Network Intrusion Detection: Dead or Alive?

[Classic Paper]

Giovanni Vigna
Department of Computer Science
University of California, Santa Barbara
vigna@cs.ucsb.edu

ABSTRACT

Research on network intrusion detection has produced a number of interesting results. In this paper, I look back to the NetSTAT system, which was presented at ACSAC in 1998. In addition to describing the original system, I discuss some historical context, with reference to well-known evaluation efforts and to the evolution of network intrusion detection into a broader field that includes malware detection and the analysis of malicious behavior.

Keywords

Intrusion Detection, Network Security

1. INTRODUCTION

Network intrusion detection systems (NIDSs) have evolved from their academic beginnings into mainstream commercial products, and network intrusion detection is now considered a "mature technology." From the early network-based systems (such as EMERALD [13], NSM [3], Bro [11], and NetSTAT [16]), dozens of network-based systems have been proposed in research and many have transitioned to the commercial world to become products (see, for example, Snort [14], which is the most popular open-source network intrusion detection system today).

Even though network intrusion detection is considered a mature technology and research in this field is sometimes considered "dead," network attacks are still prevalent, large-scale abuse of network resources are an everyday reality, and sophisticated attacks seem to be able to easily bypass commercial intrusion detection systems. So what happened to network intrusion detection?

In this paper, I look back to some early research in network intrusion detection, namely the NetSTAT system, which was presented at ACSAC in 1998 [16]. I describe the system in Section 2 and present some interesting contributions which are still unmatched by the current state-of-the-art tools.

In Section 3, I discuss how, in the late nineties, there was a push to compare and evaluate the intrusion detection research being performed at the time, which culminated in the MIT Lincoln Laboratory's intrusion detection system evaluation effort. Even though the results of this effort were criticized and misused, they still represent one of the most systematic and interesting attempts to measure, compare, and even stimulate research in security.

Then, in Section 4, I describe some of the shortcomings that gave network intrusion detection a bad name, but I also discuss how the lessons learned in developing intrusion detection systems have been taken into account in shaping a larger research field, involved with the detection of compromises at many levels.

2. THE NETSTAT SYSTEM

The NetSTAT system was a network-based intrusion detection system. NetSTAT extended the state transition analysis technique (STAT) [4] to network-based intrusion detection in order to represent attack scenarios in a networked environment. However, unlike other network-based intrusion detection systems that monitored a single sub-network for patterns representing malicious activity, NetSTAT was oriented towards the detection of attacks in complex networks composed of several sub-networks. In this setting, the messages that are produced during an intrusion attempt may be recognized as malicious only in particular subparts of the network, depending on the network topology and service configuration. As a consequence, intrusions cannot be detected by a single component, and a distributed approach is needed.

The NetSTAT approach models network attacks as state transition diagrams, where states and transitions are characterized in a networked environment. The network environment itself is described by using a formal model based on hypergraphs [1, 15].

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

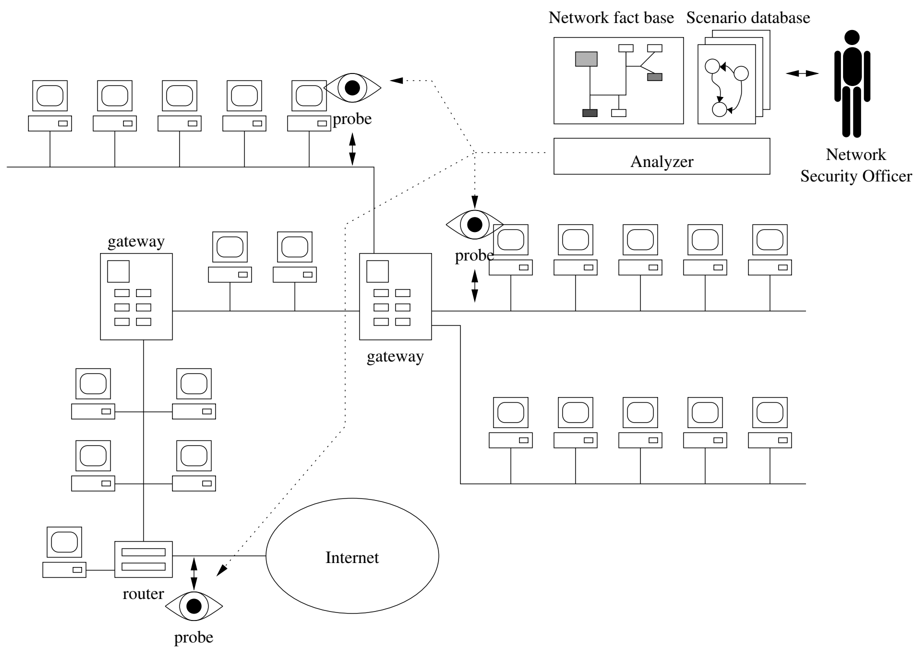


Figure 1: The NetSTAT architecture.

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 probes and focused filtering of the network event stream, delivering more reliable, efficient, and autonomous components.

2.1 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 the NetSTAT architecture is given in Figure 1.

2.1.1 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 provided.

The network topology is a description of the constituent components of the network and how they are connected. The network model underlying the NetSTAT tool uses *interfaces*, *hosts*, and *links* as primitive elements. A network is represented as a hypergraph on the set of interfaces [15]. 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. Another advantage is that this formalization allows one to model network links based on a shared medium (e.g., Ethernet) in a natural way, by representing the shared medium 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 a simple 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 has some associated information. For example, hosts have several attributes that characterize the type of hardware and operating system software installed. The reader should note that in this model “host” is a rather general concept. More specifically, 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 and printers are considered to be hosts. Links are characterized by their type (e.g., Ethernet). Interfaces are characterized by their type and by their corresponding link- and network-level addresses. This information is represented in the model by means of functions that associate the network elements with the related information.

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, that is, how services are instantiated and accessed over the network.

Figure 2 shows an example network. In the hypergraph

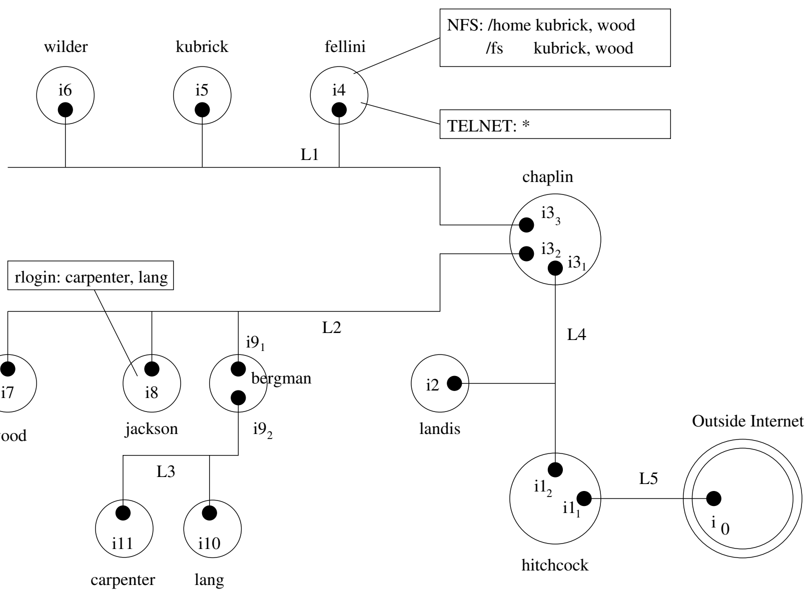


Figure 2: An example network.

describing the network, interfaces are represented as black dots, hosts are represented as circles around the corresponding interfaces, and links are represented as lines connecting the interfaces. The sample network is composed of five links, namely L_1, L_2, L_3, L_4, and L_5, and twelve hosts. Hereinafter, 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) containing all the interfaces and corresponding addresses not in use elsewhere in the modeled network. As far as services are concerned, host `fellini` is an NFS server exporting file systems `/home` and `/fs` to `kubrick` and `wood`. In addition, `fellini` is a TELNET server for everybody. Host `jackson` exports an *rlogin* service to hosts `carpenter` and `lang`.

2.1.2 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 analysis technique was originally developed to model host-based intrusions [4]. It describes 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, zero or more intermediate states, and (at least) one "compromised" ending state. States are characterized by means of *assertions*, which are functions with zero or more arguments returning Boolean values. Typically, these assertions describe 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 host-based signature actions include reading, writing, and executing files. For a complete description of the state transition analysis technique see [12]. For NetSTAT the original STAT technique has been applied to computer networks, and the concepts of state, assertions, and signature actions have been characterized in a networked environment.

States and Assertions.

In network-based state transition analysis the 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 on a network that can be verified by examining the network fact base; that is, by examining its 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 another example, the following assertion:

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

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

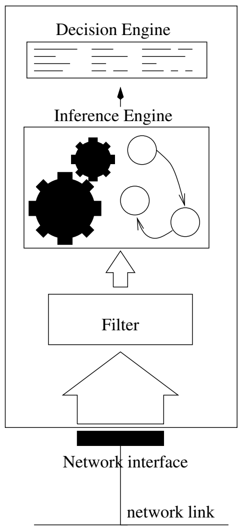


Figure 3: Probe architecture.

denotes an interface of a host, say `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 examples is `NFS-Mounted(filesys, server, client)`, which returns true if the specified file system exported by `server` is currently mounted by `client`. Another example is `ConnectionEstablished(addr1, port1, addr2, port2)`, which returns true if there is an established virtual circuit between the specified addresses and ports. These assertions are used to determine what relevant network state events should be monitored by a network probe.

Transitions and Signature Actions.

In NetSTAT, signature actions are expressed by leveraging 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.

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 an IP 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 is not the final destination interface (i.e., the message is not the last one in the delivery process).

Events representing single UDP datagrams or TCP 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_y 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_y` and the destination port is 23.

TCP virtual circuits are higher-level, composite events. A virtual circuit is identified by the tuple *(source IP address, destination IP address, source TCP port, 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 the rules of the TCP protocol (e.g., sequencing, retransmission, etc.). The streams are denoted by `streamToClient` and `streamToServer`.

For example, the signature action:

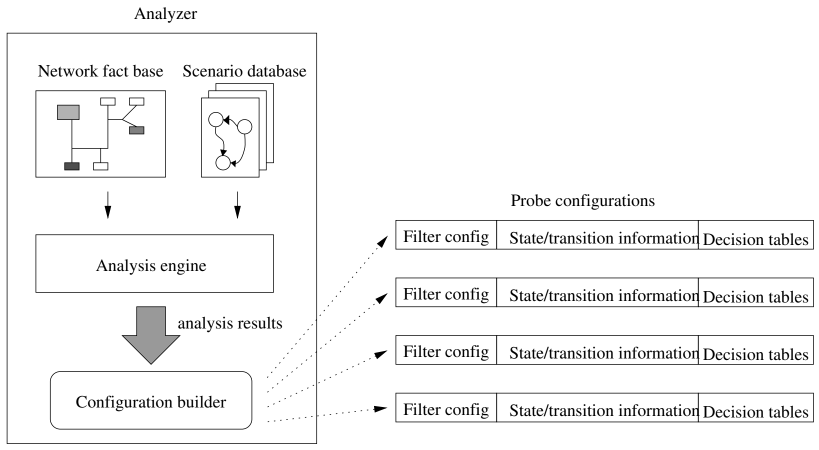


Figure 4: Analyzer architecture.

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

denotes a segment that has the SYN bit set and belongs to a virtual circuit established between interfaces `i_x` and `i_y` and that has destination IP address `a_y` and destination port 80.

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 to the server side[2].

2.1.3 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 among the huge number of messages transmitted over a network link. The filter module can be 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 [8] and a modified version of the *tcpdump* application [9].

The *inference engine* is the actual intrusion detecting system. 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 yet completed. Each entry contains information about the history of the instantiation, such as the address and services involved, the time of the attack, 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 modified datagrams that reset

[2]This original formulation of the NetSTAT state transition language was subsequently refined into a general-purpose state transition language, called STATL [2].

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 sub-scenarios 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) sub-networks.

2.1.4 Analyzer

The analyzer is used 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;
- 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 generator 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 information 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 sub-network 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 every client/server pair that satisfies the specified trust relationship[3]. 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 given the current network configuration.

[3]Thus, state assertions are treated as if they were universally quantified.

Once the attack scenarios contained in the state transition scenario database have been customized over the given network, another module, called the *configuration builder*, translates the results of the analysis engine to produce the 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 to customize the probe's decision engine.

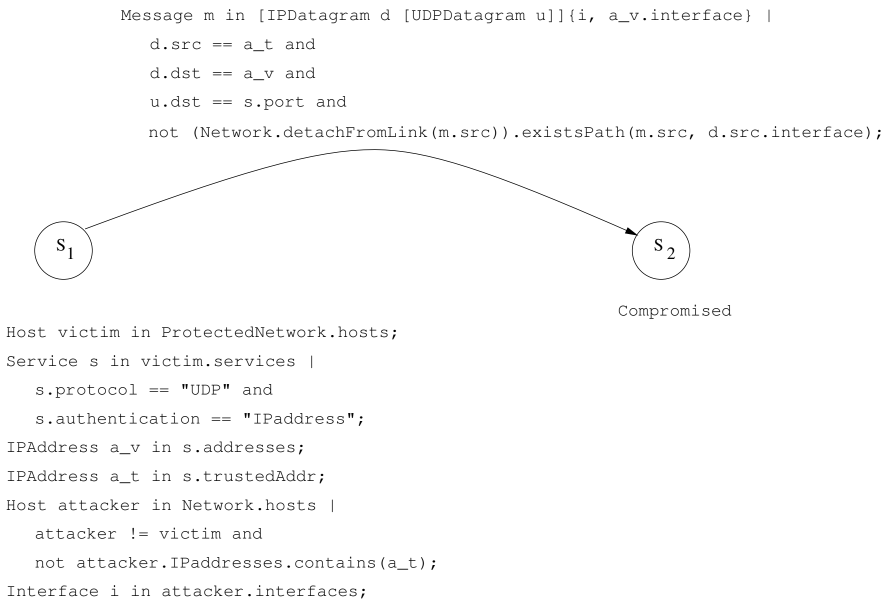
2.2 Example

Consider, as an example, 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 receiver of a spoofed datagram is usually not able to detect the attack. For this example, consider the network presented in Figure 2 and assume that host `lang` is attacking host `fellini` by providing an NFS request that pretends to come from `wood`, who is a trusted, authorized client. Host `fellini` receives the request encapsulated in a link-level message from `chaplin`'s interface i_{3_3} to `fellini`'s interface i_4. Host `fellini` has no means to distinguish this message from the final link-level message used to deliver a legitimate request coming from `wood`. Therefore, `fellini` cannot determine if the datagram is a spoofed one. The spoofing can be detected, however, by examining the message on link L_2. In this case, since the link-level message comes from `bergman`'s interface i_{9_1} while it should come from `wood`'s interface i_7, the datagram can be recognized as spoofed. In general, if one considers a single link-level message that encapsulates a UDP-over-IP datagram, the datagram may be considered spoofed if there is no path between the interface corresponding to the datagram source address and the link-level message source interface in the network obtained by removing the link-level message source interface from the corresponding link.

This attack scenario is described in Figure 5 using a state transition diagram. The scenario assumes that two networks have been defined, `Network` and `ProtectedNetwork`. `Network` is a reference to the network modeled in the fact base; `ProtectedNetwork` is a sub-network that contains the hosts that must be protected against the attack.

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 `s` 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 following assertions characterize the attacker. In particular, the fifth assertion states that there exists a host `attacker` that is different from `victim` and that doesn't have the trusted IP address. The sixth assertion states that `i` is one of the `attacker`'s interfaces.

The signature action is a spoofed service request. That is, a UDP datagram that pretends to come from one of the trusted addresses, although it did not originate from the corresponding interface. Actually the signature action is a link-level message `m` that belongs to the sequence of mes-



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

```
Host victim in ProtectedNetwork.hosts;
Service s in victim.services |
   s.protocol == "UDP" and
   s.authentication == "IPaddress";
IPAddress a_v in s.addresses;
IPAddress a_t in s.trustedAddr;
Host attacker in Network.hosts |
   attacker != victim and
   not attacker.IPaddresses.contains(a_t);
Interface i in attacker.interfaces;
```

Figure 5: UDP spoofing attack scenario.

sages used to deliver an IP datagram from interface `i` 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 `s`. 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. For example, consider a link-level message exchanged between `bergman`'s interface i_{9_1} and `chaplin`'s interface i_{3_2}. The message is an intermediate step in the delivery of a UDP-over-IP datagram to `fellini`; the IP source address of the datagram is `wood`'s a_7. Intuitively, it is clear that a message originated by `wood` and intended for `fellini` cannot come from one of `bergman`'s interfaces, because there is no path in the network that would require `bergman` to act as a forwarder of the datagram. One way to check for this is by removing the source interface of the message (i_{9_1}) and checking whether or not there still exists a path from the host whose IP address is the source of the datagram (`wood`) to the host that contains the interface that was removed (`bergman`). The second state (S_2) is a "compromised" state.

The analysis of the attack starts by identifying the possible scenarios in the context of a modeled network. Thus, the analysis engine determines all the possible combinations of victim host, attacked service, spoofed address, and attacker in a particular network. A subset of the scenarios for the network in Figure 2 is presented in Table 1. In all scenarios `fellini` is the attacked host, NFS is the service exploited, and the spoofed address can be `kubrick`'s or `wood`'s.

The next step in the analysis is to determine where the events associated with the signature action can be detected. For each of these scenarios, the analysis engine generates all the possible datagrams between the interface of the attacker and the interface of the victim. In practice, the engine finds all the paths between the interfaces defined by the scenario and, for each path, generates the sequence of messages that would be used to deliver a datagram. For each message the predicate contained in the clause of the signature action is applied. 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_{11} and delivered through three messages to `fellini`'s interface i_4. 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 either needs a probe on L_3 looking for link-level messages from `carpenter`'s interface i_{11} 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. 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.

3. EVALUATING INTRUSION DETECTION SYSTEMS

Network intrusion detection systems should not be difficult to evaluate: given a traffic dump collected during real or simulated intrusions, a NIDS should be able to detect a subset of the attacks while producing a certain (hopefully low) number of false positives. This is not as straightforward with other types of intrusion detection systems (e.g., host-based systems and application-based systems), because the quantity and quality of information collected about the actions performed by the OS and its applications can vary dramatically. In addition, systems that use an anomaly-

victim	s	a_v	a_t	attacker	i
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}
fellini	NFS	a_4	a_7	hitchcock	i_{1_1}
...	...	...	...	...	...
fellini	NFS	a_4	a_5	lang	i_{10}
fellini	NFS	a_4	a_7	lang	i_{10}
fellini	NFS	a_4	a_5	carpenter	i_{11}
fellini	NFS	a_4	a_7	carpenter	i_{11}

Table 1: Possible scenarios for the UDP spoofing attack.

based approach to intrusion detection necessitate training data, which should be realistic, complete, and attack-free (note that "realistic and attack-free" could be considered an oxymoron). This type of data is particularly hard to collect and/or generate.

Even though the creation of a dataset that can be leveraged to compare the performance of intrusion detection systems is a challenging task, in 1998 and 1999 a group of researchers from the MIT Lincoln Laboratory courageously embarked in an effort to produce such a dataset, which included both training data and test data (with truth files) in the form of network packets, OS audit records, and file system dumps [6, 7].

These datasets were used to evaluate a number of intrusion detection systems being developed by academic researchers. At the beginning of the evaluation process, the attack-free training data was given to the participants, and, after a while, the test data containing attacks was distributed (without truth files). The participants then had to identify the attacks and submit their detection alerts, which were then evaluated with respect to the truth files.

The results of the evaluation were disclosed only partially, without declaring a "winner," and with great care in not making any single group look bad. Therefore, instead of a single score, the authors of the evaluation provided a set of scores that took into consideration various characteristics of the systems involved, creating a no-winner/no-loser situation. We think that this was a missed opportunity to foster research by creating a competition with a clear winner, as was later demonstrated by other challenges (e.g., the DARPA Grand Challenge for unmanned vehicles), which by having a clear winner motivated the competitors, fostered innovation and creativity, and provided great publicity for both the participants and the funding agency.

To determine a winner, a more draconian approach to evaluation would have been to simply compose the recall and precision of the intrusion detection systems. More precisely, in order to evaluate the effectiveness of a system one could compute the percentage of hits H over the total number of attacks T, that is, $(H/T) * 100$. This is a measure of how many attacks were actually detected with respect to the overall set of attacks (i.e., the recall). Then, in order to characterize how precise the system is, one would compute the percentage of false alarms F over the total number of detections $H + F$, that is $(H/(H + F)) * 100$. For example, a system with three detections and no false alarms would have a precision of 100%, but it would not be very effective at detecting attacks if the dataset contained hundreds of attack instances. As another example, a system that flagged every single packet as malicious would have an effectiveness of 100% because all attacks would be detected, but it would also have an abysmal precision. Therefore, the obvious choice is to multiply the two measures above to take into account these two important aspects of intrusion detection.

The values of these metrics are shown in Table 2 for the systems that participated in the 1999 MIT Lincoln Laboratory evaluation. According to the proposed metrics, UCSB's NetSTAT would be the winner of the 1999 competition, closely followed by SRI's EMERALD.

Even though the evaluation failed to declare a clear winner and, in addition, there were some criticisms against the evaluation process [10], the dataset produced was immensely popular, and it is without doubt the most used dataset in the intrusion detection community.

Unfortunately, the MIT/LL dataset and the corresponding truth files were used in a series of scientific publications in which the performance of intrusion detection systems, evaluated on the non-blind dataset, were compared to the performance of the intrusion detection systems that participated in the blind evaluation, with nefarious and unfair results. Since then, the dataset has become outdated, and nowadays it is used very seldom in research publications.

4. THE DEATH (AND REBIRTH) OF INTRUSION DETECTION

In the years following the MIT/LL evaluation, there was an increased skepticism towards network intrusion detection and its ability to detect attacks, especially 0-day exploits and mutations of existing attacks [17]. In addition, researchers started to develop attacks against stateful intrusion detection system, exposing the challenge of detecting low-traffic, slow-paced attacks that last months (if not years).

In general, there was a shift from the analysis of network data to the analysis of host data, under the assumption that only by monitoring the end nodes one could possibly detect sophisticated attacks. Therefore, during the early 2000s, academia started losing interest in network intrusion detection, while, at the same time, the use of commercial network intrusion detection systems became an established best-practice in enterprise networks. This happened sometimes in disguise, for example by relabeling NIDS as "intrusion prevention systems" to describe network intrusion detection systems with traffic-blocking responses.

Around 2003-2004 it looked like research on the "classic"

	GMU	NYU	RST Elman Network	RST State Tester	RST String Transd.	SRI Derbi	SRI Estat	SRI EMERALD	SunySB	UCSB STAT
Hits	43	21	37	26	26	17	29	94	7	88
False Positives	16	74	5351	429	117	48	96	13	2	4
H/T	21.3	10.4	18.3	12.9	12.9	8.4	14.4	46.5	3.5	43.6
$H/H+F$	72.9	22.1	0.7	5.7	18.2	26.2	23.2	87.9	77.8	95.7
$H/T * H/H+F$	15.5	2.3	0.1	0.7	2.3	2.2	3.3	40.9	2.7	41.7

Table 2: Hits, false positives (in absolute values), recall, precision, and composition of recall and precision (in percentages) for the systems involved in the MIT Lincoln Laboratory 1999 IDS evaluation, which contained 202 attacks.

network intrusion detection problem (i.e., detecting attacks by looking at network packets) was dwindling fast. However, at the same time, the techniques used to characterize network attacks were applied to the detection of malicious code components, such as worms and bots. Both misuse-based and anomaly-based techniques were readily leveraged to identify malware of various kinds. In a way, these research efforts resulted in "intrusion detection" system that were closer to the meaning of the term than the early NIDSs. In fact, while the early systems focused mostly on detecting attacks, these new systems focused on detecting the actual intrusions by identifying malicious behavior that would indicate that a system had been compromised.

This "born-again" network intrusion detection research is characterized by the heavy use of data-mining and machine-learning techniques to address one of the main problems associated with misuse-based NIDS, which is the need for the manual specification of attack models (note that some of the seminal work in this field was performed in the late 90's [5]).

5. CONCLUSIONS

Even though the term "Intrusion Detection" sometimes is looked-down upon by the academic community, intrusion detection research will always be a core part of the security field. It might be the case that the focus of intrusion detection will move towards more semantically-rich domains, such as the OS and the web. For example, web-based intrusion detection systems (normally referred to as "Web Application Firewalls", for marketing purposes) leverage knowledge about the characteristics of web applications and their logic, in order to identify attacks. Nonetheless, these systems mostly use concepts that were researched and applied more than two decades ago.

This "re-invention" of network intrusion detection techniques and approaches shows how intrusion detection (be it network-based, web-based, or host-based) is still an important research problem. As new attacks and new ways of compromising systems are introduced, both researchers and practitioners will develop (or re-discover) techniques for the analysis of events that allow for the identification of the manifestation of malicious activity.

The next challenge will be to expand the scope of intrusion detection to take into account the surrounding context, in terms of abstract and difficult-to-define concepts, such as missions, tasks, and stakeholders, when analyzing data in an effort to identify malicious intent.

6. REFERENCES

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

[2] S. Eckmann, G. Vigna, and R. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, 10(1,2):71–104, 2002.

[3] L. Heberlein, G. Dias, K. 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.

[4] K. Ilgun, R. Kemmerer, and P. Porras. State Transition Analysis: A Rule-Based Intrusion Detection System. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.

[5] W. Lee and S. Stolfo. Data Mining Approaches for Intrusion Detection. In *Proceedings of the USENIX Security Symposium*, San Antonio, TX, January 1998.

[6] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. Evaluating Intrustion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition, Volume 2*, Hilton Head, SC, January 2000.

[7] R. Lippmann and J. Haines. Analysis and Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation. In *Proceedings of the Symposium on the Recent Advances in Intrusion Detection (RAID)*, pages 162–182, Toulouse, France, 2000.

[8] 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.

[9] S. McCanne, C. Leres, and V. Jacobson. Tcpdump 3.7. Documentation, 2002.

[10] J. McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM Transaction on Information and System Security*, 3(4), November 2000.

[11] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, January 1998.

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

Barbara, June 1992.

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

[14] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the USENIX LISA '99 Conference*, Seattle, WA, November 1999.

[15] G. Vigna. A Topological Characterization of TCP/IP Security. In *Proceedings of the* 12*th* *International Symposium of Formal Methods Europe (FME '03)*, number 2805 in LNCS, pages 914–940, Pisa, Italy, September 2003. Springer-Verlag.

[16] G. Vigna and R. Kemmerer. NetSTAT: A Network-based Intrusion Detection Approach. In *Proceedings of the* 14*th* *Annual Computer Security Applications Conference (ACSAC '98)*, pages 25–34, Scottsdale, AZ, December 1998. IEEE Press.

[17] G. Vigna, W. Robertson, and D. Balzarotti. Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In *Proceedings of the ACM Conference on Computer and Communication Security (ACM CCS)*, pages 21–30, Washington, DC, October 2004.