Homomorphic Encryption at Work for Private Analysis of Security Logs

Aymen Boudguiga[1], Oana Stan[1], Hichem Sedjelmaci[2] and Sergiu Carpov[1]
[1]*CEA-LIST, 91191, Gif-sur-Yvette, France*
[2]*Orange Labs, 92320, Châtillon, France*

Keywords: Privacy, Log Management, SIEM, Homomorphic Encryption.

Abstract: One important component of incident handling in cyber-security is log management. In practice, different software and/or hardware components of a system such as Intrusion Detection Systems (IDS) or firewalls analyze network traffic and log suspicious events or activities. These logs are timestamped, gathered by a log collector and centralized within a log analyzer. Security Incidents and Events Management (SIEM) system is an example of a such log analysis tool. SIEM can be a dedicated network device or a Cloud service offered by a security services provider. Providing SIEM as a cloud service raises privacy issues as logs contain confidential information that must not be disclosed to third parties. In this work, we investigate the possible use of homomorphic encryption to provide a privacy preserving log management architecture. We explain how SIEM can be adapted to treat encrypted logs. In addition, we evaluate the homomorphic classification of IDS alerts from NSL-KDD set with an SVM linear model.

1 INTRODUCTION

Nowadays, Security Incidents and Events Management (SIEM) system (Scarfone and Souppaya, 2006) are established as a reference for cyber-security log analysis. SIEM can be a dedicated network device or a Cloud service offered by a security services provider. A SIEM receives and treats logs in a quasi real-time (i.e., online) but it may support offline logs treatment (Limmer and Dressler, 2008). It is mainly in charge of logs normalization, analysis, correlation and storage. In practice, SIEM either respond to detected incidents with appropriate countermeasures or generate automated alerts about malicious activities for security administrators within a Security Operations Center (SOC) (Jarpey and McCoy, 2017; Nathans, 2015).

The SOC provides a dashboard interface for incidents visualization which facilitates their monitoring by incident analysts. The latter review the SIEM automated analysis and investigate its alerts with close scrutiny. They are in charge of attack forensics, logs storage, distant policy enforcement and incident response. That is, SOC deploys security patches and adequate countermeasures on vulnerable devices. SOC are directed by laws and SOC employees incur jail if they disclose any information about the analyzed logs of their clients.

Problem Statement – For accessibility and management ease, SIEM can be provided by third parties as distributed services in the Cloud. When an IDS sends its logs to a remote SIEM to signal an incident on a device, its logs can be intercepted by a hacker. The latter can exploit threats related to the reported incident to attempt an attack on the vulnerable device. This attack becomes more dangerous when the reported incident refers to a well-known device that is deployed in many systems simultaneously. For example, an IDS can report the presence of a malware disclosing users logins and passwords of a database. If the attacker knows that many companies are using the same vulnerable database, she can target all these companies databases with the same malware. Consequently, it becomes compulsory to ensure logs confidentiality and integrity during their transmission to a third party SIEM. Fortunately, appropriate encryption and message authentication codes can solve this problem.

However, when SIEM are managed by third parties that are honest-but-curious, classical encryption is not enough for preserving logs privacy. Indeed, the SIEM can decrypt the received logs, analyze them and send back incident responses to the log generators. In the meantime, the corrupted SIEMs take advantage of the decrypted logs to gather information about the targeted device and its hosting system. These information serve to create target profiles that interest se-

Boudguiga, A., Stan, O., Sedjelmaci, H. and Carpov, S.
Homomorphic Encryption at Work for Private Analysis of Security Logs.
DOI: 10.5220/0008969205150523
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 515-523
ISBN: 978-989-758-399-5; ISSN: 2184-4356

curity services providers. Or, they can undermine a company brand image by selling its alert reports to its concurrent companies. It is in this context that homomorphic encryption is useful to provide logs privacy and it is an attractive alternative to classical encryption. As such, SIEM will analyze encrypted data without access to sensitive information about the incident origin and nature.

Homomorphic encryption is also interesting for forensics use case. Let us consider an incident that targets a device deployed in many systems simultaneously. The log generator encrypts its logs with a homomorphic scheme to prevent the leakage of sensitive information during the investigation of the incident. As such, an adversary with access to the encrypted logs gets no information that will allow her to run a massive attack on many companies using the same vulnerable device.

Finally, current regulation efforts regarding private data management such as the EU 2016/679 General Data Protection Regulation (GDPR)[1] (European Parliament and Council, 2016) are incentives for homomorphic encryption application to logs privacy. Indeed, GDPR article 4.1. considers logs content as private data, and logs generators consent is compulsory for logs collection and classification. However, GDPR recital 49 indicates that consent is not necessary if data processing is vital to provide system security against malicious activities. Fortunately, homomorphic encryption is a good compromise between logs privacy provision and logs processing for malicious activities detection.

Contributions – In this work, we investigate homomorphic encryption usage during log management to provide private data protection. We first characterize the sensitive fields of a log entry that must not be disclosed to an adversary. Then, we make log generators transcrypt their logs before their transmission to log analyzers (i.e., SIEM). The latter processes encrypted logs and transmits the result to SOC. The SOC has only to decrypt the received result to get SIEM analysis report. Finally, we study the feasibility of our proposal by simulating a homomorphic log analysis with NSL-KDD data set.

Paper Organization – Section 2.1 reviews the main components of a log management architecture. Section 2.2 presents the related works on privacy-preserving log management. Section 3 specifies our privacy-preserving log management protocol that relies on homomorphic encryption. Section 4 discusses the performance of our proposal. Finally, Section 5 concludes the paper with future improvements.

[1]EU GDPR is applicable since May 2018 in all EU member states to harmonize privacy laws across Europe.

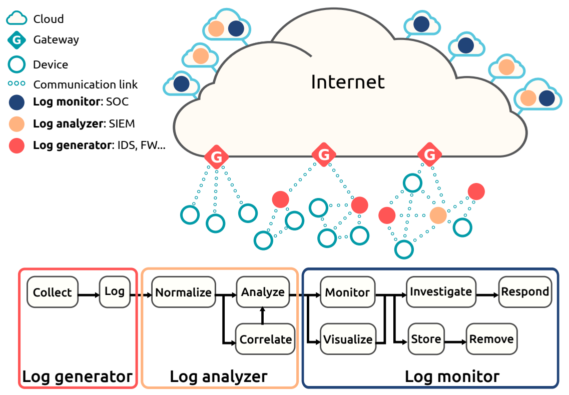

Figure 1: Log management components.

2 BACKGROUND AND TOOLS

In this section, we describe the log management architecture. In addition, we review the state of the art on privacy preserving log management. Finally, we introduce homomorphic encryption and the notations followed in this work.

2.1 Log Management Architecture

In this section, we review the main components that take part in log management architecture. To do so, we consider the abstract network architecture[2] described in Figure 1. It contains a set of heterogeneous devices. A device supports several types of communication: Device-to-Device, Device-to-Cloud and Device-to-Gateway. Some devices such as intrusion detection systems (IDS), firewalls, gateways or servers create logs to store information about their internal state or to monitor their hosting network state. We refer to these devices as *log generators* (Scarfone and Souppaya, 2006). Log generators are synchronized to ensure that the logs they produced are well timestamped. Indeed, timestamping is compulsory for logged events reordering during incident investigation.

According to (Allison et al., 2013), the most critical information for logging are related to: authentication and authorization, system configuration, network traffic, assets, malware and critical system failures. In this paper, we only consider alerts and logs coming from IDS for sake of simplicity. IDS report malicious and suspicious activities detected on a host machine (HIDS) or a network (NIDS) (Dali et al., 2015). Note that in practice, IDS implement

[2]Note that the considered architecture covers major IoT use cases: smart home, smart grids, factory 4.0 and intelligent transportation systems.

a signature-based or/and an anomaly-based detection policy. Signature-based IDS rely on a set of a predefined rules and patterns to detect misbehaviors. Meanwhile, anomaly-based IDS use machine learning algorithms to detect behavior deviation from a reference state (Aburomman and Reaz, 2017).

The *log analyzer*, namely the SIEM can be agentless or agent-based (Scarfone and Souppaya, 2006). Agentless SIEM receives logs from log generators with no need for the installation of a dedicated log transfer software in these generators. Meanwhile, agent-based SIEM relies on clients installed in log generators for logs transmission periodically. SIEM is in charge of logs normalization. Indeed, SIEM manages different types of logs with inconsistencies regarding logs content, format and timestamps. Normalization is compulsory to rewrite logs with inconsistencies under a unique form. Normalization extracts from logs *features* such as transport protocol type (e.g., TCP or UDP), port number, IP addresses and timestamps. These features are the serialization tokens of the new log format.

SIEM analyses logs content by looking for patterns of simple attacks using a rule-based or a learning-based policy. In addition, SIEM correlates logs in order to reduce the number of false positive, to remove redundant alerts and to capture linked alerts that belong to same attack pattern (Beng et al., 2013). By false positive, we refer to events not relevant for the analysis or erroneous events signaled as threats. A simple correlation algorithm consists in clustering logged events by comparing their features similarities (e.g., same source IP address and port). Other correlation algorithms rely on machine learning with a training phase to detect complex alerts.

Furthermore, SIEM is in charge of logs storage and disposal. Logs can be centralized at one SIEM or distributed over many SIEMs. In addition, SIEM can be installed in a local area network or provided as a Cloud service by a security provider (Figure 1). SIEM distribution and installation in the cloud is more advantageous than SIEM local centralization. Indeed, Cloud SIEM thwarts the risk of logs loss during an attack against the local network. It backs-up and protects logs even if their generators are compromised, their network is attacked and their local logs databases are cleared.

SIEM interacts with a Security Operations Center (SOC), i.e. *log monitor* (Scarfone and Souppaya, 2006), to define appropriate reactions to detected incidents on devices. SOC relies on a team of incident analysts to manage forensics tasks, security countermeasures and updates deployment, and incident reporting. In addition, SOC provides a dashboard for alert visualization. In this work, we make a clear distinction between SIEM and SOC functions as presented in Figure 1. SIEM is a software service which runs automated processes. Meanwhile, SOC is an infrastructure that gathers security experts which are in charge of security threats investigation and response. In literature, SIEM and SOC are sometimes grouped in unique entity (referenced to by SIEM).

2.2 Private Log Analysis

With the democratisation of the cloud and the increase complexity of the SIEM tools, a natural solution is to migrate such systems to the cloud. As such, the analysis of log systems can be deported and available as a SaaS (Software as a Service) on a remote server or a Cloud platform (e.g.,(Lin et al., 2013)). In such a context, there are raising issues regarding the security of the log transmission, treatment and storage.

Some commercial SIEM solutions such as IBM QRadar store the received logs obfuscated and hashed. Other SIEMs allow the log encryption, using a symmetric scheme such as AES, to secure the data transmission between their different software components. Meanwhile, the approaches based on Syslog (e.g., syslog-sign, syslog-pseudo, reliable-syslog), the standard for the network wide logging, does not ensure the logs confidentiality in transit or treatment.

In (Ray et al., 2013), the authors propose secure protocols for the anonymous upload, retrieval and deletion of logs data over the Tor network. However, their logging client is dependent on the chosen operating system and logs privacy is not totally addressed since the logs can be identified through their tag values.

To the best of our knowledge, there are no existing works on the confidentiality of log data, both during transmission and analysis, using homomorphic encryption techniques.

2.3 Homomorphic Encryption

Homomorphic Encryption (HE) schemes allow to perform computations directly over encrypted data. That is, with a fully homomorphic encryption scheme E, we can compute $\mathsf{E}(\mathsf{m}_1 + \mathsf{m}_2)$ and $\mathsf{E}(\mathsf{m}_1 \times \mathsf{m}_2)$ from encrypted messages $\mathsf{E}(\mathsf{m}_1)$ and $\mathsf{E}(\mathsf{m}_2)$. The first constructions of HE schemes, allowing either multiplication or addition over encrypted data date back to the seventies (Rivest et al., 1978). Then, in 2009, Gentry (Gentry et al., 2009) proposed the first Fully Homomorphic Encryption (FHE) scheme able to evaluate an arbitrary number of additions and multiplications over encrypted data.

Starting from Gentry breakthrough, many Somewhat HE and FHE schemes have been proposed in literature (Brakerski et al., 2012; Fan and Vercauteren, 2012a; Van Dijk et al., 2010; López-Alt et al., 2012; Chillotti et al., 2016; Cheon et al., 2016). In (Acar et al., 2017), FHE schemes are classified into four main families: ideal Lattice-based schemes (Gentry et al., 2009), schemes over integers (Van Dijk et al., 2010), schemes based on the Learning With Error (LWE) problem or its ring variant (RLWE) (Brakerski and Vaikuntanathan, 2011; Brakerski et al., 2012; Chillotti et al., 2016; Cheon et al., 2016) and NTRU-like schemes (López-Alt et al., 2012).

In practice, a public key encryption scheme HE = ($\mathsf{HE.Keygen}$, $\mathsf{HE.Enc}$, $\mathsf{HE.Dec}$, $\mathsf{HE.Eval}$) is defined by the following probabilistic polynomial-time algorithms with respect to the security parameter k:

- $(\mathsf{pk}, \mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{HE.Keygen}(1^{\mathsf{k}})$: outputs an encryption key pk, a public evaluation key evk and a secret decryption key sk. The evaluation key is used during homomorphic operations. evk corresponds to the relinearization key in levelled homomorphic schemes such as BFV (Fan and Vercauteren, 2012a) or to the bootstrapping key in gate boostrapped schemes such as TFHE (Chillotti et al., 2016).
- $\mathsf{c} \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(\mathsf{m})$: encrypts a message m into a ciphertext c using the public key pk.
- $\mathsf{m} \leftarrow \mathsf{HE.Dec}_{\mathsf{sk}}(\mathsf{c})$: decrypts a message c into a plaintext m using the public key sk.
- $\mathsf{c_f} \leftarrow \mathsf{HE.Eval}(\mathsf{f}, \mathsf{c}_1, \ldots, \mathsf{c_k})$: evaluates the function f on the encrypted inputs $\mathsf{c}_1, \ldots, \mathsf{c_k}$ using the evaluation key evk.

Nowadays, we dispose of several FHE schemes (e.g. BFV, TFHE, CKKS (Cheon et al., 2016), etc.), which can be mixed together using the CHIMERA framework (Boura et al., 2018). These scheme have interesting timing performances which allowed to implement interesting use cases such as evaluating simple neural networks (Bourse et al., 2017; **?**).

As for the overhead induced by the size of the homomorphic ciphertexts during their transmission and storage, we can use transciphering (Canteaut et al., 2015). This cryptographic technique changes the data encryption algorithm from a classical symmetric encryption to a HE scheme, without decrypting the data. Let m be a plaintext, SYM a symmetric scheme with key k, $\mathsf{SYM.Enc_k}(\mathsf{m})$ the encryption of m with SYM, and HE a homomorphic encryption scheme. With the transciphering, it is enough to run in homomorphic domain the decryption circuit of $\mathsf{SYM.Dec}$ using the homomorphic encryption of the symmetric key $\mathsf{HE.Enc}_{\mathsf{pk}}(\mathsf{k})$ to obtain the message encrypted with pk:

$$\mathsf{HE.Eval}_{\mathsf{evk}}(\mathsf{SYM.Dec}_{\mathsf{HE.Enc}_{\mathsf{pk}}(\mathsf{k})}(\mathsf{SYM.Enc_k}(\mathsf{m}))) = \mathsf{HE.Enc}_{\mathsf{pk}}(\mathsf{m})$$

2.4 Notations

In the following sections, we denote vectors by bold letters, for example $\mathbf{x}$. Each vector $\mathbf{x}$ of n elements can be represented as: $\mathbf{x} = (\mathsf{x}_1, \ldots, \mathsf{x_n})$. The transpose of a vector $\mathbf{x}$ is denoted $\mathbf{x^t}$. As such the dot product between two vector $\mathbf{x}$ and $\mathbf{y}$ is expressed as: $<\mathbf{x}, \mathbf{y}>= \mathbf{x^t}.\mathbf{y}$.

3 HE-BASED LOG ANALYSIS

In this section, we first present our considered threat model. Then, we discuss the requirements of log management to support homomorphic encryption operations. Finally, we specify our privacy preserving scheme for log exchange and treatment.

3.1 Threat Model

In this work, we consider a honest-but-curious model. In this model, many entities $(\mathsf{e}_1, \ldots, \mathsf{e_n})$, having as secret information $(\mathsf{s}_1, \ldots, \mathsf{s_n})$, participate to a protocol P to compute some function $\mathsf{F}(\mathsf{s}_1, \ldots, \mathsf{s_n})$. Each entity $\mathsf{e}_{\mathsf{i}, \mathsf{i} \in [1, \mathsf{n}]}$ is honest and must follow each step of P. However, $\mathsf{e}_{\mathsf{i}, \mathsf{i} \in [1, \mathsf{n}]}$ is curious. That is, $\mathsf{e}_{\mathsf{i}, \mathsf{i} \in [1, \mathsf{n}]}$ will try to find information about other entities secrets $\mathsf{s}_{\mathsf{j}, \mathsf{j} \neq \mathsf{i}}$. P is secure in the honest-but-curious model if each $\mathsf{e}_{\mathsf{i}, \mathsf{i} \in [1, \mathsf{n}]}$ has no other information than $\mathsf{F}(\mathsf{s}_1, \ldots, \mathsf{s_n})$ at the end of the protocol.

In this work, log generators and the log analyser (SIEM) are assumed honest-but-curious while the log monitor (SOC) is assumed a trusted entity. A log generator may be interested in the features of other log generators. Meanwhile, the SIEM can be interested in recovering all the private features of log generators.

The SOC, considered an honest trusted party, generates its public, evaluation and secret keys (pk, evk and sk, respectively). Then, the SOC shares pk with the SIEM and the log generator. In addition, it provides evk to the SIEM. The latter is in charge of analyzing the log inputs and returns encrypted evaluation to the SOC.

We focus here on providing confidential logs analysis and do not consider other security properties such as entities authentication or message integrity vali-

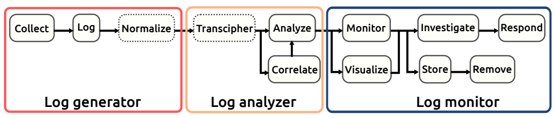

Figure 2: Log management components update for HE support.

dation. We consider that "classical"[3] cryptographic mechanisms are sufficient to provide such properties.

3.2 Requirements

In this section, we introduce the modifications needed for homomorphic encryption usage in our log management architecture. Our major concern is thwarting the performance bottleneck of existing homomorphic encryption schemes, both in terms of memory usage for log transmission and treatment, as well as execution time for log processing. To do so, we must choose with scrutiny the log fields to be encrypted. In addition, we pay attention to the operations that SIEM will run over encrypted data. Figure 2 presents the proposed extensions to the log management architecture. They concern the normalization step and transciphering of logs data.

First, we delegate SIEM logs normalization step to log generators. As such, the latter will be in charge of rewriting their logs in a common format defined by SIEM. Consequently, SIEM is relieved from the cumbersome normalization of partially encrypted file. Second, we propose to encrypt only the sensitive fields of a log entry to reduce the volume of HE encrypted data. That is, we distinguish between sensitive fields and just informative fields in a log entry. Sensitive fields can disclose information about a log generator such as: MAC address, IP address (if public), transport protocols (with their flags) and targeted services (ports). However, informative fields do not provide any information about a log generator, unless combined with a sensitive field. For example, a packet loss rate is considered as informative field. It is valuable only when combined with a service port and an IP address.

Finally, SIEM is transciphering the log files l, encrypted partially with a random symmetric key k, into files encrypted with SOC homomorphic public key $\mathsf{HE.pk_{SOC}}$. That is, SIEM is computing $\mathsf{HE.Enc_{pk}(l)}$ from the symmetric encryption $\mathsf{SYM.Enc_k(l)}$ using $\mathsf{HE.Enc_{pk}(k)}$, sent by a log generator. Transciphering not only avoids to log generators the cumbersome homomorphic encryption, but also allows a gain of bandwidth because $\mathsf{SYM.Enc_k(l)}$ is smaller than $\mathsf{HE.Enc_{pk}(l)}$.

[3]By classical, we refer, for example, to message authentication codes to provide message integrity. Or, we can add nonces to avoid replay attacks and counters to detect denial of services attacks.

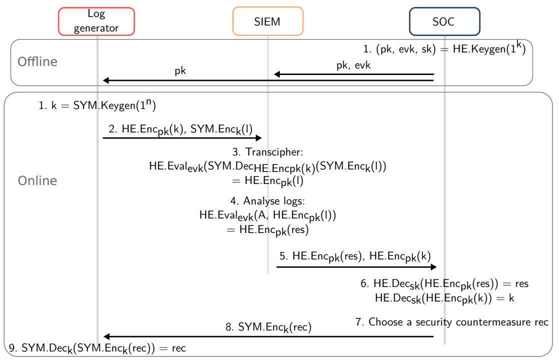
Log generator
SIEM
SOC
Offline
1. (pk, evk, sk) = HE.Keygen(1^k)
pk, evk
pk
Online
1. k = SYM.Keygen(1^n)
2. HE.Enc_pk(k), SYM.Enc_k(l)
3. Transcipher:
HE.Eval_evk(SYM.Dec_HE.Encpk(k)(SYM.Enc_k(l))
= HE.Enc_pk(l)
4. Analyse logs:
HE.Eval_evk(A, HE.Enc_pk(l))
= HE.Enc_pk(res)
5. HE.Enc_pk(res), HE.Enc_pk(k)
6. HE.Dec_sk(HE.Enc_pk(res)) = res
HE.Dec_sk(HE.Enc_pk(k)) = k
7. Choose a security countermeasure rec
8. SYM.Enc_k(rec)
9. SYM.Dec_k(SYM.Enc_k(rec)) = rec

Figure 3: HE privacy preserving log management protocol.

3.3 Private Log Analysis with HE

We detail in this section, our proposed protocol for the privacy protection of logs content. Our main idea is to make SIEM analyze logs encrypted partially with a HE scheme. In Figure 3, we show the main three entities of the log management architecture: the log generator, the SIEM and the SOC. Log generation step is carried on by an IDS or a firewall or any other information system. The SIEM and the SOC are distinct entities, hosted on remote servers or provided as cloud services.

First, the SOC runs the key generation algorithm ($\mathsf{HE.Keygen}$) and gets pk, evk and sk. Then, it transmits pk to the log generator and the SIEM, and evk to the SIEM only. The log generator chooses a symmetric key k for a homomorphic-friendly symmetric cyrptosystem (Canteaut et al., 2015). This key can be renewed periodically.

In step 2, the log generator encrypts this symmetric key using the public homomorphic public key $\mathsf{HE.Enc_{pk}(k)}$ and also encrypts the generated (and previously normalized) logs l with the symmetric key $\mathsf{SYM.Enc_k(l)}$. The encrypted logs and the homomorphically encrypted symmetric key are send to the SIEM. Note that the transmission of the encrypted key $\mathsf{HE.Enc_{pk}(k)}$ is independent of the transmission of the encrypted logs but must occur before step 3.

In step 3, the SIEM uses $\mathsf{HE.Enc_{pk}(k)}$ and $\mathsf{SYM.Enc_k(l)}$ to obtain the logs $\mathsf{HE.Enc_{pk}(k)}$ encrypted in the homomorphic domain (without having access at any moment to the clear logs).

Step 4 consists in the analysis of the logs in homomorphic format with the algorithm A. The computation of A can consist of a machine learning algorithm or a rule-based process. The homomorphic result of this analysis $\mathsf{HE.Enc_{pk}(r)}$ is send to the SOC in step 5. The protocol also requires the transmission of $\mathsf{HE.Enc}_{\mathsf{pk}}(\mathsf{k})$ to the SOC.

The SOC decrypts the result of the analysis using the secret key sk (step 6) and, based on the obtained result res, sends an adapted security countermeasure (step 7), noted rec, to the log generator (step 8). The result can be a classification of an event as unknown or dangerous, and the countermeasure can be a patch deployment or an alert report. Finally, the log generator decrypts rec and takes the appropriate actions (step 9).

4 EXPERIMENTAL RESULTS

We propose to use a Support Vector Machine (SVM) with a linear kernel for NSL-KDD inputs classification using homomorphic encryption. NSL-KDD dataset (Tavallaee et al., 2009), is an updated version of the KDDCUP'99 dataset. It is a public dataset for testing network-based anomaly detection systems. We use it to simulate logging inputs from an IDS. In the following, we review SVM algorithm. Then, we present the performance of the classification of encrypted NSL-KDD inputs, obtained with the Microsoft `SEAL` library version 3.3 (SEAL, 2019).

4.1 Support Vector Machines

Support Vector Machines (SVM) is a supervised learning algorithm used for regression and classification. The SVM classifier determines a set of vectors called support vectors that serve to construct a hyperplan in the feature spaces. In our simulation, we use SVM as a binary classifier to predict if an NSL-KDD input comes from a normal traffic or an anomaly.

Given the training dataset $(\mathbf{x_i}, y_i)_{i\in[\![1,n]\!]}$ where $\mathbf{x_i} \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$, we want to find with SVM the hyperplane that maximizes the margin: $\mathbf{w^t}.\mathbf{x} + b = 0$, where $\mathbf{w}$ is a normal vector (i.e., the hyperplan equation) and the parameter b is an offset.

We solve the following optimization problem to find the optimal hyperplan:

$$\begin{cases} \min(\frac{\|\mathbf{w}\|^2}{2} + C\sum_{i=1}^{n}\varepsilon_i) \\ s.t.\ \ y_i(\mathbf{w^t}.\mathbf{x_i} + b) \geq 1 - \varepsilon_i,\ \ \varepsilon_i \geq 0\ \ \forall i \in [\![1,n]\!] \end{cases} \tag{1}$$

$\sum_{i=1}^{n}\varepsilon_i$ relaxes the constraints on the learning vectors, and C is a constant that controls the tradeoff between the number of misclassifications and the margin maximization. Equation 1 can be dealt with the Lagrange multiplier (Schölkopf and Smola, 2003):

$$\begin{cases} \max(L(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K(\mathbf{x_i}, \mathbf{x_j})) \\ s.t.\ \ \sum_{i=1}^{n}\alpha_i y_i = 0,\ \ 0 \leq \alpha_i \leq C\ \ \forall i \in [\![1,n]\!] \end{cases} \tag{2}$$

K is the kernel function and $\alpha_{i,\forall i\in[\![1,n]\!]}$ are the Lagrange multipliers. According to the condition of Karush-Kuhn-Tucker (KKT) (Karush, 2014; Kuhn and Tucker, 1951), the $\mathbf{x_i}$ that correspond to $\alpha_i > 0$ are called support vectors (SVs).

Once the solution to Equation 2 is found, we get: $\mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i \mathbf{x_i}$. Thus the decision function can be written as: $\mathsf{sgn}(\mathbf{w^t}.\mathbf{x} + b)$. If the returned sign is positive than $\mathbf{x}$ belongs to the class above the separating hyperplan.

In this work, we use an SVM classifier with a linear kernel: $K(\mathbf{x_i}, \mathbf{x_j}) = <\mathbf{x_i}, \mathbf{x_j}>$. We do so to simplify the calculus with vectors of encrypted features as presented in the upcoming section.

4.2 Performances

Training. We use `Python 3`, the `scikit` library and the SVM available package `sklearn.svm` for the dataset preprocessing and for the SVM algorithm training.

For the training step, we use 80% of the 25192 labelled records of `KDDTrain+_20Percent` dataset. Each entry of this dataset is a vector of 42 features. First, we analyse each feature and its range (minimum, maximum and standard deviation). We also ensure that no feature is missing in the dataset. Then, we convert all categorical attributes into numerical ones, using label encoding. We scale them using the `MinMaxScaler` (except, of course, the class attribute) of the dataset. Finally, we train an SVM model with a linear kernel.

Classification. Once the training is finished, we run the prediction on the test set (i.e. the remaining 20% of `KDDTrain+_20Percent`). We get an accuracy value of 95.653%. Accuracy is defined as the ratio between the predicted values and the real ones. We also extract the normal vector $\mathbf{w}$ and the offset b.

Classification of Encrypted Inputs. To classify a vector $\mathbf{x} = (x_1, \ldots, x_n)$ without revealing any x_i, we first encrypt it as $\mathbf{x_{enc}}$ with the API for `BFV` encryption scheme (Fan and Vercauteren, 2012b) from `SEAL` library.

$$\begin{aligned} \mathbf{x_{enc}} &= (\mathsf{HE.Enc}_{\mathsf{pk}}(x_1), \ldots, \mathsf{HE.Enc}_{\mathsf{pk}}(x_n)) \\ &= \mathsf{HE.Enc}_{\mathsf{pk}}(\mathbf{x}) \end{aligned}$$

Then, using the linear SVM prediction formula, we compute $\mathsf{res_{enc}} = \mathbf{w^t}.\mathbf{x_{enc}} + b$, where $\mathbf{w^t}$ and b are in clear while $\mathbf{x_{enc}}$ is encrypted. We obtain an encrypted result $\mathsf{res_{enc}}$. Then, we decrypt it as $\mathsf{res} = \mathsf{HE.Dec}_{\mathsf{sk}}(\mathsf{res_{enc}})$, and we check the sign of res to get the final prediction result. In practice, $\mathbf{x_{enc}}$

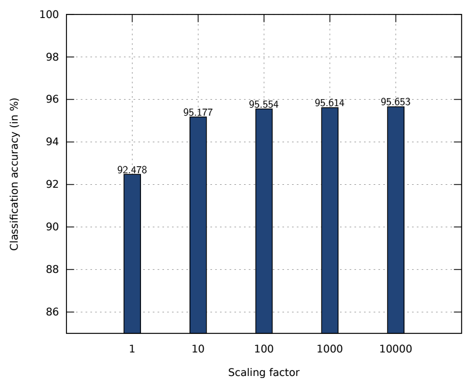

Figure 4: Prediction accuracy with respect to features scaling from reals to integers.

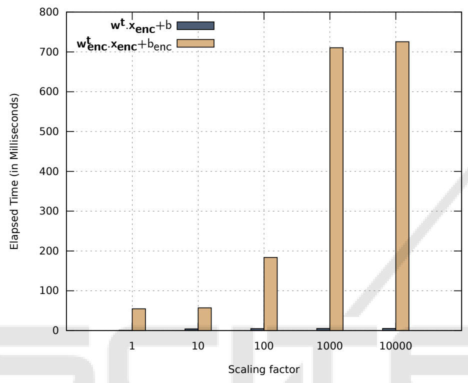

Figure 5: Prediction time with respect to features scaling from reals to integers.

corresponds to a private input from a log generator. Meanwhile, $\mathbf{w^t}$ is the public data of the SIEM. As such, SIEM computes $\mathsf{res_{enc}}$ and sends it to the SOC for decryption.

In BFV cryptosystem, a plaintext is encoded as a polynomial of degree n with integer coefficients in $\mathbb{Z}_\mathsf{t}$. Meanwhile, a ciphertext is defined as a set of two (or more) polynomials of the same degree n with integer coefficients in $\mathbb{Z}_\mathsf{q}$ where $\mathsf{q} \gg \mathsf{t}$.

To encrypt $\mathbf{x}$ with BFV, we compute appropriate scaling factor $\mathsf{s_f}$ to transform the real values of $\mathbf{w^t}$, $\mathbf{x}$ and b into integers. To get an integer $\mathsf{x}_\mathbb{Z}$ from a real value $\mathsf{x}_\mathbb{R}$, we multiply $\mathsf{x}_\mathbb{R}$ by a scaling factor ranging in $\{1, 10, 100, 1000, 10000\}$. Then, we truncate the integer value using the floor operator $\mathsf{x}_\mathbb{Z} = \lfloor \mathsf{x}_\mathbb{R}.\mathsf{s_f} \rfloor$. Finally, we proceed with the classification of encrypted inputs. Our experiment confirms that we reach the same accuracy of a prediction with real numbers while using encrypted integers starting from $\mathsf{s_f} = 10000$ (as presented in Figure 4).

Figure 5 shows that prediction time for a unique NSL-KDD input ranges from 0.7ms to 5.5ms with respect to the scaling factor. Indeed, for each $\mathsf{s_f}$ in $\{1, 10, 100, 1000, 10000\}$, each plaintext attribute $\mathsf{w_i^t}$ is encoded as a polynomial with integer coefficients of size $\lceil \log_2(\mathsf{t}) \rceil$ in $\{8, 12, 18, 25, 32\}$ (as presented in Table 1). Meanwhile, each attribute $\mathsf{HE.Enc_{pk}(x_i)}$ of $\mathbf{x_{enc}}$ is encoded as a polynomial with integer coefficients of size $\lceil \log_2(\mathsf{q}) \rceil$. The size of coefficients impacts the calculus of the dot product between the ciphertext vector $\mathbf{x_{enc}}$ and the plaintext vector $\mathbf{w^t}$. Indeed with respect to the same security level (of 128 bits), the size of coefficients ($\lceil \log_2(\mathsf{q}) \rceil$) and the degree ($\mathsf{n}$) of the ciphertext polynomial depend on the size of the coefficents of the plaintext polynomial ($\lceil \log_2(\mathsf{t}) \rceil$) as presented in Table 1. The larger are the coefficients, the longer is the computation time. We can make encrypted predictions where not only $\mathbf{x}$ is encrypted but also $\mathbf{w^t}$ and b too. Computing $\mathsf{res_{enc}}$ from totally encrypted inputs is interesting when two SIEM that are managed by different providers, collaborate in alerts classification. However, these SIEM can require that their respective model $\mathbf{w^t}$ remains secret due to IP concerns. In this case, SIEM1 encrypts and sends $\mathbf{w1^t_{enc}}$ and $\mathsf{b1_{enc}}$ to SIEM2. Then, SIEM2 computes $\mathsf{res_{enc}} = \mathbf{w1^t_{enc}}.\mathbf{x_{enc}} + \mathsf{b1_{enc}}$.

Figure 5 shows that the prediction time increases drastically when all data are encrypted homomorphically. The prediction time ranges from 55ms to 725ms with respect to plaintext coefficients size (and so, with respect to the scaling factor size). In addition, the plaintext coefficients size impacts the polynomial degree n and the ciphertext modulus q in SEAL. Table 1 summarizes the values of n and q with respect to t while considering a security level of 128 bits. Note that SEAL fixes the security parameters using the homomorphicencryption.org security standard.

Table 1: BFV parameters in SEAL.

Prediction equation	n	$\lceil \log_2(\mathsf{t}) \rceil$	$\lceil \log_2(\mathsf{q}) \rceil$
$\mathsf{res_{enc}} = \mathbf{w^t}.\mathbf{x_{enc}} + \mathsf{b}$	1024	8	27
	2048	12	54
		18	
		25	
		32	
$\mathsf{res_{enc}} = \mathbf{w^t_{enc}}.\mathbf{x_{enc}} + \mathsf{b_{enc}}$	2048	8	54
		12	
	4096	18	109
	8192	25	218
		32	

5 CONCLUSION

In this work, we specified a privacy-preserving log management architecture thanks to the use of homomorphic encryption. We made SIEM analyse encrypted log inputs using a linear SVM classifier. The first preliminary experimental results, obtained with

NSL-KDD dataset and SEAL homomorphic library, are promising in terms of accuracy.

In the future, we plan to ameliorate the execution times of the prediction over encrypted data. We will investigate the use of the plaintext space of BFV scheme to compute the prediction results of many inputs simultaneously. Second, we will use other homomorphic encryption schemes, namely TFHE (Chillotti et al., 2016) or CKKS (Cheon et al., 2016) for our classification. TFHE and CKKS are interesting as they use floating point numbers. Second, we intend to implement and test other classification algorithms (e.g. neural networks) for the analysis of the encrypted logs.

REFERENCES

Aburomman, A. A. and Reaz, M. B. I. (2017). A survey of intrusion detection systems based on ensemble and hybrid classifiers. *Computers & Security*, 65:135 – 152.

Acar, A., Aksu, H., Uluagac, A. S., and Conti, M. (2017). A survey on homomorphic encryption schemes: Theory and implementation. *arXiv preprint arXiv:1704.03578.*

Allison, J., Evans, J., Filkens, B., Moye, O., Northcutt, S., Read, J., Torres, A., and Wityszyn, M. (2013). The 6 Categories of Critical Log Information.

Beng, L. Y., Ramadass, S., Manickam, S., and Fun, T. S. (2013). A comparative study of alert correlations for intrusion detection. In *2013 International Conference on Advanced Computer Science Applications and Technologies*, pages 85–88.

Boura, C., Gama, N., Georgieva, M., and Jetchev, D. (2018). Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. Cryptology ePrint Archive, Report 2018/758. https://eprint.iacr.org/2018/758.

Bourse, F., Minelli, M., Minihold, M., and Paillier, P. (2017). Fast homomorphic evaluation of deep discretized neural networks. Cryptology ePrint Archive, Report 2017/1114. https://eprint.iacr.org/2017/1114.

Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (2012). (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325.

Brakerski, Z. and Vaikuntanathan, V. (2011). Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer.

Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., and Sirdey, R. (2015). Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. Cryptology ePrint Archive, Report 2015/113. https://eprint.iacr.org/2015/113.

Cheon, J. H., Kim, A., Kim, M., and Song, Y. (2016). Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Report 2016/421. https://eprint.iacr.org/2016/421.

Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. (2016). Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*, pages 3–33. Springer.

Dali, L., Bentajer, A., Abdelmajid, E., Abouelmehdi, K., Elsayed, H., Fatiha, E., and Abderahim, B. (2015). A survey of intrusion detection system. In *2015 2nd World Symposium on Web Applications and Networking (WSWAN)*, pages 1–6.

European Parliament and Council (2016). REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).

Fan, J. and Vercauteren, F. (2012a). Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144.

Fan, J. and Vercauteren, F. (2012b). Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144. https://eprint.iacr.org/2012/144.

Gentry, C. et al. (2009). Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178.

Jarpey, G. and McCoy, R. S. (2017). Chapter 1 - what is a security operations center? In Jarpey, G., , and McCoy, R. S., editors, *Security Operations Center Guidebook*, pages 3 – 10. Butterworth-Heinemann, Boston.

Karush, W. (2014). *Minima of Functions of Several Variables with Inequalities as Side Conditions*, pages 217–245. Springer Basel, Basel.

Kuhn, H. W. and Tucker, A. W. (1951). Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley, Calif. University of California Press.

Limmer, T. and Dressler, F. (2008). Survey of event correlation techniques for attack detection in early warning systems.

Lin, X., Wang, P., and Wu, B. (2013). Log analysis in cloud computing environment with hadoop and spark. In *2013 5th IEEE International Conference on Broadband Network Multimedia Technology*, pages 273–276.

López-Alt, A., Tromer, E., and Vaikuntanathan, V. (2012). On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM.

Nathans, D. (2015). Chapter 1 - efficient operations: Building an operations center from the ground up. In Nathans, D., editor, *Designing and Building Security Operations Center*, pages 1 – 24. Syngress.

Ray, I., Belyaev, K., Strizhov, M., Mulamba, D., and Rajaram, M. (2013). Secure logging as a service—delegating log management to the cloud. *IEEE Systems Journal*, 7(2):323–334.

Rivest, R. L., Adleman, L., and Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180.

Scarfone, K. K. and Souppaya, M. P. (2006). NIST Special Publication - 800-92: Guide to Computer Security Log Management.

Schölkopf, B. and Smola, A. J. (2003). *A Short Introduction to Learning with Kernels*, pages 41–64. Springer Berlin Heidelberg, Berlin, Heidelberg.

SEAL (2019). Microsoft SEAL (release 3.3). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA.

Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6.

Van Dijk, M., Gentry, C., Halevi, S., and Vaikuntanathan, V. (2010). Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer.