NSTAT: A Model-based Real-time Network Intrusion Detection System

Richard A. Kemmerer

Reliable Software Group
Department of Computer Science
University of California
Santa Barbara, CA 93106

The Reliable Software Group at UCSB has developed a new approach to representing computer penetrations. This approach models penetrations as a series of state transitions described in terms of signature actions and state assertions. State transition representations are written to correspond to the states of an actual computer system, and they form the basis of a rule-based expert system for detecting penetrations. The system is called the State Transition Analysis Tool (STAT).

On a network filesystem where the files are distributed on many hosts and where each host mounts directories from the others, actions on each host computer need to be audited. A natural extension of the STAT effort is to run the system on audit data collected by multiple hosts. This means an audit mechanism needs to be run on each host. However, running an implementation of STAT on each host would result in inefficient use of computer resources. In addition, the possibility of having cooperative attacks on different hosts would make detection difficult. Therefore, for the distributed version of STAT, called NSTAT, there is a single STAT process with a single, chronological audit trail. The group has designed a client/server approach to the problem. The client side has two threads: a producer that reads and filters the audit trail and a consumer that sends it to the server. The server side merges the filtered information from the various clients and performs the analysis.

I. Introduction and Overview

The Reliable Software Group at UCSB has recently developed a new approach to representing computer penetrations and has applied this approach to the development of a real-time expert system intrusion detection tool. The approach is called *state transition analysis* and is a method for representing the sequence of actions that an attacker performs to achieve a security violation.

State transition analysis is based on the premise that all computer penetrations share two common features. First, penetrations require the attacker to possess some minimum prerequisite access to the target system. This prerequisite access may range from access to certain files, devices, telephone lines, etc. to possession of information regarding a particular security-relevant function. Second, all penetrations lead to the acquisition of some previously unheld ability. That is, a subject performing a penetration is doing so to gain something. Whether the ability gained is unauthorized access to data, access to another user's privileges, or just a perverse satisfaction in harming others, something is gained. Identifying what is acquired by the penetrator is analogous to identifying what aspect of a computer's security has been compromised by the penetration.

In state transition analysis, a penetration is viewed as a sequence of actions performed by an attacker that leads from some initial state on a system to a target compromised state, where a state is a snapshot of the system representing the values of all volatile, semi-permanent and permanent memory locations on the system. The *initial state* corresponds to the state of the system just prior to the execution of the penetration, and the *compromised state* corresponds to the state of the system resulting from the completion of the penetration. Between the initial and compromised states are one or more intermediate *state transitions* that an attacker performs to achieve the compromise.

After the initial and compromised states of a penetration scenario have been identified, the key actions, called *signature actions* are identified. Signature actions refer to those actions that, if omitted from the execution of an attack scenario, would prevent the attack from completing successfully. The information produced by the above steps are represented graphically, as a state transition diagram.

What makes this model-based approach useful for penetration analysis is that it requires the analyst to identify the minimum number of key, or signature, actions for a penetration and to organize them visually, using a representation similar to state machine diagrams. This approach is useful for describing penetration scenarios in that it provides an interesting level of abstraction to the analyst: just above the system call and user command representation, and just below English descriptions. Representing penetrations as a sequence of user commands or system calls does not identify the key action sequences responsible for the compromised state, nor does it identify the minimum requirements of the penetration. English, on the other hand, is too ambiguous and often not very concise. The model-based state transition diagrams identify precisely the requirements and compromise of the penetration, and list only the key actions that must occur for the successful completion of the penetration.

A prototype of STAT, called USTAT, was implemented on SunOS 4.1.1 to validate the functional capabilities and conceptional soundness of the state transition analysis approach. The C2-BSM (Basic Security Module) was used to collect audit data [Sun 91]. USTAT consists of the following components.

- The Preprocessor
- The Knowledge-base
  - The Fact-base
    - The Fact-base Initializer
    - The Fact-base Updater
  - The Rule-base
    - The State Description Table
    - The Signature Action Table
- The Inference Engine
- The Decision Engine

The *audit record preprocessor* is responsible for reading, filtering, mapping and finally passing the BSM audit records to the inference engine in the format that is required by USTAT. The preprocessor provides the inference engine with a generic audit record format. It also enables the Site Security Officer (SSO) to create the state transition representations with the abstraction of USTAT action names instead of BSM-specific event names. Of the 239 different events that are audited by the BSM only 28 are used by the USTAT preprocessor, and they are mapped onto 10 different USTAT actions.

The *knowledge-base* has two components: the fact-base and the rule-base. The fact-base contains information about the objects of the system. Initializing the fact-base for USTAT is done by the fact-base initializer module and by some additional manual processing. The fact-base updater consists of those routines that keep the fact-base up-to-date for the consistent operation of the inference engine. The rule-base is the rule representation of the state transition sequences. USTAT's *fact-base* consists of groups of files or directories (called filesets) that share certain characteristics that make them vulnerable to certain types of attack scenarios. These filesets provide a very convenient way of generalizing the penetration scenarios. In the state transition sequences each state consists of one or more state assertions, where each state assertion consists of a function name with zero or more arguments. The signature actions correspond to the action types used by the preprocessor. The USTAT *rule-base* stores state transition information in two text files referred to as the State Description Table and the Signature Action Table, which store the state assertions and the signature actions, respectively. The inference engine uses this information to match the actions of incoming USTAT audit records to the actions of state transition scenarios.

USTAT's *inference engine* uses an event driven, forward chaining inference scheme [MO 88]. The inference engine uses a structure called the inference engine table to keep track of all possible penetration scenarios. At any point in time, this table consists of snapshots of penetration scenario instances (instantiations), which are not yet completed on the target system. Each entry contains information about the history of the instantiation, such as the users involved, files involved, related audit records, etc. The inference engine applies the rules of the state transition representation regardless of the identity of the attacker. That is, the attack of cooperating users and a single user makes no difference to USTAT. The identity of the attacker is important only if the state assertions of a certain state include an assertion requiring the same user to perform the operations. Similarly, the inference engine applies the rules in the rule-base regardless of the login sessions. The amount of time passed between two signature actions has no effect on USTAT's inference mechanism.

The *decision engine* is responsible for informing the SSO about the results of the inference engine activities. The output of the decision engine could be one or more of the following actions, which are ranked from the simplest to the most complicated.

1) Inform the SSO that a compromise has been achieved.
2) Inform the SSO whenever a state of any instance of the scenarios has been satisfied.
3) Suggest possible actions to the SSO to preempt a state transition that can lead to a compromised state.
4) Play an active role in preempting the attack.

The decision engine implemented for the USTAT prototype performs the first three.

The modular architecture of STAT requires that only the preprocessor needs to be modified for use on a different platform. The Reliable Software Group, with funding from SUN Microsystems, has recently ported the USTAT 4.1.1 implementation to Solaris 2.4. USTAT is also being ported to Data General's DG/UX.

A natural extension to the STAT effort is to run STAT on audit data collected by multiple hosts. On a network filesystem where the files are distributed on many hosts and where each host mounts directories from the others, actions on each host computer need to be audited. That means an audit mechanism needs to be run on each host. Running an implementation of STAT on each host would likely result in inefficient use of computer resources. Also, the possibility of having cooperative attacks on different hosts would make the detection difficult. The approach discussed in this report is to provide a single STAT process with a single, chronological audit trail. A client/server approach is currently being built, where the client side has two threads: a producer that reads and filters the audit trail and a consumer that sends it to the server. The server side merges the filtered information from the various clients and performs the analysis. This distributed approach is known as NSTAT.

STAT is designed to detect the same computer penetrations targeted by currently existing rule-based penetration identification tools. Like these tools, STAT is effective in detecting abuse from misfeasors as well as external attackers. Unfortunately, STAT is also equally ineffective in detecting masqueraders. Thus, when incorporated into an intrusion detection system, STAT is expected to work in combination with other intrusion detection tools that specialize in detecting masqueraders (e.g., a profile-based anomaly detector). Collectively, the tools will complement each other's coverage, providing the ability to detect masqueraders as well as misfeasors. The plan is to integrate the STAT approach with the EMERALD event monitoring environment, which is the successor to the IDES/NIDES systems, being developed at SRI [PN 97]. The plan is to combine the rule-based penetration identification capabilities of STAT with the profile-based anomaly detection capabilities of EMERALD to make an even stronger intrusion detection and analysis tool. Although the initial plan is to integrate the STAT approach with EMERALD, the integration of the STAT approach with other intrusion detection components, such as the DIDS and Network Security Monitor work at UC, Davis, will also be investigated.

Another desirable addition to STAT is query support for the security officer via the SSO interface. The most direct way to provide this support is via the interface between the decision engine and the SSO. Currently data flows only from the decision engine to the SSO interface. However, the decision engine can be redesigned to allow queries from the SSO interface to the decision engine. These queries could also be initiated from other intrusion detection components, such as the anomaly detection component.

A further enhancement to the decision engine is to add a surveillance mode capability. For example, specific subject ID's could be provided to the STAT decision engine via the SSO interface. The decision engine could then be acutely sensitive to these subjects and generate a warning message each time one of these subjects fires a penetration rule. Surveillance mode could also be implemented via an interface between STAT and a concurrently running anomaly detector. As a subject's suspicion rating increases through anomaly detection, so too could the decision engine's sensitivity to that subject.

One obvious question affecting all penetration rule-base implementations is that of completeness. Penetration rule-bases are only capable of representing those known penetrations that are traceable via audit data analysis. Since one cannot verify that all of the security flaws within a particular target system have been identified, one cannot expect a penetration rule-base to include rule chains for every possible system penetration.

Accordingly, the ability to facilitate the process by which security administrators are able to update the STAT rule-base, on-site, with new and variant penetrations is highly desirable. The STAT rule-base is specifically designed to allow non-experienced rule-base programmers, such as system administrators and security officers, to update the rule-base themselves. The plan is to develop a user-friendly interface for updating the STAT rule-base without having to take the system out of operation. Thus, penetration rule chains can be added as the security officer learns of new penetrations.

The next section discusses the details of the USTAT implementation of the STAT approach. This is followed by a description of the distributed extensions of NSTAT. Finally, the NSTAT approach is compared and contrasted to other intrusion detection approaches and systems.

II. Details of the USTAT Prototype

The UNIX State Transition Analysis Tool (USTAT) uses the information contained in a user's audit trail as input to compare the state changes produced by the user to the state transition diagrams of known penetrations. USTAT makes use of the audit trails that are collected by the C2 Basic Security Module of SunOS.

USTAT can be characterized by three basic properties: it is a real-time expert system intrusion detection tool, it employs rule-based analysis on the audit trails of multi-user computer systems, and it searches for known penetrations. One of USTAT's main features is to attempt to preempt an attack before any damage is done to the system. This preemption is possible only with real-time analysis. The major issue in real-time analysis, however, is whether USTAT will be fast enough to catch up with the audit records when the user load is high. The results of tests focusing on this issue are presented in [Ilg 93]. USTAT's ability to detect cooperative attacks, to detect penetrations, the steps of which may span more than one user session, and its ability to foresee an impending compromise distinguish it from other rule-based penetration identification systems.

The components of USTAT were briefly overviewed in the first section of this report. Further detailed descriptions of each of the components are presented in the following paragraphs.

The *audit record preprocessor* is responsible for reading, filtering, mapping and finally passing the BSM audit records to the inference engine in a special format that is required by USTAT. The USTAT audit record structure is defined by the triple:

<Subject, Action, Object>

meaning "the Subject performs the Action on the Object." Each of these attributes contains further fields that are used to reveal as much information as possible about the particular attribute. The Subject is identified by the triple:

<Real User ID, Effective User ID, Group ID>,

and the Action is identified by the triple:

<Action, Time, Process ID>.

Finally, the Object is identified by the eight tuple:

<Object Name, Permissions, Owner, Group Owner,
Inode#, Device#, File System ID, Target>.

where the Object Name is the name of the file identified with its full path, and the Target field is effective only if the action is Hardlink or Rename. All of the fields in a USTAT audit record can be obtained directly from the BSM audit records. For an in-depth discussion of BSM features and audit records as regards to USTAT, refer to [Ilg 92].

There are 239 different events that are audited by BSM. Out of these, only 28 events are used by the preprocessor and mapped onto 10 different USTAT actions. The inference engine operates using these 10 action types. The preprocessor also takes the return value of an event into account, filtering out all of the BSM records that indicate that a call was not finished successfully.

The *knowledge-base* consists of a fact-base and a rule-base. The fact-base consists of file-sets, which are groups of files or directories that share certain characteristics that allow one to generalize penetration scenarios. The current version of the fact-base used by USTAT is given in the following table.

FILESET	**CHARACTERISTICS**
Fileset #1	Restricted read files
Fileset #2	Restricted write setup files
Fileset #3	Files authorized to read Fileset #1
Fileset #4	Files authorized to write Fileset #2
Fileset #5	Non-writable system executables
NWSD	Non-writable system directories
HARDLINK	System hardlink information

USTAT Filesets

The first six of the filesets are used directly in state assertions, whereas the last one is used by the inference engine to identify variations of scenarios through references by hardlinks. Details of the filesets can be found in [IKP 95].

The rule-base consists of a State Description Table and a Signature Action Table. The state description table contains state assertions, which define each state of a penetration. The evaluation of a state assertion results in a true or false value, and the keyword "not" in front of a state assertion negates the result. Sample descriptions of state assertions are:

fullname(file_var)=full_path,

which evaluates to true if the file_var matches the pathname given in the right-hand side.

member(file_set, file_var),

which evaluates to true if file_var is a member of the file_set.

euid=user_id,

which evaluates to true if the effective user id of the subject of the audit record being processed equals the user_id.

shell_script(file_var),

which evaluates to true if the file_var is a shell_script with the #!/bin/sh mechanism.

The signature action table contains the required signature actions to move from one state of a penetration to the next. Some sample signature actions, which correspond to the action types used by the preprocessor, are:

read(file_var)
exit(file_var)
modify_owner(file_var)

modify_perm(file_var)
hardlink(file_var, file_var)

The *inference engine* uses a structure called the Inference Engine Table to handle and detect all penetration scenario instances simultaneously. At any point in time, this table consists of several snapshots of penetration scenario instances (instantiations), which are not yet completed on the target system. Each row represents one instance of a possible penetration scenario. Each column corresponds to one state of a scenario and it depicts how far a compromise is from being achieved. Each row also contains information about the history of the instantiation, such as users involved, files involved, related audit records, etc. Whenever the inference engine detects an audit record that matches the next action and satisfies the next state of a state transition diagram, it duplicates the row and "marks" the corresponding cell on the duplicated row. The reason for duplicating the row is that the original row can still represent part of another instantiation.

The *decision engine* is responsible for informing the SSO of the status of any potential attack. It employs a structure called the Decision Table, which contains a decision message for each state of a state transition scenario. Whenever the decision engine is notified by the inference engine about a state change it displays the following information to the console of the machine running USTAT.

1) the number of the state transition scenario for which this state change has occurred
2) the message in the decision table that corresponds to the last satisfied state
3) all of the filenames that were involved in this instance of the scenario
4) the real user id and the effective user id of the user who performed the last signature action for this instantiation

Using the data produced by the decision engine the SSO can foresee an impending compromise and has enough information to either take preemptive action, or to take precautions to prevent future attacks of the same nature.

The complete details of the USTAT implementation can be found in [Ilg 92, IKP 95].

III. Distributed STAT

The Reliable Software Group at UCSB is currently extending USTAT to be run on audit data collected on multiple hosts. The resulting system, called NSTAT, uses a multi-threaded client/server model where audit data is collected from multiple clients and fed to a single server for analysis. To be more precise, on each machine to be audited, a client runs as a daemon, reading the audit logs, filtering and converting the data to NSTAT format, and sending it across the network via an encrypted socket session. A single machine runs the NSTAT server, which receives the separate data streams from all the client machines, merges these streams into a single stream, and then processes that stream.

Each client runs two threads: one thread reads the audit data from the file (or whatever source may exist in the future, such as directly from the kernel), while the other filters and converts this data into NSTAT format. Once it has constructed an NSTAT record, the thread sends the record off to the server.

The design for the server is a bit more complicated. The main server thread listens on a dedicated TCP streams port; whenever a new client session begins, it creates a new thread, and hands that thread a file descriptor for the new session. The algorithm looks like this:

```
do forever
        listen to socket
        if there is a new client session
                create a new thread
                pass the new thread a file descriptor for the socket
od
```

Each of these threads operates independently, merging its own stream into the universal stream. A major issue is that all the records in the universal stream need to be in order, since NSTAT's analytical engine depends upon the records being in chronological order. Because a TCP streams socket is used, the TCP protocol guarantees that all the packets for a given session will arrive in order. But it is still necessary to make sure that each server thread puts its records in the universal stream in the appropriate order.

One can only be assured that the universal stream is in the correct order if for any given time, all the records from each client up till that time have arrived at the server machine. Otherwise, the following situation could occur: clients A and B both send records from time 1, and both these records arrive at the server at time 1. They are put in the universal stream. Next, client A sends a record from time 3 to the server, which arrives at time 3, and this record is put in the universal stream. Client B, however, sends a record from time 2 to the server, but this record is lost due to a congested network. The server assumes that all records up to time 3 have arrived from all clients, and goes ahead and processes the stream. Later, client B's delayed record finally comes in. By then it is too late, since the server cannot process out-of-order records, and the record must simply be discarded. If this record contained crucial information, such as a part of an attack, NSTAT will have failed.

To defeat this problem, a periodic synchronizations approach is used. The idea is that all clients and the server agree to send sync records at set intervals. For each interval, the server must make sure that all clients have sent sync records before proceeding to process that chunk of the universal stream. In order for this scheme to work, all the machine's clocks must agree. This is not a problem for the first prototype, since SunOS and Solaris both have daemons that ensure that a network of machines all have the same time. This is an additional problem, however, that needs to be addressed when other operating systems are included in the network.

The server also has a processing thread, which executes the USTAT analytical code. This thread has a higher scheduling priority than the other threads. Finally, there is an action thread, which is responsible for taking action upon a rule-matching. All threads communicate with each other via shared memory, signals, and semaphores. Concurrency issues are handled with mutual exclusion locks. The algorithm for the merging threads is:

```
do forever
        read a control structure from the socket
        if there is a record coming in
        begin
                read the record
                record->sync = control->sync
                INSERT the record in the queue
        end
```

```
    od
```

where procedure INSERT looks as follows:

```
    proc INSERT(record)
    begin
        if record->sync is false
        begin
            search backwards from end of queue
            insert according to time
        end
        else
        begin
            search forwards from last sync record for this thread
            insert according to time
            find the blocking record for this interval
            increment the block count by one
        end
    end
```

The algorithm for the analytical thread is:

```
    do forever
        if the queue is empty
        begin
            yield
        end
        if this record is a blocking record
        begin
            if block_count =  number_of_clients
            begin
                analyze record
                remove record from queue
            end
            else
            begin
                yield
            end
        end
        else
        begin
            analyze record
            remove record from queue
        end
    od
```

As mentioned above, execution speed is an issue with any real-time analysis system. To achieve better execution times, the plan is to implement a custom memory manager for all objects relating to the queue.

Another issue that needs to be addressed with the network extension is that NSTAT needs to distinguish between different audit trails and apply a different set of rules to each, based on the operating system being run. For example, setuid shell intrusions, which are a problem in SunOS, are not possible in Solaris.

As a first step to integrating NSTAT with other third party intrusion detection components, the plan is to integrate it with SRI's EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) environment, which is being developed for DARPA [PN 97]. The rule-based penetration capabilities of NSTAT, working as both local and higher-level EMERALD monitors should complement the statistical profiling capabilities of EMERALD.

In the prototype USTAT implementation the decision engine sends information to the SSO about the status of intrusion scenarios that are in progress or have just completed. These messages are sent when specified rules are fired by the inference engine. As other third party components are integrated with NSTAT new actions will need to be added, such as having the decision engine send/receive information directly to/from these components. For instance, the decision engine may have several levels of sensitivity. Then another component, such as an EMERALD statistical anomaly detection monitor, could advise the NSTAT decision engine to raise its sensitivity level. As a result, NSTAT may preempt an attack by shutting down a suspect user, where when running at a lower level of alert it may only have sent a warning message to the SSO.

In addition to allowing cooperating components to exchange information with the NSTAT decision engine, the plan is to allow the SSO to send information to the decision engine, such as requesting that the alert level be raised or lowered. The SSO will also have query capabilities. For instance, the SSO could query to determine the group of users that is involved in any of the ongoing penetrations instances. Based on the information that the decision engine returns, the SSO might choose to raise the alert level of the decision engine by sending it a command.

USTAT was designed to allow SSOs to update the rule-base themselves. However, when the USTAT system is running the only input that it accepts is the audit data. The plan is to allow more interaction between the SSO and the rule-base. The SSO will be provided with a standard way of entering or removing rules from the rule-base. In particular, the SSO will be able to update the state description table and the signature action table while NSTAT is running. Of course, the rule-base interface will also have to communicate with the inference engine to signal that new rules need to be inserted or deleted. When the inference engine receives the signal it should finish processing the current audit records and then retrieve the new rules. Semaphores will be used to provide mutual exclusion for updating the tables.

IV. Comparison with Other Research

The state transition analysis approach targets the same penetrations that are identifiable by current rule-based penetration identification tools. The state transition analysis approach, however, offers several key advantages over existing rule-based implementations. Some of the inherent characteristics that limit the effectiveness of current rule-based penetration identification tools are identified in the next three paragraphs.

A major weakness in current rule-based penetration identification tools is their direct dependence on audit record fields. In the current systems, rule-bases represent the expected audit trails of penetrations, and these tools essentially pattern match and bind rules in their knowledge-base to audit records. Unfortunately, there is very little flexibility in this one-to-one (rule-to-audit record) representation. For instance, for a given penetration scenario there may be slight variations of the same penetration that will produce different audit record sequences. Thus, even if a scenario is represented in the rule-base, a minor variation of the scenario can go unnoticed. One solution to improving the flexibility of the expert system's ability to identify penetration scenarios is to use higher-level representations of the scenarios in the rule-base (i.e., scenario representations that are audit record independent).

Another limitation to current penetration identification expert systems is their inability to foresee an impending compromise and preempt or limit the damage before it occurs. At best, current penetration identification systems report compromises after they are reached or take measures to terminate an intrusive process once the damage has begun. Current approaches are designed with little, if any, reasoning capabilities that allow them to take preemptive action before a compromised state is reached. Intrusion detection systems should be able to anticipate an impending compromise with some measure of confidence and either forewarn the system administrator or take steps to preclude a penetration before it achieves its compromise. Current penetration identification tools are also limited in their ability to identify even semi-sophisticated attacks, such as those performed by cooperating attackers. The IDES rule-base, for example, does not take into consideration two or more users working together to execute a penetration [LJL 89].

Lastly, current penetration rule-bases are neither easily created nor easily updated. In general, expert rule-bases tend to be nonintuitive, requiring the skills of experienced rule-base programmers to update them; penetration rule-bases are no exception. Penetration rule-bases are created by interviewing system administrators and security analysts to collect a suite of known penetration scenarios and key events that threaten the security of the target system. The rule-base programmer then identifies the audit records that correspond to the scenario or key event and constructs rules to represent the penetration based on the expected audit records. The development of penetration rules are *ad hoc* and provide little chance for the rule-base to be updated on-site. Procedures that allow system administrators and security analysts to develop and incorporate penetration rules into the rule-base locally should be provided. Doing so will result in more effective rule-base management, allowing site-specific policy information and newly discovered penetrations to be incorporated into the rule-base in a timely manner.

Unlike current rule-based analysis tools, NSTAT focuses on a penetration's signature actions rather than the audit records that record the actions. Furthermore, NSTAT rules are designed to support the representation of interdependencies among the actions within the penetration scenarios, allowing one rule chain to represent multiple variations in the order in which a penetration's actions may be performed. By providing the ability to support multiple variations within a single penetration rule chain, NSTAT increases its flexibility in identifying penetrations that would otherwise be missed by other rule-based tools.

The state transition analysis approach also has the ability to detect cooperating attackers and attacks across user sessions. The NSTAT inference engine views the audit trail globally, focusing on the events occurring on the system rather than the actions of individuals. When a sequence of actions is found to match the actions represented in a state transition sequence, NSTAT then constructs a list of the users who contributed steps within the penetration. NSTAT will also anticipate impending compromises and take preemptive action; i.e., it can detect an attack in progress and stop it before it completes. It will also be able to raise the level of concern as attacks or partial attacks are detected.

A major difference between NSTAT and the other tools is that NSTAT rule chains are constructed from state transition diagrams. An important advantage of using signature actions rather than audit records to represent penetrations is that they result in more intuitive rule chains. The NSTAT rule-base is specifically intended not to require the use of an expert rule-base programmer, but to be readable and updatable by site security officers and system administrators. State transition diagrams, which are discussed in detail in [IKP 95], aid in the development of penetration rule chains, in that they provide an analyst with a visual representation of the key actions that must occur in order for an attacker to move the system from the initial prerequisite state to the compromised state. In this sense, the state transition analysis approach is similar to the Model-based Intrusion Detection approach proposed by Garvey and Lunt [GL 91]. Both techniques provide a higher level representation of user behavior (i.e., above the audit record level) providing easier readability and rule generation.

Another advantage of NSTAT is its portability. NSTAT's modular architecture was designed to allow it to be easily ported to different operating systems and different platforms. For some ports it will only require changing the audit preprocessor component. In addition the decision engine component can easily be modified to tailor NSTAT's response to intrusions to satisfy the requirements of specific sites where it is installed.

A limiting factor to the effectiveness of state transition analysis as a detection technique is that not all modifications and references to system attributes are recorded within an audit trail. Therefore, not all penetrations that are representable using the state transition diagrams will be detectable using NSTAT. For example, audit mechanisms do not usually record references or modifications of volatile memory such as registers or user space. As a result, penetrations involving the illegal modification or reference to volatile memory space may not be detectable using state transition analysis, even though they are representable as state transition diagrams. This limitation, however, does not necessarily indicate a weakness in state transition analysis as an intrusion detection approach. No intrusion detection approach can be expected to detect penetrations that compromise attributes whose accesses are not recorded by the audit mechanism. Such penetrations execute beneath the visibility of audit data analysis tools and correspond to what [And 80] referred to as clandestined usage. More information on this topic can be found in [HLH 85], which examines a number of penetration scenarios and categorizes those penetrations that are identifiable via audit data analysis from those that are not.

NSTAT is designed to detect the same computer penetrations targeted by currently existing rule-based penetration identification tools. Like these tools, NSTAT will be effective in detecting abuse from misfeasors as well as external attackers. Unfortunately, NSTAT will also be equally ineffective in detecting masqueraders. No intrusion detection approach, however, stands alone as a catch-all for computer penetrations; each approach is technically suited to identify a subset of the security violations to which a computer system is subjected. Other intrusion detection tools (e.g., a profile-based anomaly detector) specialize in detecting masqueraders. By incorporating NSTAT into an intrusion detection system with other intrusion detection tools that specialize in detecting masqueraders the tools will complement each other's coverage, providing the ability to detect masqueraders as well as misfeasors.

Another problem with NSTAT, which is true of all rule-based penetration identification tools, is that it can only detect known attacks. A successful intrusion detection system needs to also detect unknown attacks. As discussed above, NSTAT is intended to be part of an integrated system with different intrusion detection approaches interoperating to detect both previously known attacks and new attacks.

Because the USTAT prototype was built to monitor a single workstation, its rule-base does not include network-based scenarios. Other ongoing research, such as the Network Security Monitor project at UC Davis, the EMERALD project at SRI, and the COAST penetration scenario work at Purdue have been collecting and discovering network-based scenarios. No problems are anticipated in adapting the scenarios they have collected that can be represented as state transition sequences for use in the NSTAT rule-base. In fact, network-based scenarios have already been added to the USTAT rule-base for use in NSTAT.

In summary, NSTAT overcomes most of the major weakness that are found in current rule-based penetration identification tools, but like these tools, NSTAT can not detect all security violations. However, NSTAT's model-based approach, which allows it to represent intrusion scenarios at a higher level than individual audit records, coupled with its modular architecture makes NSTAT a prime candidate to be integrated into a large-scale heterogeneous information system where there are many different operating systems running on many different platforms and generating audit trails that are in differing formats.

References

[And 80] J.P. Anderson, "Computer Security Threat Monitoring and Surveillance," James P. Anderson Co., Fort Washington, PA, April 1980.

[GL 91] T.D. Garvey and T.F. Lunt, "Model-based Intrusion Detection," *Proceedings of the 14th National Computer Security Conference,* Baltimore, MD, October 1991.

[HLH 85] L.R. Halme, T.F. Lunt, J. Van Horne, "Analysis of Computer System Audit Trails - Intrusion Classification," Sytek Technical Report TR-85012, Mountain View, CA, Oct. 1985.

[Ilg 92] K. Ilgun, "USTAT: A Real-time Intrusion Detection System for UNIX," Master's Thesis, Computer Science Department, University of California, Santa Barbara, CA, Nov. 1992.

[Ilg 93] K. Ilgun, "USTAT: A Real-time Intrusion Detection System for UNIX," *Proceeding of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1993.

[IKP 95] K. Ilgun, R.A. Kemmerer, and P.A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach," *IEEE Transactions on Software Engineering,* Vol. 21, No. 3, March 1995.

[LJL 89] T.F. Lunt, R. Jagannathan, R. Lee, and A. Whitehurst, "Knowledge-Based Intrusion Detection," *Proceeding of the 1989 AI Systems in Government Conference,* March 1989.

[MO 88] J. Martin and S. Oxman, *Building Expert Systems: A Tutorial,* Prentice-Hall, Englewood Cliffs, N.J., 1988.

[PN 97] P.A. Porras and P.G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," *Proceeding of the IEEE Symposium on Research in Security and Privacy,* Oakland, CA, May 1997.

[Sun 91] Sun Microsystems Incorporated, SunOS Release 4.1.1, C2-BSM Patch, Revision A, Mountain View, CA, 1991.