An Experience Developing an IDS Stimulator for the Black-Box Testing of Network Intrusion Detection Systems

Darren Mutz Giovanni Vigna Richard Kemmerer

Reliable Software Group
Department of Computer Science
University of California, Santa Barbara
{dhm,vigna,kemm}@cs.ucsb.edu

Abstract

Signature-based intrusion detection systems use a set of attack descriptions to analyze event streams, looking for evidence of malicious behavior. If the signatures are expressed in a well-defined language, it is possible to analyze the attack signatures and automatically generate events or series of events that conform to the attack descriptions. This approach has been used in tools whose goal is to force intrusion detection systems to generate a large number of detection alerts. The resulting "alert storm" is used to desensitize intrusion detection system administrators and hide attacks in the event stream. We apply a similar technique to perform testing of intrusion detection systems. Signatures from one intrusion detection system are used as input to an event stream generator that produces randomized synthetic events that match the input signatures. The resulting event stream is then fed to a number of different intrusion detection systems and the results are analyzed. This paper presents the general testing approach and describes the first prototype of a tool, called Mucus, that automatically generates network traffic using the signatures of the Snort network-based intrusion detection system. The paper describes preliminary cross-testing experiments with both an open-source and a commercial tool and reports the results. An evasion attack that was discovered as a result of analyzing the test results is also presented.

Keywords: Evasion Attacks, Intrusion Detection, Software Testing, Traffic Generation

1 Introduction

The ultimate goal of intrusion detection is to detect and classify each instance of misuse of a system while ignoring all instances of legitimate use. Intrusion detection is performed by analyzing one or more input event streams and looking for the manifestation of an attack. Examples of event streams are the packets sent on a network link, the audit records generated by a kernel-level auditing facility, or the logs produced by user-level applications.

Intrusion detection analysis techniques can be broadly classified into two classes: anomaly detection and misuse detection. Anomaly detection techniques rely on models of the "normal" behavior of a computer system. These models may focus on the users, the applications, or the network. Behavior profiles may be built by performing statistical analysis on historical data [10, 12] or by using rule-based approaches to specify behavior patterns [14, 25, 26]. Intrusion detection systems (IDSs) based on anomaly detection techniques compare actual usage patterns against the established profiles to identify abnormal activity. Misuse detection systems take a complementary approach. Misuse detection tools are equipped with a number of attack descriptions. These descriptions (or "signatures") are matched against the stream of audit data, looking for evidence that the modeled attack is occurring [11, 15, 22].

Anomaly and misuse detection approaches have advantages and disadvantages. Anomaly detection systems have the advantage of being able to detect previously unknown attacks. This advantage is balanced by a large number of false positives and the difficulty of training a system for a very dynamic environment. Misuse detection systems can perform focused analysis of the audit data and usually produce few false positives. However, misuse detection systems can detect only the attacks that have been modeled. In addition, signature-based IDSs are more vulnerable to attacks aimed at triggering a high volume of detection alerts by injecting traffic that has been specifically crafted to match the signatures used in the analysis process. This type of attack can be used to exhaust the resources on the IDS computing platform, to desensitize security officers, and to hide attacks within the large number of alerts produced.

The work presented in this paper proposes to use this attack technique as a means of generating test-cases for the black-box testing of signature-based intrusion detection systems. More precisely, given a set of intrusion detection systems to be evaluated, the approach uses the signatures employed by one of the systems to generate a synthetic

event stream that is then fed to the other intrusion detection systems in the set. The results are analyzed and compared to identify properties such as statefulness and resilience to over-stimulation attacks. The synthetic event stream is designed to mimic real attack traffic.

In addition to introducing the testing approach outlined above, this paper describes a tool, called Mucus, that supports cross-system testing using the proposed approach. The first prototype of the tool is a proof-of-concept testing application that focuses on network-based intrusion detection systems and uses Snort [22] signatures as input. This paper reports on preliminary experiments that apply this testing technique to both open-source and commercial tools. These experiments produced tangible insights into the workings of the two systems evaluated, leading to the development of an evasion attack in which a serious attack is disguised as a low-impact attack in Snort.

The remainder of the paper is structured as follows. Section 2 discusses testing of intrusion detection systems and introduces the cross-system testing technique. Section 3 describes the design of the Mucus tool and a prototype implementation. Sections 4 and 5 detail the experimental evaluation of the testing approach and introduce the evasion attack. Section 6 draws conclusions and outlines future work.

2 Testing Intrusion Detection Systems

Intrusion detection evaluation efforts have sought to quantify the relative performance of heterogeneous intrusion detection systems by establishing large testbed networks equipped with different types of IDSs where a variety of actual attacks are launched against hosts in the testbed [16, 5, 4].

These large-scale experiments have been a significant benefit to the network intrusion detection community. Practitioners have gained quantitative insights concerning the capabilities and limitations of their systems (e.g., in terms of the rate of false positive and false negative errors) in a testing environment intended to be an unbiased reproduction of a modern computer network. While generally competitive in flavor, these evaluations have precipitated valuable intellectual exchanges between IDS practitioners [17].

Testing and comparing intrusion detection systems is difficult because different IDSs have different operational environments and may employ a variety of techniques for producing alerts corresponding to attacks [21]. For example, comparing a network-based IDS with a host-based IDS may be very difficult because the event streams they operate on are different and the classes of attacks they detect have only a small intersection. For this reason IDS testing and comparison is usually applied to homogeneous categories of IDSs (e.g., network-based IDSs) [1, 20].

An interesting class of tools, called *IDS stimulators*, generates synthetic attack traffic specifically tailored to fire alerts in network intrusion detection systems. In some cases, these tools use IDS signatures directly, automatically generating synthetic attack traffic that will fire those signatures in the target system. Examples of these tools are Snot [24], Stick [9], PCP [19] and IDSwakeup [2]. Both Snot and Stick directly translate Snort [22] signatures to synthetic attack traffic. PCP provides a command line interface to packet creation routines, enabling the triggering of Snort alerts with some manual effort on behalf of the user. IDSWakeup is packaged with an extensible, but static, collection of routines for generating synthetic attack traffic. However, like PCP, it has no direct signature translation capability. Some IDS stimulators have been developed specifically to perform denial-of-service attacks against existing IDSs. Although these tools have not been explicitly designed to perform testing, they represent a useful building-block for a new testing method based on the automatic generation of test cases from intrusion detection signatures.

The success of this method as a tool for quantitative testing of intrusion detection systems depends heavily on the quality of the signatures present in the input. As Section 5.2 demonstrates, loosely specified signatures impair IDSs as well as IDS stimulation tools. Obtaining high quality attack signatures is a requirement for achieving the long-term goal of using IDS stimulators to cross-test homogeneous signature-based IDSs.

The basic idea of the cross-testing approach is that the scenarios from each IDS are used to generate synthetic attack traffic that will cause that particular IDS to alert on each of the attacks. The synthetic attacks for one system are then used to test each of the other systems.

To be more precise, consider a set of intrusion detection systems $I = \{i_1, i_2, \ldots, i_n\}$, all operating on the same event stream E. Each IDS i_k is characterized by a language l_k and a set of signatures $S^k = \{s_1^k, s_2^k, \ldots, s_m^k\}$, expressed in l_k. The testing methodology is performed according to the following steps.

1. For each scenario s_j^k in S^k a synthetic attack that will cause s_j^k to alert is generated by analyzing the signature s_j^k. Let E_k represent the set of synthetic attacks for IDS i_k.
2. Each $i_m \in I$ is then run using E_k as input, and the resulting alerts A_m^k are collected.
3. Each of the $A_1^1, A_2^1, \ldots, A_{n-1}^n, A_n^n$ are then evaluated.

This approach requires the development of n front-ends, one for each language l_k used to represent the signatures of each IDS i_k. A single event generator component that is able to generate instances of the event stream E is also needed. The only requirements on the intrusion detection systems is that they have a well-defined signature language and that it is possible to obtain their signature databases. For the preliminary experiments presented in this paper one open-source and one commercial intrusion detection system were used.

One advantage of this approach is that a large number of test cases can be generated with a limited development effort. In addition, adding new intrusion detection systems to the evaluation requires only that one build a front-end for the IDS-specific language. It is clear that the IDS community could benefit from this methodology, especially if sharing of signatures among practitioners were to become more common.

Furthermore, the proposed tool is useful for qualitative black-box evaluation of network IDSs even in the case where the signature list or the language used by that IDS is not available. In particular, commercial IDSs often ship with few details about their implementation, either from an algorithmic perspective, or in terms of precise descriptions of attack signatures. Thus, it is useful to have some mechanism for evaluating such systems, even if one can only draw qualitative conclusions about their design.

3 The Mucus Tool

The methodology described above has been incorporated in a tool called *Mucus*, which performs the synthetic generation of event streams for a target set of intrusion detection systems. The tool has been designed to be easily extensible with different event stream generators and IDS language front-ends.

A first prototype of the tool has been developed. The tool includes a front-end for the Snort signature-language and a back-end that generates network traffic. This version of the tool operates in a way that is similar to Snot and Stick. In contrast to these tools, Mucus was not developed in an *ad hoc* fashion to analyze Snort signatures. The extensible architecture of Mucus allows one to include other IDSs and other event streams, for example Windows NT/2000 events.

During the development of Mucus concerted efforts were made to obtain language definitions and signatures for four network-based IDSs, namely Snort, Enterasys Dragon, NFR NID, and Symantec Net Prowler. While the Snort language and signatures were readily available, it was not possible to obtain either the languages or signatures for the other systems[1]. Therefore, a first proof-of-concept tool was developed, called Mucus-1. Mucus-1 analyzes Snort signatures and generates traffic conforming to the signatures, for the purpose of evaluating network IDSs in a laboratory environment.

Even though the resulting tool is somewhat limited with respect to the possible scope of the general methodology, the results obtained are interesting and represent a valid application of the approach.

Prior to development of the Mucus-1 prototype, the possibility of using Snot to generate synthetic attack traffic from Snort signatures was explored. The problem is that Snot was designed to recognize the Snort signature language used in Snort version 1.8.3, but the intent of this work was to focus on current intrusion detection systems (and current signature sets). Therefore, the goal was to use Snort 1.8.6 in the evaluation[2].

A simple experiment was conducted to determine whether Snot could accurately translate signatures from the newer Snort ruleset. Snot was initialized with the ruleset packaged with Snort 1.8.6, another machine was installed with Snort 1.8.6, and they were both connected to the network. The alerts generated by Snort were collected and correlated with each datagram generated by Snot. The results (presented in Table 1) show that Snot is capable of triggering alerts in Snort, but it is able to translate fewer than half of the rules present. This fact, motivated the decision to design and implement a more capable tool.

Protocol	Total Signatures Present	Correct Alerts Produced
ICMP	128	89 (69.5%)
TCP	948	363 (38.3%)
UDP	136	76 (55.9%)
Total	1212	528 (43.6%)

Table 1. Alerts generated by Snort 1.8.6 from Snot traffic generated with Snort 1.8.6 rules.

3.1 The Mucus-1 Prototype

Mucus-1 was developed as a tool for automatically translating Snort rules into synthetic attack traffic. This section examines the problems encountered with Snort signature translation and how they are addressed in Mucus-1.

Snort's most basic function is to examine network packets on an individual basis and to search for matches against the ruleset it is configured with. Snort rules specify a conjunction of constraints on network packets. For example, the signature:

```
alert udp any 60000 -> 128.111.1.1 2140
  (msg:"alert"; content:"/bin/sh";
   depth:20)
```

will be triggered if a packet contains a valid UDP header ("`udp`"), the source port is `60000`, the destination address is `128.111.1.1`, the destination port is `2140`, and the packet payload contains the string "`/bin/sh`" somewhere in the first 20 bytes ("`depth:20`").

In Snort 1.8.6, if any given packet satisfies all constraints of a rule, Snort issues an alert for that rule and abandons fur-

[1] Persons of influence at Enterasys, NFR, and Symantec were contacted. Despite multiple electronic and face-to-face attempts to persuade them to release copies of their signature sets under non-disclosure agreement, no vendor would consent.

[2] When this research was conducted, Snort 1.8.6 was the most current version. At the time of writing, Snort 2.0.0 is available.

ther search[3]. Thus, translating a Snort rule into corresponding synthetic attack traffic involves parsing the rule and constructing a datagram that satisfies each specified constraint, as illustrated in Figure 1. In this way, the problem is decomposed into signature parsing and traffic generation. Further details on each of these stages are given below.

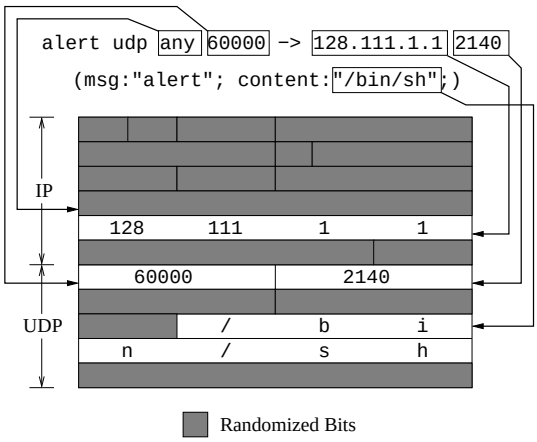

Figure 1. Translating Snort rules into network datagrams.

3.2 The Mucus-1 Parser

The goal in developing the Mucus-1 parser was to promote future extensibility while maintaining modularity with respect to the traffic generation component. Extensibility is important since the tool needs to be adaptable to other signature languages, as well as to evolution in the Snort language. A clean interface between parsing and traffic generation functions ensures these changes can be made in a straightforward manner.

One possible approach to achieving the above objectives was to modify the existing Snot tool. Unfortunately, Snot's rule parsing is implemented with low-level string operations, which made it difficult and error-prone to maintain. Therefore, instead of updating Snot, we chose to adapt the Snort-to-STATL translator, that had been previously developed by UCSB's Reliable Software Group [6, 7]. The parser for the translator was implemented as a grammar in the ANTLR translator generator [18]. Significant portions of this grammar were reused in Mucus-1. In the adaptation, STATL code generation actions were replaced with operations that constructed specifications of the rule semantics for later use. This operation takes advantage of the fact that almost all of the Snort language elements are semantically orthogonal. That is, with few exceptions, every term in a given snort rule specifies a single value, or a range of values, on a field in a protocol header or in the packet payload.

During the development of the parser, Snort language features that occurred more frequently in the standard Snort ruleset were implemented first. Language features that occurred less frequently in the Snort ruleset, `fragbits` for example, were implemented later in the development of the parser, or were not implemented at all. The implementation complexity of each feature also influenced the decision as to what to implement first. As a result, most (>95%) of the rules from the Snort 1.8.6 distribution are handled by the Mucus-1 parser. The rules that are not handled correctly occur because either the Mucus-1 parser doesn't conform completely to the Snort syntax, or because some rules in the Snort ruleset don't conform to the published Snort syntax.

Intuition suggests that analogous development decisions will be encountered when constructing front-ends for signature languages used by other intrusion detection systems. Implementing simple language features first, particularly commonly used features, enables the parser component to be utilized early in the development cycle.

3.3 The Mucus-1 Traffic Generator

The task of the Mucus-1 traffic generator is to produce single-packet instances of the rule specifications constructed by the parser. This is accomplished in the following way. First, space is allocated for the packet contents. Then, the constraints encoded in the rule abstraction are applied in turn to the bitfields in the IP and transport protocol headers. In cases where a range of values is permitted by the rule, a random value is selected from the range. In the case where the rule omits any constraints on a field, a value is selected at random from the range of legal values for that field. Next, a payload content string, if specified, is copied into the packet payload, which has a random length. If content string location constraints or constraints on the payload length are specified by the rule, then they are taken into account at this time. Any "free space" surrounding the content string is optionally padded with random bits. Finally, checksums and length fields are computed and the packet is transmitted using the `libnet` library [23].

Several features of the Snort language remain unsupported by Mucus-1. These include IP fragmentation bits, multiple content strings, regular expressions in content strings, mixed binary and ASCII content strings, IP protocol rules, and certain unusual combinations of TCP flags. In addition, TCP traffic is emitted without session establishment (three-way handshake). A few pieces of the traffic generation functionality in Mucus-1 were adapted directly from the Snot code base, with permission from the author.

3.4 A Case Study in Software Engineering

Mucus-1 exhibits several advantageous qualities from a software engineering perspective. These traits are examined

[3] Snort 2.0.0 improves upon this behavior somewhat by computing the set of all rules with matching source and destination ports, and issuing an alert for the rule which specifies the longest content string.

here.

The choice of implementing the Mucus-1 parser as an ANTLR grammar increases the maintainability of the system. The ability to extend parsers written using ad-hoc methods is questionable, and the maintenance of these systems is generally tedious and error-prone. ANTLR, like other translation tools, allows developers to specify grammars at a high level of abstraction. This allows features to be added without requiring maintainers to keep mental track of lower-level details.

Mucus-1 maintains modularity between the parsing and traffic generation components. One benefit of this approach is that it allows the tool to be easily extended. For example, if one were to write a parser for a similar signature language, it would be straightforward to integrate the new parser with the existing traffic generator. Similarly, one could introduce a new traffic generator – one that established TCP sessions before sending synthetic TCP traffic, for example – with minimal impact to the front end.

The Mucus-1 tool is also a strong example of software reuse. Integrating portions of both the Snort-to-STATL translator and of Snot saved many hours of development time and allowed the effort to focus on testing and adding new functionality.

4 Experimental Methodology

Mucus-1 was designed to automatically generate a randomized approximation of attack traffic from a set of Snort signatures. The traffic generated by Mucus-1 is fundamentally synthetic; that is, none of the generated packets actually exploit remote vulnerabilities. It is clear, therefore, that Mucus-1 should not be used to measure the rate of false positives or false negatives for a network IDS. However, it is useful to have some measure of how closely generated traffic matches the original Snort signatures and, ideally, some picture of how other network IDSs respond to this traffic.

To achieve these goals, three experiments were conducted. First, a "baseline" experiment was performed where, for each Snort signature s_k^{snort}, the ability of Mucus-1 to cause Snort 1.8.6 to generate an alert corresponding to s_k^{snort} was measured. Second, the response of a commercial IDS, Symantec Net Prowler 3.5, to the traffic generated by Mucus-1 was tested. Finally, the response of Snort and Net Prowler to a small set of real remote attacks was measured[4].

These experiments are intended to validate the behavior of Mucus-1, to gain insight into the design of Net Prowler, and to indicate qualitatively each IDS's susceptibility to IDS stimulation attack tools (such as Snot). The degree to which an IDS generates false positives when stimulated in this fashion is an important contributor to overall system performance, as others have previously noted [3].

[4] Although Net Prowler is deployed in a Windows environment, its signature set attempts to identify misuse of services on UNIX systems as well as Windows-based services.

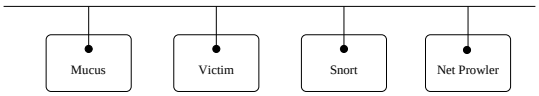

Figure 2. Topology of the Mucus testbed.

A small testbed was constructed to carry out the experiments (see Figure 2). The test network consisted of four PCs: one running Mucus-1 under Linux, one running Windows 2000 acting as a passive "victim" machine, one equipped with Snort 1.8.6 under Linux, and one with Net Prowler 3.5 under Windows NT. The 100Mb subnet shared by these four machines was private and isolated from all other networks. By default, Net Prowler automatically (via port scanning) disables signatures for hosts on its network that are not running services relevant to the signatures, but for these experiments it was manually configured to enable all signatures for the victim machine. Snort was used in the default configuration, except where noted otherwise.

The signatures used by Mucus-1 were taken without modification from the Snort 1.8.6 distribution. From each of these signatures, Mucus-1 generates a random instance of network traffic satisfying the constraints specified in the text of the rule, as detailed in Section 3.3. Synthetic attack packets were generated at a rate of 10 per second on the subnet, with minimal background traffic. This was done in an effort to minimize the possibility that either an IDS would drop a packet or that a Mucus-1-generated packet would be lost due to a collision.

In these experiments, Mucus-1 randomly selected source IP addresses from a 24-bit range of addresses, and checks were made to assure that there was no collision of source IP addresses. Bulk IDS responses were matched with the traffic generated by Mucus-1 by correlating source IP addresses. In each experiment the results are broken down by protocol (TCP, UDP, and ICMP). This was done to illuminate possible differences in the handling of connection-oriented versus connectionless traffic.

A suite of "helper" applications was developed to automate the process of running experiments on the testbed and to analyze the results. These applications enabled the experimenter to evaluate Snort against traffic generated by Mucus-1 with one or two UNIX shell commands, which allowed new Mucus-1 features to be tested quickly. An analogous interface was not possible in the case of Net Prowler, which requires reports to be scheduled manually when they are needed more frequently than once an hour.

5 Experimental Results

This section describes the experiments that were run to evaluate the Mucus-1 tool and to validate the use of automatic traffic generation from signatures as a method for black-box testing of network-based IDSs.

Protocol	Total Snort Signatures	Translated Signatures	Correct Alerts for Translated Signatures (μ)
ICMP	128	80 (62.5%)	80 (100.0%)
TCP	948	739 (78.0%)	739 (100.0%)
UDP	136	130 (95.6%)	130 (100.0%)
IP	31	0 (0.0%)	0 (100.0%)
Total	1243	949 (76.3%)	949 (100.0%)

Table 2. Average rate of correct Snort alerts with stateful TCP monitoring (*stream4*) disabled (100 runs).

5.1 The Baseline Experiment

The goal of this experiment was to quantify the ability of Mucus-1 to generate synthetic traffic that will cause a Snort IDS to produce alerts. In this experiment, Snort's signatures were used as input to Mucus-1, and then the generated traffic was run against a Snort sensor. This is a means of evaluating the ability of the tool to trigger Snort alerts: If traffic corresponding to some Snort signature causes Snort to generate an alert for the same signature, then a correct classification is recorded. No other sensor behavior contributes to the evaluation.

Of the 1243 total rules in the Snort 1.8.6 distribution, all but 188 were successfully translated by Mucus-1. Some of the missing features, such as support for multiple content strings or allowing simple regular expressions in content strings, would be relatively straightforward additions to the tool. Other features, such as supporting Snort's ability to activate a rule only when another designated rule has previously fired, would require more effort.

Out of the 1055 correctly translated Snort rules, 106 resulted in traffic that was not consistently recognized by Snort as corresponding to the rule that was used to generate the traffic. This class of error is treated in detail in the next section. The remaining 949 signatures were used to evaluate the ability of Mucus-1 to repeatedly trigger specific alerts in Snort. To obtain a realistic picture of the randomized traffic generated, this experiment was run 100 times with a different random seed for each run.

Table 2 summarizes the results of this experiment, which was carried out with stateful TCP monitoring disabled (Snort's default setting). The complete Snort 1.8.6 signature set contains 128 ICMP rules, 948 TCP rules, 136 UDP rules, and 31 IP rules. The incidence of IP rules is mentioned here for completeness, since, as mentioned in Section 3.1, the current version of Mucus-1 does not generate synthetic traffic for IP protocol Snort rules. This experiment demonstrated that Mucus-1 is able to reliably translate 949 of the 1243 signatures in the Snort ruleset into network traffic that consistently fires the correct corresponding rules in Snort.

Next, the identical experiment was run, with Snort's stateful TCP monitoring (`stream4`) enabled. The `stream4` preprocessor allows Snort to disregard TCP packets that appear outside an established TCP session. Since Mucus-1 generates TCP traffic without establishing a valid TCP session, all synthetic TCP attack traffic is ignored by Snort in this experiment. However, as expected, no difference was observed with respect to the UDP and ICMP alerts generated. This illustrates one limitation of the simplistic, stateless method of generating TCP traffic with Mucus-1. Future development of Mucus will address this shortcoming.

5.2 Analysis of Mismatched Signatures and the Resulting Evasion Attack

As mentioned in the previous section, it was found that 106 (of 1243 total) Snort signatures were translated by Mucus-1 into traffic that caused Snort to generate *mismatched* alerts. That is, traffic was generated for one signature, but Snort interpreted the activity as an attack corresponding to a different signature. Further analysis showed that in many of these cases there was, in effect, overlap between multiple signatures in the ruleset.

For example, in one case Mucus-1 generated traffic satisfying the constraints for the following signature (signature 1):

```
alert udp $EXTERNAL_NET 60000 -> $HOME_NET 2140
  (msg:"BACKDOOR DeepThroat 3.1 ICQ Alert OFF
  Client Request"; content:"88";
  reference:arachnids,106; sid:140;
  classtype:misc-activity; rev:3;)
```

However, an unexpected alert was generated for signature 2:

```
alert udp $EXTERNAL_NET 60000 -> $HOME_NET 2140
  (msg:"BACKDOOR DeepThroat 3.1 System Info
  Client Request"; content:"13";
  reference:arachnids,106; sid:122;
  classtype:misc-activity; rev:3;)
```

Visual inspection of signatures 1 and 2 revealed that the two rules specify identical constraints with the exception of their content strings (i.e., `88` and `13`), which happen to be particularly short. Examination of the datagram generated

by Mucus-1 revealed that the string `13` occurred in the random padding of the payload prior to the string `88`. Since Snort aborts its search for matching rules after it encounters the first match, only an alert for signature 2 appeared.

It can be argued that the developers of the Snort signatures that were designed to detect DeepThroat activity knew that this type of confusion never occurs in practice and that even if it did occur, both rules detect related activity. However, one can also argue that the existence of underconstrained rules in the Snort ruleset represents a fundamental shortcoming in the system, particularly since Snort 1.8.6 stops searching after it finds a match. Previous work has identified poorly specified signatures as a significant contributor to the number of false positives generated by intrusion detection systems [13]. This issue takes on even greater importance if the two rules in question differ in terms of the severity of the activity they are designed to detect. For instance, the packet payloads used to deliver a buffer overflow attack could be padded with data that will set off a low-impact attack signature.

The following is a description of how this first-match behavior can be exploited, by successfully hiding an actual attack on a running system. Consider the following two signatures from the Snort 1.8.6 ruleset. The first is designed to detect traffic that exploits a remote root buffer overflow vulnerability in the Network Time Protocol service[5]:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 123
  (msg:"EXPLOIT ntpdx overflow attempt";
  dsize: >128; reference:arachnids,492;
  classtype:attempted-admin; sid:312; rev:1;) }
```

This signature fires an alert when a UDP packet from the outside network that has a payload larger than 128 bytes and is destined for port 123 on some machine in the home network is observed. It is classified as an attempt to gain administrator privileges.

Next, consider another Snort 1.8.6 signature, written to detect DeepThroat Trojan activity between the client running in the home network and an infected machine on the outside network. This signature is classified with a less serious impact – miscellaneous activity – and fires when an inbound UDP packet with source port 4120 is detected, with the string `"--Ahhhhhhhhhh"` somewhere in the payload:

```
alert udp $EXTERNAL_NET 4120 -> $HOME_NET any
  (msg:"BACKDOOR DeepThroat access";
  content: "--Ahhhhhhhhhh";
  reference:arachnids,405;
  sid:113; classtype:misc-activity; rev:3;)
```

Given these two signatures and Snort's first-match behavior, the goal was to modify the `ntpd` exploit traffic so that Snort will fire an alert for DeepThroat activity, even though a successful `ntpd` exploit is carried out. This was accomplished in two steps. First, the shell code that exploits the `ntpd` vulnerability was patched, adding a string of characters to the shell code that matches the lower impact rule. That is, the original sequence was declared in the exploit as:

```
char lin_execve[] =
  "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07"
  "\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d"
  "\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80"
  "\xe8\xdc\xff\xff\xff/tmp/sh";
```

This sequence of bytes was changed to the following sequence, which differs only in the addition of the last 13 characters:

```
char lin_execve[] =
  "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07"
  "\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d"
  "\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80"
  "\xe8\xdc\xff\xff\xff/tmp/sh--Ahhhhhhhhhh";
```

Second, the source port of the traffic was fixed to the value 4120, as specified in the DeepThroat signature. Neither of these modifications have a significant effect on the successful operation of the exploit. However, the changes do result in the generated traffic satisfying the constraints specified in *both* the high-impact (`ntpd`) and lower impact (DeepThroat) signatures. Thus, the alert that is generated by this UDP packet was found to depend on the order in which the rules appeared in Snort's input. By switching the order of the rule sets specified in Snort's configuration file, Snort predictably generated either the `ntpd` exploit alert or the DeepThroat activity alert. This distinction is critical since administrators commonly focus only on the highest impact alerts, due to limited resources.

As previously noted, Snort's matching behavior was improved in version 2.0.0 of the system. Specifically, Snort 2.0.0 considers all rules with matching protocol and source and destination ports and issues a single alert corresponding to the rule that specifies the longest content string (i.e., the most constrained rule). In principle, a similar evasion attack would then be successful against Snort 2.0.0, as long as one could find a low priority rule in the ruleset that (a) specifies the same source and destination ports and (b) contains a content string that is longer than the higher priority rules one wants to avoid triggering.

5.3 Evaluating Symantec Net Prowler Using Mucus

Section 5.1 demonstrated that Mucus-1 is able to generate traffic from Snort signatures and that, in many cases, Snort is unable to distinguish synthetic traffic from actual attack traffic. But the broader goal of this work is to apply traffic generated from one IDS signature set to other IDSs. Therefore, the second experiment examined the behavior of Symantec Net Prowler, a commercial network IDS, with respect to Mucus-1 traffic generated from Snort rules. Mucus-1 was run with the 949 signatures that produced correct

[5]This vulnerability was documented in early 2001 and is present in version 4.0.99e of `ntpd`.

Protocol	Total Snort Signatures	Translated Signatures	Recorded Alerts for Translated Signatures (μ)
ICMP	128	80 (62.5%)	8.18 (10.2%)
TCP	948	739 (78.0%)	34.0 (4.60%)
UDP	136	130 (95.6%)	59.3 (45.6%)
IP	31	0 (0.0%)	0 (100.0%)
Total	1243	949 (76.3%)	101.5 (10.7%)

Table 3. Average rate of recorded Net Prowler alerts for Mucus synthetic attack traffic (10 runs).

alerts in Snort in the baseline experiment (Table 2). This decision ensured that the experiment would be conducted with signatures Mucus-1 translates with a known and reasonable level of fidelity.

Because Mucus-1 generates randomized packets, an average measurement of the responses was made by running this experiment 10 times with a different random seed each time. Since the signatures in Net Prowler were available only in an imprecise natural language form, the bulk number of alerts generated by Net Prowler were measured, without attempting to discern the correctness of the classifications. The approach used to match alerts with Mucus-1-generated traffic was to correlate source IP addresses. In many instances Net Prowler's response was either clearly correlated with the signature used to generate the traffic (e.g., recognizing Back Orifice and Deep Throat activity) or obviously uncorrelated (e.g., alerting on the strict source route option in the IP header for traffic simulating questionable HTTP access). This behavior will not be discussed further in this paper, but it clearly merits further investigation. Another interesting observation is that Symantec produces multiple alerts for single packet attacks. This behavior is far preferable to Snort's policy of reporting only the first matching rule.

The results of this experiment are presented in Table 3. By comparison to Snort's performance (Table 2), Net Prowler produces fewer alerts overall. However, it should be emphasized that the traffic generated by Mucus-1 is *synthetic*. Therefore, alerts generated by Net Prowler in this experiment are false positives. One tempting conclusion to draw from the fact that Net Prowler generated only a few alerts is that Net Prowler is more capable of correctly identifying synthetic attacks. That is, one might like to conclude that Net Prowler is more robust against IDS stimulation attacks. An alternative explanation, however, could be that Net Prowler has a less complete, or markedly different, set of signatures and simply is not checking for the attacks represented by the synthetic signatures that were missed.

Because the authors were not able to obtain the Net Prowler signature set it was not possible to determine, for each instance of synthetic attack traffic, which of these conflicting conjectures is correct. However, one method of determining whether Net Prowler is more discerning or just has a signature set that is disjoint from Snort is to expose Net Prowler to the actual attacks that are being simulated using Mucus-1. The alerts produced by Net Prowler should then indicate which attacks it contains signatures for. The following section precisely details this experiment.

5.4 Real Attack Traffic

As mentioned in the previous section, real attacks are one way of determining whether an IDS has a signature for some attack, without the benefit of precise knowledge of the system's signatures or implementation. This knowledge allows one to evaluate (without having access to the signature set) whether an IDS exhibits robustness against IDS stimulation attacks or whether the system in question simply does not model the attacks that are being synthetically reproduced. In addition, the ultimate goal of automated attack traffic generation for network IDS evaluation is to produce traffic that is indistinguishable from the actual attacks being simulated. Comparing the responses of an IDS to synthetic and real versions of an attack is one way of measuring the fidelity of synthetically generated attacks.

In this experiment, a sampling of exploits and information gathering attacks were gathered from sources on the Internet and run against Snort and Net Prowler. Each attack has a corresponding Snort rule, which enabled synthetic traffic for each attack to be generated for comparison. It should be noted that, in some cases, the vulnerable services were not installed on the victim host, and, therefore, the exploits were not successful. However, for the purposes of this experiment, an attempted exploit is considered to be a critical event from the perspective of a network IDS. Table 4 summarizes the results of this experiment.

The results show strong agreement for the Snort IDS: In all cases considered, Snort generates alerts for both synthetic and real versions of the attacks. This indicates that, from the perspective of Snort, the synthetically generated traffic is essentially indistinguishable from the true attacks.

Table 4 also shows that the majority of attacks selected go undetected by Net Prowler. While in the previous experiment not detecting synthetically generated traffic seemed to show Net Prowler's robustness to IDS stimulation, not detecting the real attack in this experiment gives evidence to the alternate explanation, namely that Net Prowler's signature set is smaller than, or largely disjoint from, that of Snort. Agreement between the synthetic and true attack event streams is achieved in all but one instance, the HTTP

	Snort Alert Generated?		Net Prowler Alert Generated?	
Activity/Vulnerability	**Synthetic Attack**	**Real Attack**	**Synthetic Attack**	**Real Attack**
Bay/Nortel Nautica Marlin DoS	*yes*	*yes*	*no*	*no*
OpenBSD ftpd	*yes*	*yes*	*no*	*no*
Showmount (portmap request)	*yes*	*yes*	*no*	*no*
ntpd buffer overflow	*yes*	*yes*	*no*	*no*
PHP strings buffer overflow	*yes*	*yes*	*no*	*no*
HTTP phf	*yes*	*yes*	*no*	*yes*
count.cgi	*yes*	*yes*	*yes*	*yes*

Table 4. Comparison of behavior of Snort and Net Prowler against synthetic and true attack traffic.

phf attack, in which the real attack causes Net Prowler to generate an alert while the synthetic attack fails to do so. From this, one can conclude that for this attack Net Prowler has a more robust detection capability than Snort.

6 Conclusions and Future Work

A tool for cross-testing of signature-based intrusion detection systems was presented. A new methodology for the generation of test cases was also presented and a tool, called Mucus-1, was introduced. An empirical evaluation of this first proof-of-concept prototype of the tool was carried out, with interesting results.

These experiments illuminate the fact that the lack of information flow from commercial IDS vendors is detrimental to the research community. Some commercial IDS vendors refuse to disclose precise information about their signatures, while some do not disclose precise information about their algorithms. Most do not disclose either. From the perspective of an IDS evaluator, this opaqueness makes it impossible to establish the cause of errors or misdetections. Dropped packets, poor signatures, and logical errors are indistinguishable unless system internals are known.

This initial work in the area of black-box evaluation of network IDS will be extended in several ways. First, support in Mucus-1 for signature languages other than Snort would dramatically increase the scope of this work. Semantically, some languages are similar to Snort and would require a relatively minor effort to implement, while other languages permit more powerful expressions of attacks. STATL [8], for example, allows the operator to specify stateful, multi-step, time-dependent attack scenarios. Clearly, generating random instances of such scenarios implies a greater representational complexity in Mucus-1. Some ability to reason about time constraints would also be needed. The current architecture is able to accommodate such changes.

Second, in order to improve the testing capabilities of Mucus beyond qualitative measures of IDS performance, some measure of signature quality must be developed. Without knowing how closely signatures match the event streams of attacks, it is not possible to use the proposed approach to draw quantitative conclusions about the performance of an IDS.

Finally, Section 5.2 examined the problem of signature overlap combined with the first-match behavior in Snort. That is, the situation in which, for a given datagram, the specified constraints of two or more Snort rules are satisfied, but only one match is reported. A real-world example showing a remote root exploit disguised as a lower impact alert was demonstrated. In principle, it is straightforward to compute the intersection of constraints specified by any two rules in the Snort ruleset. An automated tool to perform this computation for all pairs in the Snort ruleset would be advantageous.

Acknowledgments

The research work described in this paper builds on the results achieved by several people. We would like to thank Sniph, the author of snot, for building the first extensive tool for IDS stimulation. The code of the first Mucus-1 prototype re-used portions of his code. We would also like to thank Steve Eckmann, who developed the Snort-to-STATL translator. The Mucus-1 Snort rule parser was built by extending his original work.

This research was supported by the State of California, the Army Research Office, under agreement DAAD19-01-1-0484 and by the Defense Advanced Research Projects Agency (DARPA), and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-1-0207. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

References

[1] Anzen. nidsbench:a network intrusion detection system test suite. `http://packetstorm.widexs.nl/UNIX/IDS/nidsbench/`, 1999.

[2] S. Aubert. Idswakeup. `http://www.hsc.fr/ressources/outils/idswakeup/`, 2000.

[3] S. Axelsson. The Base-Rate Fallacy and its Implications for the Difficulty of Intrusion Detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, 1999.

[4] R. Durst, T. Champion, B. Witten, E. Miller, and L. Spagnuolo. Addendum to "Testing and Evaluating Computer Intrusion Detection Systems". *CACM*, 42(9):15, September 1999.

[5] R. Durst, T. Champion, B. Witten, E. Miller, and L. Spagnuolo. Testing and Evaluating Computer Intrusion Detection Systems. *CACM*, 42(7):53–61, July 1999.

[6] S. Eckmann. Translating Snort Rules to STATL Scenarios. In *Recent Advances in Intrusion Detection*, Davis, CA, October 2001. Short paper presentation.

[7] S.T. Eckmann. *The STATL Attack Detection Language*. PhD thesis, Department of Computer Science, UCSB, Santa Barbara, CA, June 2002.

[8] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. In *Proceedings of the ACM Workshop on Intrusion Detection Systems*, Athens, Greece, November 2000.

[9] C. Giovanni. Fun with Packets: Designing a Stick. `http://www.eurocompton.net/stick/`, 2002.

[10] Paul Helman and Gunar Liepins. Statistical Foundations of Audit Trail Analysis for the Detection of Computer Misuse. In *IEEE Transactions on Software Engineering*, volume Vol 19, No. 9, pages 886–901, 1993.

[11] 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):181–199, March 1995.

[12] H. S. Javitz and A. Valdes. The NIDES Statistical Component Description and Justification. Technical report, SRI International, Menlo Park, CA, March 1994.

[13] K. Julisch. Mining alarm clusters to improve alarm handling efficiency. In *Proceedings of the 17th Annual Computer Security Applications Conference*, Orlando, FL, 2001.

[14] C. Ko, M. Ruschitzka, and K. Levitt. Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 175–187, May 1997.

[15] U. Lindqvist and P.A. Porras. Detecting Computer and Network Misuse with the Production-Based Expert System Toolset (P-BEST). In *IEEE Symposium on Security and Privacy*, pages 146–161, Oakland, California, May 1999.

[16] 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 Offline Intrusion Detection Evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition, Volume 2*, Hilton Head, SC, January 2000.

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

[18] T. Parr. Antlr - documentation. `http://www.antlr.org/`.

[19] S. Patton, W. Yurcik, and D. Doss. An Achilles' Heel in Signature-Based IDS: Squealing False Positives in SNORT. In *Proceedings of RAID 2001*, Davis, CA, October 2001.

[20] T.H. Ptacek and T.N. Newsham. Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, January 1998.

[21] M. Ranum. Experience Benchmarking Intrusion Detection Systems. NFR Security White Paper, December 2001.

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

[23] Mike Schiffman. libnet. `http://packetfactory.net/libnet/`, 2002.

[24] Sniph. Snot. http://www.sec33.com/sniph/, 2001.

[25] D. Wagner and D. Dean. Intrusion Detection via Static Analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001. IEEE Press.

[26] C. Warrender, S. Forrest, and B.A. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.