USING A DEEP UNDERSTANDING OF NETWORK ACTIVITIES FOR SECURITY EVENT MANAGEMENT

Mona Lange[1], Felix Kuhr[2] and Ralf Möller[1]
[1]Institute of Information Systems, Universitat zu Lubeck,
[2]Technical Universitat Hamburg-Harburg, Germany

ABSTRACT

With the growing deployment of host-based and network-based intrusion detection systems in increasingly large and complex communication networks, managing low-level alerts from these systems becomes critically important. Probes of multiple distributed firewalls (FWs), intrusion detection systems (IDSs) or intrusion prevention systems (IPSs) are collected throughout a monitored network such that large series of alerts (alert streams) need to be fused. An alert indicates an abnormal behavior, which could potentially be a sign for an ongoing cyber attack. Unfortunately, in a real data communication network, administrators cannot manage the large number of alerts occurring per second, in particular since most alerts are false positives. Hence, an emerging track of security research has focused on alert correlation to better identify true positive and false positive. To achieve this goal we introduce Mission Oriented Network Analysis (MONA). This method builds on data correlation to derive network dependencies and manage security events by linking incoming alerts to network dependencies.

KEYWORDS

Network Dependency Analysis, Security Event Management, Data Correlation

1. INTRODUCTION

The United States intelligence community has identified malicious actors exploiting cyberspace as a top national security threat [15].Similarly, Dell's annual threat report states a 100% increase in SCADA attacks [16].This report is based on analysis of data gathered by Dell's global response intelligence defense network that consists of millions of security sensors in more than 200 countries. In our daily lives, we depend on network services in many aspects (e.g., Internet banking, email, file sharing, medical services and smart homes).Also, enterprise networks consist of hundreds or even thousands of network services. Network services operate on distributed sets of clients and servers and rely on supporting network services, such as Kerberos, Domain Name System (DNS), and Active Directory. Hence, network services need to interact with each other in order to function correctly. Engineers use the divide-and-conquer approach and often to fulfill a task, multiple network services are required. This allows engineers to reuse network services and not have to re-implement complex customized ones.

Some network services are used in many other network services. We define them as supporting network services. A failure in a supporting network service leads to a failure in many other network services. Hence, IT administrators and IT managers are interested in knowledge about dependencies between network services. Recent efforts have lead to different approaches for analyzing network traffic to identify dependencies between network services. □

DOI: 10.5121/ijnsa.2016.8301

For network security it is necessary to understand a distributed system's perimeters. The criticality of a network event or alert can only be assessed, if it's implications on the monitored systems are understood.

Traditionally, the success of security information and event management (SIEM) is determined by the level of protection of critical network devices and applications from attackers. Critical network devices can be routers, servers, etc. and critical applications can compromise database management software or tools for monitoring and controlling an infrastructure. At the same time, it is understood that the ultimate goal of SIEMs is to protect ongoing and planned missions. Using current methods, it is virtually impossible to determine the impact of events on the attainment of mission objectives. Do we know which mission elements are affected? A monitored network is built with a higher-level purpose, a mission, in mind. So why not change our focus from trying to assess how an event reflects a potential attacker's behavior to trying to assess its impact on a monitored network's ongoing network activities?

1.1. Related Work

Several researches have concentrated on how to reduce the number of alerts reported by host-based and network intrusion detection systems, intrusion prevention systems or firewalls as well as decrease their false positive rate [17], [18], [19]. Similarity-based alert correlation approaches rely on features to compare the similarity of two alerts or the similarity of a single alert to a cluster of alerts. A similarity-based alert correlation approach is proposed by Cuppens et al. [20], and Peng et al. [21], who all cluster similar alerts to discover high-level attack scenarios. Others focus on reducing the problem of aggregating alerts into multi-step attacks to data mining problem [22], [23] or represent and reason with operators' preferences regarding the events and alerts they want analyze in priority [24].Another machine learning based approach focuses on correlating alerts according to the information in the raw alerts without using any predefined knowledge [23]. We use a deep understanding of network activities for security event management. To derive a deep understanding of network activities, we rely on network dependency analysis.

Often, different network services are required to perform a single task. If at least one service fails, the whole task could potentially fail. Sherlock [10] introduces an inference graph to represent dependencies between network services. The dependencies are calculated using co-occurrences between services. Co-occurrence exists, when two services are used in a defined window. Orion [11] is another approach to identify dependencies between network services using spike detection analysis in delay distributions of flow pairs. Two other network-based dependency analysis approaches are NSDMiner [12] and Rippler [13]. All mentioned approaches are based on network traffic. Rippler actively embeds communication delays in order to identify network dependencies. Passive approaches to identifying network dependency do not need additional software on hosts in the network and do not need any changes of applications or systems. They are analyzing communication between network services. Sherlock and Orion are presented as host-based approaches, but it is possible to implement them in a network-based manner without additional software.

2. NETWORK DEPENDENCY ANALYSIS

Network dependency analysis has the purpose of identifying complex dependencies between network service and components that may potentially be affected. The goal is to prevent unexpected consequences. Network dependency analysis requires analyzing network flows ina monitored infrastructure.

2.1.Network Flow

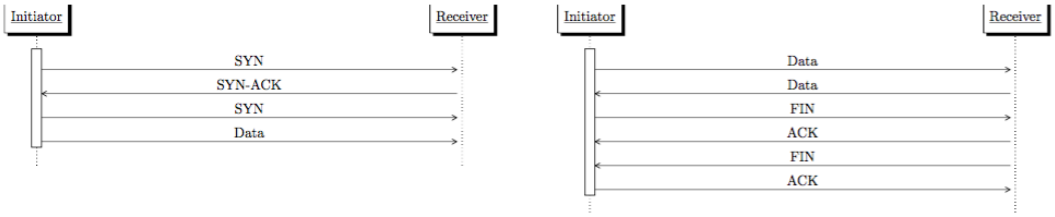

Figure 1. Before transmitting data from Initiator to Receiver a TCP connection has to be initialized by a 3-way handshake (left figure). After data is transmitted, a TCP teardown is performed (right figure).

We conduct a network dependency analysis based on packet headers (e.g. IP, UDP and TCP) and timing data in network traffic. Hence, our approach operates on network flows. A TCP flow starts with a 3-way handshake (SYN, SYN-ACK, ACK) between a client and a server and terminates with a 4-way handshake (FIN, ACK, FIN, ACK) or RST packet exchange. If network services communicate frequently, they may forgo the cost of repetitive TCP handshakes by using KEEPALIVE messages to maintain a connection in idle periods. Figure 1 represents the initialization of a TCP session and termination of initialized TCP session. We also use these messages to identify network flow boundaries. In comparison the notion of UDP flows is vague, since UDP is a state- less protocol. This is due to the protocol not having well-defined boundaries for the start and end of a conversation between server and client. In the context of this work, we consider a stream of consecutive UDP packets between server and client as a UDP flow, if the time difference between to consecutive packets is below a predefined threshold. In our analysis we exclude all network packet that are necessary for establishing a communication between server and client. So given that additional data is exchanged between network services, we term these end-to-end interactions between network services as direct dependencies.

The direct dependency between network services $s^j_i$and s^l_k is denoted as

SDEP = $\{(s^j_i, s^l_k) \mid s^j_i$sends a packet to s^l_k in the period under consideration$\}$ (1)

2.2. Network Packet

The basic building blocks of our approach are network packets exchanged between directly dependent network services. A network packet is exchanged by a source and destination IP address srcIP and dstIP via source and destination port srcPort and dstPort. In addition, a network packet relies on a specific transport layer protocol. In the context of this paper we distinguish the transport layer protocols TCP and UDP.

We define a network packet as a 6-tuple
P = (sIP, sPort, dIP, dP ort, ψ, t), (2)
for source IP addresses sIP , a source ports sPort, destination IP addresses dIP, destination ports dPort, a transport protocol $\Psi = \{\text{UDP, TCP}\}$ and timestamps t.

2.4. Network Services

Network services hosted by network devices are related to each other. HOSTS(d_j) returns a set of all network services hosed by the network device d_j. HOSTS^{-1}(s) returns the network device hosting network service s. It is possible to associate network service $s_i$with network device d_j using the notation s^{j}_{i}such that d_j =HOSTS^{-1}(s^{j}_{i})

2.5. Communication Histogram

Let us suppose that we are mirroring network traffic from an initial time point t_{min} to a time point t_{max} within an IT network. We are observing network packets $p \in P$, which is defined as a 6-tuple according to Equation 2. For communicating network services, we build communication histogram with a bin size Δ_t. In the context of work we set Δ_t to 1 second. The number of histogram bins is

bins = □ $(t_{max} - t_{min})$ □/ Δ_t, (3)

given that we want to build a communication histogram for network traffic mirrored from time point t_{min} to t_{max} with a bin size Δ_t.

Given that we are monitoring a set of S network services then the data structure for all communication histograms is defined

$H : S \times S \times \Psi \rightarrow (\{0, \cdots , bins - 1\} \rightarrow N_0)$, (4)

where the communication histogram bins $\{0, \cdots , bins - 1\}$ are mapped to N_0 . Now for every network packets exchanged between directly dependent network services, assuming it was received during the considered time period, the corresponding bin $H(s, s', \psi)$ in the communication histogram is incremented. The corresponding bin in the communication histogram is determined by $(t_{min} - t)$ mod bins, (5) assuming that the network packet p contains the time stamp t.

2.6. Indirect Dependency

Indirect dependencies are the second category of dependencies. Indirect dependencies are not easy to identify in network traffic, as they are not as obvious as direct dependencies. A direct dependency can be derived based on a network packet, which is exchanged by network services hosted by network devices. It is possible to estimate indirect dependencies by using SDEP+ in a brute force manner. This technique would overestimate indirect dependencies and does not lead to a deeper semantic understanding of complex dependencies in networks. We are interested in a model estimating indirect dependencies with a low rate of false positive. Therefore, we try to identify indirect dependencies between network services by identifying similar patterns in communication vectors of direct dependencies. There are two different types of indirect dependencies in networks: remote-remote (RR) dependencies and local-remote (LR) dependencies.

Example. An operator in a control center (CC) of an electrical power grid would like to send a request to a substation's RTU 37. First, the operator has to use human machine interface for medium voltage substation (HMI) to specify the request and then forward it to a front end server (FES), so that the FES sends the request to the remote terminal unit (RTU). Assuming HMI and FES are different network devices, and the network device hosing the HMI has the name of its FES and not the IP address. Then it is necessary that first a request is sent to the network device hosting domain name system to get the IP address of FES. After the network device has information about the IP address of FES it can send the operator's request to front end server. So there is an LR indirect dependency between network services hosted by HMI, the DNS server and network service hosted by FES. A graphical representation is given in Figure 2.

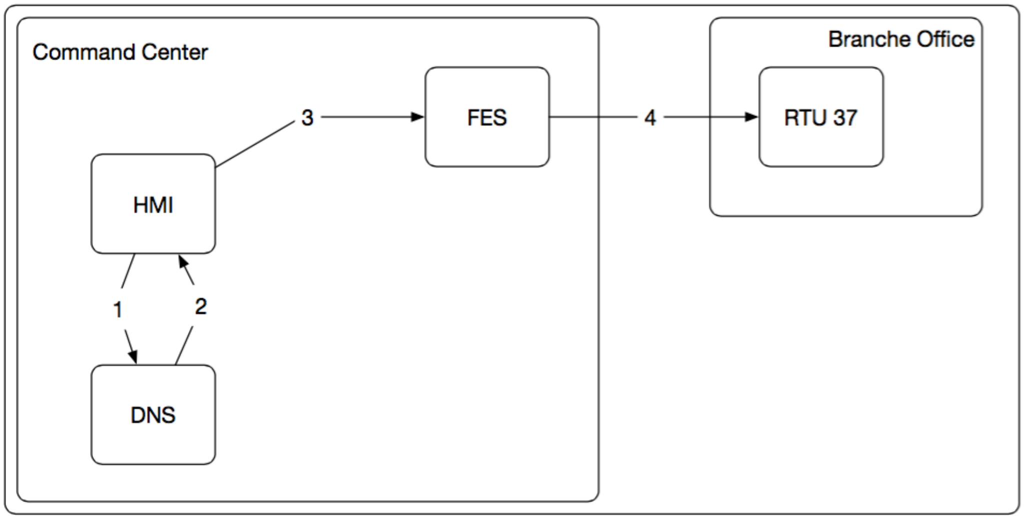

Figure 2.Graphical representation of Example 1: Communication between HMI, DNS, FES and RTU 37.

Assuming there exist an indirect dependency between network service s^j_i, s^l_k and s^o_n. Then, there exist two direct dependencies: $\delta(s^j_i, s^l_k)$ and $\delta(s^l_m, s^o_n)$. In Example 1, this is the indirect dependency between HMI, front-end server and remote terminal unit. Both direct dependencies consist of a communication vector N^m for $\delta(s^j_i,s^l_k)$ and $\delta(s^l_m,s^o_n)$. Because of their indirect dependency, both communication vectors have a similar pattern. Due to a processing delay on network device d_l, the communication vectors are shifted by a processing delay τ_{delay}. Processing delay τ_{delay} is the delay between receiving a data packet and sending a new data packet to another network service. Two direct dependencies define an indirect dependency, if the network services fulfill the following definition.

$$ISDEP_{RR} = SDEP \bowtie_{HOST\ S^{-1}(2)=HOST\ S^{-1}(1)} SDEP \qquad (6)$$

If network service s^j_isends data packets to network service s^l_k so that the data is processed on network device d_l and after processing delay a network service s^l_m sends data packets to network service s^o_n, we have two direct dependencies. These direct dependencies encompass four network services and represent one RR indirect dependency.

LR indirect dependencies between network services are given, if a network service requires a supporting network service to perform a task. Therefore, we are interested in identifying local-remote indirect dependencies, too. In Example 1, an LR indirect dependency exists because a network service of the network device hosting HMI requires DNS of another network device in order to send its request to the FES.

$$ISDEP_{LR} = SDEP \bowtie_{HOST\ S^{-1}(1)=HOST\ S^{-1}(3)} SDEP \qquad (7)$$

Using our definitions, there might be many potential indirect dependencies. If a network has N network devices, theoretically the total number of potential direct dependencies between network services is $N^2 \cdot p^2$, where p is the number of maximum concurrently operating services per network device. On most operating systems this value is 65536. Therefore, the number of potential indirect dependencies between network services could increase polynomial in networks. Networks containing at least 125 network devices have potentially more than one billion direct dependencies between network services. So, we need a technique to reduce the number of indirect dependencies. The idea is to consider only potential indirect dependencies with a high probability of being a correct indirect dependency. In our approach, we use normalized cross correlation to calculate a *similarity-value* between two communication vector of direct dependencies$\delta(s^j_i,s^l_k)$ and $\delta(s^l_m,s^o_n)$. Both communication vectors have to have the same length k to calculate the normalized cross correlation.

In Equation (8) we present the normalized cross correlation for communication vectors $\delta(s^j_i,s^l_k)$ and$\delta(s^l_m,s^o_n)$. Normalized cross correlationρ is derived by

$$\rho(\delta(s^j_i, s^l_k), \delta(s^l_m, s^o_n))[\tau] = \frac{1}{k}\sum_{n=0}^{k} \frac{\left(\delta(s^j_i,s^l_k)[n]-\overline{\delta(s^j_i,s^l_k)}\right)\cdot\left(\delta(s^l_m,s^o_n)[n+\tau]-\overline{\delta(s^l_m,s^o_n)}\right)}{\sigma_{\delta(s^j_i,s^l_k)},\sigma_{\delta(s^l_m,s^o_n)}},$$

(8)where the mean values of the compared communication vectors are represented by $\overline{\delta(s^l_m, s^o_n)}$ and $\overline{\delta(s^j_i, s^l_k)}$, respectively. $\sigma_{\delta(s^l_m,s^o_n)}$ and $\sigma_{\delta(s^j_i,s^l_k)}$ are the standard deviations of both communication vectors.

In image processing, normalized cross correlation is used to correlate an image with a template. Consider for example, you have a small picture of an orange and multiple larger images of fruit bowls. Cross correlation helps you identify, if a fruit bowl contains an orange, by matching the template to the fruit bowl image. Sometimes the brightness might differ due to illumination diverging between the larger image and the template. Normalized cross correlation reduces the influence of differing brightness within image and template.

Indirect dependencies between two network services have a similar pattern shift in time. Furthermore, the number of exchanged packets does not necessarily have to be identical two communication histograms, although they are indirectly dependent. This effect is similar to the effect of brightness in image processing. In our approach, we use normalized cross correlation to consider the situation that two communication vectors diverge in the number of data packets although the communication vectors represent an indirect dependency. Communication vectors between indirect dependencies are not aligned due to communication and processing delays. Communication delays are caused by network latency and processing delays are due to information being processed, before being passed on.

To obtain high rates in true positive (TP) and low rates in FP and FN indirect dependency estimation, communication vectors have to be shifted against each other.

$$\tau_{delay} = argmax_{\tau}\rho(\delta(s^j_i, s^l_k), \delta(s^l_m, s^o_n))[\tau] \tag{9}$$

Therefore, τ_{delay} is the period a delay or failure in network service $s_i$or s_k influences network service s_n on device d_o. To reduce the potential indirect dependencies generated by 3.1.2, we introduce a threshold and consider potential indirect dependency $\iota_{RR}(\delta(s^{j}_{i},s^{l}_{k}), \delta(s^{l}_{m},s^{o}_{n}))$ as a correct identified indirect dependency if $\rho(\delta(s_i^j, s_k^l), \delta(s_m^l, s_n^o))[\tau] > \theta$

Knowing network dependencies in a monitored network provides a basis for verifying how alerts could potentially affect the overall network. For this purposes a monitored infrastructure's alerts need to be processed and linked to network dependencies.

3. Alert Processing

Probes of multiple distributed firewalls (FWs), intrusion detection systems (IDSs) or intrusion prevention systems (IPSs) are collected throughout a monitored network such that large series of alerts (alert streams) need to be fused. As different alert streams could use different data formats, alerts need to be normalized before being further processed. In order to implement fused alert processing in this fashion, the goal of low-level correlation is to provide for alert normalization, enrichment, verification, and merging.

3.1. Alert Normalization

In the overall LLC architecture, data from the above-mentioned probes are fed into a SYSLOG engine [1] for real-time data processing. Due to the heterogeneous nature of data formats provided by probes in a distributed environment, it is necessary implement a normalization process to allow for coherent handling of alerts in subsequent processing steps. Alert normalization for the a electrical power grid's communication network test is realized by a SYSLOG configuration as well as the set of descriptive attributes of normalized alerts is inspired by the Intrusion Detection Message Exchange Format (IDMEF). For details on IDMEF see [2]. Alerts are enriched when values for required attributes are not provided by original alert sources such that sensible default values are inserted. After normalization and enrichment, LLC proceeds with alert verification. In the following we describe details on verification, using introductory examples with normalized alerts.

3.2. Alert Verification

Alert verification is subdivided into three parts, namely alert vulnerability verification, alert severity verification, and alert operational impact verification. We first discuss vulnerability verification, with the central idea to relate alerts with information about known vulnerabilities for the network under consideration. Vulnerabilities are available from the so-called network inventory, listing all monitored network devices and detected vulnerabilities. To detect vulnerabilities nmap and openvas periodically scan the monitored network. The low-level correlation process relies on this knowledge base to link normalized alerts to the monitored network. Normalized alerts can have many fields and in the following we focus on special cases in order to demonstrate central ideas using several examples.

3.2.1. Alert Vulnerability Verification

Consider, as shown in Table 1, an alert with a reference to a local exploit reported by an IDS with the analyzer ID CEDET01IDS. Local exploits take advantage of vulnerabilities or bugs. For the example in Table 1 we assume that an IDS (CEDET01IDS) has generated an alert referring to CVE vulnerability (for more details regarding CVE see [3]).

Table 1. An alert issued by an IDS probe with the analyzer ID CEDET01IDS reports a local exploit.

Analyzer ID	**Alert ID**	**Time**	**Source**	**Destination**	**CVE ID**
CEDET01IDS	1	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	CVE-2016-0034

An exploit can only be successful if the respective vulnerability or bug is indeed present on the targeted network device. Thus, alert vulnerability verification needs to check whether the vulnerability has been detected on the targeted device by extracting respective information from the network inventory. To continue the example, we assume that data shown in Table 2 are available, indicating that CVE-2016-00034 is indeed associated with network device 132.8.1.5.

Table 2. Vulnerability information extracted from the network inventory.

Network Device	**CVE ID**
132.8.1.5	CVE-2016-0034

The network inventory allows the alert vulnerability verification process to link a reported local exploit, as shown in Table 1, to vulnerabilities detected on source or destination hosts, as shown in Table 2.

Table 3 – A verified vulnerability alert with an added tag VULNVERIFIED.

Analyzer ID	**Alert ID**	**Time**	**Source**	**Destination**	**CVE ID**	**Classification ID**
CEDET01IDS	1	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	CVE-2016-0034	VULNVERIFIED

If an alert can be linked to a previously recorded vulnerability, we refer to it as a true positive attack; otherwise it is denoted as a false negative attack. The alert vulnerability verification process filters out false negative attacks. True positive attacks are passed on with the added classification ID VULNVERIFIED. Table 3 shows a true positive attack alert with the added classification ID VULNVERIFIED. In order to reduce the workload of subsequent models, in the LLC configuration, there is an option to suppress alerts for which no vulnerability can be identified.FW/IDS/IPS probes report on abnormal behavior occurring in a monitored network, and not only local exploits potentially with a high impact are being detected. The level of severity that probes assign to a reported alert is investigated by alert severity verification.

3.2.2. Alert Severity Verification

Given a sensor is certain of abnormal activity with an impact occurring, an alert with a severity data field is issued. Should we be unable to verify an alert's vulnerability, but there is an impact associated with the event then we pass on the alert with an added classification ID UNVERIFIED. An alert's vulnerability can be unverified due to two reasons: An alert reporting an exploit using a particular CVE ID, which the network inventory cannot be corroborated or the network inventory detected a vulnerability on source or destination, which is not mentioned in the alert. Consider, as shown in Table 4, for example an alert with a CVE ID "CVE-2016-0033" severity "medium" is reported by an IDS with the analyzer ID CEDET01IDS.

Table 4 – An alert issued by an IDS probe with the analyzer ID CEDET01IDS reports an abnormal activity with a severity and CVE ID.

Analyzer ID	Alert ID	Time	Source	Destination	CVE ID	Severity
CEDET01IDS	2	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	CVE-2016-0033	medium

CVE-2016-033 is not a vulnerability that was detected on source or destination. Hence, the alert does not have verified vulnerability. However, given that an alert contains a severity, this alert is passed on for further consideration with a tag "NOTVERIFIED" as shown in Table 5.

Table 5 – A verified severity alert with an added tag NOTVERIFIED.

Analyzer ID	Alert ID	Time	Source	Destination	Severity	Classification ID
CEDET01IDS	2	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	medium	NOTVERIFIED

In comparison consider an alert reports a medium severity incident with no associated vulnerability as described in Table 6.

Table 6 –An alert issued by an IDS probe with the analyzer ID CEDET01IDS reports an abnormal activity with a severity.

Analyzer ID	Alert ID	Time	Destination	Source	Severity
CEDET01IDS	2	2016-01-24 11:02:31.20	132.8.1.5	85.1.1.8	medium

The network inventory lists vulnerability “CVE-2016-0034” on the destination network device.

Table 7 - Vulnerability information extracted from the network inventory.

CVE ID	Network Device
CVE-2016-0034	132.8.1.5

Although the vulnerability could not be verified, the medium severity incident is passed on by the severity verification process with an added classification ID “NOTVERIFIED”.

Table 8 - A verified severity alert with an added tag NOTVERIFIED.

Analyzer ID	Alert ID	Time	Source	Destination	Severity	Classification ID
CEDET01IDS	2	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	medium	NOTVERIFIED

3.2.4. Alert Fusion

FW/IDS/IPS alerts are reports of abnormal activities in a monitored network. As indicated above, some alerts are clearly linked to an exploited vulnerability. However, the vast majority of alerts are simply reports of abnormal activities in a network. Also, in 2015 the security company Check Point reported that zero day attacks are rising [15]. Check Point discovered that organizations are targeted by about 100 unknown malware attacks per hour. Thus, reporting a rate 50 times more than the rate in 2013. As the vulnerabilities involved in, e.g., zero day attacks are unknown, the attack paths up as false negatives in vulnerability based high-level online correlation. Misconfiguration or insider attacks are another cause for false negatives. We attempt to address these attacks by performing alert fusion. Generally, these more sophisticated attacks require prior reconnaissance attacks, which cause FW/IDS/IPS alerts. Obviously, FW/IDS/IPS sensors involved in the detection of abnormal activities do not report these alerts with a high confidence of belonging to an ongoing attack. Combining multiple instances of abnormal activity reports augment the confidence that a reconnaissance attack is currently occurring. As these alerts would otherwise go unnoticed, alert fusion aims to reduce false negative alerts.

The purpose of alert fusion is to combine alerts that represent the same attack occurrence. The LLC process maintains a tumbling alert based window of n alerts. The size of the alert processing window n is computed as the averaged daily number of alerts occurring every second. On one hand, this allows for fixed computation time within the low level correlation process (regardless of how long the component is running), on the other hand, it must be taken into consideration that alerts can only be fused if they are in the same window. However, the parameter n can be selected in MONA's configuration such that scalability and expressivity requirements can be fulfilled in a particular context.

Duplicate Alert Fusion

The goal of alert fusion is to combine alerts representing an identical detection of the same attack by FW/IDS/IPS sensor. A single FW/IDS/IPS sensor is able to produce duplicate alerts, when an attack fits multiple rules. This phenomenon is also referred to as alert splitting. Table 9 describes how an IDS sensor with an analyser ID CEDET01IDS splits a port scan into two alerts. The LLC process performs duplicate alert fusion by merging them into a meta-alert. The meta-alert averages the timestamp of both alerts and combines all diverging data fields.

Table 9 – Example for merging duplicate alerts.

Analyzer ID	Alert ID	Time	Source	Destination	Description	Classification ID
CEDET01IDS	1	2016-01-24 11:02:31.10	85.1.1.8	132.8.1.5	Port scan	
CEDET01IDS	2	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	Port scan	
CEDET01IDS	{1,2}	2016-01-24 11:02:31.15	85.1.1.8	132.8.1.5		DUBLIMERGE

Taking the same example of port scanning, it is possible that distinct FW/IDS/IPS sensors detect the port scan. Given source and destination with an alert are identical, the low-level correlation processes performs alert fusion. Assume that two distinct IDS sensors with an analyzer id CEDET01IDS and CEDET02IDS both report a port scan. The event correlation process performs duplicate alert fusion by merging them into a meta-alert. The meta-alert averages the timestamp of both alerts and combines all diverging data fields.

In Table 10 there are two port scans shown from one source (1 to n). With an appropriate classification ID a new alert that fuses both port scan alert is generated (see the last row in Table 10).

Table 10 –Example for merging duplicate alerts with the same source address.

Analyzer ID	Alert ID	Time	Source	Destination	Description	Classification ID
CEDET01 IDS	4	2016-01-24 11:02:31.10	85.1.1.8	132.8.1.4	Port scan	
CEDET01 IDS	5	2016-01-24 11:02:31.20	85.1.1.8	132.8.1.5	Port scan	
CEDET01 IDS	{4,5}	2016-01-24 11:02:31.15	85.1.1.8	{132.8.1.4,132.8.1.5}		SRCMERGE

In Table 11 we show a port scans from different source to a single device. The fused alert shown in the last row, again fusion is indicated using a specific classification ID.

Table 11 –Example for merging duplicate alerts with the same destination address.

Analyzer ID	Alert ID	Time	Source	Destination	Description	Classification ID
CEDET01IDS	6	2016-01-24 11:02:31.10	85.1.1.8	132.8.1.4	Port scan	
CEDET01IDS	7	2016-01-24 11:02:31.20	85.1.1.9	132.8.1.4	Port scan	
CEDET01IDS	{6,7}	2016-01-24 11:02:31.15	85.1.1.10	132.8.1.4		DSTMERGE

It is possible to configure LLC in such a way that original alerts are filtered and only the meta-alerts or fused alerts are forwarded.

3.2.4. Alert Operational Impact Verification

Traditionally, the success of security information and event management (SIEM) is determined by the level of protection of critical network devices and applications from attackers. Critical network devices can be routers, servers, etc. and critical applications can compromise database management software or tools for monitoring and controlling an infrastructure. In order to understand how critical an application is for a network's mission, a mission criticality level is added to the network inventory. Per default, application ports that where found responding are listed in the network inventory with a default mission criticality value low. Network operators are able to reclassify an applications mission criticality to medium or high. Table 12 illustrates an entry from the network inventory.

Table 12 –An application's mission criticality is added to the network inventory.

CVE ID	**Network Device**	**Application Port**	**Mission Criticality**
CVE-2016-0034	132.8.1.5	80	High

$CRITIC$: $s_i \rightarrow \{low, medium, high\}$ (10)

Hence, the set of affected indirect dependencies *OPIMACT* is defined as

$OPIMPACT = SCC((\forall s_i \in S : CRITIC(s_i), map(asSet, ISDEP)))$, (11)

where SCC denotes the strongly connected components (SCC) of the hypergraph given as parameter (asSet maps a tuple into a set of components). Figure 3 gives an example for a set of affected indirect dependencies.

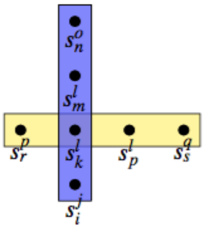

Figure 3. An example for indirect dependencies affected by an alert.

Operational impact lists all network services and thereby devices, which are potentially affected by a security incident reported in form of an alert. Network operators can predetermine what level of threat to a mission criticality is sufficient for a security incident to be reported.

4. Evaluation

A test bed based on a disaster recovery site of an energy distribution network, provided by an Italian water and energy distribution company, was available for testing. To evaluate MONA, metasploit [8] was used to emulate an attack on the test bed. To allow a more extended evaluation, a synthetic data set generator was built based on the dataset from the attacks on the test bed. This synthetic data set generator simply replays these attack patterns, which have been collected in the test bed, and is additionally able to provide more alerts per second for a scalability analysis. Additionally, we are able to replay typical network traffic patterns of dependent network services.

4.1. Test environment

The environment used for the evaluation is following:

- Operating System: Ubuntu Release 12.04 (precise) 64-bit, Kernel Linux 3.13.0- 32-generic
- RAM: 12 GB, 1600 MHz DDR3·
- 2 CPUs: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz

4.2. Precision and Recall

True Positive (TP) represents the correctly identified indirect dependencies while False Negative (FN) is the rate of correct indirect dependencies not identified. False Positive (FP) rate contains identified indirect dependencies, which are no indirect dependencies. This leads to calculation of precision and recall

Precision = TP / (TP + FP)
Recall = FP / (TP + FN) (10)

4.3. Sensitivity Analysis

NSDMiner, Sherlock, and Orion conduct their experiments based on network traces and present attractive TP and FN rates. Administrators and employees on their own networks provide the respective ground truth for NSDMiner, Sherlock and Orion. As enterprise networks rely on third parties to provide the respective ground truth, administrators cannot be certain that they are aware of all network dependencies. In fact, in our experiments, often network dependency analysis was very often able to point out unknown network dependencies to administrators. For a trustworthy evaluation, a known ground truth is essential. Hence, we conducted our experiments based on the synthetic data set generator. Our ground-truth is not dependent on knowledge from administrators or other humans. For each indirect dependency between network services, our network generator saves important details into a file. Hence, indirect and direct dependencies, size of network and simulation duration of each generated network is known.

Figure 4 shows an evaluation of two networks. Delay distribution between network services of indirect dependencies is randomly distributed between one and three seconds. The left figure shows the average rate for networks with 100 network devices, 20 indirect dependencies and 60 additional communicating network devices. The right figure shows the same experiment with a bigger network containing 250 network devices, 140 indirect dependencies and 120 additional communicating network devices. The x-axis represents the multiplication factor used to calculate a threshold in Orion's approach. Orion makes use of mean+x·stdev to calculate the threshold to classify potential indirect dependencies.

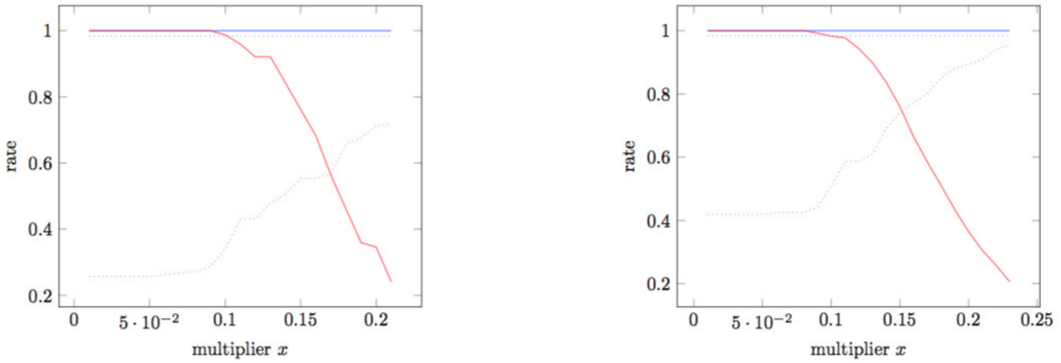

Figure 4.Comparison between MONA and Orion [Precision NDA (blue line), Recall MONA (blue dotted line), Precision Orion (red line), Recall Orion (red dotted line)]. Left Figure represents average rates for networks containing 100 network-devices. Right Figure represents average rates for networks containing 450 network-devices.

For an additional experiment, we expanded the experiment to include NSDMiner and Sherlock in addition to Orion. We compare their precision and recall rate to MONA's. Figure 6 shows the precision and recall rate of Sherlock, Orion and MONA on varying sized networks.

4.4. Comparison with Orion

Orion focuses on local-remote indirect dependencies and is not able to identify remote-remote indirect dependencies. A reason for this behavior is their host-based design. Our network-based approach MONA is able to identify more complex remote-remote indirect dependencies. To compare both approaches, our evaluation is based on local-remote indirect dependencies. In a first step we generate networks containing traffic between network services using our network generator. This leads to ground-truth and networks with different characteristics. We use our implementation of Orion and MONA to calculate indirect dependencies identified in the generated networks. Orion has a high rate of TP and a low FN rate of indirect dependencies in networks with small delays and huge amount of network communication between indirect dependent network services. Few networks fulfill these criteria. Data networks of power grids are one example not fulfilling the mentioned criteria. In enterprise networks, often devices are geographically distributed in different quarters. Therefore, delays are more spread than in networks without geographically distributed areas. Each connection to branch offices has its own characteristics. Furthermore, Orion's threshold is dependent on number of network devices and amount of network traffic involved in indirect dependencies.

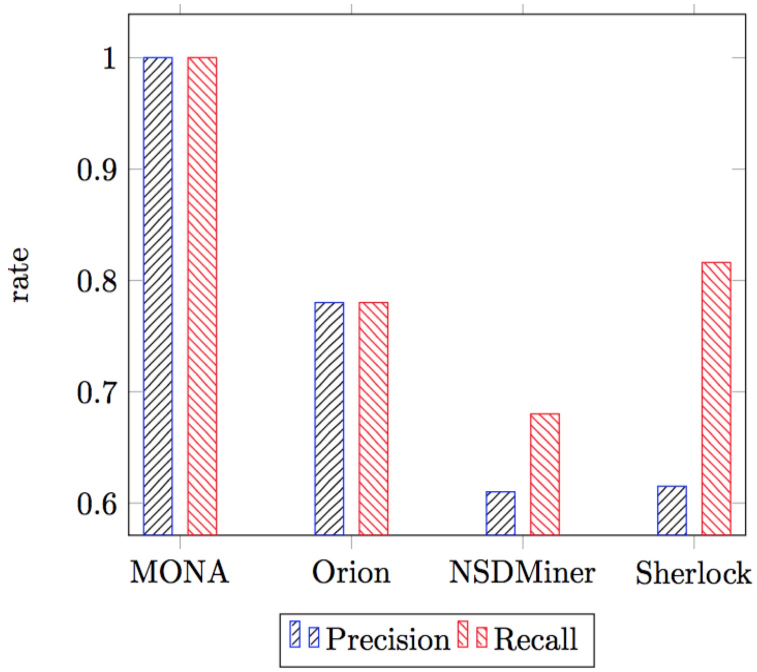

Figure 5. Comparison between NDA, Orion, NSDMiner and Sherlock.

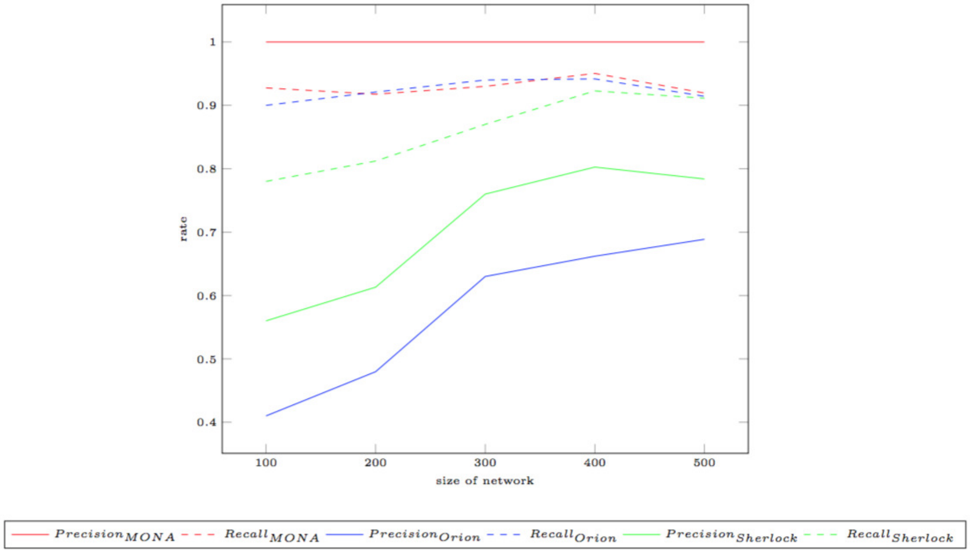

Figure 6. Precision and recall between MONA, Orion and Sherlock for networks with different amount of devices.

4.5. Performance alert processing time

As a number of low-level alerts are being fired from IDS sensors at a high pace, a fairly sophisticated knowledge of both networking technology and infiltration techniques is required to understand them. IDS alert correlation system tries to solve this problem by post-processing the alert stream from one or many IDS sensors. They goal is to aggregate low-level alert and enrich the alerts with vulnerability information. Assuming a time-based system, the processing time in seconds depends on the number of simultaneously processed alerts. If the alert processing time exceeds one second, we would not be able to hold that pace, should it continuously occur over multiple time windows. The results are illustrated in Figure 2. The computation time of simultaneously processed alerts is compared with 2 virtual CPUs and 4 virtual CPUs. This experiment shows that MONA's algorithm is parallelizable.

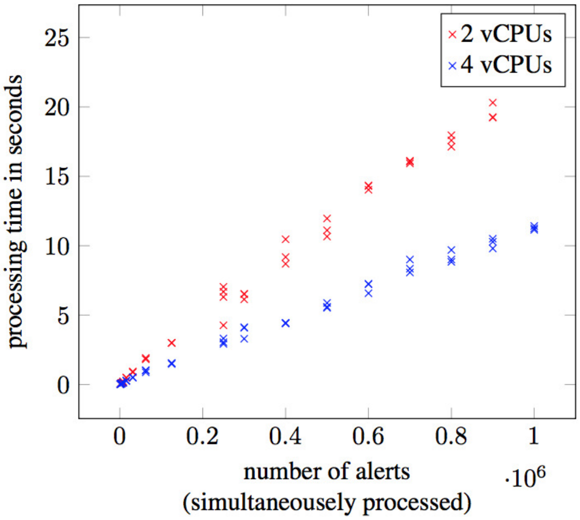

Figure 7.Alert processing time.

4.6. Performance network dependency analysis

The network dependency analysis is conducted offline. Figure 8 evaluates the performance of network dependency in a network containing 100 devices and 50 communicating network services. Computing network dependencies requires generating potential indirect dependencies and analyzing whether these indirect dependencies actually exist. To assess, whether more network devices affect the performance, we conducted the same experiment with a network containing 900 network devices. The results of this evaluation are shown in Figure 9, revealing the number of network devices does not affect the performance of network dependency analysis.

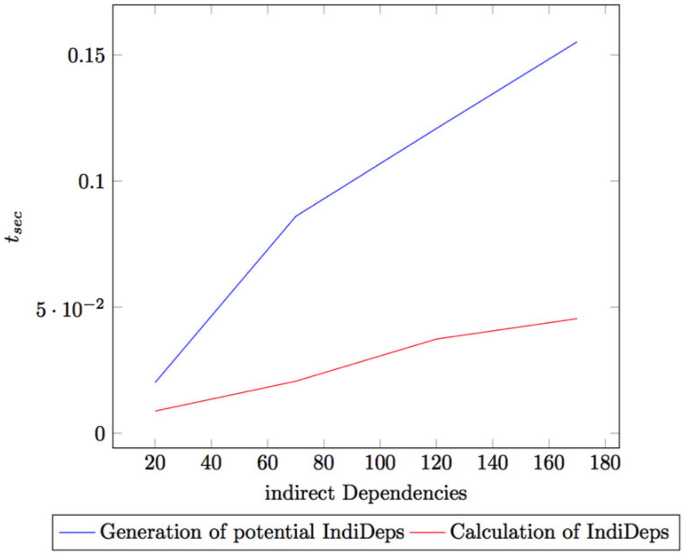

Figure 8. Network dependency analysis in a network containing 100 network devices.

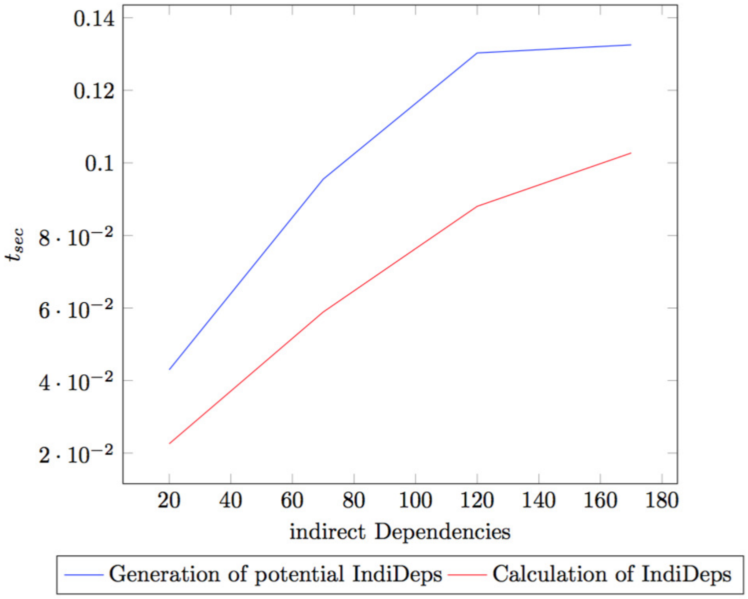

Figure 9. Network dependency analysis in a network containing 900 network devices.

5. CONCLUSIONS

We have demonstrated a novel approach to security event management based on a deeper understanding of network activities. This deeper understanding was derived based on network dependency analysis. Network dependency analysis enables deriving a deeper understanding of network activities and leverages well data-communication networks with different numbers of network devices and indirect dependencies. A heuristic evaluation based on a synthetic data generator compares the precision of our proposed approach to other network dependency discover methodologies. In addition to aggregating alerts with similar characteristics occurring in the same time window, we extend network dependencies in order to derive the operational impact of alerts. The proposed frame is able to link network dependencies with security events and thereby assesses a security event's impact on ongoing network activities.

ACKNOWLEDGMENT

The research in this paper has received funding from the PANOPTESEC project, as part of the Seventh Framework Programme (FP7) of the European Commission (GA 610416).

REFERENCES

[1] Budai, László, https://www.balabit.com/network-security/syslog-ng, last accessed on 21/03/2016.

[2] Debar, Hervé, David A. Curry, and Benjamin S. Feinstein, "The intrusion detection message exchange format (IDMEF)." (2007).

[3] Common Vulnerabilities and Exposures, https://www.cve.mitre.org, last accessed on 26/02/2016.

[4] Check Point Security Report, https://www.checkpoint.com/resources/2015securityreport/CheckPoint-2015-SecurityReport.pdf, 2015.

[5] Mona Lange, Ralf Moeller, Gregor Lang and Felix Kuhr, Event Prioritization and Correlation based on Pattern Mining Techniques, ICMLA, 2015.

[6] Technical Report, Mona Lange, Felix Kuhr, Ralf Moeller, Using a Deep Understanding of Network Activities for Network Vulnerability Assessment, 2016.

[7] Metasploit, https://www.metasploit.com, last accessed on 26/02/2016.

[8] Michael Larsen and Fernando Gont. Recommendations for transport-protocol port randomization. 2011.

[9] ParamvirBahl, Paul Barham, Richard Black, Ranveer Chandra, MoisesGoldszmidt, Rebecca Isaacs, SrikanthKandula, Lun Li, John MacCormick, David A Maltz, et al. Discovering dependencies for network management. In ACM SIGCOMM 5th Workshop on Hot Topics in Networks (Hotnets-V), pages 97–102. ACM, 2006.

[10] Xu Chen, Ming Zhang, Zhuoqing Morley Mao, and ParamvirBahl. Automating net- work application dependency discovery: Experiences, limitations, and new solutions. In OSDI, volume 8, pages 117–130, 2008.

[11] ArunNatarajan, Peng Ning, Yao Liu, SushilJajodia, and Steve E Hutchinson. NS- DMiner: Automated discovery of network service dependencies. IEEE, 2012.

[12] Ali Zand, Giovanni Vigna, Richard Kemmerer, and Christopher Kruegel. Rippler: Delay injection for service dependency detection. In INFOCOM, 2014 Proceedings IEEE, pages 2157–2165. IEEE, 2014. 2.2, 4.2.4

[13] Check Point Security Report, https://www.checkpoint.com/resources/ 2015securityreport/CheckPoint-2015-SecurityReport.pdf, 2015, last accessed on 26/02/2016.

[14] J. R. Clapper. Statement for the record, worldwide threat assessment of the US intelligence community.https://www.dni.gov/index.php/newsroom/testimonies/209- congressional-testimonies-2015/1174-statement-for-the- record-worldwide-threat-assessment-of-the-u-s-ic- before-the-sasc, 2014.

[15] D. Inc. Dell security annual threat report.https://software.dell.com/docs/2015-dell-security- annual-threat-report-white-paper-15657.pdf, 2015.

[16] B. Morin, L. M´e, H. Debar, and M. Ducass´e. M2D2: A formal data model for IDS alert correlation. In Recent Advances in Intrusion Detection, pages 115– 137. Springer, 2002.

[17] T. Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In Recent Advances in Intrusion Detection, pages 102– 124. Springer, 2004.

[18] R. Smith, N. Japkowicz, M. Dondo, and P. Mason. Using unsupervised learning for network alert correlation. In Advances in Artificial Intelligence, pages 308–319. Springer, 2008.

[19] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on, pages 202–215. IEEE, 2002.

[20] X. Peng, Y. Zhang, S. Xiao, Z. Wu, J. Cui, L. Chen, and D. Xiao. An alert correlation method based on improved cluster algorithm. In Computational Intel- ligence and Industrial Application, 2008.PACIIA'08. Pacific-Asia Workshop on, volume 1, pages 342–347. IEEE, 2008.

[21] H. Farhadi, M. AmirHaeri, and M. Khansari. Alert correlation and prediction using data mining and HMM.The ISC International Journal of Information Security, 3(2), 2015.

[22] M. GhasemiGol and A. Ghaemi-Bafghi. E-correlator: an entropy-based alert correlation system. Security and Communication Networks, 8(5):822–836, 2015.

[23] K. Tabia, S. Benferhat, P. Leray, and L. M´e. Alert correlation in intrusion detection: Combining AI-based approaches for exploiting security operators' knowledge and preferences. In Security and Artificial Intelligence (SecArt), page NC, 2011.