Situational Awareness through Reasoning on Network Incidents

Anna Cinzia Squicciarini
College of Information Sciences and Technology
Pennsylvania State University
asquicciarini@ist.psu.edu
University Park, PA
United States

Giuseppe Petracca
Computer Science and Engineering
Pennsylvania State University
gxp18@psu.edu
University Park, PA
United States

William Horne
Hewlett-Packard Research Lab
bill.horne@hp.com
Princeton, NJ
United States

Aurnob Nath
Computer Science and Engineering
Pennsylvania State University
axn218@cse.psu.edu
University Park, PA
United States

ABSTRACT

Corporations worldwide work with teams of often dedicated system administrators to maintain, detect and prevent network infringements. This is a highly user-driven process that consumes hundreds (if not thousands) of man hours yearly. User reporting, the basis of most of these incident detection systems suffers from various biases and leads to below-par security measures. In the paper, we provide an approach for near real-time analysis of ongoing events on controlled networks, while requiring no end-user interaction and saving on system administrator's effort. Our proposed solution, ReasONets, a lightweight, distributed system, provides situational awareness in case of network incidents. ReasONets combines aspects of anomaly detection with Case-Based Reasoning (CBR) methodologies to reason about ongoing security events in a network, including their nature, severity and sources. We build a fully running prototype of ReasONets, to demonstrate the accuracy of the system, in doing reasoning and inference on the network status by exploiting events and network features. To the best of our knowledge, ReasONets is the first of its kind system combining detection and classification of network events with real-time reasoning while being capable of scaling up to large network sizes.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process; C.3 [**Computer Systems Organization**]: SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS

CODASPY'14, March 3–5, 2014, San Antonio, Texas, USA.

http://dx.doi.org/10.1145/2557547.2557562.

Keywords

Situational awareness,Incident detection, Case-base reasoning

1. INTRODUCTION

Detection is just one step in an overall plan to handle security incidents. Once detected, security incidents are typically forwarded to a Security Operation Center (SOC) and to Computer Security Incident Response Team (CSIRT), where security analysts triage and investigate incidents and formulate a response [33]. This work is labor intensive and relies on extensive domain expertise. Further, user reporting, the basis of most of these incident detection systems, suffers from various biases and leads to below-par security measures. To this date, despite large amount of work in intrusion and anomaly detection (e.g. [34, 9, 8]), there is a great need to improve situational awareness upon incident detection [26], as current incident detection systems are often complex, rigid, and provide limited feedback on the incidents being detected. The objective of our work is to tackle these issues by means of an approach for deep and real-time understanding of ongoing events in a controlled network. In particular, we aim to provide in-depth reasoning of reported anomalies to determine the nature and severity of occurring security events, by using only a small amount of network information.

We propose ReasONets, an effective and lightweight system, able to process and reason about anomalies and incidents observed in closed networks. ReasONets leverages anomaly detection components with Case-Based Reasoning (CBR) methodologies [10, 16], in order to provide situational awareness in case of network incidents. Underlying the Case-Based Reasoning process deployed within ReasONets is the understanding that no security event will ever be identical to previously experienced incidents in absolute terms, but should show enough similarities to be qualified as an event of a certain type. The understanding of anomalous events is not gathered from rules or general statistics, but by the

analysis of *cases*. Each case represents a specific type of event which is already analyzed and described by means of a flexible set of metrics collected on-the-fly. Furthermore, we control the uncertain and inaccurate information collected in real-time, by exploiting the Fuzzy Logic Theory [27]. In particular, we use a Fuzzy Clustering Algorithm, as well as a Fuzzy-based Ranking for disambiguating incidents, in case a clear mapping to known cases cannot be determined.

As a result, rather than acting as a log-based network intrusion detection system which simply raises alerts or labels an incident that matches static signatures, ReasONets leverages information *from* an anomaly detection system to achieve a more accurate understanding of network specific incidents. Not only does it specifically locate an incident, but it also determines whether multiple machines were involved, whether they were affected by the same incident, what type of incident it was, whether it was human-driven, accidental, malware-driven etc. Further, it provides in-depth information about the machines connected to the incident, their behavior and (for certain cases) the incident cause.

Our system is also highly adaptive to real-time traffic events: if no previous case matches an observed event, we determine when new cases are to be adapted and, possibly, merge existing previous cases.

The detection of a machine accessing malicious domains is a simple yet effective example demonstrating how ReasONets differs from common Security Event Management (SEM) systems. SEM systems usually rely on Black Lists and White Lists, through which it is difficult to infer if a "new" domain is a good one or not [8]. ReasONets combines to this basic information a set of metrics that allow us to infer the nature of a domain on-the-fly. In details, our system measures the content type (i.e. mature, political, sport), and the geographical distance of the domain with respect to the closest known malicious domain. Further, we check if the registrant is an organization that owns other malicious domains, and if the domain is in the same network of a well-known malicious domain. The added value of our system is the capability to do inference on completely unknown domains, allowing us to obtain early identification of malicious domains, and their relationship (if any) with existing ones.

We deployed a fully running prototype of ReasONets, and carried out extensive experimental evaluation in actual networks of various sizes, using a real-world dataset. Our experiments demonstrate good accuracy and reasonable performance, even when the system is posed under great stress with a large number of incidents.

To the best of our knowledge, ReasONets is a first of its kind system combining detection and classification of network events with real-time reasoning while being capable of scaling up to to large network sizes. Our approach is however inline with current industry and government efforts for systematic classification of network incidents to increase situational awareness by means of new metrics to supplement current measures of detection [22]. In particular, our cases may be considered instances of classified threats, which can be deployed under the STIX format promoted by Mitre [21].

The rest of the paper is organized as follows. In Section 2, we compare this work with related projects. Next, we provide an overview of the ReasONets system. We then present the main components of our architecture, and discuss their functionality. In Section 5, we describe the ReasONets prototype. Section 6 is devoted to experimental evaluation. We conclude in Section 7.

2. RELATED WORK

Several host-based and network-based approaches to incident (and intrusion) detection have been proposed in literature [34, 30, 9, 10, 4]. Some of these approaches are anomaly-based, and rely on models based on traffic or events generated during a normal operation of the system [13, 32, 12, 6]. Others are based on matching observed events with predefined signatures of known incidents [15, 31]. ReasONets leverages these well-known paradigms to provide more fine-grained analysis of new and well-known incidents. Without resorting to either complex statistics or static signatures, ReasONets enables processing and reasoning of conventional anomaly detection input toward accurate and customized incident analysis.

Within the extensive body of work on anomaly detection, the closest approaches to ours are from Guha and colleagues [10] and Esmaili [7]. Guha and colleagues develop an incident detection system for wireless networks, with an underlying CBR. The system architecture and capability are however fundamentally different from that of ReasONets. In Guha, the CBR is distributed, and the main focus is on computational and network performance, rather than accuracy. The authors provide no specific discussion on how the cases are built, the similarity metric used and achieved accuracy. The level of adaptiveness achieved by the approach is unclear, given the lack of empirical evaluation. Esmaili and colleagues [7] discuss a high level model for integrating case-based reasoning techniques in intrusion detection systems. The authors highlight the benefit of introducing CBR in such systems and provide important insights on the challenges of integrating CBR in a real-time detection system. However, their focus is purely on intrusions and therefore on the analysis of audit trails, which is very different from our approach. Further, they do not develop any sensitive metrics for cases similarity and indexes nor do they provide any working prototype. Other loosely related approaches exploit the modeling of network traffic, by analyzing the packet payload [8]. These approaches are often inefficient due to the high amount of data to process, and the difficulty in analyzing encrypted packets.

Some of the recent works on incident and intrusion detection focus on the problem of false detection rate [36], in that false positive rates continue to be a significant problem in most current intrusion detection systems [2]. An interesting approach is proposed by Zomlot and colleagues, who present an efficient algorithm for carrying out Dempster-Shefer belief calculation on an intrusion detection alert correlation graph, to reduce false positives based on computed belief scores. The problem of false positive rates is also elegantly tackled in Disclosure [3], wherein Bilge and colleagues also studied a botnet detection system that uses NetFlow records to distinguish C&C channels from benign traffic. To reduce false positive rate, authors incorporate a number of external reputation scores into their system's detection procedure. As shown in our experimental evaluation (Section 6), in ReasONets, fine tuning and careful configuration are needed to control false positives. Hence, it is part of our future work to study how to adopt approaches such as the one proposed by Zomlot et al. or Bilge to reduce the false positive rate.

3. OVERVIEW OF THE REASONETS SYSTEM

The ReasONets system provides a knowledge-based reasoner system for detection and analysis of real time incidents in controlled networks. Specifically, ReasONets relies on a layered architecture, which consists of adopting anomaly detection first, and reasoning afterward. In order to function effectively, anomaly detection is based on the preliminary analysis of patterns of ordinary network activities within the local network, through the definition of standard network traffic patterns, or signatures, generated by means of network metrics. The anomalies identified using the standard patterns as reference represent the starting point of case-based reasoning (CBR).

In general, the idea of using CBR is to dynamically solve complex questions by comparing observed events with similar cases, whereby each case describes a known significant event [16], and is described using symbolic descriptors. In our context, reasoning not only provides indications about the severity of the anomaly, but also it provides large amount of information about the incident being experienced. This information includes, for example, the type of malware being used to attack the network or whether an insider is suspected. Each case is described by a set of relevant network features, collected during an initial training and adapted to the observed incidents over time. Case mapping is achieved by means of fuzzy logic operations, so as to control and abstract away from the coarse network data collected by the loggers. In particular, we use a fuzzy clustering algorithm, as well as a fuzzy-based ranking to determine the degree of similarity between incidents and classify an observed incident in a known or newly experienced case.

From an architectural standpoint, ReasONets may be deployed either as a fully centralized system or in a distributed fashion. In either deployment, the main actors are the monitored machines and a standalone network administrator machine. In the distributed settings, on monitored machines, network traffic logs are generated and optimized. The network traffic logs provide data about the machines' network activities, the amount of data being transmitted from these foreign locations, type of requests being made (GET, POST, DELETE) and how long a connection to a foreign address lasts. The network logs are kept along with local statistics at the machines level, to provide information about the network activities of the machine during these sessions with external locations. In case of centralized deployment, some of the functionality at the client is moved to network proxies or log collectors, that can process and store the data, and produce the statistics required for analysis.

On the main system, the administrator's machine, the ReasONets is in charge of aggregating all these logs from the participating monitored machines, gathering additional information, and analyze these enriched logs. This additional information takes the form of metrics related to unknown domains, traffic comparison with malware clusters, etc. It is also required for creating and dispersing traffic rules to the monitored machines to safeguard against future incidents.

We report the overall flow of execution of the ReasONets system in Figure 1. The input parameters are represented by the data and the network logs obtained from monitored computers connected to the network, whereas the output parameters are detailed indications about the suspected nature of the incident, its severity, and the confidence associated with this assessment. Further, feedback on possible actions to be completed to address the incident is provided to the ReasONets user.

4. THE REASONETS CORE COMPONENTS

The ReasONets is organized into two key layers: anomaly detection and reasoning. We begin this section with a discussion of the anomaly detection component. Reasoning, which is the core of our solution, is discussed next.

4.1 Anomaly-Detection Component

Similar to many anomaly-based systems, the ReasONets Anomaly Detection Component (ADC) operates in two different modes: training mode and monitoring mode. As we discuss next, ReasONets's ADC merely focuses on anomaly discovery, rather than incidents and intrusions analysis. Therefore, it purposely lacks sophisticated metrics and anomaly models, in that incident analysis is performed by the additional layers of the system. This component is currently deployed by leveraging the SNORT intrusion detection system, as we discuss in Section 5.

4.1.1 Training mode

The first step of the ReasONets ADC aims at defining baseline network patterns of the controlled network, to later on search for network events that do not meet these observed and ordinary patterns. Hence, during training ReaSOnets collects patterns of regular traffic of both portions of the monitored network as well as of each individual machine within the monitored network. The normal behavior of a machine is measured in terms of network traffic and application (HTTP) traffic generated, plus performance metrics that measure the average usage of the CPU and RAM, over a certain time interval.

We specifically log incident-free network activities of each monitored machine, by using a simple and lightweight representation of the traffic. A single entry in a log file has information stripped from packet headers, and specifically include the following basic elements:

< `TimeStamp`, `Protocol`, `SourceIP`, `SourcePort`, `DestIP`, `DestPort` >

To extract patterns of groups of machines, the generated logs of all systems in the network are simultaneously processed by the ReasONets. By means of pattern matching methods, ReaSOnets then generates and stores network "signatures" indicative of groups of machines displaying the same traffic [17]. In general, two or more machines belong to the same group if in the generated logs at least two entries among `Protocol`, `SourceIP` or `DestIP`, `SourcePort` and `DestPort` match.[1]

At the end of the training phase all the regular traffic is stored in traffic "clusters". Starting from these clusters, the ReasONets generates new logging rules that are sent to the logger running on the monitored machine, to exclude regular traffic from future logged activities. Intuitively, machines belonging to the same cluster receive the same set of logging rules. In particular, we generate a new `pass` rule for each cluster generated, which allows to discard the traffic belonging to it from log activities. For example, a `pass` rule in the SNORT rules file has the following form:

[1]Only one of the IP (destination or source) needs to match, due to the fact that the other IP refers to the IP of the monitored machine.

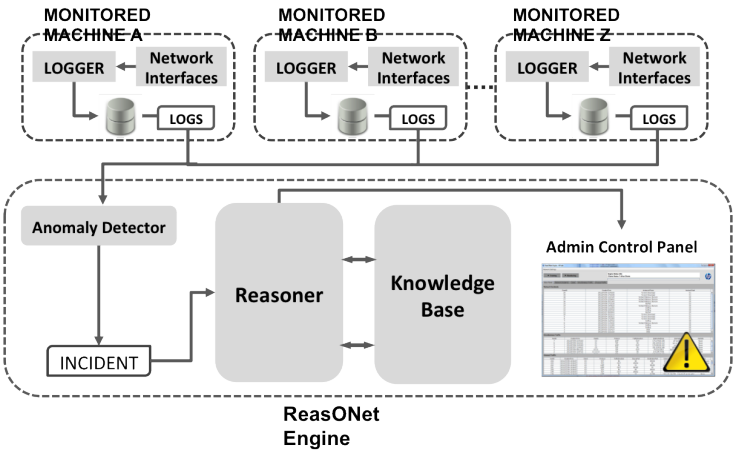

Figure 1: Overall Approach

`pass IP 130.203.157.23 80 -> 123.55.1.140 2134`

This rule states that all the IP traffic going from the machine with IP equals to 130.203.157.23 by using the port number 80, and going toward the machine with IP equals to 123.55.1.140 by using the port number 2134, must be ignored by SNORT.

4.1.2 Monitoring mode

The ADC, upon filtering out the ordinary traffic by training the model, is in charge of implementing a two-variate model to check whether the reported suspicious traffic is in fact representative of anomalies. Two types of anomaly are detected: (1) single machines that open connection toward unknown domains/machines, or toward known domains/machines but using unusual protocols or port numbers; (2) subset of machines with same sequence of connections toward specific unknown domains/ machines, not necessarily in the same order. Here, as unknown domain/machine we mean a web domain or machine external to the monitored network, never visited/contacted before by any internal machines.
The focus on these two basic anomalies, as a first level of detection, is motivated by the assumption that malicious activities most likely exhibit some specific communication pattern that is different from the norm, i.e. contacting destination that the internal hosts would not have contacted otherwise, misusing protocols, or using uncommon port number that internal host have never used before [24]. In fact, in an ordinary operation of the network, the IP addresses with which a group of hosts communicates exhibit stability over time. We specifically group the machines' behavior in clusters, since one of the most common cause of network failure, and security issues, is malware. Since malware often infects multiple machines within a large network, then the infected machines should have closely similar characteristic behaviors [24]. With the first type of anomaly we address any other case in witch only one machine is involved, for example clumsy actions performed by a user that is operating on a monitored machine.

In order to detect the above anomalies, we employ a set of metrics for machine profiling. The metrics (details are reported in Appendix A, for each class of metric) include:

- **Network Level Metrics**, aiming to measure network traffic activity. Examples of metrics in this category are: *Average Flow Packet Count* and *Ratio Flow Count per Average Package Size*, that allow to measure the machine network activity.
- **Application Level Metrics**, to approximate the HTTP traffic generated by each monitored machine, during its interaction with external machines and the Internet. Table 6 in Appendix reports additional examples of Network and Application Level metrics, respectively.
- **Machine Performance Metrics** aiming to measure the work load of monitored machine in term of CPU and RAM, metrics collected directly from the machine involved in our analysis.

Whenever an anomaly is detected in the traffic generated by the monitored machines, an additional set of metrics is gathered for each visited domain. The objective of this new set of metrics is to investigate whether the visited domain may be the source of the incident, and therefore related to the machines anomalous behavior. These metrics are then combined to derive the *maliciousness level* of a particular domain among all the domains involved in a suspect incident. Below we include simple descriptives for the metrics generated, which are normalized before computation.

- *Same Network (sn)* indicates if the suspicious domain is in the same network of a well-known malicious domain
- *Spatial Distance (sd)* between the suspicious domain and its closest well-known malicious domain
- *Bad Owner (reg)* indicates if the domain registrant is the owner of a well-known malicious domain
- *Blacklisted (b)* states if the domain is Blacklisted
- *Whitelisted (w)* states if the domain is Whitelisted

The state of a domain is determined to be blacklisted through comparison with an exhaustive database available from ShallaList [29], integrated with few other well-known lists [28, 35]. Starting from the above metrics, the domain's maliciousness level is calculated simply as follow:

$$dml = \sum_{i=\{sn,sd,reg,c,b,w\}} w_i * i \qquad (1)$$

where $i = \{sn, sd, reg, c, b, w\}$ denotes the metrics and their corresponding weight w_i, calculated in order to reflect the relevance of the feature for the overall domain maliciousness level (dml) value. The vector of weights $\overline{w}$ can be computed by training with a number of known malicious domains.

Note that dml is only useful for certain classes of incidents, and it is considered as an add-on, rather than a core metric. It is therefore disregarded in case of a suspect event where no malicious domain might have been visited.

4.2 Reasoning

The core of our architecture consists of Case-Based Reasoner (CBR) [10] on the events detected by the higher system layers. A case represents a known incident or security event, which has been experienced in the system and addressed by administrators. Each case is represented by a vector of significant features, each feature denoting a metric and the corresponding value range. For the purpose of our analysis we adopt a knowledge base (KB) which currently collects and models two classes of cases. First is the set of cases which represent the most common network spread malware (e.g., DoS, Virus, Botnet, Worms, Keylogger and Spyware). Second is the set of non-malware related incidents, that can be observed through network analysis. For example, voluntary access to a domain with adult-only content, as well as cases of unauthorized access to a monitored machine from a remote one. The cases are obtained using a hybrid approach that combines empirical evaluations and analysis of well-known security incidents affecting small enterprise networks. Additional cases are dynamically added to the KB as they are experienced, according to a set of adaptation rules within the model.

Previous attempts have used threshold-based approaches to detect incidents, with and without the aid of a CBR system. However, using simple thresholds may not provide sufficient knowledge about the event, and would fail in case of hybrid events, that appear similar to more than one incident [14]. To cope with these issues, we have designed our reasoner by adopting a multi-layer approach that builds on *Fuzzy Logic* and on *ad hoc-case Fuzzy Ranking*.

The goal is to develop appropriate similarity metric for cases analysis that are defined as a collection of feature comparison results, and use fuzzy rules to specify how these intermediate results are combined. Although each case feature may require a different type of comparison, using fuzzy, the result of the comparison is a similarity assessment between the same case feature of the problem specification and of a case from the case archive. In addition, fuzzy logic handles situations where no-crisp answers can be found, which often occurs when one needs to determine to what degree an incident is related to a known case. Beside fuzzy-based analysis, we compare selected and weighted features of relevant cases for any potential new input, in order to check for relevant information that is most indicative of a case, and therefore can help discern the nature of the incident.

4.2.1 Case Representation

ReasONets models a case c as follows:

$$< cid, \{m_1, ..., m_k, dml\}, r_c, p_c, \overline{w}^c > \quad (2)$$

where cid is the case identifier, $\{m_1, ..., m_k\}$ represents a set of triplets $m_i = \langle min(f_i), mid(f_i), max(f_i)\rangle$ for each collected metric $f_i \in [1, k]$. dml represents the domain maliciousness level, per the discussion in Section 4.1.2. r_c represents the risk level associated to the case, p_c represents the likelihood of the case. The latter two metrics can be derived either by gathering local statistics collected within the organization or through public data associated to the frequency and the impact of the incidents described by the case. Finally, $\overline{w}^c$ is the vector of weights associated to the case's metrics. Weights model how representative certain metrics are for the case they are associated with. For example, network flow metrics are of great relevance during denial of service attacks, but may matter little in case of access to prohibited domains. We discuss in detail the use of the last three components in our discussion of the case analysis.

Instances of a case c are stored in analogous way as the original case they belong to:

$$inst =< iid, \{m_1, ..., m_k, dml\}, c, deg(c) > \quad (3)$$

Here, $deg(c)$ denotes the membership degree of the instance to the case c which it is assigned to. This information is added upon incident assignment.

4.2.2 Case Retrieval and Ranking

We propose a multi-step ranking algorithm for identifying the best match among possible candidate cases. The algorithm allows us to ascertain not only to what extent the incident belongs to a given case, but also whether a new ad hoc case should be generated.

Given an observed incident inc, the first task consists of computing its membership degree with all possible cases in the knowledge base. This value, denoted as $deg_{inc}(c)$, for case c is to determine how "similar" inc is to the profiled cases. We provide two computations of deg_{inc}, depending on whether the system already has record of observed instances for the case or not. The intuition for both formulae is that membership is computed as a function of the metrics vector's distance from other occurrences of the case, and these occurrences' memberships.

Specifically, let $|m|$ denote the cardinality of the set of triplets $\{m_1, \ldots, m_k\}$ with $m_i = \langle min(f_i), mid(f_i), max(f_i)\rangle$ of Equation 2, representing the values stored for each case's metric f_i. Let $diff(f) = \max(f) - min(f)$. If there are recorded instances (denoted as $inst$) of a case c, $deg_{inc}(c)$ leverages the information collected for each previous incident of the case, and it is computed as follows:

$$\frac{\sum^{inst \in c}\left[deg_{inst}(c) \times \sum^{|m|}\left(diff(f) - \left|\left|\frac{inc(mid(f))}{\left(\frac{\max(f)}{diff(f)}\right)} - \frac{inst(mid(f)}{\left(\frac{\max(f)}{diff(f)}\right)}\right|\right|\right)\right]}{\sum^{|f|}\left(diff(f) - \left|\left|\frac{inc(mid(f))}{\left(\frac{\max(f)}{diff(f)}\right)} - \frac{inst(mid(f))}{\left(\frac{\max(f)}{diff(f)}\right)}\right|\right|\right)} \quad (4)$$

Here, $max(f)$ and $diff(f)$ are used for normalization, whereas $inc(f)$ and $inst(f)$ represent respectively the measured value of f for the current incident inc and the instance $inst \in c$. Finally, $deg_{inst}(c)$ is the membership degree of a previously experienced incident associated with case c, and therefore labeled as instance ($inst$). Note that in Equation 4, we use all the instances of case c stored in the KB, so as to increase the accuracy of the detection algorithm.

As seen in Equation 4, the assigned memberships are influenced by the inverse of the distances from the existing instances of the case and their class memberships. The in-

verse distance serves to weigh a vector's membership more if it is closer and less if it is farther from the vector under consideration.

If the case has no recorded instances, the membership degree is instead computed in a similar manner, by relating the metrics of the incident directly with the case record, as follows:

$$\sum^{|m|}\left(diff(f) - \left|\left|\frac{inst(mid(f))}{\left(\frac{\max(f)}{diff(f)}\right)} - \frac{c(mid(f))}{\left(\frac{\max(f)}{diff(f)}\right)}\right|\right|\right) \quad (5)$$

In the above equation, $c(mid(f))$ represents the value of the metric f that is associated with the case c.

Next, given the cases C=$[c_1, \ldots c_n]$ ordered according to the corresponding membership degree (per Equation 4) for incident inc $[deg_{inc}(c_1), \ldots, deg_{inc}(c_n)]$ such that $deg_{inc}(c) > 0.1\ \forall c\ \in C$, we compute K-NN clustering algorithm among all incident instances recorded for $c_1, .., c_n$. Note that this step is particularly useful for large KBs with a rich history of incidents, since the computed membership degrees represent an average of the incident similarity with the case, but these values may vary greatly even within incidents of a same case. Hence, they may not represent crisp indicators of similarity. This step is instead omitted or less relevant if few cases' instances are recorded. In the latter case, the incident may simply be assigned to the case with the highest membership degree. If the majority of the close neighbors in the K-NN set belong to a same case c_i with a degree above a system defined threshold β (set to 0.65 on our experiments), we model the incident as an instance of c_i.

If no dominant case is found, that is

$$deg_{inc}(c_1) - deg_{inc}(c_2) < \epsilon$$

where ϵ is a small threshold, we compute an additional ranking for the cases in C. The additional ranking algorithm requires recomputing a refined membership degree $deg_{inc}(c)$, determined using fuzzyfied feature values, to control the noise introduced by comparing empirically collected values which may be coarse or not accurate. At the core of the algorithm is the following computation of membership degree to replace $deg_{inst}(c)$ in Equation 4. This is again computed for every case the incident may be associated with - given the crisp membership degree values obtained in the previous step of the algorithm. We first compute the distance among weighted metrics of previous instances:

$$\alpha = \sum^{|m|}(w_c(f) \times Defuzzy[diff(f) - ||Fuzzy(inc(f)) - Fuzzy(c(f))||]) \quad (6)$$

Then, we compute the actual membership degree as follows:

$$deg_{fuz\_inc}(c) = r_c \times p_c \times \alpha \quad (7)$$

Note that $deg_{fuz\_inc}(c)$ differs from $deg_{inst}(c)$ in two ways. First, in computing α we assign a different weight $w_c(f)$ to each specific feature. Further, risk parameter r_c and a likelihood parameter p_c from the case representation in (2), are also taken into account, to classify the incident with a greater accuracy. Finally, α includes fuzzification and defuzzification steps of the measured features, represented in the equation 6 by means of $Fuzzy$ and $Defuzzy$ functions - in accordance with the Fuzzy Theory [27].

If even fuzzy ranking does not reveal a dominant case, ReasONets classifies the incident as new case by starting the case profiling process and stores it in the KB.

Also note that the KB is periodically optimized: If several incidents with same features are mapped on the same subset of cases, we execute a merge operation that allows the creation of a single case representing all the cases in the subset.

5. REASONETS PROTOTYPE

ReasONets was built to be an Operating System (OS) independent incident-detection solution. The system includes multiple monitored client machines in the network, and one federated administrative server machine.

Each client machine is equipped with 3 major components in addition to the ReasONets client. First, each monitored machine hosts and runs SNORT[1], a lightweight network packet sniffer and logger, which generates and stores network traffic logs. Second, each machine also runs "TypePerf" [19], which helps capture client system statistics like processor usage by various processes. TypePerf is pre-installed program in Windows 7 and requires no additional plugins for its usage. Analogous performance loggers are available for Unix and Macintosh OS. The third component is a database where all the network and machine performance logs, once captured, can be stored and appropriately accessed by the admin console machine (e.g. MySQL, or Microsoft SQL Server). All client programs including the client version of ReasONets are designed to be non-intrusive and create minimal interference for the user. Note that in principle, the three client programs could be offloaded to a network appliance that monitors a local subnet. This approach would deploy a decentralized model of ReasoNets, with the following advantages: (1) it would make the management of endpoints easier, (2) it might be more secure since if malware gets on an endpoint it could disable these services, while it is more difficult to compromise an appliance, and (3) it might be more scalable.

The administrative machine is equipped with a MySQL database and the server version of ReasONets. The ReasONets houses the main logic for incident detection, reasoning and learning and is known as the *engine*. The engine is responsible for all database interactions across the machines, for data gathering and network traffic policy dispersal. The engine is configured to retrieve logs from remote clients using an encrypted connection. The input is given by the data obtained from monitoring computers connected to the network, whereas the output are detailed indications about the suspected nature of the incident, its severity, and the confidence associated with this assessment. Feedback and possible actions to complete in order to address the incident are provided to the network administrator, as shown by the interface presented in Figure 2. The admin console is a multi-tab GUI which displays synthesized data on the experienced incidents and their types. In the main control panel, the top half of the GUI displays a processed log of recent network incidents, including creation time, case, and clients involved. The middle window visualizes anomalous patterns of behavior across multiple machines, along with detailed information on the specific of the behavior, including traffic type, direction, external IPs etc. Finally, the bottom window displays unusual traffic originated by individual machines, and that is yet to be processed and assigned to

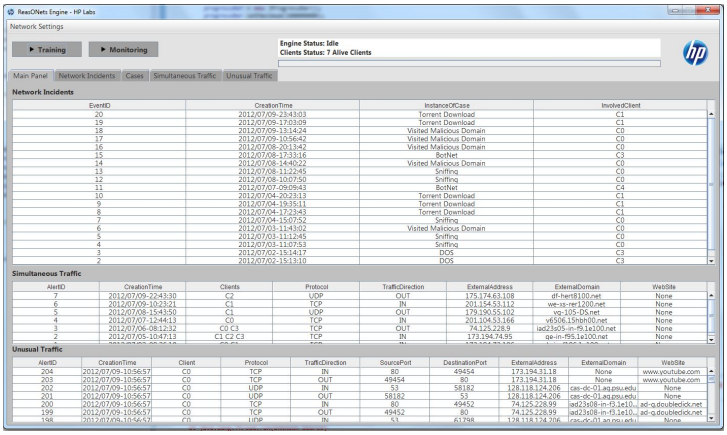

Figure 2: ReasONets console

cases. Other tabs (not shown in Figure) report expanded information on the above data, including a description of the incident and its association with alike cases, possible overlapping cases and feedback for future action.

6. EXPERIMENTAL EVALUATION

In this section, we discuss our experimental evaluation. We first present the datasets used for our analysis, and the experiments we focused on. Next, we discuss our system setup. Finally, we present the results of our experiments.

6.1 Test Data

For the purpose of the experiments, the client prototype, described in the previous section, was modified by discarding the packet sniffing tool SNORT and replacing it with a dataset (DITL_I2-20080317 from PREDICT [25]) which contains 48 hours worth of clean, real traffic IP packet headers collected from an academic ISP. This dataset allows us to generate a constant and varied stream of traffic, in absence of incidents. The data was parsed and fed into the client databases so that each client machine acted as a concentrator and stored data from up to 5 different IP addresses. To this clean traffic data, various traffic records were injected, so as to introduce/replicate/emulate various network incidents.

We generated malicious traffic to specify baseline cases for our tests. To ensure real-world (or at least, realistic) traces, we leveraged existing specialized datasets or generated the actual traffic as ground-truth. When generating malicious traffic, test machines were used, which were first formatted with a clean image of Microsoft Windows (Xp or 7), configured to run a packet sniffer (SNORT) followed by running the programs related to that certain test case (the machines were promptly re-formatted afterwards). A detailed example of the adopted methodology for case generation is reported in Appendix B.

Table 1: Core Cases

Case Id	Case Type
1	REGULAR - No Browsing
2	REGULAR - Browsing
3	Torrent Download
4	Bot Net
5	DOS (Denial of Service)
6	Sniffing
7	Visited Malicious Domain

On average, we observed about 50 incidents per generated case. We report examples of our approach for developing some interesting cases below.

Torrent traffic was studied and replicated by running the actual torrent programs, uTorrent[23] and BitTorrent[5]. Each program was used to download 2 hours worth of data during which, all incoming and outgoing IP headers were recorded. Similarly, BotNet attacks were simulated by accessing/running known malware [18] while sniffing and recording all incoming and outgoing IP headers.

"Visited Malicious Domains" corresponds to the case of users accessing blacklisted or prohibited sites. This case was instantiated by performing browsing sessions of various duration to sites listed on the ShallaList's blacklist [29], a downloadable, regularly updated list of blacklisted websites which is freely available online. Likewise, denial of service attacks (DoS) were studied by analyzing multiple datasets available online for researchers. The first dataset downloaded from CAIDA[11] provided 1 hour worth of anonymized trace data for a DDoS attack which occurred on Aug 4,2007. The other two datasets from PREDICT ("iperf_emulated_attacks-20090621" and "DoS_traces-20020629")[25] provide trace data for DoS related traffic captured from 4 different attacks.

Like for the former cases, the CAIDA dataset too was passed onto the ReasONets system for case-based metric generation and classification. Studying the traffic records from the PREDICT database (4 different attacks lasting for up to 10 minutes), allowed us to specifically isolate the traffic responsible for the DoS attacks. As this contained both single source to destination machine based attack and multiple source to single destination attacks, we simply replicated the same traffic during our testing for simulating an attack on one of our client machines, when required.

Furthermore, in order to model network issues that are not strictly related to the actions of malware but are instead due to the actions of a malicious user, we generated two additional cases. The first involved voluntary access to "restricted" domain; these domains were restricted due to either its mature (e.g. pornographic) nature or malicious content (e.g. malware repository). The second one was the unauthorized access to a monitored machine over a remote connection.

Upon having obtained the desired traces for each case, we stored each of them along with the network metrics of Expression (2). Some of the representative cases are reported in Table 1. Additional cases were subsequently added for testing purposes, inspired by the analysis of patterns of common attacks described in [20]. Subsequently, to generate testing data, we designed ad-hoc scripts that modified the original incidents logs to reflect the same pattern of behavior with customized and disguised dimensions.

6.2 Experiments

We completed several experiments, to check for scalability, performance and accuracy of ReasONets. We briefly introduce each set of experiments below.

- **Performance and Accuracy.** The first set of experiments consisted of 5 rounds for each of the 5 network size configurations. The purpose of this set of experi-

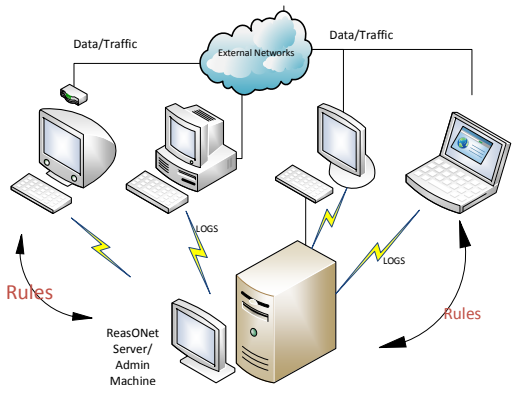

Figure 3: Network layout

ments was to assess the performance and accuracy of the system against 5 types of cases when tested against increasing number of client machines plugged into the system. In addition, we tested the accuracy achieved when varying the weights' feature vector for the tested cases, and studied how features' weights of representative cases can be configured for best results.

- **Stress Test.** We tested the detection rate in case of an increasingly high number of incidents, and in case of a large KB.
- **Comparison with baseline approach.** We aim to compare the accuracy of our system against a standard baseline system which provides basic traffic classification.
- **Scalability.** We complete scalability tests to determine the efficiency and scalability of the system over larger network sizes. Networks of up to 80 machines, with traffic from about 450 distinct IP addresses, were considered.

6.3 Setup

The experiments were run in an environment having a client-server architecture, with one dedicated machine acting as the central server (our admin console) and multiple remote clients connected to the admin machine via an ethernet based L2 (data link layer) network (see Figure 3). Each client machine was running Windows 7 on a 2.5 Ghz Intel-based dual core processors with at least 4 GB of RAM.

The server was kept in the same subnet as the rest of the client machines. This achieves two goals: It simulates a pure point-to-point client-server interaction and it lowers network latency. The central admin console periodically polls (every 5 minutes) the remote clients to look for new traffic data in their database(s), which are then filtered, moved in a compressed manner over the network and processed to be analyzed. Incidents are displayed to the system admin through the engine's graphical interface, shown in Figure 2.

At the start of every experiment, all client and server databases were erased. Next, approximately 6 to 10 hours worth of clean traffic data was introduced from the LANDER dataset, succeeded by injection of traffic data related to the test being conducted. The ratio of incident-to-clean data was kept between 10%-15%. Scalability tests had around 4 hours worth of clean data introduced along with 10% worth of malicious traffic. Our incident data also included "false" incidents. True incidents were real traffic representations of the case whereas false incidents were traffic data with patterns similar to either a) some other case b) random behavior resembling other incidents.

To handle a larger set of machines while having a workable network performance, the outgoing traffic from the client databases was compressed. It was observed that, for every poll made by the admin console for new data from the client machines approximately 3,000 to 4,500 traffic records were transferred per machine. Average size of each traffic record was i) for Uncompressed data (before transmission) about 16 Kb; ii) For Compressed data (during network transmission), about 10-12 Kb. Note that a large number of traffic records together comprise an incident. For example, the DDoS attack required at least 3000-4000 records from multiple machines to be identified as an incident. Torrent required a lower number, of around 500 records[2].

Except for the scalability tests, all the experiments were conducted under cold start conditions. This means that before performing an experiment, we did not perform any training, so as to test the worse possible setting for our system. The content of the KB varied according to the experiment type. As the training phase is responsible for weeding out all the regular/ safe traffic, not running the training phase leads to presence of several 100's or even 1000's of traffic logs which should otherwise have been filtered out. Note that running a scalability test in such a scenario leads to a potential distributed DoS attack on our admin machine.

Thus, to reduce the chances of clogging the engine, for the scalability test only we reduced the number of records being sent over from the remote clients, while still injecting 10-15% of incident data into the system. The rate of traffic flow was pre-determined by running the training phase of ReasONets for 40 clients over a time span of 7 days, followed by observation of the average number of traffic records being sent over a 6 hours time interval.

6.4 Results

Accuracy The first experiment involved varying the number of test (client) machines from a range of 7 to 15 while evaluating the system on the basis of its time performance, CPU utilization, false positive (FP) and precision values. We kept the network small for this experiment to better focus on the accuracy of specific cases. As we show in our scalability test, the results with larger networks are comparable. The FP ratio for a system is important in assessing the accuracy of the system, as a high FP ratio has the ability to drown out legitimate incident detection alerts. False negative FN are also relevant, but we note that our system has consistently less than 1% FN rates. Hence, we do not discuss it here.

The expected trend for change of values of precision and FP ratio for a learning based system is an increasing curve for precision and a decreasing curve for FP ratio. This is because, as the system gains more information, it should able to take more informed decisions. This trend is confirmed

[2]Generally, we observe a range of 0-25 or more traffic records for a Source and Destination IP pair within the time span of a second.

experimentally. Figure 4 reports the overall precision values recorded during this experiment. Recall that precision values express the ability of the system to correctly identify and tag an incident. This pattern is observed consistently in all of the cases. We notice that for DoS (Case 5), precision is consistently increasing, except for the transition from 9 to 11 machines, where the precision remains constant. This transition can be speculated as the minimum threshold of network configuration size for the critical/ stable functionality of the system. In other words, it is the minimum network size required for the system to gather sufficient data for a specific kind of incident.

Further, for all cases, we observed a false positive rate (not reported in the graphs) ranging from 6.4% to about 3.7%, with a notable improvement when the number of machines increases. Higher FP rates were always observed with smaller network configurations. For instance, for case 3 (DoS), we find the FP rates for configurations under 7 machines to be at 6.67% from where it linearly decreases to 5.13% for 11 machines succeeded by no visible change over the next size configuration of 13 machines, after which it promptly decreases to 3.67% over the final size configuration of 15 machines (corresponding to over 60 distinct IP addresses).

The decrease in FP ratio as the network size configuration increases follows the expected decreasing pattern except for transition from 11 to 13 machines. During this transition, the evaluations show no change in performance rate. Given the randomized distribution of incidents over its timeline across multiple machines and the interval between each poll by the server, we speculate that it is not un-realistic to see the system learning the same amount of information in both configurations (11 and 13 machines), hence leading to no change in output. Concerning the output for Botnet (case 4), here too we find a decreasing trend, with a non changing transition over 9 to 11 machines, before and after which the graph plots are always decreasing. The same trend is seen for the rest of the three cases, which lends more credibility to the explanation that the randomized distribution of incidents over the timeline leads to unpredictable jumps in increase in learning.

To further investigate whether the accuracy can be improved, we repeated the same tests on a network counting 20 hosts (78 IP addresses) and 20 cases (8 cases were variations of the original cases in Table 1), and varied uniformly weights for some of the features. We particularly focused on the aggregate feature *dml*, and the weights used for the "spatial distance" (*sd*) metric. Recall that spatial distance aims at measuring the distance between the domain observed during the incident and its closest well-known malicious domain. This metric was chosen for our analysis as it is the least intuitive to be interpreted as compared to the other ones forming *dml*. Variations on other metrics (e.g. black list, white list etc) also reported less notable results.

We study the changes in accuracy upon changing in the same way *dml* for all cases. This test was executed for 24 hours, and the machine's data included 200 true and actual incidents per case. 100 of these incidents were of DoS type, the remaining were randomly distributed across the other cases. We report the most interesting results in Table 2. In

Table 2: Weights influence on detection accuracy - weights are changed for for *sd* in *dml*

Setting	Case	FP	TP	Prec.
Baseline	DoS	6	94	94.059
	Other cases	5	94	94.949
Increasing $\overline{w}$	Dos	1.8	91	97.849
	Other cases	9	98	91.588
Decreasing $\overline{w}$	Dos	11	97	89.814
	Other cases	2.4	89	96.739

the table, we highlight the performance achieved with DoS detection, as compared to other cases. This is to highlight the distinct pattern observed in the DoS versus other cases. As reported, varying *dml* has a great influence on the overall accuracy and can bring our precision close to 98% and a false positive rate to 1.8% for DoS and of 96.7% precision (about 2% FP) for the other cases, respectively. When we increased weight to the *sd* metric used to compute *dml*, and in specifics we gave it a stronger weight in case of farther distances, a better accuracy was achieved for DoS and the performance suffered on the other cases. The reverse pattern was observed for the other cases. We speculate that, since during a DoS attack multiple source machine tend to attack a single destination machine, our study shows (roughly) that the machines tend to be spatially apart from each other so as to prevent early detection.

Stress Testing. In the second set of experiments we tested *how the system performs when an increasing number of incidents occur*. The number of incidents were varied from 5 to 500, with an increase of 10 incidents for every round of experiment. Each round was executed for over 16 hours. The results are reported in Figure 5. Except for the initial rounds where the system has apparently not gathered enough knowledge (5 to 50 incidents) and therefore results in higher FP, overall we consistently observed a decreasing FP ratio. The exact reverse is seen for precision metrics, which decreases initially, and after 50 incidents it gradually increases.

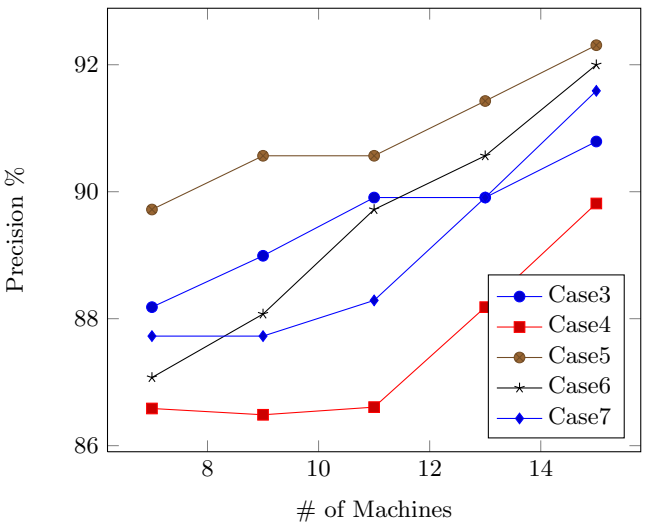

Figure 4: Case-wise Precision % Rates

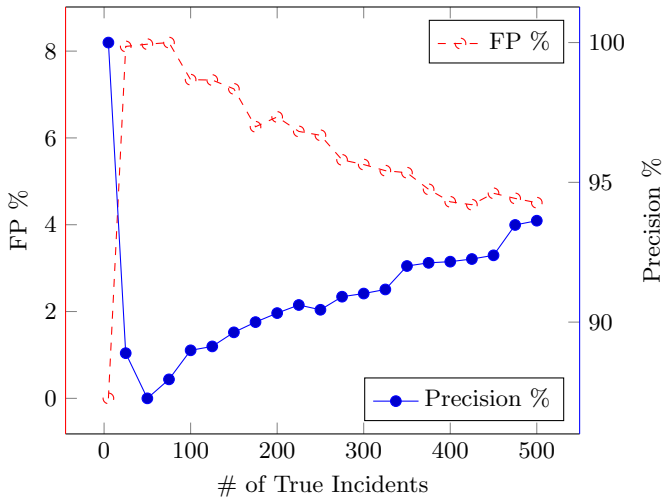

Figure 5: Stress Precision Rates

Table 3: Advantage of Fuzzy'ification (accuracy)

Settings	FP	TP	FN	TN	Prec.
Basic	16.66	61.5	5.2	16.66	78.163
ReasONets	4.36	64.93	2.6	26.66	93.55

Table 4: Scalability Test; Accuracy

#	FP	TP	FN	TN	Prec.
40	7.123	64.86	2	26	89.97
90	4.146	65.33	1.2	29.36	90.74

As part of our experiments on stress testing, we also tested the *performance of the reasoner with an increasing number of cases* in the system. The number of cases ranged from 7 to 90. The system's performance does not appear affected by the number of cases in the system. The traffic data included 320 incidents with around 200 true incidents over 45 client machines with worth 10 hours of clean traffic. Overall, our experiments show a false positive rate of average 3.9% (with a lowest performance at 5.3% for 50 cases), a false negative rate below 1% and an overall precision of 93.33%.

Comparison with baseline approach. The next test was to gauge the overall accuracy performance of ReasONets when compared to a generic traffic classification system. We calculated precision, FP, FN, TN and TP rates under two different scenarios; a) Under normal functionality of the system b) After disabling the fuzzy logic reasoning part of ReasONets and replacing with a simple detection logic, based on standard cluster analysis (through crisp K-nn). Table 3, showcases the decrease in FP and increase in precision under ReasONets. This is expected as ordinary data classification based systems cannot properly serve the numerous borderline cases which fall under more than one category, which is where our ranking algorithm based on fuzzy performs at its best.

Scalability. Scalability tests allow us to observe the ability of the system to incorporate an ever growing network size and measure the hit/drop in performance that comes with it. The CPU usage was found to be relatively high in previous tests due to the un-naturally high number of records (data) being transferred from the client machines, we reduced the number of records being transferred to a more realistic number, as described in Section 6.3. This allowed us to test the system in near real-world conditions. Results from these tests are reported in Tables 4 and 5. As explained earlier, under normal functionality of a learning based system, the system should show trends of lowering of FP ratio and increasing in precision. This is exactly what is seen across 40 (corresponding to about 185 distinct IP addresses) and 90 machines (corresponding to about 450 distinct IPs).

Table 5: Scalability Test; Performance Metrics (seconds)

#	Server Poll	Process time	Avg. CPU
40	400	109	71.3
90	800	124	82.7

7. CONCLUSION

Incident detection is an ongoing issue among corporations worldwide, as highlighted by recent industry and government efforts for classifying and addressing network threats [21, 20]. Though many approaches to incident and intrusion detection exist, they seem too strict to provide real-time situational awareness to system administrators.

To address this shortcoming, in this paper, we presented ReasONets, a novel approach for situational awareness of network incidents. ReasONets aims to leverage input from classic anomaly detection tools to learn about ongoing incidents and notable events within a closed network. Through extended experiments, we demonstrate the performance of the system against several system and network parameters. To date, this is the first system offering a concrete incident reasoner for increased awareness in controlled networks.

Our next step consists of integrating our case representation with the Structured Threat Information eXpression (STIX) format promoted by Mitre [21]. We will deploy the KB in an extended STIX format, and characterize each case by means of observables and STIX indicators. Further, we will refine our case processing for improved accuracy. We also plan to conduct tests on very large networks, of over 1,000 hosts. The ability of our system to scale seamlessly to over 400 IP addresses with no significant performance delay is a strong indicator of the capability of ReasONets. Nevertheless, a slight architectural re-design may be required, wherein the functionality of the engine may be replicated among collaborating systems managed by a set of network administrators.

8. ACKNOWLEDGEMENTS

Portions of this work from Squicciarini, Petracca and Nath was supported by a Hewlett-Packard Innovation Research Program Award: "Analytics for Situational Awareness", 2011. The work from Squicciarini was partially supported by National Science Foundation under award No. 1250319.

9. REFERENCES

[1] Snort, a lightweight network intrusion detection system. http://www.snort.org/.

[2] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans. Inf. Syst. Secur.*, 3(3):186–205, August 2000.

[3] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138. ACM, 2012.

[4] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Proceedings of the 21st Network and Distributed System Security Symposium (NDSS)*, 2011.

[5] BitTorrent. Official website for bittorrent. http://www.bittorrent.com.

[6] Eleazar Eskin. Anomaly detection over noisy data using learned probability distributions. 2000. http://academiccommons.columbia.edu/item/ac:125813.

[7] Mansour Esmaili, Bala Balachandran, Reihaneh Safavi-Naini, and Josef Pieprzyk. Case-based reasoning for intrusion detection. In *12th Annual Computer Security Applications Conference*, pages 214–223. 1996.

[8] Juan M. Estévez-Tapiador, Pedro Garcia-Teodoro, and Jesús E. Díaz-Verdejo. Measuring normality in http traffic for anomaly-based intrusion detection. *Computer Networks*, 45(2):175–193, 2004.

[9] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1–2):18–28, 2009.

[10] R. Guha, O. Kachirski, D. Schwartz, S. Stoecklin, and Y. Yilmaz. Case-based agents for packet-level intrusion detection in ad hoc networks. In *Proceedings of the 17th International Symposium on Computer and Information Sciences*, pages 315 – 320. CRC Press, October 2002.

[11] Paul Hick. The CAIDA DDoS Attack 2007 Dataset (collection). http://imdc.datcat.org/collection/1-06Y1-W=The+CAIDA+DDoS+Attack+2007+Dataset (accessed on August 2013).

[12] Xuan Dau Hoang, Jiankun Hu, and Peter Bertok. A multi-layer model for anomaly intrusion detection using program sequences of system calls. In *Proceedingi of the 11th IEEE IntÕl. Conference on Networks (ICON)*, 2003.

[13] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 01 1998.

[14] Eyke Hüllermeier, Didier Dubois, Henri Prade, De Toulouse, and Universit'e Paul Sabatier. Fuzzy rules in case-based reasoning. In *in Conf. AFIA99 Raisonnement à Partir de Cas*, pages 45–54, 1999.

[15] Christopher Kruegel and Thomas Toth. Using decision trees to improve signature-based intrusion detection. In *Proceedings of Recent Advances in Intrusion Detection (RAID)*, pages 173–191. Springer, 2003.

[16] David B. Leake. Case-based reasoning. *The Knowledge Engineering Review*, 9(01):61–64, 1994.

[17] Wenke Lee, S.J. Stolfo, and K.W. Mok. A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[18] Malware Domain List. http://www.malwaredomainlist.com.

[19] Microsoft. Windows based performance counter data logger. http://technet.microsoft.com/en-us/library/bb490960.aspx.

[20] Mitre. Common attack pattern enumeration and classification. http://capec.mitre.org/data/definitions/113.html.

[21] Mitre. Structured threat information expression. http://stix.mitre.org/.

[22] Soumyo D. Moitra. Situational awareness metrics from flow and other data sources. 2013.

[23] Official Website for uTorrent. http://www.utorrent.com.

[24] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 26–26, Berkeley, CA, USA, 2010. USENIX Association.

[25] Predict. Protected repository for the defense of infrastructure against cyberthreats. http://www.predict.org.

[26] J. Reason. Too little and too late: A commentary on accident and incident reporting systems. 1991.

[27] Timothy J. Ross. *Fuzzy Logic*, pages i–xxi. John Wiley & Sons, Ltd, 2010.

[28] Sans Education. https://isc.sans.edu/feeds/suspiciousdomains_high.txt.

[29] Shalla Secure Services KG. Shalla list website blacklist database. http://www.shallalist.de/Downloads/shallalist.tar.gz.

[30] Jessica Steinberger, Lisa Schehlmann, Sebastian Abt, and Harald Baier. Anomaly detection and mitigation at internet scale: A survey. In *Emerging Management Mechanisms for the Future Internet*, pages 49–60. Springer, 2013.

[31] Vimal Vaidya. Dynamic signature inspection-based network intrusion detection, August 21 2001. US Patent 6,279,113.

[32] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 133–145. IEEE, 1999.

[33] M. West-Brown, D. Stikvoort, K.-P. Kossakowski, G. Killcrece, R. Ruefle, and M. Zajicek. Handbook for computer security incident response teams (csirts), 2003. Technical Report CMU/SEI-2003-HB-002.

[34] Dit-Yan Yeung and Yuxin Ding. Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition*, 36(1):229 – 243, 2003.

[35] Zeus Tracker Domain Blocklist. https://secure.mayhemiclabs.com/malhosts/malhosts.txt.

Table 6: Application Metrics

Metric name	Description
# HTTP requests	Level of machine activity in the network
# GET requests	Number of GET requests for retrieving data from specific resources
# POST requests	Number of POST requests sent to an identified resource
# PUT requests	Number of requests for uploading a representation of a specified resource
# DELETE requests	Number of requests for deleting a specified resource
Avg. URLs length	Avg number of character in the URL
Avg. parameters in request	Avg number of parameters in the URL
Avg. data sent by POST requests	Avg size of POST requests received
Avg. responses length	Helps identify change in the traffic behavior, should remain constant over time
Avg. re-directions	Occurrences of the class of status code 3xx (further action needs to be taken by the user agent in order to fulfill the HTTP request)
Avg. client errors	Occurrences of class of status code 4xx (client seems to have erred)
Avg. server errors	Occurrences of class of status code 5xx (server failed to fulfill an apparently valid request)

Table 7: Network Metrics

Metric name	Description
Flow count	A flow is identified by: sourceIP, sourcePort, destinationIP, destinationPort, and Protocol
Avg. flow packet count	average number of packets in a flow
Avg. flow byte count	average number of bytes in a flow
Avg. packet size	Average of bytes per packet in a flow
Flow Behavior	Ratio of the flow count and the average packet size

[36] Loai Zomlot, Sathya Chandran Sundaramurthy, Kui Luo, Xinming Ou, and S. Raj Rajagopalan. Prioritizing intrusion analysis using dempster-shafer theory. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, AISec '11, pages 59–70, 2011.

APPENDIX

A. APPLICATION METRICS

In tables 6 and 7 we report examples of application and network metrics, respectively.

B. TRAFFIC GENERATION

Below is a brief description of the approach taken to generate network traffic for our case base. We provide an example of the DoS case.

- We inspected the data set available to us (i.e. Caida and Predict, as discussed in Section 6.1) to identify the smallest unit of traffic (minimal set of traffic records) which would constitute (be detected) an incident (say a DoS attack). This was achieved by running all the traffic records through ReasONets, so as to recognize the machine(s) and time it came under attack.
- After identifying the machine and time, we segregated all the traffic records reflecting traffic coming in or going out from the victim machine until the time of the attack.
- We then filtered records which were too far apart (timewise) to reduce the traffic records being considered and tested whether ReasONet would still recognize the reduced traffic record set as an attack. In this way, we obtained a *minimal traffic set identifiable as a DoS attack* which we would re-use later to create multiple attacks.
- To create the synthetic attacks, the script changed the IP address of the machine which was attacked to that of the client machine which we wanted to simulate an attack on. The script also changed the source of the attacks (the external IP addresses from which the attacks had originated) to random values from a lookup table maintained by us. It also updated the timestamp and attack distribution, to ensure it was not a perfect replica of the original attack.

  Example 1. *Assume the Src IP = 130.203.157.12 (the machine inside the network which got attacked in the Predict data set)*
  New Src IP = local client Machine's IP address
  Old Dest IP = 69.203.157.12
  New Dest IP = 145.10.1.1

- Every occurrence of the `old Dest IP` was changed to the above new value and the association stored in a table for convenience and consistency.