An Architecture for Processing a Dynamic Heterogeneous Information Network of Security Intelligence

Anagnostopoulos, Marios; Kidmose, Egon; Laghaout, Amine; Olsen, Rasmus L.; Homayoun, Sajad; Jensen, Christian D.; Pedersen, Jens M.

Published in:
Network and System Security

DOI (link to publication from Publisher):
10.1007/978-3-030-92708-0_11

Publication date:
2021

Document Version
Accepted author manuscript, peer reviewed version

An architecture for processing a dynamic heterogeneous information network of security intelligence *

Marios Anagnostopoulos ✉[1 [0000−0002−9193−8517]], Egon Kidmose[1 [0000−0003−0542−5334]], Amine Laghaout[2 [0000−0001−7891−4505]], Rasmus L. Olsen[1 [0000−0002−0815−5395]], Sajad Homayoun[3 [0000−0001−9371−9759]], Christian D. Jensen[3 [0000−0002−0921−7148]], and Jens M. Pedersen[1 [0000−0002−1903−2921]]

[1] Department of Electronic Systems
Aalborg University, Denmark
{mariosa,egk,rlo,jens}@es.aau.dk
[2] CSIS Security Group A/S,
Vestergade 2B, 4. sal,
Copenhagen K, Denmark
[3] Department of Applied Mathematics and Computer Science
Technical University Denmark, Denmark
{sajho,cdje}@dtu.dk

Abstract. Security intelligence is widely used to solve cyber security issues in computer and network systems, such as incident prevention, detection, and response, by applying machine learning (ML) and other data-driven methods. To this end, there is a large body of prior research works aiming to solve security issues in specific scenarios, using specific types of data or applying specific algorithms. However, by being specific it has the drawback of becoming cumbersome to adjust existing solutions to new use cases, data, or problems. Furthermore, all prior research, that strives to be more generic, is either able to operate with complex relations (graph-based), or to work with time varying intelligence (time series), but rarely with both. In this paper, we present the reference architecture of the SecDNS framework for representing the collected intelligence data with a model based on a graph structure, which simultaneously encompasses the time variance of these data and providing a modular architecture for both the data model and the algorithms. In addition, we leverage on the concept of belief propagation to infer the maliciousness of an entity based on its relations with other malicious or benign entities or events. This way, we offer a generic platform for processing dynamic and heterogeneous security intelligence with an evolving collection of sources and algorithms. Finally, to demonstrate the modus operandi of our proposal, we implement a proof of concept of the platform, and we deploy it in the use case of phishing email attack scenario.

* This research is carried out in the SecDNS project, funded by Innovation Fund Denmark

Keywords: Security intelligence · Belief propagation · system architecture · Graph network · Design matrices

1 Introduction

As a rule of thumb, security intelligence deals with the collection, analysis and presentation of data within a computational or network system with the purpose to improve the security of that system [2]. There exists a plethora of information and networking resources exposed to potential threats and involved in security incidents, thus data from these sources can reveal the perpetrators' modes of operation and their intentions. Furthermore, the compilation and standardisation of the intelligence data can contribute to cross-organisation intelligence sharing and hence enhance the collaborative actions against the evildoers [19].

However, the diversity of intelligence sources, along with the variations of the analysing methods, as well as the evolution of the cyber threats and the emergence of new ones, renders the development of a universal solution challenging [10]. Usually, the related research focuses on specific application domains or sources of data. On the contrary, the works aspiring to provide a more universal approach, typically they operate by expressing the relationships of the data through complex graph structures, or they function with time varying data, but they are usually ineffective to combine both approaches.

Although, there exist solutions to spatio-temporal problems in graph machine learning (GML), they do not satisfy the requirements of 1) the heterogeneity of nodes that are attributed, 2) the time-dependence of these nodes and their attributes, 3) the time-dependence of their relationships, 4) scoring of the nodes, and 5) expressing arbitrary interactions other than bipartite, like these of hyperedges. All these conditions are however required in the cybersecurity domain. Attempts to combine graph and time-based approaches, such as spatio-temporal graphs [20], take into account the space-time aspects of a problem but cannot accommodate the heterogeneity of the graph. On the other hand, solutions, such as GraphSAGE [7], that consider the heterogeneity of the nodes, they cannot model the time-evolution of the node attributes or the edges. Furthermore, the main shortcoming of GML lies in the fact that interactions between entities are limited to bipartite relationships. This renders GML unsuitable for problems where an interaction involves an arbitrary number of vertices with no particular directionality.

In our work, we present the architecture of a novel security intelligence framework based on a graph model for representing the intelligence data along with their time variance. The main novelty of our proposal is that it capitalizes on the belief propagation concept to deduce the maliciousness or trustworthiness of the monitored entities as their status changes over time or are involved in new emerging events. Furthermore, the framework is domain agnostic, namely the structure of the graph does not depend on the specifics of the application domain, but rather provides the generic relationships between events or, as we call it, the formation of the *entity-relationship diagram* (ERD). In addition, it

consists of a modular architecture enabling the incorporation of both the data model and the algorithms applied to these data. Finally, as a proof of concept, we demonstrate the handling of the interactions between the various components by applying the model to the use case scenario of the phishing email attack. Bear in mind, that the framework defines the components of the model in an abstract layer, while the actual designation of the entities is based on the expert's knowledge of the specific domain.

Our contribution is to offer a framework that advances the state of the art for managing security intelligence by:

- providing a graph-based model for representing the intelligence data. This model is also capable of capturing the evolution of the intelligence data over time,
- utilizing the belief propagation to spread the impact of an observation to the related instances,
- enabling modular changes to the data model, this way it is straightforward to extend the entity and relationship types or add new ones on the fly, and
- enabling modular changes to the intelligence processing routines, namely it is trivial to remove, add, or update functionalities on the fly.

The rest of the paper is structured as follows: Section 2 introduces the related work. Section 3 details the architecture of the proposed framework and explains its components. Section 4 describes the application of the framework to the use case scenario of the phishing campaigns. Finally, section 5 concludes the paper with a discussion and draws possible future directions.

2 Related work

Graph-based methods have been applied to cyber security intelligence, e.g., to infer new appearing malicious domain from known malicious domains in a bipartite client-domain graph [16]. Similar problems have been addressed with homogeneous graphs of domains and bipartite host-domain graphs [8, 11]. For example, ATIS is a generic framework for processing cyber threat intelligence similar to our proposal, offering modularity and a heterogeneous graph as the data model, but without the notion of time offered by our framework [12]. HinCTI applies ML methods, namely Graph Convolutional Network (GCN), on a heterogeneous graph, or Heterogeneous Information Network (HIN) as they refer it [4]. Moreover, Sun et al. [17] proposed to apply a spatial HIN model for domain names and five related entity types. A GCN variant is applied, namely *meta-path guided short random walk* (with six specified meta-paths), having intrinsic features for domains, but not for entities in general. In addition to the graph-based methods, there are also research works that rely on time-based analysis, e.g., to detect botnet infected clients [3] or malicious domain names [13]. On the other hand, Tran et al. [18] proposed a graph mechanism that capitalizes on the belief propagation to infer a domain's reputation score based on the relationship between domain names and IP addresses.

Overall, there are ample of examples applying graph-based or time-based methods, demonstrating how they can be used independently to solve security problems. At the same time, there exist works that combine graph-based and time-based methods, however, they deal with specific use case of threat intelligence. For instance, Garcia-Lebron et al. [5], although, considered both the graph-based and time-based aspects of the relations, their proposed framework is tailored to the use case of detecting reconnaissance behaviour of cyber attacks and may not be applicable to other scenarios. To the best of our knowledge, no other work has explored the combination of graph-based and time-based methods in a generic framework that also prioritises modularity and the ability to support general ML techniques. Put it differently, the novelty of the proposed framework lies in its modularity and ability to generalize to virtually any use case that consists of attributed entities with a score of maliciousness.

3 Method

In this section, we detail our framework by providing initially the preliminaries that are essential for understanding the general overview of our proposal. Then, we present the theoretical framework with a formal representation of the entities and processes, while afterwards we describe and explain the system architecture, before concluding with some considerations on implementation suggestions.

3.1 Preliminaries

Cyber security intelligence data, or simply *intelligence*, is any collection of data useful for preventing, detecting, or responding to security incidents [15]. To be suitable, intelligence must be related to the security of the use case of interest, as in our paradigm to phishing. Namely, intelligence should contain one or more specific instances of some entity types, and it must describe the entity (or entities), either through attribute(s) or by their relationship. For example, it can describe the knowledge that a client resides on the network of interest (identification of an instance, e.g., by an IP address), that the client is powered on (an attribute characterising the state of the client), and that the client used the Domain Name System (DNS) to resolve a domain name (interaction between the client and DNS servers).

The complete corpus of all security intelligence is not practically available and processable, however significant fragments of it can be monitored and analysed. The types of intelligence incorporate also enriched observations, such as the relation between a host's IP address and the hostname obtained via reverse DNS lookup on the IP. Either way, monitoring of data is one approach to collect intelligence. Another option is to gather intelligence from third parties, via public or private feeds, which are provided for free or under some commercial agreement.

Whether intelligence originates from controlled monitoring systems, third parties, or other sources, the observation of new intelligence is expected to occur at specific points in time, because monitoring reveals events from observed

data, or as new data arrive from a feed. To capture this, we define an *event* as a timestamped observation of intelligence data, where an observation may for example be either a first time observation, an intermediate since last modification or an affirmation that the previous intelligence data are still current. For instance, a DNS query from a client is an event which encompass several pieces of intelligence; there is a client on the network that has a certain IP address, it is active, and it aims to connect to the domain name in question.

Intelligence may also be obtained from ML, heuristics, manual processing and so forth. The common characteristic for all these processes is that they receive some intelligence as input and generate some new or updated intelligence as output. This type of process is referred hereafter as a *Map* process. Essentially, a map encapsulates the knowledge of a variety of domain experts into an automated framework that enriches the intelligence. The mapping process is the fundamental aspect of our proposed approach for providing security intelligence, and is formalised in Section 3.2.

3.2 Theoretical framework

Problem domain We shall first present a data-centred—as opposed to system-centred—overview of the components that are the building blocks of our scheme for the generation and storage of intelligence. Our starting point is to acknowledge that the cyberspace can be represented by an *entity-relationship diagram* (ERD) that links a heterogeneous set of entity types, such as domain names, IP addresses, e-mail addresses, executables, physical devices and so on. Each entity is characterised by a set of *intrinsic* features. As an example, the entity representing the domain `example.com` has as features the length of its string with value 11 and its top-level domain, which is `com`. Such features are deemed intrinsic as they are independent of any other entity or event. Cyberspace, however, is not simply build up from disjoint entities, but rather it is the scene of all sorts of well-defined *relationships* between them. For example, sending an email forms a relationship between the sender and one or more recipients, along with one or more attached files. Likewise, a DNS request establishes a relationship between the querying IP address of the end-user, the queried domain name, the resolving IP address, and one or more name servers involved in the resolution. These relationships, called *hyperedges* in the ERD, play an important role into the feature space of the involved entities. The features resulting from interactions shall be called *extrinsic*, since they depend on neighbouring entities. For example, in the case of a domain, extrinsic features could be considered the registrant or the IP addresses hosting the domain and their respective geolocation. A partial representation of the ERD for cyberspace is illustrated in Fig. 1, where the nodes (notated as ε) represent possible entities and the edges (notated as ρ) constitute a subset of relationships among these entities. An exhaustive diagram is heavily reliant on domain expertise and beyond the scope of this work.

The last component is the *belief* that an entity is involved in a given cyber threat. These beliefs, which shall be quantified as probabilities, are the end products of the intelligence generation pipeline in the sense that they are readily

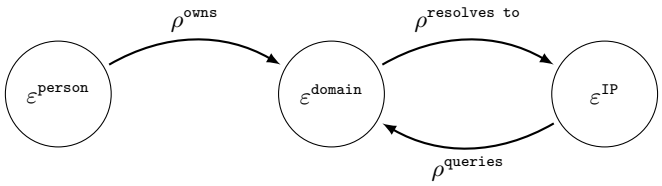

Fig. 1. Subset of the ERD spanning cyberspace. The nodes are entities, whereas the edges are relationships or part of them.

actionable. For example, access to a given domain can be blocked if it is believed that it hosts a certain malware with a probability of say 60%.

The name of the game in intelligence generation is therefore to infer the beliefs of out-of-sample entities with the best possible accuracy compared to the (presumed) ground truth. Intuitively, this is achieved by monitoring *events* and updating accordingly the beliefs of the involved entities using well-defined computational modules, namely the *maps* as introduced in Section 3.1. Features and beliefs shall be jointly referred to as *intelligence* throughout our research.[4]

Notation We shall denote each entity instance by $\varepsilon_k^{(j)}$ where k is the unique instance label and

$$j \in \mathcal{E} = \{\texttt{domain}, \texttt{IP address}, \texttt{registrant}, \texttt{ASN}, \cdots\} \tag{1}$$

is the entity type among the overall set of types $\mathcal{E}$. To each entity, we shall indicate its intrinsic features by $\boldsymbol{f}_k^{(j)}$, its extrinsic features as per a relationship type i by $\boldsymbol{r}_k^{(j,i)}$, and its beliefs by $\boldsymbol{b}_k^{(j)}$. Note that the elements of the belief vector $\boldsymbol{b}$ are probabilities that span an ordered tuple of cyber threats[5]

$$\mathcal{B} = (\texttt{benign}, \texttt{botnet}, \texttt{phishing}, \cdots)\,. \tag{2}$$

The concatenation[6]

$$\boldsymbol{d}_k^{(j)}(t) = \boldsymbol{b}_k^{(j)}(t) \oplus \boldsymbol{f}_k^{(j)}(t) \oplus \left[\bigoplus_i \boldsymbol{r}_k^{(j,i)}(t)\right] \tag{3}$$

of these three vectors shall be called the *data vector* of the entity $\varepsilon_k^{(j)}$ and encodes all the known information about that entity at the time t. Note that the extrinsic

[4] A rough, qualitative distinction can be made between features and beliefs in that the former represent raw intelligence whereas the latter represent "business-grade"—i.e., actionable—intelligence.

[5] Note that $\boldsymbol{b}$ need not add up to unity since the threats may overlap. Indeed, the threats to be predicted fall into a taxonomy which is eminently domain-specific and beyond the scope of this article.

[6] The concatenation of vectors shall be represented by the symbol $\oplus$ for the direct sum.

features are themselves concatenated over all the relationship types i that an entity of type j can be involved in, i.e., i is drawn from the overall set

$$\begin{aligned}\mathcal{R} = \{\ &\texttt{DNS queried},\ \texttt{e-mail sent},\\ &\texttt{domain registered},\ \texttt{packet received},\\ &\texttt{file opened},\ \cdots\}\end{aligned} \tag{4}$$

of possible relationship types. For example, a domain can be involved in a DNS query (as the queried domain), or in an e-mail being sent (as the sender's or recipient's domain), but cannot be involved in the opening of a file—at least not directly.

Design matrices The cornerstone of the data storage architecture proposed herein is the *design matrix*. The matrices are indexed in time, and express the features and beliefs of all entity instances of a given type j.

$$\mathcal{D}^{(j)}(t) = \begin{bmatrix} \vdots \\ \boldsymbol{d}_k^{(j)}(t) \\ \vdots \end{bmatrix}, \tag{5}$$

Thus, they encode a snapshot of the captured intelligence up to time t of all entities of that given type j.

Maps While the design matrices are the passive repositories of the intelligence, the *maps* are the active operations on that intelligence. Namely, they ensure that the design matrices are updated to reflect the latest events observed "in the wild", i.e., a map $\mathcal{M}$ is any read or write operation on the design matrices. Even more, a map can run on historical data, i.e., on older snapshots of the design matrices. More specifically, in addition to merely reading design matrices, a map is able to perform updates to:

1. the intrinsic features,
2. the extrinsic features upon the instantiation of a relationship,
3. the beliefs.

In the most general sense, the maps can thus be formalised as an update of the data vectors over a time increment δt between successive events, i.e.,

$$\mathcal{M} : \bigcup_{\varepsilon_k^{(j)}} \left\{ \boldsymbol{d}_k^{(j)}(t-\delta t) \right\} \to \bigcup_{\varepsilon_{k'}^{(j')}} \left\{ \boldsymbol{d}_{k'}^{(j')}(t) \right\}, \tag{6}$$

where the sets of input and output entities, $\varepsilon_k^{(j)}$ and $\varepsilon_{k'}^{(j')}$, may or may not overlap. Paradigms of maps could be:

– a DNS lookup of some domain instance k

$$\mathcal{M}^{\texttt{dig}} : \boldsymbol{d}_k^{(\texttt{domain})}(t-\delta t) \to \boldsymbol{r}_{k'}^{(\texttt{IP address, DNS query})}(t), \tag{7}$$

– a blacklisting of malicious domains k

$$\mathcal{M}^{\texttt{blacklist}} : \boldsymbol{b}_k^{(\texttt{domain})}(t-\delta t) \to \boldsymbol{b}_k^{(\texttt{domain})}(t) \tag{8}$$

– a detector of algorithmically-generated domains based on ML classifier

$$\mathcal{M}^{\texttt{ML-DGA}} : \boldsymbol{f}_k^{(\texttt{domain})}(t-\delta t) \to \boldsymbol{b}_k^{(\texttt{domain})}(t), \tag{9}$$

which would perform lexical analysis on the domain string $\boldsymbol{f}_k^{(\texttt{domain})}$.

Maps are therefore attuned to any change in the environment—e.g., instantiations of relationships, or inputs from third-party intelligence—and thus govern the appropriate data enrichment logic on the design matrices. This requires an algorithm which can aggregate intelligence both in time (as per the history of updates) and space (as per the topology of the ERD, see Fig. 1). A generic architecture that deals with such aggregation is proposed in [9].

3.3 System architecture

Figure 2 illustrates the architecture of our framework, accompanied by the components and data flows. In its conceptual essence, we propose a system comprised of:

– A mechanism as a generic concept for distributing the occurring events, that is the Data Platform,
– A storage mechanism that directly conceives the concept of design matrices, that is the Intelligence Database,
– The concept of Maps that outline the subsystems responsible for querying the intelligence database, consuming and producing events from the external "world" and from internal events regarding data processing, and updating the intelligence database, and finally
– The Graphical User Interface (GUI) that is connected to the data stream and is in charge of visualising the statistics and other relevant information.

The core requirement of the system is to provide an architecture that supports execution of potentially large number of Maps. These Maps need access to the historical data, i.e., on the older snapshots of the design matrices, hence a database is required to efficiently preserve these historical data. We call this database the Intelligence Database and, while in practice could be instantiated by several distributed databases, conceptually operates as a single. To support these two key components, we introduce the Data Platform that connects and facilitates the data exchange between the Maps and the Intelligence Database. The data exchange requires also data normalisation and the handling of time properties. Finally, the GUI component is linked to the Data Platform to visualize the important events, and to accommodate the user's interactions to the Intelligence Database and organisation of the Maps.

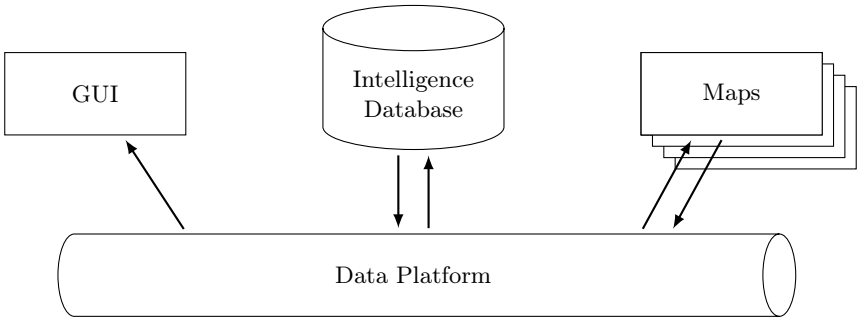

Fig. 2. System architecture

3.4 Maps

To retrieve, enrich, refine, and expand the intelligence, some form of processing is required. This capability is formally defined with Eq. (6), where the Maps process a set of instances and produce new intelligence. By supporting the instances to have historical data as property, we enable the processing of time-series data with the Maps, and by recording their relations we enable graph-based processing. The capability to maintain the Maps independently (including operations such as adding, removing, updating, re-training, etc., of the Maps) is accomplished by considering each Map as a component on its own designated by the corresponding domain expert.

At a high level, a map is essentially a "subsystem" with access to lookup/ query the intelligence database, as well as with the capability to monitor the flow of "things happening". Thus, each map is able to store data in the intelligence database by triggering updates that allow the map to be responsible for dedicated areas of the design matrices, namely a set of features (columns) of these matrices. In other words, a Map can be considered as an API providing an answer to the question "How likely is this entity involved in a (specific) malicious action". This process is expressed as a lookup to the intelligence database where specific belief(s) generated by (an) other map(s) are resolving this query and returned as response.

The flexibility, modularity, and scalability requirements adhere that multiple Maps may produce intelligence data for their own feature/column in the design matrix, while running in parallel. This will introduce typical consistency challenges for distributed data processing. One approach to resolve this issue could be to let the collection of Maps maintain their own data, but this conflicts with the goal of each Map being a simple and modular component. Therefore, the task of managing and storing the data is delegated to the Intelligence Database as discussed below. However, a Map may still store internally the state between calculations, and may cache intelligence data whenever required.

The body of intelligence data is expected to be of significant size, as many instances and long-term historical data are anticipated. This implies that the volume of data transferred from the Intelligence Database to the Maps can cause

stability issues. To mitigate this, the data transfers should be limited to include only the design matrices for the relevant entity types, and sliced to the instances (rows), features (columns) and time spans of relevance.

The Maps may have different implementations and functionalities, but the general characteristics for generating intelligence are common across all of them. However, throughout our research, we recognize two types of Maps according to the input source they use:

- External Maps: these Maps receive intelligence feeds from external sources. They rely on frameworks and tools outside the system, such as `dig`, and they usually monitor for events and are invoked whenever new ones are raised.
- Internal Maps: these Maps are based on the aforementioned mathematical framework for belief calculation. They are fed from the Intelligence Database or from the output of other Maps. In essence, these Maps implement variations of Eq. (6) for specific use case entities.

3.5 Intelligence Database

Essentially, the Intelligence Database provides intelligence data to the Maps, and in return receives events with new intelligence. Although that the Intelligence Database seems as a static graph database at any given instant, it is actually evolving through time and thus the intelligence data is treated as time-dependent by our framework, as indicated by Eq. (6).

As already explained, the Intelligence Database is responsible for storing the intelligence data. This scheme is illustrated by a set of entity types $\mathcal{E}$ having attributes features $\boldsymbol{f}^{(\varepsilon_k)}$, a set of relationships $\mathcal{R}$ determining relationship features $\boldsymbol{r}^{(\varepsilon_k)}$, and beliefs $\mathcal{B}$ declaring the contents of the belief vector $\boldsymbol{b}^{(\varepsilon_k)}$. As motivated from the aforementioned description of Maps, the data provided by the Intelligence Database should be slice-able (i.e., the range of instances, features, and time span can be chosen).

Being the single component that preserves the complete view of data in the system, the Intelligence Database is also the central point in the architecture where all changes in data, i.e., events, are observed. This makes it feasible, given adequate knowledge on which Maps rely on what data, to determine the appropriate Maps to invoke. Therefore, the Intelligence Database also has an active role in triggering the Maps when required, and thus it is not simply a passive storage component. Section 3.8 delves into more details of the implementation considerations, however we expect that an event-driven approach is promising. Even more, an event-driven approach can possibly include timers that periodically schedule execution of Maps, when needed. This is appropriate for Maps that rely on dynamic resources as input which however do not notify about the changes, like in the case of DNS data or blacklisting of online resources. Nevertheless, the discussion of how to dispatch events to maps, e.g., by broadcasting or by a publish-subscribe pattern, is deferred to future work.

3.6 Data platform

The role of the Data Platform is decisive as it facilitates the data exchange among the Intelligence Database, Maps and GUI. Key requirements, such as data normalisation and time synchronisation across components, are also handled by the Data Platform. A subscription/notification and request interaction pattern that is flexible to support a high variety of information types and amount is required as well.
The Intelligence Database is not limited to run on a single machine, but rather may be composed of a distributed set of machines hosting different databases, which in conjunction operate as the Intelligence Database. Therefore, it is critical that the Data Platform also supports load balancing of not only the processing capabilities, but also the storage and network resources to ensure that the individual machines do not become bottlenecks.

3.7 Graphical User Interface (GUI)

Finally, the GUI component has the role of the Front-End of the framework. This component is responsible for providing the interface to the external consumers, i.e., systems and users, allowing them to retrieve the intelligence. In the simplest form, the Front-End relays queries to the Intelligence Database. However, an event-driven interface, e.g., using subscriptions, can also be supported.

Examples of relevant Front-End functionalities include:

- Functionality for users/domain experts to manually evaluate entities or their beliefs.
- Functionality to retrieve real time updates, for example like a heat map overlaid over an atlas of sinkhole activity.
- An API for specific domain's data retrieval.

3.8 Implementation considerations

In this section, we elaborate into some practical issues regarding the distribution of the events in the system. The execution of Maps requires some form of time scheduling of the Map's invocation, and this can be addressed by two different approaches, viz. event-driven (Fig. 3) and periodic scheduling (Fig. 4).

Event driven: In an event-driven scheduling setup, events will be triggered sporadically from maps with external interface, and whenever internal maps have produced new results. In both cases the events, are forwarded to the Intelligence Database, as outlined in Section 3.5. In addition, the events are re-transmitted to relevant Maps, i.e., to those that use the updated intelligence as input, and therefore they need to be informed about the changes. After completing execution, each Map sends the results to the Intelligence Database, for storage and re-transmission of this new relevant event.

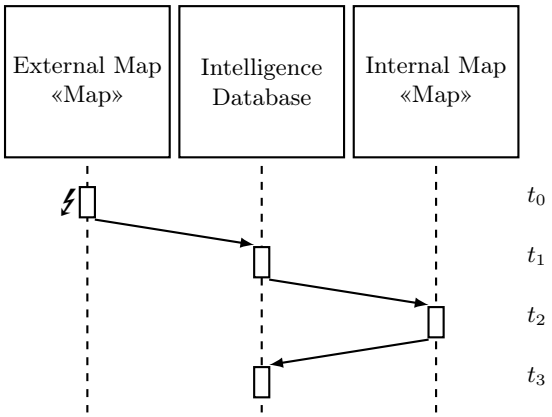

Fig. 3. Interaction diagram with events transmitted under an event-driven scheduling approach. ↯ indicates external events triggering interactions in the system.

This is the quickest possible path from the producing Map through the Intelligence Database to the consuming Map/GUI, as events are transmitted and map execution is completed with the minimum delay from the inception of the event to the propagation of its results. However, special care must be paid of what triggers a message, i.e., if the intelligence changes constantly, then the data platform will be flooded with event notifications. The event driven approach also ensures that maps are only executed when their input, as aware by the Intelligence Database, is modified, which is the least likely to introduce delays in the overall process. However, an evident drawback of the event-driven scheduling approach combined with the feedback loop between the Intelligence Database and the Maps (Fig. 2), is that it might lead to indefinitely propagation of events. For instance, if the belief of an IP address is used as input, together with other intelligence, to derive the belief of an email address being used for phishing, and the belief of this email address is afterwards used to calculate a belief for the same IP address, then this event would in turn be used to generate a belief for the email address and so forth. This situation will potentially lead to an infinite loop. To avoid such infinite loops, we can adopt for instance a *Max Hop Counter*, in such way that an event may be forwarder a fixed number of loops before it is ceased propagation.

Periodic scheduling: Periodic scheduling is an alternative approach where Maps are executed at fixed time, so they produce events periodically. This approach has the benefit that the Intelligence Database does not need to trace to which Maps a given event needs to be re-transmitted, and thus the aforementioned issue with the possible infinite loops is avoided. After the execution of the periodic Maps, the events are stored in the Intelligence Database without any

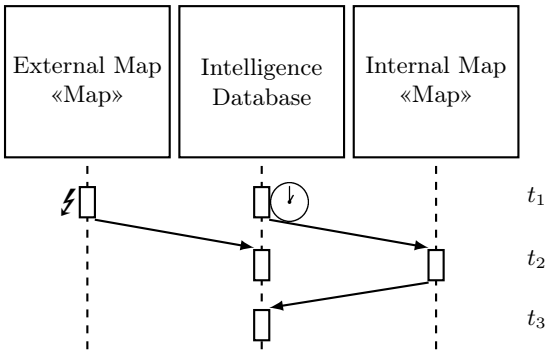

Fig. 4. Interaction diagram with events transmitted under a periodic scheduling approach. ↯ indicates external events triggering interactions in the system. 🕐 indicates internal periodic timers triggering interactions in the system.

further requirement for re-transmission. On the contrary, one apparent drawback is that occasional events might appear just after a period commenced. In that case, the new intelligence is passed to the relevant maps only after the next period, i.e., making this approach sensitive to the selection of the time interval of the period. Furthermore, the propagation of intelligence can only happen for one map per period, so if for instance three maps are chained one depending on the other, it will take three periods for the propagation to complete and that is the best - quickest - case, where the map execution completes within a period. Another drawback is that without tracking by the Intelligence Database of data changes and which Maps rely on that data, all maps have to be re-executed on all the data on every period.

Lastly, recall that maps can use any part of the design matrices, including historical data. Consequently, the periodic scheduling implies that all data have to be transferred to all Maps, and all Maps have to be evaluated for each period. This is expected to be cumbersome, given the anticipated amount of intelligence data, let alone complicated to perform frequently enough in order to achieve propagation that approaches near real-time.

4 Use case Scenario: Phishing attack

In a nutshell, phishing email attack is a well known cyber crime that seeks to trick a victim into revealing personal data, credit cards, passwords and other sensitive information [14]. Phishing can be performed in various ways, but typically evolves around deceiving an unsuspecting user to access a malicious website under the control of the attacker. However, this website aims to mimic a trustworthy one. One main approach for detecting phishing campaigns is the analysis

of domain names contained in potential phishing emails [6]. Since the resolution of a domain name to IP address takes place before the browser connects to the phishing site, such an approach can protect a victim before they access the website.

Following, we present the application of our framework to the use case of phishing attack. Specifically, we construct a scenario to demonstrate the interactions within our system, in the case a phishing email is spreading over the Internet. The definition of the entities and maps is determined based on the expert's knowledge of the specific domain. For the specific attack scenario, we implemented a proof of concept of our system that incorporates in total five Maps:

- A Map to monitor the email exchanges.
- A Map to report the client's DNS resolutions.
- A Map using an undefined ML classifier to calculate a belief of whether a DNS request is about a malicious domain.
- A Map implementing the heuristic if a *URL* is hosting malicious content, such as a phishing page or a malicious script.
- Finally, a Map inferring if an email is part of a phishing attack.

The interactions of the involved system components are depicted in Fig. 5 and constitute three major phases:

1. At the beginning, the *Email Monitoring* Agent reports an email containing a URL. This event is transmitted to the *Intelligence Database* and propagated to the *Email is Phishing* Map, which applies the heuristic whether a given email is malicious. Then, the *Email is Phishing* Map triggers the *URL is Malicious* Map, which derives a maliciousness score according to its internal heuristic. The *Email is Phishing* then outputs a phishing score to the *Intelligence Database*. If the URL is not considered malicious then the email does not contain phishing contents. The relevant belief is thus set to 0.0.
2. Following, the *DNS Resolver* Agent reports that the client, which had previously received the email, resolves afterwards the domain `evil.com`. This event is forwarded to the *Domain is Malicious* Map, where the domain is determined as malicious, and the result is stored in the database.
3. After a while the *Email Monitoring* Agent reports a new email containing a URL similar to the previous. The *Intelligence Database* triggers the *Email is Phishing* Map. In turn, this Map hands over the URL to *URL is Malicious* and receives the maliciousness score of 1.0 for the URL. The *Email is Phishing* Map then outputs *Phishing Email:*1.0 to the *Intelligence Database*. Since, the user resolved a malicious domain (`evil.com`) right after receiving an email containing this malicious URL, the heuristics of *Email is Phishing* Map predicts that the email is phishing attack, and thus set the value of that belief to 1.0.

This example scenario follows a belief propagation approach to spread the impact of an event/interaction to the involved instances. In this case, the data

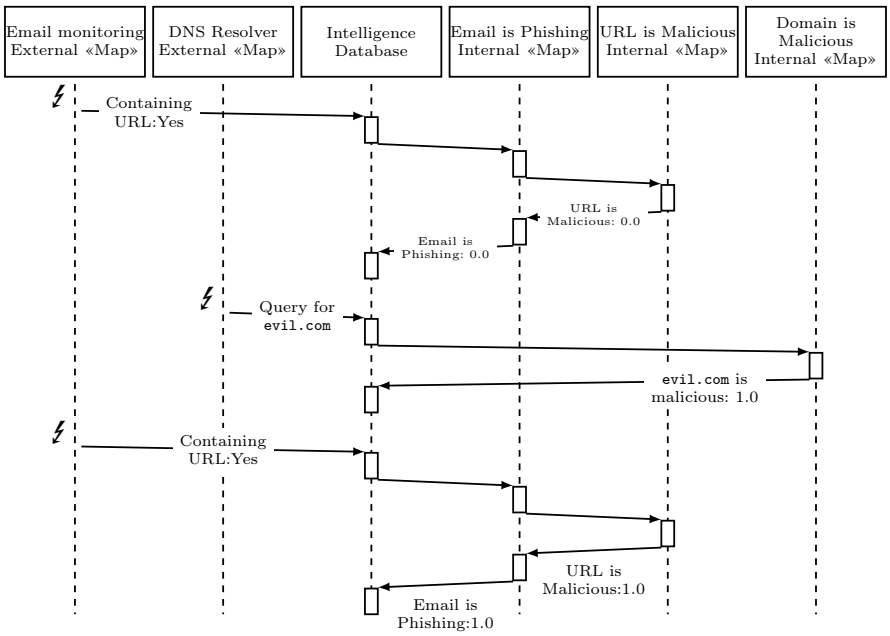

Fig. 5. Events propagation during the phishing email scenario. ↯ indicates external events triggering interactions in the system. Arrows signify event transmission, with labels informally summarising the new or updated intelligence (New rows for, or updates to, the design matrices). The Maps rely on pulling additional data from the Intelligence Database, but this is omitted here for brevity and simplicity.

related to instances involved in the event would be updated to reflect the dynamic behaviour of that instance. In our use case scenario, the query for the domain *evil.com* by the recipient of the email affects the maliciousness score of the email and the URL of the subsequent events.

5 Conclusion

As the sophisticated techniques from the side of the evildoers are evolving with the purpose to disguise their actions and deceive the end-users, more advanced methodologies are also required for distinguishing between benign and malicious resources and activities. The perpetrators behind malicious actions are constantly in move, thus, making a static counteraction against the involved entities unsuitable for such a dynamic environment. What can be considered benign one day, it can turn malicious the next day, and vice versa on a later day. By statically blacklisting and blocking the malicious sources of today's cannot provide protection in the long run. In this context, security intelligence aims to uncover threats and threat actors, in order to prevent and mitigate their im-

pact. However, traversing through all available intelligence requires automated processes, making the belief propagation approach a necessity to meet requirements and expectations of end-users who anticipate a safer Internet.

To this end, we present the reference architecture of a framework that allows the use of intelligence to provide beliefs in which degree specific entities or resources may be involved in malicious activities. The framework is designed on the basis to accommodate intelligence from various topics and applications, so it will be generic, domain agnostic, and capable to incorporate a non-limited number of sources and information types. Furthermore, to exemplify the framework's mode of operation, we provide as a proof of concept its deployment to the use case of phishing attacks.

In essence, our framework is centred around a graph structure representation of the intelligence data and leverages on the concept of belief propagation in accordance with the trustworthiness of the events that it is related to. In addition, based on functionality of the Maps and their invocation as new threats appear, the intelligence knowledge is updated and evolved. Over time, new types of intelligence sources will be discovered or relation will be formulated. Hence, the framework supports extensions to new maps, both for external and internal interactions. Furthermore, the design matrix approach enables a flexible and extensible method to achieve this requirement.

The formulation of the data types and the operations on the intelligence data depend on the expert's knowledge of the specific domain. Therefore, as future work, we plan to enhance our platform with a variety of use case scenarios related with the domain of DNS security and how domain names are correlated with malicious actions, like botnets, malware propagation and phishing campaigns [1]. Finally, we intend to investigate the appropriate ML algorithms for the accurate and timely calculation of the belief propagation in an event-driven implementation of the proposed platform.

References

1. Anagnostopoulos, M., Kambourakis, G., Gritzalis, S.: New facets of mobile botnet: architecture and evaluation. International Journal of Information Security **15**(5), 455–473 (2016)
2. Barnum, S.: Standardizing cyber threat intelligence information with the structured threat information expression (stix). Mitre Corporation **11**, 1–22 (2012)
3. Choi, H., Lee, H.: Identifying botnets by capturing group activities in dns traffic. Computer Networks **56**(1), 20–33 (2012)
4. Gao, Y., Xiaoyong, L., Hao, P., Fang, B., Yu, P.: Hincti: A cyber threat intelligence modeling and identification system based on heterogeneous information network. IEEE Transactions on Knowledge and Data Engineering (2020)
5. Garcia-Lebron, R.B., Schweitzer, K.M., Bateman, R.M., Xu, S.: A framework for characterizing the evolution of cyber attacker-victim relation graphs. In: MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM), pp. 70–75. IEEE (2018)

6. Hageman, K., Kidmose, E., Hansen, R.R., Pedersen, J.M.: Can a TLS Certificate Be Phishy? In: 18th International Conference on Security and Cryptography, SECRYPT 2021, pp. 38–49 (2021)
7. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 1025–1035 (2017)
8. Khalil, I., Yu, T., Guan, B.: Discovering malicious domains through passive dns data graph analysis. In: Proc. of the 11th ACM ASIACCS, pp. 663–674. ACM (2016)
9. Laghaout, A.: Supervised learning on heterogeneous, attributed entities interacting over time. arXiv preprint arXiv:2007.11455 (2020)
10. Li, V.G., Dunn, M., Pearce, P., McCoy, D., Voelker, G.M., Savage, S.: Reading the tea leaves: A comparative analysis of threat intelligence. In: 28th USENIX Security Symposium, pp. 851–867 (2019)
11. Manadhata, P.K., Yadav, S., Rao, P., Horne, W.: Detecting malicious domains via graph inference. In: ESORICS, pp. 1–18. Springer (2014)
12. Modi, A., Sun, Z., Panwar, A., Khairnar, T., Zhao, Z., Doupé, A., Ahn, G.J., Black, P.: Towards automated threat intelligence fusion. In: 2nd IEEE CIC, pp. 408–416. IEEE (2016)
13. Moura, G.C., Müller, M., Wullink, M., Hesselman, C.: ndews: A new domains early warning system for tlds. In: NOMS 2016, pp. 1061–1066. IEEE (2016)
14. Panum, T.K., Hageman, K., Hansen, R.R., Pedersen, J.M.: Towards adversarial phishing detection. In: 13th USENIX Workshop on CSET20 (2020)
15. Qamar, S., Anwar, Z., Rahman, M.A., Al-Shaer, E., Chu, B.T.: Data-driven analytics for cyber-threat intelligence and information sharing. COSE **67**, 35–58 (2017)
16. Rahbarinia, B., Perdisci, R., Antonakakis, M.: Segugio: Efficient behavior-based tracking of malware-control domains in large isp networks. In: 45th Annual IEEE/IFIP DSN, pp. 403–414. IEEE (2015)
17. Sun, X., Wang, Z., Yang, J., Liu, X.: Deepdom: Malicious domain detection with scalable and heterogeneous graph convolutional networks. COSE p. 102057 (2020)
18. Tran, H., Nguyen, A., Vo, P., Vu, T.: Dns graph mining for malicious domain detection. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 4680–4685. IEEE (2017)
19. Wagner, T.D., Mahbub, K., Palomar, E., Abdallah, A.E.: Cyber threat intelligence sharing: Survey and research directions. COSE **87**, 101589 (2019)
20. Zhang, J., Shi, X., Xie, J., Ma, H., King, I., Yeung, D.Y.: GaAN: Gated attention networks for learning on large and spatiotemporal graphs. arXiv preprint arXiv:1803.07294 (2018)