NetSTAT: A Network-based Intrusion Detection System

Giovanni Vigna and Richard A. Kemmerer
Reliable Software Group
Department of Computer Science
University of California Santa Barbara
[vigna,kemm]@cs.ucsb.edu

Abstract

Network-based attacks are becoming more common and sophisticated. For this reason, intrusion detection systems are now shifting their focus from the hosts and their operating systems to the network itself. Network-based intrusion detection is challenging because network auditing produces large amounts of data, and different events related to a single intrusion may be visible in different places on the network. This paper presents a new approach that applies the State Transition Analysis Technique (STAT) to network intrusion detection. Network-based intrusions are modeled using state transition diagrams in which states and transitions are characterized in a networked environment. The target network environment itself is represented using a model based on hypergraphs. By using a formal model of both the network to be protected and the attacks to be detected the approach is able to determine which network events have to be monitored and where they can be monitored, providing automatic support for configuration and placement of intrusion detection components.
Keywords: *Security, Intrusion detection, Networks, STAT.*

1 Introduction

In the last decade, networks have grown in both size and importance. In particular TCP/IP networks, and most notably the world-wide Internet, have become the main means to exchange data and carry out transactions. They have also become the main means to attack hosts.

The growing importance of the Internet and of network security issues has shifted security concerns from the operating system domain to the network itself. Security efforts are experiencing a similar shift. Local, centralized approaches are evolving into distributed and networked approaches, in an effort to cope with the interconnected heterogeneous platforms and to provide scalable solutions.

Intrusion detection is an approach to security that is also evolving towards networked environments. Intrusion detection began as a technique for detecting masqueraders and misfeasors in standalone systems [1, 8], but in the last few years the focus of intrusion detection has moved towards networks [23]. In Network-oriented Intrusion Detection Systems (NIDSs) the source of events (and the object) for the analysis is a distributed system composed of many hosts and network links. The goal of NIDSs is to detect attacks that involve the network and may span different hosts.

Network-level monitoring and distribution pose some new requirements on intrusion detection systems:

- Networks produce a large amount of data (events). Therefore, a network-based intrusion detection system should provide mechanisms that allow the Network Security Officer (NSO) to customize event "collectors" so that they listen for only the relevant events.
- Relevant events are usually visible in only some parts of the network (especially in the case of large networks). Therefore, a NIDS should provide some means of determining where to look for events.
- A NIDS should generate a minimum amount of traffic over the network. Therefore, there should be some local processing of event data.
- A NIDS needs to be scalable. At a minimum, a "local" NIDS should interoperate with other NIDSs (possibly in a hierarchical structure).
- For maximum effectiveness, NIDSs should be able to interoperate with host-based IDSs so that misuse patterns include both network events and operating system events.

The NetSTAT tool presented in this paper addresses the aforementioned issues. The NetSTAT approach models network attacks as state transition diagrams, where states and transitions are characterized in a networked environment [32]. The network environment itself is described by using a formal model based on hypergraphs, which are graphs where edges can connect more than two nodes [4]. By using formal representations of both the intrusions and the network, NetSTAT is able to address the first three issues listed above. The analysis of the attack scenarios and the network formal descriptions determines which events have to be monitored to detect an intrusion and where the monitors need to be placed. In addition, by characterizing in a formal way both the *configuration* and the *state* of a network it is possible to provide the components responsible for intrusion detection with all the information they need to perform their task autonomously with minimal interaction and traffic overhead. This can be achieved because network-based state transition diagrams contain references to the network topology and service configuration. Thus, it is possible to extract from a central database only the information that is needed for the detection of the particular modeled intrusions. Moreover, attack scenarios use assertions to characterize the state of the network. Thus, it is possible to automatically determine the data to be collected to support intrusion analysis and to instruct the detection components to look only for the events that are involved in run-time attack detection. This solution allows for a lightweight, scalable implementation of the monitors and focused filtering of the network event stream, delivering more reliable, efficient, and autonomous components.

In order to address the remaining issues, namely scalability and integration with other IDSs, NetSTAT is designed to be *interoperable*. A NetSTAT instance protecting a network can interact with other NetSTAT

instances in an integrated way. That is, reports produced by the analysis of a network's activity can be used as input events by other NetSTAT instances monitoring different networks. Thus, enterprise-wide or even cross-enterprise attacks can be detected. A NetSTAT instance can also interact with other IDSs. NetSTAT is designed to interoperate with IDSs that focus on different targets (e.g., host-based IDSs) or with systems that use different approaches to characterize intrusions (e.g., IDSs based on statistical anomalies or thresholds). Therefore, it is possible to compose different approaches for maximum effectiveness. For example, a host-based IDS could notify the NetSTAT tool of suspicious activity occurring at a particular host. As a consequence, NetSTAT could raise its monitoring level for the scenarios that involve the host under attack. As another example, "doorknob-rattling" attacks, which can be naturally modeled and detected by a NIDS based on statistics, can be used as a basis to model more complicated attacks. Although the interoperability of NetSTAT is an important and interesting issue, it will not be discussed in this paper.

The remainder of this paper is structured as follows. The next section presents some related work on NIDSs and introduces the state transition analysis technique. Section 3 presents the NetSTAT architecture. Section 4 describes NetSTAT at work on several example attack scenarios. Finally, Section 5 draws some conclusions and outlines future work.

2 Related work

Network-oriented intrusion detection systems can be roughly distinguished into *distributed IDSs* and *network-based IDSs*. A survey of network-oriented IDSs is given in [23].

Distributed IDSs are an extension of the original, single-host intrusion detection approach to multiple hosts. Operating intrusion detection analysis over audit streams collected from several sources allows one to identify attacks spanning several systems. Examples of this kind of systems are IDES [15], NSTAT [16], ISOA [33], and AAFID [2].

Network-based IDSs take a different perspective and move their focus from the computational infrastructure (the hosts and their operating systems) to the communication infrastructure (the network and its protocols). These systems use the network as the source of security-relevant information. Examples of this kind of systems are NSM [9] and DIDS [29]. NSM is a network security monitor that analyses traffic on a local area network to detect network-based intrusions. The system is tailored to operate on a single broadcast link in order to access all the traffic between hosts. DIDS is an evolution of NSM that integrates data retrieved from the network with events gathered by the auditing facilities of the hosts composing the network. In this case, analysis is also limited to a single-link LAN and event processing is centralized.

Recently, the target of network-based IDSs has been widened to address detection in large, complex network topologies. GrIDS [5] and EMERALD [25] are two examples of this evolution. GrIDS is a graph-based intrusion detection system whose goal is the detection of attacks spanning large network infrastructures. GrIDS aggregates the results of localized host-based and network-based intrusion detection systems into a graph-structure. The analysis of the resulting graph allows one to highlight correlation among different attacks. This system is particularly suitable to detect "doorknob rattling" and worm-like attacks. EMERALD is a system whose design attempts to address the scalability issues associated with intrusion detection in large networks. EMERALD follows a building-block approach that allows for hierarchical composition of decentralized *service monitors*, that, in the current design, apply statistical analysis to network data. Even though decentralization of security monitors and hierarchical composition is a promising approach, neither GrIDS nor EMERALD provide any automatic mechanism to determine what events are to be monitored and where the security event collectors are to be placed. This leaves the configuration of the system totally in the hands of the Network Security Officer. This is also true for commercial network-based intrusion detection systems like NetRanger [6] and RealSecure [14]. Both NetRanger and RealSecure evolved from single-link monitor approaches to a distributed architecture. The NetRanger intrusion detection systems is composed of two type of modules: *Sensors* and *Directors*. Sensors are network security monitors that analyze the network traffic on a network segment and the logging information produced by Cisco routers

to detect network-based attacks. A Director is a module responsible for the management of a group of Sensors. Directors can be structured hierarchically to manage large networks. The RealSecure intrusion detection system is composed of *Network Engines*, *System Agents*, and *Managers*. The Network Engines are network monitors equipped with attack signatures that are matched against the traffic on a network link. The System Agents are host-based intrusion detection systems that monitor security sensitive log files on a host. These modules report their findings to the central Monitor. The Monitor displays the information to the user. In addition, the Monitor provides functionalities for the remote administration of the installed System Agents and Network Engines. In both NetRanger and RealSecure the deployment and configuration of the network-based intrusion detection components is a task performed manually by the Network Security Officer. In addition, none of the two commercial systems provides the user with a well-defined complete language to specify new, possibly complex, attack signatures.

The approach presented in this paper extends the *State Transition Analysis Technique* (STAT) [13] to network-based intrusion detection. The state transition analysis technique was conceived as a method to describe computer penetrations as sequences of actions that an attacker performs to compromise the security of a computer system. Attacks are graphically described by using *state transition diagrams*. *States* represent snapshots of a system's volatile, semi-permanent, and permanent memory locations. A description of an attack has a "safe" starting state and at least one "compromised" ending state. States are characterized by means of *assertions*, which are predicates on some aspects of the security state of the system, such as file ownership, user identification, or user authorization. *Transitions* between states are indicated by *signature actions* that represent the actions that if omitted from the execution of an attack scenario would prevent the attack from completing successfully. Typical examples of signature actions include reading, writing, and executing files.

The state transition analysis technique has been applied to host-based intrusion detection, and a tool, called USTAT [24, 11, 12], has been developed. USTAT uses state transition diagrams as the basis for rules to interpret changes in a computer system's state and detect intrusions in real-time. The changes in the computer system's state are monitored by leveraging off of the auditing facilities provided by security-enhanced operating systems, such as Sun Microsystems' Solaris equipped with the Basic Security Module [30]. USTAT is able to interpret the audit trail produced by a single operating system.

To detect attacks that involve multiple hosts sharing network file systems, a new tool, called NSTAT [16], has been developed. NSTAT uses a client-server architecture to collect audit records from different sources (hosts), merge them into a single audit trail, manage synchronization and correlation among the different trails, and then perform state transition analysis on the resulting trail. Even though NSTAT's components are distributed over a network, NSTAT does not perform monitoring at the network level; its event sources are the auditing facilities available on the monitored hosts.

The natural evolution of state transition analysis is its direct application to networks. NetSTAT is the result of this evolution. NetSTAT is a tool aimed at real-time network-based intrusion detection. NetSTAT takes advantage of the peculiar characteristics of intrusion detection based on analysis of network traffic. Networks provide detailed information about computer system activity, and they can provide this information regardless of the installed operating systems or the auditing modules available on the hosts. In addition, network auditing can be performed in a non-intrusive way, without notching the performance of either the monitored hosts or the network itself, and network audit stream generation cannot be turned off. Finally, network traffic has more precise and timely timing information than the audit records produced by the existing OS auditing facilities.

The next sections present the design of the NetSTAT tool and some examples of its use.

3 NetSTAT Architecture

NetSTAT is a distributed application composed of the following components: the *network fact base*, the *state transition scenario database*, a collection of general purpose *probes*, and the *analyzer*. A high level view of

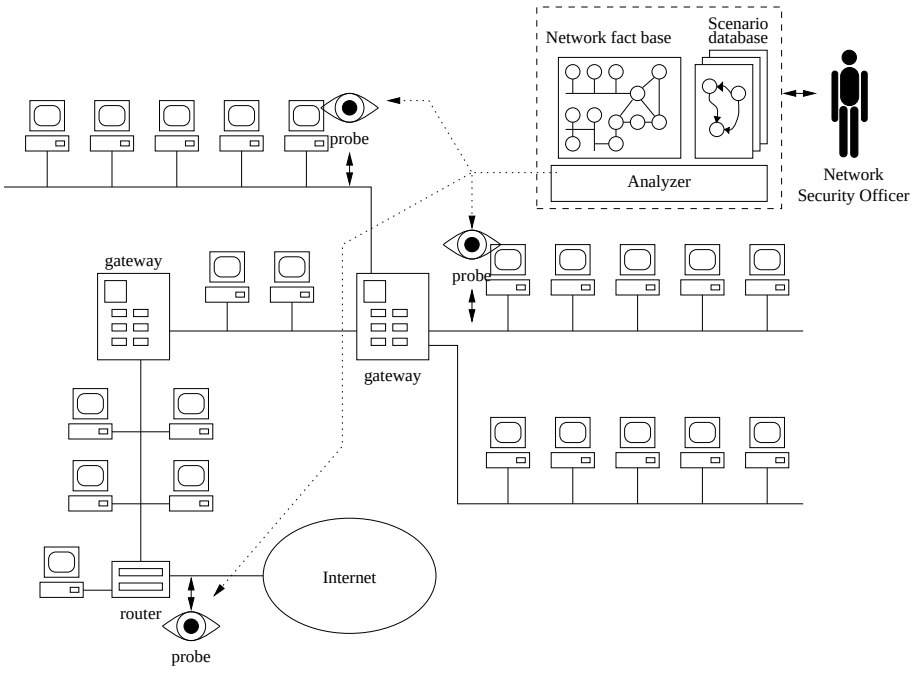


Figure 1: NetSTAT architecture.

the NetSTAT architecture is given in Figure 1. The design of the main NetSTAT components is presented in the following subsections.

3.1 Network Fact Base

The network fact base component stores and manages the security relevant information about a network. The fact base is a stand-alone application that is used by the Network Security Officer to construct, insert, and browse the data about the network being protected. It contains information about the network topology and the network services deployed.

The network topology is a description of the constituent components of the network and how they are connected. The network model underlying the NetSTAT approach uses *interfaces*, *hosts*, and *links* as constituent elements. A network is represented as a hypergraph on the set of interfaces [31]. In this model, interfaces are nodes while hosts and links are edges; that is, hosts and links are modeled as sets of interfaces. This is an original approach that has a number of advantages. Because the model is formal, it provides a well-defined semantics and supports reasoning and automation. This formalization also allows one to model network links based on a shared bus (e.g., Ethernet) in a natural way, by representing the shared bus as a set containing all the interfaces that can access the communication bus. In this way, it is possible to precisely model the concept of network traffic eavesdropping, which is the basis for a number of network-related attacks. In addition, topological properties can be described in an expressive way since hosts and links are treated uniformly as edges of the hypergraph.

The network model is not limited to the description of the connection of elements; each element of the model also has some associated information. For example, hosts have several attributes that characterize the type of hardware and operating system software installed. Note that in this model "host" is a rather general concept: a host is a device that has one or more network interfaces that can be the explicit source and/or destination of network traffic. For example, by this definition, gateways, routers, and printers are considered to be hosts. In the model, links are characterized by their type (e.g., Ethernet), and interfaces are

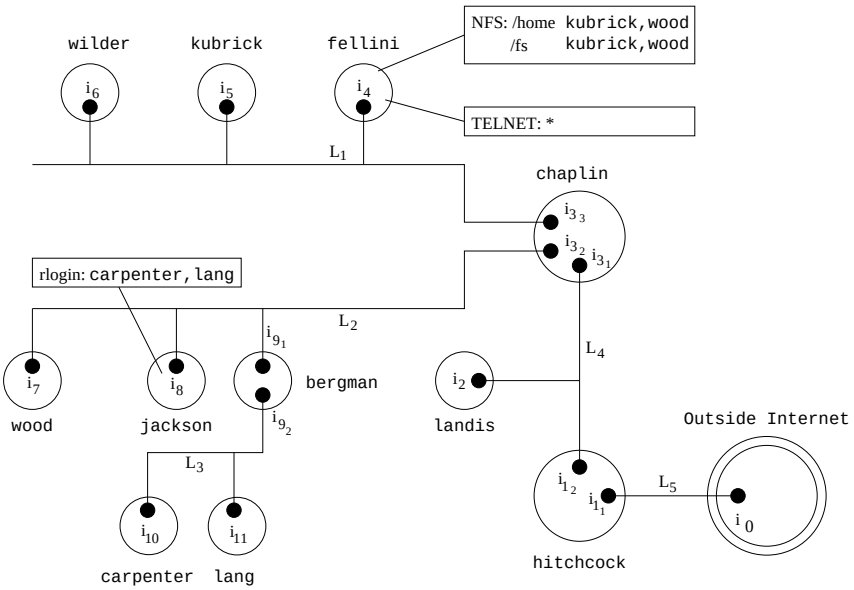


Figure 2: An example network.

characterized by their type and by the corresponding link- and network-level addresses. The current NetSTAT prototype is tailored towards TCP/IP networks, but the underlying model can easily accommodate different protocol suites (e.g., ISO/OSI protocols).

The network services portion of the network fact base contains a description of the services provided by the hosts of a network. Examples of these services are the Network File System (NFS), the Network Information System (NIS), TELNET, FTP, "r" services, etc. The fact base contains a characterization of each service in terms of the network/transport protocol(s) used, the access model (e.g., request/reply), the type of authentication (e.g., address-based, password-based, token-based, or certificate-based), and the level of traffic protection (e.g., encrypted or not). In addition, the network fact base contains information about how services are deployed (i.e., how services are instantiated and accessed over the network).

Figure 2 shows an example network. In the hypergraph representing the network, interfaces are drawn as black dots, hosts are drawn as circles around the corresponding interfaces, and links are represented as lines connecting the interfaces. The example network is composed of five links, twelve hosts, and sixteen interfaces. Hereafter, it is assumed that each interface has a single associated IP address, for example interface i_7 is associated with IP address a_7. The outside network is modeled as a *composite host* (the double circle in the figure). Composite hosts contain a set of interfaces specified by an expression. In this way it is possible to represent in a compact way parts of the network whose details are unknown or not relevant for intrusion detection. In the example of Figure 2 the composite host is used to model the outside Internet and contains all the interfaces and corresponding addresses not in use elsewhere in the inside network.

As far as services are concerned, in the example network host `fellini` is an NFS server exporting file systems `/home` and `/fs` to `kubrick` and `wood`. In addition, `fellini` is a TELNET server for any host. Host `jackson` exports an *rlogin* service to hosts `carpenter` and `lang`.

The Network Security Officer can access the network fact base through a graphic interface. The interface provides functionalities to design the network topology and to define the service infrastructure, as well as to browse the network-related information. Note that while the network model is a complex, formal system, the interface used to manage the network fact base is intuitive and user-friendly. NetSTAT is intended to represent large networks containing a large number of hosts that provide diverse services. For this reason, as an alternative to the graphic interface, all the information can also be inserted and retrieved by using ASCII-

based tools. Thus, scripting languages like Perl or the Bourne Shell can be used to automate the retrieval of information from the network hosts, by querying the network information services (e.g., yellow pages, DNS) or by examining the configuration files of the involved hosts (e.g., `inetd.conf` on UNIX systems). Because large networks by their very nature are subject to changes, the network fact base component is designed to provide procedures that allow the Network Security Officer to verify the internal representation of the network against the actually deployed infrastructure to identify inconsistencies and incomplete or outdated information.

3.2 State Transition Scenario Database

The state transition scenario database is the component that manages the set of state transition representations of the intrusion scenarios to be detected. The state transition scenario database can be executed as a stand-alone application that allows the Network Security Officer to browse and edit state transition diagrams using a friendly graphic interface.

As mentioned in Section 2, the state transition analysis technique was originally developed to model host-based intrusions [13]. For NetSTAT the original STAT technique has been applied to computer networks, and the concepts of state, assertions, transitions, and signature actions have been characterized in a networked environment.

3.2.1 States and Assertions

In network-based state transition analysis the network state includes the currently active connections (for connection oriented services), the state of interactions (for connectionless services), and the values of the network tables (e.g., routing tables, DNS mappings, ARP caches, etc). For instance, both an open connection and a mounted file system are part of the state of the network. A pending DNS request that has not yet been answered is also part of the state, such as the mapping between IP address `128.111.12.13` and the name `hitchcock`. For the application of state transition analysis to networks the original state transition analysis concept of assertion has been extended to include both *static assertions* and *dynamic assertions.*

Static assertions are assertions about a network that can be verified by examining the network fact base; that is, by examining the network topology and the current service configuration. For example, the following assertion:

```
Service s in server.services |
  s.name == "www" and
  s.application.name == "CERN httpd";
```

identifies a service `s` in the set of services provided by host `server` such that the name of the service is `www` and the application providing the service is the CERN http daemon[1]. As a second example, the following assertion:

```
Interface i in gateway.interfaces |
  i.link.type == "Ethernet";
```

denotes an interface of a host, `gateway`, that is connected to an Ethernet link.

These assertions are used to customize state transition representations for particular scenarios (e.g., a particular server and its clients). In practice, they are used to determine the amount of knowledge about the network fact base that each probe must be provided with during configuration procedures.

Dynamic assertions can be verified only by examining the current state of the network. One example is `NFSMounted(filesys, server, client)`, which is true if the specified file system exported by `server` is currently mounted by `client`. Another example is `ConnectionEstablished(addr1, port1, addr2,`

[1]The only (possibly) nonstandard notation used in the assertions is the use of "`|`" for "such that".

port2), which is true if there is an established virtual circuit between the specified addresses and ports. These assertions are used to determine what events relevant to the network state should be monitored by a network probe.

3.2.2 Transitions and Signature Actions

In NetSTAT, signature actions are expressed by leveraging off of an *event model*. In this model events are sequences of messages exchanged over a network.

The basic event is the *link-level message*, or *message* for short. A link-level message is a string of bits that appears on a network link at a specified time. The message is exchanged between two directly connected interfaces. For example, the signature action:

```
Message m {i_x, i_y} |
  m.length > 512;
```

represents a link-level message exchanged between interfaces `i_x` and `i_y` whose size is greater than 512 bytes[2].

Basic events can be abstracted or composed to represent higher-level actions. For example, IP datagrams that are transported from one interface to another in a network are modeled as sequences of link-level messages that represent the intermediate steps in the delivery process. Note that the only directly observable events are link-level messages appearing on specific links. Therefore, the IP datagram "event" is observable by looking at the payload of one of the link-level messages used to deliver the datagram. For example, the signature action:

```
[IPDatagram d] {i_x, i_y} |
  d.options.sourceRoute == true;
```

represents an IP datagram that is delivered from interface `i_x` to interface `i_y` and that has the source route option enabled. This event can be observed by looking at the link-level messages used in datagram delivery along the path(s) from `i_x` to `i_y`. It is also possible to write signature actions that refer to specific link-level messages in the context of datagram delivery. For example, the signature action:

```
Message m in [IPDatagram d] {i_x, i_y} |
  m.dst != i_y;
```

represents a link-level message used during the delivery of an IP datagram such that the link-level destination address of the message is not the final destination interface (i.e., the message is not the last one in the delivery process).

Events representing single UDP [26] datagrams or TCP [28] segments are represented by specifying encapsulation in an IP datagram. For example, the signature action:

```
[IPDatagram d [TCPSegment t]] {i_x, i_y} |
  d.dst == a_x and
  t.dst == 23;
```

denotes the sequence of messages used to deliver a TCP segment encapsulated into an IP datagram such that the destination IP address is `a_x` and the destination TCP port is 23.

TCP virtual circuits are higher-level, composite events. A virtual circuit is identified by the tuple *(source IP address, source TCP port, destination IP address, destination TCP port)* and is composed of two sequences of TCP segments exchanged between two interfaces. Each of these two sequences defines a byte stream. The byte stream is obtained by assembling the payloads of the segments in the corresponding sequence, following

[2]Hereafter, by convention interfaces are denoted by identifiers beginning with `i_`, and IP addresses are denoted by identifiers beginning with `a_`.

the rules of the TCP protocol (e.g., sequencing, retransmission, etc.) [28]. The streams are denoted by `streamToClient` and `streamToServer`.

For example, the signature action:

```
TCPSegment t in [VirtualCircuit c] {i_x, i_y} |
  c.dstIP == a_y and
  c.dstPort == 80 and
  t.syn == false and
  t.ack == true;
```

denotes a TCP segment `t` that belongs to a virtual circuit that has been established between interfaces `i_x` and `i_y`. The virtual circuit has destination IP address `a_y` and destination port 80. The segment has the SYN bit not set and the ACK bit set.

Events at the application level can be either encapsulated in UDP datagrams or can be sent through TCP virtual circuits. In the former case, the application-level event can be referenced by indicating the corresponding datagram and specifying the encapsulation. For example, the signature action:

```
[IPDatagram d [UDPDatagram u [RPC r]]] {i_x, i_y} |
  d.dst == a_y and
  u.dst == 2049 and
  r.type == CALL and
  r.proc == MKDIR;
```

represents an RPC request encapsulated in a UDP datagram representing an NFS command.

In the TCP virtual circuit case, application-level events are extracted by parsing the stream of bytes exchanged over the virtual circuit. The type of application event determines the protocol used to interpret the stream. For example, the following signature action:

```
[c.streamToServer [HTTPRequest r]] |
  r.method == "GET";
```

is an HTTP GET request that is transmitted over a TCP virtual circuit (defined somewhere else as `c`), through the stream directed from the client to the server.

3.3 Probes

The probes are the active intrusion detection components. They monitor the network traffic in specific parts of the network, following the configuration they receive at startup from the analyzer, which is described in the following section. Probes are general purpose intrusion detection systems that can be configured remotely and dynamically following any changes in the modeled attacks or in the implemented security policy. Each probe has the structure shown in Figure 3.

The *filter* module is responsible for filtering the network message stream. Its main task is to select those messages that contribute to signature actions or dynamic assertions used in a state transition scenario from the huge number of messages transmitted over a network link. The filter module is configured remotely by the analyzer. Its configuration can also be updated at run-time to reflect new attack scenarios, or changes in the network configuration. The performance of the filter is of paramount importance, because it has strict real-time constraints for the process of selecting the events that have to be delivered to the inference engine. In the current prototype the filter is implemented using the BSD Packet Filter [21].

The *inference engine* is the actual intrusion detecting component. This module is initialized by the analyzer with a set of state transition information representing attack scenarios (or parts thereof). These attack scenarios are codified in a structure called the inference engine table. At any point during the probe execution, this table consists of snapshots of penetration scenario instances (instantiations), which are not

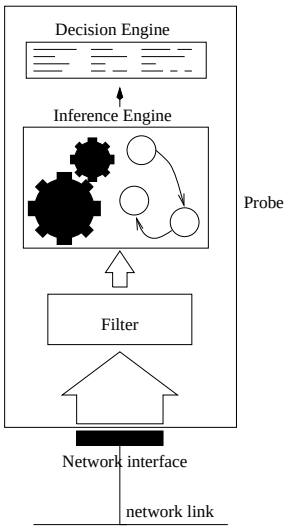


Figure 3: Probe architecture.

yet completed. Each entry contains information about the history of the instantiation, such as the addresses and services involved, the time the attack started, and so on. On the basis of the current active attacks, the event stream provided by the filter is interpreted looking for further evidence of an occurring attack. Evolution of the inference engine state is monitored by the *decision engine*, which is responsible for taking actions based on the outcomes of the inference engine analysis. Some possible actions include informing the Network Security Officer of successful or failed intrusion attempts, alerting the Network Security Officer during the first phases of particularly critical scenarios, suggesting possible actions that can preempt a state transition leading to a compromised state, or playing an active role in protecting the network (e.g., by injecting datagrams that reset network connections.)

Probes are autonomous intrusion detection components. If a single probe is able to detect all the steps involved in an attack then the probe does not need to interact with any other probe or with the analyzer. Interaction is needed whenever different parts of an intrusion can be detected only by probes monitoring different subparts of the network. In this case, it is the analyzer's task to decompose an intrusion scenario into subscenarios such that each can be detected by a single probe. The decision engine procedures associated with these scenarios are configured so that when part of a scenario is detected, an event is sent to the probes that are in charge of detecting the other parts of the overall attack. This simple form of forward chaining allows one to detect attacks that involve different (possibly distant) subnetworks.

3.4 Analyzer

The analyzer is a stand-alone application used by the Network Security Officer to analyze and instrument a network for the detection of a number of selected attacks. It takes as input the network fact base and a state transition scenario database and determines:

- which events have to be monitored; only the events that are relevant to the modeled intrusions must be detected;
- where the events need to be monitored;
- what information about the topology of the network is required to perform detection;

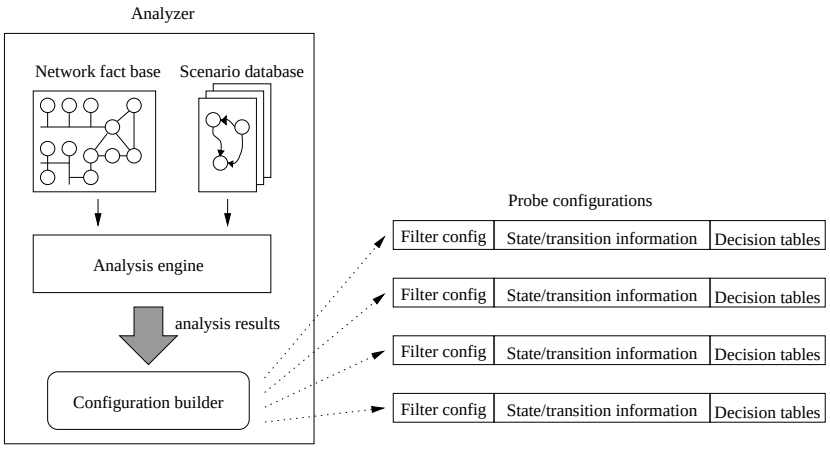


Figure 4: Analyzer architecture.

- what information must be maintained about the state of the network in order to be able to verify state assertions.

Thus, the analyzer component acts as a probe manager that customizes a number of general-purpose probes using an automated process based on a formal description of the network to be protected and of the attacks to be detected. This customization takes the form of a set of probe configurations. Each probe configuration specifies the positioning of a probe, the set of events to be monitored, and a description of the intrusions that the probe should detect. These intrusion scenarios are customized for the particular subnetwork the probe is monitoring, which focuses the scanning and reduces the overhead.

The analyzer is composed of several modules (see Figure 4). The network fact base and the state transition scenario database components are used as internal modules for the selection and presentation of a particular network and a selected set of state transition scenarios. The *analysis engine* uses the data contained in the network fact base and the state transition scenario database to customize the selected attacks for the particular network under exam. For example, if one scenario describes an attack that exploits the trust relationship between a server and a client, that scenario will be customized for the particular services deployed in the actual network, such as an NFS server and each of its clients. This customization allows one to instantiate an attack in a particular context. Using the description of the topology of that context it is then possible to identify what the sufficient conditions for detection are or if a particular attack simply cannot be detected for the given network configuration.

Once the attack scenarios contained in the state transition scenario database have been customized with respect to the given network, another module, called the *configuration builder*, translates the results of the analysis engine into a set of configurations to be sent to the different probes. Each configuration contains a filter configuration, a set of state transition information, and the corresponding decision tables that customize the probe's decision engine.

3.5 The Architecture at Work

The architecture of NetSTAT supports a well-defined configuration and deployment process. Firstly the Network Security Officer builds a database of attack scenarios. This database may include pre-modeled well-known scenarios or may be extended by the Network Security Officer following his/her perception of what an attack is, according to the local security policy. The attack scenario database is created and maintained using the state transition scenario database as a stand-alone application. Next, the Network Security Officer builds a network fact base describing the network to be protected. This task can be achieved by a combination

of methods ranging from manual data introduction to automated information retrieval. This operation is performed by using the network fact base as a stand-alone application. In the next phase, the Network Security Officer invokes the analyzer. Using the integrated network fact base and state transition scenario database modules the Network Security Officer selects a particular network description and a set of scenarios that have to be detected. Again, this choice is dictated by the particular level of security that has to be achieved and by the network security policy in force. The Network Security Officer then starts the analysis and customization process. This process is mostly automated, but in some circumstances it may require some interaction with the Network Security Officer, for example to manage situations where some attack scenarios cannot be detected. When the analysis is completed, the probe configurations are created and sent to the probes installed across the network.

4 Examples

This section presents some network attack scenarios, demonstrates how they are described using network-based state transition analysis, and shows how the necessary probes for detecting the attacks can be placed and configured by using the analyzer tool. The attacks discussed in this section are analyzed under the assumption that attackers are not able to block traffic, which is a reasonable assumption since blocking attacks are difficult to implement. This assumption greatly simplifies the analysis and allows for more efficient probe configurations[3].

4.1 IP spoofing

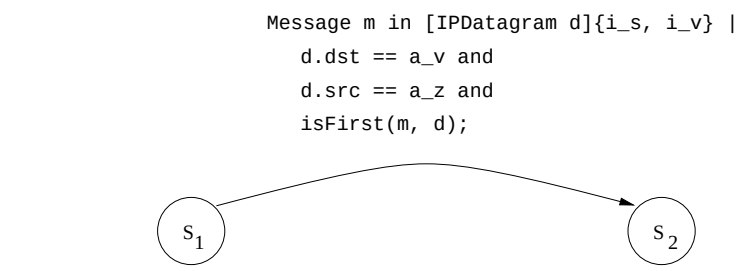
In an IP spoofing attack a host A tries to impersonate another host B towards a third host C by producing a *spoofed* IP datagram directed to one of the addresses of C. The source address of this datagram is the IP address associated with one of the interfaces of B even though the datagram is initially generated by one of the interfaces of A. When C receives the spoofed datagram it processes it as if it were originally sent by B. This attack is the basis for several other attacks against services that use the source IP address of a request as the credentials for the client. IP spoofing succeeds because the original version of the IP protocol [27] does not provide any form of strong authentication. Recently, a new version of the IP protocol has been designed [7, 19]. The new IP protocol includes mechanisms for origin authentication [17] and protection of payload [18]. Although the new version of the IP protocol provides much stronger security, the current version of the Internet Protocol is still widely used, and attacks based on IP spoofing are frequent.

The modeling language of NetSTAT allows one to represent attacks in many different ways. In general, the way an attack is described may influence the effectiveness of detection. In the following subsections two different characterizations of the IP spoofing attack are described. Each version is analyzed in detail on a simple example network and the resulting probe placement and configuration are presented. In the last subsection, a variation of the IP spoofing attack that exploits UDP-based services is analyzed with respect to a more complex network.

4.1.1 A simple description

Attack The most intuitive, direct way to describe the IP spoofing attack in NetSTAT is shown in Figure 5. The assertions of state S_1 define six entities in the attack scenario. The first clause identifies a host, `spoofer`, in the set of hosts contained in the modeled network. The second clause identifies one of `spoofer`'s interfaces, `i_s`. The next two clauses identify another host, `victim`, that is different from `spoofer` and one of `victim`'s interfaces, `i_v`. The fifth clause states that IP address `a_v` is associated with interface `i_v`. The last clause states that IP address `a_z` is not one of those associated with `spoofer`. Note that the identifier `Network` is used to denote the network represented in the network fact base.

[3]Although blocking attacks are not considered in this paper, there is nothing inherent in the network-based state transition analysis approach that would not allow their inclusion.




```
Host spoofer in Network.hosts;
Interface i_s in spoofer.interfaces;
Host victim in Network.hosts |
   victim != spoofer;
Interface i_v in victim.interfaces;
IPAddress a_v in i_v.ipAddresses;
IPAddress a_z in Network.ipAddresses |
   not spoofer.ipAddresses.contains(a_z);
```

Figure 5: A first description of the IP spoofing attack.

The signature action is represented by a link-level message `m` belonging to the sequence of messages generated by the delivery of an IP datagram `d` between endpoint interfaces `i_s` and `i_v`. The IP datagram has `a_v` as its destination address and `a_z` as its source address. The message `m` is the first message in the delivery sequence, i.e., its source interface is `i_s`. More specifically, this means that the IP datagram originates at one of the `spoofer`'s interfaces but has a source IP address that is not associated with host `spoofer`, and therefore the datagram is considered spoofed. The second state (S_2) is a "compromised" state.

Analysis To demonstrate the analysis technique, this attack is analyzed with respect to the simple network shown in Figure 6. The network is composed of four hosts (`lucas`, `gilliam`, `coppola`, and `cameron`), two links (L_1 and L_2), and five interfaces (i_0, i_1, i_2, i_3, i_4). Each interface has a corresponding IP address, e.g., interface i_0 is associated with IP address a_0.

NetSTAT's analysis engine takes as input the description of the attack and the fact base of the network and tries to determine what events must be monitored and where these events can be monitored to detect the attack. To achieve this goal, the analysis engine generates all the possible instantiations of the selected attack scenario with respect to the network under exam. In the IP spoofing attack, the engine first generates all the possible combinations of attackers and victims (and corresponding interfaces and IP addresses). The results are shown in Table 1. For example, consider the fourth row. This row identifies a scenario where the `spoofer` is host `lucas` sending a spoofed datagram from its interface i_0 to interface i_4 of host `cameron`, which plays the role of the `victim`. The destination address for the IP datagram is `cameron`'s address a_4 while the source IP address can be any of a_1, a_2, a_3, or a_4 (a_0 is `lucas`' IP address and therefore is not

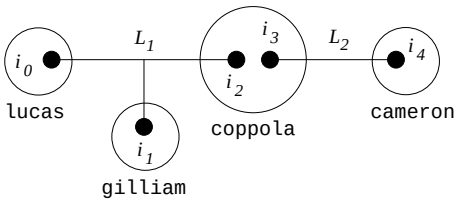


Figure 6: A simple network.

Scenarios					
spoofer	i_s	a_z	victim	i_v	a_v
lucas	i_0	a_{1-4}	gilliam	i_1	a_1
lucas	i_0	a_{1-4}	coppola	i_2	a_2
lucas	i_0	a_{1-4}	coppola	i_3	a_3
lucas	i_0	a_{1-4}	cameron	i_4	a_4
gilliam	i_1	$a_{0,2-4}$	lucas	i_0	a_0
gilliam	i_1	$a_{0,2-4}$	coppola	i_2	a_2
gilliam	i_1	$a_{0,2-4}$	coppola	i_3	a_3
gilliam	i_1	$a_{0,2-4}$	cameron	i_4	a_4
coppola	i_2	$a_{0-1,4}$	lucas	i_0	a_0
coppola	i_2	$a_{0-1,4}$	gilliam	i_1	a_1
coppola	i_2	$a_{0-1,4}$	cameron	i_4	a_4
coppola	i_3	$a_{0-1,4}$	lucas	i_0	a_0
coppola	i_3	$a_{0-1,4}$	gilliam	i_1	a_1
coppola	i_3	$a_{0-1,4}$	cameron	i_4	a_4
cameron	i_4	a_{0-3}	lucas	i_0	a_0
cameron	i_4	a_{0-3}	gilliam	i_1	a_1
cameron	i_4	a_{0-3}	coppola	i_2	a_2
cameron	i_4	a_{0-3}	coppola	i_3	a_3

Table 1: Possible IP spoofing attack scenarios.

considered).

For each of the scenarios of Table 1, the analysis engine generates all the possible datagrams between the interface of the spoofer and the interface of the victim[4]. In practice, the engine finds all the paths between the interfaces defined by the scenario and, for each path, it generates the sequence of link-level messages that would be used to deliver the spoofed IP datagram. For example, in the scenario where `lucas` is attacking `cameron` the spoofed datagrams would be delivered through two link-level messages. The first message is from `lucas`' interface i_0 to `coppola`'s interface i_2. The second message is from `coppola`'s interface i_3 to `cameron`'s interface i_4.

The analyzer identifies which are the relevant messages by applying the predicate contained in the clause of the signature action of Figure 5 to each message. More specifically, the analyzer has to determine if the message is the first in the message sequence used to deliver the datagram. For example, in the scenario previously considered (`lucas` attacking `cameron`) only the first message satisfies the predicate. Therefore, to detect an IP spoofing attack in this particular scenario, one needs to place a probe on the link where this message appears (i.e., L_1), and the probe is instructed to look for messages from i_0 to i_2, such that the source IP address is one of a_1, a_2, a_3, a_4 (denoted a_{1-4}), and the destination IP address is a_4. This process

[4]More precisely, all the possible combinations of the datagram fields used in the attack representation (e.g., source/destination IP address).

Scenarios		*Messages*
spoofer	victim	
lucas	gilliam	$\underline{\langle i_0, i_1, \langle a_{1-4}, a_1 \rangle\rangle}$
lucas	coppola	$\underline{\langle i_0, i_2, \langle a_{1-4}, a_{2-3} \rangle\rangle}$
lucas	cameron	$\underline{\langle i_0, i_2, \langle a_{1-4}, a_4 \rangle\rangle}, \langle i_3, i_4, \langle a_{1-4}, a_4 \rangle\rangle$
gilliam	lucas	$\underline{\langle i_1, i_0, \langle a_{0,2-4}, a_0 \rangle\rangle}$
gilliam	coppola	$\underline{\langle i_1, i_2, \langle a_{0,2-4}, a_{2-3} \rangle\rangle}$
gilliam	cameron	$\underline{\langle i_1, i_2, \langle a_{0,2-4}, a_4 \rangle\rangle}, \langle i_3, i_4, \langle a_{0,2-4}, a_4 \rangle\rangle$
coppola	lucas	$\underline{\langle i_2, i_0, \langle a_{0-1,4}, a_0 \rangle\rangle}$
coppola	gilliam	$\underline{\langle i_2, i_1, \langle a_{0-1,4}, a_1 \rangle\rangle}$
coppola	cameron	$\underline{\langle i_3, i_4, \langle a_{0-1,4}, a_4 \rangle\rangle}$
cameron	lucas	$\underline{\langle i_4, i_3, \langle a_{0-3}, a_0 \rangle\rangle}, \langle i_2, i_0, \langle a_{0-3}, a_0 \rangle\rangle$
cameron	gilliam	$\underline{\langle i_4, i_3, \langle a_{0-3}, a_1 \rangle\rangle}, \langle i_2, i_1, \langle a_{0-3}, a_1 \rangle\rangle$
cameron	coppola	$\underline{\langle i_4, i_3, \langle a_{0-3}, a_{2-3} \rangle\rangle}$

Table 2: Messages generated for all scenarios. Message are represented in the format $\langle i_s, i_d, \langle a_s, a_d \rangle\rangle$ where i_s is the source interface, i_d is the destination interface, and a_s and a_d are respectively the source and destination IP addresses of the encapsulated IP datagram. Messages that satisfy the clause in the signature action of Figure 5 are underlined.

is repeated for each scenario. Table 2 shows all the datagrams (and the corresponding messages) for every scenario. The messages that satisfy the signature action's clause are underlined.

The detection of the "first" message can be a tricky task. If the source interface belongs to a host with a single interface it is sufficient to place a probe on the link where the message appears. However, if the message is generated from the interface of a multi-interface host it is necessary to verify that the message is the first in the delivery process, otherwise messages forwarded during the delivery process of a perfectly legal datagram would be flagged as part of an IP spoofing attack. For example, consider message $\langle i_3, i_4, \langle a_0, a_4 \rangle\rangle$; the message is sent from `coppola` to `cameron` and encapsulates an IP datagram from `lucas` to `cameron`. This message could represent either the second step in the delivery of a normal IP datagram or the result of `coppola`'s attempt to spoof `lucas`' address with respect to `cameron`. In this case, the only way to detect the attack is to place a probe on link L_1 *and* a probe on link L_2. When a message directed to `coppola` and intended for `cameron` appears on link L_1, the corresponding probe sends a message to the probe on link L_2 so that the subsequent message appearing on L_2 can be ignored. This solution creates an enormous traffic overload because every message that is forwarded by `coppola` requires synchronizations between the two probes. The resulting probe configurations are shown in Table 3.

4.1.2 An "improved" description

The IP spoofing attack can be modeled in a different way that puts a "hint" in the description of the attack that allows for smarter probe placement. This "improved" description takes advantage of NetSTAT's modeling language ability to explicitly refer to the target network topology.

	Probe on link L_1			
	Source interface	Destination interface	Source IP address	Destination IP address
	i_0	i_1	a_{1-4}	a_1
(1)	i_0	i_2	a_0	a_4
	i_0	i_2	a_{1-4}	a_2
	i_0	i_2	a_{1-4}	a_3
	i_0	i_2	a_{1-4}	a_4
	i_1	i_0	$a_{0,2-4}$	a_0
(2)	i_1	i_2	a_1	a_4
	i_1	i_2	$a_{0,2-4}$	a_2
	i_1	i_2	$a_{0,2-4}$	a_3
	i_1	i_2	$a_{0,2-4}$	a_4
	i_2	i_0	a_{0-1}	a_0
(3)	i_2	i_0	a_4	a_0
	i_2	i_1	a_{0-1}	a_1
(4)	i_2	i_1	a_4	a_1

Messages (1) and (2) are not part of an attack but the probe on link L_2 must be notified about them. Messages (3) and (4) are part of an IP spoofing attack if the probe on link L_2 did not send any notification about events (5) and (6) respectively.

	Probe on link L_2			
	Source interface	Destination interface	Source IP address	Destination IP address
	i_4	i_3	a_{0-3}	a_0
(5)	i_4	i_3	a_4	a_0
(6)	i_4	i_3	a_4	a_1
	i_4	i_3	a_{0-3}	a_1
	i_4	i_3	a_{0-3}	a_2
	i_4	i_3	a_{0-3}	a_3
(7)	i_3	i_4	a_0	a_4
(8)	i_3	i_4	a_1	a_4

Messages (5) and (6) are used by the probe on link L_1 to detect `coppola`'s attacks. Messages (7) and (8) are part of an IP spoofing attack if the probe on link L_1 did not send any notification about events (1) and (2) respectively.

Table 3: Probe configurations.

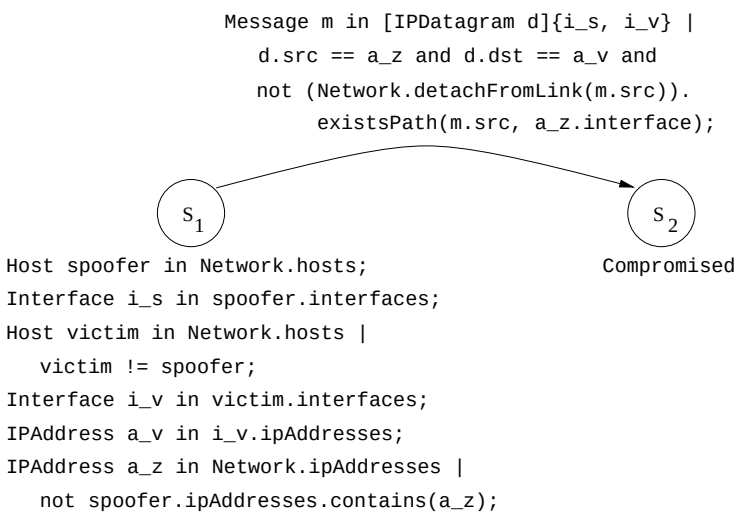


Figure 7: A second way to describe the IP spoofing attack.

Attack As described in the previous section, a spoofed IP datagram is one such that the message that originates the datagram is sent by a host that is not associated with the source IP address of the datagram. Another way to model the IP spoofing attack is to characterize the fact that one or more of the messages used to deliver an IP datagram cannot "legally" appear on some paths. Consider the state transition diagram of Figure 7. The assertions associated with state S_1 are analogous to the previous case. They characterize the attacker and attacked hosts together with their interfaces and the impersonated address. The signature, again, is a link-level message `m` encapsulating an IP datagram `d`, and `d` has `a_z` as its source IP address and `a_v` as its destination IP address. The difference from the previous description occurs in the definition of what it means to be spoofed. In this description the datagram can be considered spoofed if there is no path between the message source interface (`m.src`) and the interface associated with the source IP address (`a_z.interface`) in the network obtained by removing the message's source interface from its link (`Network.detachFromLink(m.src)`). If there is no such path, the datagram delivery process could not have "legally" reached the current step and one can deduce that the datagram was spoofed. The second state (S_2) is a "compromised" state.

Analysis The analysis of this attack follows the steps that have been described in the previous section. First, the analysis engine determines the attack scenarios with respect to the selected target network (see Figure 6); the state assertions have not changed with respect to the previous IP spoofing attack description and therefore the scenarios are still those listed in Table 1. Next, the messages used to deliver the spoofed IP datagrams are generated and they are matched against the signature action clause of Figure 7. Table 4 shows the matching messages. Table 5 shows one possible configuration for the probes.

As a result of the "improved" description, for some attack scenarios one can place a probe in more than one place. For example, to detect `cameron` attempting to impersonate `lucas` with respect to `gilliam` one can place a probe either on link L_2 looking for message $\langle i_4, i_3, \langle a_0, a_1 \rangle \rangle$ or on link L_1 looking for message $\langle i_2, i_1, \langle a_0, a_1 \rangle \rangle$. Therefore, probes are not necessarily required to be placed on the same link as the attacker host. This advantage is not very evident in the example network of Figure 6, but it becomes an asset in large networks. In addition, probes do not need to interact to detect the attacks, reducing the traffic overhead to zero.

This solution also has some disadvantages. First, some of the attacks are no longer detected, i.e., host `coppola` impersonating `cameron` with respect to `gilliam` and `lucas`, and `coppola` impersonating `gilliam`

Scenarios		*Messages*
spoofer	victim	
lucas	gilliam	$\underline{\langle i_0, i_1, \langle a_{1-4}, a_1 \rangle\rangle}$
lucas	coppola	$\underline{\langle i_0, i_2, \langle a_{1-4}, a_{2-3} \rangle\rangle}$
lucas	cameron	$\underline{\langle i_0, i_2, \langle a_{1-3}, a_4 \rangle\rangle}, \langle i_3, i_4, \langle a_{1-3}, a_4 \rangle\rangle$
		$\underline{\langle i_0, i_2, \langle a_4, a_4 \rangle\rangle}, \underline{\langle i_3, i_4, \langle a_4, a_4 \rangle\rangle}$
gilliam	lucas	$\underline{\langle i_1, i_0, \langle a_{0,2-4}, a_0 \rangle\rangle}$
gilliam	coppola	$\underline{\langle i_1, i_2, \langle a_{0,2-4}, a_{2-3} \rangle\rangle}$
gilliam	cameron	$\underline{\langle i_1, i_2, \langle a_{0,2-3}, a_4 \rangle\rangle}, \langle i_3, i_4, \langle a_{0,2-3}, a_4 \rangle\rangle$
		$\underline{\langle i_1, i_2, \langle a_4, a_4 \rangle\rangle}, \underline{\langle i_3, i_4, \langle a_4, a_4 \rangle\rangle}$
coppola	lucas	$\underline{\langle i_2, i_0, \langle a_{0-1}, a_0 \rangle\rangle}$
		$\langle i_2, i_0, \langle a_4, a_0 \rangle\rangle$
coppola	gilliam	$\underline{\langle i_2, i_1, \langle a_{0-1}, a_1 \rangle\rangle}$
		$\langle i_2, i_1, \langle a_4, a_1 \rangle\rangle$
coppola	cameron	$\langle i_3, i_4, \langle a_{0-1}, a_4 \rangle\rangle$
		$\underline{\langle i_3, i_4, \langle a_4, a_4 \rangle\rangle}$
cameron	lucas	$\underline{\langle i_4, i_3, \langle a_{2-3}, a_0 \rangle\rangle}, \langle i_2, i_0, \langle a_{2-3}, a_0 \rangle\rangle$
		$\underline{\langle i_4, i_3, \langle a_{0-1}, a_0 \rangle\rangle}, \underline{\langle i_2, i_0, \langle a_{0-1}, a_0 \rangle\rangle}$
cameron	gilliam	$\underline{\langle i_4, i_3, \langle a_{2-3}, a_1 \rangle\rangle}, \langle i_2, i_1, \langle a_{2-3}, a_1 \rangle\rangle$
		$\underline{\langle i_4, i_3, \langle a_{0-1}, a_1 \rangle\rangle}, \underline{\langle i_2, i_1, \langle a_{0-1}, a_1 \rangle\rangle}$
cameron	coppola	$\underline{\langle i_4, i_3, \langle a_{0-3}, a_{2-3} \rangle\rangle}$

Table 4: Messages generated by the analysis of the attack of Figure 7 with respect to the network of Figure 6. Messages that satisfy the clause in the signature action of Figure 7 are underlined.

Probe on link L_1			
Source interface	Destination interface	Source IP address	Destination IP address
i_0	i_1	a_{1-4}	a_1
i_0	i_2	a_{1-4}	a_2
i_0	i_2	a_{1-4}	a_3
i_0	i_2	a_{1-4}	a_4
i_1	i_0	$a_{0,2-4}$	a_0
i_1	i_2	$a_{0,2-4}$	a_2
i_1	i_2	$a_{0,2-4}$	a_3
i_1	i_2	$a_{0,2-4}$	a_4
i_2	i_0	a_{0-1}	a_0
i_2	i_1	a_{0-1}	a_1

Probe on link L_2			
Source interface	Destination interface	Source IP address	Destination IP address
i_3	i_4	a_4	a_4
i_4	i_3	a_{0-3}	a_0
i_4	i_3	a_{0-3}	a_1
i_4	i_3	a_{0-3}	a_2
i_4	i_3	a_{0-3}	a_3

Table 5: Probe configurations resulting from the analysis of the attack of Figure 7 with respect to the network of Figure 6.

or `lucas` with respect to `cameron`. Second, in some cases it is not possible to identify the host that performed the spoofing. For example, from message $\langle i_2, i_1, \langle a_0, a_1 \rangle \rangle$ one cannot determine if the attack is from `coppola` or `cameron`.

One may also note that despite the "hint" in the attack described in Figure 7 the number of probes required to detect the attacks is not reduced with respect to the attack discussed in Section 4.1.1. In fact, in both cases the goal of the analysis is to detect *any* possible attempt of spoofing and therefore there *must* be a probe on every link. In most situations, however, one is interested in a limited set of possible "victims" or spoofed addresses because IP spoofing is used to gain privileged access to a service that authenticates its clients on the basis of the IP address. In the next subsection the IP spoofing description discussed above is applied to detect attacks to UDP-based services.

4.1.3 UDP spoofing

Attack The example attack considered is an active UDP spoofing attack. In this scenario an attacker tries to access a UDP-based service exported by a server by pretending to be one of its trusted clients, that is, by sending a forged UDP-over-IP datagram that contains the IP address of one of the authorized clients as the source address. The attack is described in Figure 8. In this case, there is a subnetwork, called `ProtectedNetwork`, that contains the hosts that have to be protected against spoofing attacks. The starting state (S_1) is characterized by assertions that define the hosts, interfaces, addresses, and services involved in the attack. The first assertion states that the attacked host `victim` belongs to the protected network. The second assertion states that there is a service `x` in the set of services provided by `victim` such that the transport protocol used is UDP, and service authentication is based on the IP address of the client. The third assertion states that `a_v` is one of the IP addresses where the service is available. The fourth assertion says that `a_t` is one of the addresses that the service considers as "trusted". The next two assertions characterize

```
Message m in [IPDatagram d [UDPDatagram u]]{i_s, a_v.interface} |
   d.src == a_t and d.dst == a_v and
   u.dst == x.port and
   not (Network.detachFromLink(m.src)).existsPath(m.src, a_t.interface);
```


```
Host victim in ProtectedNetwork.hosts;                    Compromised
Service x in victim.services |
   x.protocol == "UDP" and
   x.authentication == "IPaddress";
IPAddress a_v in x.ipAddresses;
IPAddress a_t in x.trustedAddr;
Host spoofer in Network.hosts |
   spoofer != victim and
   (spoofer.ipAddresses.intersect(x.trustedAddr)).isEmpty;
Interface i_s in spoofer.interfaces;
```

Figure 8: An IP spoofing attack against a host in the protected network.

the attacker. In particular, the fifth assertion states that there exists a host `spoofer` that is different from `victim` and that is not associated with any of the trusted IP addresses[5]. The sixth assertion states that `i_s` is one of the `spoofer`'s interfaces.

The signature action is a spoofed service request. Actually the signature action is a link-level message `m` that belongs to the sequence of messages used to deliver an IP datagram from interface `i_s` to the interface associated with the address of the attacked host. The IP datagram enclosed in the message has source address `a_t` and destination address `a_v`. The IP datagram encloses a UDP datagram, whose destination port is the port used by service `x`. In addition, the message is such that, if one considers the network obtained by removing the message source interface from the corresponding link (i.e., `Network.detachFromLink(m.src)`), there is no path between the interface corresponding to the datagram IP source address and the link-level message source interface. The second state (S_2) is a "compromised" state.

Analysis The attack will be analyzed with respect to the example network of Figure 2. The protected network is composed of the internal hosts, that is all of the represented hosts with the exception of the composite host representing the outside Internet.

The analysis of the attack starts by identifying the possible scenarios in the context of the modeled network. Thus, the analysis engine determines all the possible combinations of victim host, attacked service, spoofed address, and attacker. In the network under exam the only possible victim is `fellini`, the service under attack is NFS, and the trusted addresses are those of `kubrick` and `wood`, a_5 and a_7 respectively. The attacker can be any host that is not `kubrick` or `wood`. The possible scenarios are listed in Table 6.

The next step in the analysis is to determine where the events associated with the signature action can be detected. For each scenario the analysis engine generates the messages used to perform the attack. Then, each message is matched against the predicate in the signature action of Figure 8. The messages that satisfy the predicate are candidates for being part of the detection of the scenario. For example, consider the scenario where `carpenter` is attacking `fellini` by pretending to be `wood`. In this case, the spoofed datagram is generated from interface i_{10} and delivered through three messages to `fellini`'s interface i_4.

[5] One may decide to also consider attacks where a trusted client spoofs another trusted client.

Scenarios					
victim	x	a_v	a_t	spoofer	i_s
fellini	NFS	a_4	a_5	Outside	i_0
fellini	NFS	a_4	a_7	Outside	i_0
fellini	NFS	a_4	a_5	hitchcock	$i_{1_1,1_2}$
fellini	NFS	a_4	a_7	hitchcock	$i_{1_1,1_2}$
fellini	NFS	a_4	a_5	landis	i_2
fellini	NFS	a_4	a_7	landis	i_2
fellini	NFS	a_4	a_5	chaplin	$i_{3_1-3_3}$
fellini	NFS	a_4	a_7	chaplin	$i_{3_1-3_3}$
fellini	NFS	a_4	a_5	wilder	i_6
fellini	NFS	a_4	a_7	wilder	i_6
fellini	NFS	a_4	a_5	jackson	i_8
fellini	NFS	a_4	a_7	jackson	i_8
fellini	NFS	a_4	a_5	bergman	$i_{9_1,9_2}$
fellini	NFS	a_4	a_7	bergman	$i_{9_1,9_2}$
fellini	NFS	a_4	a_5	carpenter	i_{10}
fellini	NFS	a_4	a_7	carpenter	i_{10}
fellini	NFS	a_4	a_5	lang	i_{11}
fellini	NFS	a_4	a_7	lang	i_{11}

Table 6: Possible scenarios for the UDP spoofing attack.

The first message is between `carpenter` and `bergman`, the second one is between `bergman` and `chaplin`, and the third one is between `chaplin` and `fellini`. Of these three messages only the first two satisfy the predicate of the signature action. Therefore, to detect this particular scenario one needs either a probe on L_3 looking for link-level messages from `carpenter`'s interface i_{10} to `bergman`'s interface i_{9_2}, or a probe on L_2 looking for messages from `bergman`'s interface i_{9_1} to `chaplin`'s interface i_{3_2}. In both cases, the IP source address is a_7, the destination IP address is a_4, and the destination UDP port is the one used by the NFS service. The complete results of the analysis are presented in Table 7. By analyzing all the scenarios, one finds that in order to detect all possible spoofing attacks it is necessary to set up probes on links L_1, L_2, and L_4. Their configuration is shown in Table 8. Note that the scenario where `chaplin` attacks `fellini` by pretending to be `wood` cannot be detected. This scenario could be detected if the signature action of the attack would be modeled analogously to Figure 5, but in this case the analysis would require one to detect all the "first" messages. As a consequence, one would need a probe on every link and probes would have to synchronize on NFS requests that go through `chaplin`, `hitchcock`, or `bergman`.

4.2 TCP spoofing

Attack In a TCP spoofing attack, the attacker host A tries to impersonate a host B (actually one of its addresses) towards a third host C for a whole TCP session. As in the UDP attack described in Section 4.1.3, impersonation aims at exploiting some trust relationship between the attacked and the spoofed host. For example, the *rsh* and *rlogin* protocols allow sessions to be established without the need to provide any password if the connecting host name (or address) is listed in a user's or system's configuration file. That is, the name (or address) of the host is used as the only means of authentication. This weak authentication procedure is often misused to gain unauthorized access. This attack has been discussed in [3, 22].

In order to successfully impersonate host B, A must be able to:

1. prevent any TCP segment sent by C during the session from being processed by the impersonated host B;
2. determine the first sequence number produced during connection setup by the attacked host C.

The first condition must be satisfied because if replies to spoofed segments are delivered to B, then B will deduce that some error occurred and it will send a reset segment to C, resulting in connection shutdown. TCP segments can be prevented from being processed by the spoofed host if either (i) A is a forwarder of every segment of the session, that is A is a gateway and it is able to block traffic, or (ii) B is "deaf" (or responding slowly) to traffic coming from C. As discussed earlier, it is assumed that the former case cannot hold; yet, the latter case can be easily achieved by means of a flooding attack that makes B unable to process TCP/IP traffic.

The sequence number condition is posed by the three-way handshake used to establish a TCP connection. The TCP spoofing attack starts with host A sending a spoofed first segment pretending to be from B and requesting a connection to C. The segment has the SYN flag set and contains an initial sequence number seq_B^0 (determined by the attacker). Host C replies by sending B a segment containing C's initial sequence number seq_C^0 and the acknowledgment of the previous sequence number $ack_C^0 = seq_B^0 + 1$; the segment is marked with both the SYN and ACK flags. To complete the handshake, host A must send C a segment with the ACK flag set that pretends to come from B and contains the sequence number $seq_B^1 = seq_B^0 + 1$ and the acknowledgment number $ack_B^1 = seq_C^0 + 1$. Therefore, A must know seq_C^0, in order to be able to complete the handshake successfully. There are at least two ways to achieve this: the attacker guesses the right number by interpolating the function used to generate the sequence number as described in [22], or the attacker sniffs C's reply. In both cases, the TCP spoofing attack can be modeled as in Figure 9.

The starting state (S_1) is characterized by assertions that define the entities involved in the attack. Host `victim` provides a TCP-based, address-authenticated service `x`. The service is available at address `a_v`. IP address `a_t` is one of the service's trusted addresses. The trust relationship is exploited by host `spoofer` that is different from `victim` and it is not associated with any of the trusted addresses.

Scenarios			*Messages*
spoofer	victim	a_t	
Outside	fellini	a_5	$\underline{\langle i_0, i_{1_1}, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{1_2}, i_{3_1}, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{3_3}, i_4, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}$
Outside	fellini	a_7	$\underline{\langle i_0, i_{1_1}, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{1_2}, i_{3_1}, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}, \langle i_{3_3}, i_4, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle$
hitchcock	fellini	a_5	$\underline{\langle i_{1_2}, i_{3_1}, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{3_3}, i_4, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}$
hitchcock	fellini	a_7	$\underline{\langle i_{1_2}, i_{3_1}, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}, \langle i_{3_3}, i_4, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle$
landis	fellini	a_5	$\underline{\langle i_2, i_{3_1}, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{3_3}, i_4, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}$
landis	fellini	a_7	$\underline{\langle i_2, i_{3_1}, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}, \langle i_{3_3}, i_4, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle$
chaplin	fellini	a_5	$\underline{\langle i_{3_3}, i_4, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}$
chaplin	fellini	a_7	$\langle i_{3_3}, i_4, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle$
wilder	fellini	a_5	$\underline{\langle i_6, i_4, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}$
wilder	fellini	a_7	$\underline{\langle i_6, i_4, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}$
jackson	fellini	a_5	$\underline{\langle i_8, i_{3_2}, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{3_3}, i_4, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}$
jackson	fellini	a_7	$\underline{\langle i_8, i_{3_2}, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}, \langle i_{3_3}, i_4, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle$
bergman	fellini	a_5	$\underline{\langle i_{9_1}, i_{3_2}, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{3_3}, i_4, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}$
bergman	fellini	a_7	$\underline{\langle i_{9_1}, i_{3_2}, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}, \langle i_{3_3}, i_4, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle$
carpenter	fellini	a_5	$\underline{\langle i_{10}, i_{9_2}, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{9_1}, i_{3_2}, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{3_3}, i_4, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}$
carpenter	fellini	a_7	$\underline{\langle i_{10}, i_{9_2}, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{9_1}, i_{3_2}, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}, \langle i_{3_3}, i_4, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle$
lang	fellini	a_5	$\underline{\langle i_{11}, i_{9_2}, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{9_1}, i_{3_2}, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{3_3}, i_4, \langle a_5, a_4, \langle *, p\rangle\rangle\rangle}$
lang	fellini	a_7	$\underline{\langle i_{11}, i_{9_2}, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}, \underline{\langle i_{9_1}, i_{3_2}, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle}, \langle i_{3_3}, i_4, \langle a_7, a_4, \langle *, p\rangle\rangle\rangle$

Table 7: Messages generated by the analysis of the attack of Figure 8 with respect to the network of Figure 2. Message are represented in the format $\langle i_s, i_d, \langle a_s, a_d, \langle p_s, p_d\rangle\rangle\rangle$ where i_s is the source interface, i_d is the destination interface, a_s and a_d are respectively the source and destination IP addresses of the encapsulated IP datagram, and p_s and p_d are the source and destination ports of the encapsulated UDP datagram. In the above messages, p is the port of the NFS service on `fellini`. Messages that satisfy the clause in the signature action of Figure 8 are underlined.

Probe on link L_1					
Source interface	Destination interface	Source IP address	Destination IP address	Source UDP port	Destination UDP port
i_{3_3}	i_4	a_5	a_4	*	p
i_6	i_4	$a_{5,7}$	a_4	*	p

Probe on link L_2					
Source interface	Destination interface	Source IP address	Destination IP address	Source UDP port	Destination UDP port
i_8	i_{3_2}	a_7	a_4	*	p
i_{9_1}	i_{3_2}	a_7	a_4	*	p

Probe on link L_4					
Source interface	Destination interface	Source IP address	Destination IP address	Source UDP port	Destination UDP port
i_{1_2}	i_{3_1}	a_7	a_4	*	p
i_2	i_{3_1}	a_7	a_4	*	p

Table 8: Probe configurations for the UDP spoofing attack.

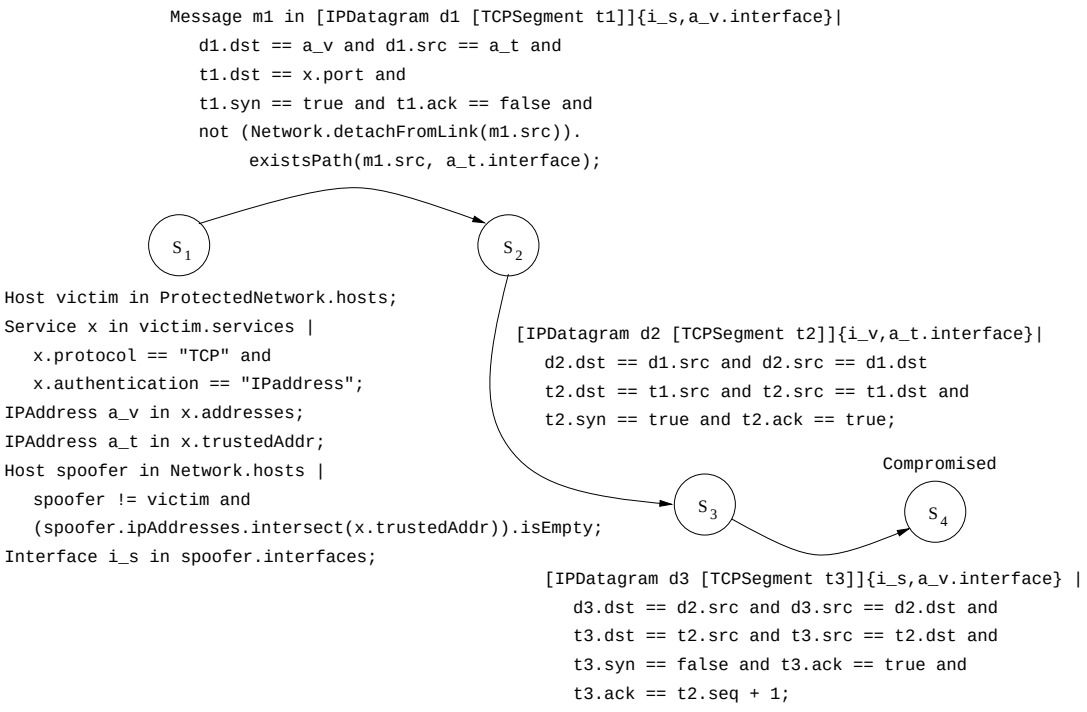


Figure 9: TCP spoofing attack.

The first signature action is a message `m1` that belongs to the sequence of messages used to send `victim` a spoofed IP datagram `d1` whose source IP address is the trusted address `a_t`. The datagram encapsulates a TCP segment `t1` with the SYN flag set and whose destination port is the port used by service `x`. Message `m1` is such that in the network obtained by removing the message's source interface from the corresponding link, there is no path between the message source interface and the interface associated with the trusted address.

The second signature action is the `victim`'s reply. The event is a TCP segment encapsulated in an IP datagram. The source/destination addresses/ports are inverted with respect to the previous signature action. The SYN and ACK flags are both set. Note that in this case it is not necessary to specify a particular message among those involved in datagram delivery because there is no reference to link-level information. This means that any of the messages used to deliver the IP datagram can be used to detect this signature action.

The final signature action is the last step in the TCP connection setup handshake. It is a TCP segment enclosed in an IP datagram that is delivered following a path from the attacker to the victim. The attack is successful if the TCP segment acknowledgment number is equal to the increment of the sequence number contained in the TCP segment of the previous signature action. The destination state is a "compromised state"

Analysis In the initial step of the analysis the network fact base is queried for all hosts that provide TCP-based, address-authenticated services and the corresponding trust relationships. In the example network of Figure 2, the results are host `jackson` exporting the *rlogin* service to `carpenter` and `lang`. Therefore, the attack scenarios feature `jackson` as the `victim`, *rlogin* as service `x`, a_8 as the attacked address, and either a_{10} or a_{11} as the spoofed address. The `spoofer` can be any host that is not `jackson`, `carpenter`, nor `lang`.

For each attack scenario the analysis engine generates the messages involved and then it matches them against the corresponding signature actions. The first signature action is a spoofed IP datagram. In this case the analysis is analogous to the one performed in Section 4.1.2. The results are that a single probe on L_2 is able to detect the first step of any scenario. The probe is configured so that it looks for messages coming from any interface that is not `bergman`'s i_{9_1} and directed to `jackson`'s interface i_8. The messages contain requests for `jackson`'s *rlogin* service to create a TCP virtual circuit and pretend to come from IP address a_{10} or a_{11}. The second signature action is the legitimate reply of `jackson` to either `carpenter` or `lang`. For any scenario the reply can be detected with a probe either on link L_2 or on link L_3. The event of the third signature action can be detected by a probe on link L_2, analogously to the first signature action. As a result, a single probe on link L_2 is able to detect any possible attack scenario in the network of Figure 2.

4.3 UDP race attacks

Attack Race attacks are another kind of spoofing-based attack. As in the attack presented in Section 4.1.3, in this scenario there is a server exporting a UDP-based service, whose protocol follows a request/reply schema, and whose authentication is based on the host address or name. During the attack, a legitimate client performs a service request to the server. The attacking host is able to detect that a request has been issued and produces a reply that appears to come from the server. If the spoofed reply reaches the client before the legitimate one, the client will take actions on the basis of the contents of the spoofed reply and will (usually) ignore the legitimate one, considering it as a transmission error. An instance of this class of attack is represented by the NIS authentication race [10]. In this attack, an intruder races against an NIS server in answering a request for the password file performed by an NIS client for authentication purposes. Another application of this attack is server-side spoofing of the NFS protocol, in which, for example, the attacker delivers trojan horse versions of the executables requested by the client.

The attack can be described as in Figure 10. The assertions of state S_1 identify a `server` providing a service `x` that uses UDP as the transport protocol, and that performs authentication on the basis of the IP address. The service can be contacted at IP address `a_s`. The client is identified by stating that there is

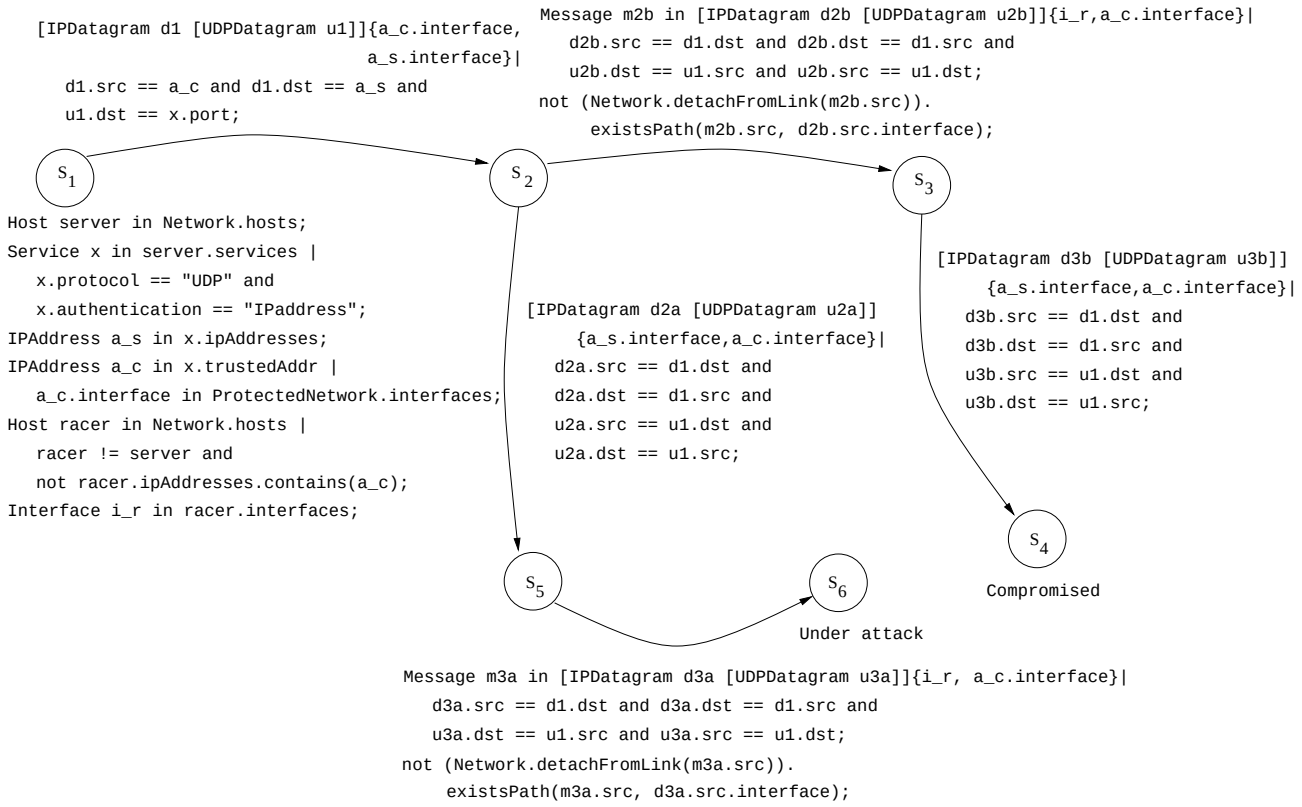


Figure 10: UDP race attack.

a trusted IP address `a_c` such that `a_c`'s interface is in the `ProtectedNetwork`. Then the attacker host, `racer`, is modeled together with its interface `i_r`.

The first signature action is the client's request. The event is a UDP datagram delivered following a path from the client's interface to the server's interface.

Starting from state S_2 two signature action sequences are possible: the attacker wins the race and it is able to deliver a spoofed reply to the client or the attacker loses the race and the attack is not successful. In the first case the attack evolution follows states S_3 and S_4 to reach a final "compromised" state. In the second case the attack proceeds through states S_5 and S_6. The final state is described as *safe but under attack*. In both cases the signature actions are the legitimate `server`'s reply and a spoofed reply coming from `racer` and pretending to be from `server`.

Analysis Consider the network of Figure 2. By matching the scenario description against the network fact base the analysis engine obtains host `fellini` playing the role of the `server` and providing an NFS service to clients `kubrick` and `wood`.

A first class of scenarios is the one that has `kubrick` as the attacked client. In this case, every attempt to impersonate `fellini` with respect to `kubrick` can be detected by a probe on link L_1 instructed to look for any message that is an NFS reply from address a_4 but is originated from any interface other than i_4. On the same link it is possible to detect the other two events involved in the scenario, i.e., the client's request and the server's legitimate response. Therefore, a probe on link L_1 is able to detect this class of scenarios.

The second class of scenarios features `wood` as the attacked client. In this case, the client's request and the server's legitimate responses can be detected either on link L_1 or on link L_2. Detection of the spoofed reply can be achieved by placing probes on links L_1, L_2, and L_4.

The detection of scenarios where the racers are `jackson`, `bergman`, `carpenter`, or `lang` can be detected by a single probe on link L_2. The scenario where `wilder` attacks `wood` can be detected by a single probe on link L_1. In contrast, the events involved in attack scenarios where `landis`, `hitchcock`, or the `Outside` composite host play the role of racers, appear on different links of the network: the client's request and the server's reply can both be detected either on link L_1 or on link L_2, while the spoofed reply can be detected on link L_4 only. The analysis engine tries to solve this problem in two ways: (i) trying to place a probe on host `chaplin` that is connected to links L_1, L_2, and L_4, or (ii) splitting the scenario in two subscenarios that are assigned to two different probes. In the latter case, the probe on link L_4 is instructed to look for UDP datagrams that pretend to come from `fellini` and are directed to `wood`. When this event is detected a message will be sent to a probe on link L_2 (or L_1), which is responsible for the detection of the other events in the scenario.

5 Conclusions and future work

State transition analysis has proved to be an effective approach to host-based intrusion detection. This paper presents a further application of the state transition analysis approach to detect network-based intrusions. The approach is based on formal models of attack scenarios (state transition diagrams) and of the network itself (network hypergraphs). These two models have been composed in order to determine the configuration and positioning of intrusion detection probes. The resulting architecture is distributed, autonomous, and highly customized towards the target network. In this way probes are more focused and the high volume event streams generated by networks can be filtered resulting in decreased overhead.

The current prototype of the NetSTAT tool allows the Network Security Officer to define a network and the state transition diagrams describing the attacks. A number of sample networks have been constructed using the prototype network fact base component. In addition, network-based state transition representations for more than 30 attacks have been defined. The attacks include different flavors of UDP/TCP spoofing attacks, UDP race attacks, CGI-based attacks, RPC-based attacks, DNS-based attacks, remote buffer overflows, and denial of service attacks.

The analyzer is in a very initial stage of development. It provides a limited, *ad hoc* set of analytic capabilities, mostly related to topological analysis of network hypergraphs. Currently, the analyzer is not able to produce actual probe configurations and, although several configurations have been generated manually using the existing algorithms, it has not yet been tested on real networks. One of the problems that is anticipated when dealing with real networks is the scalability of the proposed network analysis process. The sample attacks described in this paper have been analyzed with respect to a "toy" network composed of few hosts. The question is whether this technique will be applicable to networks composed of thousands of hosts. Although the approach has not been tested on a large network, preliminary investigation indicates that the approach can still be applied. To be more explicit, in the analysis process the most critical operation is the generation of all the possible instances of an attack scenario with respect to a given target network. This is a combinatorial process and therefore may produce a large number of cases. Nevertheless, all the attack scenario instances need not be considered at the same time. Therefore, the analyzer does not require a large amount of memory to be able to operate on large networks, even though it may require a large amount of time. The number of attack scenario instances to be considered can also be reduced by modularizing the network representation. That is, if parts of the target network can be modeled as "composite hosts", which is often the case with real networks, then the number of cases can be greatly reduced.

During the second half of 1998, the NetSTAT prototype was evaluated as part of both the MIT Lincoln Laboratory's off-line intrusion detection system evaluation and the Air Force Research Laboratory (AFRL) real time evaluation [20]. In the first case, NetSTAT was used to analyze dumps of several weeks of traffic looking for attack signatures. In the second case, the NetSTAT prototype was installed on a testbed network at AFRL. The testbed network was composed of several subnetworks connected by seven routers. The subnetworks were composed of heterogeneous hosts (PCs with Linux and Windows NT, SPARCstations with different versions of SunOS and Solaris, and IBM workstations with AIX). In both efforts the NetSTAT prototype performed very well. Participating in this event gave both new insights into the approach being used to detect intrusions and strong positive feedback on the research that has been performed so far.

Future work will focus on completion of the first prototype and on the refinement of its architecture. Currently, the language used to describe attacks in the NetSTAT tool is *ad hoc*. Research is in progress to define a new language for the STAT toolset. The language is intended to provide a well-defined semantics and to allow the security officer to describe host-based and network-based intrusions in a uniform way.

Acknowledgments

This research was supported by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-1-0207.

References

[1] J.P. Anderson. Computer Security Threat Monitoring and Surveillance. James P. Anderson Co., Fort Washington, April 1980.

[2] J. Balasubramaniyan, J.O. Garcia-Fernandez, D. Isacoff, E.H. Spafford, and D. Zamboni. An Architecture for Intrusion Detection using Autonomous Agents. Technical Report Coast TR 98-05, Department of Computer Science, Purdue University, 1998.

[3] S. Bellovin. Security Problems in the TCP/IP Protocol Suite. *Computer Communications Review*, 19(2), 1990.

[4] C. Berge. *Hypergraphs*. North-Holland, 1989.

[5] S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, J. Rowe, S. Staniford-Chen, R. Yip, and D. Zerkle. The Design of GrIDS: A Graph-Based Intrusion Detection System. Technical Report CSE-99-2, U.C. Davis Computer Science Department, January 1999.

[6] CISCO. Netranger intrusion detection system. Technical Information, April 1999.

[7] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) – Specification. RFC 2460, December 1998.

[8] D.E. Denning. An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, February 1987.

[9] L. T. Heberlein, G.V. Dias, K.N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A Network Security Monitor. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 296 – 304, Oakland, CA, May 1990.

[10] D.K. Hess, D.R. Safford, and U.W. Pooch. A Unix Network Protocol Security Study: Network Information Service. Technical report, Texas A&M University, November 1992.

[11] K. Ilgun. USTAT: A Real-time Intrusion Detection System for UNIX. Master's thesis, Computer Science Department, University of California, Santa Barbara, July 1992.

[12] K. Ilgun. USTAT: A Real-time Intrusion Detection System for UNIX. In *Proceedings of the IEEE Symposium on Research on Security and Privacy*, Oakland, CA, May 1993.

[13] K. Ilgun, R. A. Kemmerer, and P. A. Porras. State Transition Analysis: A Rule-Based Intrusion Detection System. *IEEE Transactions on Software Engineering*, 21(3), March 1995.

[14] Internet Security Systems. *Introduction to RealSecure Version 3.0*, January 1999.

[15] H. S. Javitz and A. Valdes. The SRI IDES Statistical Anomaly Detector. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1991.

[16] R.A. Kemmerer. NSTAT: A Model-based Real-time Network Intrusion Detection System. Technical Report TRCS-97-18, Department of Computer Science, UC Santa Barbara, November 1997.

[17] S. Kent and R. Atkinson. IP Authentication Header. RFC 2402, November 1998.

[18] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). RFC 2406, November 1998.

[19] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, November 1998.

[20] MIT Lincoln Lab. The 1998 DARPA Intrusion Detection Evaluation. http://ideval.ll.mit.edu/1998_index.html, 1998.

[21] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of the 1993 Winter USENIX Conference*, San Diego, CA, January 1993.

[22] R.T. Morris. A Weakness in the 4.2BSD UNIX TCP/IP Software. Technical report, AT&T Bell Laboratories, February 1985.

[23] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network Intrusion Detection. *IEEE Network*, pages 26–41, May/June 1994.

[24] P.A. Porras. STAT – A State Transition Analysis Tool for Intrusion Detection. Master's thesis, Computer Science Department, University of California, Santa Barbara, June 1992.

[25] P.A. Porras and P.G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the 1997 National Information Systems Security Conference*, October 1997.

[26] J. Postel. User Datagram Protocol. RFC 768, August 1980.

[27] J. Postel. Internet Protocol. RFC 792, 1981.

[28] J. Postel. Transmission Control Protocol. RFC 793, September 1981.

[29] S.R. Snapp, J. Brentano, G. Dias, T. Goan, T. Heberlein, C. Ho, K. Levitt, B. Mukherjee, S. Smaha, T. Grance, D. Teal, and D. Mansur. DIDS (Distributed Intrusion Detection System) – motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference*, Washington, DC, October 1991.

[30] Sun Microsystems, Inc. *Installing, Administering, and Using the Basic Security Module.* 2550 Garcia Ave., Mountain View, CA 94043, December 1991.

[31] G. Vigna. A Topological Characterization of TCP/IP Security. Technical Report TR-96.156, Politecnico di Milano, November 1996.

[32] G. Vigna and R.A. Kemmerer. NetSTAT: A Network-based Intrusion Detection Approach. In *Proceedings of the* 14^{th} *Annual Computer Security Application Conference*, Scottsdale, Arizona, December 1998.

[33] J.R. Winkler. A Unix Prototype for Intrusion and Anomaly Detection in Secure Networks. In *Proceedings of the* 13^{th} *National Computer Security Conference*, Washington, D.C., October 1990.