests: one from the PXE ROM, and another from the OS once the server has booted into the OS.

Each ratio is called an *invariant*, and log sessions that do not follow this relation are said to *violate* the invariant; sessions that contain violations are said to be *anomalous*. Once we have this set of invariants, finding anomalies is straightforward: to check an individual session, we simply count occurrences of log keys and check whether they violate any invariants.

The invariant miner has a simple data model in that it just looks for integer ratios in the counts of event occurrences. Advantages of this approach include the fact that it does not require the semantic information that would be required to truly match up specific events, and that these ratios stay the same (under normal conditions) no matter how many boot cycles are observed in a given window. What it gives up in return is that while it can identify the presence of an anomaly, the invariant miner but itself cannot point to specific log messages that caused this anomaly. For example, if the anomaly detector finds that there are more "started booting" messages in a session than "successfully booted" ones (violating the expected one-to-to ratio), it can flag an anomaly, but finding *which* boot attempt failed requires additional processing or manual inspection.

For this chapter, half of the data set is used as a training dataset and the other half is used as a test dataset.

3.4.2 Comparison Across Time

To analyze change over time, we divided the data from year 2019 into four quarters (January–March, April–June, etc.) and trained the invariant miner with each quarter's data independently. We then compared the *sets* of invariants found in each quarter. These results were used to study how usable the invariants are (that is, whether administrators found them accurate and actionable), which invariants are persistent over the year, etc. We also compare the number of invariants violated in each quarter and analyze the reasons behind the difference in invariants violations between the quarters. Since the persistent

violations occur in different time periods through the year, we highlight them as the most persistent violations.

3.4.3 Implementation

Our implementation is based on `loglizer` [40] by the Logpai team[7]. The original tool's invariant mining output was not sufficient by itself to list all invariants and sessions that violated them, so modified the source code to produce the needed information. This information included the mapping between the numbering of event types and the actual Event IDs in our log files. It also included the number of violations for each invariant, which sessions violated which invariants and which sessions are completely "clean."

We also wrote a post processing program to map the arbitrary session numbering used internally by `loglizer` to the original session IDs from the log files. It also maps the event in the invariants to its original text format to make it easier for human interpretation.

We programmed our data parser to output log files in the same format needed by the invariant miner. The most important feature in the obtained log files is to group log events according to their types. These logs groups are formed into session which contain all log entries for a particular server in a single day. The invariant miner counts the number of occurrences for each type, this count is used to find the ratio for occurrences for multiple event types and thus finding the needed invariants.

3.5 Findings

We now return to the main motivating questions for this chapter of the dissertation: Is invariant mining accurate at finding actual anomalies in this dataset? Are the invariants it finds meaningful to administrators? Are the set of anomalies fairly constant over time, or do they vary? We start by looking at the invariants themselves.

3.5.1 Invariants Found

In the invariant mining process, log messages are grouped together according to a set of parameters that correspond to the same event type. The invariant miner then utilizes the event types and their frequencies to produce invariants such as the following:

---

[7] `https://github.com/logpai/loglizer`

```
(11, 29): [ 1.0, -1.0]
(17, 18): [ 1.0, -1.0]
(1,  65): [-4.0,  1.0]
```

The first invariant corresponds to two event types, 11 and 29, and the ratio of their occurrences in a normal session is $1 : 1$.[8] When this ratio is not satisfied the invariant is considered violated and an anomaly is reported. The second invariant shows another pair of event types that appear in a one-to-one ratio in normal sessions, and the third reports a one-to-four ratio.

Mapping the event IDs to actual log messages, a set of example invariants are shown in Figure 3.1. Each shows a pair of log lines that are expected to appear in a one-to-one ratio in a "normal" session.

In our dataset, the miner found an invariant space dimension of 16 in the first quarter, meaning that it found 16 unique invariants. For the second quarter of the year, the invariant miner produced 17 different invariants. For the third quarter, the result was 13 invariants. And for the last quarter of the year, the result was 19 different invariants. Table 3.1 shows the total number of sessions and percentage of anomalies for each quarter in both the training data set and test data set.

[8] One part of the ratio is always shown as negative, as the miner is solving equations of the form $a \cdot x + b \cdot y = 0$. Which part is positive and which is negative is arbitrary.

```
Invariant 1: Ratio: ['1.0', ' -1.0']
 <DATE> <TIME> <NODE_ID>: in PXEWAIT, sending PXEWAKEUP
 <DATE> <TIME> boss bootinfo[<PID>]: <IP>: SEND: query bootinfo
Invariant 2: Ratio ['1.0', ' -1.0']
 <DATE> <TIME> [<PID>]: <NODE_ID>: RESET done, bootwhat returns NORMALv2
 <DATE> <TIME> [<PID>]: <NODE_ID>: Clearing reload info
Invariant 3: Ratio: ['-1.0', ' 1.0']
 <DATE> <TIME> <NODE_ID>: ssh reboot returned 255
 <DATE> <TIME> <NODE_ID>: waiting 30s for reboot
```

Figure 3.1. Example invariants found by the invariant miner.

Dataset	Jan-Mar		Apr-Jun		Jul-Sep		Oct-Dec	
	sessions	anomalies	sessions	anomalies	sessions	anomalies	sessions	anomalies
Training dataset	5220	3.8%	5713	4.4%	6154	2.7%	8600	4.7%
Test dataset	5220	3.1%	5713	5.0%	6154	1.9%	8601	4.8%

Table 3.1. Number of sessions and percentage of anomalies found per quarter in our dataset.

3.5.2 Usefulness and Interpretability

We found that while some invariants were "useful", not all were. Here, we define "useful" by three criterias.

First, they must be *non-trivial* in the sense that it is *possible* to violate them. In some cases, the invariant miner found two types of log entries that are produced by the same function in the same program. In this case, it is nearly impossible to see one message without the other: the program would have to hang or crash within a few lines of code. No violations of this type were found in this dataset. We found six distinct invariants of this type.

They are easy to identify, because they are never violated, and do not affect the accuracy of the anomaly detection.

Second, an invariant must be *sensible*. We evaluate this by looking at the expected ratio produced by the miner. While most invariants have ratios such as $1:1$ or $2:1$ that would be expected from a system of this type, the miner found some "invariants" with ratios as high as $311,785:1$. The highest-ratio event that, by manual inspection, appeared sensible was $1:7$. This corresponds to the number of times that one of the processes will retry apparent failures before giving up. Over the full year, the miner found 14 "invariants" with ratios of $15:1$ or higher. We find it highly likely that "violations" of these represent false identification of anomalies. Fortunately, they are easy to filter out, since there is a large gap between the largest "sensible" ratio ($7:1$) and the smallest "insensible" ratio ($15:1$); we can simply filter out invariants with ratios above $10:1$. We speculate that these false invariants were found due to a few highly-anomalous nodes that had behavior that persisted over multiple sessions. For example, one node was stuck in a "boot loop" for months, unnoticed by the operators. This resulted in many thousands of spurious DHCP messages intermingled with a few messages of other types. These sessions tended to be

flagged as anomalous due to violating other invariants.

Third, invariants must be *interpretable*, meaning that administrators are able to understand, in a general sense, what the reasons behind a violation are or what the consequences of it might be. This is a much harder criterion to evaluate quantitatively, so we examine it qualitatively. The pattern that we find among the most interpretable invariants is that they follow one or more of the following properties:

- They involve entries that appear in more than one logfile. In other words, to detect the invariant, it is necessary to correlate information across logfiles. This provides significant value to administrators, who tend to inspect a single log file at a time.
- There is a clear way to match events. That is, it is possible, either through timing or unique identifiers, to confirm that one event is in some way a response to or consequence of the other. Note that the invariant-based anomaly detector does not produce such a matching itself, but this can generally be done by additional processing or manual inspection.
- They are asynchronous operations: an operation on a node is started by one process; the node performs some actions that are not directly in the logfiles, may take a variable amount of time, and may fail; and the success or failure of those actions are observed by a different process.

An example of invariant that meets all of these criteria would be one that relates a log message indicating that a node is to be rebooted with one that logs a successful DHCP response to the node later, after the node has shut down, made it through BIOS, and the NIC's boot ROM, etc. We found five invariants that we deemed highly interpretable by these criteria: some of these are discussed in more detail in the following subsections.

3.5.3 Accuracy of Anomaly Detection

In order to assess the accuracy of anomaly detection using invariant mining, we compared the labels produced by the invariant-based detector with labels assigned by humans. To do this without requiring undue operator effort, we ran the invariant miner on the dataset described in Section 4.3 and labeled sessions according to the invariants found. We then created five sets, each containing ten sessions that the invariants had labeled as

"normal" and ten that it had labeled as "anomalous". After correcting for a few invariants with "insensible" ratios, the base rate of "normal" sessions in the dataset we gave to administrators was 56%. Each set was given to a different administrator of the CloudLab testbed, who was asked to label each session. The administrators were told that their set contained a mix of normal and anomalous sessions, but were not told how many there were of each, and were not given a definition of "anomalous"; the intention behind this methodology was to see how well the precise ratios found by the invariant mining process match up with human intuition.

In our evaluation, we consider a "normal" label as negative result and an anomalous label as positive results. Therefore, a false negative refers to an incorrect labeling by the classifier for a truly positive results and a false positive refers to an incorrect labeling by the classifier for a truly negative result. The accuracy of the invariant miner was reasonable: it correctly labeled 70% of sessions identified by the administrators as normal, and 73% of the sessions labeled as anomalous. This gives us an overall false positive rate of 30% and false negative rate of 27%. The overall precision obtained is 0.7087, recall is 0.7300 and the F1-score is 0.7192.

There were two other interesting findings from this portion of our study. First, we found that the administrators made a distinction between behavior that indicated a problem with the system and unusual user behavior. One example of this in our context is that most boot sequences are initiated by the system in response to higher-level user requests, such as the start or termination of experiments. These kinds of sessions have telltale log entries indicating the start of the process. If users shut down or reboot machines themselves (eg. by running `shutdown` on the machine itself), these telltale log entries are absent, and there may be log entries indicating an unexpected shutdown. Most administrators independently came up with this third label, which, in terms of frequency, is anomalous, but is likely to not require administrator intervention. For the purposes of the calculations above, we considered these to be anomalous, but it suggests the potential for future work to distinguish known-benign classes of anomalies from those that might require intervention.

Our second finding was that the sessions that were mis-labeled by invariants tended to fit very specific patterns. One of the biggest discrepancies was a single server that exhibited

the same anomalous behavior (according to the administrator) over three different days, but was labeled as normal according to the invariants. Nearly all the rest of false positives were caused by a single set of three related invariants, that caused sessions identified as normal by the administrators to be flagged as anomalous by the detector. This seems to suggest that relatively simple heuristics could be used to greatly improve the accuracy rates, and suggests an avenue for future work.

3.5.4 Evolution of Invariants Over Time

First, we compared the invariants through the four quarters of the year 2019 using the number of unique invariants in the current quarter compared to the previous quarter and next quarter. We also used the number of shared invariants between quarters as a measure for the evolution of invariants. In our study of the invariants, we focus more on the persistent invariants through quarters as they are the most meaningful invariants.

Figure 3.2 shows the comparison between the number of invariants obtained through the year. From quarter to quarter, we see that approximately half of the invariants change; that is, the number of invariants that each quarter has in common with its neighboring quarters is about half of the total number of invariants for the quarter. This points out the need to periodically re-train the anomaly detector.

When comparing the invariants across all quarters, we find that we have 6 core invariants that persist through the year. This means that in most cases, when Figure 3.2 shows invariants in common with the previous and/or next quarters, it is referring to this set. As with most useful invariants that we find, these all occur with ratio $1 : 1$. Two of these invariants have to do with using `ssh` to "gracefully" reboot nodes. (One of these can be seen as Invariant 3 in Figure 3.1.) These two often cause false positives; they are two of the three associated with false positives (as determined by human administrators) above. Another pair (such as Invariant 2 from Figure 3.1) have to do with CloudLab's disk imaging process: they show the state transitions that are supposed to occur when the imaging process finishes and the node boots into its new image. The fifth invariant is a trivial one as defined in Section 3.5.2: it documents a node requesting information from the `bootinfo` process and the reply that is sent out.

The sixth (Invariant 1 from Figure 3.1) is the most interesting: it contains one message

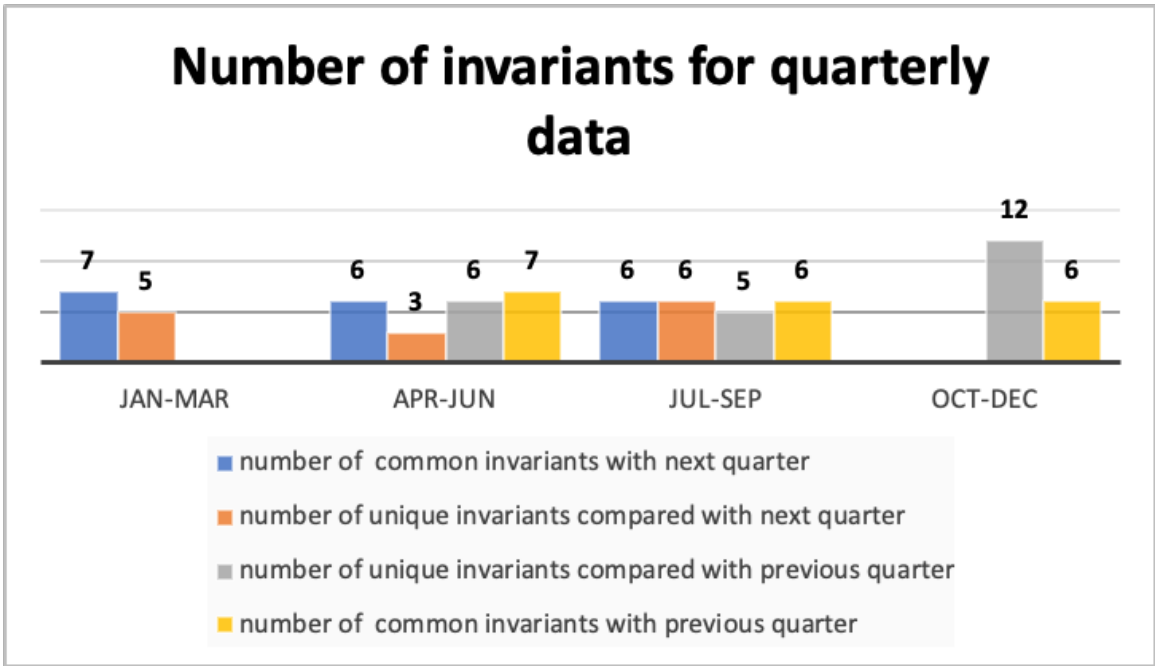

Figure 3.2. Comparison of number of invariants throughout one year. Note that the first quarter has no preceding quarter, and the last has no succeeding one.

from the `reboot` log, and another from `bootinfo`. The former indicates that the CloudLab software has decided to reboot a node, and the latter indicates that the node has reached an important point, several steps into the boot process. This is interesting not only because it crosses multiple log files, but because the point identified comes *after* several other log messages (such as ones from `dhcpd`) would normally be seen. This strongly suggests two things: (1) Nodes that CloudLab decides to reboot do normally come up, which is expected (2) The lack of earlier invariants from the `dhcpd` log suggests that it is *not* terribly uncommon for nodes to fail early in the boot process and require power cycling by the CloudLab control software. If the sequence "reboot, DHCP, PXE, bootinfo" (the 'normal' boot sequence) were dominant, we would expect to see invariants for the "reboot, DHCP" part of the sequence. This lack suggests that CloudLab not infrequently times out waiting for the DHCP message and power cycles the node. The fact that the "reboot, bootinfo" invariant *does* exist implies that when this power cycle occurs, the node does eventually reach the "bootinfo" stage, suggesting that these kinds of failures are transient. This understanding meshes well with our findings in Section 3.5.3, in which we found that

there are a significant number of abnormal occurrences that are dealt with automatically by CloudLab and do not require operator intervention.

3.6 Conclusion

We find that invariant mining is fairly accurate on our real-world dataset, agreeing with the "anomaly" labels assigned by system administrators more than 70% of the time. The patterns of these inaccuracies suggest that simple heuristics, or occasional manual pruning of invariants, may substantially improve accuracy. Applying such heuristics is a topic for future work. We found that, in contrast to the binary nature of classification performed by most anomaly detection, administrators naturally and independently arrived at a *trinary* system of classification. This classification system subdivides anomalous events into those that require human attention, and those that, while rare, are in some sense "expected" and do not require additional attention. Building this distinction into anomaly detection is likely to be another fruitful avenue for future work.

We also found that anomaly rates vary substantially between quarters (from 1.9% to 5%), and that the set of invariants that describes these anomalies varies as well. This points to the need to periodically re-train anomaly detectors, as they become stale over time.

CHAPTER 4

DEEP-SEC: FINDING MALICIOUS NEEDLES IN LOGFILE HAYSTACKS

System administrators must deal with a wide range of behaviors on the systems that they operate, including steady-state "normal" behavior, traffic spikes, anomalies due to bugs, etc. One of the most important classes of such behavior is *malicious* behavior that represents attempts to compromise the security of the facility. Such behaviors require special attention, because they may require swift action to block the perpetrators, look for evidence of successful compromise, and potentially clean-up if an attack succeeded. For this reason, it is useful to be able to distinguish malicious behavior from other types when examining system logfiles.

Previous work has used a variety of *anomaly detection* techniques to find usual behavior in system logs with machine learning. This work, however, has generally not distinguished *malicious* behavior from other types of anomalous behavior: a common assumption, for example, is that all anomalous behavior is malicious. In this chapter, we present Deep-Sec, an anomaly detection tool that takes a more nuanced view of ML-driven anomaly detection. Using a novel fine-grained measure of anomalous traffic, we train a classifier to distinguish between *benign* and *malicious* behaviors. We apply this technique to a large dataset from logs on a production webserver, and show that our technique is very effective at distinguishing between these classes of anomalous behavior. We also show that this method is complementary to other methods for detecting vulnerabilities: when compared to professional scans performed on the same system, Deep-Sec found a substantially different set of URLs probed by malicious requests.

Computer security has a large impact on how systems are administered, especially with evolving attacks and the constant discovery of new systems vulnerabilities. As systems become larger and more complex, they become harder to secure.

System logs are a rich information source for system debugging, anomaly detection, and identification of threats. They record the system state from different aspects such as booting information, network traffic, and security-related information. However, as systems grow and see more activity, it becomes less and less feasible to find anomalies through manual inspection of system logs. It also becomes harder to distinguish anomalies that represent malicious activity from other types of anomalies.

One promising method for finding security incidents—and other out-of-the-ordinary events—from logfiles is by using learning techniques to identify *anomalies* in them [27, 45, 49, 66]. There has been extensive research in the area of anomaly detection in general, and more specifically, in the area of detecting suspicious or malicious activity using system logs. Most of the existing anomaly detection approaches for solving the problem of detecting malicious actions either considers any anomalous behavior found as malicious without looking deeper at the nature of this abnormality (such as in Gao et al. [33]), or performs binary classification of the data to anomalous and normal without dividing the anomalous sessions into multiple classes (such as in Du et al. [24]).

We go beyond this binary classification system in our design of Deep-Sec, an anomaly-detection system that further subdivides anomalies into *benign* and *malicious* sets. This enables administrators to more directly and effectively deal with security incidents, which are in fact a fairly small subset of all anomalies. Deep-Sec leverages an LSTM network to model a system log as a natural language sequence. The result of this model is used to detect anomalies in the system and specifically detect malicious sessions based on a novel identifier called the *anomaly-score*. The anomaly-score value is determined according to the dataset using a clustering algorithm. Any session that attains a score equal to or higher than this anomaly-score is considered malicious.

In this work, we build a security anomaly detection system and perform extensive evaluation and testing on the log files of a scientific research testbed called Cloudlab [69]. Deep-Sec can use multiple logs types such as the web-data logs, and does not require a specialized security-related logging system. Building on DeepLog [24], Deep-Sec utilizes LSTM to memorize long-term dependencies over sequences of events.

The contributions made by this dissertation in this chapter are:

- We introduce a three-class system for logs in which anomalies are further subdivided

into *benign* and *malicious* categories. This helps system administrators, who typically must take different investigative and remediation actions for malicious activities.

- We present the design of Deep-Sec, an anomaly detection system that improves on DeepLog by introducing fine-grained scoring system that (a) enables us to distinguish between benign and malicious anomalies, and (b) improves accuracy on long sessions driven by human behavior rather than program structure.

- We have collected, and manually labeled a dataset for ML research that makes the benign/malicious distinction. This dataset is from a real datacenter environment, and was collected over a time span of 4 months. To our knowledge, this is the first such publicly-available dataset that specifically subdivides anomalies into benign and malicious categories.

- We demonstrate that Deep-Sec is complementary to other methods of detecting vulnerabilities by showing that it uncovers attempts by adversaries to scan for potential vulnerabilities that are quite different from those probed by professional vulnerability scans.

We begin in Section 4.1 by going into more detail regarding our motivation for developing Deep-Se, and cover related work in Section 4.2. The dataset we have collected and labeled is described in Section 4.3, and we describe how it is processed and how we extract features from it. Section 4.4 describes Deep-Sec's design, including the full end-to-end pipeline from data collection to label prediction and Deep-Sec's feedback system. We evaluate Deep-Sec in Section 4.5 in two ways: quantitatively (accuracy rate) and qualitatively (types of malicious behavior found, etc.) Section 4.6 concludes, including thoughts on future work and the availability of the system and dataset.

4.1 Motivation

From a system administrator's point of view, prioritizing among all anomalous session is challenging: some may need immediate attention, and others are less dangerous. For example, in a large institution, security incidents might be assigned to be handled by a different team. Thus, it is very valuable to have the ability to assign priority labels to anomalous sessions based on their possible imposed threat to the system.

While some sessions are truly abnormal, not all abnormalities are malicious. For example, web crawlers are considered anomalous since they don't represent genuine user behavior. Another example of a benign anomalous session is a user that can behave in a way that does not resemble typical user behavior, such as leaving a web page open for days or weeks. Some of the other benign anomalies could be caused by a bug in the website itself that breaks the expected workflow. All of these behaviors are indeed anomalous but should not be considered malicious since they are non-threatening. Hence, there is a need for an anomaly detection tool that can further subdivide the anomalies and flag the most likely malicious sessions. To the best of our knowledge, all existing anomaly detection tools either assume all anomalous sessions are malicious or rely solely on security-related logs to detect malicious sessions.

Since false positives and false negatives directly and dramatically impact the effectiveness of any security measures, it is critical that system administrators understand the false positive rates and false negatives rates of each security system they rely on. In the case of deploying any anomaly detection system that considers all abnormalities benign, the danger arises from the false negatives since system administrators will overlook all the malicious sessions and consider them non-threatening. Conversely, in security systems that considers all anomalous session malicious, system administrators will have to handle each session as a true positive and waste their resources studying these sessions to differentiate between the false and true positives.

The impact of false positives and false negatives on the reliability of security systems is extensively discussed in research. For example, Vassilev et al. [22] present a theoretical model, algorithms, and quantitative assessment of the impact of false positives and false negatives on the security risks during transaction processing.

Anomaly detection tools that target malicious sessions can produce false negatives (some attacks are not being reported) or false positives (label a benign anomaly caused by something like a unique user behavior as malicious). False positives cause immediate problems to system administrators who have to sift through them and try to differentiate between the true and false positives, but false negatives are much more dangerous because they lead to a false sense of security. Deep-Sec aims to minimize the false positives by utilizing an automated feedback loop using the initial result of Deep-Sec itself without the

need for human expert intervention.

Moreover, it can be very valuable for organizations to perform post-mortem analysis in case they want to analyze the status of their system security. Deep-Sec can be specifically useful for this because it does not rely on intrusion detection systems logs or any security-specific logging system. Rather, it can utilize the basic logs produced by the system.

Deep-Sec is a computer security tool that can be used with or without IDS. It studies the behavior of systems and tries to construct a pattern of expected normal behavior. Thus it can detect security incidents that have not been seen before or specifically tailored to attack a specific machine as long as they deviate from the usual patterns considered normal.

The primary dataset used to evaluate Deep-Sec is a web dataset from Cloudlab. Each session in this dataset represents a week's worth of Apache logs for a single IP or user. These sessions are much longer compared to the HDFS dataset used to evaluate DeepLog [24] and many other recent anomaly detection systems, making the "one surprising entry makes the whole thing anomalous" rule used by Deeplog less applicable. The sessions represent human activity instead of just the workflow of a code. This means it is much harder to define what is considered a normal activity and how to define anomalies, but since there is a structure to any website we believe that there is also structure for the expected human behavior in this website even if the range of "normal" behavior is much wider. We hypothesize that the more surprising the session, the more likely it is malicious since it highly deviates from the normal range of behavior. In Section 4.5, we demonstrate that using the Cloudlab dataset, we have found that almost all of sessions that are more than 80% anomalous are indeed malicious, proving our hypothesis holds true.

Deep-Sec can be used with the logs from the common logging mechanism of the system and does not rely on security-related logs to detect malicious sessions. This makes the deployment of Deep-sec less tedious.

4.2 Related Work

Broadly speaking, the techniques used for anomaly detection depend mostly on two factors. One is the type of input data, which means that the instances of the input data possess some sequence, for example, natural language, text, etc. The second is the availability of labels for instances of data. Based on the data labels, there are three techniques

used: supervised, unsupervised and semi-supervised.

4.2.1 Supervised

Anomaly detection through supervised techniques involves training classifiers with normal and abnormal labels [32, 46]. The availability of labeled training data with different classes (ie. normal an anomalous) and class imbalance (ie. the fraction of the data that is anomalous) are the two factors that impact the popularity and performance of these models. Patterns in our webserver logs possess a high degree of variation as they are highly coupled with user behavior. This makes detecting new normal or abnormal patterns challenging as the classifier is trained on pre-existing normal and abnormal instances.

4.2.2 Unsupervised

Unsupervised techniques are used when it is difficult to label data instances as normal or abnormal. The major setback for unsupervised techniques is that they have a high false-positive rate and suffer from low detection rates. The most common traditional methods are principal component analysis (PCA) [80], support vector machines (SVM) [21] and Isolation Forests [44]. There are many recent unsupervised models like recurrent neural network (RNN) [58], generalized denoising autoencoders [75], and deep belief networks [60]. These methods have better performance as compared to the traditional methods but the high variation in patterns in websites and absence of labels yield inefficient results in terms of detection.

4.2.3 Semi-Supervised

Semi-supervised techniques allow training with only normal or abnormal instances. The instances that do not exhibit the same behavior as the training instances are considered outliers or false cases. Usually, it is easier to recognize and mark at least one label for all the instances in data, which may be normal or abnormal depending upon the data. This is one of the key reasons that this technique is widely used [49, 66]. There are semi-supervised techniques based on Autoencoders [27, 30], Deep Belief Networks [84], Generative Adversarial Network [38, 59], Convolution neural networks [57]. In webserver logs it is not possible to label and train on all the abnormal instances. Hence it is best suited to train the model with normal instances and detect instances which don't exhibit this behaviour.

4.2.4 Anomaly Detection on Logfiles

In terms of input data, it may exhibit sequential patterns or not. Based on this, different techniques can be applied. As our website logfiles involve the human aspect, it is widely distributed, but it exhibits some sequential pattern on the series of websites/webpages visits by a user. The key idea is that anomalies will not exhibit the same sequential behavior as genuine users. After exploring different machine learning methods for anomaly detection such as log clustering [43] which groups and organizes logs to ease log-based problem identification, invariant mining [45], and isolation forests [44] which is a model-based method that explicitly isolates anomalies rather than profiles normal instances, we found that LSTM or more specifically DeepLog [24] is the most suitable tool on which Deep-Sec can be built. Wu et al. [83] found that the LSTM-based model has superior performance when compared to other models such as the Hidden Markov model (HMM-based models). In addition, DeepLog can detect anomalies at the log-entry level rather than at the session-level like most other methods and can support online training to the LSTM model by incorporating user feedback.

The anomaly-based approach has been deployed in the field of system security and intrusion detection system. It examines the behavior of the network and finds patterns, automatically creates a data-driven model for profiling the normal behavior, and thus detects deviations in the case of any anomalies. For example, Sarke et al. [61] present an intrusion detection tree (IntruDTree) machine-learning-based security model. Liu et al. [42] presented an approach to classify the predefined attack categories such as DoS, Probe or Scan, U2R, R2L, as well as normal traffic utilizing the popular KDD'99 Cup [72] dataset by using a hyperplane-based support vector machine classifier. Sommer and Paxson [65] discuss the imbalance between the extensive amount of research conducted in the field of machine-learning-based anomaly detection for intrusion detection systems versus actual deployment of such systems in the industry. They conclude that this imbalance is caused by some domain-specific challenges that make it significantly harder to deploy it effectively. Siddiqui et al. [64] use an Isolation Forest [44] to detect cyber-security attacks on a particular network using anomaly detection, then generate an explanation of why a specific user or computer was identified as anomalous. This explanation is given to the system administrators to collect feedback on whether the identified anomalies were true

or false positives. The model is then updated according to the administrators' feedback to improve its detection capabilities further and minimize the false positives rate.

There is extensive research in anomaly detection and computer security using deep learning and machine learning methods. Most of the work in the area of anomaly detection in large-scale log files performs *binary* classification on data and classifies log entries to either anomalous or normal, without further considering the nature of the abnormalities. A common assumption [33] is that all anomalies detected are malicious. This approach has many shortcomings because we have multiple classes of anomalies in most of the datasets. For example, some of the anomalies in our dataset consist of legitimate bugs in the Cloudlab website or web crawlers and not actual malicious behaviors. While administrators need to be aware of all of these anomalies, the type of response required is quite different between anomalies that are malicious and those that are not.

Deep-Sec differs from previous work in that it seeks to distinguish anomalies that represent malicious behavior from those that are benign. Specifically, we build on Deeplog [24] and expand it from binary classification to multi-class classification based on LSTM predictions. Our classifier can tell with reasonable certainty whether a session is anomalous or not. And if it is anomalous, whether it is malicious or benign. Moreover, our prototype can be easily deployed in most systems with a logging mechanism because it does not rely on intrusion detection systems logs, firewall logs, or any unique security-related logs to train the classifier. Instead, we utilize regular system logs such as the Apache log files from Cloudlab.

4.3 Cloudlab Dataset

Our dataset was gathered on Cloudlab over a period of four months. To the best of our knowledge, it is unique: unlike existing public datasets, the data is not divided into normal and anomalous sessions only, but we have manually labeled the sessions as normal, benign, and malicious. This allows us to test the capabilities of Deep-Sec.

4.3.1 Webserver Logfiles

Our dataset comes from the Apache webserver log files from Cloudlab. They consists of a total of 127 million entries. After preprocessing of data we got 1,317 different source IP

addresses in the log, and 24,296 different webpages accessed by these users. Cloudlab has two sections of its website—some pages accessible to the public, and others only accessible to logged-in users. Our dataset covers both.

4.3.2 Forming Sessions

We parse the logs to extract certain fields. In Deep-sec, we use source IP addresses and their respected visited web address as our features. We use the *path* portion of the URL and discard the *query* portion—thus, visits to the same page, but with different parameters (eg. viewing different users, objects, etc.) are considered to be the same URL. An IP address is termed as a "user," and set of URLs visited by it is termed as an event. Usually, a session has many events. Some of the sessions have an enormous number of event IDs (hundreds of thousands), and it consumes a huge amount of time and memory for training due to the relatively smaller sliding window size. To reduce the training time and increase the efficiency of Deep-sec we divided sessions on weekly basis where each session consists of all the events encountered in the particular week in chronological order. We omit short sessions (less than ten entries) as they are unsuitable for behavior analysis. Dividing all the sessions on weekly basis and omitting short sessions produced 3,118 different sessions.

Our goal is to classify these sessions as normal, benign or malicious sessions based on the sequence of the events. Normal sessions consist of events that exhibit a genuine user, such as a logged-in user of the facility or a person visiting a set of the public-facing web pages. Typically, after fetching a page, normal users will fetch the associated assets (scripts, stylesheets, images, etc.) of the web page—though this behavior is affected by client-side caching of those objects as well. Benign sessions exhibit a slight deviation with respect to the normal sessions, the majority of these sessions are crawlers or misspelled web address. Malicious sessions consist of events that are intended to discover, exploit, or take advantage of a vulnerability forcefully.

4.3.3 Feature Extraction

Our implementation uses sequences of log keys as input. We process logfiles to assign an index to each unique URL. Every line in the input file will represent a session, and each entry in that session represents an event. An annotated example of a session is show in Figure 4.1.

```
7     : /
15265 : /wp-includes/js/jquery/jquery.js
15266 : /administrator/help/en-GB/toc.json
15267 : /administrator/language/en-GB/install.xml
15268 : /plugins/system/debug/debug.xml
1370  : /administrator/
15269 : /misc/ajax.js
7     : /
15270 : /admin/view/javascript/common.js
15271 : /admin/includes/general.js
15272 : /images/editor/separator.gif
15273 : /js/header-rollup-554.js
15274 : /vendor/phpunit/phpunit/build.xml
50    : /.env
7     : /
```

Figure 4.1. Annotated session—only integer IDs are included in the data used for training and testing, URLs are added here for readability.

The sample log entry shown in the figure shows 15 events, and each entry in the figure represents an event occurring in this session. The integer is an event ID (also called a "log key" in some anomaly detection literature) that uniquely identifies the URL. As can be seen, if a URL is seen multiple times, it is assigned the same ID each time; this is true both within an individual session and across all sessions.

Since our dataset consists of URLs that represents specific users' activities, in the data processing phase we decided to omit all URL parameters that tie the URL to a certain user. The idea is to make the URLs generic enough to construct patterns across different users.

4.3.4 Splitting the Dataset

After manually labeling the dataset, we split the data into a training dataset and a test dataset. The training dataset consists of normal sessions that exhibit normal behavior. It is important to point out that if is a certain behavior pattern or event is considered normal but is not included in the training dataset, Deep-Sec will consider it anomalous as it hasn't seen such event in any pattern or variation of the data. We use two different feedback mechanisms, described in Section 4.4, to correct for such omissions in the original training set.

We train Deep-sec in a semi-supervised manner. We have labeled the normal sessions in our data and used these sessions for the training.

4.4 Design

Deep-Sec is designed to detect anomalies in log files and flag the security-related incidents and malicious sessions. The key intuition for Deep-Sec is that system logs are usually constructed by programs or human behavior workflows and thus we can expect them to have patterns that can be predicted. Our hypothesis is that malicious sessions are far less predictable compared to benign anomalous sessions and so using a non-binary classifier i.e. Deep-Sec we can further subdivide the anomalous sessions and flag the malicious data. Deep-Sec is automatically updated through a feedback loop that can reduce the false positives rate in future runs.

Figure 4.2 explains the workflow of Deep-Sec, beginning with the raw web logs, structuring them into sessions, and performing the necessary data processing and feature extraction as described in the previous section.

4.4.1 LSTM Training

After processing our dataset into a sequence of log keys, we train an LSTM network on a subset of the dataset that only consists of normal sessions. The goal is to predict the probability of having $k_i \in K$ as the next key after sequence $S = \{k_0, k_1, \ldots k_{i-1}\}$. In other words, for a sequence of log keys S, we train the LSTM to return a set of predicted next keys along with probabilities for each key. The key actually seen after S in the training set should be associated with a high probabilities—the loss function penalizes the network for mis-predicting the next key.

We use a sliding window to produce sequences S from longer sessions, so for a session of length n, there are $n - |S|$ windows. Our current implementation uses $|S| = 20$, which we find to be effective for web logs: each window needs to be long enough to capture a set of repeated dependent actions, such as loading a web page and then loading the assets (CSS, javascript, images, etc.) that it references. This is a longer window than used in previous work, and is a hyperparameter that depends on the system being analyzed.

This step the same as in DeepLog [24], which automatically learns a model of log

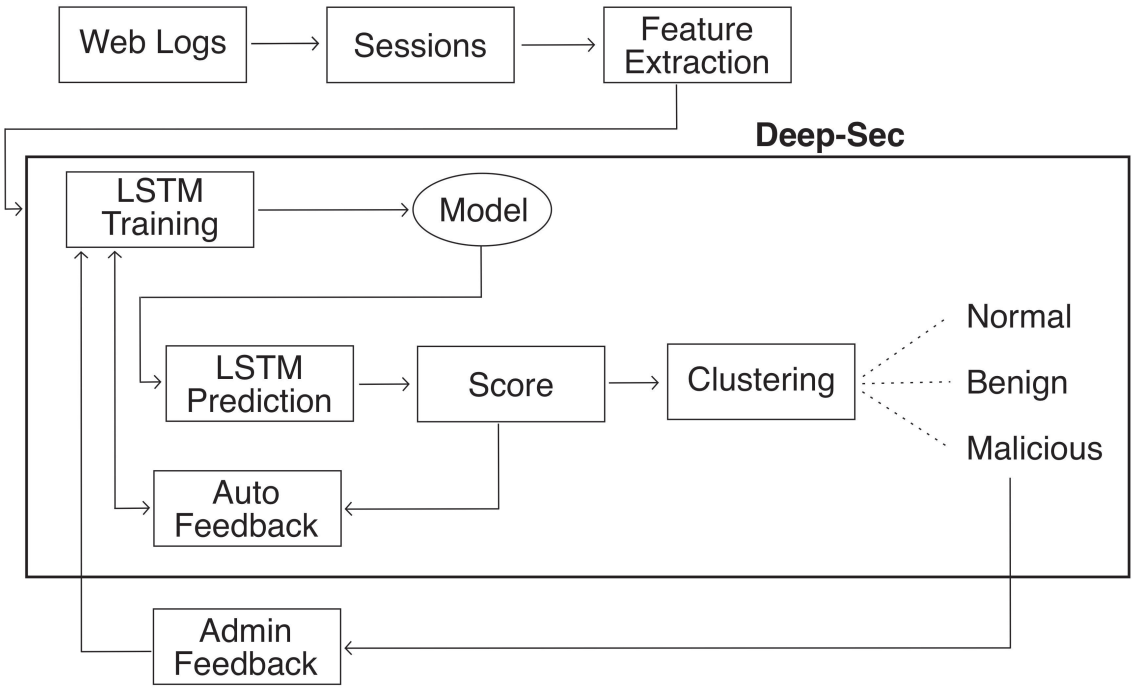

Figure 4.2. Deep-Sec workflow.

patterns from normal execution. Deep-sec leverages LSTM networks to model a system log as a natural language sequence, just as in DeepLog [24].

The final output of the LSTM network is a model that, given a sequence of log keys, predicts probabilities for all possible next keys. This model is used as input to the next step.

4.4.2 LSTM Prediction

Once the LSTM is trained, we are ready to use it to flag new sessions as normal or anomalous. Again following DeepLog, Deep-Sec divides up each session into sequences of size $|S|$ using a sliding window, as was done during training. For each sequence, the trained model is asked to produce a vector of probabilities giving the likelihood of seeing each of the possible keys next. If the key that *actually* came next was decided to be probable, this is considered normal—if it was improbable, the outcome is considered anomalous.

Whether or not a next-key was considered probable is determined by whether it appears in the top J most-likely keys (with the vector sorted by probability.) Deep-Sec differs substantially from its predecessors in the value it uses for J: while earlier work uses small values for J such as 5, Deep-Sec uses a value that is a substantial fraction of the total number of possible keys: our implementation uses $J = 2000$ for a total possible keyspace of $|K| = 24,296$. This is due to the much wider range of behaviors exhibited by humans than by automated processes: where a typical program may be well-modeled by a state machine with a small number of successor states, a human navigating a web interface has many more options regarding clicking on different links, using the Back button, using bookmarks or history to visit different pages, etc.

The other change we make from previous work is our scoring, as described next: previous work considers that if any key in a session is anomalous (low-probability) this marks the entire sessions as anomalous. Instead, we calculate an anomaly score using the predicted values.

4.4.3 Score

To distinguish benign and malicious behaviors, Deep-Sec computes a novel metric called the anomaly-score. This anomaly-score is an indication of how anomalous a certain session is. The anomaly-score is defined as:

$$\frac{\sum_{i=0}^{n-|S|} F(S_i)}{n-|S|}$$

...where n is the number of keys in the session, S_i is a sub-sequence (window) of the session), and $F(S_i)$ is 1 if the LSTM considers the key following S_i to be unlikely (using the definition above) and 0 otherwise. Thus, a session that has no anomalous windows gets a score of 0, and one in which every window was deemed anomalous gets the maximum score of 1. We find that in order for the score to be meaningful, it must encompass a reasonable number of predictions—our current implementation uses a minimum of $n - |S| > 15$. Below this length, sessions are assumed to be non-malicious. This results in a very small number of false negatives among short sessions.

For a session to be considered malicious it has to obtain an anomaly-score above a certain threshold. Most of the other related work either performs binary classification (anomalous and normal) without further considering the nature of the abnormalities, or they assume that all anomalies detected are malicious. This approach has many shortcomings because we have multiple classes of anomalies in most of the datasets. For example, in our dataset, some of the anomalies consists of legitimate bugs in the Cloudlab website or web crawlers and not only malicious behaviors. To determine the appropriate threshold, we use a clustering step.

4.4.4 Clustering and Labeling

The clustering is performed to determine the threshold value for the anomaly-score above which a session is considered malicious. This is in place of arbitrarily choosing a threshold and then relying on trial-and-error until we reach the ideal anomaly-score. We use the anomaly-scores produced by Deep-Sec during classification to cluster sessions based on their anomaly score. The clustering algorithm used is DBSCAN[29], which relies on a density based notion of the clusters.

Figure 4.3 shows the clustering for our Cloudlab dataset. It is clear from this figure that the appropriate threshold for the anomaly-score is 80%.

Density based clustering is efficient for the case of arbitrary shaped clusters. According to DBSCAN, if a point is close to many points of another cluster, then that point belongs to that cluster. There are two parameters on which it bases its clustering. The first is the

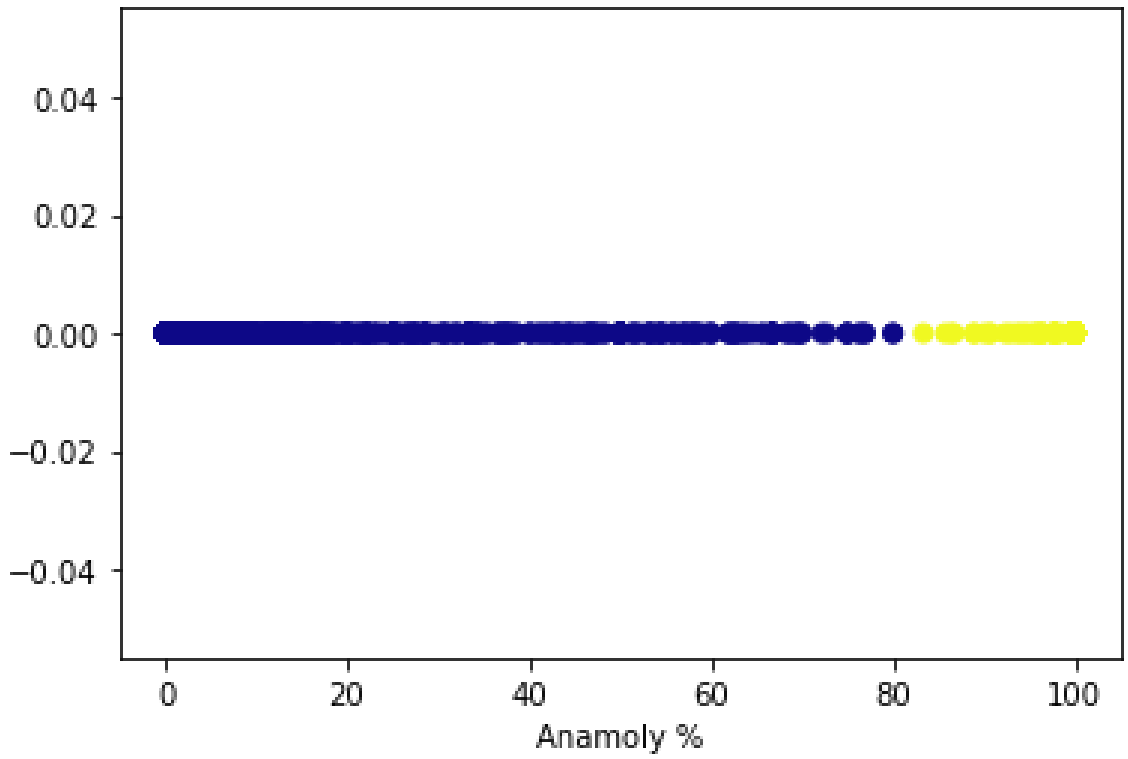

Figure 4.3. DBSCAN clustering of the Cloudlab dataset. The points belonging to the two clusters found by DBSCAN are shown as yellow and blue.

distance between points which specific neighborhood: *eps*. If the distance between two points are less than or equal to *eps* than those two points are considered neighbors. The second is the *minPoints*, which is the minimum number of data points that can form a cluster. The *minPoints* depends on the dimensionality of the data set (D), $minPoints \geq D + 1$. Here we set $eps = 0.075$ and $minPoints = 2$. This produces two clusters, which is the number we are looking for: we are trying to divide anomalies into two classes. The output from the clustering determines the predicted label: if the anomaly-score is 0 or very close to it ($0 + \epsilon$), the session is considered normal. If it is below the threshold determined by the clustering process, it is benign, and above this threshold, it is considered malicious.

4.4.5 Administrator Feedback

Deep-Sec incorporates a method for administrators to provide feedback to improve the system: if any of the sessions labeled as Malicious are false positives, administrators can

mark them as such. False-positive sessions are then added to the training set, causing them to be learned by the LSTM as "normal."

4.4.6 Automated Feedback

In addition to utilizing administrators' feedback, Deep-Sec includes a novel feedback loop in its design. To reduce the dependence on human experts needed for identifying the false positives, and make this process more automated, we apply a recursive approach where we take the results of one pass of Deep-Sec and apply it to a second pass using only the sessions originally identified as anomalous. Because the malicious anomalies are less likely to have a specific pattern or look like each other, we train Deep-Sec using the benign class of anomalies—in this pass, they are considered "normal," and any session that is found to be highly anomalous according to this trained model is very likely malicious. This serves as a confirmation for the malicious labels obtained in the first pass. To make sure that our training dataset has only benign anomalies we only train on the sessions that have an anomaly-score less than 50% (less than half of the session is anomalous) from the first pass of Deep-Sec. In this second-pass of Deep-Sec, we will record the sessions classified as normal and add them to the training dataset from the first pass to improve its accuracy. This method reduces the false positives, as shown in the evaluation, and allows Deep-Sec to expand its definition of a normal behavior without the need for human intervention.

4.5 Evaluation

The code for Deep-Sec was built on top of Wu's implementation of DeepLog [82]. Our evaluation was done in the CloudLab [26] testbed on a machine with two Intel Xeon Silver 4114 10-core CPUs at 2.20 GHz and one NVIDIA 12GB PCI P100 GPU.

To measure the performance of our prototype, we have divided our evaluation into three parts: (a) We perform a quantitative evaluation by calculating the confusion matrix to measure the performance in terms of accuracy, precision, specificity, sensitivity, and F1-score. This is mainly done to estimate how reliable Deep-Sec is in subdividing anomalies into malicious and benign. (b) We examine the types of malicious sessions detected by Deep-Sec to demonstrate its ability to give helpful information to system administrators.

(c) We measure the predictiveness of Deep-Sec on prefixes of full sessions to evaluate the feasibility of building an online version of Deep-Sec that can block malicious requests as they occur.

4.5.1 Quantitative Evaluation

The dataset used to evaluate Deep-Sec comes from Cloudlab log files. These sessions were grouped based on specific user activity within a given week. From a total of 3,118 sessions, 404 sessions were used for training, and 2,714 were used for testing.

Table 4.1 measures the performance of Deep-Sec in detecting malicious activity in a given dataset with mixed entries of normal sessions, benign anomalies, and malicious anomalies.

Figure 4.4 display the true positives, false positives, true negatives, and false negatives obtained from using Deep-Sec to identify the malicious sessions in our dataset. Out of 2,714 sessions used for testing, 1,408 were identified as anomalous, including both benign and malicious abnormalities. From these 1,408 sessions, 50 sessions were labeled as malicious. Only 3 of them are false positives due to a new pattern appearing in the test dataset that wasn't introduced before in the training dataset. This can be mitigated by using a feedback loop and adding these sessions to the training dataset for future passes. In the first run of Deep-Sec, we initially found 9 false positives due to a mistake in the manual labeling of some of the normal sessions as anomalous. These 9 false positives were identified by the automatic feedback loop and removed for future runs.

More importantly, out of 2714 sessions, Deep-Sec only mislabeled two malicious sessions as benign. Both false negatives were due to sessions that fell below our threshold for meaningful scoring—while their anomaly scores were high, Deep-Sec did not consider the sessions long enough to reliably label as malicious. This is promising because it indicates

Measure	Value
F1-score	0.9495
Precision	0.9400
Recall	0.9592
Accuracy	0.9982

Table 4.1. Performance evaluation.

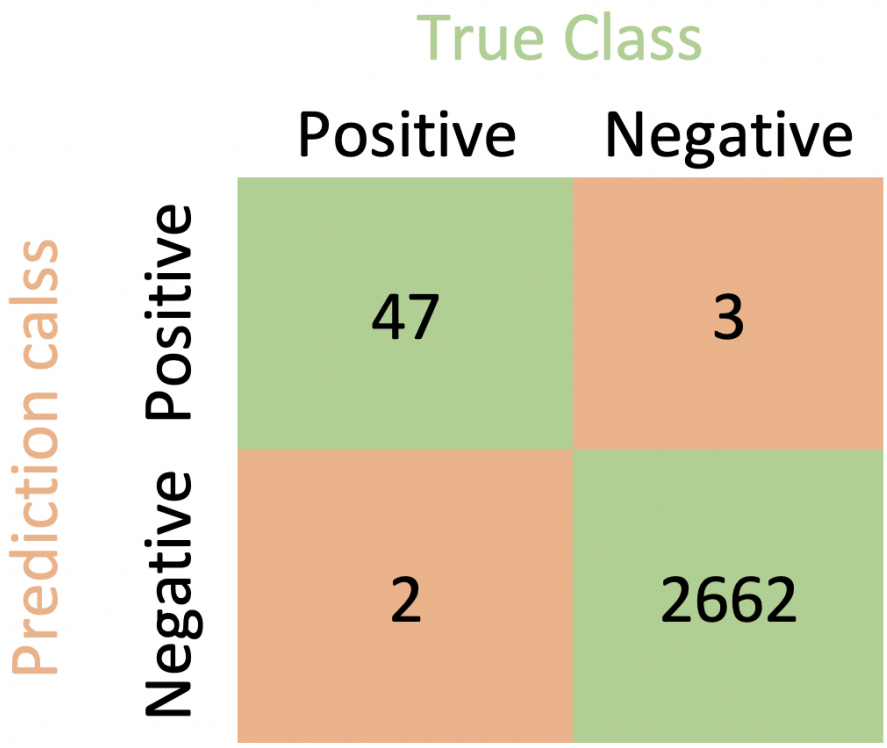

Figure 4.4. True and prediction classes.

that it is unlikely for system administrators to overlook attacks that could compromise the system by using Deep-Sec.

Since 47 of the 1,408 sessions that were originally identified as anomalous are labeled as malicious this means that without Deep-Sec, system administrators who rely solely on LSTM for binary classification would have two options: (a) Either to consider every anomalous session malicious and end up wasting time checking on all of the 1,408 sessions where only 3.5% are indeed malicious, (b) Or more dangerously, consider all of the anomalous sessions benign and risk compromising the system security by not properly handling those 47 sessions.

4.5.2 Qualitative Evaluation

To get a better understanding of the types of malicious anomalies present in our dataset, we took a closer look at the malicious sessions. In this section, we present a qualitative description of some notable sessions. Most malicious sessions appear to be looking for

the presence of software with known security vulnerabilities—some of it very common, such as WordPress [81], but others that seem more obscure, such as apparent e-commerce platforms that appear to be limited to specific language groups such as Russian or Dutch.

The 'common' thing that we find amongst these sessions is the lack of consistent patterns. There is wide variety among the malicious sessions, any many seem to use strategies for looking for specific URLs that are disjoint with each other. This illustrates why Deep-Sec is effective at finding malicious sessions—their anomaly scores are high due to this varied behavior—and suggests that it would be difficult to capture many of them with pre-formed list of "suspicious" URLs.

4.5.2.1 Security Scans

Cloudlab's institution contracts with an outside security firm to do periodic scans of its network, looking for known vulnerabilities. We classify these as malicious in our dataset, as they represent clear attempts to find vulnerable software. Such sessions show up approximately once a week (six times), and comprise the longest malicious sessions, at about 16,000 requests each. These sessions follow fairly predictable patterns, though they do vary from week to week, presumably as the scanning company updates its database of known vulnerabilities.

One interesting thing we note about these scans is that they have almost no overlap with URLs requested during the other malicious sessions that we found. The little overlap that does exist is in the form of very generic URLs such as `/` and `/login`—in other words, the vulnerabilities searched for by the security scanning firm and malicious sources seem entirely disjoint. We believe this helps to illustrate the value of Deep-Sec's approach of learning from client behavior: to the best of our knowledge, the security firm is using a (large) list of known vulnerabilities that likely come from public sources such as CVEs and the firm's own vulnerability databases. While it is no doubt valuable to watch for, and fix, any vulnerabilities found by these professional scans, Deep-Sec uncovers a different set that, if they were present, could represent a problem for the site owner. Deep-Sec's approach finds malicious behavior that may not be widely known, reflects ongoing attacks, and is specialized to a particular site. This gives the site a more specific set of malicious behaviors to defend against.

4.5.2.2 Complex Scans for WordPress

WordPress is widely deployed and the subject of numerous security advisories, so it is not surprising that it is a common target of malicious sessions. What we found more surprising, however, is that some malicious actors seem to put significant effort into finding copies of it installed in alternate locations. Looking at sessions that have clear WordPress-scanning behavior, we find them looking in alternate paths such as `blog`, `2020` (and other years), `new`, `old-site`, `bak`, and many others. Some malicious actors look for the presence of specific WordPress plugins, presumably ones with vulnerabilities. Others try to fetch WordPress configuration files (which may contain secrets such as database passwords or other information that could help an attacker exploit the system); some go to significant effort to find backup or temporary versions of such files, likely because such files might be missed in rules designed to block access to the config file itself. Examples of such filenames include `.wp-config.php.swp` (used by the `vim` editor), `wp-config.php~` (used by Emacs), `wp-config.orig` (which can be created by the `patch` utility), and `wp-config.php.bak`. Overall, we found 16 malicious sessions that had clear signs of scanning for WordPress; what is notable is that most of these sessions had very little in common with each other in terms of the specific sequence of URLs they visited. This is why they ended up with high anomaly scores. Interestingly, some of these sessions contained short sections of 'normal' behavior (such as fetching the front page or other prominent public pages), suggesting that the attacker either visited the page manually before launching the scan or that the scan includes a webcrawler phase.

4.5.2.3 Attempts to Find Archive or Backup Files

Three sessions exhibited interesting behavior in which they were looking for apparent archives of the entire site, with filenames such as `<sitename>.tar.gz`, `<sitename>.zip`, etc. There were a number of variations of these filenames that replaced `<sitename>` with strings such as `www`, `wwwroot`, `web`, etc. We surmise that these attackers are looking for cases in which the site was deployed by copying, say, a tarball, to the webserver, untarring it, and neglecting to remove the original tarball. Such archive files could contain the source for server-side code that would not normally be fetchable, configuration files that could contain secrets, and other files that might otherwise be blocked from public access by rules

in the webserver. Variants of this attack also look for files with suffixes `.sql`, `.sql.zip`, etc., presumably looking for uploads or backups of database contents.

4.5.2.4 Other Behaviors

We found a variety of other interesting behaviors in the malicious sessions. Some, for example, appear to look for other prior signs of breakins (eg. `indoxploit.php`, `shell.php`). These sessions also contained more innocuous-looking URLs, such as `new_license.php` and `V5.php`—this may provide an interesting list of files to look for that may (at least according to this attacker) be indication of a previous compromise. Others seem to look for guest books and contact pages (sometimes using spellings in multiple languages, such as `gaestebuch` and `gastenboek`) and/or trying a variety of different file extensions (`.html`, `.php`, `.jsp`, `.aspx`, etc.) We consider it likely that these are attempts to harvest email addresses or other contact information. Scans looking for specifically for Russian-language pages were fairly common, as evidenced by paths with `/ru/`, `ru_utf8`, etc. Other particular language locales appear as well, including `en-GB`. Various other CMS, administration tools, e-commerce software, developer tools, and server-side frameworks was looked for, including `kcfinder`, `jboss`, `magneto`, `owncloud`, and `phpunit` None were anywhere near as common as WordPress, which dominated the clearly-identifiable software packages scanned by attackers. Many malicious sessions looked for `.env` files under a wide variety of paths; we presume this is an attempt to gain more metadata about the site. We found one session that tried to fetch a variety of what appeared to be RSS feeds, many of which were clearly unrelated to the websites hosted on the server, including `feeds/justice-news.xml` and `/pittsburgh/Rss.xml`. We surmise that this was a buggy or misconfigured crawler or feed aggregator.

4.5.3 Predictive Ability

The final question we ask is how *predictive* our malicious anomaly detection system is: that is, if it sees just a subset of a full session, does it produce the same label as it would for the full session? This gives us a sense of how effective Deep-Sec would be for building a system that blocks malicious requests on-line. We leave the actual construction of such a system as future work, but use this measurement to show that such a system is feasible.

To evaluate the predictive ability of Deep-Sec, for a session S, we define a prefix p_n

of the session to be the ordered elements of the session $S_0 ... S_n$. Note that the shortest meaningful prefix in this context is the window size used by Deep-Sec plus one ($|S| + 1$), since this is the minimum required to make a prediction; the longest prefix is the complete session. Label $L(S)$ is the label assigned by Deep-Sec for the whole session, and label $L(p_n)$ is the label that Deep-Sec assigns if shown only p_n. What we are looking for is the length n of the prefix at which Deep-Sec *latches* to the correct label: that is, the value N at which $\forall n \geq N, L(p_n) = L(S)$. Said another way, this is the prefix length at which Deep-Sec assigns the correct label to the session, and then continues to produce the correct label for all other prefixes up to the full length of the session.

We report N as a fraction of the full length of the session. If this value is close to one, Deep-Sec does not converge on the correct label until it has seen nearly the whole session, and therefore would not be useful in deciding whether to block further requests in this session. If it is close to zero, this means that Deep-Sec produces the correct label given a short prefix, and could reliably indicate whether the rest of the session should be allowed or blocked.

Figure 4.5 shows a histogram of the values of N. As can be seen, most of the time, Deep-Sec can produce correct labels given less than 5% of a session, meaning that, for malicious sessions, 95% of the session could be blocked. The cases in which it takes Deep-Sec a higher fraction of the session tend to be short sessions: if they are not much longer than the window size, Deep-Sec is only able to start making predictions after a substantial fraction of the session has passed. This shows that Deep-Sec has great potential to be used on-line to block malicious sessions as they are seen in real-time.

In 67% of the test dataset, Deep-Sec could predict the correct label in the first window examined. And in 2.5% of the data, it correctly labeled the session in the second window. And only 0.5% of the datasets needed more than 10 windows examined to latch to the correct label. More importantly, Deep-Sec was able to flag 84% of the malicious sessions in ≤ 5 windows, and 66% of these malicious sessions could be flagged and blocked from the first window examined.

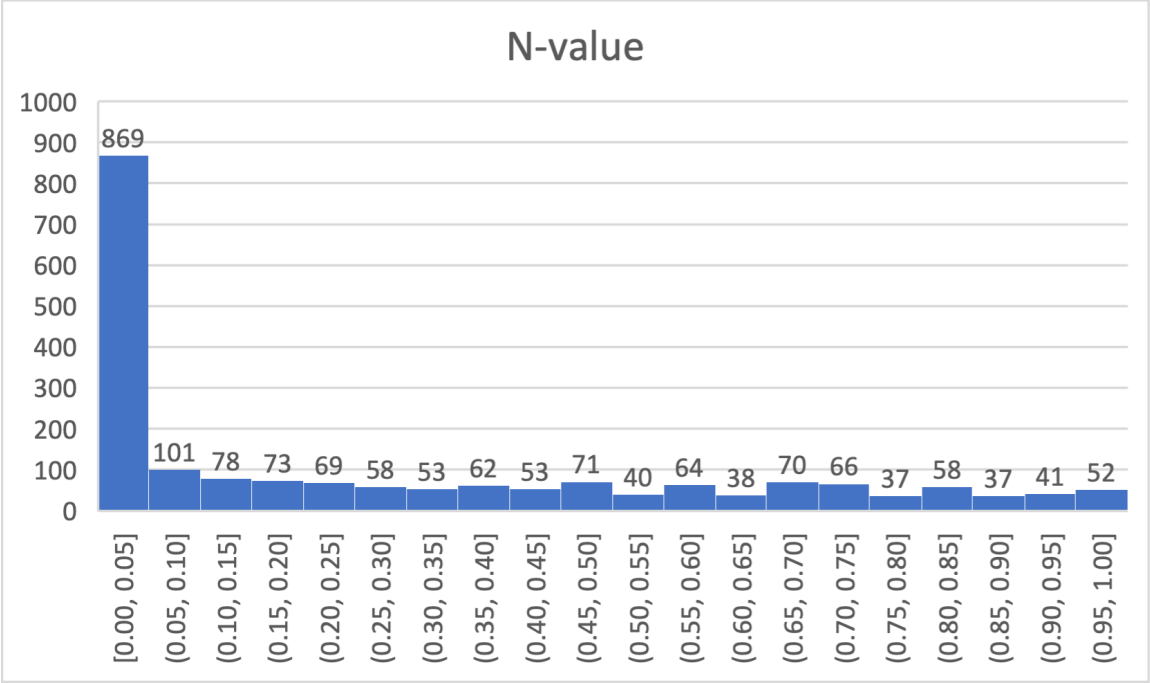

Figure 4.5. N-values for test dataset.

4.6 Conclusion

It is important for system administrators to be able to distinguish between anomalous incidents that require a security response, and those that come from other sources. Deep-Sec builds on earlier anomaly detection work to distinguish between *benign* and *malicious* anomalies, giving administrators and security teams the information they need to do their jobs. It has proved to be highly accurate on a dataset drawn from a production web system, and we attribute this to two facts: First, malicious anomalies turn out to be *more anomalous* according to the detectors than benign ones. Second, events in the class of benign anomalies have more in common with each other than do events in the class of malicious anomalies. As a result, we are able to label highly-anomalous results as malicious, and can train an auto-feedback step on benign anomalies, further reducing false positives.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Systems failures do not only involve systems' unavailability. It can also include the failure in delivering any guarantees or services users rely on these systems for. These failures affect many people depending on these systems for critical services or for storing sensitive information. The costs of system failures may be extremely high for both ends, the service provider and the user, because failures generally have a financial impact and can cause costly physical damages. For example, the Facebook outage on October 5th, 2021, lasted for only six hours but costed Facebook an estimated total loss of revenues of roughly $99.75 million and prevented more than 2.89 billion users from accessing Facebook's services [48]. Additionally, data breaches are extremely expensive, especially to more prominent institutes, the average global cost of a cybersecurity data breach is $3.83 million [12].

In this dissertation, I presented studies on complicated computer systems and their respective dependability with a goal of proposing practical methods that can improve the reliability and assist systems' users achieving their goals.

In Chapter 2, I examined users' privacy in content distribution networks. CDNs improve the overall performance by bringing the services closer to the end-users. It also enhances reliability by allowing origin servers to offer uninterrupted service even during some failures. And most importantly, it improves the resilience against attacks by impersonating the origin server. However, this arrangement requires the end-users to share some of their sensitive secrets with the CDN to allow it to fully impersonate the origin server. This inflicts a dangerous system vulnerability especially with the emergence of serverless computing such as in the AWS Lambda@edge service [2] and Akamai's delivery of edge computing solutions [37], because even if we assume the integrity of the CDN, sometimes failure within the CDN can cause data leakage, such as in the Cloudbleed

incident [36]. To address this problem, I presented Harpocrates. It is a framework that utilizes Intel's SGX technology to protect the privacy of users' secrets by protecting it from direct CDN access without limiting the users from fully leveraging the CDN's capabilities. Harpocrates enables the users to push active computation that might require private users' information to the edge without relinquishing any privacy guarantees.

Although this proposed design improves performance and scalability by eliminating the need for the long-distance connection to the origin server, the scalability in the current prototype can present a problem because of the need for a separate SGX enclave for each origin server. This problem can be mitigated with a method to guarantee clean separation between different CDN's clients' data. Furthermore, I propose to further improve performance by leveraging Intel QuickAssist Technology (QAT) [63]. QAT-based accelerators can speed HTTPS connection services by securely offloading TLS operations on private keys to the NIC. The use cases I presented demonstrate the power of Harpocrates, but this prototype is certainly not limited to those use cases. It is also possible to implement additional cloud-related applications, for example, in banking, healthcare management, government services, or any service that requires both secrecy and high performance.

In Chapter 3, I examined systems logs that manifest code execution in data center environments. This study aims to model the correct system behavior and help systems administrators find anomalies before being reported by users. I also investigated the persistence of systems invariants through time to evaluate if anomalies detectors need to be periodically re-trained or not. Furthermore, I explored the usefulness and interpretability of the mined invariants to measure how useful they can be for system administrators.

In this dissertation, I have used fixed time periods for sessions (24 hours) and retraining periods (3 months); for future work, it might be interesting to investigate other values, including using sliding windows for sessions or flexible size windows. It might also provide insightful information if applied to other system's provisioning logs other than the boot info logs.

In Chapter 4, I looked at a different kind of logs that are derived from human behaviors rather than automated processes. To help system administrators detect malicious needles in the logfiles haystacks, I designed a framework called Deep-Sec. It is an anomaly detection system that improves on DeepLog [24] by introducing a novel scoring system that

enables system admins to prioritize different anomalies by labeling them as malicious and benign. To evaluate the quantitative and qualitative performance of Deep-Sec on. I used a labeled dataset from Cloudlab. To the best of my knowledge, this dataset is unique: unlike existing public datasets, because not only did I group the data into sessions, but it was manually labeled into three different classes instead of two: normal, benign, and malicious. Moreover, Deep-Sec achieved adequate performance in a dataset that reflects actual human behavior within a website rather than reflecting an automated code execution.

There is still more work that can be done to expand this research. Though I have some promising preliminary experiments with predicting classes for ongoing (partial) events, more work would still need to be done to turn this into a system that could block malicious traffic in real-time. It would also be interesting to apply this work to other types of security-relevant logfiles, beyond web server logs; I expect that Deep-Sec will be most valuable on logs that record human behavior—in this setting, malicious behavior, which is often automated, stands out. Furthermore, although the linear scoring system used in Deep-Sec has achieved upstanding results, it might be valuable to explore different scoring systems on other datasets.

Collectively, these studies aim to improve systems' dependability by enhancing availability, safety, resilience, and security.

REFERENCES

[1] Amazon, *Amazon CloudFront*. `https://aws.amazon.com/cloudfront/`. (20 September 2018).

[2] Amazon Web Services, *Lambda@edge*. `https://aws.amazon.com/lambda/edge/`. (16 August 2018).

[3] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell, et al., *SCONE: Secure linux containers with intel SGX*, in 12th USENIX Symposium on Operating Systems Design and Implementation OSDI 16, 2016, pp. 689–703.

[4] H. J. Asghar, L. Melis, C. Soldani, E. De Cristofaro, M. A. Kaafar, and L. Mathy, *Splitbox: Toward efficient private network function virtualization*, in Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, 2016, pp. 7–13.

[5] E. Baseman, S. Blanchard, N. DeBardeleben, A. Bonnie, and A. Morrow, *Interpretable anomaly detection for monitoring of high performance computing systems*, in Outlier Definition, Detection, and Description on Demand Workshop at ACM SIGKDD, 2016.

[6] E. Baseman, S. Blanchard, Z. Li, and S. Fu, *Relational synthesis of text and numeric data for anomaly detection on computing system logs*, in 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), 2016, pp. 882–885.

[7] P. C. Bates and J. C. Wileden, *High-level debugging of distributed systems: The behavioral abstraction approach*, J Syst Software, 3 (1983), pp. 255–264.

[8] A. Baumann, M. Peinado, and G. Hunt, *Shielding applications from an untrusted cloud with Haven*, in OSDI '14, USENIX – Advanced Computing Systems Association, 2014.

[9] K. Bhargavan, I. Boureanu, P. A. Fouque, C. Onete, and B. Richard, *Content delivery over TLS: A cryptographic analysis of keyless SSL*, in 2017 IEEE European Symposium on Security and Privacy (EuroS&P), 2017, pp. 1–16.

[10] E. Bos, *Analyzing the performance of Cloudflare's Anycast CDN, a case study*, in 27th Twente Student Conference on IT, 2017.

[11] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch, and R. Kapitza, *Securekeeper: Confidential ZooKeeper using Intel SGX*, in Proceedings of the 17th International Middleware Conference, 2016, p. 14.

[12] K. Brisco, *Cost of a data breach: Behind the numbers of a cybersecurity response plan*. `https://www.secureworks.com/blog/data-breach-response-planning-cyber-threat-intelligence`. (20 October 2020).

[13] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, *Recurrent neural network attention mechanisms for interpretable system log anomaly detection*, in Proceedings of the First Workshop on Machine Learning for Computing Systems, 2018, pp. 1–8.

[14] S. Bursic, V. Cuculo, and A. D'Amelio, *Anomaly detection from log files using unsupervised deep learning*, in International Symposium on Formal Methods, 2019, pp. 200–207.

[15] F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, *Measurement and analysis of private key sharing in the https ecosystem*, in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 628–640.

[16] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, *SgxPectre attacks: Leaking enclave secrets via speculative execution*, IEEE Secure. Priv., 18(2020), pp. 255–264.

[17] M. Cinque, D. Cotroneo, and A. Pecchia, *Event logs for the analysis of software failures: A rule-based approach*, IEEE Trans. Softw. Eng., 39 (2012), pp. 806–821.

[18] Cloudflare, *Aggregating multiple requests*. `https://developers.cloudflare.com/workers/recipes/aggregating-multiple-requests/`.

[19] ——, *Cloudflare - the web performance & security company*. `https://www.cloudflare.com/`. (20 September 2018).

[20] ——, *Overview of keyless SSL*. `https://www.cloudflare.com/ssl/keyless-ssl/`. (30 September 2018).

[21] C. Cortes and V. Vapnik, *Support-vector networks*, Mach Learn, 20 (1995), pp. 273–297.

[22] D. Donchev, V. Vassilev, and D. Tonchev, *Impact of false positives and false negatives on security risks in transactions under threat*, in International Conference on Trust and Privacy in Digital Business, 2021, pp. 50–66.

[23] M. Du and F. Li, *Spell: Streaming parsing of system event logs*, in Proceedings of 16th IEEE International Conference on Data Mining (IEEE ICDM), 2016.

[24] M. Du, F. Li, G. Zheng, and V. Srikumar, *Deeplog: Anomaly detection and diagnosis from system logs through deep learning*, in Proceedings of 24th ACM Conference on Computer and Communications Security (CCS), 2017.

[25] Duktape, *Duktape*. `http://duktape.org/`. (13 August 2018).

[26] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, *The design and operation of CloudLab*, in Proceedings of the USENIX Annual Technical Conference (ATC), 2019.

[27] R. Edmunds and E. Feinstein, *Deep semi-supervised embeddings for dynamic targeted anomaly detection*, preprint (2017). `http://rileyedmunds.com/pdf/iat2017.pdf`.

[28] Elastic Co., *Grok filter plugin (documentation)*. `https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html`. (15 March 2020).

[29] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., *A density-based algorithm for discovering clusters in large spatial databases with noise.*, in KDD, (1996), pp. 226–231.

[30] H. Estiri and S. N. Murphy, *Semi-supervised encoding for outlier detection in clinical observation data*, Comput Meth Prog Bio, 181 (2019), p. 104830.

[31] Flux Research Group, University of Utah, *Emulab source code*. `https://gitlab.flux.utah.edu/emulab/emulab-devel`. (10 April 2020).

[32] S. R. Gaddam, V. V. Phoha, and K. S. Balagani, *K-means+ID3: A novel method for supervised anomaly detection by cascading k-means clustering and ID3 decision tree learning methods*, IEEE Trans Knowl Data Eng, 19 (2007), pp. 345–354.

[33] Y. Gao, Y. Ma, and D. Li, *Anomaly detection of malicious users' behaviors for web applications based on web logs*, in 2017 IEEE 17th International Conference on Communication Technology (ICCT), 2017, pp. 1352–1355.

[34] D. Goltzsche, S. Rüsch, M. Nieke, S. Vaucher, N. Weichbrodt, V. Schiavoni, P. Aublin, P. Costa, C. Fetzer, P. Felber, et al., *Endbox: Scalable middlebox functions using client-side trusted execution*, in Proceedings of the 48th International Conference on Dependable Systems and Networks, vol. 18, 2018.

[35] Google, *Google Cloud CDN - low latency content delivery*. `https://cloud.google.com/cdn/`. (25 August 2018).

[36] J. Graham-Cumming, *Incident report on memory leak caused by Cloudflare parser bug*, Cloudflare Blog. December, 11 (2017).

[37] G. Griffiths, *Computing at the edge*. `https://www.akamai.com/blog/edge/computing-at-the-edge`. (20 October 2020).

[38] J. Gu, M. Schubert, and V. Tresp, *Semi-supervised outlier detection using a generative and adversary framework*, Int. J. Comput. Inf. Sci., 12 (2018), pp. 891–898.

[39] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, *An evaluation study on log parsing and its use in log mining*, in Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2016.

[40] S. He, J. Zhu, P. He, and M. R. Lyu, *Experience report: System log analysis for anomaly detection*, in 27th IEEE International Symposium on Software Reliability Engineering (ISSRE), 2016.

[41] Intel, *Intel software guard extensions developer guide*. `https://download.01.org/intel-sgx/linux-1.7/docs/Intel_SGX_Developer_Guide.pdf`. (10 April 2018).

[42] Y. Li, J. Xia, S. Zhang, J. Yan, X. Ai, and K. Dai, *An efficient intrusion detection system based on support vector machines and gradually feature removal method*, Expert Syst. Appl., 39 (2012), pp. 424–430.

[43] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, *Log clustering based problem identification for online service systems*, in 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), 2016, pp. 102–111.

[44] F. T. Liu, K. M. Ting, and Z.-H. Zhou, *Isolation forest*, in 2008 8th IEEE international Conference on Data Mining, 2008, pp. 413–422.

[45] J. G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, *Mining invariants from console logs for system problem detection.*, in USENIX Annual Technical Conference, 2010, pp. 1–14.

[46] J. Ma, L. Sun, H. Wang, Y. Zhang, and U. Aickelin, *Supervised anomaly detection in uncertain pseudoperiodic data streams*, ACM Trans. Internet Technol., 16 (2016), pp. 1–20.

[47] M. Milutinovic and W. He, *Secureworker, 2017*. `https://github.com/luckychain/node-secureworker`. (18 July 2018).

[48] C. Morris, *Facebook's outage cost the company nearly $100 million in revenue*. `https://fortune.com/2021/10/04/facebook-outage-cost-revenue-instagram-whatsapp-not-working-stock/`. (25 October 2021).

[49] M. Nadeem, O. Marshall, S. Singh, X. Fang, and X. Yuan, *Semi-supervised deep neural network for network intrusion detection*, in Proceedings of the KSU Conference on Cybersecurity Education, Research and Practice, 2016.

[50] K. Nagaraj, C. Killian, and J. Neville, *Structured comparative analysis of systems logs to diagnose performance problems*, in 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), 2012, pp. 353–366.

[51] D. Naylor, R. Li, C. Gkantsidis, T. Karagiannis, and P. Steenkiste, *And then there were more: Secure communication for more than two parties*, in Proceedings of the 13th International Conference on Emerging Networking Experiments and Technologies, 2017, pp. 88–100.

[52] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste, *Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS*, Comput. Commun. Rev., 45 (2015). pp. 199–212.

[53] M. Newbold, *Reliability and state machines in an advanced network testbed*, Master's thesis, University of Utah, Salt Lake City, UT, 2004.

[54] E. Nygren, R. K. Sitaraman, and J. Sun, *The Akamai network: A platform for high-performance internet applications*, Oper. Syst. Rev., 44 (2010), pp. 2–19.

[55] D. O'Keefe and J. Tian, *SGXSpectre: Sample code demonstrating a Spectre-like attack against an Intel SGX enclave*. `https://github.com/lsds/spectre-attack-sgx`. (18 March 2018).

[56] M. Orenbach, P. Lifshits, M. Minkin, and M. Silberstein, *Eleos: Exitless OS services for SGX enclaves*, in Proceedings of the 12th European Conference on Computer Systems, 2017, pp. 238–253.

[57] E. Racah, C. Beckham, T. Maharaj, S. E. Kahou, C. Pal, et al., *Extremeweather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events*. `https://arxiv.org/abs/1612.02095`. (10 July 2021).

[58] P. Rodriguez, J. Wiles, and J. L. Elman, *A recurrent neural network that learns to count*, Connect. Sci., 11 (1999), pp. 5–40.

[59] M. Sabokrou, M. Khalooei, M. Fathy, and E. Adeli, *Adversarially learned one-class classifier for novelty detection*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 3379–3388.

[60] R. Salakhutdinov and H. Larochelle, *Efficient learning of Deep Boltzmann Machines*, in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 693–700.

[61] I. H. Sarker, Y. B. Abushark, F. Alsolami, and A. I. Khan, *Intrudtree: A machine learning based cyber security intrusion detection model*, Symmetry, 12 (2020), p. 754.

[62] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, *Blindbox: Deep packet inspection over encrypted traffic*, Comput. Commun. Rev., 45 (2015), pp. 213–226.

[63] X. Shuai, L. Yao, and Z. Wang, *Qat: Evaluation of a dedicated hardware accelerator for high performance web service*, in 2018 20th International Conference on Advanced Communication Technology (ICACT), 2018, pp. 277–280.

[64] M. A. Siddiqui, J. W. Stokes, C. Seifert, E. Argyle, R. McCann, J. Neil, and J. Carroll, *Detecting cyber attacks using anomaly detection with explanations and expert feedback*, in ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019, pp. 2872–2876.

[65] R. Sommer and V. Paxson, *Outside the closed world: On using machine learning for network intrusion detection*, in 2010 IEEE Symposium on Security and Privacy, 2010, pp. 305–316.

[66] H. Song, Z. Jiang, A. Men, and B. Yang, *A hybrid semi-supervised anomaly detection model for high-dimensional data*, Comput. Intell. Neurosci, 2017 (2017).

[67] N. Sullivan, *Keyless SSL: The nitty gritty technical details*, Cloudflare blog. (23 August 2017).

[68] The CloudLab Team, *Cloudlab hardware (documentation)*. `https://docs.cloudlab.us/hardware.html#(part._cloudlab-utah)`. (10 March 2020).

[69] ——, *The CloudLab website*. `https://cloudlab.us`. (13 January 2020).

[70] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, *Shieldbox: Secure middleboxes using shielded execution*, in Proceedings of the Symposium on SDN Research, 2018, p. 2.

[71] C. Tsai, D. E. Porter, and M. Vij, *Graphene-SGX: A practical library OS for unmodified applications on SGX*, in Proceedings of the USENIX Annual Technical Conference (ATC), 2017, p. 8.

[72] UCI Knowledge Discovery in Databases, *KDD Cup 1999 data*. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`. (5 June 2021).

[73] K. Varda, *Introducing cloudflare workers: Run javascript service workers at the edge*, Cloudflare Blog. (28 June 2018).

[74] H. Vill, *SGX attestation process*. `https://courses.cs.ut.ee/MTAT.07.022/2017_spring/uploads/Main/hiie-report-s16-17.pdf`. (30 May 2018).

[75] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol, *Extracting and composing robust features with denoising autoencoders*, in Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 1096–1103.

[76] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, *CAPTCHA: Using hard AI problems for security*, in Proceedings of The International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), 2013.

[77] W3C, *Web crypto API*. `https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API`. (17 October 2018).

[78] C. Weng, J. Li, W. Li, P. Yu, and H. Guan, *STYX: A trusted and accelerated hierarchical SSL key management and distribution system for cloud based CDN application*, in Proceedings of the ACM Symposium on Cloud Computing (SOCC), 2017.

[79] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, *An integrated experimental environment for distributed systems and networks*, in Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI), 2002.

[80] S. Wold, K. Esbensen, and P. Geladi, *Principal component analysis*, Chemom. Intell. Lab. Syst., 2 (1987), pp. 37–52.

[81] WordPress, *Wordpress*. `wordpress.com`. (13 September 2021).

[82] Y. Wu, *DeepLog*. `https://github.com/wuyifan18/DeepLog`. (1 May 2021).

[83] Y. Wu, Y. Sun, C. Huang, P. Jia, and L. Liu, *Session-based webshell detection using machine learning in web logs*, Secur. Commun. Netw., 2019 (2019).

[84] D. Wulsin, J. Blanco, R. Mani, and B. Litt, *Semi-supervised anomaly detection for EEG waveforms using deep belief nets*, in 2010 9th International Conference on Machine Learning and Applications, 2010, pp. 436–441.

[85] T. Yang and V. Agrawal, *Log file anomaly detection*, CS224d Fall, 2016 (2016).

[86] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, *Tools and benchmarks for automated log parsing*, in Proceedings of the International Conference on Software Engineering (ICSE), 2019.