Enhancing Interoperability and Stateful Analysis of Cooperative Network Intrusion Detection Systems

Michele Colajanni
University of Modena and Reggio Emilia
michele.colajanni@unimo.it

Daniele Gozzi
University of Modena and Reggio Emilia
daniele.gozzi@unimo.it

Mirco Marchetti
University of Modena and Reggio Emilia
mirco.marchetti@unimo.it

ABSTRACT

A traditional Network Intrusion Detection System (NIDS) is based on a centralized architecture that does not satisfy the needs of most modern network infrastructures characterized by high traffic volumes and complex topologies. The problem of decentralized NIDS based on multiple sensors is that each of them gets just a partial view of the network traffic and this prevents a stateful and fully reliable traffic analysis. We propose a novel cooperation mechanism that addresses the previous issues through an innovative state management and state migration framework. It allows multiple decentralized sensors to share their internal state, thus accomplishing innovative and powerful traffic analysis. The advanced functionalities and performance of the proposed cooperative framework for network intrusion detection systems are demonstrated through a fully operative prototype.

Categories and Subject Descriptors

C.2.0 [**COMPUTER-COMMUNICATION NETWORKS**]: General—Security and protection; C.2.3 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Operations—Network monitoring

General Terms

Design, Experimentation, Security

Keywords

Network intrusion detection systems, Traffic analysis, Distributed architectures, State migration

1. INTRODUCTION

A *Network Intrusion Detection System* (NIDS) that analyzes network traffic by looking for evidences of illicit activities is a valuable component in all modern network security infrastructures. While the first generation of *stateless* NIDS is vulnerable to a multitude of insertion and evasion attacks [13], the most recent NIDS architectures perform a *stateful* traffic analysis that cannot be easily eluded.

Stateful traffic analysis comes at the price of an increased computational complexity that may prevent a single NIDS sensor to perform a sound, stateful analysis on all the traffic flowing on high speed network links. Moreover, stateful analysis relies on the management of large quantities of contextual data (the so called *state*) that pose additional burden on NIDS sensors in terms of computational load and memory usage [6, 14]. As a recent tendency, to allow the analysis of high traffic volumes and to control networks with complex topologies, a common approach is to pass from a centralized NIDS solution to parallel or distributed NIDS architectures [8, 15, 19, 4] consisting of multiple *cooperative* sensors. A distributed analysis is not a problem when it is stateless, but it is still an open problem to apply stateful algorithms when the traffic is spread among multiple sensors. In a similar context, each distributed or parallel sensor is reached only by a subset of the overall traffic, and this partial view prevents the possibility of a fully reliable analysis.

In this paper we aim to enable cooperative sensors to exchange valuable state information, thus allowing a distributed or parallel NIDS to perform a stateful and fully reliable analysis. To this purpose, we present an innovative state management and migration framework that makes multiple NIDS sensors able to exchange internal state information. Moreover, we integrate the proposed framework into an operative prototype that is based on the open source software Snort. (Snort represents the most widespread NIDS software as well as the de-facto standard in the field of signature-based traffic analysis [17, 3].)

The proposed state migration framework propagates internal state information among cooperative NIDS sensors with the goal of providing each sensor with all the information that is necessary to perform a stateful traffic analysis. To the best of our knowledge, this is the first NIDS cooperation mechanism that allows a true exchange of structured internal state information and a consequent execution of advanced traffic analysis algorithms, such as packet content inspection. We remark the necessity and the novelty of a real stateful and reliable traffic analysis, because existing parallel and distributed NIDS architecture do not fully comply to these requirements independently of their claims. We will return on this issue in Sections 2 and 7.

The rest of this paper is organized as follows. Section 2 reviews the concept of internal state for each of the three

ANCS'07, December 3–4, 2007, Orlando, Florida, USA.

main schemes employed in traffic analysis. Section 3 evidences the most relevant issues and requirements that are related to state migration, and contains a detailed design of the proposed state migration framework. Section 4 describes the internal modifications to enable the Snort software to migrate its state, thus obtaining a fully operative NIDS prototype. Section 5 demonstrates the efficacy and functional validity of the proposed framework. Section 6 presents several experimental results that measure the performance of the prototype implementation. Section 7 compares our work with other results in the field of distributed and parallel NIDS cooperation. Finally, Section 8 summarizes main results and outlines future work.

2. STATEFUL ANALYSIS

We classify the network traffic analysis procedures that can be carried out by existing NIDS systems in three major sets:

- **Statistical anomaly analysis**. This type of analysis is based on the computation of several statistical metrics on large packet sets. Intrusions are detected when the measured statistics diverge from the traffic model that is considered normal and benign. Examples of common attacks that are detectable by this method are ping sweeps, port scans, SYN floods, worm propagation attempts.
- **Signature pattern matching**. This method is the most common detection system that is employed to detect known attack patterns by inspecting the packet payloads. A stateful NIDS that is based on signature pattern matching comes with a custom language used for signature definition. It looks for known attack signatures in analyzed network packets, where a signature is represented by a sequence of bytes that characterize one or more intrusions. The underlying concept is that it is (ideally) impossible to perform a certain attack without sending to the victim host a sequence of network packets including the signature. This method can effectively detect remote buffer overflow attacks (for example, by intercepting the NOOP sleds or some common shellcode variants), remote exploits, and all other intrusions that can be characterized by a known signature.
- **Protocol anomaly detection**. The idea is that it is possible to identify anomalies in a data stream behavior by describing all protocol details through a Finite State Machine. A stateful inspection of that protocol may reveal attempts to divert from the legitimate usage patterns, that are interpreted as attempted attacks. Brute force cracking and FTP bounce attack are typical examples of illicit activities that rely on application protocol misuse and that are detectable through a protocol anomaly detection.

The internal state of a NIDS is a representation of the network traffic features that are relevant to the specific detection algorithms. Statistical anomaly analysis and protocol anomaly detection algorithms intrinsically rely on information stored during previous detection activities. Indeed, both the value of an aggregated, statistical feature and the current state in a protocol FSM include information related to the traffic analyzed in the past. On the other hand, it is possible to perform a *stateless* signature pattern matching by analyzing each packet on its own, without any information related to previous network activities. The problem is that stateless signature pattern matching can be easily evaded through well known techniques, such as the fragmentation of the attack signature in different network packets [13]. When a signature is fragmented, there is not a single network packet containing all the information necessary to detect the intrusion attempt. Hence, a NIDS sensor analyzing each packet separately is not able to identify a fragmented attack.

Table 1: State information classification

Type of analysis	State data profile
Statistical anomaly analysis	Aggregated statistical information related to all the network traffic that has been previously collected on the monitored network segment.
Signature pattern matching	Reordered and reassembled information streams built from previously collected network packets. A separate data stream should be stored for each currently active connection.
Protocol anomaly detection	Protocol FSM state for all active connections.

To effectively combat evasion techniques, all modern pattern matching NIDS sensors aim to perform a stateful traffic analysis. This requires that all received packets are temporarily stored, reordered and reassembled. Stream reassembly allows a NIDS sensor to inspect a complete and ordered information stream, in which all the possible attack signatures can be easily detected.

From the above analysis, it emerges that all the intrusion detection approaches rely on some sort of internal state information that reflects the current state of the network traffic. However, different detection algorithms adopt different logical abstractions for the generation of heterogeneous state information. Table 1 summarizes the internal state representation for each of the three methods.

The generation and maintenance of a complete internal state is commonly carried out by traditional centralized NIDS where one sensor analyzes all the network traffic flowing through a monitored link. However, when the analyzed traffic is processed by multiple sensors operating on the same link (*parallel NIDS*) or on different links (*distributed NIDS*), the global network state is fragmented because each sensor has only a limited view of the global state.

In these contexts, it is impossible to detect attacks that are perpetrated through traffic portions analyzed by different sensors. Our proposal addresses this issue through a novel sensor cooperation mechanism that supports migration of relevant state information from one sensor to another. Such framework enhances detection accuracy and allows us to deploy NIDS sensors among any kind of network topology.

3. STATE MIGRATION

3.1 Main requirements

The aim of the proposed framework is to enable cooperative sensors to exchange valuable state information, thus enabling a distributed and/or parallel NIDS to detect illicit activities that would not be discovered otherwise. This scenario poses several new challenges, that we addressed through the careful design choices outlined below.

- It is important for NIDS sensors to perform traffic analysis in real-time, hence the impact of state migration operation on the sensors has to be as low as possible. Packet losses due to state migration overhead are not acceptable.
- Both state import and export operations have to keep consistent the internal state of sensors involved in cooperation. These operations should not alter the analysis reliability and cause residual dependencies on imported state information.
- State migration has to be robust to transmission delays. Moreover, it must be possible to perform state migration even out of order with respect to the contextual network traffic.
- The framework should be easily adaptable to various applications, hence it must provide state migration mechanisms without imposing limitations about state management, dispatching policies and initiation schemes.
- To achieve high performance and adaptability, it should be possible to export and import only partial state information (e.g., all the information related to a particular host or transport level connection).
- It should be easy to extend the communication protocol for the transmission of external state representations, even with the goal of allowing the migration of new state information among heterogeneous network components.

A similar architecture can address scalability issues related to the analysis of large amounts of traffic flowing through high capacity networks and to the inspection of packets flowing through complex network topologies consisting of multiple segments.

In network-related contexts, state migration has been successfully employed to realize level 4 [5] and level 7 [1] front-end switches for Web clusters [2]. In this case, the transmitted state information consists of a description of a TCP connection current state. TCP state migration is usually carried out through a custom migration protocol, that usually requires the exchange of one network packet, and where migration activities are initiated by the Web switch. The NIDS state migration proposed in this paper is by far more complex than a TCP connection migration. The proposal in some sense is more similar to process migration [10] schemes that have been extensively analyzed in distributed (operating) systems and mobile agents [11].

Another proposal for NIDS sensor cooperation [16] represents a preliminary and partial solution that is related to a specific NIDS software (see Section 7 for further details).

3.2 NIDS state migration framework

A typical NIDS [12] consists of three main components[1]:

- A packet input layer feeds the NIDS sensors with the data to be analyzed. Following the CIDF architecture, we denote this layer as the *Event generator* or E-box.
- One or multiple detection engines perform the network traffic analysis (this component corresponds to the CIDF *Event analyzer* or A-box).
- An output layer handles alerting, logging and packet dumping (that is, the CIDF *Event database* or D-box).

Cooperation at the packet input layer is impossible, because input data depend on network setup and its contents are independent variables. On the other hand, cooperation at the output layer is a well established practice, with many products allowing alert aggregation and correlation (e.g., Prelude).

However, NIDS cooperation becomes a real challenge at the detection layer, especially when real-time intrusion analysis is necessary. Transferring state data between stateful detection engines permits a great improvement of the analysis accuracy. In particular, this kind of cooperation allows a detection engine to discover an intrusion by importing external state data that are generated by other detection engines, thus detecting activities that could not be discovered otherwise.

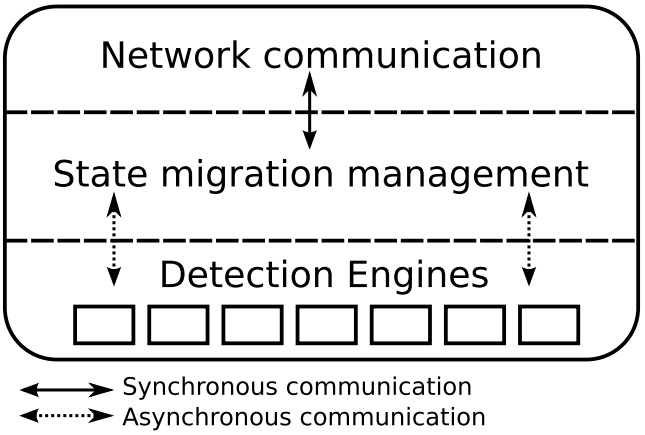

Figure 1: NIDS state migration framework

The proposed state migration framework consists of three main components that are shown in Figure 1:

- The network communication layer transfers state data through the network.
- The state migration management layer interprets cooperative sensors communications.
- The detection engines layer feeds each engine with imported data and extracts states.

[1]We omit the fourth component of the CIDF specification, Response units ("R-boxes"), because intrusion detection does not necessary imply a reaction to hostile activities. R-boxes are supposed to *kill processes, reset connections, alter file permissions, etc.*

The network communication layer of the cooperative sensors acts as a Remote Procedure Call (RPC) server that listens for remote procedure calls. Cooperative sensors or heterogeneous external elements, such as intermediate components for state migration management, interact easily with the state migration enabled NIDS sensors by calling state *import* or *export* remote procedures. This design choice has several advantages:

- NIDS sensors are not required to be aware of the cooperative sensors logic and of the message distribution policy.
- Each NIDS is always ready to perform required import and export operations at run-time, without any need for ad-hoc configurations.
- State export and import are implemented as a one-to-one communication, hence the computational overhead related to a state exchange operation is independent of the number of cooperative sensors.
- Values returned by remote procedures allow the caller to handle possible import and export errors.

The proposed framework does not pose any limitation on the deployment of the cooperative elements. However, performance and security issues suggest to rely on a dedicated network for sensors management, rather than transmitting internal states and other control information through the same link of the monitored traffic.

The state migration management layer handles the state import and export requests received by the network communication layer and interacts with the detection engine interfaces.

To ensure maximum flexibility, the state migration framework should allow the transfer of only parts of a detection engine state data. To this purpose, we must characterize the network traffic part which is mapped in a certain block of the state data structure. Our choice is to follow the same criteria that are adopted by the detection engine. Detection performed on protocols belonging to the OSI layer 3 (network level) are discriminated on the basis of the network addresses. Layer 4 (transport level) and higher layer detections are discriminated on the basis of transport level streams that are identified by transport level protocols, source network address, source port, destination network address and destination port. As the transport level identifiers are a superset of those needed by the network level, they can be easily integrated into a single structure type.

To guarantee adequate performance to the detection mechanism, the state migration management layer should be decoupled from the detection engines layer. However, it is essential to use a synchronization construct to lock the detection engines internal state representation, thus allowing to perform state import and export operation on a consistent internal state. Concurrent detection may be slowed because of the state locking, hence the locking duration has to be sufficiently short to prevent network packets loss. Experimental results, described in Section 6 demonstrate that this constraint can be easily satisfied by our reference implementation.

Our framework provides cooperative sensors with all the mechanisms required for state exchange. Moreover, to ensure the highest flexibility, the sensor mechanisms are completely decoupled from the state migration policies that depend on the specific deployment scheme.

To describe by example how state information can be exchanged among cooperative NIDS sensors, we refer to a realistic deployment scheme that is composed by three sub-networks as in Figure 2. Following the typical distributed intrusion detection approach, we deploy a NIDS sensor in each of the sub-networks. Moreover, a centralized element called *coordinator* is used to manage and control state migration activities.

A complete state migration transaction is performed in two logical steps: *state export* and *state import*. In the state export phase, the coordinator sends to one of the cooperative sensors a request for the generation of the external state representation that is related, for example, to a particular host. The sensor finds all the relevant information in its internal state and generates a reply to the coordinator containing an external representation of the requested state information. The state import phase starts when the coordinator sends to a cooperative sensor a set of external state representations (previously generated by other sensors) that should be imported. The destination NIDS sensor merges the external state representation with its internal state information, thus increasing its ability to detect illicit network activities.

Although the scheme shown in Figure 2 refers to a distributed NIDS architecture, we recall that the proposed framework is highly modular and adaptable to heterogeneous applications and implementations. For example, the same framework can be utilized to create a parallel NIDS architecture, where load balancing adjustments of network traffic are followed by the transfer of detection state data [4]. In this parallel context, the coordinator element can be easily embedded in the parallel NIDS load balancer.

Another example may refer to the case of wireless mobile networks, where state migration can be effectively used to force an intrusion detection state to *follow* a mobile user. In this context, state migration has to be performed whenever the mobile host changes its access point, as a part of the handover procedure.

Purely distributed deployment schemes without a central coordination element can also be created by embedding a coordinator in every cooperative NIDS sensor. A similar scheme would provide each sensor with an effective way to retrieve the state information deemed as necessary by the analysis logic. As an example, a sensor that receives only part of a connection could ask for other relevant state information to cooperative sensors before analyzing an incomplete stream.

Finally, state migration can be effectively used to transfer all the state information from one sensor to a new sensor that has just been installed on the same link. This possibility enables the substitution of NIDS sensors without interruptions in stateful traffic analysis. The details of the proposed architectures are described in the next section.

4. STATE MIGRATION ARCHITECTURE

To demonstrate the viability of our proposal, we realize a fully operative implementation of the described state migration framework. To this purpose, we extend the source code of the open source NIDS Snort. We chose this particular software for several reasons: it is widely adopted (almost a *de facto* standard), hence the proposed framework can be

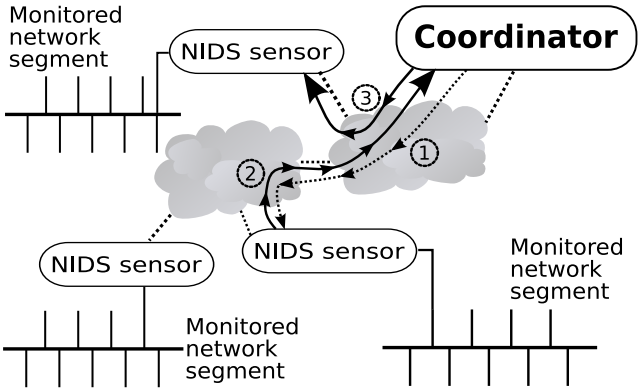

Figure 2: Example deployment scheme

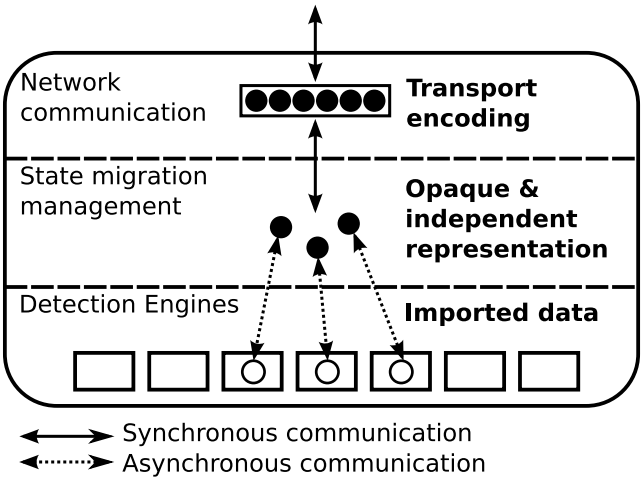

Figure 3: State representation in different framework layers during migration.

used in real world environments; Snort allows us to measure the performance in a well known environment where inner working, performance and computational requirements have been extensively analyzed [17, 3]; Snort employs pattern matching analysis and may require the exchange of possibly complex state information (nothing similar has been achieved by other cooperation architectures). While the described implementation is tailored to a specific software, we remark the general applicability of the proposed framework.

Snort architecture is split across several layers: data flows from the capture interface through the decoding and preprocessing stage; then, they are analyzed by the detection engine and, when necessary, the output layer emits alerts.

Each layer consists of many plugins. Most plugins perform stateful analysis for a specific protocol, hence their state information may be shared with analogous plugins to achieve the cooperation benefits. Each detection engine utilizes a state data type whose contents depend on its analysis policy.

The core detection engine consists of an optimized pattern matcher which applies the rules described in a set of configuration files. While pattern matching is hardly stateful, most of the Snort pre-processors perform the stateful analysis of a specific protocol.

The design of a state migration framework in Snort can take advantage of its modular architecture. We require each pre-processing plugin to be able to export its own state data and import external state data. In our reference implementation, we implemented the necessary state migration capabilities into the `stream4` pre-processor because it has a general purpose and its operations influence heavily the rule-based detection. Indeed, stream4 is used to generate an ordered data stream out of disordered, fragmented and overlapping network packets belonging to the same level 4 connection, thus enabling stateful pattern matching of TCP connections and UDP streams. Allowing the stream4 state to be transferred between Snort instances greatly enhances detection efficacy in a cooperative NIDS environment, because most of the available Snort rules require to inspect the contents of a packet stream, thus relying on TCP and UDP stream reassembly and tracking.

We remark that our framework is specifically designed for real-time traffic analysis, hence one of the main design issues is to avoid interferences among the network traffic analysis and the state import and export operations. For example, it is not possible to block the analysis of network packets for all the time required to perform a complete network communication among cooperative sensors. This issue has been effectively addressed through a multithread design, that allows state import and export operations and traffic analysis to run concurrently.

The network communication framework has been implemented through XML-RPC [18] by means of the XMLRPC-C library. Snort pre-processor state data can be easily represented in XML, because it resides in C language structures.

Exchanging semi-structured data is useful for the communication of NIDS sensors which employ different detection models, because we do not impose a strict format for the state representation (data representation may be extended). *Binary XML* would be preferable as an exchange format because of its space efficiency, but it is not as standardized as XML yet.

Since XML-RPC uses HTTP as its transfer protocol, the communication follows the client/server model. In the proposed model, all Snort instances have the server role and wait for incoming RPC calls. Communication starts when a client calls one of the two available methods: `state.export` or `state.import`.

The export method takes a list of traffic identifiers as its arguments. The method output contains an array of structures, each one consisting of a label that identifies a specific pre-processor and an opaque data structure that is created by the pre-processor using the traffic identifiers as its input (see Figure 3). Thanks to this modular approach, a NIDS can implement state migration gradually, by enabling one engine at a time to export its data. The format of the exported data is outlined in Figure 4. The transferred state is composed by several blocks, each one containing data about a specific pre-processor. These blocks are further divided into sections which contain state data extracted from a specific set of streams. A stream description is attached to every state data block in order to identify the streams it is related to. Pre-processors only need to handle the data blocks that are specifically requested from them or headed to them, while the migration framework provides XML wrapping and routing. The import method takes as its input an array of structures that are analogous to those produced by

the export method. The migration-enabled Snort will feed each data structure to the matching pre-processor.

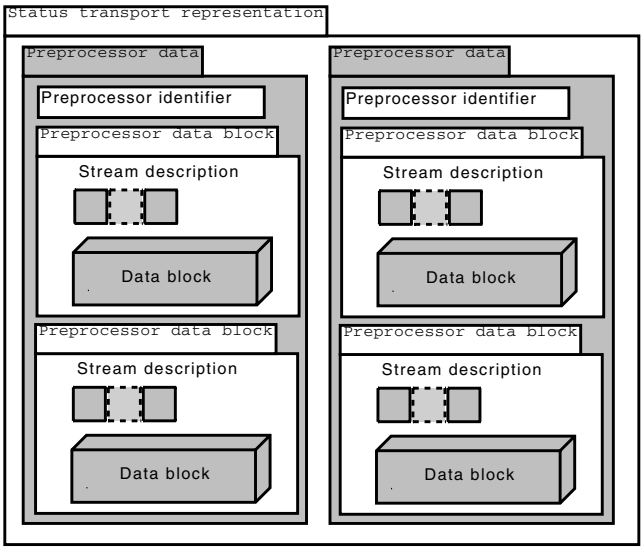

Figure 4: State data representation during migration.

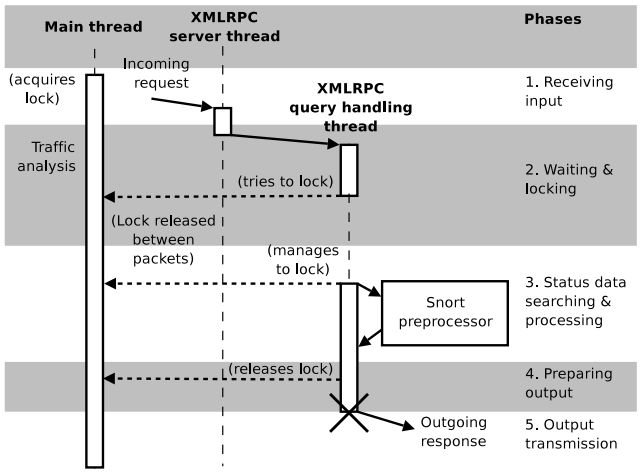

Figure 5: Thread interaction during different migration phases.

Figure 5 outlines the main steps that a NIDS sensor has to perform to export a state representation:

1. The state export process begins when the XML-RPC server receives a call to the export procedure. The XML-RPC server is implemented as a concurrent thread that runs in parallel along with the normal sensor operations, without blocking or delaying traffic analysis.

2. A handler thread is generated to handle each state export request, thus allowing the server to receive other export or import requests. The handler thread decodes the export request, and tries to obtain a lock on the sensor internal state information. A lock is necessary to ensure that the state information to be exported is consistent.

3. When the sensor is not performing packet analysis (that is, between the analysis of two packets), the main thread releases the lock, allowing the handler thread to analyze the internal state maintained by the Snort pre-processors and to look up the required state information.

4. When all the required state information has been found, the handler thread releases the lock over the internal state information and allows the main thread to resume the normal traffic analysis. After that, the handler thread prepares a well formed external state representation.

5. The external state representation is sent to the remote caller, thus completing the state export process.

The state import process is analogous to the export process with the main difference that during phase 3 (in which the handler thread holds the lock on the internal state representation) each pre-processor merges the imported state information with the current internal state.

5. VALIDATION

The first set of experiments aims to demonstrate the functional validity of the proposed state migration framework. In particular, we demonstrate the ability of the prototype to detect an attack as a result of cooperative analysis based on state migration. We remark that state migration capabilities allow a set of cooperative sensors to properly recognize complex attacks, with payloads fragmented in multiple, disordered segments.

The test involves the detection of a fragmented attack that is sent to two different sensors, hence no centralized NIDS receives the whole network traffic generated by the intrusion. The attack is represented by two TCP segments containing the payload *aaaaaaaaaaaaaaaaaaaa* (21 "a"s) that is sent to a destination port where a stream reassembly is performed. The Snort rule triggered for this type of traffic states:

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET
any (msg:"SHELLCODE x86 NOOP"; content:"aaaaaaaaaa-
aaaaaaaaaa"; classtype:shellcode-detect; sid:1394;
rev:5;)
```

A first part of the payload (including the three way handshake) is fed into a Snort sensor, while the second part of the attack (including the connection termination) is sent to a Snort instance running on a different machine. A similar scenario may occur when traffic is split over multiple links, when parallel NIDS architectures are adopted [15], and when handover procedures are carried out in wireless mobile networks.

We initially verify that one Snort sensor is unable to detect the attack if TCP stream reassembly is disabled, while the attack is correctly identified if TCP reassembly is enabled. This demonstrates that the fragmented attack is detectable only if the NIDS sensor is able to properly perform stateful analysis on the overall data stream. It is important to notice that TCP stream reassembly is a complex task, which employs identifiers such as sequence numbers to ensure that the correct order of packets is preserved. For example, by using a NOOP sled of 21 identical one-byte instructions it would seem that the reassembly order does not matter. However, TCP reassembly does not rely on stream contents, but on

sequence numbers and this makes every "a" in the payload logically different.

Then, we split a dump of the attack traffic in two parts with the guarantee that the hostile payload is divided between the two parts. Snort alone is unable to identify the attack by inspecting the two parts separately.

To demonstrate the efficacy of the proposed framework for cooperation we use two instances of the modified version of Snort. The first part of the traffic dump (containing only the first part of the payload) is sent to the first sensor, then we export the internal state representation generated by the stream4 pre-processor and we import it in the second sensor. We send the second part of the traffic dump to the second sensor. The second sensor emits the expected alert after having received the last part of the attack, thus demonstrating the functional validity of the cooperation framework and of the state migration implementation.

In the previous experiment all the activities are carried out in their logical order: analysis of the first part of the attack, state migration and analysis of the last part of the attack. However, in a real scenario it is impossible to guarantee a similar event synchronization and consequent coordination among the cooperative NIDS sensors. To demonstrate that our framework does not require an external event synchronization, we carry out a similar experiment by simulating a disordered and more realistic state migration scenario. In this experiment, we send the first part of the attack to the first sensor, and the second part of the attack to the second sensor. After that, we migrate the state from the first sensor to the second sensor. Even in this case, the second instance of Snort emits the correct alert. The same result is obtained through a state migration from the second to the first sensor.

We can conclude that when a Snort sensor which inspects just half of the attack receives a representation of the other part through state migration, the attack is identified correctly. The order between the packet analysis from live capture and state import does not matter: an alert is correctly issued whichever of the two events happens first. This result guarantees the robustness and the utmost flexibility of the state migration framework, which fulfills all requirements for a stateful intrusion detection even in systems consisting of distributed cooperative components.

6. PERFORMANCE EVALUATION

State migration is a complex process that requires several activities. However, a framework for NIDS cooperation should be suitable to real-time traffic analysis, hence it is important to evaluate the impact of the implemented state migration framework on the overall NIDS performance.

The first experiment aims to measure the third phase shown in Figure 5, during which the sensor cannot perform traffic analysis. The test machines are based on an Intel Core Duo 64bit CPU with a 2.40GHz clock frequency and 2GB of memory. The operating system uses the Linux kernel version 2.6.20 and libpcap version 0.8. During the test, we replay the IDEVAL [9] traffic dump to the NIDS sensor at different rates. We send to the XML-RPC server a request for the migration of the stream4 pre-processor internal state related to a single host and measure the duration of phase 3 for different values of the incoming traffic.

Experimental results are summarized in Figure 6. The X-axis shows the throughput of the input traffic analyzed by the sensor in Mb/s, while the Y-axis represents the duration of this phase in seconds. The points labeled with "Phase 3 duration" represent the duration of phase 3 measured for different traffic rates. From Figure 6 it is clear that the duration of phase 3 grows exponentially with respect to the analyzed throughput until 137 Mb/s (the exponential regression fits the measured data). For higher traffic volumes, the duration of this phase becomes independent of the analyzed traffic throughput. This result depends on the inner operations of the stream4 pre-processor, that uses a hash table to store reassembled network packets for each active connection. When it is necessary to find the state information for all the connections of one host, it is necessary to traverse all the stored information sequentially, hence the time grows as a function of the amount of stored states.

We measure also the number of trackers that are stored by the stream4 pre-processor (each tracker represents a network connection whose packets are stored and reassembled). The results are summarized in Figure 7, where the X-axis represents the analyzed traffic, and the Y-axis the number of trackers. It is noticeable that the number of trackers grows exponentially with respect to the analyzed throughput. Saturation is reached at 137 Mb/s, because the number of trackers reaches 8192, which corresponds to the limit of the default Snort configuration.

In Figure 8 we show the duration of phase 3 as a function of the number of trackers. The X-axis denotes the number of trackers and the Y-axis the time taken to search for the state information. We can see that the time for the export operation scales linearly with the number of trackers until saturation occurs at 8192 stored trackers, as clearly shown by the linear regression.

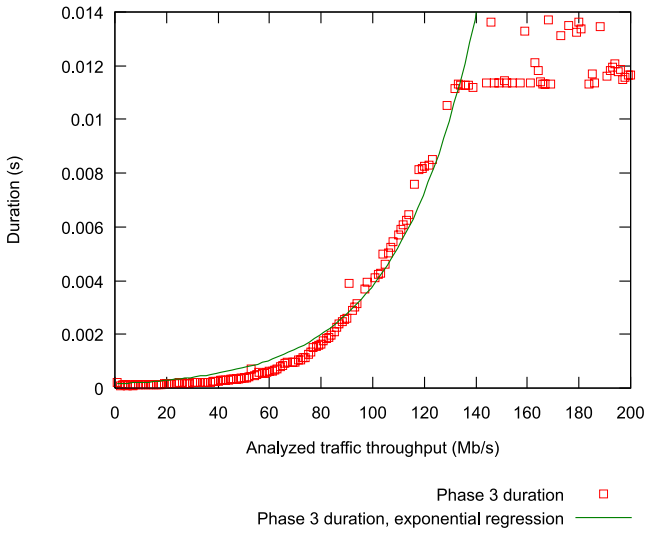

Figure 6: Phase 3 duration as a function of the analyzed throughput.

Using the standard Snort configuration, that limits the number of stored trackers to 8192, the state export operation requires to lock the internal state (thus blocking the network traffic analysis) for at most 0.014 seconds, independently of the analyzed traffic throughput.

With an analyzed traffic throughput of 200 Mb/s (the highest generated in our experimental setup) and an aver-

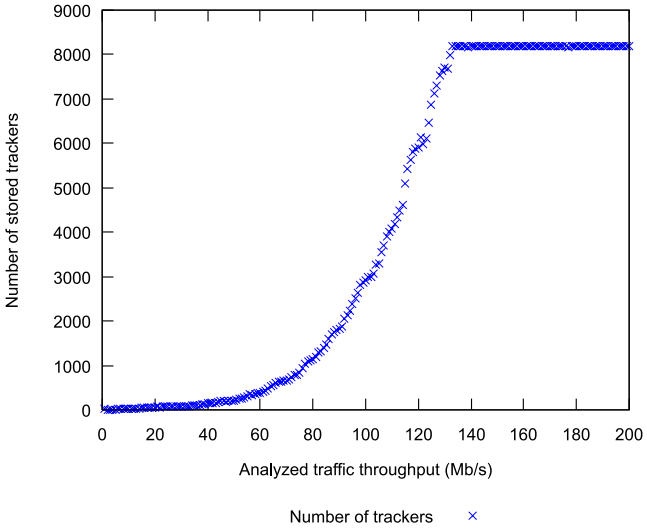

Figure 7: Number of stream4 trackers as a function of the analyzed throughput.

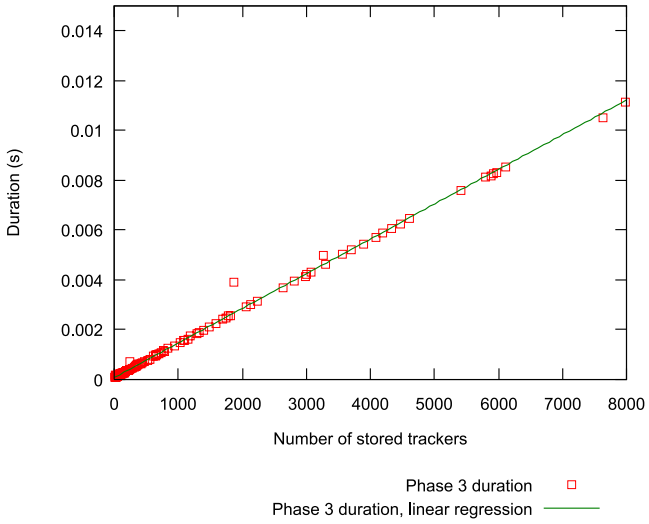

Figure 8: Phase 3 duration as a function of the stream trackers number.

age packet size of 213 bytes, during phase 3 the sensor will buffer an average of 1644 packets, corresponding to a required buffer size of 350 KB. The packet capture buffer size has to be dimensioned appropriately to avoid dropping packets. However, our experiments show that the needed buffer size is very low, compared to the hardware capabilities of a real-time NIDS sensor.

The following experiment aims to measure the time required by a sensor to perform all the operations of phases 2, 3 and 4 (all the state migration operations, excluding the network communication). While the duration of the phase 3 is independent of the amount of state information that is actually imported or exported, the number of transferred state information represents a key parameter for the duration of other state migration phases, in which all the involved state information has to be serialized (for state export) or de-serialized and merged with the internal state information (for state import).

Figure 9 shows the time required by the implemented sensor to perform phases 2, 3 and 4 during the export and import of a variable number of streams. The X-axis shows the number of transferred (imported or exported) sessions, while the Y-axis represents the duration time in seconds. Each migrated session is related to one TCP stream and contains all the information generated by the stream4 preprocessor for that connection (mainly, a normalized form of the already received data stream). As expected, the time taken for the operation grows linearly with the number of transferred sessions.

In the last set of experiments, we measure the time required for a full state migration operation, including the network communication. This experiment is carried out by exporting and importing a single stream at variable analyzed traffic. Results are summarized in Figure 10, where the X-axis represents the analyzed traffic in Mb/s, and the Y-axis the duration in seconds. The points labeled with "export time" show the durations of the phases 2, 3 and 4, while the points labeled with "overall export time" include the network communication overhead (phases from 1 to 5).

The results show that if the traffic rate is lower than 100 Mb/s, the network communication adds an almost constant overhead of 0.3 seconds, which is (at least) one order of magnitude higher than the time required by the phases 2, 3 and 4. The measurements which include network communication are noticeably noisier at higher throughput, probably because the internal state serialization has to wait for the main Snort thread to unlock the mutex guarding the stream4 internal data. The time needed for this operation may vary, because the two threads are competing. (We are considering the possibility of using a timed mutex to address this issue.) However, this also means that the lock is not acquired by the state exporting thread during the network communication. Hence, we guarantee that the main Snort thread is stopped for as little time as possible to avoid an overfill of the packet capture buffer. Analogous results have been obtained for the state import, where the network communication adds a constant overhead of about 0.3 seconds.

We can conclude that the most critical phase, in which the analysis is blocked, is also the shortest, and even a small buffer is adequate to avoid packet drop. In the overall state migration process, most of the time is spent in network communications between the coordinator and the sensor. However, this is not a critical issue because communications can be performed in parallel with respect to traffic analysis. Moreover, we remark that cooperative sensors do not need any form of synchronization, and that delays of some seconds are quite acceptable.

7. RELATED WORK

The most relevant effort in NIDS cooperation up to now is represented by the *Intrusion Detection Message Exchange Format* (IDMEF) [7]. This scheme sees a NIDS cooperation that is limited to the aggregation of the alerts generated by each individual sensor, and does not lead to an improvement in intrusion detection efficacy and reliability. Our cooperation approach is based on state migration. It allows a much more effective NIDS interoperability, because it enables the exchange of useful state information at the detection layer.

To the best of our knowledge, the only other proposal of exchanging some sort of internal state information is in

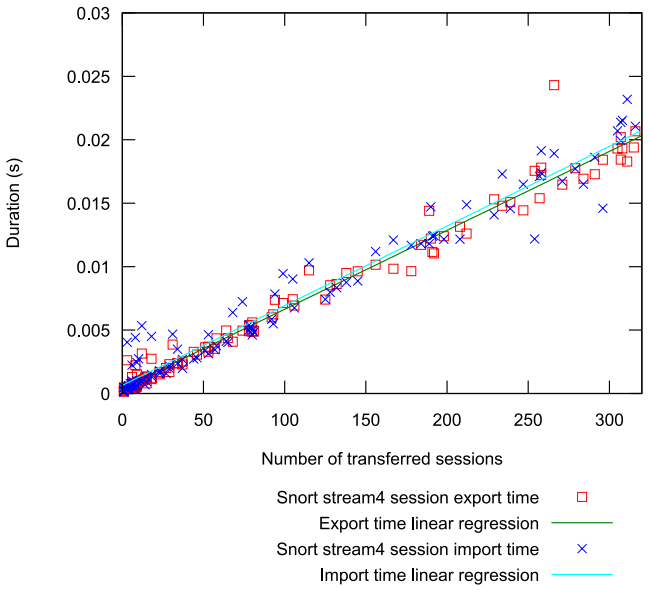

Figure 9: State import and export durations as a function of migrated sessions number.

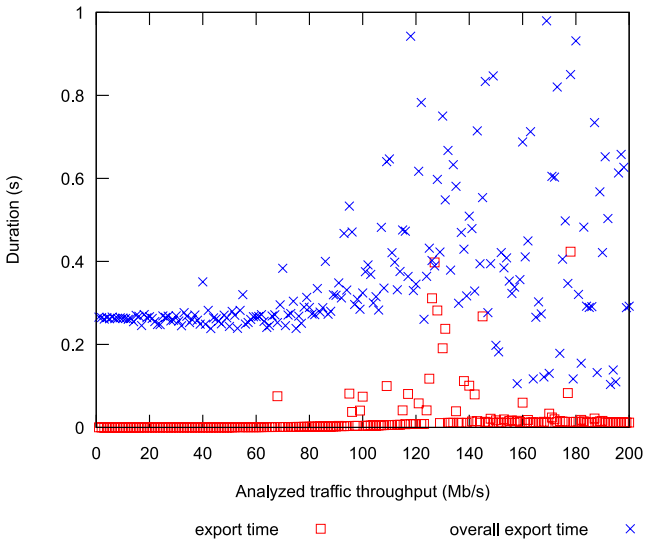

Figure 10: Impact of network communication on the state exporting time.

[16]. However, this framework follows a completely different approach from that proposed here. Their main idea is to synchronize one or more NIDS internal variables (this is a naive concept of state) that are identified in the NIDS configuration, thus generating a pool of variable values that are shared among all the cooperative NIDS sensors. Every change in the value of a synchronized variable causes a sensor to send an update message to all the cooperative sensors through a push-based model. On the other hand, the framework proposed in this paper is directly accessible by external users and network components, which are able to select at run-time which state information has to be imported or exported. This approach allows a fine-grained state information exchange. For example, it is possible to export only the state information pertaining to connections in which a particular host is one of the end-points and does not require any special sensor configuration. Each state-migration enabled NIDS sensor is always ready to export/import all the desired state information at run-time. Moreover, our state migration design permits the exchange of complex data structures such as linked lists, hash tables and packet contents, while only atomic variables may be synchronized in [16].

Another relevant difference is represented by the communication scheme employed to exchange state information. In [16] each sensor sends an update message to all the other cooperative sensors, without expecting for replies or acknowledgments. In our framework, external state import and export operations rely on a RPC mechanism, and are always implemented as a one-to-one communication. This choice ensures easy management of possible communication errors, better scalability and adaptability to heterogeneous state management policies, as described in Section 3.2.

8. CONCLUSIONS

In network scenarios characterized by growing traffic, large capacity and complex infrastructures, it is becoming necessary to refer to distributed architectures of NIDS where multiple sensors cooperate to guarantee stateful and fully reliable analysis on all traffic.

We propose an innovative state management approach and state migration framework that allow the system to share traffic state information among cooperative network intrusion detection sensors. This framework permits the migration of even complex state information among homogeneous cooperative NIDS. It can be applied to achieve higher accuracy and packet analysis flexibility in a parallel NIDS architecture, in wireless mobile networks, and in all other distributed architectures where network topology and packet routing may change. We are currently working on the definition of a standard format for state data exchange that should enable a cooperative intrusion analysis even among heterogeneous NIDS systems.

9. REFERENCES

[1] M. Andreolini, M. Colajanni, and M. Nuccio. Scalability of content-aware server switches for cluster-based web information systems. In *Proc. of the 12th International World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 2003.

[2] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2):263–311, 2002.

[3] C. J. Coit, S. Staniford, and J. McAlerney. Towards faster string matching for intrusion detection or exceeding the speed of snort. In *Proc. of the DARPA Information Survivability Conference and Exposition*, 2001.

[4] M. Colajanni and M. 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.

[5] A. Constantine and R. Stadler. Adaptable sevrer cluster with QoS constraints. In *Proc. of the 9th IFIP/IEEE International Symposium on Integrated Network Management*, Nice, France, May 2005.

[6] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational experiences with high-volume network intrusion detection. In *Proc. of the 11th ACM conference on Computer and communications security*, 2004.

[7] IETF Intrusion Detection Working Group. The intrusion detection message exchange format, 2006.

[8] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed networks. In *Proc. of the IEEE Symposium on Research on Security and Privacy*, Oakland, CA, USA, May 2002.

[9] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. Analysis and results of the 1999 darpa off-line intrusion detection evaluation. In *Proc. of the Third International Workshop on Recent Advances in Intrusion Detection*, Toulouse, France, October 2000.

[10] D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. *ACM Comput. Surv.*, 32(3):241–299, 2000.

[11] V. A. Pham and A. Karmouch. Mobile software agents: An overview. *IEEE Communication Magazine*, 36(7):16–37, 1998.

[12] P. Porras, D. Schnackenberg, S. Staniford-Chen, Davis, M. Stillman, and F. Wu. The common intrusion detection framework architecture, 1999.

[13] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.

[14] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. Characterizing the performance of network intrusion detection sensors. In *Proc. of the Sixth International Symposium on Recent Advances in Intrusion Detection*, Pittsburgh, PA, USA, September 2003.

[15] L. Schaelicke, K. Wheeler, and C. Freeland. Spanids: a scalable network intrusion detection loadbalancer. In *Proc. of the 2nd conference on Computing frontiers*, Ischia, Italy, May 2005.

[16] R. Sommer and V. Paxson. Exploiting independent state for network intrusion detection. In *Proc. of the 21st Annual Computer Security Applications Conference*, Tucson, AZ, USA, December 2005.

[17] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic memory-efficient string matching algorithms fo intrusion detection. In *Proc. of the IEEE Conference on Computer Communication*, Hong Kong, China, March 2004.

[18] D. Winer. XMLRPC, 2007.

[19] K. Xinidis, I. Charitakis, S. Antonatos, K. G. Anagnostakis, and E. P. Markatos. An active splitter architecture for intrusion detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, 03(1):31–44, 2006.