PriVeil: Privacy-preserving threat information sharing

Tiago Filipe Ribeiro Gonçalves
Instituto Superior Técnico, Universidade de Lisboa, Portugal
tiago.r.goncalves@tecnico.ulisboa.pt

***Abstract*—Corporate cybersecurity is now more challenging than ever. Information infrastructures are susceptible attack targets, as their potential value gain is higher than that of domestic systems. Companies are unaware of all the security vulnerabilities present in their systems, because some of them are not known by anyone. Protection efforts can be leveraged by cooperating with other companies to identify and address security vulnerabilities together. However, trust cannot be fully placed on other parties by default, since their true intentions are not known. For this reason, existing solutions allow threat information to be shared yet using filters to prevent leakage of sensitive information. Nevertheless, relevant details can be lost during the anonymization process. All of these factors discourage companies to cooperate.**

This work proposes a threat information sharing system design that is able to match participants with identical security events, encouraging cooperation to identify the associated vulnerabilities while retaining full control over their private data. To this end, Privacy-preserving techniques like *Homomorphic Encryption* and *Secure Multi-party Computation* are used to assure that no sensitive information is unwillingly disclosed by the system. The system was evaluated qualitatively by fulfillment of a set of requirements and quantitatively through user and performance tests.

***Keywords*—Cybersecurity; Privacy; Trust; Vulnerabilities; Homomorphic encryption; Secure multi-party computation.**

I. Introduction

The world of corporate cybersecurity has changed in the last decade. Threats are now commodities, available for sale and purchase, and illegal business models are built around cybercrime tools [1]. Publicly available attack tools, like *exploit kits* and *botnets*, lower the entrance barrier for ill-minded perpetrators to commit illegal activities [2]. As a result, it is now harder to manage attack surfaces and effectively defend all of them. These activities hinder several industry sectors, leading countries to invest significant resources to cope with them. According to recent studies, cybercrime may now cost the world almost US$600 billion, or 0.8% of global gross domestic product (GDP) [3]. In an enterprise setting, where prevention is imperative, companies need to have timely access to valuable vulnerability and threat information about their systems. Unfortunately, companies do not know about all the vulnerabilities present in their systems as some of them remain undiscovered. Communities organized around specific industries or sectors often face actors with similar *modus operandi* which target the same types of systems and information. *Threat information sharing communities* may emerge as a defense mechanism against vulnerabilities, by making them more widely known. However, for a company to share this kind of information, it must be certain that its privacy is preserved, without leakage of sensitive information about the company's infrastructure. Companies may desire to exchange information without publicly disclosing their security issues, which implies the initial challenge of discovering other companies who have a similar problem. Threat information sharing is one of the most critical and effective mechanisms to use in the ever-changing world of cyberdefense.

The main objective of this work is to create a system to match participants with identical security vulnerabilities, encouraging users to cooperate to identify these issues while retaining full control over their private data. Companies are the only ones with control over their data and must retain their volition of only sharing contents as they desire, in function on the level of trust they choose to deposit on other users. PriVeil allows cyber threat information gathered by each company to be disseminated to a set of companies or a business sector, to achieve "herd immunity" [4].

II. Background

Cryptography can protect data and communications in the presence of third parties. For a given system or protocol, we can evaluate whether it possesses a desirable security service to measure its resilience against the corresponding type of threat. Security services can be divided into four fundamental categories: confidentiality, data integrity, authentication and non-repudiation. These services can be implemented by relying on a set of core cryptographic components: symmetric ciphers, hash functions, message authentication codes and asymmetric ciphers. The properties of cryptographic protocols are usually proven by relying on trust models.

Trust models are defined when developing a secure protocol, and include the set of properties of the network where the solution will operate and those of the adversary. A *semi-honest*, also called *honest-but-curious*, is a model adapted to Cloud computing deployments and defines an untrusted network where all parties are restricted to follow the protocol. Adversarial parties may analyze the data they have received and try to recover further insights from the computation, like the inputs of other players.

Privacy-Preserving Computation (PPC) appears as a solution to securely process sensitive data for more challenging trust models. PPC provides several mechanisms to securely

chain together different services without exposing sensitive data nor depositing unwarranted trust in third party entities. PPC encapsulates techniques like Homomorphic encryption (HE), Secure Multi-party Computation (SMPC) and Zero-knowledge proofs (ZKP).

Homomorphic encryption (HE) allows computation to be performed directly on ciphered data, without the need to decipher it, and providing the same final result. This result can only be accessed by those who know the secret key used to decipher the initial data. HE is divided between Fully Homomorphic Encryption (FHE) and Partially Homomorphic Encryption (PHE). FHE supports arbitrary computation on ciphertexts without the decryption key but has several limitations, like computation overhead from running very large and complex algorithms [5]. PHE allows arithmetic operations like sum, multiplication or both. Several real life use cases for HE already exist like electricity market forecast prediction [6] and privacy-preserving multi-keyword ranked search [7].

Secure multi-party computation (SMPC) allows several parties to jointly compute a function while keeping their input values private. The main disadvantages lie in the inherent computational cost and complexity, resulting from the recurrent communication between parties. Recent works present more efficient and scalable protocols, with a cloud computing setting in mind [8]. Some of SMPC applications include voting, market clearing price calculation [9], biomedical data distributed analysis [10] and tax fraud detection [11].

III. Threat Information Systems

Threat Information Sharing systems are able to store and correlate threat intelligence and vulnerability information from several sources and to disseminate cyber security indicators. Harnessing benefits from this information is only possible through the collection and analysis of data from threat intelligence reports and contextual information. Contextual information can be sensitive, as it may contain personal or confidential data that should be kept private. It is critical to be aware of the system's state to better understand the implied vulnerabilities and correlate gathered information. Nevertheless, openly disclosing these contents may result in unwanted privacy violations.

Vulnerabilities are usually distinguished by their Common Vulnerabilities and Exposures (CVE) identifier, a unique sequence for publicly known cybersecurity vulnerabilities, described by CVE List's entries. CVE entries contain the identifier, a brief description of the vulnerability and pertinent references to related content, like vulnerability reports mentioning and describing the problem. Security events, generated by software running on a machine, are collected by intrusion detection systems (IDS), firewalls, hardware sensors or user submitted reports, as depicted in Figure 1. *Security information and event management* (SIEM) systems use data inspection tools to centralize the storage and interpretation of logs and security events generated by software running on a machine.

Threat Information Sharing systems enable collaborative analysis, leveraging work done by other entities. Through a more effective diffusion of information about vulnerabilities, the chances of detecting and reacting to attacks as soon as possible are also amplified. A specific example is MISP [12] an open source project, funded by NATO, that provides a platform for inter-organizational information sharing. However, solutions to mitigate the exposure of sensitive data from a participant's system may be required, like to *anonymize* the identity of the information sharer, as proposed by Machado [13]. However, such procedure reduces the published information credibility, which is associated to the publisher notoriety, and an entity can also be identified by characteristics such as area of operation or network activity.

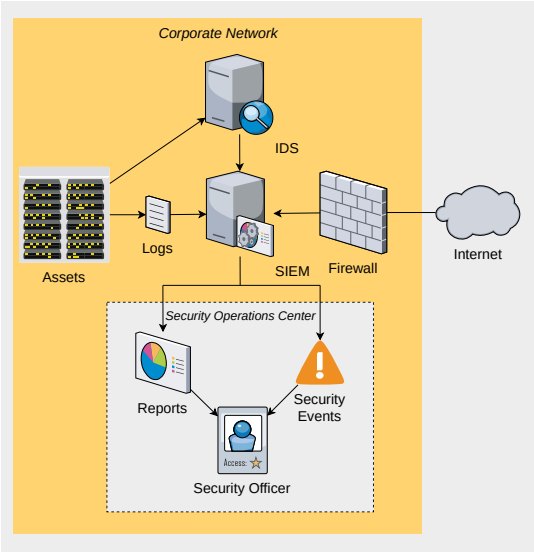

Fig. 1: Security operations architecture.

IV. Priveil

The main objective of this work is to design a system able to match users with a common vulnerability, allowing them to cooperate in order to determine the details of the security issue they share, while retaining control of their information disclosure. Participants must be willing to provide their sensitive, but unaltered, information to the platform, as illustrated by Figure 2, and likewise be able to gather trustworthy intelligence from other participants.

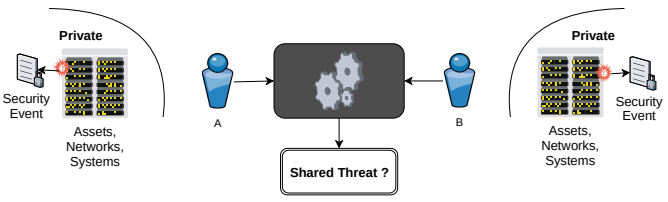

Fig. 2: Privacy-preserving Threat Information Sharing.

For simplicity, the collection of vulnerability intelligence and information to be submitted to the system will be des-

ignated as *Security Events*. The system should be able to preserve the identity of the involved companies and their submitted contents. Should there be a match of events, the involved companies will then decide the best course of action: whether to disclose further information or withhold it. By design, the system should suppress the risk of information disclosure without the companies' express consent. The ability for participants to cease information exchange at any given time is also a requirement.

A. Conceptual Design

To accomplish this goal, we propose PriVeil, which is structured in two phases, the *Concourse* and the *Conclave*. The *Concourse* encompasses the use of Homomorphic Encryption to guarantee that computation is possible over the ciphered security events stored by the system, concretely, to find elements that suggest a common vulnerability. We also introduce a secure environment, the *Conclave*, where users whose events matched are able to gather and incrementally disclose more information about their events. Secure Multi-party Computation will assure that these participants can use their sensitive data to compute this common task, while keeping their inputs private. This environment will be comprised of several stages, with the next disclosing more information than the previous, where users transition to the following stage only in they want to disclose more information about their events.

PriVeil, represented in Figure 3, is composed by:

- Concourse: users submit their security events to the system, which tries to find matching information between them. Users whose events matched are then notified, so that a *Conclave* may be assembled;
- Conclave: users, whose events matched, ensemble and try to understand more about the vulnerability they have in common.

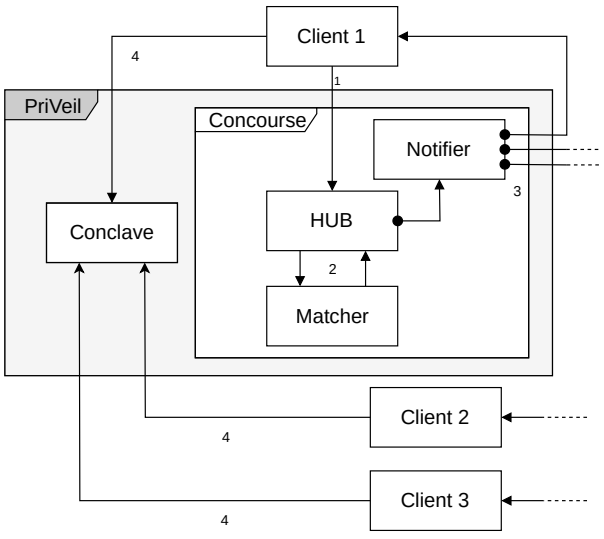

Fig. 3: System architecture.

The *Concourse* phase is divided in three subphases: *Submission*, *Matching* and *Notification*. During the *Submission* subphase, users will send their security events to the system [step 1 in Figure 3]. Ideally, all submitted information should be unaltered, without application of filtering nor anonymization techniques, in order to retain the maximum possible knowledge. Homomorphic Encryption will be used to allow computation over the ciphered data stored in the system, concretely, to find matching contents between vulnerability information [step 2], during the *Matching* subphase. When a match between events occurs, the *Notification* subphase is triggered, where the users who submitted these events will be notified [step 3]. This notification contains a proposal to take part in a meeting environment, just with the implied users. The notified users then decide whether or not to enter the *Conclave* environment [step 4].

During the *Conclave* phase, the environment will gradually evolve throughout several stages, as intended by the participants. More information will be successfully disclosed in each stage, in favor of better comprehension of the existing vulnerability. In this environment, the system will manage communication between users, assuring through SMPC that their inputs will only be revealed in accordance to the user's intent. These inputs will be incrementally disclosed in accordance to the user's will, who may choose to cease sharing information, thus, exiting the environment. Assuming that some users decide to leave the environment, subsets of the already present users are allowed to remain and continue the information exchange.

B. Requirements

The proposed system is divided in two phases: matching and disclosure. The *Matching phase* requires user information to be compared in order to find similar issues between them. This phase implies less information being shared between companies, but in a broader user base. The *Disclosure phase* requires user information to be shared with others while keeping individual inputs private. In this phase there is more information being shared between companies, but with a more restricted number of users.

The system handles three types of documents: events, published events and event match results. An *Event* is a complete security event produced by a user, containing a unique ID, a creation timestamp, status, tags, and CVE ids if referenced. A *Published Event* is a security event that will be sent through the network and stored by the system. An *Event Match Result* is data produced when the system determines similarity between a set of published events.

Users submit their *Events* to the platform, which will be responsible by finding similarities between them. Concretely, the *Matching phase* will try to match user's *Published Events*. When a match between events is found, an *Event Match Result* will be produced. The corresponding users will then be notified and may take part in the *Disclosure phase*.

The intended functionality and behavior of the system can be expressed as a list of requirements, which are divided into Functional Requirements and Quality Requirements.

1) Functional Requirements (FR:): constituting the specification of behavior of the proposed system, are the following:

- FR1: Users must be able to publish security events on the system.
- FR2: Users must be notified when one of their events matches those of other users.
- FR3: Users must be able to choose whether to take part in a disclosure phase.
- FR4: Users must be able to withdraw from a disclosure phase between stages.

2) Security Requirements (SR:): define the core principles to avoid the disclosure of sensitive information about the organization's infrastructure, and are the following:

- SR1: The events can only be disclosed by their publisher.
- SR2: The publisher of the event must not be traceable through the content of the published event.
- SR3: The published event must limit exposure of information about the publisher organization and its infrastructure.
- SR4: The published event must have enough information to allow matching with other events, regarding the same or similar vulnerabilities.
- SR5: The report match result must contain information to allow the event's publisher to make an informed decision about the desired course of action.
- SR6: The event match results should not allow the inference of the publishers identities.

3) Performance Requirements (PR:): detail the minimum required metric values for the system to be useful as a real platform. Considering that there was no previous system, similar in functionality, whose time constraint metrics could adapted to this project, the following values were defined in collaboration with domain experts[1]:

- PR1: The system must be able to match between two similar events in a timely way, below a threshold of 20 seconds.
- PR2: The system may present some incorrect event matches, but below a threshold of 5% false positives.
- PR3: The system may miss some correct event matches, but below a threshold of 5% false negatives.

We will mostly focus on exploring the values of PR1 and PR2 the time to match two similar events and the false positives threshold, respectively.

C. Attacker Model

In the considered setting, we define two base sets of properties to model different types of attackers. The first set of properties to be presented is related to the actions available to the attacker, while the second set constitutes the system insights he possesses. We consider the following set of properties to model different types of actions (A) available to the attackers:

[1]José Tolentino da Silva Martins, director, and Miguel Seabra, senior software engineer, from the cybersecurity department of NAV Portugal

- A_1: Eavesdrop all network traffic.
- A_2: Record and replay any number of messages between a given source and destination.
- A_3: Modify and suppress any message from a given source and destination.
- A_4: Read and record all data processed by a client.
- A_5: Pose as a legitimate client.
- A_6: Read and record all data processed by a server.
- A_7: Pose as a legitimate server.

Each of these properties may be present if the attacker is able to position himself in a convenient location, as depicted in Figure 4. The following set of properties is used to model

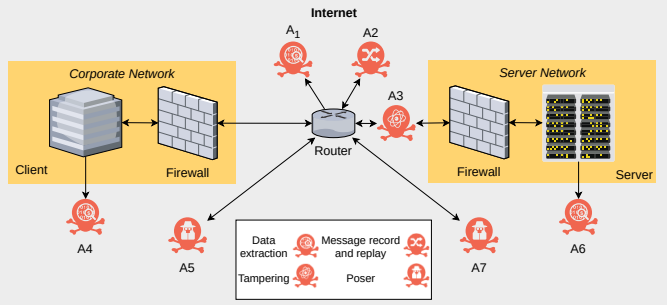

Fig. 4: Attacker positioning.

different types of insight (I) detained by the attackers, defining what they know:

- I_1: Cipher suites used by system, e.g., algorithms, key sizes and padding scheme.
- I_2: Configuration values of client and server applications, e.g., queue sizes, timeout values.
- I_3: Message origin and history of a client that communicated with the system.
- I_4: Mapping between client identity and its messages.
- I_5: Mapping between plaintext and ciphertext of a client's messages.
- I_6: Private keys of the client.

An attacker with properties A_1, A_2 and I_1 will be represented by $\{A_{1+2}, I_1\}$, for example.

V. Concourse Prototype

The *Concourse* is the PriVeil component that allows privacy-preserving security event sharing. During the Concourse, users will submit their ciphered events to the system, which will try to find matching information between them. Its goal is to match these events, with limited disclosure of information, and notify users whose events matched. For the design of the *Concourse* we chose an attacker model using the previously defined notation. The most powerful attacker which the system is able to withstand is defined by $\{A_{1+2+3+4+6}, I_{1+2+3+4}\}$.

A. Overview

The Concourse comprises three main application entities: server, client, and a message broker. The server is responsible for storing and matching ciphered events submitted by users.

Users own the original events which will be sent to the server through the client application. The message broker, through which all exchanged messages will pass, is responsible for assuring secure and reliable communication between clients and server.

When a user intends to submit an event to the platform he is responsible for normalizing its contents and ciphering it using the homomorphic public key. At the same time, the user also subscribes a pub/sub topic corresponding to the submitted event. When the message is received by the server, the ciphered event, digest and sender information are processed and stored. The sender information is a sequence which will be used to verify that a notification was for the user who received it, without any explicit reference to the event owner. If this event matches any other, a notification request is emitted, sent to the user who will receive it through the subscribed topic's message channel. The server will periodically try to match the received ciphered events and send a match notification to the topics associated with the matched events. PriVeil relies on both client and server applications to perform these tasks. The client application is responsible for processing security events and submitting them to the server. The server side is responsible for storing user submitted data, matching events and sending notifications.

B. Technical Architecture

PriVeil's client and server applications were developed in Java 1.8, from where Java KeyStore (JKS) is used for secure key and certificate storage. Protobuf was used as the serialization mechanism responsible for converting structured data into binary format to be passed between services. Javallier was chosen as PriVeil's Paillier homomorphic encryption library because it has the most usable, mature and accessible documentation, despite not being currently maintained.

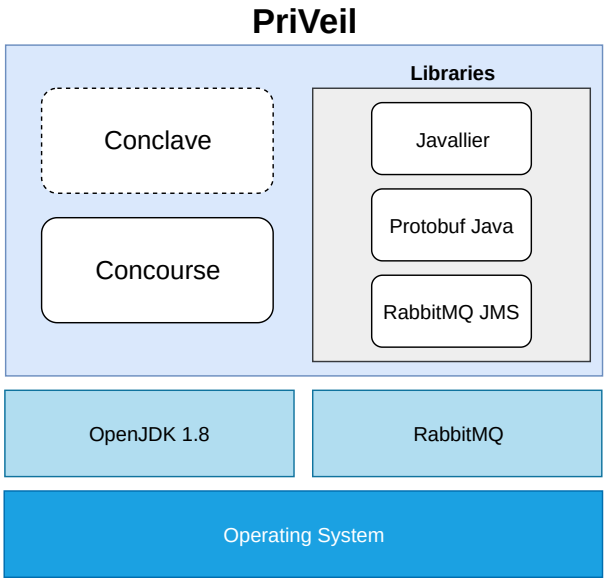

Fig. 5: PriVeil Architecture Stack.

All client-server communication is relayed through a message queuing service, both for direct messaging (point-to-point) and publisher/subscriber models. Message queues provide asynchronous communication, fault tolerance, granular scalability and decouple the several components of the underlying distributed systems. Transport Layer Security (TLS) is used to provide privacy and data integrity for all communications. Java Message Service (JMS) provides loosely coupled implementation of the asynchronous communication between the different components of the system, independently of the chosen JMS provider implementation. PriVeil's main components, its architecture, are represented in Figure 5. RabbitMQ was the chosen JMS provider, since it is one of the most used open-source message-broker software with JMS and TLS support.

C. Use of Secure Channels

TLS support will provide ciphered message traffic and allow server identity verification. TLS ciphered communication does not allow the attacker to understand the eavesdropped contents, provided by A_1. TLS avoids Man-in-the-Middle (MITM) attacks, since the client is able to validate the server certificate.

Replay attacks, A_2, are avoided through the use of TLS, if attempted by an external attacker capturing a communication session. TLS uses a MAC (Message Authentication Code), computed using the MAC secret and a sequence number for each session, which both cannot be adapted by the attacker. In some enterprise systems, the message transport is multi-hop, which can compromise the guarantee of freshness provided by TLS. Furthermore, a persistent register of message identification and freshness may be required, but the freshness information provided by TLS is transient. Therefore, a freshness token, F_S, will be added to each message, providing another mechanism, independent from the transport layer. This token contains the message creation timestamp and a randomly generated nonce. When a message is received, the server will verify if the timestamp is within an acceptance interval, and afterwards verify if any message with that nonce has been previously received. If that nonce has already been received, the message is dropped.

Minding A_3, TLS protects messages from being tampered due to the built-in MAC verification, which allows the receiver to detect that the information was meddled with. If TLS handshake messages are dropped by the attacker there is no way to notify either entity about what happened. Each one is only able to infer that the sent message was not received by the other given their lack of response. However, if an attacker drops any message during a TLS session, both entities will know that their communications were unintendedly terminated.

D. Event Submission

A_4 and A_6 require sensitive information to be carefully handled and that no plaintext is passed through the network. Furthermore, inputs received by clients and server should not contain details that allows explicit nor concrete identification of any user. An homomorphic public key is required by the

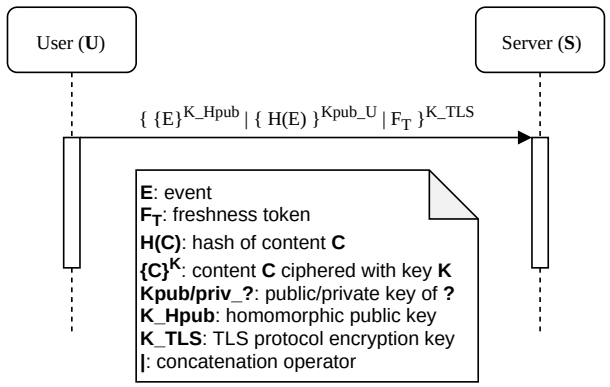

Fig. 6: Event submission protocol.

system and the corresponding keypair is assumed to have been generated in a secure location. The generated homomorphic public key, K_Hpub, will be accessible to the server and to all clients, while the private key, K_Hpriv, will only be accessible by the server, stored in a secure location and with limited usage. The event submission protocol, represented in Figure 6, occurs when a user publishes an event in the platform. The original event E is not published to the server, instead, relevant event fields like CVE id and tags are ciphered using the homomorphic public key. This list of ciphered fields forms the ciphered event and will be represented as $\{E\}^{K\_Hpub}$. The only user information that is submitted to the *Concourse* is the homomorphically ciphered event $\{E\}^{K\_Hpub}$ and the challenge $\{H(E)\}^{Kpub\_U}$.

After publishing an event, the client subscribes to its corresponding topic, $H(\{H(E)\}^{Kpub\_U})$. This topic identifier allows client and server to implicitly agree on a topic identifier without the need to exchange more messages or provide more information than the ciphered event and identity challenge. This topic value can be computed independently on both entities, considering that both have access to the challenge, and simply need to use a digest of it as topic. The client must locally store both the event E and the event digest $H(E)$, to be able to map the challenge, $H(\{E\}^{Kpub\_U})$, included in future match notification to his submitted event digest $H(E)$. Knowing $H(E)$ allows the client to reply to the server by sending $H(\{E\}^{K\_Hpub})$. Upon the initial event submission, the challenge response value, $H(\{E\}^{K\_Hpub})$, can also be generated and stored in a protected location, to be reused as challenge response when required later, with reduced overhead. When a ciphered event is received, the server will store $\{E\}^{K\_Hpub}$ together with $\{H(E)\}^{Kpub\_U}$ and associate them with the publisher/subscriber topic $H(\{H(E)\}^{Kpub\_U})$.

E. Event Matching

A list of digests for all, relevant, unique lowercase words found in the event will be created, hiding the word frequency of each term. Each digest is ciphered and the resultant list will be randomly reordered, hiding the original position of each word in the event. Only this list of ciphered terms will be sent to the server.

Given the homomorphic private key K_Hpriv, two ciphered attributes $a1$ and $a2$ match if:

$$dec_{K\_Hpriv}(a1 - a2) = 0 \tag{1}$$

Two ciphered events match if the required number of common attributes is reached. Considering events, $C1$ and $C2$, as two lists of ciphered attributes, the event size, represented by $c1$ and $c2$ respectively, corresponds to the number of attributes of each event.

When matching two events, $C1$ and $C2$, each attribute in $C1$ is compared with each attribute in $C2$ using Function 1. The required proportion of similarity for common attributes, RP, is adapted depending on the required level of certainty intended while matching events for a given deployment, where $RP \in]0, 1]$. By using this method, given two events where one has several more attributes than the other, they might never match, depending on the value of RP. The number of items they would need to have in common to be considered as matching would need to be greater than $RP \cdot max(c1, c2)$. Based on the list's size and the value of RP we are able to reduce the comparison overhead when trying to compare events whose sizes are too distinct. Two events may only match if their respective number of attributes, a and b, allows:

$$(1 - RP)(a + b) - (1 + RP)(\mid a - b \mid) \geq 0 \tag{2}$$

By relying on Function 1 referred as *match*, two ciphered events $C1$ and $C2$ of sizes $c1$ and $c2$, respectively, match if Function 2 is respected and if:

$$\sum_{i=0}^{c1} \sum_{j=0}^{c2} match(C1[i], C2[j]) \geq (RP \max(c1, c2)) \tag{3}$$

1) Match transitivity: allows sets of matched events to be merged if they have any previously matched event in common. Events that, even if not compliant with equation 2, may be matched through match transitivity, in order to further improve the matching algorithm results. For example, given 3 events, A, B and C, if A matches both B and C, but B does not match C, a matching set is formed with all these events during the match transitivity step, using A as the element common to the initial sets. The theoretical complexity of comparing two ciphered events corresponds to the cost of comparing all ciphered attributes of one event with the other's attributes, $\mathcal{O}(A^2)$, where A corresponds to the maximum number of attributes in an event. Considering that all events in the system will be compared with each other, the complexity of this operation will correspond to $\mathcal{O}(A^2 \cdot N^2)$, where N corresponds to the number of events being matched.

F. Anonymous Match Notification

Event owners are notified when their reports match. If a queryable system was used instead, where users queried the system for a specific type of event or tag, A_6 would be able to infer contextual information about the user. If a user were searching for information about a given vulnerability, he would

most likely have it or a similar one. In contrast, PriVeil's notification system solution places the communication initiative on the server side. According to the match notification protocol, in Figure 7, a client that receives a notification must prove to have submitted the matched event by answering to the identity challenge $\{H(E)\}^{Kpub\_U}$.

A notification will be sent to several users, despite being intended for a particular user, who is the only one able to understand its contents. By sending multiple notifications, a network eavesdropper is unable to know who was the user for which the notification was intended. When a user publishes an event, he also subscribes a topic corresponding to a segment of size s of the following hash value $H(\{H(E)\}^{Kpub\_U})$, a digest of the identity challenge, where $H(E)$ corresponds to that event's digest. When a match is found, the generated notifications are sent to all clients who subscribed that topic. Most of the clients will receive a notification message which is not destined to them. The client application is able to know that the message was not intended for the user after trying to decipher $\{H(E)\}^{Kpub\_U}$ and verifying that the information is unintelligible. Those who receive the notification and are able to correctly decipher it become aware that the message was destined to them. An external observer will be unable to infer the number of matched events or the corresponding event owners from the number of notification messages sent. By employing this concept of anonymous user notification, the server will only know to which users the message was sent and how many can participate in the *Conclave*, but not which of them are able to take part in it, as they are the ones able to decipher the identity challenge value.

In summary, for the user to answer the identity challenge implies deciphering that value using his private key $Kpriv\_U$ and mapping $H(E)$ to the originally submitted event E, sending $H(\{E\}^{K\_Hpub})$ as proof. Only the original event submitter is able to correctly answer this challenge. When the server detects that a user answered the challenge correctly, it will send him a key to ingress in the *Conclave*, K_C. $ConclaveID$ is a randomly generated value which identifies each *Conclave* session. The value must also be included in the user response because each user may take part in several *Conclave* sessions at the same time and must identify which notification he is answering to.

Providing that all clients communicate with the server by establishing a TLS session, there is no way to avoid the server knowing the user address of a submitted event. TLS can only assure that the message's contents are ciphered, while headers, which contain source and destination addresses, cannot be ciphered. These headers, may allow users' data traffic patterns to be intercepted, monitored, and analyzed by A_6.

VI. Evaluation

The evaluation of the solution was made both qualitatively, by complying with the set of requirements, and quantitatively based on two central experiments: a user reporting study and a matching performance study. The user reporting study aims to analyze the similarity between user produced security events.

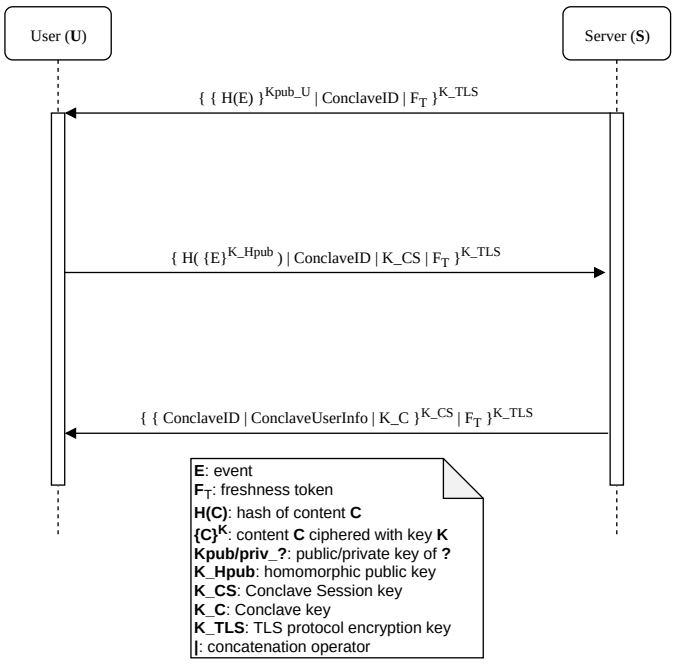

Fig. 7: Match notification protocol.

A matching performance study is also considered, since the system must provide timely answers when deployed in a real world context.

A. Requirements Assessment

The elements that will be used to qualitatively evaluate the proposed solution are a set of functional, security and performance requirements.

1) Functional Requirements: when considering the functional requirements, FR1 and FR2 are fulfilled by the *Concourse* while FR3 and FR4 are fulfilled by the *Conclave*. The event submission protocol fulfills FR1 by allowing user to publish ciphered security events to the system. The match notification protocol fulfills FR2 by notifying users whose events matched.

2) Security Requirements: SR1 states that the events can only be disclosed by their publisher, which is fulfilled by the system. The event matching operation does not require the plaintext events, therefore the event E does not leave the user's premises during the *Concourse*. The original event cannot be obtained from the submitted ciphered event $\{E\}^{K\_Hpub}$ without the homomorphic private key, which only the server has access to. However, even if a miscreant had access to this private key, which is unlikely, it would be unable to revert the digests. As intended, events can only be disclosed by their publisher, which is the only entity in possession of the original documents. SR2 and SR3 are also fulfilled, considering that the other piece of submitted data, $\{H(E)\}^{Kpub\_U}$, does not leak any information about the event, as $Kpriv\_U$ would be required to decipher it. Even if $H(E)$ was available, the digest function is non-invertible, therefore, the original content E would not be obtainable. None of the sensitive

user information is passed in plaintext to the network nor to the server, to avoid exploitation by attackers $\{A_{1+2+3}, A_5\}$ respectively. A user that receives a notification must prove itself to be the event owner by answering the identity challenge $\{H(E)\}^{Kpub\_U}$. This implies deciphering the challenge data using $Kpriv\_U$, being able to relate $H(E)$ to the originally submitted event E, and replying $H(\{E\}^{K\_Hpub})$ afterwards. Only the original user is able to answer it correctly and receive the key to ingress in the *Conclave*. Thus, there is no risk for the user to be traced through the published event, since the disclosure of these sensitive contents is limited through the use of cryptographic mechanisms. SR4 is assured through the matching of ciphered events detained by the server. Each ciphered event is a list of ciphered terms which will be matched against the contents of other ciphered events. These matched terms are homomorphically ciphered digests of relevant words extracted from the original event, which, in spite of these transformations, are still comparable. For two events to match, a subset of their tags must match in accordance to the required portion of common tags. For SR5, when reaching a *Conclave*, users are provided a metric of trust for each other, a rating value, which offers some information about the base level of trust deserved by that user, without disclosing his identity. Additionally, if intended, users can disclose their identity as proof of trust. This information should allow the event's publisher to make an informed decision about the desired course of action, whether to proceed to the *Conclave* or not. For SR6, when a match occurs, a large set of users will be notified. Where only the matched events' publishers are able to correctly answer the challenge and obtain the key for the *Conclave*. The remaining users will only send a dummy reply. For an external observer, a vast quantity of users was contacted and all of them answered, however, it is impossible to know for which of them the message was intended. These event match notifications do not allow the inference of the publishers' identities.

3) Performance Requirements: for PR1, where the system must detect the match between two similar events in a timely way, around 20 seconds. In the performance experiment, 2 with 10 tags were matched in 779 ms. Therefore, we can conclude that PR1 was successfully addressed. For PR2, while matching events, the system may present a false positive rate up to 5%, thus, the false positive rate of 17,5% obtained for the selected matching factor is not compliant with this requirement. PR3 is out of the solution's scope, thus not evaluated, since the false negative rate is strongly constrained by the quality of the inputs and cannot be directly related to the matching algorithms used in this system.

The requirements can be summarized by Table I, from where we can conclude that all functional and security requirements destined to the Concourse have been met.

Requirement	Status
FR1	✓
FR2	✓
FR3	✗
FR4	✗
SR1	✓
SR2	✓
SR3	✓
SR4	✓
SR5	✗
SR6	✓
PR1	✓
PR2	✗
PR3	✗

TABLE I: Requirements fulfillment summary.

B. Experiments

To evaluate the Concourse *matching phase*'s performance we will consider the matching time between events while scaling the number of submitted events. We devised two sets of experiments, one focused on evaluating the similarity of user produced events while observing security events, while the other is focused on matching performance.

1) Event Reporting User study: this experiment aims to provide further insight if, when exposed to the same security event, users produce a similar set of tags while evaluating the observed information, both in quantity and value. It also legitimates the need for preprocessing normalization mechanisms, like lower casing all characters or replacing all space characters by underscores.

In this experiment, 30 participants with at least 3 years of academical or professional experience in information systems were shown 5 images of real security events, provided by NAV's monitoring systems. Using a web application, each of the 30 participants submitted a form for all 5 events, which rendered 150 filled forms were obtained from this experiment. For each security event image, the participant should write any number of tags seen fit to describe the observed event. They should also attribute CVSS Version 3.0 tags [2], if applicable, or *N/A* otherwise. By also employing classification tags, though lacking flexibility, we further enhance the chances of matching based on a further standardized medium of classification.

The gathered results registered 416 tags, with 148 unique distinct values. The initial results included some typos, but also synonyms or partially similar terms, leading us to execute a preprocessing phase to improve data quality. We sanitized the gathered data by replacing spaces by underscores, lowercasing words, fuzzy string matching and removing duplicates. After preprocessing, 415 tags were registered, with 104 unique distinct values. The classification tags, different from *N/A*, were added to this sanitized dataset using a tag representation of the classifier name followed by underscore and value. The gathered results registered 1427 tags, with 126 being unique distinct values. This dataset was also employed to determine an appropriate value for RP, to be used during the matching performance test.

[2]https://www.first.org/cvss/v3.0/specification-document

2) Matching Factor Analysis: The matching function was re-implemented in Python 3.7 to run these tests but the homomorphic comparison step was not employed in this experiment, since the comparison operation is reducible to string equality verification. For this test, pairs were formed using the 150 user filled forms, with no repetition, which correspond to $\binom{150}{2}$, thus, 11175 pairs. Minding the 5 events shown to users, we establish that 5 groups exist, each containing the corresponding 30 filled forms. For each pair, it was verified if both elements referred to the same event, consequently, to the same group. If both elements in a pair belong to the same group, they are considered similar. The matching algorithm was ran against each of these pairs and evaluate whether they are considered similar. From there, the following metrics were extracted: True Positives (TP), False Positives (FP), False Negatives (FN) and True Negatives (TN). The False Positive Rate (FPR) and False Negative Rate (FNR) were also calculated using these metrics. All these values were obtained while iterating from 1% to 100%, in 1% increments, and using said value as matching factor.

RP (%)	Accuracy	TP	FP	FN	TN
34	0.7021	1271	2425	904	6575
37	0.7315	1135	1961	1040	7039

RP (%)	FPR (%)	FNR (%)	F1 score	Matthews
34	21.7	8.1	0.432976	0.265014
37	17.5	9.3	0.430658	0.268890

TABLE II: Metrics for best RP values.

To select the most promising matching factor, we isolated the factors corresponding to the maximum values of F1 score and Matthews correlation coefficient, and summarized the corresponding metrics in Table II. Considering that the complete matching algorithm employs transitive matching, during this experiment we got no indication of this transitive nature, and so, the number of false negatives/true positives are not good indicators of the matching quality. Instead, we aim at attaining a good false positive rate to comply with PR2, which limits said value at 5%. Therefore, we will select 0.37 as the matching factor for the performance test.

3) Matching Performance study: to evaluate the matching performance of the developed algorithm an experiment was devised where the computation time is evaluated with a growing number of events. In this experiment, the computation time is evaluated with the algorithm's matching factor defined based on experimentation with user filled reports, which were also used as input during the test procedure.

This dataset based exploration test was run in a non-dedicated machine with SSD persistent data storage, Intel Core i9-8950HK Processor, 2.90 GHz clock, 6 cores, 12 threads. Two virtual machines running Ubuntu 18.04 LTS were launched: one running the server application and RabbitMQ software, with 8 vCPU and 8Gb RAM; another with 2 vCPU and 4Gb RAM running two applications: a client and a workload distributor. The workload distributor was responsible for sending events to the client, to be submitted to the server.

Three situations were experimented using the 150 filled forms either directly or as a base to create new events. In one case the system was fed with the exported raw YAML events. By using the exported events as templates for the number of tags, another two sets of events have been synthesized, one where all events match and another where none match. For the full matching case, each tag was replaced by an incremental numerical value until the corresponding number of tags was reached. For the non-matching case, each tag was replaced by a random string. We also registered the time required to match 2 similar events which was of 779 ms, while 2 non-matching events took 1383 ms. The obtained results were extrapolated the outcome for larger pools of events, represented in Figure 8. For the average case, represented by our raw events, the outcome tends to approximate to the worst case situation where no events match. Given the existing optimizations, matching events are more easily processed than non-matching events, considering that the iteration is halted if two tags match. The best case scenario happens when all events match, whilst no events matching is the worst case scenario, requiring all events to be fully compared.

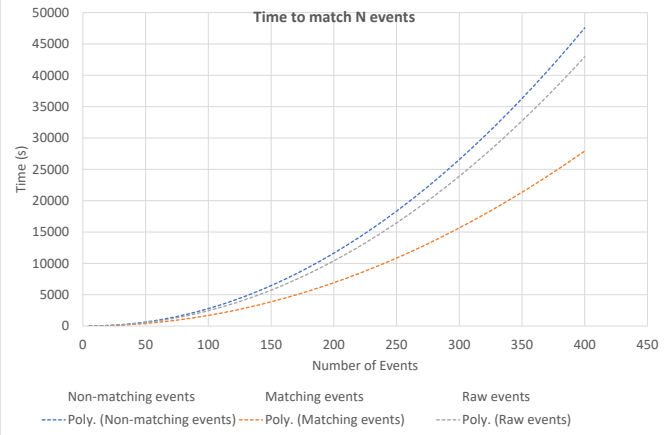

Fig. 8: Time performance polynomial trend curve.

From the performed experiments we are able to verify that users do not demonstrate predictable or homogeneous behavior on their selection of tags. Alas, it would be beneficial for the system to obtain these tags from the original events automatically, without user intervention. Since the system will be responsible for matching a large pool of events simultaneously, other strategies like batch processing or incremental matching could be employed to reduce the pool size for each operation, thus achieving better performance. Partial results could also be stored to avoid re-computation of previously executed matches. Altogether, other matching approaches could be developed to minimize the cost matching all events. In a real life deployment, better performance could be obtained through logical and physical separation between server and user applications. It would also be desirable to deploy the server application in a dedicated machine with more CPU cores, given the matching algorithm computational overhead.

VII. Conclusion

This work proposes PriVeil, a privacy-preserving threat information sharing platform, able to assure that organizations retain full control over how their information is disclosed. PriVeil is structured in two stages: Concourse and Conclave. In our work we implemented the Concourse, the stage responsible for submitting, matching events and notifying their owners to assemble with others and discuss their common issue or vulnerability. The implementation is configurable in order to adapt to the requirements of each deployment and group of participant organizations. The implementation was evaluated using tags extracted from real security events to estimate the system's matching phase performance in a real life deployment. The Concourse prototype meets all functional and security requirements related to the *Concourse* phase. However, the algorithm needs further improvements to reduce its complexity. PriVeil allows privacy-preserving threat information sharing among several entities without the risk of exposing their identities and infrastructure while cooperating to identify common security vulnerabilities. As future work, the Conclave can be implemented and the matching algorithm could be redesigned for reduced complexity.

References

[1] R. Wegberg and S. Tajalizadehkhoob, "Plug and prey? Measuring the commoditization of cybercrime via online anonymous markets," *Proceedings of the 27th USENIX Security Symposium*, pp. 1009–1026, Aug. 2018. [Online]. Available: https://www.andrew.cmu.edu/user/nicolasc/publications/VW+-USENIXSec18.pdf

[2] S. Broadhead, "The contemporary cybercrime ecosystem: A multi-disciplinary overview of the state of affairs and developments," *Computer Law & Security Review*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S026736491830308X

[3] J. Lewis, "Economic impact of cybercrime - no slowing down," McAfee, Research Report, Feb. 2018. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/restricted/rp-economic-impact-cybercrime.pdf

[4] S. L. Pfleeger, "Spider-man, hubris, and the future of security and privacy," *IEEE Security & Privacy Magazine*, vol. 13, pp. 3–10, Nov. 2015. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7349076

[5] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, ser. CCSW '11. New York, NY, USA: ACM, 2011, pp. 113–124. [Online]. Available: http://doi.acm.org/10.1145/2046660.2046682

[6] J. W. Bos, W. Castryck, I. Iliashenko, and F. Vercauteren, "Privacy-friendly forecasting for the smart grid using homomorphic encryption and the group method of data handling," in *Progress in Cryptology - AFRICACRYPT 2017*, M. Joye and A. Nitaj, Eds. Cham: Springer International Publishing, 2017, pp. 184–201.

[7] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 222–233, Jan 2014.

[8] C. Hazay, E. Orsini, P. Scholl, and E. Soria-Vazquez, "Tinykeys: A new approach to efficient multi-party computation," in *Advances in Cryptology – CRYPTO 2018*, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 3–33. [Online]. Available: https://eprint.iacr.org/2018/208.pdf

[9] I. Damgard and T. Toft, "Trading sugar beet quotas : secure multiparty computation in practice," *ERCIM News*, no. 73, pp. 32–33, 2008.

[10] B. Hie, H. Cho, and B. Berger, "Realizing private and practical pharmacological collaboration," *Science*, vol. 362, no. 6412, pp. 347–350, oct 2018. [Online]. Available: https://doi.org/10.1126/science.aat4807

[11] D. Bogdanov, M. Jõemets, S. Siim, and M. Vaht, "How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation," in *Financial Cryptography and Data Security*, R. Böhme and T. Okamoto, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 227–234.

[12] C. Wagner, A. Dulaunoy, G. Wagener, and A. Iklody, "MISP: The design and implementation of a collaborative threat intelligence sharing platform," in *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*, ser. WISCS '16. New York, NY, USA: ACM, 2016, pp. 49–56. [Online]. Available: http://doi.acm.org/10.1145/2994539.2994542

[13] J. Machado, "Trusted cooperative exchange system for security vulnerabilities and exposures," mathesis, Instituto Superior Tcnico, May 2018.