ARGUS

Projektbericht

Recognition of Cyber Attacks Using Strategic Threat Models

Rainer Herzog, Lothar Hotz, Björn Kulas, Bernd Neumann, Sonja Weichler

7.7.2018

Content

1 Introduction

In today's connected world, cyber attacks occur with unprecedented frequency, causing ubiquitous damages and enormous costs. Numerous publications report about the impressive sophistication and variability of attacks, induced by vulnerabilities in software systems and user neglect[1]. Defensive measures are often tailored for known specific attacks and only available, when damage has occurred and the attack pattern is known. Preventive protection is clearly more desirable, leading to defensive approaches based on the recognition of suspicious parameter patterns ("signature-based intrusion detection") or anomalous activities ("anomaly detection").

Both approaches, taken by themselves, suffer from uncertain classifications of isolated suspicious events: They either miss out on real threats or report too many suspicions, causing inacceptably large amounts of manual post-checking. Consequently, new approaches were developed which try to classify suspicious events based on a larger context. By considering several events as parts of a strategic attack, one can hope to collect more decisive evidence before raising an alarm, this way avoiding false positives.

Several approaches have been developed for strategic attack recognition, see the section on related work for details. In the work reported here, we propose a new approach following several of these ideas but emphasizing the need for declarative models. Our strategic attack models are formulated as compositional structures with parts obeying constraints, e.g. regarding temporal relations. High-level attack models are defined in an ontological framework and designed to recognize effects of attacks rather than specific signatures. Components of attack models can be shared among different models, and declarative representations facilitate modifications and understanding by non-specialists.

As another advantage, we can automatically generate the high-level part of our attack recognition system, exploiting previous work on behaviour recognition with ontology-based rules [Bohlken et al. 2013]. Low-level recognition of primitive events which form the parts of an attack model is performed by hierarchically structured "recognizers" which evaluate streams of log information from the system under potential attack.

In Section 2, we describe the architecture of our strategic attack recognition system ARGUS in greater detail. Its major components are a high-level model-based recognition system realized by the tool SCENIOR , cooperating with a low-level analytics section where recognizers operate on a continuous stream of log information, and a probabilistic framework for optimizing search. A real-life attack, encountered by the magazine "Süddeutsche Zeitung" in May 2016 and dubbed "SZ attack", will be used as an in-depth example, presented in Section 3. After a discussion of related work in Section 4, we conclude with remarks on lessons learnt and plans for further work.

[1] http://media.kaspersky.com/pdf/it-risks-survey-report-cost-of-security-breaches.pdf

2 Concept of Strategic Attack Recognition

In this section, we present the basic concept of attack recognition with compositional models. We first describe the structure of compositional attack models. In the following subsections, we present the high-level and low-level recognition ramework. Finally, we describe a probabilistic framework which controls the order of evaluation.

2.1 Compositional Models

We know that strategic attacks typically consist of attack phases or parts related to each other and designed to achieve the overall attack goals in collaboration. We represent this by compositional models where attacks are composed of parts in a hierarchical way. This is illustrated in Figure 1 for a webserver attack.

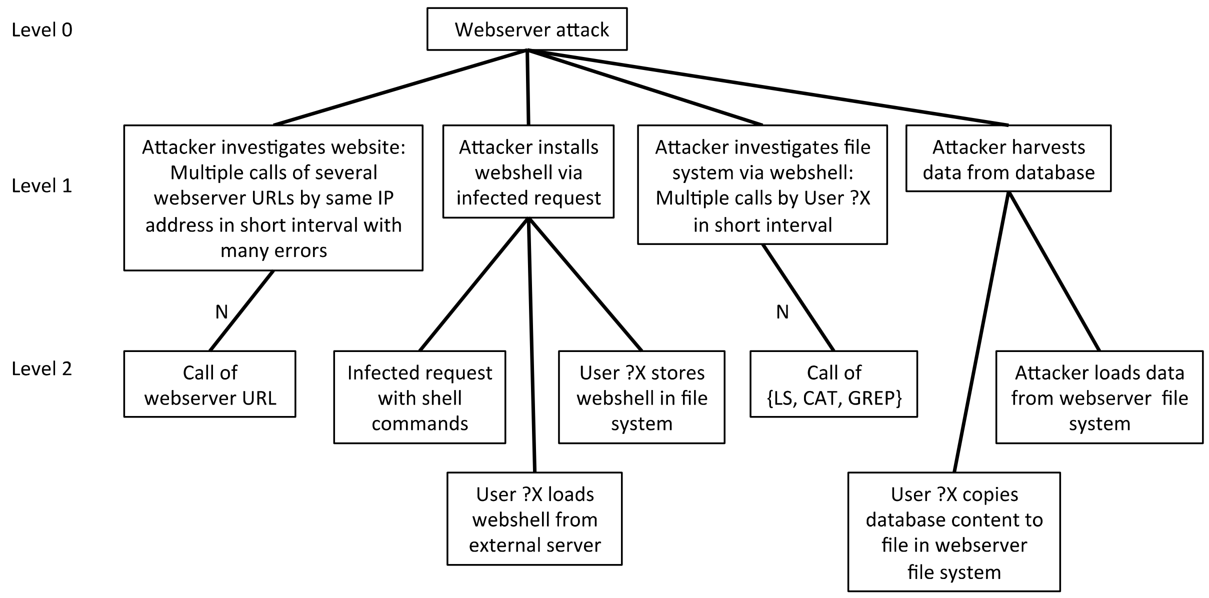
Level 0
Webserver attack
Level 1
Attacker investigates website: Multiple calls of several webserver URLs by same IP address in short interval with many errors
Attacker installs webshell via infected request
Attacker investigates file system via webshell: Multiple calls by User ?X in short interval
Attacker harvests data from database
N
N
Level 2
Call of webserver URL
Infected request with shell commands
User ?X stores webshell in file system
Call of {LS, CAT, GREP}
Attacker loads data from webserver file system
User ?X loads webshell from external server
User ?X copies database content to file in webserver file system

Figure 1: Webserver attack as a compositional model

The top node at level 0 of the model represents the type (or class) of the attack, in this case featuring an intrusion via a webserver. The nodes at Level 1 describe distinct parts of the attack, in this case corresponding to the typical attack phases reconnaissance - intrusion - exploration - exfiltration. The attack phases are further decomposed at Level 2, some of them with parts repeated several times (indicated by N). The leaves of the hierarchical structure represent primitive event types which must be instantiated by sensor information and low-level processing.

Formally, nodes of a compositional model are object classes with attributes and relations, including the special relation "has-part" if a node is composed of parts. Such nodes are called aggregates. Aggregates may also specify constraints which must be fulfilled by their parts. Typical constraints refer to the temporal order or temporal distance of the parts (the order shown in the illustration is not binding), or compare object IDs as illustrated by the use of a common variable ?X in some of the parts. The formal structure of compositional models for recognition applications and the resukting structure of the recognition process is described in [Neumann & Möller 2006].

In view of vulnerabilities of new software and the creativity of attackers, attack models must be easily adaptable to new attack forms. This can be in principle achieved by declarative representations, understandable also by non-specialists. How this can be achieved, has been shown in work on ontology-based behavior representation and recognition for various application domains [Guesgen & Marsland, 2013]. For the domain of cyber attacks, however, we have to observe the need for highly efficient processing of large data volumes at the sensor level. Our approach is therefore based on hybrid attack models with ontology-based declarative structures at higher levels and carefully programmed "Recognizers" at lower levels. By our experience, a significant amount of adaptability and reusability can also be maintained for the low-level Recognizers without jeopardizing efficiency. In our implementation described in more detail in Section 2.3, attack recognition based on the model shown in Figure 1 is realized by ontology-based recognition above Level 2 and Recognizers below.

2.2 High-level Event Recognition

In hybrid attack models, the high-level part refers to an upper section where nodes are described declaratively, and the recognition process is automatically generated from these representations. In our case, this is realized by the tool SCENIOR, originally developed for scene interpretation [Bohlken et al. 2013]. Figure 2 shows the main components of SCENIOR.

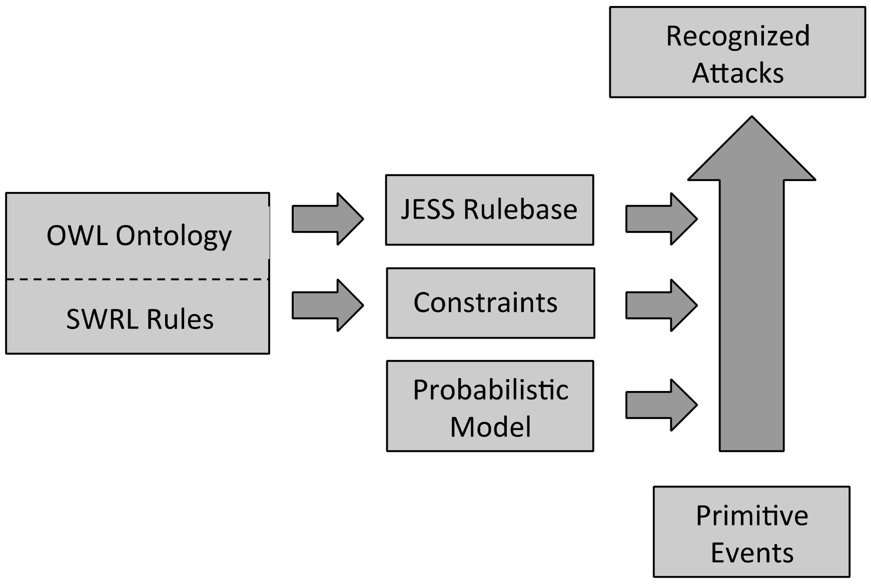

Figure 2: Main components of SCENIOR

Nodes of the compositional attack model are defined in OWL-DL as object classes with attributes and relations. Objects related to parts with a part-of relation are aggregates. Part-of relations are used to generate bottom-up interpretation rules for a JESS rule engine. For each object class which is part of an aggregate, there is a rule to (partially) instantiate a hypothesis for the aggregate if an instance of the part has been received as input, either as a primitive event from the low-level section of the recognition system or from other instantiated aggregates within the high-level system. Rules also express constraints formulated via the SWRL rules attached to the ontology.

Parts, in particular primitive events, may be related to several different aggregates. The recognition process must therefore be able to create and follow up many hypotheses in parallel. It may also be necessary to allow for sensor noise, either in terms of missing or erroneous evidence. SCENIOR can simultaneously accommodate several hundreds of "possible worlds", i.e. alternative sets of instantiations. This may set limits to keeping partial instantiations alive over a long time interval, but the ranking described in Section 2.4 provides a reasonable approach.

2.3 Low-level Event Recognition

We now give an overview of the low-level processes which provide input for model-based recognition described in the preceding subsection. As shown in Figure 3, low-level evidence is obtained by monitoring system logs. Most of these may be unrelated to a possible attack and are ignored for further processing. Classifiers pick out the logs known as relevant for any of the high-level events, such logs are termed C-logs. As an example, a classifier may single out logs documenting file creation. In a further processing stage, logs which satisfy constraints referring to more than a single log are merged into more complex events, called M-logs. For example, logs containing identical IP addresses may be merged this way. This may continue in a hierarchical fashion until evidence required for model-based recognition has been generated.

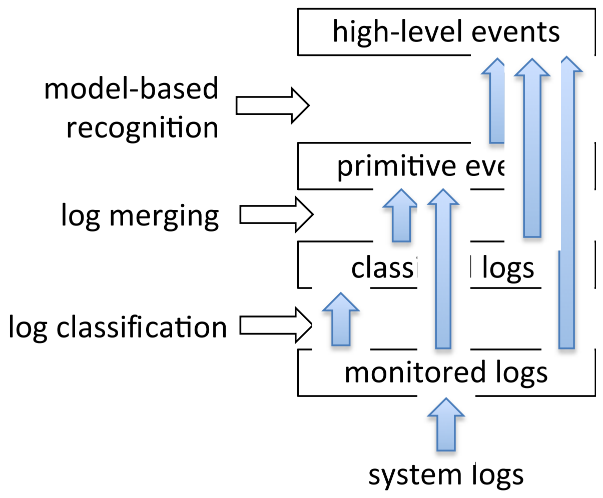

Figure 3: Low-level processing of system logs

Because of the richness and heterogeneity of log information and the special tools required for extracting interesting events in a steady stream of data, we need to provide further details about low-level processing. To avoid confusion, we will distinguish between a "user system" (which must be protected) and the "recognition system" (which is the ARGUS system), although both have to be intertwined to some degree.

Typically, a user system generates a large number of separate logs which can provide diagnostic information. Which logs are monitored, depends on the primitives used by attack models, since log information is used to feed the primitives. Information of different logs may overlap, hence a judicious choice must be made to avoid excessive low-level processing. The following logs typically play important parts:

Gateway log - monitors internet traffic

Webserver log - monitors webserver activities

File system log - monitors access to a file system

Process log - monitors processor activities

Database log - monitors access to a dabase

Figure 4 shows the components and processing steps for log selection and formatting, refining Figure 3. In our work, we used the OpenSource tools Fluentd and osquery to transform logs to JSON and produce input for Kafka. The Kafka platform is used to provide access to selected logs ("monitored logs") as well as intermediate results produced by the Recognizers. The Kafka data stream is organized into "topics" which correspond to various types of log information playing a part in the recognition process. All topics are kept in a Kafka database and remain accessible.

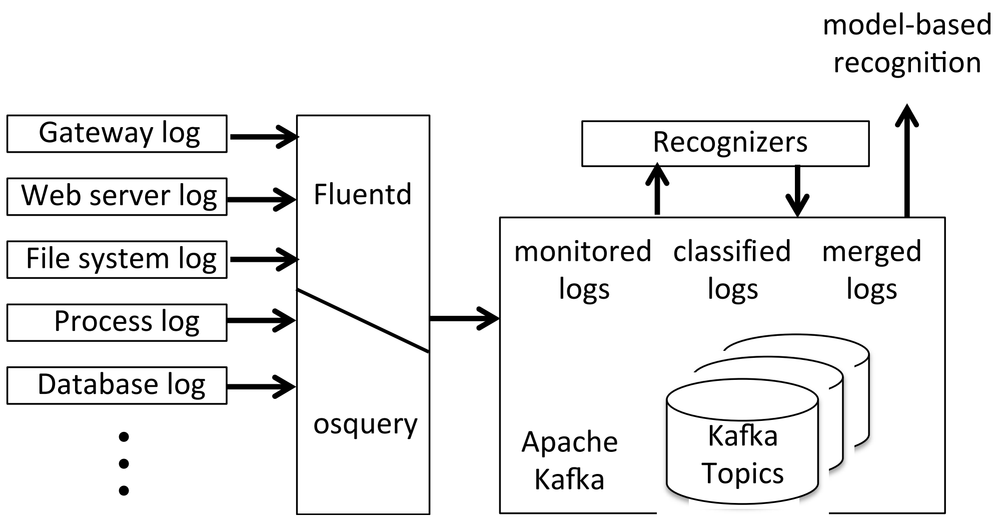

Figure 4: Log selection, formatting, and low-level processing

In the next subsection, we describe the Recognizers which cooperate to generate primitive events for the high-level part of the ARGUS system.

Recognizers can be viewed as filters which continuously screen monitored logs and intermediate Recognizer results. The repertoire of Recognizers is designed to efficiently generate primitives for attack models according to several criteria. First, the processing load resulting from the continuous operation of Recognizers must be kept within manageable limits. This is achieved by a hierarchical structure where low-level Recognizers are kept simple and higher-level Recognizers are only conditionally active depending on low-level results.

A second criterion is reusability. Recognizers should serve as many models as possible and possibly also anticipate attacks not yet covered by existing models. This is achieved by keeping Recognizers as general as possible, avoiding, for example, unnecessary distinctions of nomenclature.

The third criterion is ease of modification. If a new attack model must be served, there should be as little reproramming as possible. This is achieved by limiting the expressivity of Recognizers and by allowing external parametrization.

As indicated in Figure 3, the lowest-level Recognizers are classifiers which filter out single logs whose attributes meet certain constraints, resulting in classified logs (C-logs). For example, a C-log Recognizer may check an URL against a blacklist or report access to certain files. C-log Recognizers are parametrized and can be modified without reprogramming.

More complex Recognizers have several logs as input and generate merged logs (M-logs) as output. An M-log Recognizer may, for example, compare user IDs of selected activities and generate an M-log if the IDs are equal. A more complex example is a Recognizer which counts login attempts or monitors reconnaissance activities on a website in terms of multiple calls to website URLs.

Detailed descriptions of the Recognizers for the SZ attack are provided in Section 3.

2.4 Probabilistic Framework

At any given time, there may be thousands of partially instantiated models representing attack hypotheses. In order to cope with this load, a probabilistic ranking scheme has been developed allowing prioritized processing of the most likely hypotheses. In what follows we sketch the probabilistic framework. It extends prior work by [Neumann & Terzic 2010] by A* ranking of attack hypotheses.

An attack model M^k is associated with a probability distribution P^k which estimates the joint probability of evidence for the primitive nodes of M^k meeting the constraints in the aggregate nodes:

$$P^k = P(\underline{X}^k_1, .. , \underline{X}^k_{Nk}, \underline{Y}^k \mid M^k)$$

The $\underline{X}^k$ are random variables for primitive evidence, $\underline{Y}^k$ describes higher-level properties functionally derived from the $\underline{X}^k$. Constraints, for example on time distances, are reflected by corresponding probabilities, typically 0 for violated constraints. Clearly, these probabilities are difficult to obtain, but even rough estimates are useful for controlling the search for possible attacks. Furthermore, the probabilistic model allows to investigate parameter sensitivity in simulations.

In order to determine the probability of an attack given evidence, we need the priors q^k for the attack models. Again, these reliable values may not exist, but statistics of comparable attacks can be exploited. Finally, a decision about an attack should be based on the risk, i.e. the cost c^{ik} associated with decision alternatives. Assuming that correct decisions have no costs, a decision for an attack with distribution P^1 and prior q^1 must be taken if

$$c^{01}q^1(N)P^1(\underline{a}_1, .. , \underline{a}_N) > c^{10}q^0(N)P^0(\underline{a}_1, .. , \underline{a}_N)$$

where P^0 is the distribution for normal behavior and q^0 the corresponding prior.

The risk of a partially instantiated attack hypothesis is estimated according to the A* Algorithm, based on risk caused by available evidence and an estimate of the risk by missing evidence. This way, it is possible to rank all attack hypotheses, highest-risk first, and react on possible threats in this order. Threats with risk exceeding a threshold, even if only partially instantiated, cause immediate action. Threats below this threshold are kept as hypotheses or discarded.

3 Detecting the SZ Attack

The attack on the Süddeutsche Zeitung (SZ attack) is a typical example of an attack realizing the "kill chain" reconnaissance - intrusion - exploration - exfiltration. Subject of the attack was the SZ blogger service which involves an Apache webserver providing access to Wordpress and the the plugin Popular Posts, the latter using the vulnerable script Timthumb.php. Other kinds of attacks may have a different structure. In a ransomware attack, for example, the main phases are intrusion - encryption - ransom collection - decryption (where the last phase is highly uncertain). We have chosen the SZ attack for a detailed analysis because its software components are freely available, thus allowing a simulation of the attack and its recognition in a lab environment.

3.1 Testbed

The main components of our testbed are shown in Figure 5.

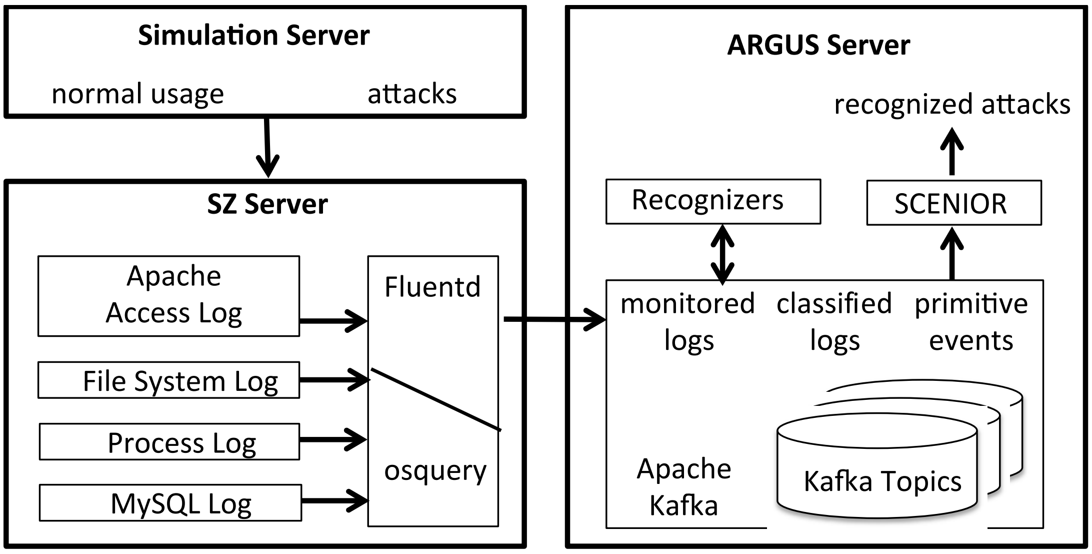

Figure 5: Main components of testbed for simulating and recognizing attacks

As mentioned above, the SZ webserver provides clients with access to Wordpress and related services, it is the target of potential attacks. In order to realistically simulate the use of the ARGUS system, sensors are installed on the SZ Server, while the processing of sensor information takes place on a separate server, the ARGUS Server, minimizing impairment of the SZ services.

A simulation of user behavior and attacks is carried on a third server, the Simulation Server. This simulation was exploited in two ways, first to reconstruct the SZ attack as reported in the media, second to test the ARGUS attack recognition system.

To reconstruct the SZ attack, we investigated how the Timthumb script of Wordpress reacted to an infected command. To this end, an external user (the simulated attacker) sent a request extended by malicious parameters, that allowed for a shell command injection, and we observed in the logs whether the shell command was actually executed. It turned out that this happened in a particular version of Popular Posts which could have been mounted on the SZ server at the time of the attack.

After inspecting several examples of malicious shell injections documented in the internet, we decided to implement a short sequence which requests the download of a file from an external IP address, and prepared the corresponding file to contain a webshell. After having installed the webshell, the attacker would be able to explore the webserver and exfiltrate customer data, as reported for the SZ attack.

The compositional structure of the reconstructed SZ attack is shown in Figure 6.

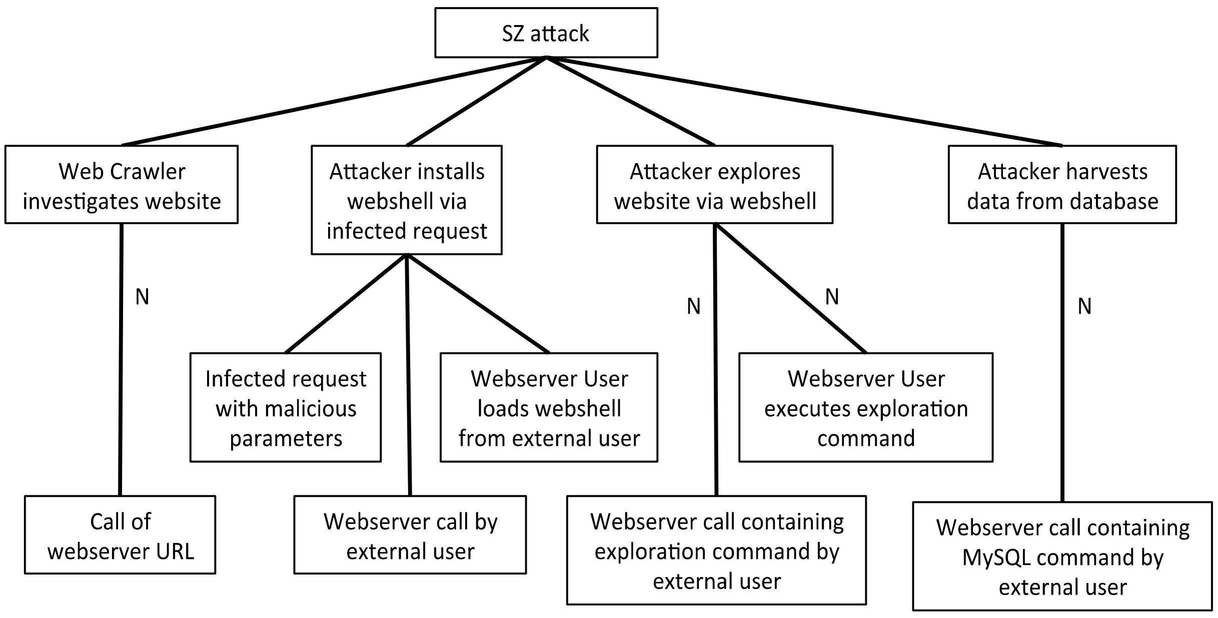
SZ attack
Web Crawler investigates website
Attacker installs webshell via infected request
Attacker explores website via webshell
Attacker harvests data from database
N
N
N
N
Infected request with malicious parameters
Webserver User loads webshell from external user
Webserver User executes exploration command
Call of webserver URL
Webserver call by external user
Webserver call containing exploration command by external user
Webserver call containing MySQL command by external user

Figure 6: Reconstructed SZ attack. Edges marked with "N" indicate repeated events.

In order to simulate attack recognition in our testbed, we set up 10 simulated users providing normal behavior during a timespan of 5 days. Normal behavior consisted of a random mix of typical Wordpress usage:

- CreateNewPostWithRoleAutor
- CommentWithoutLogin
- ApproveCommentsWithRoleAdmin
- TakePause

There were about 30.000 normal user actions on each of the 5 days, distributed over ca. 8 hours daily.

The attack was executed according to the reconstruction in Figure 6, distributed over the 5 simulated days as follows:

- Day 1: normal use
- Day 2: normal use + web crawler (click all links) + first try (generate text file) + second try (download of a web page)
- Day 3: normal use + webshell download
- Day 4: normal use
- Day 5: normal use + webshell execution

The exfiltration part was not simulated. The critical parts of the attack (reconnaissance - intrusion - exploration) were recognized successfully in all simulation runs.

In the following sections, we first describe the low-level and then the high-level processing activities of the ARGUS system for recognizing the SZ attack.

3.2 Monitored Logs

The tool osquery is configured to provide (1) log information for activities regarding files and folders, and (2) log information about processes. In addition, Fluentd is configured to monitor MySQL activities.

3.2.1 Information about Files and Folders

To recognize all changes regarding files and folders, we make use of the osquery table “file_events”. To configure osquery, one saves the query:

```
"file_events": {
  "query":       "SELECT * FROM file_events;",
  "interval":    1,
  "removed":     false
 }
```

One can also describe, which folders to observe. The paths are defined as follows:

```
"file_paths": {
    "etc":      [ "/etc/%%" ],
    "home":     [ "/home/%%" ],
    "var":      [ "/var/www/%%" ],
    "wordpress":[ "/var/www/html/wordpress/%%" ]
}
```

A result of file system monitoring looks like this:

```
{"name":"file_events",
"hostIdentifier":"argusvm",
"calendarTime": "Mon Jun  4 11:24:54 2018 UTC",
     "unixTime":"1528111494",
     "epoch":"0",
     "counter":"0",
     "columns":
          {"action":"OPENED",
          "atime":"1528110081",
          "category":"wordpress",
          "ctime":"1528110062",
          "gid":"33",
          "hashed":"0",
          "inode":"262177",
          "md5":"",
          "mode":"0755",
          "mtime":"1528110062",
          "sha1":"",
          "sha256":"",
          "size":"4096",
          "target_path":"\/var\/www\/html\/wordpress\/",
          "time":"1528111494",
          "transaction_id":"0",
          "uid":"33"},
     "action":"added"}
```

This way, useful information can be read out of individual files or folders, e.g. what type of file manipulation was used and which user was responsible.

3.2.2 Information about Processes

The tool osquery offers several tables which can be queried for process information. We have used the tables "processes" and "process_ events", the former with information about running processes, the latter with accurate information about start and end times. The following shows an example returned by a query to "process_ events":

```
     {"name":"process_events",
     "hostIdentifier":"argusvm",
     "calendarTime":"Wed Jun 13 11:49:22 2018 UTC",
     "unixTime":"1528890562",
     "epoch":"0",
     "counter":"0",
     "columns":
          {"atime":"1528832109",
          "auid":"1000",
          "btime":"0",
          "cmdline":"sudo tail -f
          \/var\/log\/osquery\/osqueryd.results.log",
          "ctime":"1519376758",
          "cwd":"\/home\/argus",
          "egid":"0",
          "euid":"0",
          "gid":"1000",
          "mode":"0104755",
          "mtime":"1499153837",
          "owner_gid":"0",
          "owner_uid":"0",
          "parent":"1495",
          "path":"\/usr\/bin\/sudo",
          "pid":"2915",
          "status":"",
          "time":"1528890562",
          "uid":"1000",
          "uptime":"353"},
     "action":"added"}
```

3.2.3 Information about MySQL

MySQL logs are collected by Fluentd using the pcap-ng plugin. Pcap-ng makes use of the program tshark and directs the returned data directly to a file or to a Kafka topic . So there is no fixed textual log format that could be presented here.

3.2.4 Data Enrichment

There was a need to find a tool that could connect to Kafka, transport the log lines and enrich the log lines with further information. To recognize attack steps later, each record must uphold a unique ID. Furthermore the start and end timestamps of the attack phases must be changed to unix time. It turns out that Fluentd offers several plugins for transportation and enrichment. It is also possible to run ruby code within the configuration file, which is nice for transforming the apache timestamps into unix timestamps and to create unique IDs for each log line.

3.3 Recognizers

The two types of Recognizers used in the ARGUS system, C-log and M-log Recognizers, have been introduced in Section 2. Here we describe the events recognized for the SZ attack. Figure 7 provides an overview.

In order to understand the logics behind the Recognizers, it is necessary to explain the conditions checked in the aggregate nodes. We begin with the top-level events which constitute primitives for the high-level section of the ARGUS system.

The node "Webserver file created by webserver access" is an M-log generated from two components by a corresponding M-log Recognizer. The first component, "Webserver file created. Filename in webserver access within restricted interval.", signals that the webserver has created a file, where the filename has occurred in a webserver call a restricted time ago. In the case of a script file this is highly unusual due to the Webserver User as file creator: The Webserver User is a fixed ID (User 33 in the case of our Apache webserver in our Ubuntu-Linux environment) only used by the webserver. In combination with an access call containing the same file name, this raises the likelihood of an installment of malicious code.

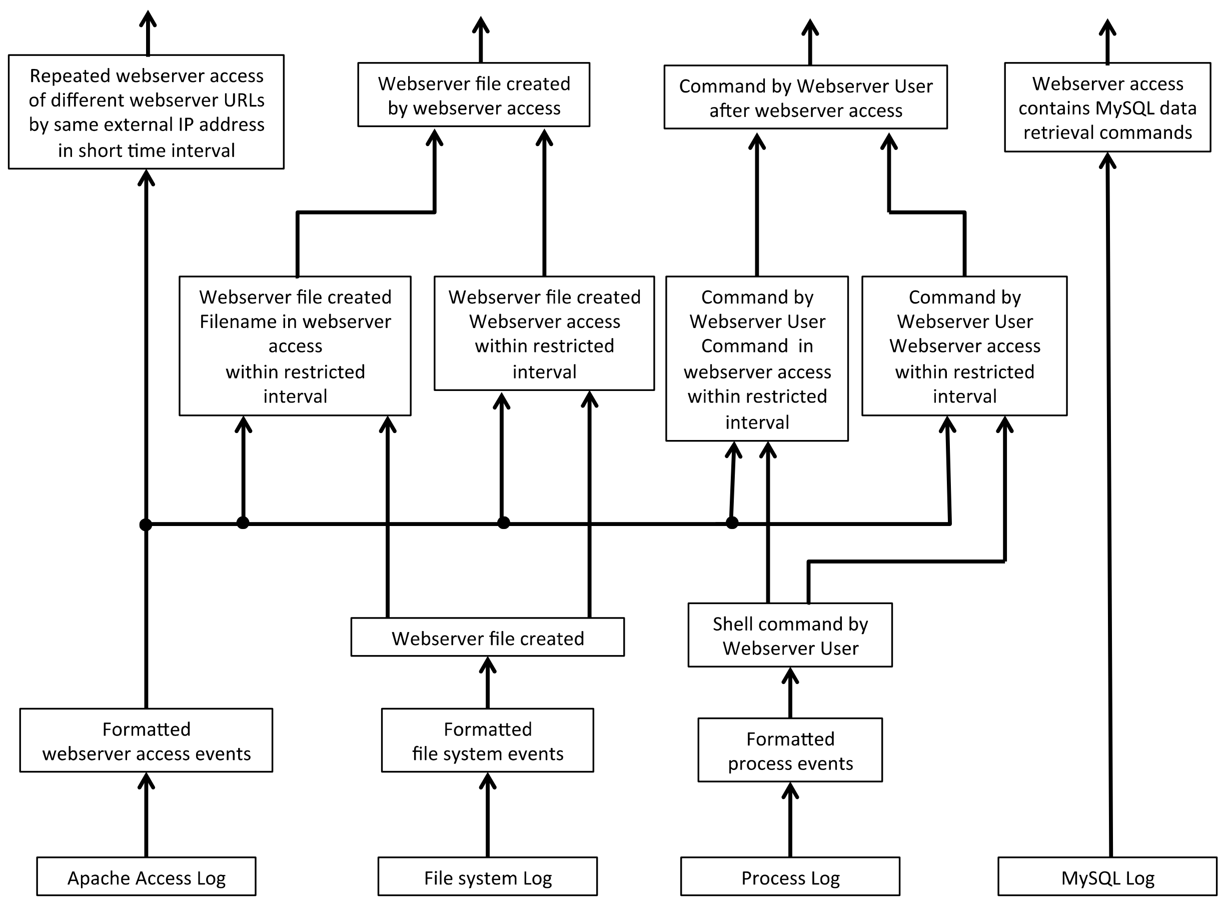
Repeated webserver access of different webserver URLs by same external IP address in short time interval
Webserver file created by webserver access
Command by Webserver User after webserver access
Webserver access contains MySQL data retrieval commands
Webserver file created Filename in webserver access within restricted interval
Webserver file created Webserver access within restricted interval
Command by Webserver User Command in webserver access within restricted interval
Command by Webserver User Webserver access within restricted interval
Webserver file created
Shell command by Webserver User
Formatted webserver access events
Formatted file system events
Formatted process events
Apache Access Log
File system Log
Process Log
MySQL Log

Figure 7: Events recognized for the SZ attack

The second component, "Webserver file created. Webserver access within restricted interval" signals a less restricted event, also in conjunction with a webserver access, but without detecting the filename in the webserver call. The two components are merged by selecting the more likely.

The node "Command by Webserver User after webserver access" is generated from two components in a similar way. The first component, "Command by Webserver User. Command in webserver access within restricted interval." signals that the Webserver User (explained above) has issued an exploration command, e.g. a CAT, CD or LS, and this command has been transmitted via the HTTP method GET by a prior webserver access.

The second component signals the somewhat less suspicious event that such a command has been issued, a prior web access has been monitored, but the command has not been recognized in that access log, as it might have been transferred with the HTTP method POST.

The components described above are M-logs based on webserver access logs and on two C-logs derived from file system logs and process logs, respectively. The events "Webserver file created" and "Shell command by Webserver User" are typical examples of multipurpose intermediate results, stored as Kafka topics for further usage and possibly supporting other attack models.

3.4 High-level Recognition of the SZ Attack

The basic method for model-based recognition of cyber attacks and the tool SCENIOR have been presented in Section 2.2. We now first describe the models used for recognizing the SZ attack, and then explain the high-level recognition process..

3.4.1 Ontology for the SZ Attack

Figure 8 illustrates the main OWL-DL ontology classes used for the attack definition.

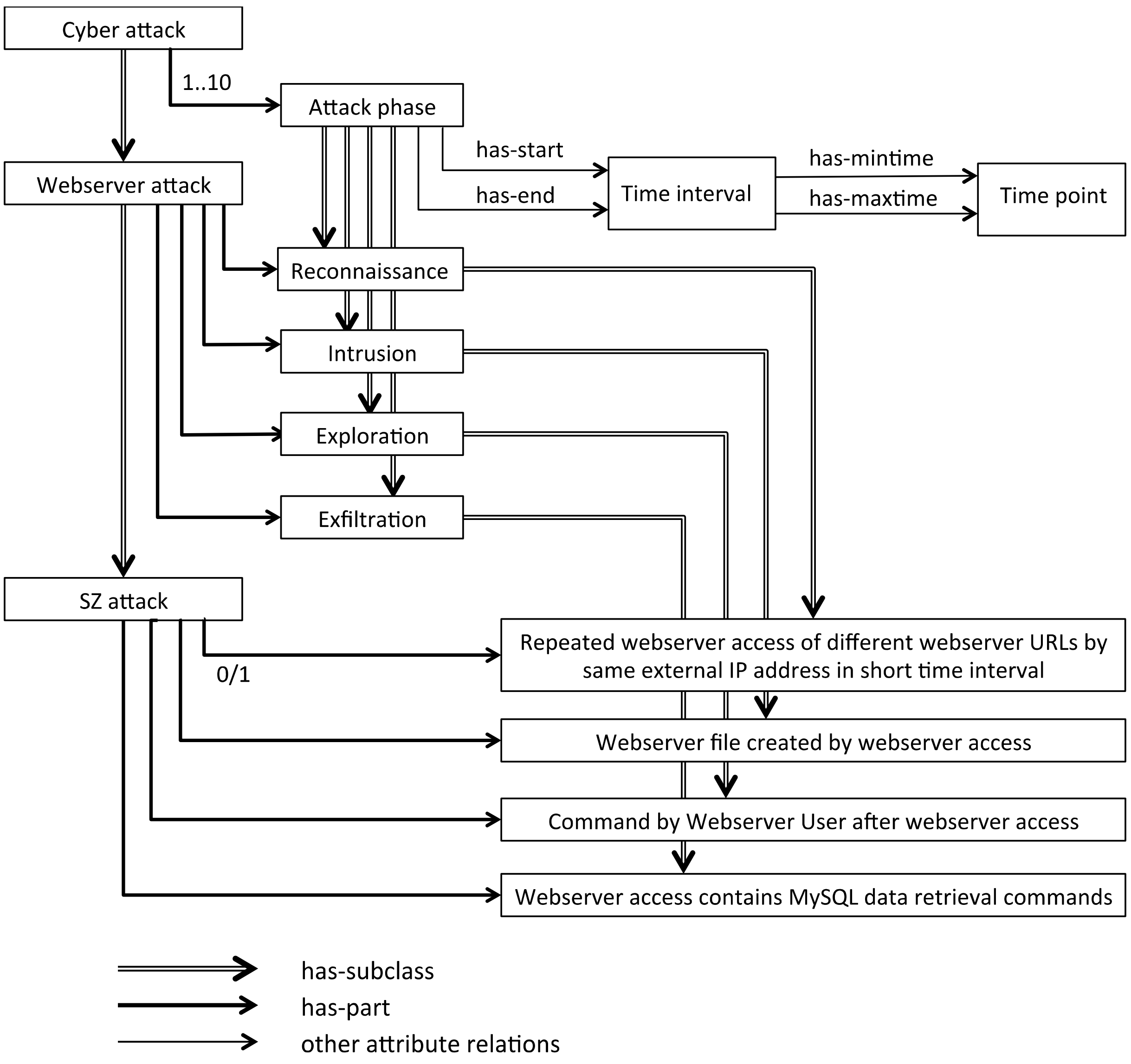
Cyber attack
1..10
Attack phase
has-start
has-end
Time interval
has-mintime
has-maxtime
Time point
Webserver attack
Reconnaissance
Intrusion
Exploration
Exfiltration
SZ attack
0/1
Repeated webserver access of different webserver URLs by same external IP address in short time interval
Webserver file created by webserver access
Command by Webserver User after webserver access
Webserver access contains MySQL data retrieval commands
has-subclass
has-part
other attribute relations

Figure 8: Objects and relations of the Cyber attack ontology

In this ontology, a *Cyber attack* may have up to 10 parts, each defined as an *Attack phase*. Attack phases are the most general constituents of an attack. Each attack phase has a start interval and an end interval defining the earliest and the latest occurrence of start and end, respectively.

A *Webserver attack* is a specialization of a *Cyber attack*, it inherites all relations of the superclass. The parts of the *Webserver attack* are *Reconnaissance, Intrusion, Exploration* and *Exfiltration*. An *SZ attack* is a further specialization of a *Webserver Attack*. The parts of an SZ attack are specializations of the corresponding parts of a *Webserver attack*. In the simple attck model presented here, they describe the primitives which will be accepted as evidence. The compositional hierarchy is quite flat in our experiment owing to our focus on the low-level implementation.

The ontology includes several further attributes not shown in the figure for clarity. The primitives, in particular, provide information which must satisfy constraints between different parts of the SZ attack. For example, the external IP used for exploration and exfiltration must be the same for data retrieval from MySQL to be judged as malicious.

There is no general way to extend OWL-DL by constraints because of decidability issues. The Semantic Web Rule Language SWRL, however, is a formalism which can be used in a "DL safe" way as long as rules are only applied to known individuals. Fortunately, this can be guaranteed for our system architecture: Rules (containing constraint checks) are only invoked for concrete evidence. In SCENIOR, this is realized by automatically translating the ontology and the SWRL rules into JESS rules. JESS is a rule-based system in the OPS5 tradition, implemented in Java.

3.4.2 Rule-based Recognition with JESS

Initially, all attack models of the ontology are transformed into structures in the JESS working memory and marked as hypotheses without evidence. In the case of the SZ attack ontology, the possibility of having an attack with or without a reconnaissance phase is reflected by two corresponding hypotheses. Each hypothesis is equipped with a temporal constraint net which keeps track of the admissable time intervals for start and end times of all events which can be assigned to the hypothesis.

In addition, rules are created for the JESS rule base, which control the recognition process. SCENIOR generates five types of rules which allow mixed bottom-up and top-down processing depending on the kind of evidence. For our attack recognition application, we only need bottom-up processing where Evidence Assigment Rules and Aggregate Instantiation Rules perform all recognition steps.

In Evidence Assignment Rules, the precondition part of a rule checks whether the evidence has the proper class for part of an aggregate, whether this part is not yet instantiated, and whether time constraints and possibly further conditions are met. If the rule fires, the corresponding part of the agggregate hypothesis is marked as instantiated.

If all parts of an aggregate are instantiated, the Aggregate Instantiation Rule fires and markes the aggregate as instantiated. This may trigger Evidence Assigment Rules which assign the instantiated aggregate to a part of a higher-level aggregate, or signal an attack if the instantiated aggregate represents this event at the top level.

Recognition of the SZ attack was performed with only few failing attack hypotheses because of parts rarely occurring in normal behavior. The only normal behavior events which partially instantiated an attack hypothesis wre of the kind "Webserver access contains MySQL data retrieval commands". But all such events did not meet the constraint that the external IP used for exploration and exfiltration must be the same. Hence these hypotheses died after their time constraints expired.

4 Related Work

It has been recognized for many years that cyber attacks occur in several steps or phases, and should be described coherently, both to facilitate recognition and to eleviate communication about attacks.

An early usage of a stepwise decomposition of attacks in terms of attack trees is reported in [Byres et al. 2004]. The authors suggest to evaluate the vulnerability of SCADA systems by means of eleven attack trees identified as possible attack structures. The work aims at supporting humans, hence the structural representation.

Regarding communication, an important modern contribution is STIX[2] (Structured Threat Information Expression), designed to enable organizations to share cyber threat intelligence with one another

> "in a consistent and machine readable manner, allowing security communities to better understand what computer-based attacks they are most likely to see and to anticipate and/or respond to those attacks faster and more effectively."

The core concepts of offer a rich vocabulary for structured attack descriptions, albeit with the main purpose of giving organizations a tool for communication. The importaant step towards automatic attack recognition based on STIX descriptions has not been attempted to our knowledge.

The work reported in [Paudel et al. 2017] has similar goals. They describe the multi-level hierarchical structure of cyber attacks on WAMSs (Wide Area Monitoring Systems) using AND-OR trees, and thus provide comprehensive information about different ways to achieve intrusion and about collaboration between attack steps. The trees are similar to the compositional models in our work, but the authors do not mention a formal procedure for generating a corresponding attack recognition system.

Looking at existing approaches to activity recognition in real-life scenarios, there are obvious parallels which can be exploited: Sequences of actions can be modelled as parts of higher-level activities. Prior knowledge about a compositional structure can be exploited for recognition. [Artikis et al. 2013] consider the logical structure of activities and derive rules which can be evaluated with a rule-based system. Different from our approach, they do not ressort to a formal ontology such as OWL-DL. Furthermore, their sensor-based primitives fit more easily into higher-level models than computer logs of cyber attacks.

In view of the variability of cyber attacks, there are also strong arguments to turn to probabilistic models as a basis for recognition. One approach, described in [Valeur et al. 2004], propagates aggregation and correlation of alerts obtained by different intrusion detection systems (IDSs). While this undoubtedly improves recognition rates, the attack

[2] https://docs.oasis-open.org/cti/stix/v2.0/stix-v2.0-part1-stix-core.pdf

itself is taken to be monolithic. Hence the correlations between different parts or phases of a strategic attack are not captured.

[Qin & Lee 2004] investigate tree-shaped causal networks as attack models. They show that such models can be inferred from correlations between low-level events, and that higher-level strategies can be discovered this way. For examples they use DARPA's Grand Challenge Problem (GCP) as data set which is known to be much easier accessible than modern attack traces.

In summary, our sampling of past work on attack recognition based on compositional models has shown several examples of structured and tree-shaped attack models, but no attempts to derive from these models corresponding attack recognition systems, as presented in this report.

5 Conclusions

We have presented a new approach to building systems for the recognition of strategic cyber attacks based on compositional models. We have described a hybrid recognition system with a model-based high-level section, adapting existing behavior recognition technology to the cyber security domain, and a programmed low-level section for real-time analysis of log data. Experiments in a testbed have shown 100% recognition rate for a real-life attack embedded in simulated normal user behavior.

The work can be considered a first step towards a commercially attractive defense system with an extremely low rate of false positives due to alarms based on a larger contextof typically several low-level attack steps instead of a single suspicious occurrence.

There remain, however, caveats and open questions to be addressed in further work. In view of the ingenuity of attackers, a complementary approach based on anomaly detection is important. This has, in fact, been realized in the ARGUS project with promising results, reported in [Lorenz 2018]. Furthermore, user interfaces reacting to alarms and maintaining the model database have not been tried in realistic environments. Hence the promise of understandable attack models and adaptability has not yet been proved. But the sucvcess of declarative knowledge-based systems in other application areas is a good indicator.

References

[Artikis et al. 2013]
Alexander Artikis, Marek Sergot, Gergios Paliouras: A logic-based approach to activity recognition. In [Guesgen & Marsland 2013], 1-13.

[Bohlken et al. 2013]
Wilfried Bohlken, Patrick Koopmann, Lothar Hotz, Bernd Neumann: Towards Ontology-Based Realtime Behaviour Interpretation. In [Guesgen & Marsland 2013], 33-64.

[Byres et al. 2004]
E.J. Byres, M. Franz, D. Miller: The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems. International Infrastructure Survivability Workshop (IISW'04), Institute of Electrical and Electronics Engineers, Lisbon, 2004

[Dudorov et al. 2013]
Dmitry Dudorov, David Stupples, Martin Newby: Probability Analysis of Cyber Attack Paths against Business and Commercial Enterprise Systems. 2013 European Intelligence and Security Informatics Conference, 2013

[Guesgen & Marsland 2013]
Hans-Werner Guesgen, Stephen Marsland (eds.): Human Behavior Recognition Technologies: Intelligent Applications for Monitoring and Security, IGI Global, 2013

[Lorenz 2018]
Max Lorenz: Log-Analysis, Anomaly Detection. ARGUS Projektbericht, 2018

[Neumann & Möller 2006]
Bernd Neumann, Ralf Möller: On Scene Interpretation with Description Logics. In: Cognitive Vision Systems, H.-H. Nagel and H. Christensen (Hrsg.), Springer, LNCS 3948, 2006, 247-275

[Neumann & Terzic 2010]
Bernd Neumann, Kasim Terziç: Context-based Probabilistic Scene Interpretation. In: Proc. IFIP AI-2010, Brisbane, Sept. 2010, 155-164

[Paudel et al. 2017]
Sarita Paudel, Paul Smith, Tanja Zseby: Attack Models for Advanced Persistent Threats in Smart Grid Wide Area Monitoring. In CPSR-SG'17 Proceedings of the 2nd Workshop on Cyber-Physical Security and Resilience in Smart Grids, 61-66

[Qin & Lee 2004]
Xinzhou Qin, Wenke Lee: Attack Plan Recognition and Prediction Using Causal Networks. 20th Annual Computer Security Applications Conference, 2004

[Valeur et al. 2004]
Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, Richard A. Kemmerer: A Comprehensive Approach to Intrusion Detection Alert Correlation. IEEE Trans. on Dependable and Secure Computing, Vol. 1, No. 3, 2004, 146-169