A Tool for Offline and Live Testing of Evasion Resilience in Network Intrusion Detection Systems (Extended Abstract)

Leo Juan[1], Christian Kreibich[2], Chih-Hung Lin[1], and Vern Paxson[2]

[1] Institute For Information Industry, Taipei City, Taiwan
{lichou,chlin}@nmi.iii.org.tw
[2] International Computer Science Institute, Berkeley, USA
{christian,vern}@icir.org

Abstract. In this work we undertake the creation of a framework for testing the degree to which network intrusion detection systems (NIDS) detect and handle evasion attacks. Our prototype system, `idsprobe`, takes as input a packet trace and from it constructs a configurable set of variant traces that introduce different forms of ambiguities that can lead to evasions. Our test harness then uses these variant traces in either an *offline configuration*, in which the NIDS under test reads traffic from the traces directly, or a *live* setup, in which we employ replay technology to feed traffic over a physical network past a NIDS reading directly from a network interface, and to potentially live victim machines. Summary reports of the differences in NIDS output tell the analyst to what degree the NIDS's results vary, reflecting sensitivities to (and possible detections of) different evasions. We demonstrate `idsprobe` using two popular open-source NIDSs and report on their respective abilities in dealing with evasive traffic.

1 Introduction

Network intrusion detection systems (NIDS) monitor network traffic for potential threats and successful exploits. However, such monitoring faces a fundamental problem: the traffic as observed by an intermediary such as a NIDS does *not* necessarily appear to the recipient in the same semantic terms. Instead, the recipient may either observe a *different* pattern of traffic or may impose an alternative *interpretation* on ambiguous traffic (such as two packets spanning the same sequence range in a TCP flow, but offering different payload bytes for that sequence). While attackers can actively exploit such ambiguities to confuse NIDS, ambiguities unfortunately also arise in traffic streams for benign reasons, requiring valuable analyst time for ascertaining whether the condition constitutes a threat.

Given the fundamental significance of evasion attacks for network intrusion detection, it is striking how little has been documented regarding the efficacy with which modern NIDS address the threat. Vendors publish extensive performance testing results regarding linespeed and breadth of attacks detected by a given system, but little information regarding its resilience to evasion. Because evasion constitutes a fundamental problem, however, for vendors to ignore it risks building a "house of cards": their products increasingly provide more of an appearance of security than a reliable foundation. Recently, third-party testing of NIDS products has begun to include an assessment of evasion vulnerabilities [1]. This testing environment, however, is proprietary: it is not available for inspection, modification and extension by others. In this work, we argue that there is significant utility for the network security community at large to have an easy-to-use, transparent, open-source environment for testing NIDS for resilience in the presence of evasion.

To this end, we have designed and implemented a framework, termed `idsprobe`, to facilitate the creation of evasion test-cases in a pluggable fashion, coupled with fully automated testing of different NIDS on the resulting test-cases. In the next section, we describe the requirements that guided our system development. In § 3 we present the architecture of the overall framework, and in § 4 some initial experimental results obtained with using it. We discuss related work in § 5, and offer final thoughts as well as a look at important future work in § 6.

2 Requirements

For our evasion-testing environment we consider two sets of requirements: creating test cases, and then applying those test cases to evaluate a given NIDS.

For the former, we have the following considerations. First, the framework should support both trace-based test cases and live network operation. Trace-based test cases offer very large advantages in terms of repeatability, portability, and ease of inspection and verification of correctness. However, some forms of evasion testing *require* live testing. These include: *(i)* NIDS that gain information from end systems [2–4]; *(ii)* NIDS that employ some form of traffic *modification* to remove ambiguities to prevent evasions from exploiting them [5, 6]; and *(iii)* evasion attacks that rely on *resource exhaustion* thus causing it to drop packets and consequently miss an attack. Second, the framework needs to accommodate elementary and modular traffic transformations across the relevant layers of the protocol stack. For example, a single test case might include network-layer (e.g., fragmentation), transport-layer (e.g., ambigu-

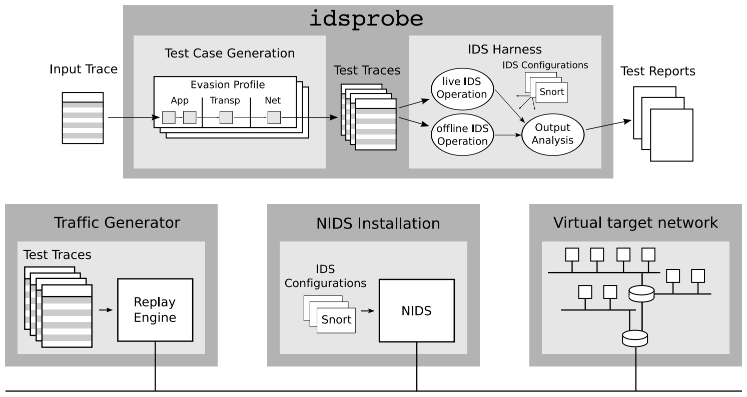

Fig. 1. The `idsprobe` framework. Top: offline testing, bottom: live environment.

ous TCP retransmissions) and application-layer (e.g., ambiguous HTTP character encoding) evasions all together.

To use the resulting test cases for evaluating a NIDS, we desire the following. First, reusability of the generated test traces for live testing. On-the-fly introduction of evasive actions to live traffic is complicated by the fact that it requires selectiveness as well as careful sequencing. The ability to leverage input traces containing ready-made evasions in live environments both reduces effort and improves reliability. Second, automation of the process of executing the NIDS and capturing its full set of outputs, including summaries of differences among individual runs. Finally, suitable postprocessors to inspect these differences to highlight patterns corresponding to susceptibility to or thwarting of evasion attempts, particularly to shed light on *architectural* issues reflected in the results (such as whether a given NIDS lacks sufficient state).

3 Framework Architecture

Figure 1 illustrates the current architecture of the `idsprobe` framework, which accommodates both offline and live testing.

3.1 Overview

For simplicity, we limit the presentation of the framework to reflect a single set of related test cases. The process begins with a single, non-evasive trace which contains some attribute, such as a particular payload string in a particular context, for which we can configure a NIDS to detect its presence. We then repeatedly apply a series of *transformation profiles* to copies of this trace to yield a set of variants, each of which reflects a particular potential evasion. After generating these traces, we then employ a "test harness" to run a set of NIDS-under-test against the traces (including the original, unmodified trace), capturing their outputs, from which we then construct a set of reports summarizing the NIDS's behavior in the presence of different evasions.

3.2 Test Case Generation

To support modularity, we encapsulate a set of elementary transformations in scripts that can be individually invoked and then subsequently composed. Each script takes as input (from a file or *stdin*) a `libpcap` trace and produces as output a new trace (to a given output file or *stdout*). In addition, the `idsprobe` framework transparently manages any temporary storage a script requires to perform the transformation, which facilitates chained application of transformations.

We currently provide tools for the following transformations:

Application layer. We support rewriting of application-layer contents using the framework developed in our previous work [7] built upon the Bro intrusion detection system [8]. This framework allows application-level specification of trace transformations that are then reflected down to the transport layer (adjustment of sequence numbers, checksums, and acknowledgments) and network layer (repacketization where required).

Transport layer. This level currently supports adjustment of relevant header control bits, payload modifications, and adjustment of checksums. We implement these using plug-ins for *Netdude* [9].

Network layer. Our current support for network-layer modifications—also based on Netdude plug-ins—comprises modification of arbitrary header fields, duplication/insertion/removal of individual packets, IP fragmentation, and checksum correction.

Trace file manipulation. We provide additional plug-ins to *(i)* adjust packet timestamps in trace files, *(ii)* correct the flow of time (sort packets with non-monotonic timestamps), and *(iii)* recombine multiple sets of packets/traces into a single trace file.

We emphasize that the scripting interface to the transformation tools can readily accommodate other tools that can provide trace manipulation at different semantic levels.

Finally, we also note a somewhat subtle point regarding composition of different evasions: multiple types of evasions need to be applied "top down" in terms of network protocol layering. That is, we must first apply application-layer transformations, then transport-layer ones, and finally those operating at the network-layer. The reason for this is that tools that manipulate one layer generally assume that the lower layer is unambiguous (and thus the tool is free to rewrite it accordingly).

3.3 Offline Evasion Testing

Once a set of test traces have been generated, the `idsprobe` framework then enables automated assessment of a number of NIDS against the suite. Adding a NIDS is a simple process: all that is required is to provide a shell script that will invoke the NIDS given a number of environment variables including, among others, the trace file to be analyzed. The test harness then invokes the script repeatedly to execute the NIDS across each of the traces in the variant set, storing the generated files separately. After execution, `idsprobe` invokes *diff*-based file-differencing to determine the degree to which the NIDS's behavior changed for given variants. Once differenced, the results currently require manual inspection to assess their significance.

3.4 Live Evasion Testing

As mentioned in Section 2, some forms of evasion testing require live tests. To facilitate these, we extended the `idsprobe` framework to function in live environments, while allowing us to re-use the evasive test traces generated for offline testing whenever possible. Three components, connected via a physical link, facilitate live testing: *(i)* a traffic generator, which establishes connections to the victim machine(s) and drives the data exchange; *(ii)* a NIDS installation which monitors the link; and *(iii)* a virtual target network which hosts the victim machines, responding to the traffic sent by the traffic generator.

The key challenge for the traffic generator is enabling re-use of the existing test traces. Our approach is to replay traces *adaptively*, relying on the causality of exchanged application data units (ADUs) at the application level and to ignore the actual content of the responder's ADUs, while patching up the sequence and acknowledgement numbers in the input trace's packets to keep the TCP exchange working. We used the `scapy` packet processing tool [10] to build this replay functionality.

We used `honeyd` [11] to realize the virtual target network. `honeyd` provides the major benefits of allowing easy adjustment of the network topology (for example in order to introduce additional routers for reachability evasions relying on the IP TTL field), while providing flexible victim responder configurations. ranging from simple shell scripts to forwarding to live external systems via `honeyd`'s subsystem mechanism.

4 Initial Experimental Results

As a preliminary evaluation of the `idsprobe` framework, we developed an initial set of 10 different types of test cases. We evaluated each against the *Snort* [12] (version 2.6.1.4) and *Bro* [8] (version 1.2.1) NIDSs.

4.1 Test Cases

In all test cases, we use a set of traces of entire, full-packet TCP connections. Each contains a single HTTP request with lengths ranging from 8 to 256 bytes, and a corresponding HTTP response. The main objective is to determine whether the NIDS under test can match a signature (not necessarily of an attack) that we *know* is present in the generated, evasive traffic, while also checking for any signs of evasion or other unusual activity that the NIDS might signal. `idsprobe` automatically generated 196 test traces based on 5 input traces. Table 1 shows the sets of transformations.

4.2 NIDS Configurations

For the Bro NIDS, we used its default configuration settings. We instructed it to monitor all TCP traffic (`-f tcp`) and loaded the `mt`, `frag`, and `signatures` analyzers. We configured signatures for the HTTP requests in the input traces, with each signature matching exactly one of the HTTP requests. For Snort, we removed the large list of signature file `include` directives, since none of the listed rule sets were actually included in the Snort distribution, verified that the `frag3` and `stream4` preprocessors were enabled, and that evasion-related alerts would be generated.

4.3 Findings

Output of `idsprobe`-generated traces Table 2 summarizes our findings based on the `idsprobe`-generated evasive packet traces. Overall, Bro

- **TC1** A single, consistent, and immediate retransmission 1 μsec after the original of a TCP segment carrying the signature-bearing application-layer payload. This test case checks whether the NIDS performs a simple form of TCP stream reassembly correctly.
- **TC2** Like TC1, but the retransmission consists of only part of the original TCP segment. We retransmit a right-aligned part of the original segment with correct checksum and sequence number. This constellation likewise presents neither threat nor ambiguity.
- **TC3** Like TC1, but we change the TCP payload on the first (subtest TC3a) or the second (subtest TC3b) variant of the duplicated packet, respectively, without any checksum corrections. This test does not pose any actual ambiguity.
- **TC4** Like TC3, except now the checksums are corrected. Our payload modification is *careless*, thus leading to a different checksum value. This test case represents the first truly ambiguous traffic. The NIDS needs to decide which version of the byte stream to analyze, and ideally should note the inconsistency.
- **TC5** Like TC4, but we change the TCP payload *carefully*, leaving the checksum unchanged. We achieve this by swapping 16-bit fields, though one could derive more complex modifications due to the incremental nature of the checksumming algorithm. As with TC3, this presents a real ambiguity, requiring the NIDS to compare the actual payloads.
- **TC6** We duplicate one of the IP datagrams in the TCP flow, setting its IP fragment offset to a non-zero offset value (adjusting the IP header checksum to reflect the change) on the first (subtest TC6a) or second (subtest TC6b) variant, respectively. This test case creates an ambiguous, malformed fragment.
- **TC7** We duplicate one of the IP datagrams in the TCP flow and set its IP TTL value to a number of different values (again with header checksum updated) on the first (subtest TC7a) or second (subtest TC7b) variant, respectively. This test case does not introduce a serious ambiguity but can confuse NIDS evasion detection that examines TTL values for anomalies.
- **TC8** We consistently fragment one of the IP datagrams in the TCP flow carrying the signature-bearing payload, using various different fragment sizes. This test case tests whether the NIDS correctly processes well-formed fragments.
- **TC9** Like TC8, but we duplicate one of the fragments, and alter its payload in the first (subtest TC8a) or second (subtest TC8b) variant, respectively. The alteration is again careless, i.e., reassembly of the datagram using the modified payload leads to an incorrect TCP checksum for the full datagram.
- **TC10** Like TC9, but with a careful payload alteration, i.e., reassembly of the datagram using the modified payload leaves the TCP checksum unchanged. Figures 2 and 3 present the workings of TC10 in detail.

Table 1. Test cases used for evaluating `idsprobe`.

and Snort performed similarly as far as signature detection is concerned. They differ, however, in the amount of detail delivered in addition to the relevant alerts. After excluding from file-differencing Bro's `.state` directories (which remain empty) and Snort's *tcpdump* log files, the total amount of difference in Bro's output amounts to 1,665 lines, as opposed to 17,018 for Snort. Ignoring Snort's verbose summary output reported on *stdout* and *stderr* reduced the differential data volume to 1,329 lines.

```
00:01:37.427628 10.48.0.1.2013 > 10.48.0.81.80: . 1:158(157) ack 1 win 32768
0x0000   4500 00c5 7566 0000 4006 f01b 0a30 0001        E...uf..@....0..
0x0010   0a30 0051 07dd 0050 0000 092a 3838 4e57        .0.Q...P...*88NW
0x0020   5010 8000 0fc2 0000 4745 5420 2f31 6162        P.......GET./1ab
0x0030   6364 6566 6768 696a 6b6c 6d6e 6f70 7172        cdefghijklmnopqr
0x0040   7374 7576 7778 797a 3261 6263 6465 6667        stuvwxyz2abcdefg
0x0050   6869 6a6b 6c6d 6e6f 7071 7273 7475 7677        hijklmnopqrstuvw
0x0060   7879 7a33 6162 6364 6566 6768 696a 6b6c        xyz3abcdefghijkl
0x0070   6d6e 6f70 7172 7374 7576 7778 797a 3461        mnopqrstuvwxyz4a
0x0080   6263 6465 6667 6869 6a6b 6c6d 6e6f 7071        bcdefghijklmnopq
0x0090   7273 7475 7677 7879 7a35 6162 6364 6566        rstuvwxyz5abcdef
0x00a0   6768 696a 6b6c 6d6e 6f70 7172 7320 4854        ghijklmnopqrs.HT
0x00b0   5450 2f31 2e31 0d0a 484f 5354 3a6e 6f6e        TP/1.1..HOST:non
0x00c0   650d 0a0d 0a                                   e....
```

Fig. 2. *tcpdump* output for relevant packet from the TC10 input trace. A single TCP segment contains the relevant application-layer content, "`GET /1abcdef`".

```
00:01:37.427628 10.48.0.1.2013 > 10.48.0.81.80: [|tcp] (frag 30054:8@0+)
00:01:37.427629 10.48.0.1 > 10.48.0.81: tcp (frag 30054:8@8+)
00:01:37.427630 10.48.0.1 > 10.48.0.81: tcp (frag 30054:8@16+)
0x0000   4500 001c 7566 2002 4006 d0c2 0a30 0001        E...uf..@....0..
0x0010   0a30 0051 0fc2 0000 4745 5420                  .0.Q....GET.
00:01:37.427631 10.48.0.1 > 10.48.0.81: tcp (frag 30054:8@24+)
0x0000   4500 001c 7566 2003 4006 d0c1 0a30 0001        E...uf..@....0..
0x0010   0a30 0051 2f31 6162 6364 6566                  .0.Q/1abcdef
00:01:37.427632 10.48.0.1 > 10.48.0.81: tcp (frag 30054:8@24+)
0x0000   4500 001c 7566 2003 4006 d0c1 0a30 0001        E...uf..@....0..
0x0010   0a30 0051 2f31 6364 6162 6566                  .0.Q/1cdabef
00:01:37.427633 10.48.0.1 > 10.48.0.81: tcp (frag 30054:8@32+)
0x0000   4500 001c 7566 2004 4006 d0c0 0a30 0001        E...uf..@....0..
0x0010   0a30 0051 6768 696a 6b6c 6d6e                  .0.Qghijklmn
00:01:37.427634 10.48.0.1 > 10.48.0.81: tcp (frag 30054:8@40+)
00:01:37.427635 10.48.0.1 > 10.48.0.81: tcp (frag 30054:8@48+)
...
00:01:37.427650 10.48.0.1 > 10.48.0.81: tcp (frag 30054:8@168+)
00:01:37.427651 10.48.0.1 > 10.48.0.81: tcp (frag 30054:1@176)
```

Fig. 3. *tcpdump* output of resulting TC10 evasive traffic. The TCP segment shown in Figure 2 has its application-layer content rewritten, fragmented into 24 8-byte fragments, with a duplicate fragment with the original TCP stream content inserted after the third fragment. The sensitive payload is now spread across three IP datagrams. The payload variation preserves the TCP checksum's validity. Finally, `idsprobe` patches the packet timestamps to preserve chronological ordering.

	Output	TC1	TC2	TC3a/b	TC4a/b	TC5a/b	TC6a/b	TC7a/b	TC8	TC9a/b	TC10a/b
Bro	Sig. match	✓	✓	✓	✗/✓	✗/✓	✓	✓	✓	✗/✓	✗/✓
	Evasion			③	①	①			④	②④	②④
Snort	Sig. match	✓	✓	✓	✗/✓	✗/✓	✓	✓	✓	✗/✓	✗/✓
	Evasion	⑤	⑥		⑤			⑤		⑦	

Table 2. Bro's vs. Snort's results on 10 test cases generated by `idsprobe`. The first line per NIDS summarizes signature detection, the second reports evasion-related alerts or messages. The numbers reflect the following: ① "RetransmissionInconsistency". ② "WeirdActivity" of type "fragment_inconsistency". ③ Bad checksums in weird.log. ④ "WeirdActivity" of type "excessively_small_fragment" for fragments of 32 bytes or less. ⑤ "Possible evasive FIN detection" with nonsensical parameters. ⑥ "TCP checksum changed on retransmission". ⑦ "Fragmentation overlap".

In TC2, Snort erroneously reported a TCP checksum change on a retransmission, where in fact no divergent payload was transferred. In the event of careful payload alterations that do not affect the TCP checksum

```
08:00:09.176192 IP 10.48.0.1.2010 > 10.48.0.81.80: . 1:13(12) ack 1 win 32768
        0x0000:  4500 0034 f178 0000 4006 749a 0a30 0001  E..4.x..@.t..0..
        0x0010:  0a30 0051 07da 0050 0000 092a 3392 88d8  .0.Q...P...*3...
        0x0020:  5010 8000 4582 0000 4745 5420 2f31 6162  P...E...GET./1ab
        0x0030:  6364 7878                                cdxx
08:00:09.176194 IP 10.48.0.1.2010 > 10.48.0.81.80: . 11:14(3) ack 1 win 32768
        0x0000:  4500 002b f178 0000 4006 74a3 0a30 0001  E..+.x..@.t..0..
        0x0010:  0a30 0051 07da 0050 0000 0934 3392 88d8  .0.Q...P...43...
        0x0020:  5010 8000 80f0 0000 6566 67              P.......efg
```

Fig. 4. *tcpdump* of inconsistent retransmission not reported by Snort 2.8.0.1.

(TC5/TC10), however, Snort fails to notice the (rather likely) evasion attempt. Bro handled both cases correctly, remaining silent on the former but alerting on the latter case. Snort also generated a total of 60 potential evasive TCP FIN detections in 4 of the test cases. A number of the values reported in these alerts are nonsensical, such as IP TTL values of 240, IP ToS fields with values 0x10, and IP IDs of 0. None of these values exist in the FIN packets in question; in addition, none of the traces actually reflects an ambiguous TCP FIN packet.

Bro correctly reports TCP retransmission inconsistencies, IP fragment inconsistencies, the presence of bad checksums, and the presence of excessively small IP fragments. For Snort, the only correct evasion-related output concerns IP fragmentation overlap.[3]

During the course of our work, new releases of Snort appeared. We experimented with the latest release available, Snort version 2.8.0.1, to see how its behavior might have changed. The erroneous evasive FIN alerts have been repaired. However, the new stream reassembly module `stream5` introduced new issues: a partially overlapping retransmission (shown in Figure 4) is not reported, while Snort 2.6.1.4 did report a changed TCP checksum on the retransmission.

Output after long-term operation To better understand the usability of evasion/anomaly-related events reported by different IDSs, we ran Bro 1.2.1 along with Snort 2.6.1.4 and 2.8.0.1 on a 24-hour, 21 GB trace recorded at ICSI on 16 March 2007. The NIDSs were not configured to detect attacks, but only to report anomalous or potentially evasive activity.

Table 3 summarizes our findings. The absence of consensus in the reported events is striking, particularly between Bro and the Snort versions, but to a lesser degree even between two different Snort releases. TCP SYNs with payload data seem a rare case where there is near-

[3] Even that is not the best description of the problem, since IP fragments can overlap for rare-but-benign reasons. Better would be to highlight that the overlap is inconsistent.

Bro 1.2	
14,591	ContentGap
7,546	AckAboveHole
2,249	window_recision
735	bad_TCP_checksum
460	SYN_with_data
311	possible_split_routing
290	data_before_established
98	bad_ICMP_checksum
85	above_hole_data_without_any_acks
35	connection_originator_SYN_ack
30	bad_TCP_header_len
18	inappropriate_FIN
15	SYN_seq_jump
15	premature_connection_reuse
9	active_connection_reuse
8	data_after_reset
3	SYN_inside_connection
3	SYN_after_reset
3	bad_SYN_ack
2	TCP_christmas
1	RetransmissionInconsistency
1	FIN_advanced_last_seq
1	bad_UDP_checksum
26,509	

Snort 2.6	
161,862	possible EVASIVE FIN detection
36,873	possible EVASIVE RST detection
27,384	TCP CHECKSUM CHANGED ON RETRANSMISSION
1,933	Possible RETRANSMISSION detection
458	DATA ON SYN detection
67	WARNING: ICMP Original IP Header Truncated!
30	WARNING: TCP Data Offset is less than 5!
18	Truncated Tcp Options
12	Experimental Tcp Options found
2	Tcp Options found with bad lengths
1	WARNING: ICMP Original IP Fragmented and Offset Not 0!
228,640	

Snort 2.8	
4,844	TCP Timestamp is outside of PAWS window
2,058	Data sent on stream not accepting data
807	Bad segment, adjusted size <= 0
461	Data on SYN packet
67	WARNING: ICMP Original IP Header Truncated!
30	WARNING: TCP Data Offset is less than 5!
18	Truncated Tcp Options
12	Experimental Tcp Options found
5	Data sent on stream after TCP Reset
2	Tcp Options found with bad lengths
1	WARNING: ICMP Original IP Fragmented and Offset Not 0!
8,305	

Table 3. Aggregate summaries of anomalies and evasion-related events reported by the NIDSs under test, on the 24h ICSI trace.

consensus, with the three NIDSs reporting 460, 458, and 461 instances, respectively. Bro reports a single retransmission inconsistency (which we have verified to be correct, but it does not reflect a malicious evasion). Snort 2.6 reports this as one of 36,873 "possible EVASIVE RST detection" events, and Snort 2.8 as 3 of the 5 "Data sent on stream after TCP Reset" events recorded. For the 22,137 flow reassembly issues reported by Bro ("ContentGap" and "AckAboveHole"), which have direct significance for content-based analysis, there is no apparent corresponding alert in either of the Snort logs. These events account for the main reason why Snort 2.8 reports fewer events than Bro, whose output volume is almost an order of magnitude below Snort 2.6's.

5 Related Work

The fundamental problem of NIDS evasion was first framed in the seminal paper by Ptacek and Newsham [13]. Aspects of the problem also appear in the discussion of the Bro system [8], particularly in the context of inconsistent TCP retransmissions. In response to the threat of evasion, researchers have developed several types of countermeasures, such as traffic normalization [5, 6], active mapping [2], passive fingerprinting [4], and the use of host-based context [3].

Several tools have been developed for testing NIDS for vulnerabilities to evasion. Fragrouter[4] implements some network-layer evasions based on IP fragmentation. Unlike our framework, it modifies live traffic only. The libwhisker[5] library provides basic functionality for testing HTTP implementations. Nikto[6] leverages the library, adding HTTP content obfuscation techniques. Both tools primarily target live-traffic operation.

Regarding systematic evaluation of NIDS in the presence of possible evasions, Vigna and colleagues present a framework for NIDS testing based on traffic transformation [14]. Rather than testing the NIDSs' awareness of evasion, they emphasize evaluating the robustness of individual signatures used by such NIDSs. Their system takes as input an attack trace, to which it applies semantically invariant transformations and then and monitors for changes in the alerts generated by the NIDSs.

[4] Per `http://www.securityfocus.com/tools/176`, nominally available at `http://www.anzen.com/research/nidsbench/`, but in fact that location no longer resolves.

[5] `http://www.wiretrip.net/rfp/libwhisker/`

[6] `http://www.cirt.net/code/nikto.shtml`

Similarly, Rubin et al. developed a framework to facilitate traffic transformations on different network layers [15], again aiming to produce variants of a specific attack. Marty [16] similarly proposed a platform for subjecting NIDS to automatically generated variations of attack traffic. His system exclusively operates on live traffic.

In contrast to these efforts, our framework does not assume the existence of an attack, but instead determines the general effects of traffic transformations. This allows us to separate the NIDS's specific attack detection logic from its architectural analysis limitations. In addition, the work of Rubin et al. develops a formal model of possible transformations, which allows them to exhaustively test a NIDS against attack variants. Our work, on the other hand, aims to facilitate a public, open-source effort for developing NIDS evasion test suites, with a related emphasis for our framework on modularity and a plug-in architecture.

6 Discussion and Future Work

The `idsprobe` framework does *not* attempt to provide "turnkey" evaluation of NIDS evasion vulnerabilities. Rather, our aim is to provide the means for an experienced assessor to more readily construct good test cases, and more efficiently apply those test cases in a repeatable fashion across a set of NIDS under consideration. We also do not strive to ourselves provide a *comprehensive* set of evasion tests; rather, we aim to facilitate that others can *collectively* work towards such a goal. These considerations motivate our open-source, modular/plug-in approach.

The focus of our future work is to devise methodologies for assessing live-traffic evasions based on overloading NIDS resources and to assess the efficacy of on-line anti-evasion technology.

7 Summary

We have designed and implemented the `idsprobe` framework to facilitate the creation of offline as well as live evasion test-cases in a pluggable fashion, coupled with fully automated testing of different NIDS on the resulting test-cases. We aim for the system to encourage extension and broad use by the community, and to this end will provide the software to others upon request, and ultimately aim to maintain it in as a public open-source resource.

8 Acknowledgments

This work was partially supported by the iCAST project sponsored by the National Science Council (NSC), Taiwan, under Grants 95-3114-P-001-002-Y02, 95-3114-P-307-003-Y, 96-3114-P-001-002-Y, 95-2221-E-017-007, 95-3113-P-017-001, as well as by the Ministry of Economic Affairs, Taiwan, under Grant 96-EC-17-A-31-F1-0824, and by the US National Science Foundation, under grant NSF-0433702. Christian Kreibich was supported by a postdoctoral grant provided by DAAD. The opinions expressed in this work are solely those of the authors and should not necessarily be considered to be the opinions of any government, funding agency, or other organization.

References

1. Group, N.: Network IPS Testing Procedure (V4.0) (2006) `http://www.nss.co.uk/certification/ips/nss-nips-v40-testproc.pdf`.
2. Shankar, U., Paxson, V.: Active mapping: resisting NIDS evasion without altering traffic. Proc. Symposium on Security and Privacy (2003) 44–61
3. Dreger, H., Kreibich, C., Paxson, V., Sommer, R.: Enhancing the accuracy of network-based intrusion detection with host-based context. In: Proc. Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA). (2005)
4. Taleck, G.: Ambiguity resolution via passive os fingerprinting. In: Proc. Conference on Recent Advances in Intrusion Detection (RAID). (2003) 192–206
5. Handley, M., Paxson, V., Kreibich, C.: Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. Proc. USENIX Security Symposium (2001)
6. Watson, D., Smart, M., Malan, G.R., Jahanian, F.: Protocol Scrubbing: Network Security through Transparent Flow Modification. IEEE/ACM Transactions on Networking **12**(2) (April 2004) 261–273
7. Pang, R., Paxson, V.: A High-Level Programming Environment for Packet Trace Anonymization and Transformation. In: Proceedings of the ACM SIGCOMM Conference. (August 2003)
8. Paxson, V.: Bro: A system for detecting network intruders in real-time. Computer Networks **31**(23-24) (1999) 2435–2463
9. Kreibich, C.: Design and Implementation of Netdude, a Framework for Packet Trace Manipulation. Proc. USENIX Technical Conference, FREENIX track (2004)
10. Biondi, P.: Scapy, a powerful interactive packet manipulation program `http://www.secdev.org/projects/scapy/`.
11. Provos, N.: A Virtual Honeypot Framework. Proceedings of the 13th USENIX Security Symposium (2004) 1–14
12. SourceFire: Snort, the Open Source Network Intrusion Detection System `http://www.snort.org/`.
13. Ptacek, T., Newsham, T.: Insertion, evasion, and denial of service: Eluding network intrusion detection. Secure Networks, Inc., Jan (1998)
14. Vigna, G., Robertson, W., Balzarotti, D.: Testing network-based intrusion detection signatures using mutant exploits. Proceedings of the 11th ACM Conference on Computer and Communications Security (2004) 21–30
15. Rubin, S., Jha, S., Miller, B.: Automatic Generation and Analysis of NIDS Attacks. Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)-Volume 00 (2004) 28–38
16. Marty, R.: Thor – A Tool to Test Intrusion Detection Systems by Variations of Attacks. Master's thesis, Swiss Federal Institute of Technology, Zurich, Switzerland (2002)