# Efficient Distributed Intrusion Detection applying Multi Step Signatures

Michael Vogel and Sebastian Schmerl

**Brandenburg University of Technology**
**Cottbus, Germany**
{mv|sbs}@informatik.tu-cottbus.de

## Abstract

Intrusion Detection Systems (IDS) offer valuable measures to cope with today's attacks on computers and networks. But the increasing performance of networks and end systems and the growing complexity of IT systems lead to rapidly growing volumes of observation data and large signature bases. Therefore, IDS are forced to drop observations in high load situations offering chances to attackers to act undetectable. We introduce an efficient dynamically adaptable, distributed approach for a multi-step signature based IDS. Finally, we discuss initial performance evaluations of a prototype implementation and motivate future work scopes.

**Keywords and phrases** Computer Security, Distributed Intrusion Detection, Attack Signatures

**Digital Object Identifier** 10.4230/OASIcs.KiVS.2011.188

## 1 Motivation

Intrusion Detection Systems (IDS) have been proven as important instruments for the protection of computer systems and networks. IDSs consist of sensors and analysis units. Sensors are either network-based or host-based. *Host sensors* monitor the activities of applications and the operating system on observed hosts (e.g. system calls). *Network Sensors* monitor network traffic at mirror ports of routers and switches. The sensors capture security relevant activities from these observations and continuously output a stream of *audit events* which is passed to an *analysis unit*. IDSs apply either pattern anomaly or misuse detection. Misuse detection searches for traces of security violations in captured observations (audit data), using known attack patterns – the signatures.

A challenge that all intrusion detection systems are facing today is the increasing performance of both networks and end systems. This leads to a rapid growth of audit data volumes to be analyzed. On the other hand, the growing complexity of the IT-systems causes novel vulnerabilities and offers new possibilities for running attacks so that the number of signatures to be analyzed increases as well. Already today intrusion detection systems are forced to drop audit data in high load situations. Thus, countermeasures become impossible. This provides the attackers also the possibility to apply a "be patient"-attack-strategy which portions their malicious activities over several days to exploit the fact that overloaded IDS discard detection results regularly, due to limited memory resources.

To cope with this situation several approaches have been proposed, e.g. the detection of intrusions based on an analysis of more compact, less detailed network log data [3]. But all these approaches aim at optimizing the non-distributed, single threaded signature analysis. The Gnort approach in [7] utilizes the massive parallel computing capabilities of expensive, high end graphic processors (GPUs), but it only doubles the analysis throughput compared to the sequential Snort tool. In contrast to today's highly loaded IDS, extensive spare computing resources are available on cheap desktop machines in every network. Therefore,

17th GI/ITG Conference on Communication in Distributed Systems (KiVS'11).
Editors: Norbert Luttenberger, Hagen Peters; pp. 188–193

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a distributed signature analysis is a possible way to overcome this issue. So far, today's network-based IDS only apply primitive means to parallelize packet analysis by load balancing mechanisms [1]. There are also almost no approaches to parallelize host-based audit data analyses [2].

## 2 Efficient Distributed Intrusion Detection

Today, most IDS used in practice use a centralistic approach and apply misuse detection (e.g. Snort [6]). The implementation and configuration of misuse detection systems are simpler and allow for a significantly higher detection accuracy compared to anomaly detection. Typically, single step signatures are applied to detect attacks by analyzing a single audit event (e.g. network packet). In contrast to single step signatures, multi-step signatures allow for a fine-grained modeling of attacks. They specify more detailed characteristic attack traces, their dependencies, and the chronological order of the steps. Many semantic aspects of multi-step attacks cannot be modeled by single-step signatures or only insufficiently, which results in an increased false alarm rate (false positives). Therefore, in the following we focus on misuse detection based on the analysis of multi-step signatures. Our approach bases on an existing non-distributed IDS, which applies multi-step signatures defined in the Event Description Language (EDL) [5]. The state/transition approach EDL uses a Petri net like description principle.

### 2.1 EDL Signatures

EDL descriptions consist of places and transitions connected by directed edges. *Places* represent relevant system states of an attack. Hence, they characterize the attack progress. *Transitions* describe state changes triggered by audit events from the audit data stream recorded during an attack. An example of an EDL signature with four places ($P_1$ – $P_4$) and three transitions ($T_1$ – $T_3$) is depicted in Fig. 1. Ongoing attacks are represented by tokens

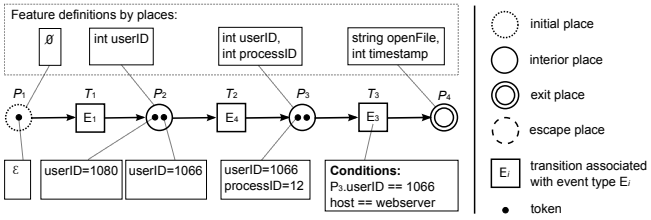


**Figure 1** EDL Signature Example

on places. Tokens can be labeled with values like colored Petri nets. A place defines zero or more features which specify the properties of tokens located on this place. EDL distinguishes four place types: initial, interior, escape and exit. *Initial places* are signature's starting places which are marked with an initial token at start up. The *exit place* represents the completion of an attack instance. Thus, a token reaches this place implies that an attack has been detected. *Escape places* indicate that events occurred in the audit data stream which make the completion of an attack instance impossible. Therefore, tokens reaching places of this type are discarded. All other places are *interior places*.

A transition that is triggered by an event of a certain type can also contain a set of conditions. As shown in Fig. 1, these conditions can specify constraints over certain features of the triggering event (e.g., `host=webserver`) and token values (e.g., `P3.userID=1066`). The

evaluation of these transition conditions requires CPU time depending on the complexity of the conditions and the frequency of the evaluation, which is determined by the number of occurring events in the audit data and the number of tokens on input places of a transition.

## 2.2 Distributed Analysis

In order to cope with overload situations, which often forces today's IDS to drop audit data, the required analysis effort can be distributed among a set of cooperating analysis units on different hosts or multiple CPU cores. Primitive load balancing mechanisms, assigning audit events to some analysis units cannot be used for multi-step signature-based IDS, because an attack consists of a chain of distinct attack traces (audit events). Instead, the signature base of the IDS can be split up into a number of distinct subsets which are assigned to different cooperating analysis units. But this requires to transmit each audit event many times (from the sensor to each host that runs an analysis unit). Therefore, a more sophisticated signature distribution, which minimizes the need to duplicate captured audit events is desirable. This can be achieved by identifying fine grained minimal parts in each multi-step signature that can be independently assigned to different analysis units. This allows to optimize the required communication effort for distributing and duplicating captured audit data for each distributed analysis unit. By pooling minimal signature parts which analyze the same type of audit events and assigning them to the same analysis unit, events of this type only have to be sent to a subset of the analysis units. As an example, we consider an EDL signature that

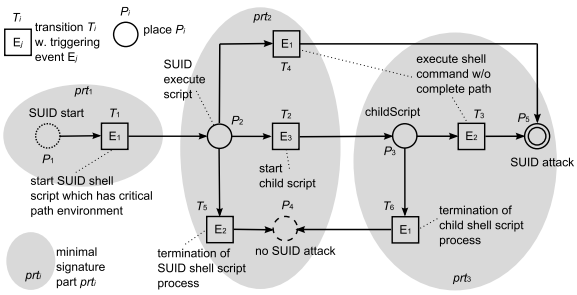


**Figure 2** Minimal parts of SUID example signature

describes the SUID (set user ID) script attack on a former version of the Solaris OS, which is depicted in Fig. 2. Without explaining the attack in detail, the attacker tries to gain administrative privileges by exploiting a vulnerability of the extended file access rights.

The gray shaded spheres in Fig. 2 represent the three minimal parts, this signature can be partitioned into. These parts cannot be partitioned further, because the transitions have to evaluate the tokens on their input places. Therefore, a transition and its input places have to be assigned to the same signature part. Now, if signature parts are assigned to different analysis units, a token forward mechanism is required. In Fig. 2, the execution of transition $T_2$ from signature part $prt_2$ requires to place a token on the output place $P_3$ which belongs to different signature part $prt_3$. If $prt_2$ and $prt_3$ are assigned to different analysis hosts the token has to be forwarded via a communication channel between both analysis hosts.

The basic concept of a prototype implementation of our approach is depicted in Figure 3. A *sensor* logs audit events and classifies them into different event types (e.g. network protocol, port, specific system calls). The signature base is splitted up into minimal signature parts (prt*ij*) and each part is assigned to one *analysis unit* (AU), which allows for many different assignments of the signature base to avaliable analysis units. A configurable *filter*, knowing

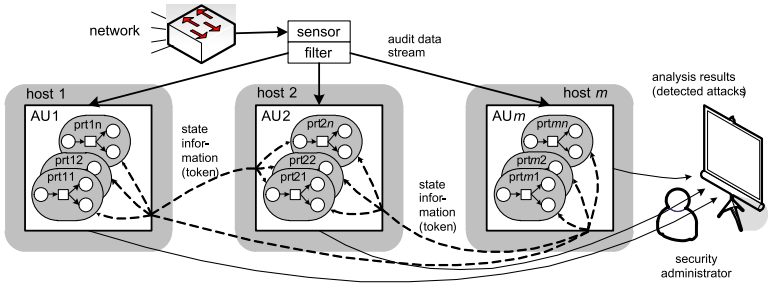


**Figure 3** Distributed IDS Overview

the current signature partitioning, discards non-relevant audit data and transmits only those event types to each analysis unit, which are analyzed by them to limit the communication effort. The distributed analysis units examine incoming audit events by evaluating the transition conditions of assigned signature parts. If a transition is activated and a token has to be placed on an output place, which was assigned to a remote analysis unit, then this token is transmitted to the responsible analysis unit. The results of the distributed analysis are aggregated and evaluated by a *security administrator*.

## 3 Initial Evaluations

The distribution strategy described in the previous section has been implemented and evaluated using a distributed Intrusion Detection System, which bases on EDL multi-step signatures [4]. We modified and extended our existing Intrusion Detection tool SAM (signature analysis module) according to Figure 3 to convert it into a distributed IDS. SAM sensors forward audit events to a configurable number of analysis units over network connections (sockets). Each SAM analysis unit possesses a configurable set of EDL signatures. This implies that a configurable set of minimal signature parts is assigned to each analysis unit. The analysis units exchange state information (tokens) between each other also over network connections. We choose three example signatures and calculated all possible partitions (assignments) of the contained signature parts to three analysis units. We evaluated each of these 965 partitions on three parallel running SAM analysis units. These partitions include very efficient distributions as well as completely inappropriate, inefficient ones, causing badly balanced loads of the analysis units. Because of limited space, only two of the used signature examples are explained in the following.

The first example describes the SUID (set user ID) script attack that already has been introduced in the previous section. Another used signature example, consisting of 8 places and 15 transitions, which is not depicted here due to lack of space, describes the link shell attack which exploits a vulnerability of a former version of the Solaris OS. The attack exploits a specific shell function as well as the SUID (set user ID) mechanism. If a link refers to a shell script and the scripts filename starts with a hyphen "-" then an attacker can get an interactive shell having the access rights of the script owner (e.g. root) by executing the link.

In order to evaluate the analysis efficiency of our distributed system we first used a generic set of audit data. The data set was created by capturing system calls of a host while the described attacks were executed. Afterwards, all logged system calls, which do not belong to the executed attacks have been discarded manually. Therefore, the audit data set only contains relevant attack traces of the applied signatures. Concerning the required analysis effort, this represents the worst case scenario, demanding for the maximum computation

effort for analysis. Additionally, the captured attack traces have been duplicated to create a sufficiently large audit data file of 6,000 events (system calls). The experiments were conducted on four separate machines (Intel Xeon, 2.66 GHz, 512 KB L2 cache, 2 GB RAM) which are connected by switched Fast Ethernet links. One machine executes the sensor; the others each run an analysis unit. At first we applied the generic audit data set and evaluated, that even assigning the three example signatures to different analysis units, without splitting them into parts, leads to a runtime improvement. We measured the run time separately for each analysis unit. Then, we evaluated if further runtime improvements can be achieved, by fine grained assignment of signature parts to different AUs and which signature partition turned out to be efficient. Therefore, we evaluated all 965 possible different distributions of our signature examples to three analysis units. The Table 1 contains runtime evaluations for some selected signature distributions. The sensor runtime is related to the slowest analysis unit, as the sensor terminates after transmitting the last audit event (to the slowest AU). The first row (id 0) represents the non-distributed case. Thus, only one analysis unit (client)

| **distribution** | **Sensor** | **Client 1** | | **Client 2** | | **Client 3** | |
|---|---|---|---|---|---|---|---|
| id. | real [s] | real [s] | user [s] | real [s] | user [s] | real [s] | user [s] |
| 0 | 47.953 | 47.625 | 46.89 | | | | |
| 96 | 29.625 | 37.718 | 28.672 | 29.328 | 9.828 | 32.390 | 7.844 |
| 302 | 110.641 | 110.656 | 108.469 | 116.343 | 33.891 | 113.718 | 19.313 |
| 626 | 19.719 | 19.750 | 17.948 | 25.437 | 18.078 | 23.453 | 17.969 |

**Table 1** Selected runtimes for different signature distributions

is used, which gets all signatures assigned. This is the benchmark for any optimizations by signature distribution. The second row (id 96) represents the obvious distribution, which assigns each of the three example signatures completely to a different analysis unit (clients). Thus, the signatures are not split up into parts and no state information (tokens) have to be exchanged between different analysis units. The runtime evaluation shows a relevant improvement for the distributed case. The real runtime of the sensor, which captures and sends the audit data decreases by roughly 60 %. That means the distributed analysis run completes 60 % faster than the non-distributed run. But the really consumed CPU time (user time) of the three clients indicate that the analysis distribution is not well balanced. Client 1 (28.672 sec) consumes significantly more CPU time than the other clients (9.828 sec, 7.844 sec). Further, the last row shows the signature distribution (id 626) requiring the least runtime to complete, which is significantly more efficient than the distribution (id 96). Here, parts of a signature are fine-grained assigned to the different AUs and the sensor requires only half of the runtime compared to the non-distributed case. Further the required CPU times (user time) of the clients indicate a well balanced signature distribution among three clients. Finally, the third row (id 302) shows runtime results of the expected worst suitable analysis distribution requiring the maximum runtime (110 sec). Thus, by choosing a bad signature distribution, where signatures are split up poorly, the distributed analysis can take substantially longer compared to the non-distributed case (id 0). The evaluation of all possible 965 different partitions of the signature base shows that there are many efficient signature partitions offering short run times as well as some completely unsuitable partitions.

An efficient distributed IDS applying EDL signatures has to choose one of the partitions, requiring a low overall runtime. Therefore, it is necessary to introduce, resp. create a metrics which maps features of the audit data characteristics (e.g. number of occurred

specific audit events), monitored by the sensor, as well as the statistics maintained by the analysis units (detailed logs of spend communication and computation effort) for the currently used signature distribution to a metrics value $M$. This metrics then should be applied to continuously predict suitable assignments of signature parts to available analysis units. The distributed IDS then can use the metrics predictions to dynamically adapt the analysis configuration to the ever changing characteristics of the captured audit data, as well as changing available analysis resources, by reassigning signature parts to other analysis units. But even without a predicting metrics our approach is not useless. A suitable, resp. efficient initial partition can always be achieved by simply equally assigning whole signatures (without splitting them) to the analysis units in a round-robin manner (e.g. id 96 in Table 1).

## 4 Summary and Future Work

We presented an efficient approach for distributed IDS, which aims at balancing the computation load of complex signature-based IDS across multiple analysis units. The approach is not limited to EDL signatures and thus can be adapted to any multi-step signature based IDS. A prototype implementation was used to initially examine the achievable performance improvements by distributing the audit data analysis to a number of parallel running analysis units. We applied a set of three example signatures, split up into minimal signature parts. Then, we evaluated the analysis performance for the non-distributed case (as a benchmark) and all possible, different assignments of signature parts to three distinct analysis units. Evaluation shows many suitable signature partitions, which run significantly faster, but also worse ones, requiring much more run time, compared to the non-distributed baseline. Further work to create a metrics which predicts expectable computation and communication effort for different signature partitions based on audit data characteristics is in progress. The metrics should enable a distributed IDS to dynamically reconfigure its analysis distribution in order to adapt to changing analysis effort and available resources.

## References


1 Michele Colajanni and Mirco Marchetti. A parallel architecture for stateful intrusion detection in high traffic networks. In *Proc. of the IEEE/IST Workshop on "Monitoring, attack detection and mitigation" (MonAM 2006)*, Tuebingen, Germany, September 2006.

2 Christopher Krügel, Thomas Toth, and Clemens Kerer. Decentralized event correlation for intrusion detection. In *ICISC*, volume 2288 of *Lecture Notes in Computer Science*, pages 114–131. Springer, 2001.

3 John McHugh. Sets, bags, and rock and roll: Analyzing large data sets of network data. In *ESORICS*, volume 3193 of *Lecture Notes in Computer Science*, pages 407–422. Springer, 2004.

4 Michael Meier. A model for the semantics of attack signatures in misuse detection systems. In *ISC*, volume 3225 of *Lecture Notes in Computer Science*, pages 158–169. Springer, 2004.

5 Michael Meier, Sebastian Schmerl, and Hartmut König. Improving the efficiency of misuse detection. In *DIMVA*, volume 3548 of *Lecture Notes in Computer Science*, pages 188–205. Springer, 2005.

6 Martin Roesch. Snort: Lightweight intrusion detection for networks. In *LISA*, pages 229–238. USENIX, 1999.

7 Giorgos Vasiliadis, Spyros Antonatos, Michalis Polychronakis, Evangelos P. Markatos, and Sotiris Ioannidis. Gnort: High performance network intrusion detection using graphics processors. In *RAID*, volume 5230 of *Lecture Notes in Computer Science*, pages 116–134. Springer, 2008.