A Streaming Intrusion Detection System for Grid Computing Environments

Matthew Smith, Fabian Schwarzer, Marian Harbach, Thomas Noll, Bernd Freisleben
Department of Mathematics and Computer Science, University of Marburg
Hans-Meerwein-Str. 3, D-35032 Marburg, Germany
Email: {matthew, schwarzer, harbach, noll, freisleb}@informatik.uni-marburg.de

Abstract—**In this paper, a novel architecture for a streaming intrusion detection system for Grid computing environments is presented. Detection mechanisms based on traditional log-files or single host databases are replaced by a streaming database approach. The streaming architecture allows processing of temporal attack data across multiple sites and offers the potential for performance benefits in large scale systems, since data is processed during its natural flow and only stored as long as necessary for analysis. Two cross-site example attacks in a Grid environment and the streaming detection logic for these attacks are presented to illustrate the approach. Experimental results of a prototypical implementation are presented.**

I. Introduction

Although significant effort has been invested in protecting Grid computing infrastructures, it is next to impossible to guarantee absolute security for any software or system with the complexity of today's Grid systems. For this reason, intrusion detection systems (IDS) are an important addition to existing security mechanisms to detect attacks that can not be prevented.

Grid computing environments are continuously growing in size and complexity, crossing organizational borders and including more and more heterogenous resources. The advent of ad-hoc, desktop and personal Grid systems brings a set of resources into the Grid that are not necessarily administered by experts and further increases the number of administrators responsible for the Grid as a whole. This creates several new problems intrusion detection systems must face. The homogeneous Grid layer presents an attacker with a convenient way to do network reconnaissance (scanning for open ports, checking service configurations) without raising traditional IDS alarms on the separate site by enabling the attacker to spread the reconnaissance over several dislocated resources. Furthermore, a single badly administered Grid site may function as a beach-head for further attacks within the Grid infrastructure. For instance, if a node in a desktop Grid also uses password based login (which is common), then a weak password on this node can lead to a legitimate proxy-certificate being stolen and then used to attack other Grid sites. These kind of attacks are difficult to detect without having the big picture. This leads to the need for cross-site IDS data collection and correlation, which in turn leads to huge amounts of data flowing through the Grid IDS. Furthermore, small Grid sites and desktop Grid nodes may not wish to deploy the manpower necessary to adequately monitor their Grid resources, preferring to out-source IDS duties to larger specialised organisations, in particular since attack detection in a Grid environment is a multi-organisational task anyway.

Thus, IDS for Grid environments have to meet demands above and beyond traditional IDS. Firstly, the IDS has to be able to cope with the large amount of network traffic created by multiple Grid sites simultaneously. The IDS must then be able to detect attack patterns that potentially are distributed over multiple sites. Since the multi-organisational data is not required[1] for logging or auditing use, it is not necessary to store the entire data in a traditional database, and data reduction capabilities can be used to decrease the amount of data that needs to be processed. It should also be possible to add new sites to a running intrusion detection system, distribute the load without a lot of configuration work or requiring rule updates. Traditional log-file or database based IDS do not naturally lend themselves to this sort of task, due to their local nature. All data distribution must be hand-coded, and more seriously, data retention and deletion rules must be enforced. Also, the benefit of using traditional IDS structures is minimal, since existing rules can not detect the new distributed attacks without modification.

In this paper, a novel *Stream Intrusion Detection System (S-IDS)* for Grid computing environments is presented. The S-IDS is a first step towards a new form of IDS design tailored to cross-site attack detection for large amounts of transient data. The paper concentrates on network intrusion detection systems (NIDS), but the concepts can also be applied to host intrusion detection systems (HIDS). Since network traffic or other audit data, such as process activity or login attempts, continuously accumulates over time and is only of interest for attack

[1]Due to privacy reasons, it is actually desirable that as little cross-organisational data is stored as possible

detection in a temporal context, handling this data as a stream has the potential of creating a more natural attack detection system, reducing the required resources needed to handle large amounts of data. It also reduces the legal problems inherent with data retention. The presented S-IDS is based on the stream-based database management system PIPES [1]. PIPES's temporal operators facilitate simple definition of detection rules, and its stream-orientation enables a distributed detection infrastructure through the stream aggregation operations. S-IDS needs no permanent data storage to detect attacks, since the attack analysis is done directly in the streaming context. Two cross-site example attacks in a Grid environment and the streaming detection logic for these attacks are presented to illustrate the use of S-IDS. Experimental results of a prototypical implementation are presented.

The paper is organized as follows. Section II briefly describes the functionality of PIPES. Section III present the proposed S-IDS. Experimental results are given in Section IV. Section V discusses related work. Section VI concludes the paper and outlines areas for future research.

II. PIPES

PIPES (Public Infrastructure for Processing and Exploring Streams) [1], [2], [3] is a powerful infrastructure for processing data streams. A data stream is a potentially infinite sequence of data items, such as network or measurement data. PIPES is able to continuously query data streams produced by autonomous data sources. The data streams of interest do not have to be stored in traditional databases for later querying. PIPES stores and uses data only as long as it is of interest for the evaluation process. This is called the temporal context of data. The management of data mainly takes place in main memory, unlike traditional databases management systems that work on data without a temporal context, which necessitates the use of slow external memory like hard disks. This has a huge impact on the performance of systems that use PIPES for stream querying, since data does not have to make a detour via external storage.

A query on streams can be expressed in CQL (Continuous Query Language) [4] that extends SQL by window operators. These operators enable query formulation in a SQL-like SELECT-FROM-WHERE style for streams. Windows are time periods in which a data item is considered for processing. It is important to assign a window to a data item, otherwise it would not be possible to use blocking operations in the query graph, since a stream is potentially infinite.

PIPES uses acyclic, directed operator graphs for processing data streams generated from a CQL statement. Such a query graph is similar to a graph derived from a SQL statement. A PIPES graph consists of three different types of nodes: sources, pipes, and sinks. *Sources*, like a network interface, continuously deliver data that is processed in a pipe-graph. The *pipes* are operators like selects or joins as known from the extended relational algebra. Pipes are both sinks and sources, since they consume and deliver data. The data finally flows into terminal sinks where the final evaluation and presentation is done. PIPES has the ability to attach new sources at runtime, which is required for S-IDS to include new sites.

For the query executing environment, PIPES offers a scheduler, a memory manager, and a query optimiser. Besides using CQL for the construction of queries, PIPES graphs can also be constructed directly by using the Java object representation of sources, pipes, sinks and other components of the PIPES library. While CQL offers a more high level and human readable query definition, S-IDS uses the latter approach, since it offers access and fine control to cutting edge extensions of PIPES that are not yet available in CQL. For a final product, the use of CQL is preferable. However, the Java interface is clear and easy to use. New detection algorithms often require only a few lines of code, connecting PIPES' stream operators. These newly created detection graphs can then easily be added to existing ones.

III. A Novel Streaming Intrusion Detection System

This section describes the architecture of a distributed intrusion detection system based on PIPES. The use of PIPES enables the efficient analysis and recognition abilities of the S-IDS. To not reinvent the wheel, the traditional NIDS Snort [5] is integrated into the S-IDS. This leverages existing single node detection logic into S-IDS, allowing us to concentrate on multi-site threats and the issues involved in coupling multiple sites using a streaming database. The graph-based architecture of PIPES suits the needs of a distributed analysis process. The S-IDS is therefore set up in a tree structure, consisting of sensors on the leaf level, an arbitrary number of aggregators on the node levels and a master node on the root level.

The S-IDS core components are sensor nodes for information production, aggregator nodes to correlate and reduce the data stream and a master node that acts as a final aggregation entity as well as management and monitoring facility. On startup, all sensors and aggregators register with the master to receive configuration data, such as parameters for the detection query graphs, report intervals or neighbors within the infrastructure to connect with.

Figure 1 shows the architecture of the sensor nodes. All sensors are equipped with two network adapters. The first adapter carries the Grid network traffic (1). This network traffic is analysed by Snort, and a raw package capture using LibPCap is used to grab raw header information. If either component recognises an

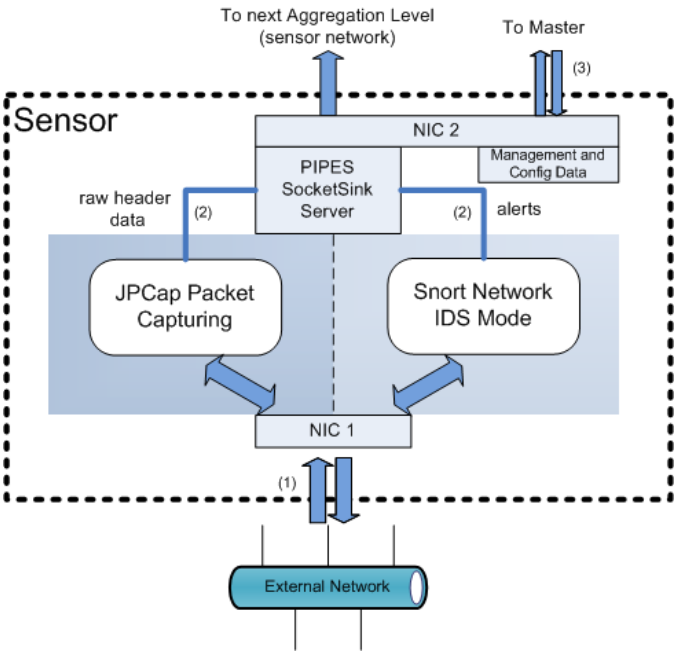

Fig. 1. S-IDS Sensor

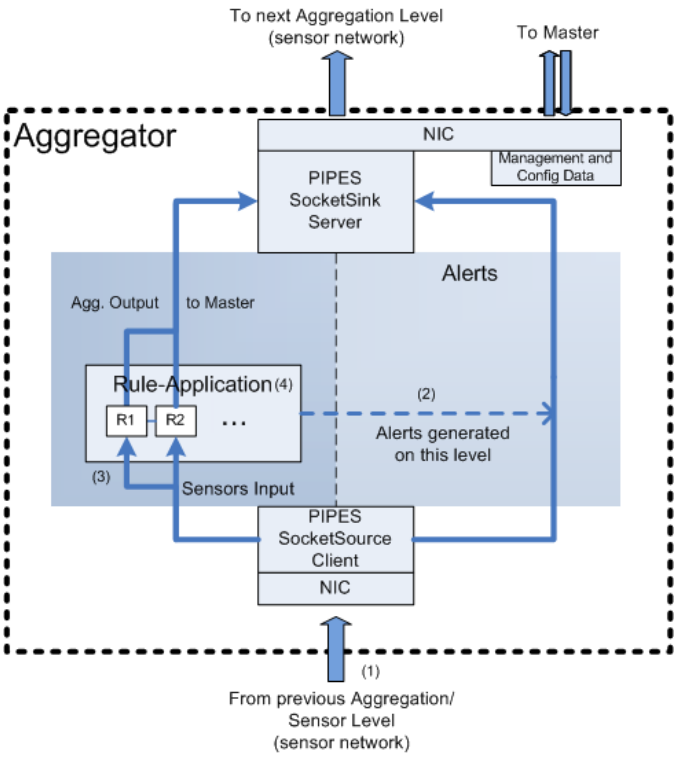

Fig. 2. S-IDS Aggregator

attack, in generates an alert, otherwise only the raw header information is streamed to the next level. Alerts are wrapped into IDMEF alert objects, as specified by RFC 4765 [6] for interoperability reasons. This enables other existing monitoring systems to be easily plugged in as a sensor and at the same time enables S-IDS to be plugged into other security systems that understand IDMEF. Both alerts and the raw header information is then transferred into a PIPES socket sink (2). The second network adapter then streams that information to the next level. The second adapter is also used to receive control information from the master. The second adapter is required since the primary adapter should not be degraded through the NIDS, since this would impact the primary Grid performance. If the Grid node carrying the sensor is under a Denial of Service (DoS) attack, the primary adapter could be overloaded so that NIDS messages would not get trough. For monitoring and load balancing purposes, every sensor periodically reports some system parameters, such as network, CPU and memory usage to the master (3). An arbitrary number of aggregators may subscribe to a sensor to aggregate the gained data in a wider scope.

Figure 2 shows the architecture of an aggregator node. Aggregators receive data from sensors or other aggregators it has subscribed to (1). Alerts received from the previous level are passed directly on to the next level (2). All received raw data from the previous level is analysed, and if an attack is discovered, an alert is generated and passed to the alert channel (3). Depending on the amount of raw packet-header data, the data can be reduced before it is streamed to the next level. The alert detection rules are PIPES subgraphs that process non-alert input data (4). The rule implementation using stream operators benefits from the temporal aspect of PIPES, since no data needs to be stored persistently but is handled as a continuous stream. Querying this stream for information concerning a certain time window is done using PIPES temporal operators. They consume a stream of temporal objects and generate a result stream. All objects in the data stream have a validity interval that starts with the time of their creation. Alerts and aggregated data is then streamed to the next level.

A master Node is responsible both for organising the S-IDS infrastructure as well correlating the data from all top level aggregators. It holds a list of sensors and aggregators ((2)) as well as their status. All alerts generated and data aggregated on lower levels is gathered and processed here. Like the aggregators, the master is capable of running attack detection graphs based on the input from previous levels. The master is the final point in the graph that has an overview of the entire infrastructure. All alerts are then presented to the user.

All graph nodes report their system parameters in regular intervals to the master. This enables the master to detect problems within the NIDS network. For example, if an aggregator fails, the master can assign the affected sensors to a different aggregator or if a sensor stops transmitting its heartbeat, the master can inform the administrator of the site where the sensor went down.

A. Sample Attacks

The use of PIPES allows distributed Grid attacks to be detected in a natural way. Two sample attacks investigated in this paper are briefly described below.

1) Distributed Portscan: The prelude to a specific attack is quite often a portscan to determine which ports are accessible. A portscan targeting a single host can easily be detected by Snort or a similar NIDS. In a Grid environment, the situation is more complicated, since many nodes of the Grid often offer the same services on the same ports, so that an attacker does not have to scan ports on a single node. The attack is cloaked by running partial scans on multiple targets using the homogeneity of the Grid layer to get a complete picture. By aggregating and analysing all network data in a Grid, a distributed portscan can easily be detected by the S-IDS. A small attack script periodically scans 4 ports on each Grid node, so that Snort with a standard configuration does not recognise the portscan.

2) Grid DoS: The second sample attack is a distributed denial of service attack (DDoS). Characteristic for a DDoS is that a large number of entities target a single entity with large amounts of traffic. If the number of requests exceeds the number of requests an entity can handle, the service breaks down. This is true for all distributed environments, but there are two special issues in Grid computing. In a service-oriented Grid, a single master node may receive data from many worker services simultaneously which could be misinterpreted as an attack by a single node IDS. S-IDS tracks which Grid nodes contact which other Grid nodes and automatically creates a temporary white list with which this false detection can be avoided. It could also be used to offer a certain quality of service during a real DDoS by informing a border router which packets are more likely to be legitimate (i.e. packets from nodes recently contacted). While this would not allow new nodes to contact the attacked resource directly, it would prevent nodes returning requested results from being blocked. Since Grid systems perform delegation, it is important that the decision which nodes to block is made with a full view of the Grid and not purely on local information.

B. Custom Pipes Queries for Sample Attacks

Figure 3 shows a detailed view of the PIPES based aggregation and detection logic with two active rules: distributed portscan and DDoS detection. An aggregator receives objects via multiple streams from the *SocketSource* (*(1)*). There are two streams arriving from each child attached to an aggregator, identified by a sourceID. One stream contains alert objects and the other stream contains raw data (e.g. the packet header data). The streams are grouped by their sourceID and passed from the *SocketSource* into an initial *Union*, which unifies the incoming streams (*(2)*) for rule application/alert forwarding. Their incoming information is then passed

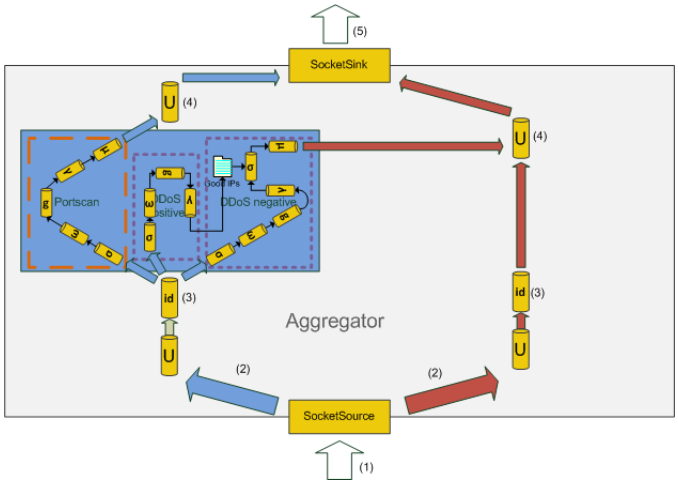

Fig. 3. Aggregator PIPES Queries

through an *IdentityPipe* (*(3)*). This is necessary because of PIPES ONC-mechanism: during runtime, new child nodes can be attached and therefore the union has to be recreated. Without the identity pipe as connection point for downstream rules, PIPES would assume the query graph was closed and as a result shutdown the rule graphs. The red (dark grey) arrows in Figure 3 denote the flow of alert objects, the light blue (light grey) ones denote the flow for non-alert objects. The blue (dark grey) box contains the graph parts for the detection rules based on non-alert information. In the upper part of the figure are two unions (*(4)*), unifying the corresponding output of all alert and non-alert analysis parts before piping them to the *SocketSink* (*(5)*).

The distributed TCP portscan detection rule is shown in the left dashed box in Figure 3. First, the *TemporalFilter* (σ) filters all objects that do not represent TCP packets. Then, a *TemporalWindow* (ω) operator is applied to the data stream, setting the validity of objects to a configurable window size, for instance 5 seconds. This operator is followed by a *TimeGranularity* (g) operator that rounds the validity intervals of the stream objects to a configurable base, for instance to the full second. This is necessary for the following aggregation operator, since the granularity of validity timestamps controls the intervals in which aggregation results are propagated. This mechanism is used to reduce the number of messages that need to be processed. Next, the *TemporalGrouperAndAggregator* (γ) groups relevant TCP packets by their incoming IP and port number before counting the number of packets per time window. Finally, the aggregation output is packaged for shipping, using a *TemporalMapper* operator (μ) using the source's IP address as the key.

The DDoS attack detection rule is shown in the middle and right dashed boxes in Figure 3. Unlike a

traditional single node solution, simple SYN counting or traffic volume measurement is not sufficient in a Grid scenario. In a service-oriented Grid, a single node will request results from many other nodes and the answers depending on the volume could be misidentified as a DDoS attack. To prevent this kind of false positive, the detection query must take Grid job submission into account to filter out wanted from unwanted traffic. For this reason, the DDoS rule is divided into two parts. The smaller box on the left detects job submission patterns and adds matching senders to a list of IPs that may send large amounts of traffic from the monitored nodes. Employing the same approach as with the distributed portscans query shown above, the input object stream is filtered for TCP packets, then *TimeGranularity* and *TemporalWindow* operators are applied. In combination with a *TemporalGrouperAndAggregator*, grouping by source IP and counting distinct destination IPs, every IP that contacts a number of distinct nodes in a certain time window at a given number of Grid specific ports is added to an IP white list. Here, the temporal aspect of PIPES offers a natural advantage since the white list automatically expires at the end of the time windows. The second part of the DDoS detection query shown on the right counts the connections to the same hosts. A *TemporalFilter* checks if a corresponding entry exists in the IP white list for the corresponding host and raises an alert if the count reaches a certain threshold of non white listed connections. The alert is then piped into the alerts channel.

IV. Experimental Results

The S-IDS system was implemented using the PIPES library [1], the JPcap library [7] and Snort [5]. Snort is used to leverage existing single node NIDS capability into the S-IDS. JPcap provides an interface to capture raw network traffic in Java. It was used as the main sensor information source for the PIPES detection queries. PIPES was used to connect the different sensor sites and perform distributed temporal queries on the sensors.

In a single host environment, an IDS only needs to detect attacks targeted at that host. The presented S-IDS is set up to detect attacks that are not visible on a single host but need a global view to be detected. The proof-of-concept implementation was geared towards two Grid specific attacks that are not detected by a single host IDS, but they are correctly detected by S-IDS, through its ability to aggregate data from more than one node. The first attack is a distributed portscan over multiple hosts, the second is a DDoS attack that can be differentiated from legitimate traffic where many Grid nodes respond to a single entity. All test nodes were running on 3 GHz, 1 GB RAM, AMD64 machines running Debian 4.1.1-21. The nodes were equipped with two separate Gigabit Ethernet NICs. During all tests, random traffic was generated between the nodes to simulate an active Grid network. The network environment of nodes used for testing was based on a 1 GBase-T Ethernet network with a theoretical throughput of 125 MB/s. An unthrottled data transfer of */dev/zero* between two nodes reached a maximum real throughput of 117 MB/s. In actual application scenarios, like network file transfers or Grid job submissions, the employed transfer protocols limit the achievable transfer speed, e.g. due to encryption. Testing showed an average throughput of approx. 42 MB/s for a secure file transfer using the command-line tool *scp* that encrypts data with *SSL*. To get an idea how much traffic should be expected from a Grid site, the maximum throughput between two Grid sites was measured. Figure 4 shows the speed distribution between the Grid sites of the University of Marburg and the University of Siegen in Germany, which are connected by the X-Win network of the German research network association (DFN). One measurement was done during the day, two others during the night. Each test was run for one hour. The average transmission rates for the different experiments did not vary significantly, with the average speed being roughly 10 MB/s.

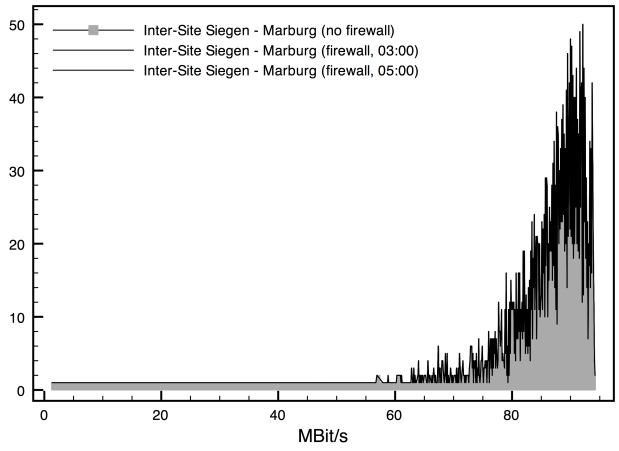

Fig. 4. Inter-Site iperf Results

The performance evaluation of the S-IDS design was conducted using two experiments. The first test consisted of inducing high network traffic on *eth0*, the interface to the external network, of the sensor nodes, while monitoring the output on *eth1*, the interface to the S-IDS network, CPU usage of the sensor and the alert latency (time from the attack to its recognition at the master). In the second test, the behaviour of an aggregator with five connected sensors was observed. The *eth0*/*eth1* traffic of the sensors was monitored and the *eth0*/*eth1*/CPU usage and alert latency of the aggregator was monitored. Figures 5 and 6 show the results for the two test cases, which are averaged over two minutes.

Network traffic was induced by one node, sending

data produced by */dev/zero* via *netcat* to another node, writing the received data to */dev/null*.

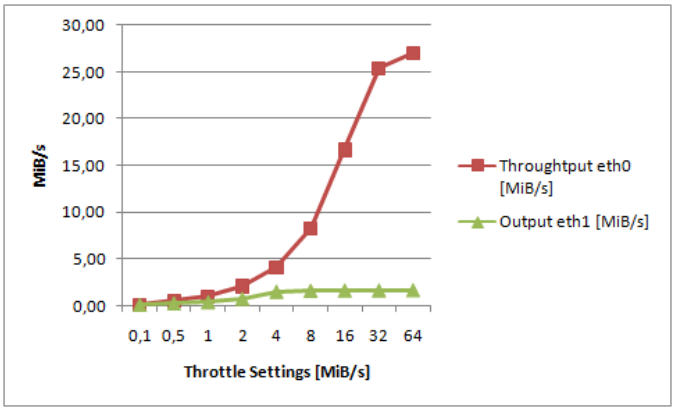

Fig. 5. Graph of Sensor Test Results

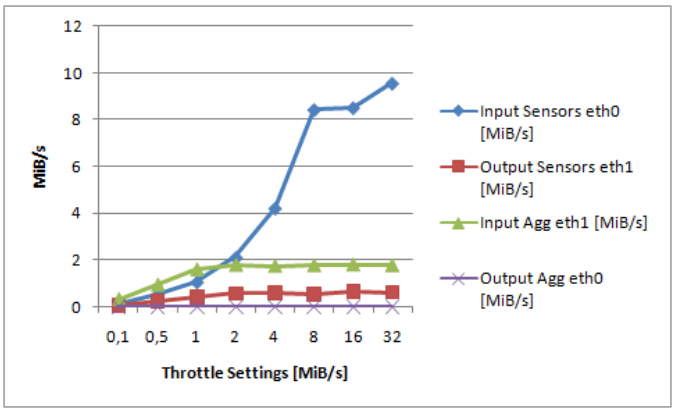

Fig. 6. Graph of Aggregator Test Results

In Figures 5 and 6, it can be seen that at a speed of approximately 1.6 MB/s incoming traffic at the aggregator's *eth1* or approximately 5 MB/s incoming traffic at the sensor's *eth0*, the output speed of the sensor does not increase further. The reason for this behaviour was identified in the second test, where the aggregator was observed: At an input speed of approximately 1.6 MB/s on the aggregator's incoming interface, the aggregator automatically throttles its input speed. This is underlined by a CPU usage of around 100% on the aggregator at this volume of traffic. The reason for this is the slow Java serialisation process.

Based on this poor performance, some opitimisations were implemented. The main problem lies in the fact that the object-oriented Java language used by PIPES creates an object for each piece of data that is streamed through the system. Since serialisation and deserialisation of complex objects is quite expensive, measures were taken to lighten this burden. The AlertOrObjectBean was separated into two entities to eliminate one object hierarchy and the IDMEF objects were replaced by a minimal flat storage object. This step reduces the compatibility, but increases the performance. Figures 7, 8 and 9 show the new performance values. Figure 7 shows one sensor and one aggregator. At a speed of 20 MB/s, the sensor is at 100% CPU utilisation through the netcat, libpcap and Java processes. The aggregator does not throttle the connection at this speed. In Figure 8, two sensors are connected to one aggregator each transmitting up to their maximum capacity, and the aggregator starts throttling the input at roughly 40 MB/s combined traffic. In Figure 9, six sensors are connected to one aggregator. Here, the sensor starts throttling the input at a combined volume of roughly 24 MB/s. This is due to the higher CPU consumption of the detection logic.

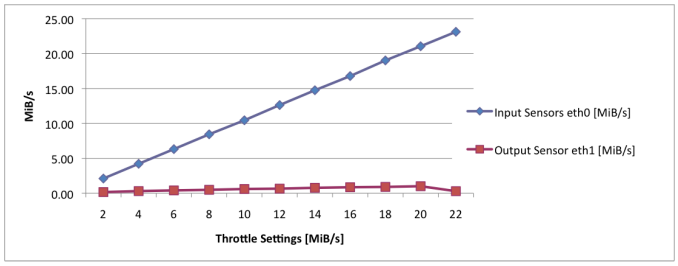

Fig. 7. Optimised S-IDS with One Sensor

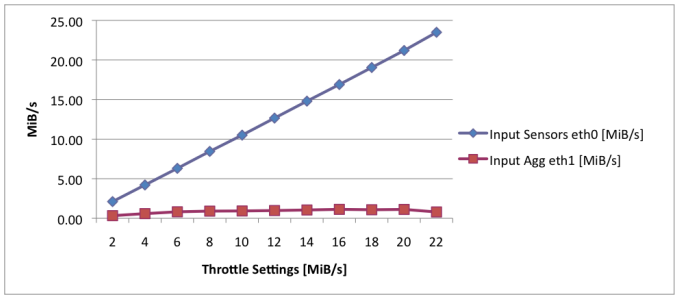

Fig. 8. Optimised S-IDS with Two Sensors

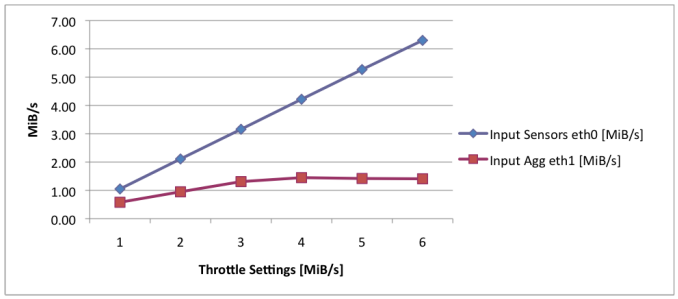

Fig. 9. Optimised S-IDS with Six Sensor

During all tests, attacks were launched at random intervals to test the responsiveness and accuracy of the system. Up to the throttle points, all attacks were detected reliably. The alert latency varied between one and five seconds. Even when the aggregator could not handle

any more incoming traffic, it continued to operate, but attacks were not always detected, due to the fact that not all packets could be analysed.

With the performance optimisation, the S-IDS system can cope with moderate Grid traffic from a small number of nodes. However, beyond that, the performance overhead created by the serialisation and object creation of Java stalls the system. As a consequence, the vision of analysing the "big picture" over a large number of nodes cannot be realised with the presented proof-of-concept implementation. The Java serialisation and streaming configuration is the most relevant area for optimization. Nevertheless, the general concept of using streaming databases for intrusion detection is promising, but a more efficient implementation is required. A production quality implementation of the system in a is an interesting endeavor, but lies outside of the scope of this paper.

V. Related Work

Schulter et al. [8] identify different types of attacks against a Grid infrastructure. Besides conventional attacks against Grids, the paper lists several Grid specific attack types: unauthorised access, misuse (by authorised users) and exploits (attacks towards services, protocols or Grid applications). The authors point out that present Grid-IDS approaches do not fulfil the important qualities (completeness - recognition of all attack-types, scalability and Grid-compatibility) for protection of Grid systems. Their proposal is as follows: To prevent costly development of new software, the involved Grid nodes are equipped with conventional Network and Host IDS (N- and HIDS). These are able to recognise and report standard attacks and furthermore send their data to a Grid IDS, which may additionally identify Grid-specific attacks. The Grid IDS consists of a scheduler, agents, analysers and databases. Data from N- and HIDS is collected into databases, distributed and than analysed on Grid nodes. While some focus is given to the global collection of IDS data, no cross-site analysis is done and thus no new distributed attacks can be discovered. All data is stored locally and analysed in the traditional way.

Fang-Yie Leu et al. [9] propose an IDS concept that employs a Grid for intrusion detection. The monitored system does not have to be a Grid. The main design goal of the IDS was that it should be particularly robust against high amounts of data traffic, as it occurs with DoS attacks. The concept of the Grid IDS is as follows: a dispatcher collects network traffic from a switch using tcpdump and dispatches the collected data periodically to a detection node. A scheduler controls which dispatcher sends data to which detection node to achieve an equal load in the entire system. To be able to detect attacks with a longer time scope, data analysed by detection nodes is stored in a database and analysed periodically by a detector. In an extension [10] to the original system backup brokers are introduced which make the system robust against malfunctions of detection nodes. Although trace data is distributed, the system does not correlate the data gained from different network segments and thus it is not able to detect attacks that take advantage of the distributed nature of the Grid.

Yonggang et al. [11] propose a tree-like structuring of an IDS for distributed networks (LDIDS). The leafs of the tree consist of conventional IDSs that generate alerts in a first step. The nodes on the central levels of the LDIDS tree aggregate alerts as well as *"data, which cannot be analysed locally"* sent by child nodes. Finally, the root node has an overview of the entire network to be monitored. The paper does not mention what kind of data is shared between nodes besides alerts, if this additional data is only sent in special cases or how it is later analysed. Although this paper employs a similar structuring as used by the S-IDS, it does not deal with the problem of handling the large amounts of data or detection of distributed attacks.

Kenny and Coghlan [12] describe a system that allows the querying of log files through the Relational Grid Monitoring Architecture (R-GMA), which can be used to build a Grid-wide intrusion detection system. Their implementation of this system, called SANTA-G (Grid-enabled System Area Networks Trace Analysis), queries Snort log files by using SQL. SANTA-G is composed of three elements: A Sensor, a QueryEngine and a Viewer GUI. The log files are monitored by the sensor, which, in case of a change to the log files, inform the SANTA-G QueryEngine. The QueryEngine then publishes the relevant information to the SANTA-G Viewer to show them in a GUI. In future implementations, a high-level incident detection, tracking and response platform is planned which will use custom coded Consumers to filter and analyse the alerts published in order to detect patterns that would signify an attempted distributed attack on the Grid infrastructure. Currently, SANTA-G does not provide a functionality that detects distributed attacks. It allows an easy access to Snort log files, but does not correlate between the different nodes. Additionally, information is queried by SQL when changes on the log files occur, no other data is monitored.

Choon and Samsudin [13] propose an architectural concept for a Grid-IDS. They define a number of requirements for an effective Grid-IDS: flexibility, scalability, reusability, small overhead, speed, autonomy and adaption to Grid environments. They also suggest that an IDS virtual organisation should be used to offer IDS services to the rest of the Grid. News of attacks and importantly the vulnerabilities that lead to the attacks can then be distributed via the VO to help protect the other Grid site.

Silva et al. [14] describe a system named "Distributed IDS on Grid" (DIDSoG), that aims to join heterogeneous Intrusion Detection System over a Grid middleware. This should be achieved by a two dimensional hier-

archy of Sensors, Correlators/Aggregators, Analysers, Monitoring services and Countermeasure services. The infrastructure should be used to combine the strengths and reduce the weaknesses of different existing IDS systems. However, no IDS or Grid systems are used, rather a GridSim simulation is presented covering the graph construction, thus it is difficult to judge the capabilities of the system. It is, however, clear that distributed attacks cannot be detected, since the monitoring components consist of standard IDS system and no raw data is exchanged.

Feng et al. [15] argue that traditional Host-IDS systems are not suitable for Grid systems, since certain information such as Grid user to local user mappings are not known on the local node. Furthermore, it is argued that since the Grid requires its computational power for its users, traditional H-IDS systems are to heavyweight. A Grid H-IDS is proposed specifically targeting Grid specific attacks on a single nodes. While alerts are passed on to the Grid level, the IDS logic is confined to a single host.

To summarise, all of the above mentioned work use traditional logging and querying to detect intrusions. The presented S-IDS's utilisation of the stream based database system PIPES enables a simple and flexible implementation of information flow and analysis, employing the temporal context of data, which is not possible using a standard IDS approach.

VI. Conclusions

In this paper, a novel architecture for a streaming intrusion detection system (S-IDS) for distributed environments was presented. A stream-oriented database management system was used for the querying of network data. It use facilitates a natural handling of data analysed within the S-IDS. The streaming architecture allows processing of temporal attack data across multiple sites and offers the potential for performance benefits in large scale systems, since data is processed during its natural flow and only stored as long as necessary for analysis. Two cross-site example attacks in a Grid environment and the streaming detection logic for these attacks were presented to illustrate the approach. Experimental results of a prototypical implementation were discussed.

There are several areas of future work. While the concept of using stream queries for intrusion detection has been validated, the performance issues are a major area of future work. Apart from improving the performance and scalability of the system there are three further areas of particular interest. The extension of the CQL syntax to include network pipes will enable distributed attack detection queries to be formulated using the CQL language, instead of requiring the PIPES to be constructed using Java. The use of CQL is desirable for a number of reasons: a) CQL statements are shorter and thus easier to read and understand than Java programs, b) a CQL compiler can perform optimisations which cannot be automated on Java constructed PIPES queries and c) CQL statements are easier to exchange and integrate into new environments making the detection rules more portable. Once all relevant PIPES operations for the IDS can be modelled in CQL, some thought can be given to extending the number of attacks which can be detected by PIPES. Also, an automated mapping of rules from other IDS systems such as SNORT to S-IDS CQL rules might be possible. Finally, an anomaly detection framework could be integrated into S-IDS, since the large number of Grid nodes gives a good base line indicator and comparative evaluation can easily be formulated in PIPES.

References

[1] Arbeitsgruppe Datenbanksysteme, "PIPES Project Homepage," http://dbs.mathematik.uni-marburg.de/Home/Research/-Projects/PIPES.

[2] J. Krämer and B. Seeger, "PIPES – A Public Infrastructure for Processing and Exploring Streams," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, 2004, pp. 925–926.

[3] ——, "A Temporal Foundation for Continuous Queries over Data Streams," in *Proceedings of the International Conference on Management of Data*, 2005, pp. 70–82.

[4] A. Arasu, S. Babu, and J. Widom, "CQL: A Language for Continuous Queries over Streams and Relations," in *International Workshop on Database Programming Languages*. Springer, 2003, pp. 1–19.

[5] Snort Development Team, "Snort Network Intrusion Detection," http://www.snort.org, February 2007.

[6] H. Debar, D. Curry, and B. Feinstein, "The Intrusion Detection Message Exchange Format (IDMEF)," RFC 4766, pp. 1–23, 2007.

[7] K. Fujii, "JPcap," http://netresearch.ics.uci.edu/kfujii/jpcap/-doc/index.html, 2008.

[8] A. Schulter, F. Navarro, F. Koch, and C. Westphall, "Towards Grid-based Intrusion Detection," in *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, 2006, pp. 1–4.

[9] F. Y. Leu, J. C. Lin, M. C. Li, C. T. Yang, and P.-C. Shih, "Integrating Grid with Intrusion Detection," in *19th International Conference on Advanced Information Networking and Applications, 2005*, 2005, pp. 304–309.

[10] F. Y. Leu, M. C. Li, and J. C. Lin, "Intrusion Detection based on Grid," in *ICCGI '06: Proceedings of the International Multi-Conference on Computing in the Global Information Technology*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 62–62.

[11] Y. Chu, J. Li, and Y. Yang, "The Architecture of the Large-scale Distributed Intrusion Detection System," in *PDCAT '05: Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 130–133.

[12] S. Kenny and B. Coghlan, "Towards a Grid-Wide Intrusion Detection System," in *Advances in Grid Computing*. Springer, 2005, pp. 275–284.

[13] O. T. Choon and A. Samsudin, "Grid-based Intrusion Detection System," in *Proceedings 9th Asia-Pacific Conference of Communications, vol. 3*, 2003, pp. 1028–1032.

[14] P. F. Silva, C. B. Westphall, C. M. Westphall, and a. Marcos D. Assunç "Composition of a DIDS by Integrating Heterogeneous IDSs on Grids," in *MCG '06: Proceedings of the 4th International Workshop on Middleware for Grid Computing*. New York, NY, USA: ACM, 2006, pp. 12–12.

[15] G. Feng, X. Dong, W. Liu, Y. Chu, and J. Li, "GHIDS: Defending Computational Grids against Misusing of Shared Resources," in *APSCC '06: Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing*. IEEE Computer Society, 2006, pp. 526–533.