Collaborative Cloud-based Application-level Intrusion Detection and Prevention

Omar Iraqi*†, Meryeme Ayache*, and Hanan El Bakkali*
*Rabat-IT Center, ENSIAS, Mohammed V University, Rabat, Morocco
†School of Science and Engineering, Al Akhawayn University, Ifrane, Morocco
Email: o.iraqi@aui.ma, meryeme.ayache@um5s.net.ma, h.elbakkali@um5s.net.ma

***Abstract*—The recent years have witnessed an increasing number of coordinated and large-scale attacks. This comes at no surprise as data processing, transfer and storage have got and continue to be faster and cheaper. A standalone Intrusion Detection System (IDS) may only be exposed to a narrow subset of such attacks, which could be too insignificant to raise suspicion. In contrast, a Collaborative Intrusion Detection System (CIDS) leverages collaboration among its members across multiple networks and organizations. In this work, we extend our Application-level Unsupervised Outlier-based Intrusion Detection and Prevention framework by leveraging the benefits of CIDSs. More specifically, we design a collaborative intrusion detection architecture made of three levels: the organization level, the domain level and the overarching root level. This hierarchical architecture combined with streaming and clustering offers very good privacy, scalability, accuracy and resilience tradeoffs. Moreover, the adoption of the cloud as a cost-effective and elastic platform allows us to handle big data generated by millions of applications as alarm streams. We also specify a lightweight Application Alarm Message Exchange Format (A2MEF) to support collaboration among the different stakeholders. Finally, we design a reputation-based alarm correlation algorithm that manages the iterative and bidirectional relationship between the reputation of involved parties and the accuracy of their reported alarms.**

***Keywords*—Collaborative Intrusion Detection; Application-level Intrusion Detection; Hierarchical Architecture; Alarm Correlation; Cloud Computing; Big Data.**

I. INTRODUCTION

With the ever growing and affordable processing power and network bandwidth, coordinated and large-scale attacks are steadily prevailing. For example, adequately-equipped attackers may launch Internet-wide scans, discover and infect vulnerable systems to finally use them in Distributed Denial of Service (DDoS) attacks worldwide. A standalone IDS may only be exposed to a narrow subset of such attacks, which could be too insignificant to raise suspicion [1]. In contrast, a *Collaborative Intrusion Detection System (CIDS)* leverages collaboration among its members, which may spread over multiple networks or even different organizations. This global, cross-network and cross-organizational approach does not only reduce false negatives, but it also reduces false positives thanks to alarm correlation and filtering [1]. Moreover, CIDSs improve overall performance while reducing the overhead on each node thanks to load sharing [2].

In this work, we extend our Application-level Unsupervised Outlier-based Intrusion Detection and Prevention framework [3] by leveraging the benefits of CIDSs. In our initial framework, methods to be instrumented are selected statically/manually by the application owner. Indeed, when applying our framework to immunize a target application [3], the owner has to explicitly specify methods or entire packages to intercept, monitor and analyze using unsupervised outlier detection. Such a choice may not be well-informed or may even be arbitrary. Some irrelevant methods may be instrumented, incurring an unjustified overhead, while other pertinent methods may be missed, causing some critical intrusions to go undetected.

Therefore, we aim at empowering our application-level intrusion detection and prevention framework to make well-informed, risk-based and adaptive decisions about methods to (un)instrument. Risk identification, or at least threat identification, shall be supported through collaboration. Peer applications (instances of the same application running in different nodes and different organizations eventually) would start by instrumenting seed methods that can be downloaded from the collaborative system, manually selected by the node owner, or even based on information gathered from external sources, such as Computer Emergency Response Teams (CERTs) and security advisories. Then, as an application instance detects an intrusion at the level of a method, related information propagates through the collaborative system, ultimately making the other instances activate the monitoring of that method. Peer applications may be further narrowed under communities and alliances based on the geopolitical context, business sector, causes and interests, as well as other criteria to be defined. Once these communities are designated, their application instances shall be considered as *community members*. This allows our initial framework to evolve from empowering applications with immunity like the *human body*, to providing them with a sense of belonging like in *human societies*.

This work makes the following contributions:

- A collaborative cloud-based framework for application-level intrusion detection
- A hierarchical architecture for collaborative application-level intrusion detection
- An application alarm message exchange format
- A reputation-based alarm correlation algorithm

This paper is organized as follows. Section II reviews the related work in terms of application-level intrusion detection and collaborative intrusion detection, as well as collaboration-specific threats and countermeasures. Section III describes our Collaborative Cloud-based Application-level Intrusion Detection framework in terms of architecture, alarm exchange format and alarm correlation algorithm. Finally, we conclude our paper by stating future work and direction.

 ISBN: 978-1-61208-719-1

II. Related Work

A. Application-level Intrusion Detection

This work leverages and extends our Application-level Unsupervised Outlier-based Intrusion Detection and Prevention framework [3]. The ultimate goal is to empower software applications with artificial immunity against cyber attacks. We contributed to the fulfillment of such a goal by allowing applications themselves to play a central and active role in the intrusion detection and response processes. While traditional network and host intrusion detection systems have access to raw strings and bytes through I/O operations only, our framework allows tracking application domain objects all along the processing lifecycle. Thanks to unsupervised learning, our framework leverages the application business context and learns from production data, without creating any training burden on the application owner. Moreover, as our framework uses runtime application instrumentation, it incurs no additional cost on the application provider.

More specifically, we built a fine-grained and rich-feature application behavioral model that gets down to the method level and its invocation context. We consider the call stack as a key indicator of such a context. Indeed, under different call stacks, the same method may take completely different sets of inputs, follow different control flows, and yield different sets of outputs. Unsurprisingly, the call stack is extensively used in the current work. As a matter of fact, the main purpose of the collaborative framework proposed in this paper is to identify, prioritize and share call stacks to monitor per application.

Other approaches to application intrusion detection have been proposed. A useful review is given by [3] - Table 1. It classifies and compares different works in terms of *what* data is collected, *who* collects it, from *where* it is collected, *when* it is collected and *how* it is analyzed.

B. Collaborative Intrusion Detection

The main artifacts related to CIDSs are challenges and requirements, architecture, analysis target, technique and timeline, shared information and interoperability.

1) Challenges / Requirements: While collaborative intrusion detection offers several benefits, it also introduces several challenges, which drive the main requirements of CIDSs. These are privacy, scalability, accuracy, resilience and incentive [4]. As explained below, these requirements are oftentimes conflicting, e.g., privacy and scalability vs. accuracy.

1) Privacy: as the collaborating members need to share information with each other, privacy becomes a concern. While secure channels can protect shared data from eavesdropping by external parties, sensitive information may still be divulged to other CIDS members. Therefore, shared data shall be carefully specified in order to abide by the security policy, legal obligations and contractual agreements of involved parties.
2) Scalability: collaboration creates a network overhead that depends on the amount of exchanged data, as well as the number of nodes. The adopted architecture too has a direct impact on scalability. For example, in a centralized architecture, central servers may be overloaded, affecting system scalability. However, in a distributed / P2P architecture, higher scalability is supported as nodes play a symmetric role, but network overhead grows quadratically with the number of nodes [4].
3) Accuracy: while collaboration is supposed to enhance intrusion detection accuracy, hiding some data to preserve privacy or reducing it to support scalability may have a negative impact on accuracy. Moreover, in distributed / P2P architectures where higher scalability is supported, accuracy is negatively impacted as no member holds complete knowledge about the system [4]. Therefore, tradeoffs between data privacy and system scalability versus detection accuracy shall be made depending on organizational priorities and operational constraints.
4) Robustness / Resilience: attacks against CIDSs may have disastrous consequences since protected networks, systems and applications become directly exposed to subsequent threats. This is why CIDSs shall avoid single points of failure (SPoF) and be resilient to both external and internal attacks [1]. These are described below along with corresponding state of the art countermeasures, such as membership, trust and reputation management.
5) Incentive: why would an organization join a CIDS? Why would it offer its processing power and network bandwidth, and maybe sacrifice its privacy in the name of collaboration? Organizations have mainly two incentives: coercion incentive and benefit incentive [4]. Coercion incentive means that nodes have *no choice* to ”survive” but collaborate. This could be due for instance to their insufficient processing power or their incapacity to detect intrusions without external help. As opposed to coercion incentive, benefit incentive means that members are *encouraged* to join the collaborative sytem and if they do, they will gain benefits from the CIDS. More specifically, these are ”merit-based”, which means that higher contributions lead to higher benefits.

2) Architecture: A CIDS is made of monitoring units, correlation units and decision units [4]. A monitoring unit gathers data locally and, depending on the node capability and design choices, may partially or fully process it. Raw, partially processed or fully processed data is then passed over to a correlation unit. This latter is what really characterizes CIDSs. It communicates with other correlation units and exchanges security-relevant information with them according to a protocol, such as the Intrusion Detection Exchange Protocol (IDXP) [5] or a common data exchange format, such as the Intrusion Detection Message Exchange Format (IDMEF) [6]. Finally, the decision unit collects and processes shared information to make a decision. Where these units are deployed and how they integrate with each other depend on the adopted architecture. Tradeoffs made in fulfilling the requirements stated above lead to different architectures. In addition to the aforementioned

 ISBN: 978-1-61208-719-1

centralized and distributed / P2P architectures, a hierarchical architecture is also suggested and used [1] [4].

1) Centralized architecture: whereby a central server collects and analyzes information shared by monitoring nodes. The central server hosts the unique decision unit along with a correlation unit. As mentioned earlier, centralized architectures promote intrusion detection accuracy since there is a central decision unit to which all shared information converge. However, the central server creates a Single Point of Failure (SPoF) and a bottleneck against scalability [1] [4].
2) Distributed / P2P architecture: whereby all nodes of the CIDS play a symmetric role. Hence, each and every node hosts a monitoring unit, a correlation unit and a decision unit. As previously mentioned, a peer-to-peer architecture does not suffer from any SPoF, scales to increasing numbers of nodes, but at the expense of intrusion detection accuracy. To avoid overloading the underlying network by peer-to-peer traffic, peer selection criteria need to be defined in order to narrow the number of peers that each node communicates with.
3) Hierarchical architecture: whereby a compromise between the centralized and the distributed architectures is sought. To this end, the centralized architecture is enhanced by inserting additional (tree / hierarchical) layers between monitoring nodes and the root central server. Thanks to these layers, monitoring nodes communicate with the central server via their parents. These may gather and aggregate information shared by their children, before sharing it with their own parents. At higher levels, in addition to communicating with their parents, nodes may also communicate peer-to-peer.

Table I summarizes the architecture support for privacy, scalability, accuracy and resilience requirements. Incentive has not been included as it is not directly affected by the architecture.

TABLE I. ARCHITECTURE SUPPORT FOR REQUIREMENTS

Architecture	Privacy	Scalability	Accuracy	Resilience
Centralized	●	•	●	•
Distributed	•	●	•	●
Hierarchical	●	●	●	●

3) Analysis: This is the core process in intrusion detection. It targets specific data and processes it using a specific technique in a specific timeline.

1) Target – *what* data is analyzed: network packets, system logs, other host data, or application-level data.
2) Technique – *how* data is analyzed: signature-based or anomaly-based: (semi-)supervised, unsupervised.
3) Timeline – *when* data is analyzed: offline or online.

4) Shared Information: Collaboration units may share raw, partially processed or fully processed data [4] depending on node resources and capabilities, as well as design choices.

1) Raw data: whereby low-capability nodes send gathered data ”as is” to higher-capability nodes. This practice affects data privacy and causes a higher network overhead.
2) Partially processed data: whereby capable nodes perform some data preprocessing, filtering, and/or compression in addition to sensitive data hiding. These practices reduce network traffic and strive to preserve data privacy.
3) Processed data: whereby nodes perform full data processing and analysis to identify intrusions locally and send corresponding alarms to other nodes for further correlation and/or final decision.

5) Interoperability: An underlying collaboration mechanism needs to be defined and implemented at the level of CIDS nodes. This mechanism can be a communication protocol, such as the Intrusion Detection Exchange Protocol (IDXP) [5], a data exchange format, such as the Intrusion Detection Message Exchange Format (IDMEF) [6] or even a collaboration framework, such as JXTA, Pastry, Scribe, GNUnet and FreeNet.

6) Taxonomy: A detailed taxonomy of research-oriented and commercial CIDSs based on their architecture, as well as their support for privacy, scalability, accuracy and resilience requirements is given by [1] - Table III and by [4] - Table VII. An equally useful taxonomy of CIDSs based on their analysis target and timeliness, architecture, shared information and interoperability is given by [4] - Table IV.

C. Collaboration-specific Threats and Countermeasures

While IDSs are subject to DDoS and mimicry attacks in general, we focus here on attacks that specifically target the collaboration aspect of CIDSs. First, we will identify collaboration-specific threats, as well as their compensating baseline controls in terms of trust/reputation management and shared information protection. Then, we will describe popular attacks against these baselines and their countermeasures.

1) Collaboration-specific Threats and Baseline Controls: These can affect the confidentiality, as well as the integrity of shared information, hence exposing the CIDS and protected systems to a host of risks. Confidentiality-related threats take advantage of shared information to divulge sensitive data to attackers. Examples include exposing monitors location and target like networks, systems, applications and data, as well as their vulnerabilities. Integrity-related threats can be summarized as having rogue or compromised nodes ”telling lies” or ”not saying the whole truth” when sharing information, in addition to malicious parties tampering with such information in transit. Examples include nodes spreading fake alerts and/or selectively forwarding received information. In particular, a so-called *Sybil* attack – named after the famous dissociative identity disorder case and book – consists of using an army of pseudonymous nodes to influence CIDS decisions [7].

To address confidentiality-related attacks, information hiding, e.g. hashing, is needed. As far as integrity-related attacks are concerned, *trust/reputation management* has been introduced. It consists of managing a trust/reputation value for each

 ISBN: 978-1-61208-719-1

node, as well as to identify, penalize and ultimately kick off rogue and compromised nodes. Trust/reputation management requires node identification and authentication. Moreover, Sybil attacks can only be addressed through a certification authority (CA), which scrutinizes and validates nodes [7].

2) Attacks against Trust/Reputation Management and Countermeasures: Trust/reputation management can be compromised by several attacks. A so-called *Betrayal* attack consists of compromising a trusted node and using it to expose the confidentiality and/or integrity of the CIDS. A variant called *Sleeper* attack uses a rogue node that spends an initial period faking a normal behavior to gain a higher trust/reputation value before exploiting it against the CIDS. The *Newcomer* attack tries to make the CIDS "forget" about the bad reputation of a rogue node by joining it again under a new, clean identity.

To counteract these attacks, several enhancements have been suggested. More specifically, the impact of both Betrayal and Sleeper attacks can be reduced through fast degradation of trust against nodes that exhibit a malicious behavior, while the Newcomer attack can be controlled through a probation period for newcomers. Table II summarizes attacks against trust/reputation management and their countermeasures.

TABLE II. ATTACKS VS. COUNTERMEASURES

Attack	Trust/Reputation Management
Sybil	CA
Betrayal / Sleeper	Fast degradation of trust
Newcomer	CA, Probation period

D. Cloud-based Intrusion Detection

Cloud computing provides organizations with computing resources featuring easy deployment, connectivity, configuration and scalability. There are three cloud service delivery models and IDS cloud deployment differs from one model to another.

- Software as a Service: in SaaS users merely depend upon their providers to deploy their services. Hence, the SaaS cloud provider is responsible of deploying the IDSs. In this case, the users may only get some logs or configure some cosutoms monitoring alerts.
- Platform as a Service: in PaaS, IDSs are deployed outside applications by the cloud service provider. However, users can configure their applications and platforms to log out onto a central location to be used by a central IDS.
- Infrastructure as a Service: this delivery model is more flexible in term of IDS deployment. In fact, the IDS can be deployed at several levels in the IaaS cloud layer: the virtual machine, the hypervisor and the network.

As listed and compared in table III, we can classify the deployment of IDSs in the cloud into five categories:

- In-Guest agent based approach, which consists of deploying the IDS at the Virtual Machine (VM) level. The advantage of this approach is that it does not require any modification of the hypervisor and runs as an application in a tenant VM, which is configured and controlled by the tenant. Moreover, the IDS has a good visibility of the monitored VM. Hence, it can perform deep scanning of packets leaving or entering the VMs and can perform host audit log analysis, system call analysis and program analysis of the VMs. One limitation of this approach is that it fails to detect collaborative attacks.
- In-VMM agent based approach, which consists of deploying the IDS at the hypervisor level in IaaS environments. The hypervisor acts like a central location for intrusion detection. In fact, it can monitor both the hypervisor and data traveling between the hypervisor and the virtual machine (for any VMM attacks). However, just like the In-Guest agent based approach, the In-VMM agent based approach fails too in detecting collaborative attacks.
- Network-monitor based approach, which allows monitoring the network traffic between VMs and the host machine and between VMs themselves. IDSs are deployed at network points, such as the core switch or other network switches. However, this approach yields a poor visibility of the monitored VMs and can not detect host-based anomalies, such as VM escapes, rootkits, viruses, worms and collaborative network attacks.
- Collaborative agent based approach, which places IDS components at different locations, such as at the VM, VMM or network points. These components collaborate to detect various attacks including collaborative attacks.
- Distributed approach, which runs IDS instances over tenant VMs (TVMs) on a Cloud Compute Server but are controlled by a Cloud Controller Server (CCS).

TABLE III. INTRUSION DETECTION SYSTEMS IN THE CLOUD

References	Year	IDS Method	IDS Type
Lee et al. [8]	2011	In-Guest Agent (A)	Anomaly-based
McGee [9]	2013	In-Guest Agent (A)	Prevision-based
Shi et al. [10]	2016	In-VMM Agent (B)	Anomaly-based
Maiero et al.[11]	2011	In-VMM Agent (B)	Intrusion-based
Chiba et al. [12]	2018	Net. Monitor (C)	Anomaly-based
Bharadwaja et al. [13]	2011	Collab. Agent (D)	Anomaly-based
Lo et al. [14]	2010	Collab. Agent (D)	Attack-based
Gupta et al. [15]	2014	Disributed (E)	Attack-based

E. Comparison of our Framework with Existing CIDSs

As opposed to existing CIDSs that use either the client-server model or the peer to peer model for synchronous and tightly-coupled communication, we adopt a hierarchical architecture that leverages cloud-based, clustered, and brokered streaming for asynchronous, loosely-coupled and scalable communication. Another unique aspect of our framework is the integration of social, news and security advisories feeds to enhance collaboration. We also define a lightweight JSON message exchange format to share alarms as fully processed data among nodes hidden behind brokers, hence avoiding sensitive information leakage. Finally, we design an alarm correlation algorithm that manages the iterative and bidirectional relationship between the reputation of involved parties and the accuracy of their reported alarms.

 ISBN: 978-1-61208-719-1

III. Our Collaborative Cloud-based Application-level Intrusion Detection

A. Collaborative Cloud-based Intrusion Detection Architecture

As shown in Figure 1, we adopt a hierarchical collaborative intrusion detection architecture leveraging the cloud. The choice of a hierarchical architecture is motivated by the privacy, scalability, accuracy and resilience tradeoffs it offers in comparison with the centralized and distributed architectures. Moreover, the cloud is a key element in our architecture thanks to its cost-effectiveness, elasticity and capacity to handle alarm streams generated by millions of applications as big data.

In the proposed architecture, organizations can have their applications running on premise or in the cloud. These applications are instrumented to dynamically (un)select methods for raw data extraction and secure streaming to the organization-level Kafka cluster through Kafka Streams API. These data streams are continuously consumed by the Organization-level Application Intrusion Detection Nodes (OAIDNs). These are managed by Kafka Streams API as a group/cluster (OAIDC) and implement our unsupervised, application-level intrusion detection framework [3]. Moreover, we distribute the load among the OAIDNs while making sure each OAIDN receives a coherent and complete stream. To this end, we create a single Kafka topic for all applications, with a separate partition for each call stack of each selected method within each monitored application. This way, the whole stream of data extracted from a method call in a given call stack will be processed by the same OAIDN. Other streams may be assigned to other OAIDNs in the cluster for load sharing.

Intrusions detected at the level of an organization are streamed as alarms to the domain-level correlation and decision cluster (DCDC) through the domain-level Kafka cluster. We would like to underscore that while alarms are shared with the DCDC, applications remain hidden behind the OAIDC. Without using other offline techniques like social engineering, it is not possible to reveal which application generated which alarm. As previously mentioned, to defend against Sybil attacks, the organization-level intrusion detection cluster needs to present a trusted certificate to the DCDC. With its broader cross-organizational view, the DCDC is responsible for correlating alarms generated by all organizations under its domain. Here again, we define a single Kafka topic and we partition it based on the application identifier. Then, the DCDC propagates the aggregated scores to higher levels up to the Root Correlation and Decision Cluster (RCDC) for decision making and information sharing with other CIDS branches. Moreover, the DCDC may make decisions at its own level and share related alarms with organizations under its domain.

Finally, the DCDC may leverage news, social and security advisories feeds to augment correlated data with geopolitical and cyber trends. Here, we consider authoritative security advisories, such as the National Vulnerability Database (NVD), as well as feeds from trusted security product vendors. We also consider other general-purpose feeds, such as news and social feeds from sources that are not necessarily trusted. Nevertheless, such sources can bring valuable information timely. AI and text mining techniques shall be applied to filter corresponding feeds and extract meaningful information.

B. Application Alarm Message Exchange Format

We define an Application Alarm Message Exchange Format (A2MEF) that maps a list of alarms to an application. A2MEF is specified as a lightweight JSON object whose schema is exhibited in Listing 1. The *app* property consists of SHA-512 hash of the application software ID (SWID) in compliance with the ISO/IEC 19770-2 standard. According to this standard, the SWID specifies the software name, edition, version and publisher in XML format [16]. The *alarms* property is an array of alarms described each by the *method* and specific *call stack* that raised it. It is worth mentioning that the *app* property may be omitted if it can be inferred from the queue to/from which the alarm message is streamed, e.g., Kafka stream topic or partition. The same format can be used to propagate information upward the hierarchy for alarm correlation and decision making, as well as downward the hierarchy for alarm feedback and response. In the latter case, the *alarms* array shall represent an *ordered* list of correlated alarms.

```
{
  "$schema": "http://json-schema.org/schema#",
  "title": "Application Alarms",
  "type": "object",
  "properties": {
    "app": {
      "type": "string",
      "description": "SHA-512 hash of the ISO/IEC 19
          770-2 SWID"
    },
    "alarms": {
      "type": "array",
      "description": "Alarms bound to app",
      "items": {
        "type": object,
        "properties": {
          "method": {
            "type": "string",
            "description": "Fully qualified method
                name"
          },
          "callstack": {
            "type": "string",
            "description":  "SHA-512 hash of the
                call stack"
          }
        }
      }
    }
  }
}
```

Listing 1. JSON Schema of A2MEF

C. Reputaion-based Alarm Correlation

We aim here at designing a correlation algorithm that, given a stream of alarms reported by different parties, emits an *ordered* list of methods and call stacks per application. This

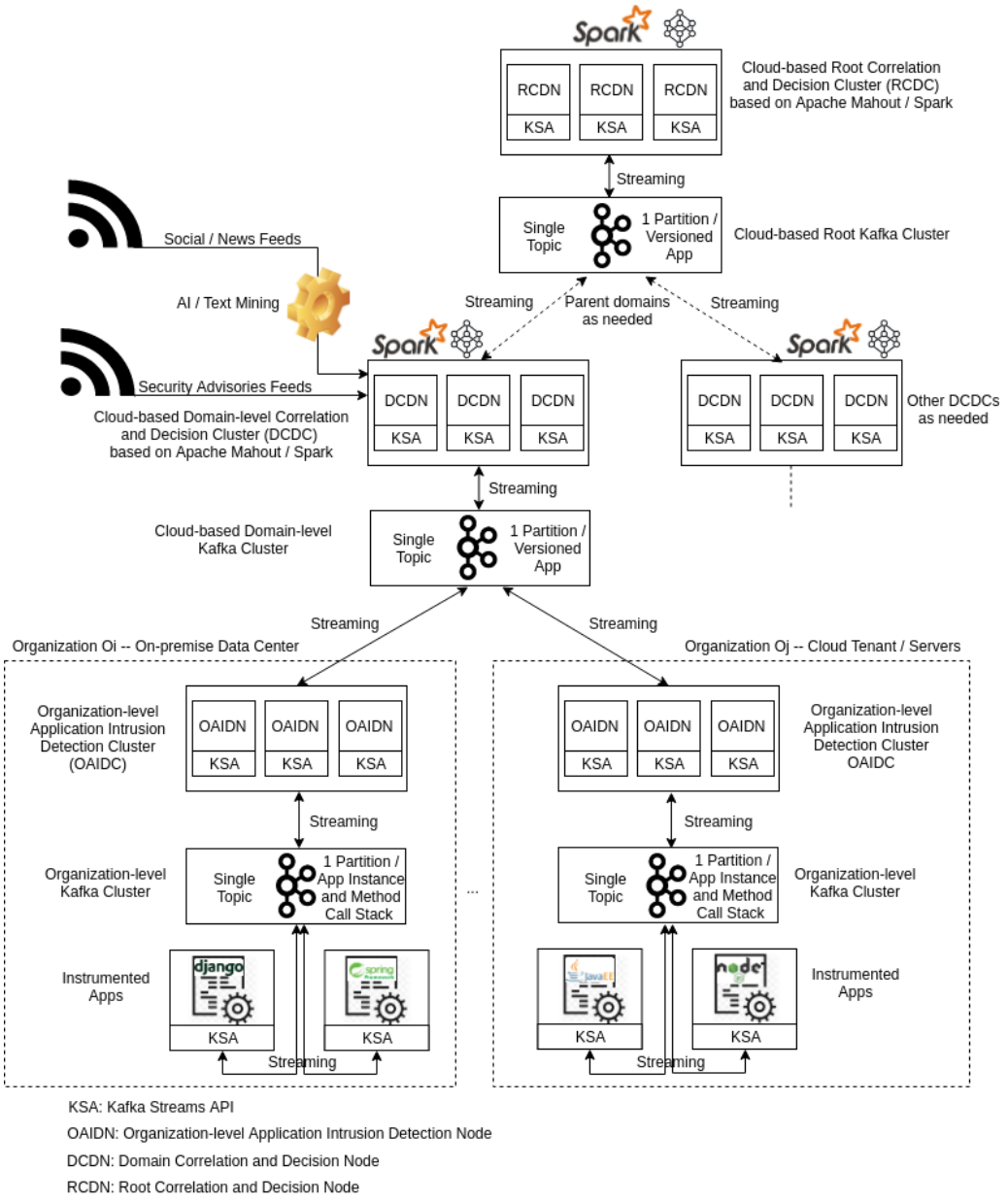

Fig. 1. Collaborative Cloud-based Intrusion Detection Architecture

list can be used by concerned/subscribed parties to prioritize and optimize their application monitoring. Our algorithm shall manage the bidirectional relationship between the reputation of each party and its reported alarms. Indeed, while reputation shall be based on the accuracy of reported alarms, it shall also reflect on the correlation weight of these alarms.

We could base our alarm correlation on a score aggregation technique from the Multiple Attribute Decision Making (MADM) field. As an example, the spectral method [17] could be a good starting point. We would model our call stacks as candidates that are scored or ranked by our parties, considered in this context as sources of information, voters or judges. This requires specifying a score for each alarm or at least a rank from which a score can be derived. It would be a good fit for our application-level unsupervised outlier-based intrusion detection framework [3] as it already provides such a score.

However, MADM methods suppose that every judge provides a score for every candidate. Other improved methods alleviate this constraint, but still require a minimum overlap between candidates scored by different judges [18]. Since in our case we may have thousands of applications with thousands of call stacks to score by thousands of judges, finding those minimum overlaps would not be guaranteed, or at least not in a linear time. Moreover, each aggregated score must be continually recomputed as new alarms are received. Therefore, we had to design a new correlation method.

 ISBN: 978-1-61208-719-1

For a given application with an internal identifier (*aid*) mapped to the SHA-512 of its SWID, we consider a matrix $A^{aid}_{c,p}$ that represents the alarms received from p parties about c call stacks. So A^{aid}_{ij}reflects alarms reported by party j about call stack i. A^{aid} evolves through time with the alarm input stream. Part of A^{aid} evolution, p and c are supposed to grow as new parties related to the given application join the system or alarms about new call stacks are reported. We also consider a $R^{aid}_{p,1}$ vector that represents the reputation of the p parties and evolves through time depending on the accuracy of alarms reported by each party about the given application. Finally, we consider a vector $S^{aid}_{c,1}$ that represents the scores of the c call stacks. Based on these scores, call stacks are sorted before being sent back to the p parties. For simplicity, $A^{aid}_{c,p}$, $R^{aid}_{p,1}$ and $S^{aid}_{c,1}$ will be referred to as A, R, S respectively.

As shown in Algorithm 1, we create three observers: O_1, O_2 and O_3. O_1 is an event observer that updates S and R for each predefined number of received alarms. O_1 also sends the updated ordered list of call stacks (*SortedCS*) to concerned parties. O_2 is a stream observer that updates A for every received alarm. O_3 is a time observer that updates A periodically to take account of aging.

1) Reputation and Score Manager: Our method is based on an iterative, bidirectional, never-ending relationship between the reputation of parties and the accuracy of their reported alarms. Lines 10 and 11 of Algorithm 1 reflect such a relationship. Each element of S, representing the score of a call stack, is computed as a weighted sum of reported alarms. The weights reflect the reputation of parties that have reported these alarms (A x R). The other parties will be naturally excluded from giving any judgement as their corresponding A cells are null. Likewise, each element of R, reflecting the reputation of a party, is computed as a weighted sum of reported alarms. The weights are the scores of call stacks that have been reported by these parties (A^T x S). The intuition here is that the more a party reports about a higher-score call stack, the higher its reputation will be. Nevertheless, there is a pitfall that could be exploited by parties to easily acquire a good reputation. Indeed, as ordered lists of call stacks are published to all subscribers, any party could just "vote on the winner(s)" by echoing top-ranked call stacks back to the system. This will be addressed in the next subsection. Otherwise, our method allows parties to join or leave at any time, as well as to report or not alarms about any call stack at their own discretion.

2) Alarm Observer: The alarm observer is responsible for initializing A, R and S elements, as well as continually updating A elements. As shown on line 18 of Algorithm 1, whenever a new party joins the system, the alarm observer sets its reputation to 1. Similarly, whenever an alarm is reported about a call stack for the first time, it sets the score of the call stack to 1 as shown on lines 25 of Algorithm 1. It also sets the alarm cells of the same call stack to 0, except for the current alarm cell that is set to 1 as shown on lines 28 and 30 of Algorithm 1 respectively. Line 35 is the most important line of the alarm observer. It determines how an A cell gets

Algorithm 1 Reputaion-based Alarm Correlation

```
1:  A ← [][]
2:  R ← []
3:  S ← []
4:  SortedCS ← []
5:  c ← 0
6:  p ← 0
7:  changed ← false
8:  procedure REPUTATION AND SCORE MANAGER: O1
9:      if changed then
10:         S ← A x R
11:         R ← A^T x S
12:         SortedCS ← Sort call stacks based on S
13:         if SortedCS has changed since last time then
14:             encapsulate SortedCS as A2MEF and stream
15:         changed ← false
16: procedure ALARM OBSERVER: O2(callstack, party)
17:     if new party then
18:         R[p] ← 1
19:         bindParty(party, p)
20:         pi ← p
21:         p ← p + 1
22:     else
23:         pi ← getPartyIndex(party)
24:     if new callstack then
25:         S[c] ← 1
26:         for index in 0..p-1 do
27:             if index != pi then
28:                 A[c][index] ← 0
29:             else
30:                 A[c][index] ← 1
31:         bindCallStack(callstack, c)
32:         c ← c + 1
33:     else
34:         ci ← getCallStackIndex(callstack)
35:         A[ci][pi] ← A[ci][pi] + 1/S[ci]
36:     changed ← true
37: procedure TIME OBSERVER: O3(agingFactor)
38:     A ← agingFactor . A
```

updated when the corresponding call stack has already been reported, either by the same party or other parties. The idea here is to favor "breaking news". An alarm about a call stack whose score is already high does not "help much". This is why we update the cell by adding a component as a decreasing function of the call stack score ($1/S[ci]$). More importantly, this addresses the shortcoming highlighted in the previous section. Echoing top-ranked call stacks back to the system will not help parties grow their reputation any faster.

3) Time Observer: The past is important, but the present is more relevant. While learning from previous events, live information should be given a higher weight. The time observer fulfills this objective by introducing an aging factor. By

multiplying *A* by an *agingFactor* on line 38 of Algorithm 1, we increase the relative effect of live updates by the alarm observer. It also reflects on *R* and *S* when they are updated by the reputation and score manager. The *agingFactor* is to be tuned through experiments. A typical value would be 0.8.

IV. Conclusion and Future Work

In this paper, we presented our framework for Collaborative Cloud-based Application-level Intrusion Detection and Prevention, which extends our Application-level Unsupervised Outlier-based Intrusion Detection and Prevention framework by leveraging the benefits of CIDSs. We designed a collaborative intrusion detection architecture made of three levels: the organization level, the domain level and the overarching root level. This hierarchical architecture combined with streaming and clustering offers very good privacy, scalability, accuracy and resilience tradeoffs. Moreover, the adoption of the cloud as a cost-effective and elastic platform allows to handle big data generated by millions of applications as alarm streams.

We also specified a lightweight Application Alarm Message Exchange Format (A2MEF) to support collaboration among the different stakeholders. Finally, we designed a reputation-based alarm correlation algorithm that, given a stream of alarms reported by different parties, emits an ordered list of methods and specific call stacks per application. This list can be used by concerned parties to optimize their application monitoring. Our algorithm manages an iterative, bidirectional, never-ending relationship between the reputation of parties and the accuracy of their reported alarms. It also aims at coping with known attacks against CIDSs.

We have started the implementation of our framework in order to evaluate its effectiveness and efficiency. This will allow us to fine tune the proposed architecture, the message exchange format, as well as the alarm correlation algorithm. We are faced with two main challenges: dynamically (un)instrumenting application methods without creating an unacceptable overhead, as well as using the right AI techniques and tools to mine the unstructured social and news feeds.

References

[1] E. Vasilomanolakis, S. Karuppayah, M. Muhlhauser, and M. Fischer, "Taxonomy and survey of collaborative intrusion detection," *ACM Computing Surveys*, vol. 47, no. 55, pp. 1–33, 2015.

[2] C. V. Zhou, C. Leckie, and S. Karunasekera, "A survey of coordinated attacks and collaborative intrusion detection," *Computers and Security*, vol. 29, pp. 124–140, 2010.

[3] O. Iraqi and H. E. Bakkali, "Application-level unsupervised outlier-based intrusion detection and prevention," *Security and Communication Networks*, 2019.

[4] G. Meng, Y. Liu, J. Zhang, A. Pokluda, and R. Boutaba, "Collaborative security: A survey and taxonomy," *ACM Computing Surveys*, vol. 48, no. 1, pp. 1–42, 2015.

[5] B. S. Feinstein and G. A. Matthews, "The intrusion detection exchange protocol (idxp)," *The Internet Engineering Task Force (IETF)*, 2007.

[6] H. Debar, D. A. Curry, and B. S. Feinstein, "The intrusion detection message exchange format (idmef)," *The Internet Engineering Task Force (IETF)*, 2007.

[7] J. R. Douceur, "The sybil attack," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, (London, UK, UK), pp. 251–260, Springer-Verlag, 2002.

[8] J. Lee, M. Park, J. Eom, and T. Chung, "Multi-level intrusion detection system and log management in cloud computing," in *13th International Conference on Advanced Communication Technology (ICACT2011)*, pp. 552–555, Feb 2011.

[9] W. G. McGee, "System and method for intelligent coordination of host and guest intrusion prevention in virtualized environment," May 14 2013. US Patent 8,443,440.

[10] J. Shi, Y. Yang, and C. Tang, "Hardware assisted hypervisor introspection," *SpringerPlus*, vol. 5, no. 1, p. 647, 2016.

[11] C. Maiero and M. Miculan, "Unobservable intrusion detection based on call traces in paravirtualized systems," in *Proceedings of the International Conference on Security and Cryptography*, pp. 300–306, IEEE, 2011.

[12] Z. Chiba, N. Abghour, K. Moussaid, A. El Omri, and M. Rida, "Novel network ids in cloud environment based on optimized bp neural network using genetic algorithm," in *Proceedings of the 3rd International Conference on Smart City Applications*, p. 26, ACM, 2018.

[13] S. Bharadwaja, W. Sun, M. Niamat, and F. Shen, "Collabra: a xen hypervisor based collaborative intrusion detection system," in *2011 Eighth International Conference on Information Technology: New Generations*, pp. 695–700, IEEE, 2011.

[14] C. C. Lo, C. C. Huang, and J. Ku, "A cooperative intrusion detection system framework for cloud computing networks," in *2010 39th International Conference on Parallel Processing Workshops*, pp. 280–284, IEEE, 2010.

[15] S. Gupta and P. Kumar, "System cum program-wide lightweight malicious program execution detection scheme for cloud," *Information Security Journal: A Global Perspective*, vol. 23, no. 3, pp. 86–99, 2014.

[16] "ISO/IEC 19770-2:2015 - Software Identification Tag," tech. rep., ISO/IEC, 2015.

[17] M. Xiao and Y. Wang, "Score aggregation via spectral method," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 451–457, 2017.

[18] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank aggregation methods for the web," in *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, (New York, NY, USA), pp. 613–622, ACM, 2001.

 ISBN: 978-1-61208-719-1