Proceedings of the Workshop on Intrusion Detection and Network Monitoring

Santa Clara, California, USA, April 9–12, 1999

Transaction-based Anomaly Detection

Roland Büschkes and Mark Borning
Aachen University of Technology

Dogan Kesdogan
o.tel.o communications GmbH & Co.

Transaction-based Anomaly Detection

Roland Büschkes, Mark Borning
Aachen University of Technology - Department of Computer Science
Informatik 4 (Communication Systems)
D-52056 Aachen, Germany
{roland, borning}@i4.informatik.rwth-aachen.de

Dogan Kesdogan
o.tel.o communications GmbH & Co
Dept. Enterprise Security
D-51063 Köln
Dogan.Kesdogan@o-tel-o.de

Abstract

The increasing complexity of both tele and data communication networks yields new demands concerning network security. Especially the task of detecting, repulsing and preventing abuse by in- and outsiders is becoming more and more difficult. This paper deals with a new technique that appears to be suitable for solving these issues, i.e. anomaly detection based on the specification of transactions. The traditional transaction and serialization concepts are discussed, and a new model of anomaly detection, based on the concept of transactions, is introduced. Applying this model to known attacks gives a first insight concerning the feasibility of our approach.

1 Introduction

Modern tele and data communication networks provide users with all kinds of services. In the future, the variety of available services will increase, ultimately offering any service anytime and anywhere. Yet, the growing range of available services also increases the complexity of the underlying networks. Therefore, it is becoming increasingly difficult to detect, repulse and prevent abuse by both in- and outsiders. Classical security mechanisms, i.e. authentication and encryption, and infrastructure components like firewalls cannot provide perfect security. Therefore, *intrusion detection systems* (IDS) have been introduced as a third line of defense.

The techniques classically applied within IDS can be subdivided into two main categories [20]:

- Misuse Detection, and
- Anomaly Detection.

Misuse detection (see e.g. [7, 11, 13]) tries to detect patterns of known attacks within the audit stream of a system, i.e. it identifies attacks directly. The main disadvantage of this approach is that the underlying database of attack patterns must be kept up-to-date and consistent. Because misuse detection techniques depend on the knowledge of recognized attack patterns, they cannot detect new attacks.

Explicitly describing the sequence of actions an attacker takes, misuse detection is based on the specification of the undesirable or *negative behavior* of users and processes. The opposite approach would be the specification of the desired or *positive behavior* of users and processes. Based on this normative specification of positive behavior attacks are identified by observing derivations from the norm. Therefore, this technique is called *Anomaly Detection*.

The main problem with anomaly detection techniques is to determine the positive behavior. Two general approaches exist:

1. Learning user and process behavior, and
2. Specification of user and process behavior.

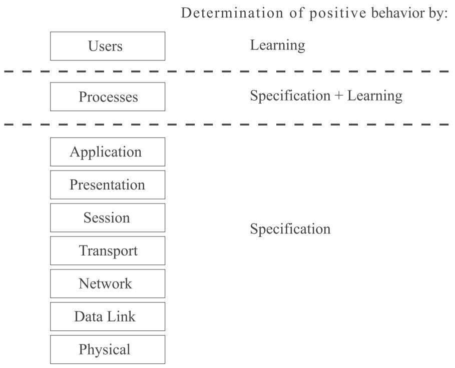

Figure 1: Anomaly Detection Approaches.

The former approach is often based on statistical methods like the calculation of means, variations and multivariate statistics [9]. Other methods use learning algorithms like e.g. neural networks or Bayesian classifiers [2]. This approach is particular popular for the profiling of users.

The latter approach, specification-based anomaly detection, was first proposed in [10]. In this paper we describe a new model called transaction-based anomaly detection and its application in communication networks. In general the transaction-based approach is similar to the specification-based approach as it formally describes positive behavior. In contrast to the specification-based approach it specifies the desired actions and sequence of actions by the definition of transactions. This explicit definition of allowed transactions becomes an integral part of the local security policy.

Figure 1 summarizes the different approaches to anomaly detection and their possible application. On the one hand, the expected behavior of the network protocol stack is well defined. This allows the application of non-intelligent techniques to monitor it. On the other hand, users are less predictable in their behavior and therefore intelligent techniques must be applied. It would be theoretically possible to formally define process behavior, but in many cases their complexity makes this approach impossible. Nonetheless, from our point of view non-intelligent monitoring techniques should be preferred over intelligent ones whenever possible. Although intelligent techniques can improve the security of a system, they rarely give a clear picture of the level of security they can guarantee. In contrast non-intelligent techniques like e.g. specification-based approaches extend the general security policy, and clearly define their guaranteed level of security.

The paper is subdivided into six main parts. Following this introduction we first define the requirements on a formal specification of positive behavior. Subsequently we introduce our transaction-based approach, briefly reviewing the general transaction model known from the database systems domain and adapting it to the IDS scenario. In section 4 we introduce sample applications of our model. After comparing the transaction-based concept with other approaches in section 5, we finally draw some conclusions and finish with an outlook on future work.

2 Specification Requirements

Any misuse or anomaly detection component should adhere to some basic design principles. These design principles also define the common criteria for the comparison of individual techniques. In the following we will review them briefly.

Obviously, the specification of positive behavior requires some formalism. Although formalisms are often very expressive and powerful, they also involve the danger of being too complicated. Complicated mechanisms tend to introduce errors, and errors open new security holes.

An appropriate formalism should therefore provide at least for the following properties:

1. ease of specification,
2. ease of monitoring,
3. completeness (no false negatives),
4. soundness (no false positives),
5. efficiency, and
6. universal validity.

As the first five desired properties in this list are well-known, we will only briefly elaborate on the requirement of universal validity to motivate our approach. Current IDS still mainly focus on the monitoring of operating system environments like UNIX or NT. Only recently the first papers dealing with the monitoring of network infrastructures [1, 3, 14, 16, 19] have been published.

However, future communication platforms like middleware (CORBA), electronic commerce, intelligent network or mobile phone (UMTS) architectures will also be vulnerable to attacks.

Misuse detection techniques can obviously not yet be studied in this context, as these systems are not in place and therefore no attacks have been identified. Anomaly detection techniques have the advantage that they can be applied right from the start of such systems. As an additional advantage, they can provide the administrators not only with information concerning potential attacks, but also with general information concerning system activity, faults and performance. Following this approach anomaly detection components establish an essential part of general network management platforms.

From our point of view two general observations hold for present and future communication platforms:

- Communication protocols can generally be specified as state transition systems.
- The protocols can be considered as defining valid transactions.

Therefore, our approach tries to maximize its universal validity by utilizing the transaction model, which is typical for many processes and especially communication processes.

3 Transaction-based approach

Transactions are a well known concept, originating from the field of *database managing systems* (DBMS) and now widely applied in other environments like distributed systems [4]. Before we will discuss their application in the context of intrusion detection we will briefly review the general concept.

3.1 The Classical Transaction Concept

Transactions generally consist of a sequence of read and write operations. If several transactions concurrently read from and write to a database, the following problems can arise:

1. Lost Update Problem
2. Dirty Read
3. Phantom Updates

$Process_1$	Time	$Process_2$
read(x)	1	
	2	read(x)
update(x)	3	
	4	update(x)
write(x)	5	
	6	write(x)

Table 1: Lost update problem.

The Lost Update Problem is depicted in Table 1. Process$_1$ reads the variable x. Immediately after process$_1$ process$_2$ also reads x. Subsequently both processes update x and write it back to the database. Obviously the update from process$_1$ is lost.

To avoid these problems transactions describe atomic operations. The provision of an atomic operation [4] means that the effect of performing any operation on behalf of one client is free from interference with operations being performed on behalf of other concurrent clients; and either an operation must be completed successfully or it must have no effect at all. Atomic transactions are often characterized by the ACID principle [6]:

1. Atomicity: All operations of a transaction must be completed, i.e. a transaction is treated as a single, indivisible unit.
2. Consistency: A transaction takes the system from one consistent state to another.
3. Isolation: Each transaction must be performed without interference with other transactions.
4. Durability: After a transaction has successfully been completed all its results are saved in permanent storage.

Obviously, the simple serial execution of transactions preserves the ACID properties, but is not efficient. Therefore, the task of a *scheduler* is to maximize concurrency and to execute transactions interlocked. The scheduler ensures that the transactions are executed in a *serially equivalent* way. Schedulers usually belong to one of the following categories:

1. Blocking scheduler

Operation	Return value	Comment
BeginTransaction	TransId	Starts a new transaction
EndTransaction(TransId)	Commit ∨ Abort	Ends a transaction
Commit(TransId)		Commits a transaction
Abort(TransId)		Aborts a transaction

Table 2: Transaction commands.

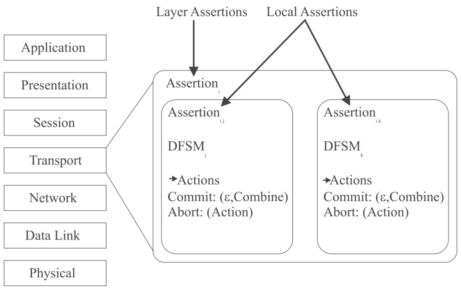

Figure 2: Layered Transaction Model.

2. Non-blocking scheduler

A blocking scheduler applies read/write-locks in order to serialize transactions. A non-blocking scheduler does not use such locks. Instead, it coordinates the transactions using e.g. timestamps. A non-blocking technique of particular interest in the context of intrusion detection is the optimistic concurrency control. This technique is based on the assumption that the likelihood of conflicts (attacks) is low. Therefore, the transactions simply proceed and after termination the scheduler checks them for any conflicts.

What actually has to be considered by the scheduler as a transaction must be specified by the clients. The sequence of operations which should be executed according to the ACID principle is grouped by a `BeginTransaction`/`EndTransaction` pair. This groups several valid commands and defines a single meta command.

Depending on whether or not the scheduler is able to execute the transaction without conflicts it is committed or aborted (see Table 2).

3.2 Transaction-based Anomaly Detection

One common argument concerning the difficulty of detecting intrusive behavior is that attacks normally consist of single steps, each of which performs a legal operation. These legal steps are generally used to interfere with another process also performing legal operations. Considering the victim's and the attacker's sequence of operations as transactions, an attack will obviously not be successful if the ACID properties are guaranteed for the victim's transaction.

Although it will probably not be efficient to design transaction-based operating and network systems, the transaction concept itself can be used to detect intrusions. Like an optimistic scheduler, an IDS can check each transaction at its end concerning its serially equivalent execution and its adherence to the ACID properties. If the check fails, the transaction is trapped. This is the general idea of transaction-based intrusion detection.

As we focus on communication systems in this paper, our model is based on the ISO/OSI reference model [5], which is made up of seven protocol layers[1]. The protocols of each of these layers can be described by *deterministic finite state machines* (DFSM) (Figure 2).

A DFSM A is a 5-tuple $A = (Q, \Sigma, q_0, \delta, F)$ with Q being the set of states which can occur during a transaction, Σ the union of all inputs, q_0 the initial state, $\delta : Q \times \Sigma \rightarrow Q$ the transition function, and F the set of final states. The syntax and semantics of a DFSM are unambiguously determined by the transition function.

The initial state of the DFSM is considered to be consistent. The DFSM passes through the operations (input events) that form the transaction. The transaction is successful if the final state is also consistent. To ensure this, atomicity, consistency and isolation properties have to be checked. The atomicity is checked on the base of the DFSM. The DFSM defines a language $\mathcal{L}(A)$. By checking whether $w \in \mathcal{L}(A)$ holds for a trace w extracted from the audit stream the transaction is tested for the atomicity property.

Subsequently the assertions (consistency property) are checked, which are expressed with the help of first order logic (FOL). Each DFSM can include assertions. Assertions can be valid only for a specific transaction (local assertions) or, on a more global level, for a certain layer (layer assertions, see Figure 2).

Finally the serially equivalent execution (isolation) of the transaction is checked (only if necessary).

[1] However, for concrete examples we use the TCP/IP protocol stack.

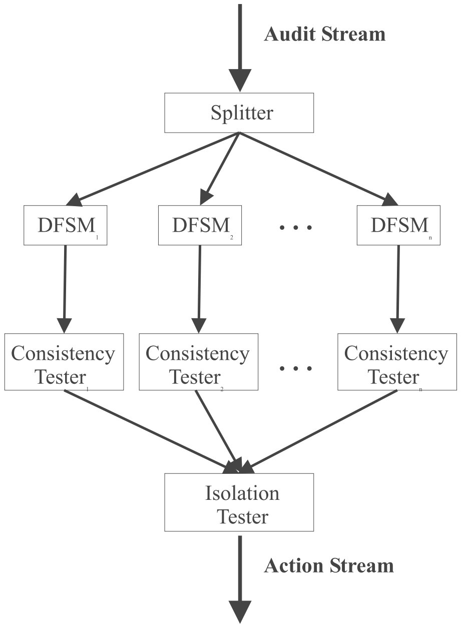

Figure 3: General architecture.

The overall architecture is shown in Figure 3 and described in more detail in the following subsections.

3.2.1 The Splitter

The audit stream contains all events received from the different sources. As we focus on network protocols in this paper the audit stream consists of packets received from the network. The task of the splitter is to distribute these single events to the corresponding state machines. The assignment of events to a state machine is straightforward, as it follows the rules of the corresponding communication protocols. A TCP session, for instance, is uniquely identified by the 4-tuple *(source address, source port, destination address, address port)*, whereas a UDP packet is identified by its socket address. For a fragmented IP packet the 16-bit identification field, in combination with the `more fragments` bit of the flags field, the source address, the destination address and the protocol type allows the splitter to feed the corresponding audit data into the appropriate state machine.

The splitter is also responsible for the scheduling of the isolation tester. The scheduler monitors the event stream for any potential conflicts and feeds the isolation tester with the necessary information.

3.2.2 Template State Machine

As mentioned above, the theory of DFSMs is in general sufficient to represent the protocol state machines. Nonetheless, a major disadvantage of the DFSM theory is the nonexistence of variables. Therefore, a separate transition has to exist for each possible value of a variable during a protocol run. This blows up the state machines substantially.

To keep the state machines small we introduce the concept of a *Template State Machine* (TSM). A TSM represents a protocol in a value independent way. The actual instance of the protocol template, i.e. the concrete DFSM, is derived during run time from the audit data.

A TSM $\mathcal{A}$ is a 6-tuple $(Q, \Sigma, V, q_0, \delta, F)$ with

- Q - set of states which can occur during a transaction,
- Σ - the union of all inputs,
- V - set of variables,
- q_0 - initial state,
- δ - transition function, and
- F - set of final states.

The transition function δ is defined as follows:

$$\delta : Q \times (AExp(\Sigma, V))^+ - \rightarrow Q$$

$AExp(\Sigma, V)$ is defined by

$$\begin{array}{rcl} e & ::= & z \mid x \mid e_1 + e_2 \mid e_1 - e_2 \in AExp(\Sigma, V) \\ \text{with} & & z \in \Sigma, \text{and} \\ & & x \in V. \end{array}$$

The semantics of the TSM, i.e. the derivation relation $\Rightarrow$, is defined by:

$$\Rightarrow \subseteq \big(Q \times \Sigma^+ \times (V \rightarrow \Sigma_\perp)\big) \times (Q \times (V \rightarrow \Sigma_\perp))$$

with

$(q, (z_1, \ldots), \sigma) \Rightarrow (q', \sigma')$, if
ex. $(e_1, \ldots) \in (AExp(\Sigma, V))^+$ with
$\delta(q, (e_1, \ldots)) = q'$ and
$\forall i : [(\sigma(e_i) = z_i \ \wedge \ \sigma' = \sigma) \vee$
$(\sigma(e_i = \perp \ \wedge \ e_i \in V \ \wedge \ \sigma' = \sigma[e_i/z_i])]$.

In contrast to the DFSM, the semantics of a TSM is not determined solely by the transition function, but also by the instantiation of the variables.

The TSM mechanism serves several purposes:

- It enables the formal and compact specification of a protocol, which can dynamically be fed into a monitor.
- It prevents state explosions.
- It can be used for the graphical specification of protocols.
- It can be used for a consistency and correctness check of the protocol specification.

In this paper we concentrate on the two former points.

3.2.3 Consistency Tester

After the atomicity of a transaction has been checked with a TSM, the next step is to ensure that the transaction leaves the system in a consistent state. The consistent state itself is defined by so-called assertions. In general, we distinguish between two kinds of assertions, local assertions and layer assertions, respectively.

To express the assertions we use first order logic (FOL) with the restriction that the negation is only allowed for atomic formulas (FOL^+)[2].

Local Assertions Local assertions are related to a specific TSM, and are checked during the execution of transitions. Therefore, we have to add Boolean expressions to our TSM concept.

The transition function is adapted as follows:

$$\delta : Q \times (\texttt{AExp}(\Sigma, V))^+ \times \texttt{BExp} \longrightarrow Q$$

`BExp` is defined by

$$\begin{array}{lll} b & ::= & \texttt{true} \mid \texttt{false} \mid e \\ & \mid & e_1 = e_2 \mid e_1 < e_2 \mid e_1 \neq e_2 \\ & \mid & b_1 \wedge b_2 \mid b_1 \vee b_2 \end{array}$$

The semantics of a TSM changes in the following way:

$$\Rightarrow \subseteq \big(Q \times \Sigma^+ \times (V \to \Sigma_\perp)\big) \times (Q \times (V \to \Sigma_\perp))$$

with

$(q, (z_1, \ldots), \sigma) \Rightarrow (q', \sigma')$, if
ex. $(e_1, \ldots) \in (AExp(\Sigma, V))^+$, $b \in Bexp(\Sigma, V)$ with
$\delta(q, (e_1, \ldots)) = q'$ and $\sigma(b) = \texttt{true}$ and
$\forall i : [(\sigma(e_i) = z_i \ \wedge \ \sigma' = \sigma) \vee$
$(\sigma(e_i) = \perp \ \wedge \ e_i \in V \ \wedge \ \sigma' = \sigma[e_i/z_i])]$.

Obviously, we do not need the whole expressiveness of FOL^+ for local assertions, as quantifiers do not necessarily make sense within the context of a TSM. Each transition is triggered by a single packet, which is checked for obeying the local assertions. No quantifiers are necessary in this context.

Layer Assertions Layer assertions are valid for all TSMs belonging to a single layer. In contrast to local assertions layer assertions can make use of the complete FOL^+, including universal and existential quantification. They can be checked either at the initial state or at a final state, i.e. before or after checking for atomicity. In general, a violation of an assertion should be detected as early as possible. Therefore, layer assertions will be checked at the initial state of a TSM whenever possible.

As layer assertions can be expected to be more complex than local assertions, they will not be checked during the execution of the corresponding TSM. In general, failure of a check of a layer assertion during run-time is possible, because intermediary states of the TSM must not necessarily fulfil the layer assertions. Intermediary states only describe the transformation process from one consistent state to another. This does not imply that all intermediary states have to fulfil the layer assertions, they only have to fulfil the local assertions. The layer assertions must hold for the initial state and for the final state.

3.2.4 Isolation Tester

The isolation tester is responsible for checking the isolation property, i.e. for checking whether a transaction performed without interference from other transactions. For general processes running on a single node the application of the isolation tester is obvious: it can detect attacks based on race conditions. To do so, the splitter has to monitor the audit stream for accesses to critical

[2] Any formula can be transformed in an equivalent negation-free formula.

objects (e.g. directories and files) and start the isolation tester, which checks for a violation of the isolation property.

With regard to communication processes the use of an isolation tester is not immediately obvious. Nonetheless, certain attacks on the protocol level can be interpreted as a violation of the isolation property. This is especially true if we take the distributed nature of a communication process into account, i.e. if we do not only consider the effects of a protocol run on a single system, but broaden our transaction concept to include all nodes involved in the communication process. This results in the so-called compound transaction concept.

3.3 Compound Transactions

Signaling and communication protocols form the core of modern tele and data communication networks. They define the communication between two or more nodes. Therefore, it is not sufficient to consider only local transactions. Instead, it must also be possible to monitor *distributed transactions*.

Distributed transactions describe the parallel execution of single transactions on different nodes. This is not always sufficient in our context. Therefore, we enhance the concept of distributed transactions and also take the communication between the single transactions into account, i.e. the transactions communicate with each other, and the progress of one transaction depends on the progress of the other. The communication itself can also be considered as a transaction. Together with the local protocol actions of the sender and the receiver the communication forms a *compound transaction*.

Compound transactions cannot only be used to describe the horizontal communication between peer layers, but also the vertical communication between different layers within a protocol stack.

Whether a communication process is monitored as a distributed or a compound transaction depends on the level of granularity demanded by the IDS. We use the following syntax to distinguish between these two kinds of transactions:

1. Distributed Transactions: $A \parallel B$
2. Compound Transactions: $A \leftrightarrow B$

In order to describe the communication process between the involved components, we use *Message Sequence Charts* (MSCs).

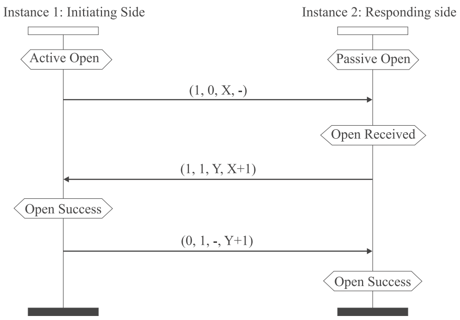

Figure 4: MSC for TCP's three-way handshake.

MSCs [8] are a well-known technique for the visualization of communicating processes. A single MSC consists of different instances, which can communicate with each other. Each instance represents a DFSM, which is additionally allowed to communicate with another DFSM. Figure 4 shows the MSC for TCP's three-way handshake protocol in graphical notation.

An MSC is not a description of the complete behavior of a system. Rather it expresses one execution trace. To specify a system more detailed a collection of MSCs may be used [12].

The concrete textual syntax of the core language of MSCs [12] is given in Figure 5. The events `in` and `out` are used for the communication with other instances and the environment.

The correspondence between message outputs and message inputs has to be uniquely defined by message name identification. A message input must not be executed before the corresponding message output has been executed [12]. In addition, the activities along one single instance axis are completely ordered, but no notion of global time is assumed. Events of different instances are ordered only via messages, which imposes a partial ordering on the set of events being contained.

This partial ordering is checked by the IDS. The communication processes on both sides are identified by a unique transaction identifier (`TransId`). The transaction identifier is composed of the session characteristics already mentioned in section 3.2.1. For a TCP session, for example, the pair of socket addresses and the corresponding sequence numbers are used. To check the compound transaction for its correct execution two checks have to be performed:

```
msc        ::=  msc mscid; mscbody endmsc;
mscbody    ::=  instdef mscbody | ε
instdef    ::=  instance instid; instbody endinstance
instbody   ::=  event instbody | ε
event      ::=  in msgid from instid;
             |  in msgid from env;
             |  out msgid from instid;
             |  out msgid from env;
             |  action actid;
```

Figure 5: Basic Message Sequence Charts.

- Check the local transactions for their successful execution (aborted or committed?).
- Check the communication between the local transactions according to the MSC.

Step 1 is similar to the general approach taken for distributed database transactions. Each node involved in the transactions is polled by a coordinator for the success of its local transaction. Based on the information gathered from the nodes the coordinator decides on whether to commit or abort the transaction and informs the nodes (two-phase commit protocol).

In our case the coordinator will either be the receiver or a trusted third party (TTP). Our monitoring of distributed transactions follows the approach taken by the two-phase commit protocol. Nonetheless, for compound transactions the communication between the nodes has also to be taken into account. Communication between the nodes is checked to conform with the corresponding MSC. As we do not assume a global synchronized time, this check focuses on the partial order of the events defined by the MSC, particular the message in- and outputs. In the sense of the state operator λ_M defined in [12] we check that a message input is not executed before the corresponding message output. The coordinator therefore requests the input and output events from the participating nodes and performs this check.

Compound transactions can be mapped onto distributed transaction. The checking of the MSC simply defines an additional local transaction that leads to a local decision about committing or aborting the whole transaction based on its course of communication.

This method will only work if the source of the audit data is reliable. Otherwise, an attacker could send the requesting node a faked set of in- and output events. The IDS components therefore need to communicate in an authentic way.

4 Examples

To demonstrate the feasibility of our approach, we will examine two aspects related to our model using practical examples. On the one hand we will elaborate on the aspect of how to subdivide a communication process horizontally and vertically into transactions for monitoring. On the other hand we will have a closer look at certain attack scenarios and describe how they will lead to trap events based on violations of the ACID principle. Regarding the latter we will distinguish between attacks which are to be considered in the context of local transactions (e.g. Ping of Death) and attacks to be considered in the context of compound transaction (e.g. IP spoofing).

The typical scenario for these examples is a *Virtual Private Network* (VPN).

4.1 Scenario

Future applications will increasingly involve several distributed sites, which will typically be interconnected by a VPN.

VPNs are a well suited scenario for studying IDS and distributed IDS functionality as they involve two or more parties belonging to different security domains with different trust relationships. We consider three general scenarios for our approach:

1. Local security domain.
2. Cooperation of trusting security domains (e.g. sites interconnected via a VPN).
3. Cooperation of non-trusting security domains (e.g. VPN sites and network provider).

The two latter scenarios are of special interest to distributed and compound transactions.

4.2 Atomicity

The check for atomicity of a transaction helps to detect various attacks as abnormal behavior. This includes *SYN flooding*, a *denial of service* attack.

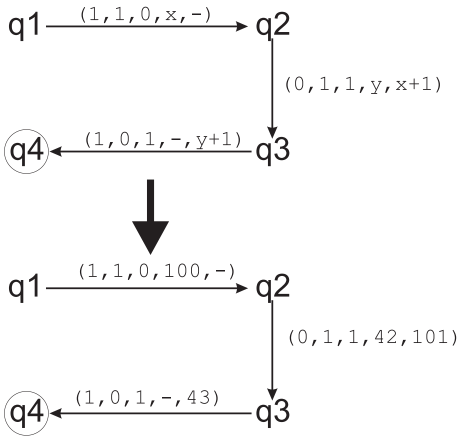

Figure 6: TSM for the 3-Way-Handshake.

With SYN flooding a server is hit by a large number of connection requests. The sender address is normally spoofed.

By defining the TCP connection establishment process as a transaction it becomes obvious that the corresponding DFSM on the victims side is not completed. The final state q_4 is not reached, and therefore the atomicity property is violated.

Figure 6 shows the corresponding TSM of the 3-way-handshake. The 5-tuple used obeys the following scheme:

$$(b_0, b_1, b_2, c_0, c_1)$$

with

- b_0 - Boolean value, indicating whether a packet was sent ($b_0 = 0$) or received ($b_0 = 1$);
- b_1 - Boolean value, indicating whether or not the SYN flag was set;
- b_2 - Boolean value, indicating whether or not the ACK flag was set;
- c_0 - sequence number of the local node;
- c_1 - sequence number of the remote node, incremented by 1 (acknowledgement number).

Obviously, a SYN flooding attack can be detected by the violation of the atomicity property. The protocol itself recognizes the incomplete 3-way-handshake through a timeout. This timeout triggers the receiver to send a reset (RST bit set). As a result the next message received by our monitor is an invalid 5-tuple, which aborts the transaction.

An example for an attack involving two sides in the detection process (second scenario) is *IP-spoofing* based on sequence numbers. In fact, the SYN flooding attack is part of this attack. If host C hijacks a connection between sender A and receiver B, the local transaction monitored at site B will end with a `Commit(TransId)`. By considering the communication as a distributed or compound transaction the whole communication is decomposed into two or three local transactions. Only if all of these local transactions commit, the distributed or compound transaction also commits. For IP-spoofing the local transaction at sender A will obviously abort and therefore the whole communication transaction will be aborted.

Other attacks that can be detected by monitoring the atomicity of a transaction include e.g. *port scanning attacks*.

4.3 Consistency

The consistency of a transaction guarantees that the communication process is error free and adheres to the protocol specification. This also helps to detect various attacks.

One example is the *Ping of death*. This attack is based on the fragmentation of IP packets. According to [17], IP packets including the header may have a size of up to 65535 octets. Without any option fields the header has a size of 20 octets. Packets which exceed the size accepted by layer 2 of the protocol stack[3], are fragmented and reassembled at the receiver side.

An ICMP ECHO request consists of an eight octet ICMP header [18], which is followed by the request data. Therefore, the maximum size of the data field is 65507 octets.

Nonetheless, it is possible to construct and send an invalid packet, in which the data part contains more than 65507 octets. This is based on the fragmentation process. The fragmentation is done by specifying an offset and the size of the fragment. For the last fragment it is therefore possible to exceed the 65535 octet limit for the whole packet.

A layer assertions therefore specifies that the total size of a packet cannot exceed 65535 octets. This layer assertion has to be put into action by the TSM, i.e. the layer assertion is transformed into a local assertion and

[3] Defined by the maximum transmission unit (MTU).

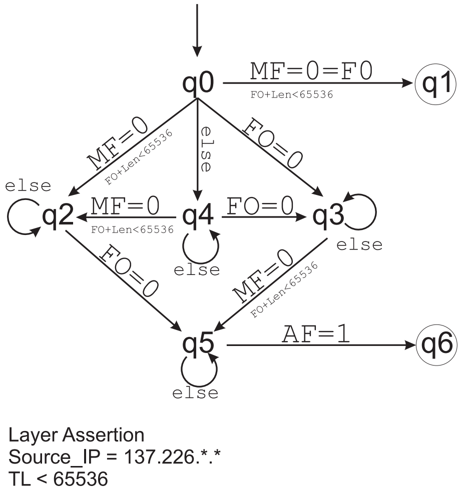

Figure 7: TSM for the reassembling of IP fragments.

a corresponding check. In our example the check can be done when receiving the final fragment, because for this fragment the sum of the fragment offset and fragment length has to be less than 65536.

Figure 7 shows the corresponding TSM using the following abbreviations:

- *FO* (Fragment Offset) - offset (in 8-byte units) of the fragment from the beginning of the original datagram;
- *MF* (More Fragments) - one bit in the *flags* field turned on for each fragment comprising a datagram except the final fragment;
- *AF* - Boolean expression, indicating whether or not all fragments have been received;
- *TL* (Total Length) - total length of the IP datagram;
- *Source_IP* - IP source address of the fragment sender.

The TSM uses both local and layer assertions. The local assertions are part of the inscription of the transitions, the layer assertions are explicitly specified below the TSM.

The local assertions ensure that the fragment with $MF = 0$ is checked if the sum of the fragment offset and the fragment length is actually less than 65536 octets. This puts the second layer assertion into action.

The first layer assertion defines the admissible general conditions of the communication. In this simple example the set of valid sources is limited to IP addresses of the form $137.226.*.*$ or written as a FOL^+ expression:

$$\exists a_3, a_4 \in \{0, \ldots, 255\} : \; SrcAddr = 137.226.a_3.a_4$$

This assertion is checked before the run through the TSM.

4.4 Isolation

Isolation means that each transaction must be performed without interference with other transactions, i.e. the intermediate effects of a transaction must not be visible to other transactions [4]. As already stated the violation of the isolation property is not immediately obvious in the network context discussed so far, although e.g. IP-spoofing based on sequence numbers could also be interpreted as a violation of the isolation property. We will therefore not elaborate on this topic in this paper. Nonetheless, as one major design goal of our approach is universal validity, we will briefly stress its validity in the context of processes. With processes, monitoring of the isolation property helps to prevent security failures due to improper synchronization of processes or race conditions.

An abstract example for improper synchronization is given in Table 1. A concrete example for such a lost update problem [10] describes a scenario in which a user invokes the *passwd* program to change his password while an administrator is editing the password file. The two programs modify the password file simultaneously, leaving it with an incorrect content.

Following our approach we can consider either one or both of the accesses to the password file as a transaction. By doing this, the transaction is monitored for its isolated execution and the password file cannot be left with an incorrect content. In contrast to the network scenario the allowed transactions have not already been defined by any kind of protocol specification. When entering the process and user level depicted in Figure 1, the allowed transactions must be explicitly specified in the security policy. In order to avoid restrictions for the normal user it is sufficient to define transactions on the basis of valid action sequences for administrators and security critical processes.

4.5 Durability

So far, we have not considered in detail the property of durability. As atomicity, consistency and isolation are properties which focus on the execution of the transaction, the durability property influences the period after the successful execution of a transaction. The effects of a successful transaction have to survive even hard- and software failures or, in our case, attacks.

What does that mean in the context of an IDS? For databases the durability property (in combination with the atomicity) ensures that after a failure the database state to be reconstructed is well defined. This property is also useful for systems under attack. After an attack has been detected it is often difficult to determine the latest safe state. If the administrator cannot determine the exact time of a successful intrusion, he cannot be sure that his backups are free from security leaks. Therefore, an IDS should not only support the detection of attacks, it should also support the administrator in the reconstruction of the system and the reestablishment of a safe state. Nonetheless, the details of this approach will be discussed elsewhere.

5 Related Works

The approach most similar to ours is described in Calvin Ko's paper ”Execution Monitoring of Security-Critical Programs in a Distributed System: A Specification-Based Approach” [10]. It follows a specification-based approach and also describes the meaning of atomic actions in relation to intrusion detection. However, [10] does not follow the parallels to the classical transactional model and therefore, from our point of view, looses some of its generality. For instance [10] cites the examples of a super user editing the password file and an user changing his password in the same file. The given solution is a specification in the form of a so-called *Parallel Environment Grammar* (PEG). In the PEG the parallel execution of two programs[4] is specified as shown in Figure 8.

Although we focus on communication processes in this paper it is obvious that the transaction model also applies to this process centric scenario. Following the transaction model it is sufficient to specify the most high-level (su level) action as a transaction. It is not necessary to explicitly specify the parallel execution, because of the check for the serially equivalent execution performed by the scheduler.

```
   Environment Variables
1.  Int E=0;

   Start Expression
2.  <progA> || <progB>

   Hyperrules
3.  <progA> -> <writeA, E>.
4.  <writeA, 0> -> <openA> <closeA>{E=E-1;}.
5.  <open> -> open_A {E=E+1;}.
6.  <close> -> close_A.

7.  <progB> -> <writeB, E>.
8.  <writeB, 0> -> <openB><closeB>{E=E-1;}.
9.  <openB> -> open_B {E=E+1;}.
10. <closeB> -> close_B.
```

Figure 8: Parallel Environment Grammar (PEG).

Another system related to ours is the Bro system [15]. The specifications made by Bro can be integrated into our approach as a part of the `Assertion`-section of a transaction or a layer. As we do not yet have specified the exact data types to be used in our approach, we are currently considering the reuse of some of the data types already specified by Bro.

The main limitation of our model is related to the specification process. In general, the specification of communication processes can be extracted immediately from the protocol specification. Therefore, the specification can either be provided by the vendor or any other trusted third party (ease of specification). Any attack based on an implementation error can be indirectly detected by our approach because it will be recognized as an anomaly. Any closer examination and classification (error or attack) of the anomaly can be done either by a human operator or a misuse intrusion detection component.

Nonetheless, the transaction model cannot deal with specification or management errors, which form the other two sources of vulnerabilities. The transaction model only deals with implementation errors. If the specification of a communication protocol or its transformation into a specification for the anomaly detection system itself is faulty, attacks based on these errors will remain undetected because by definition they follow the specification and can therefore not be considered to be

[4] The concrete and therefore longer PEG for the password file example can be found in section 5.4 of [10]. The general structure is the same as for this simple example.

anomalies. The same holds for any management errors of the environment.

6 Conclusions and Outlook

In this paper we have proposed a new technique for anomaly based intrusion detection. The detection of anomalies is based on the definition of correct transactional behavior. This definition of correct, desired behavior defines the system's multi-level security policy, which is monitored during run-time by the IDS. In contrast to classical database and other transactional systems we do not enforce the distinct transactions to be executed according to the ACID properties. Instead, in the sense of an optimistic scheduler, we monitor the system only for any potential conflicts.

Obviously, it is neither desirable nor feasible to monitor all host activities and connections. The monitoring will therefore be performed dynamically, i.e. the different sensors will be activated and configured on demand by a control instance being part of the general network management environment. This is one of the main design criteria for our *Aachener Network Intrusion Detection Architecture* (ANIDA), which forms the broader context to which the described anomaly detection approach belongs.

Our approach is currently under development and first results of the prototype will be available soon.

Acknowledgements

We would like to thank Dr. Thomas Noll for his helpful comments and suggestions.

References

[1] K. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and Ronald A. Olson. Detecting disruptive routers: A distributed network monitoring approach. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, May 1998.

[2] R. Büschkes, D. Kesdogan, and P. Reichl. How to increase security in mobile networks by anomaly detection. In *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC'98)*. ACM, December 1998.

[3] S. Cheung and K. Levitt. Protecting routing infrastructures from denial of service using cooperative intrusion detection. In *Proceedings New Security Paradigms Workshop 1997*, Cumbria, UK, September 1997.

[4] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems - Concepts and Design*. Addison-Wesley, 2nd edition, 1994.

[5] Fred Halsall. *Data communications, comuter networks, and open systems*. Addison-Wesley, 3rd edition, 1992.

[6] T. Härder and A. Reuter. Principles of transaction-oriented database recovery. *Computing Surveys*, 15(4), 1983.

[7] K. Ilgun, Richard A. Kemmerer, and Phillip A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.

[8] ITU-T. *ITU-T Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-T, Geneva, 1993.

[9] H. Javitz and A. Valdes. The SRI IDES statistical anomaly detector. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 316–326, May 1991.

[10] C. Ko. *Execution Monitoring of Security-Critical Programs in a Distributed System: A Specification-Based Approach*. PhD thesis, University of California, Davis, 1996.

[11] S. Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Department of Computer Science, Purdue University, August 1995.

[12] S. Mauw and M. A. Reniers. An algebraic semantics of basic message sequence charts. *The Computer Journal*, 37(4), 1994.

[13] A. Mounji. *Languages and Tools for Rule-Based Distributed Intrusion Detection*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix Namur, Belgium, September 1997.

[14] B. Mukherjee, L. Heberlein, and K. Levitt. Network intrusion detection. *IEEE Network*, pages 26–41, May/June 1994.

[15] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, January 1998.

[16] Phillip A. Porras and Peter G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, October 1997.

[17] J. Postel. *RFC 791: Internet Protocol*, September 1981.

[18] J. Postel. *RFC 792: Internet Control Message Protocol*, September 1981.

[19] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. Grids: A graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, Baltimore, 1996.

[20] A. Tucker Jr., editor. *CRC Computer Science and Engineering Handbook*. CRC Press, December 1996.