Evasion Techniques: Sneaking through Your Intrusion Detection/Prevention Systems

Tsung-Huan Cheng, Ying-Dar Lin, *Senior Member, IEEE,* Yuan-Cheng Lai, and Po-Ching Lin, *Member, IEEE*

***Abstract*—Detecting attacks disguised by *evasion* techniques is a challenge for signature-based Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs). This study examines five common evasion techniques to determine their ability to evade recent systems. The *denial-of-service* (DoS) attack attempts to disable a system by exhausting its resources. *Packet splitting* tries to chop data into small packets, so that a system may not completely reassemble the packets for signature matching. *Duplicate insertion* can mislead a system if the system and the target host discard different TCP/IP packets with a duplicate offset or sequence. *Payload mutation* fools a system with a mutative payload. *Shellcode mutation* transforms an attacker's shellcode to escape signature detection. This study assesses the effectiveness of these techniques on three recent signature-based systems, and among them, explains why Snort can be evaded. The results indicate that duplicate insertion becomes less effective on recent systems, but packet splitting, payload mutation and shellcode mutation can be still effective against them.**

***Index Terms*—IDS/IPS, evasion, attacks, signature.**

I. Introduction

THE WAR between attackers and IPS[1] developers never ceases. Attackers continually try to find new exploits to intrude a system, while system developers attempt to analyse and detect attacks. IPSs are traditionally divided into two categories in terms of how they detect attacks: *signature-based* and *anomaly-based*. The former ones look for signatures of known attacks in the network traffic [1], so they require frequent signature updates to maintain an up-to-date signature database. The latter ones use machine-learning approaches to find *anomaly* in the network traffic that behaves differently from normal profiles [2]. Instead of inspecting the payload content for signatures, they typically analyse statistics such as throughput, payload size, and the number of flows or established connections in the network traffic, as well as their states [3]. The alert messages generated from them indicate the existence of anomaly, but cannot point out exactly the attack names and their associated CVE (Common Vulnerabilities and Exposures) numbers.

IPSs in both categories face a challenge from attackers: an arsenal of *evasion* techniques. First introduced by Ptacek and Newsham [4], the techniques include several ways to escape the detection of an IPS. The success of evasion relies on that the IPS and the end host (i.e., the victim) may interpret or process the packet content in the input traffic differently, due to different system implementations or depths in which both systems interpret the packet content. Ever since [4] and [5], there have been substantial evaluations (e.g., [6]–[10]) and studies (e.g., [12]–[19]) on this topic. Evasion techniques have evolved from exploiting tactics at the TCP/IP layer [4] to transforming *application-layer* messages, such as HTTP IDS evasion [13], polymorphic shellcode [14], polymorphic worms [15] and evading an emulation detection mechanism [16], [20], as well as polymorphic blending attacks [21] that make attacks similar to normal traffic for evading anomaly-based IPSs.

On the other hand, IPS researchers and developers have proposed countermeasures such as [18], [24]–[27], [31], [32] to enhance the anti-evasion capability of intrusion detection for resolving the ambiguity in interpreting the packet content due to evasion. Thus, it is necessary for network testers and product reviewers to re-examine the effectiveness of evasion techniques on modern IPS products. This work not only reviews evasion techniques and their countermeasures, but also conducts actual evasions to assess three modern IPS products or open-source packages, and points out which evasion techniques are still effective on them, which are not, and why. The assessment can help IPS developers to improve the products and network administrators to understand the potential limitations of IPS products.

The remainder of this article is organized as follows. Section II reviews the evasion techniques, the relevant tools, and the countermeasures against the evasion techniques. Section III assesses the effectiveness of the evasion techniques by conducting several experiments on three recent IPS products. Experimental results in Section IV reveal the weak spots of the IPS products, which evasion techniques are still effective, and the reasons. Section V concludes this work.

II. Survey of Existing Evasion Techniques, Tools and Countermeasures

In this section, we review the evasion techniques, the tools that can realize the techniques, and the countermeasures against the techniques.

Manuscript received 23 June 2010; revised 14 August 2011, 6 September 2011 and 7 September 2011. This work was supported in part by National Science Council (NSC) and Institute for Information Industry (III) in Taiwan, and in part by grants from ZyXEL Inc. and Cisco Systems.

T.-H. Cheng and Y.-D. Lin are with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, 300 (e-mail: {raijin, ydlin}@cs.nctu.edu.tw).

Y.-C. Lai is with the Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan, 106 (e-mail: laiyc@cs.ntust.edu.tw).

P.-C. Lin is with the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, 621 (e-mail: pclin@cs.ccu.edu.tw).

Digital Object Identifier 10.1109/SURV.2011.092311.00082

[1]For simplicity in terminology, we use the term IPS to denote both IDS and IPS in the rest of this paper.

A. Evasion techniques

Five common techniques can evade the examination of an IPS: (1) denial-of-service (DoS), (2) packet splitting, (3) duplicate insertion, (4) payload mutation, and (5) shellcode mutation. We will introduce each of them as follows.

1) Denial-of-service: The first is the denial-of-service (DoS) attack, which intends to overwhelm network bandwidth or system resources such as the CPU and the memory space of the IPS [33]. Besides generating a large volume of network traffic, the attack can exploit the weakness of the detection algorithms. An example in [34] demonstrates that it is possible to significantly slow down the rule-matching algorithm in Snort 2.4.3 by manipulating the input network traffic to exploit the worst-case execution of the rule matching in Snort, which uses backtracking in an attempt to cover all possible pattern matches in a rule. Just 4.0kbps of bandwidth for the manipulated input traffic is sufficient for the DoS attack. This *algorithmic complexity attack* is a typical example of DoS attacks without consuming high network bandwidth.

2) Packet splitting: Packet splitting, which includes *IP fragmentation* and *TCP segmentation*, chops IP datagrams or TCP stream into *non-overlapping* fragments or segments, particularly small ones. If an IPS does not completely reassemble the IP fragments or TCP segments to restore the original application content, it may neglect an attack embedded in the content targeted at the victim host. For example, an IPS may look for a signature `/bin/sh` in the packet payload, but the attacker can deliberately split the payload with the signature into two segments, one containing `/bin` and the other containing `/sh`. If the IPS does not reassemble the two segments, it will be unable to find the signature in either segment, so the attacker can evade its detection.

Since the IPS monitors all the traffic through the network under its supervision, ideally, it has to reassemble all the IP fragments and TCP segments in the traffic to counter potential evasion. However, an IPS may have limited system resources to keep track of *per-connection* information [35] (e.g., the TCP states and the reassembled application content) in a large network. For example, the space allocated to a buffer for reassembly may be insufficient. Therefore, an attack may still have a chance to appear in the IP fragments or TCP segments that an IPS happens to not reassemble.

3) Duplicate insertion: Duplicate insertion is a technique in which attackers insert *duplicate* or *overlapping* segments (or IP fragments) to confuse the IPS. This technique depends on that the IPS and victim may handle the duplicate/overlapping fragments or segments[2] inconsistently because the IPS lacks related information such as network topology and the victim's operating system. Figure 1 illustrates how the technique works with a trivial example. In this example, the attacker inserts segments with small Time-To-Live (TTL) values (the two `X`'s in the figure), so that they will be dropped before reaching the target. If the IPS cannot predict whether the segments will reach the victim, it will be unable to consistently reassemble the segments and see the same content as the victim.

[2]This work classifies the evasion techniques into packet splitting or duplicate insertion according to whether the fragments/segments are non-overlapping or duplicate/overlapping.

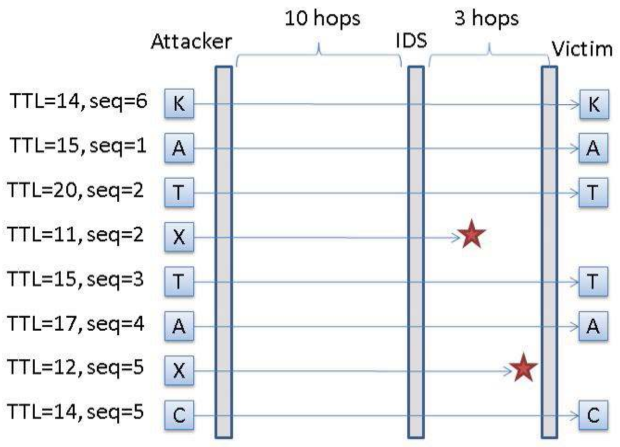


Fig. 1. Duplicate insertion with small Time-to-Live (TTL) values.

Overlapping IP fragments and TCP segments can be ambiguous to an IPS. For example, suppose that a TCP segment carries the sequence number 10 and the content `ATTXYZ`, and another segment in the same connection carries the sequence number 13 and the content `ACK`. The victim host will interpret the application content as either `ATTACK` or `ATTXYZ` upon receiving the two segments, depending on the victim's operating system [18]. Without knowing the operating system, an IPS will be likely to interpret the application content inconsistently with the victim. An attacker therefore can leverage the ambiguity to evade the detection.

Ideally, administrators can configure the policy to set how the operating system on each host in the internal network will interpret the packets for a certain ambiguity, so that the IPS can interpret the incoming packets consistently. Manual configuration is error-prone, so the work in [18] proposes an *active mapping* method to actively test each host and derive the policy. Several factors can complicate the mapping in practice. For example, associating an IP address to a host is not one-to-one with the use of NAT and DHCP. The active testing may be imprecise due to packet filtering by firewalls or unexpected packet drops on an intermediate router when the traffic volume through it is high.

4) Payload mutation: Payload mutation means an attacker transforms malicious packet payloads into semantically equivalent ones. The transformed payloads will look different from the signatures that an IPS expects, so the attack can evade the detection. Since the semantics of the transformed payloads is the same, the attack is still effective to the victim. For example, an attack in the Uniform Resource Identifier (URI) of an HTTP request can be transformed into several mutated expressions using the `libwhisker` library (www.wiretrip.net/rfp/txt/whiskerids.html). The transformation can be URI hexadecimal encoding, self-reference directories, reverse traversal directories and so on (see the aforementioned Web page for the details) to represent the URIs in several semantically equivalent forms. Take URI hexadecimal encoding as an example. If the directory name `cgi-bin` in the request is encoded as `%63%67%69%2d%62%69%6e`, detecting the signature `cgi-bin` will fail if the mutated representation is not normalized into `cgi-bin` before the IPS detection. A

TABLE I
THE EVASION TOOLS.

Evasion technique	Tool name
packet splitting	Fragroute, Sploit
duplicate insertion	Fragroute, Sploit
payload mutation	Nikto, Sploit, Havij
shellcode mutation	ADMmutate, Sploit, Metasploit

string of SQL injection can also be encoded in a similar way for evasion.

Like normalization for duplicate insertion, normalization for payload mutation can be ambiguous because the destination hosts may process the content differently, depending on the applications. For example, an Apache Web server does not accept backslashes as legitimate slashes, but a Microsoft IIS server does. Therefore, administrators have to configure the IPS to be aware of the applications on the destination hosts and have a consistent view of the application payloads as the destinations.

5) Shellcode mutation: Shellcode mutation encodes a *shellcode* (i.e., a piece of code to exploit a software vulnerability) into *polymorphic* forms to evade an IPS that detects a shellcode according to the signatures extracted from one or a few variants of that shellcode. Several methods are feasible for the polymorphism. For example, an attacker can encrypt or compress the shellcode, and prepend a piece of code to decrypt or decompress the shellcode in the exploit. An attacker can also replace a piece of the original code with different, but semantically equivalent instructions. A trivial example in the latter case is inserting the `nop` instructions, i.e., no operation, to make the code look different. An instruction, say `mov eax, ebx`, can be also replaced with two instructions `push ebx` and `pop eax`, for example. Since the signature for the shellcode does not appear in the polymorphic form, the IPS will fail to detect it. The techniques are also found in malicious programs such as viruses and worms. The authors in [36] offered a survey of such evasion techniques.

Since there are too many ways to mutate a shellcode, detecting polymorphic codes on an IPS is particularly tricky. An IPS may need to decrypt the encrypted code to restore the original signature, or even emulate the code execution (e.g., on a sandbox that emulates the execution on the target hosts) to find malicious behavior. The tasks of restore the shellcode semantics on-line are therefore computationally expensive, and burden the load of an IPS.

B. Evasion tools

Table I summaries several well-known free tools for the aforementioned evasion techniques: *Fragroute* (www.monkey.org/~dugsong/fragroute), which can exploit TCP/IP protocols, (2) *Nikto* (cirt.net/nikto2), which can transform URI requests, (3) *ADMmutate* (www.databaseofspyware.com/ADMmutate.php), which can transform the shellcode of attacks, (4) *Sploit* [8], which can provide a framework for most of the evasion techniques above, (5) *Metasploit* (www.metasploit.com), which can provide many shell-mutation encoders, and (6) Havij (itsecteam.com/en/projects/project1.htm), which can launch SQL injection attacks with mutated strings.

Fragroute implements packet splitting and duplicate insertion at the TCP/IP layer and helps the attackers evade signature matching on the IPSs. The attackers can write a simple script to arrange the sequence of evasion techniques to be launched before running Fragroute, which then automatically transforms the attack traffic into the specified format to cheat the IPSs.

Nikto, a Web scanning tool for generating multiple malicious URI requests, helps developers and network administrators test their Web servers for possible security problems. This tool also provides anti-IPS methods that can exercise payload mutation to help a Web scanner evade IPSs. The attackers can leverage these methods for evasion with payload mutation.

ADMmutate is an engine for shellcode mutation that helps an attack program evade IPSs. The engine can obfuscate the appearance of a piece of shellcode, but retain its effectiveness to exploit a software vulnerability. A program compiled with the ADMmutate API can generate different forms of shellcode to confuse a signature-based IPS.

Sploit is an evasion testing framework that allows the testers and attackers to develop new attacks and evasion techniques. This tool includes most of the evasion techniques provided by Fragroute, Nikto and ADMmutate. It also implements payload mutation on the messages of application protocols such as FTP and IMAP. For example, `\xff\xf1` is a telnet control sequence that denotes "no operation", so a `PASS` FTP command can be replaced with `P\xff\xf1ASS` to evade detection by an IPS matching the pattern `PASS` [8].

Metasploit is a framework for penetration testing, IDS signature development, and exploit research. It supports development of exploit codes against a remote target host. The Metasploit Framework also provides more polymorphic shellcode encoders [11] than ADMmutate and Sploit. The encoders allows a penetration tester to mutate the exploit codes for evasion testing.

Havij is an automated SQL injection tool to exploit SQL injection vulnerabilities on Web pages. It supports evasion by manipulating white space in the attack strings, replacing space with the comment syntax of the C language, encoding the characters in hexadecimal, BASE64, and so on.

Besides the evasion tools listed in Table I, a few other tools can also generate evasion traffic, such as FTester (dev.inversepath.com/trac/ftester), idsprobe [37] and AGENT [38]. These tools can play several combinations of evasion techniques based on packet splitting and duplicate insertion, as well as a few payload mutations.

C. Countermeasures against evasion techniques

In this subsection, we will review the countermeasures against the aforementioned evasion techniques.

1) Countermeasures against DoS attacks: In the preceding example of DoS attacks, the authors of [34] have proposed a remedy by memorizing the intermediate matches to reduce the time complexity in the worst case. For countering DoS attacks in general, the administrators can increase available bandwidth, use IPS clustering, and so on to enhance the robustness to a DoS attack beforehand, while monitoring the resource consumption of the IPS execution. If a DoS attack happens, the administrators can block certain malicious input traffic and investigate the attack vector. They may report it to the IPS developers to see whether the IPS implementation is vulnerable to an algorithmic complexity attack.

2) Countermeasures against packet splitting and duplicate insertion: Countering packet splitting relies on packet reassembly on the IPS to restore the packet content. A few research works have proposed signature-matching algorithms without packet reassembly to relieve the additional computation of packet reassembly [31], [32]. The basic ideas of these algorithms are splitting the signatures into short ones and matching them in the split segments without reassembly. However, the algorithms do not deal with the other evasion techniques. If an attacker uses other techniques such as payload mutation, the signature-matching algorithms will still fail to detect an attack.

Two main techniques can counter duplicate insertion. The first is flow modification (e.g., [5], [22], [39]), which removes protocol ambiguities by picking one interpretation of the protocols and normalizing network traffic according to the interpretation, so that the IPS and the victims can see the traffic consistently. The memory consumption in the normalization can be a problem if the normalizer on high-speed links buffers unacknowledged packets to detect data in TCP segments for the same sequence number. The authors in [39] present a hashing method for unacknowledged segments to reduce memory consumption in the detection.

The second technique is collecting networking policies of the destination. The methods include *communicating sequential processes* (CSP) [17], *active mapping* [18], and *ambiguity resolution* via passive OS fingerprinting [19]. An IPS can therefore obtain the environment information (e.g., the hop count from the IPS to the victim, the system on the victim, etc.), and view the traffic from the victim's perspective.

CSP manually models the IPS and the environment for each type of ambiguity [17], and thus the IPS can interpret the packets according to the model. However, the manual method is certainly not scalable for the IPS to know the environment. Active mapping can actively send probing packets to infer the environment [18], but the probing packets may be too noisy, and blocked by a firewall. Passive OS fingerprinting infers the environment information by passive monitoring the packets across the IPS [19] without generating probing packets. The inference may be imprecise if the victim does not generate the right packets for the IPS to recognize it. The network administrators have to consider the advantages and disadvantages of each method, since none of them is ideal.

3) Countermeasures against payload mutation: An IPS can normalize the transformed request to defend against payload mutation. For example, when seeing a hexadecimal encoded character `%20` in an URI request, the IPS can decode it back to a space character. An IPS should be able to handle the encoding methods in various applications. Because the encoding methods supported by the victim application may vary significantly, normalizing payload mutation requires careful configurations to ensure the IPS and the victim application interpret the payload consistently. Another defense against payload mutation is host-based context scanning [23], in which the IDS must cooperate with the backend Web server. After getting a mutated request from the attacker, the Web server decodes the URL and sends the normalized URL back to the IDS for detection.

4) Countermeasures against shellcode mutation: Several methods can detect polymorphic shellcodes. Buttercup [24] uses the range of the return address to detect polymorphic shellcodes for buffer overflow vulnerabilities. Since the method manually figures out the the abnormal range of the return address for each exploit in the detection rules, it is not scalable, even though its accuracy is high — the false-positive rate is only 0.01%, and the false-negative rate is 0%.

A hybrid engine [25] uses neural network (NN) based techniques to classify the disassembled instructions. This method attempts to find out possible execution paths from the payload, and use the frequencies of selected instructions along an execution path as the features in the training (from known polymorphic shellcode engines) and classification by the neural network. Although both the false-positive and false-negative rates are quite low in this method, it may be unable to detect a zero-day shellcode if the shellcode does not follow common evasion strategies such as decryption loops, junk instructions, and so on.

Network-level polymorphic shellcode detection [26], [27] emulates the shellcode execution on a CPU emulator. This approach treats the input byte stream as the instruction codes and tries to execute it. The detection is feasible because executing random bytes in the input usually stops soon (e.g., due to encountering an illegal instruction code), while polymorphic codes can be executed until the encrypted payload is fully decrypted. The detection finds out the decryptor based on the following heuristic: if the instructions associated with deriving the location of encrypted payload are found, and there are an excessive number of memory reads within a small memory region (i.e., likely to be the decryption process), the condition is an indication of executing a polymorphic shellcode. However, if the shellcode does not rely on self-modification such as decrypting the encrypted payload for evasion, the detection will fail. Moreover, the emulation may be unable to deeply analyse complicated shellcodes on-line in the network level for the sake of performance.

The method, Spector, in [28] also symbolically emulates the shellcode execution and finds out the behavior in the shellcode, such as executing certain instructions or calling certain application programming interfaces (APIs). Despite the deep analysis in the emulation, attackers can evade the detection with some advanced techniques, such as *memory-scanning attacks* [29]. For example, the shellcode can scan the code regions of the victim process for the `ret` instruction, and call to the location. The execution will return to the original location if the shellcode execution is in the victim process. Since the emulator is unable to access the code within the victim process, the emulation will be disrupted. Yataglass proposed in [30] can detect the memory-scanning loop in the shellcode execution, and prepare a code region to trick the shellcode into believing it has found the code in the search. This strategy can prevent the shellcode from evading network-level code emulation by memory-scanning attacks.

III. Evasion Testing

This section describes the environment and the experiments in the testing, which checks whether the IPSs under the tests are resilient to the evasion techniques or not. The study uses Fragroute, Nikto, ADMmutate, Sploit and Metasploit to conduct experiments on three IPSs, and examines the effectiveness of the evasion techniques from the tools. The study also uses *libemu* to examine the effectiveness of detecting shellcode encoded by Metasploit. We assess only signature-based IPSs since they are dominant in operational environments [2]. We also notice that Juan et al. [37] have conducted evasion tests on the open-source IPSs: Snort (www.snort.org) and Bro (www.bro-ids.org). Their testing involves only evasion techniques based on packet splitting and duplicate insertion, and the IPSs under test do not include commercial ones. This work covers a wider range of evasion techniques, and tests more than open-source IPSs.

A. The Testing Environment

This study tests three IPSs: the FortiGate, Snort and ZyXEL IPSs[3]. They all operate with the up-to-date firmware/code and rules at the time of testing. The experiments focus on only the resilience of the IPSs to various evasion techniques, but not the capability of IPS signature matching to the latest attacks in the wild, which is not the purpose of this work. Therefore, it is sufficient to choose attacks that trigger specific signatures in the IPSs, apply various evasion techniques, and check whether the IPS can still detect the attacks or not.

Among the three IPSs, Snort allows users to configure the maximum number of bytes and segments in the queue for packet reassembly with the options such as `max_queued_bytes` and `max_queued_segs` in the Stream5 preprocessor. We set these options to be unlimited, so that packet reassembly will not fail due to insufficient queue space. The other two IPS products do not provide the options for configuring the anti-evasion policy, so we just leave their default configuration intact. The detailed configurations of the three IPSs are available at www.cs.ccu.edu.tw/~pclin/evasion. For attack generation, we execute the attack tools with the default configuration and use only the required options in the command to launch each attack instance.

B. The Experiments to Assess the IPSs Against Evasion

This study designs the following experiments to cover various evasion techniques as many as we can. Each experiment begins with a baseline test to ensure the IPSs can successfully detect an attack, and then checks whether they can still find that attack after the evasion techniques have been applied.

For testing with ADMmutate and Fragroute, it requires a vulnerable service on the victim and an attack program exploiting the vulnerability. This study uses Sendmail version 8.12.5 buffer overflow vulnerability (www.securityfocus.com/advisories/5054) and an exploit program (from www.securityfocus.com/bid/6991/exploit) for this vulnerability to access root privilege of the target. We select the vulnerability because buffer overflow is a typical attack, and it is easy to validate the effect of evasion by comparing the received email with and without evasion (see the later discussion).

[3]We anonymize the two commercial IPSs as IPS-A and IPS-B in the rest of the paper.

After the three-way handshake, the exploit program uses a long string to overflow the `From` header of the `DATA` command, inserts a shellcode into the message body, and immediately resets the connection. We repeat the attack with various evasion techniques, and validate the success of each evasion. Besides observing whether or not the IPS generates an alert, we also check if the attack still works on the victim in the experiments. If the attack succeeds, the victim's root account will receive an email that contains the overflowed header and message, which include the input string that the vulnerable application sees.

The email from the packets transformed by Fragroute should be the same as that from a successful attack without evasion, except for the timestamp information. Therefore, comparing both emails can validate the success of the attack after evasion. Also, if an IPS successfully detects the attack and blocks the connection, the victim will not receive any mail. Because ADMmutate applies polymorphic shellcode mutation, a message body that contains the shellcode is different from the original one, and success in evasion can only be validated from the IPS alerts.

Nikto, which tests a Web server by sending dangerous requests, is equipped with an extensive database of request URIs for generating many variations of HTTP requests. Since the experiments in this study focus on evasion techniques, the number of baseline request URIs in the database is reduced to save time. The Nikto evasion options for each technique of payload mutation (see the test items in Table IV) are then enabled to compare the alerts in the baseline test and the evasion test. We also mutate the following strings of SQL injection with Havij to test the anti-evasion capability of IPSs. The strings are selected because they can be detected by the IPSs in the baseline test. We can see whether they can be still detected with evasion.

```
1. &query=SELECT uid FROM page_fan WHERE page_id =
   80360974615 AND (uid == 1784906357 OR (uid IN (SELECT
   uid2 FROM friend WHERE uid1 = 1784906357) ) )&call_id =
   12879276368062
2. GET /bbs/viewthread.php?extra=page=1&tid=74654'
   /**/OR/**/ '1'='1
3. GET /bbs/viewthread.php?extra=page=1&tid=74654; insert
   into name
```

A previous study shows that the two exploits, "Wu-ftpd Remote String Stack Overwrite" and "Wu-imapd Remote Buffer Overflow", can evade Snort 2.1.2 using telnet control sequences and shellcode mutation [8]. This study simulates the same environment, but replaces the devices under test with the three IPSs to see whether there is any difference from the previous study. Sploit provides the two FTP evasion techniques. The first is to change the case of the command, and the second is to insert a telnet control sequence within a word. We also test how well *libemu* can handle shellcode mutation to understand the capability of IPSs if they were equipped with a code emulator.

IV. Experimental Results and Observations

This section presents the result of each above experiment, and then explains the results.

TABLE II
RESULTS OF THE FRAGROUTE EVASION TEST.

Test Items	IPS-A	IPS-B	Snort
Fragment IP packet	OK	OK	OK
Segment TCP packet	91	357	2
Overlap new IP fragment	OK	OK	OK
Overlap new TCP segment	45	187	357
Interleave bad IP payload	OK	OK	OK
Interleave bad IP option	OK	OK	OK
Interleave short-TTL IP packet	OK	OK	OK
Add IP lsrr|ssrr option	OK	OK	OK
Modify TTL value	OK	OK	OK
Modify TOS value	OK	OK	OK
Change order of packet	OK	OK	OK
Interleave bad checksum	OK	OK	OK
Interleave null flags	OK	OK	OK
Interleave zero timestamp	OK	OK	OK
Interleave fake retransmit	OK	OK	OK
Interleave fake packet seq.	OK	OK	OK
Interleave SYN during conn.	OK	OK	OK
Insert short-TTL TCP packet	OK	OK	OK
Modify TCP MSS options	OK	OK	OK
Modify TCP wscale option	OK	OK	OK

A. Results of Testing with Fragroute and ADMmutate

This study first conducts the baseline testing on the three IPSs. They all block and log the attacks without evasion. The alert messages can be all subsumed under the category of "SMTP from header overflow." This study then conducts the testing with the evasion techniques by Fragroute and ADMutate. Table II lists the Fragroute evasion techniques and the test results. "OK" denotes that the IPS can still detect the attack with evasion, while "Evaded" denotes that the IPS fails to detect the attack. The numbers in the first four rows mean the largest size (in bytes) of the split TCP segments that can successfully evade detection. The remaining rows show the testing results from miscellaneous techniques described in [5] and the man page of Fragroute.

According to Table II, the three IPS boxes can counter the techniques of duplicate insertion at the TCP/IP layer. Although it is possible to evade their detection with overlapping new TCP segments, the success in evasion is still due to packet splitting. The IPSs use different countermeasures against duplicate insertion. For example, the Snort Stream5 preprocessor tracks the state of each connection, so that it can ignore invalid packets that should be dropped. Examining the traffic going through the other IPSs, we find that they adopt the strategies of traffic normalization [5] and protocol scrubbing [22] for countering the evasion. The IPSs support rewriting the Time-To-Live field value to 64, reassembling fragmented packets, removing options, and dropping invalid packets. Regardless of these measures, splitting TCP segments can still evade the detection. Adjusting the segment sizes reveals the largest TCP segment sizes for evading the detection of each IPS (see Table II).

To investigate why packet splitting is effective in IPS evasion, this study examines the Snort source code, since it is the only open-source system among the three IPSs. Before scanning the packet content for intrusion signatures with its detection engine, Snort uses the Stream5 preprocessor to inspect the state of TCP sessions and reassemble TCP streams. In the following experiments, a buffer overflow attack triggers the default SMTP rule with the alert message: "SMTP from comment overflow attempt." This rule tries to recognize four specific keywords: (1) `From|3A|`, (2) a string of 22 contiguous pairs of "<>", (3) `|28|` and (4) `|29|`, where `|hh|` denotes a character whose ASCII code is `hh` in hexadecimal[4]. We added some C `printf` functions to the function `CheckANDPatternMatch` in `src/detection-plugins/sp_pattern_match.c` and rebuilt Snort to print out the pattern buffer, packet data and an indication of whether the pattern is found or not.

Figure 2(a) and Figure 2(b) list the results of packet splitting into 3-byte and 2-byte segments after running the experiments again. The figures demonstrate why 3-byte segments fail to evade the detection, but 2-byte segments succeed. Figure 2(a) shows that the stream of 3-byte segments is reassembled into 484-byte data, which matches all the keywords we have mentioned. However, Figure 2(b) shows that the stream of 2-byte segments is reassembled into 269-byte data, which is too short to match the third and the fourth keywords.

The results indicate that splitting TCP segments into small ones can be still effective for attackers to evade IPS detection because the packet reassembly may be incomplete and thus the detection fails to match the signatures in the detection rule. However, if an IPS is configured to detect a burst of such small segments, the evasion technique can still be detected with a different alert, such as "Too Many Small Segments Found."

After we obfuscated the attack program by encrypting the code with the ADMmutate API, the shellcode appearance was completely different from the original. As a result, IPS-B and Snort were successfully evaded. That is to say, polymorphic shellcode mutation works on some IPSs. This work also studied Snort for the reason and found that Snort was evaded because the Snort only inspects the buffer overflow in the mail header, but does not check the shellcode in the message body.

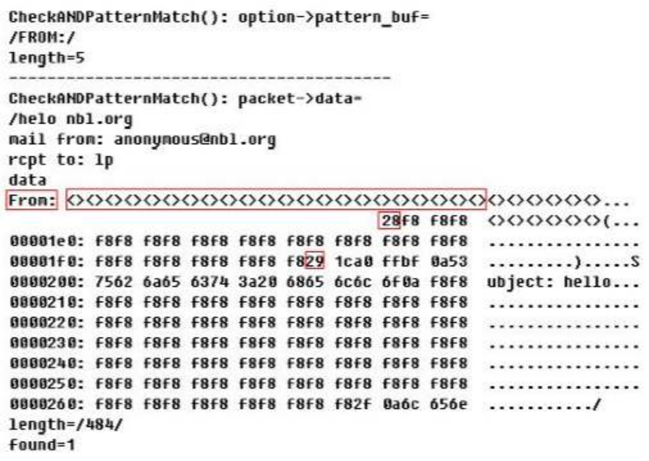
B. Results of Testing with Nikto and Havij

Table III lists the detailed alerts triggered by Nikto without applying any evasion techniques. In this testing, Snort is equipped with two sets of rules: one is the default Snort rules for Web attacks, and the other is the rule set for network scanning, downloaded from Bleeding Edge Threats[5] (www.bleedingthreats.net), which provides a large number of various Snort rules for users to download.

Table IV lists the experimental results of applying nine Nikto evasion techniques of payload mutation, where "OK" denotes that the IPS can correctly handle the evasion and generate the same alerts. The table shows that payload mutation with Nikto is still effective to Snort, but not to the other two commercial IPSs. Compared with the case without evasion, most of the tests on Snort have fewer alerts after payload mutation, but a few exceptions exist. Examples of the exceptions are from the techniques of "Premature URL ending" and "Prepend long random string to request", in which the payloads after mutation happen to trigger some additional

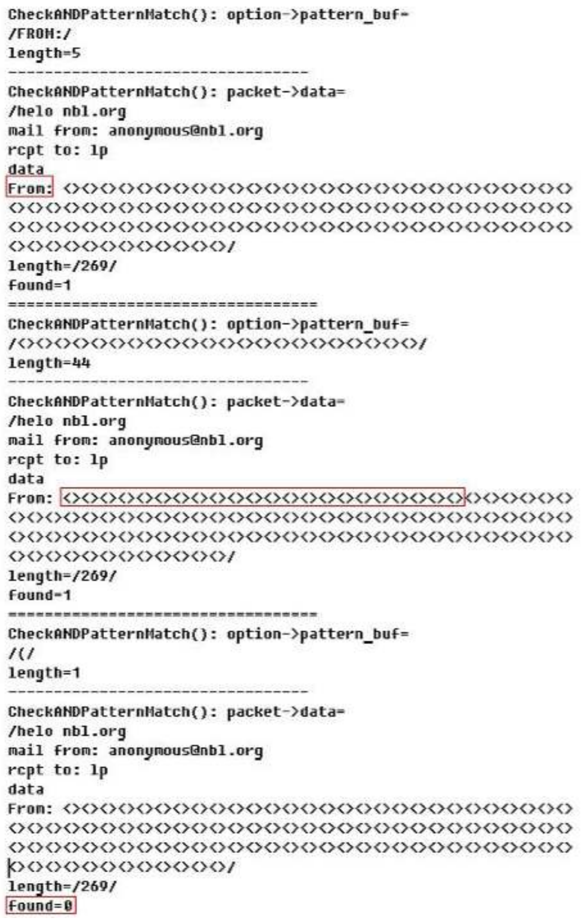
[4]The keywords `|3A|`, `|28|`, and `|29|` therefore denote the characters of colon, left parenthesis, right parenthesis, respectively.

[5]The Web site has become www.emergingthreats.net at the time of paper revision.



```
CheckANDPatternMatch(): option->pattern_buf=
/FROM:/
length=5
------------------------------------------
CheckANDPatternMatch(): packet->data=
/helo nbl.org
mail from: anonymous@nbl.org
rcpt to: lp
data
From: <><><><><><><><><><><><><><><><><><><><><><><><><><><><>...
                                       28f8 f8f8  <><><><><><>(...
00001e0: f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8  ................
00001f0: f8f8 f8f8 f8f8 f8f8 f829 1ca0 ffbf 0a53  .........).....S
0000200: 7562 6a65 6374 3a20 6865 6c6c 6f0a f8f8  ubject: hello...
0000210: f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8  ................
0000220: f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8  ................
0000230: f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8  ................
0000240: f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8  ................
0000250: f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8 f8f8  ................
0000260: f8f8 f8f8 f8f8 f8f8 f8f8 f82f 0a6c 656e  .........../
length=/484/
found=1
```

(a) Failure to evade Snort with 3-byte segments.



```
CheckANDPatternMatch(): option->pattern_buf=
/FROM:/
length=5
------------------------------------
CheckANDPatternMatch(): packet->data=
/helo nbl.org
mail from: anonymous@nbl.org
rcpt to: lp
data
From: <><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><>/
length=/269/
found=1
====================================
CheckANDPatternMatch(): option->pattern_buf=
/<><><><><><><><><><><><><><><><><><><><><><>/
length=44
------------------------------------
CheckANDPatternMatch(): packet->data=
/helo nbl.org
mail from: anonymous@nbl.org
rcpt to: lp
data
From: <><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><>/
length=/269/
found=1
====================================
CheckANDPatternMatch(): option->pattern_buf=
/(/
length=1
------------------------------------
CheckANDPatternMatch(): packet->data=
/helo nbl.org
mail from: anonymous@nbl.org
rcpt to: lp
data
From: <><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><>/
length=/269/
found=0
```

(b) Success in evading Snort with 2-byte segments.

Fig. 2. Splitting TCP segments to evade Snort.

alerts on Snort, such as "invalid HTTP version string" and "http directory traversal." The additional alerts are more than the eliminated alerts due to payload mutation, so the total number of alerts increases because of the exceptions.

We also test the anti-evasion capability of the IPSs for SQL injection attacks generated by Havij. Table V presents the results that the SQL injection attacks can evade the detection by white-space manipulation, various encodings of the characters in the SQL injection, and so on. The three IPSs all perform poorly when detecting SQL injection attacks with evasion. The results are unsurprising. Snort does not have the preprocessor to normalize transformed strings of SQL injection, except white space manipulation, which can be handled by matching the payloads with regular-expression patterns such as `/insert\s+into\s+[^\/\\]+/Ui` (i.e., allowing one or more space characters between the words). We believe the other two IPSs have similar limitations, even though they are black boxes to us. In summary, equipping an IPS with a capable preprocessor to normalize the mutation of SQL injection attacks, not just regular-expression patterns, is still strongly desired for countering the evasion.

TABLE III
ALERTS TRIGGERED BY NIKTO WITHOUT EVASION TECHNIQUES.

System	Alerts
IPS-A	1. CGI Website cgi-bin Access 2. HTTP Password File Download (2 alerts)
IPS-B	1. Web-cgi /cgi-bin/ (2 alerts) 2. Web-IIS (1 alert) 3. IIS-back-evasion-attack (1 alert) 4. Web-misc (4 alerts)
Snort	1. From Web-attack rules (47 alerts) 2. Nikto progress (53 alerts) 3. attempt to execute java code (1 alert)

TABLE IV
RESULTS OF THE NIKTO EVASION TEST.

Test items	IPS-A	IPS-B	Snort
Random URI encoding (non-UTF8)	OK	OK	1. 6 alerts 2. 0 alert 3. 0 alerts
Add directory self-reference /./	OK	OK	1. 38 alerts 2. 0 alert 3. 1 alert
Premature URL ending	OK	OK	1. 161 alerts 2. 0 alert 3. 1 alert
Prepend long random string to request	OK	OK	1. 161 alerts 2. 36 alerts 3. 1 alert
Fake parameters to files	OK	OK	1. 103 alerts 2. 0 alert 3. 1 alert
TAB as request spacer instead of spaces	OK	OK	1. OK 2. 0 alert 3. 1 alert
Random case sensitivity	OK	OK	1. 8 alerts 2. 0 alert 3. 0 alert
Use Windows directory separator instead of /	OK	OK	1. 41 alerts 2. 0 alert 3. 1 alert
Session splicing	OK	OK	1. 3 alerts 2. 0 alert 3. 0 alert

C. Results of Testing with Sploit

Table VI lists the detailed alerts triggered by the exploit "Wu-ftpd Remote String Stack Overwrite" in Sploit without applying evasion techniques, and Table VII lists the experimental results after applying the Sploit evasion techniques, where "OK" denotes that the alerts are the same with and without evasion, and "Evaded" denotes that the IPS does not generate any alerts. The table demonstrates that the evasion techniques can avoid generating some of the alerts.

TABLE V
RESULTS OF THE HAVIJ EVASION TEST.

Test items	IPS-A	IPS-B	Snort
White space manipulation	OK	OK	OK
C-style comment	Evaded	Evaded	Evaded
Encoding: HEX	Evaded	Evaded	Evaded
Encoding: BASE64	Evaded	Evaded	Evaded
Encoding: DECIMAL	Evaded	Evaded	Evaded

TABLE VI
ALERTS TRIGGERED BY SPLOIT WITHOUT EVASION TECHNIQUES.

System	Alerts
IPS-A	1. FTP PASS Command Overflow
	2. WuFTP SITE EXEC Attempt
	3. FTP Text Line Too Long
IPS-B	1. FTP SITE EXEC format string attempt
	2. FTP command overflow attempt
Snort	1. FTP PASS overflow attempt
	2. FTP SITE EXEC format string attempt
	3. FTP SITE overflow attempt
	4. FTP SITE EXEC attempt
	5. SHELLCODE x86 NOOP
	6. FTP command parameters were too long

TABLE VII
RESULTS OF THE SPLOIT EVASION TEST.

Test items	IPS-A	IPS-B	Snort
FTP Command Change Case	1. OK	1. OK	1. OK
	2. OK	2. OK	2. OK
	3. OK		3. OK
			4. OK
			5. OK
			6. OK
Insert telnet control sequence	1. Evaded	1. Evaded	1. OK
	2. Evaded	2. OK	2. OK
	3. OK		3. OK
			4. OK
			5. OK
			6. OK
Shellcode mutation (nop encoder)	1. OK	1. OK	1. OK
	2. OK	2. OK	2. OK
	3. OK		3. OK
			4. OK
			5. Evaded
			6. OK
2-byte TCP segment splitting	1. OK	1. Evaded	1. Evaded
	2. OK	2. Evaded	2. OK
	3. OK		3. Evaded
			4. OK
			5. OK
			6. OK

TABLE VIII
RESULTS OF THE SHELLCODE MUTATION TEST.

Encoder	EC	BS	RS	ED
None (baseline)	N/A	OK	OK	OK
Pex	OK	OK	OK	OK
Countdown	N/A	N/A	N/A	OK
PexFnstenvSub	Evaded	OK	OK	N/A
PexAlphaNum	OK	OK	OK	OK
PexFnstenvMov	OK	OK	OK	OK
JmpCallAdditive	OK	OK	OK	OK
ShikataGaNai	OK	Evaded	Evaded	Evaded
Alpha2	OK	OK	OK	OK

Figure 3 lists the three Snort signatures successfully evaded by Sploit, as well as their corresponding alert messages and the evasion techniques that make it. The reason that 2-byte TCP segment splitting can evade the "FTP PASS overflow attempt" and "FTP SITE overflow attempt" signatures is that Snort reassembles the split segments into 82 bytes before passing them to the detection engine, which matches the string `PASS`. However, since the length of the remaining data is shorter than 100 bytes, the rule option `isdataat:100`, which specifies there should be at least 100 bytes of data after the end of string `PASS`, is certainly not matched. Therefore, the alert will not be generated. This case again demonstrates that evasion with packet splitting can succeed due to incorrect reassembly in the IPS implementation.

Inserting telnet control sequences cannot evade Snort thanks to the new Stream5 protocol preprocessor for TCP reassembly. However, if Stream5 is replaced with the old Stream4, the detection engine even fails to match the `PASS` and `SITE` keywords, and Snort will be successfully evaded. Changing the case of the FTP command does not help to evade Snort because the Snort signatures are case-insensitive due to the `nocase` rule option.

For the third signature in Figure 3, `NOP` (no operation, 0x90 in hexadecimal) in the shellcode disappeared after applying shellcode mutation. As the result, Snort failed to match the signature in the rule for "SHELLCODE x86 NOOP" because it does nothing about countering the shellcode mutation.

D. Libemu experiment: Detecting Metasploit encoded shellcodes

Since signatures based on shellcode patterns can be evaded by shellcode mutation, we add an additional test with *libemu* to see how well an IPS could detect shellcode mutation if it were equipped with a code emulator like *libemu*, which tries to detect encoded shellcode by emulating the code execution.

This study detects shellcodes generated from Metasploit using the *libemu* library. Table VIII lists the four types of shellcodes (Execute Command (EC), Bind shell (BS), Reverse Shell (RS) and Executable Download and Execute (ED)) and eight different encoders [11] in the tests to determine if *libemu* can detect an encoded shellcode. The results show that most of encoded shellcodes can be detected, and imply that emulation is effective to detect most shellcode mutations.

V. CONCLUSION

This article provides a survey and tutorial of evasion techniques, tools, and countermeasures. Besides the survey and tutorial parts, we also design several experiments to assess whether the evasion techniques are still effective to modern IPSs or not, and explain why some evasion techniques can work by studying the inner working of the open-source Snort IPS. The experimental results have the following major observations.

1) The success of evasion techniques depends on the ability of an IPS to restore the original semantics of the attacks or design the signatures to cover the variants of the attacks. For example, Snort is unable to handle various encoded strings of SQL injection because it lacks a pre-processor to normalize the encoded strings. Signatures in regular expressions can express the variants from simple mutation such as manipulating space characters, but they

```
1) Alert message: FTP PASS overflow attempt
   Signature: content:"PASS"; nocase; isdataat:100,relative;
         pcre:"/^PASS(?!\n)\s[^\n]{100}/smi";
   Evasion technique: 2-byte TCP segment splitting
2) Alert message: FTP SITE overflow attempt
   Signature: content:"SITE"; nocase; isdataat:100,relative;
         pcre:"/^SITE(?!\n)\s[^\n]{100}/smi
   Evasion technique: 2-byte TCP segment splitting
3) Alert message: SHELLCODE x86 NOOP
   Signature: content:"|90 90 90 90 90 90 90 90 90 90 90 90 90 90|"; depth:128
   Evasion technique: Shellcode mutation
```

Fig. 3. Snort signatures evaded by Sploit.

are still ineffective against various encoding techniques. Equipping an IPS with the preprocessors to restore the original semantics of the attack is therefore essential, but the implementation should be accurate and sufficient to cover each possibility of evasion, or the detection can still be somehow evaded.

2) Packet splitting in IP fragmentation and TCP segmentation can still help attacks to evade the three modern IPSs in this study because of the problematic implementation of packet reassembly. In contrast, duplicate insertion is less effective than other evasion techniques on the IPSs. Therefore, the implementation of packet reassembly should be carefully tested in terms of its robustness to various fragment and segment sizes during the IPS development.
3) Payload mutation and shellcode mutation can still help attackers successfully suppress some alerts due to the drawbacks of signature-based IPSs. Even though normalizing the payload content (e.g., with a preprocessor) and code emulation (e.g., with *libemu*) can relieve the problems, the implementations of the countermeasures on the three modern IPSs are still unable to correctly handle all of the test cases in this study.

Based on the above observations, we think the robustness of the IPS implementation to various evasion techniques still deserves further research. For example, an issue can be evaluating the coverage of test cases to assess the robustness to various evasion techniques. Moreover, since the execution of normalization and code emulation can be time-consuming, the trade-offs between performance and capability to anti-evasion have to be carefully evaluated in the future.

References

[1] P. C. Lin, Z. X. Li, Y. D. Lin, Y. C. Lai and F.C. Lin, "Profiling and accelerating string matching algorithms in three network content security applications," IEEE Commun. Surveys Tutorials, vol. 8, issue 2, pp. 24-37, Second Quarter 2006.

[2] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning For Network Intrusion Detection," In Proc. IEEE Security and Privacy, May 2010.

[3] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras and B. Stiller, "An Overview of IP Flow-Based Intrusion Detection," IEEE Commun. Surveys Tutorials, vol. 12, issue 3, pp. 343-356, Third Quarter 2010.

[4] T. H. Ptacek and T. N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection," Technical Report from Secure Networks, Inc., http://insecure.org/stf/secnet_ids/secnet_ids.html, Jan. 1998.

[5] M. Handley, V. Paxson, and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-end Protocol Semantics," In Proc. USENIX Security Symposium, Aug. 2001.

[6] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," Computer Networks, vol. 34, issue 4, pp. 579-595, Oct. 2000.

[7] H. Debar and B. Morin, "Evaluation of the Diagnostic Capabilities of Commercial Intrusion Detection Systems," In Proceedings Recent Advances in Intrusion Detection (RAID), Oct. 2002.

[8] G. Vigna, W. Robertson, and D. Balzarotti, "Testing Network-based Intrusion Detection Signatures Using Mutant Exploits," In Proc. 11th ACM Conference on Computer and Communications Security (CCS), Oct.2004.

[9] D. J. Chaboya, R. A. Raines, R. O. Baldwin and B. E. Mullins, "Network Intrusion Detection: Automated and Manual Methods Prone to Attack and Evasion," IEEE Security & Privacy Magazine, vol. 4, no. 6, pp. 36-43, Nov./Dec. 2006.

[10] S. Zanero, "Flaws and frauds in the evaluation of IPS technologies," In Forum of Incident Response and Security Teams, June 2007.

[11] Metasploit encoding, http://www.derkeiler.com/pdf/Mailing-Lists/securityfocus/pen-test/2006-03/msg00253.pdf.

[12] R. Bidou, "IPS shortcomings," In Black Hat Briefings, July 2006.

[13] D. Roelker, "HTTP IDS evasions revisited," Technical report, Sourcefire, Sept. 2004.

[14] T. Detristan, T. Ulenspiegel, Y. Malcom and M. Underduk, "Polymorphic Shellcode Engine using Spectrum Analysis," Phrack, 11(61), Aug. 2003.

[15] O. Kolesnikov and W. Lee, "Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic," In Proc. USENIX Security Symposium, Aug. 2006.

[16] P. Bania, "Evading network-level emulation," Available at piotrbania.com/all/articles/pbania-evading-nemu2009.pdf, June 2009.

[17] G. Rohrmair and G. Love, "Using CSP to Detect Insertion and Evasion Possibilities within the Intrusion Detection Area," In Proc. BCS Workshop on Formal Aspects of Security, Dec. 2002.

[18] U. Shankar and V. Paxson. "Active Mapping: Resisting NIDS Evasion without Alerting Traffic," In Proc. IEEE Symposium on Security and Privacy, May 2003.

[19] G. Taleck, "Ambiguity Resolution via Passive OS Fingerprinting," in Proc. International Conference on Recent Advances in Intrusion Detection (RAID), Sept. 2003.

[20] S. P. Chung, A. K. Mok, "Swarm Attacks against Network-Level Emulation/Analysis," In Proc. 11th international symposium on Recent Advances in Intrusion Detection (RAID), Sept 2008.

[21] P. Fogla and W. Lee, "Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques," In Proc. ACM Conference on Computer and Communications Security (CCS), Oct.–Nov. 2006.

[22] D. Watson, M. Smart, G. R. Malan and F. Jahanian, "Protocol Scrubbing: Network Security Through Transparent Flow Modification," IEEE/ACM Trans. Netw., vol. 12, issue 2, pp. 261-273, Apr. 2004.

[23] H. Dreger, C. Kreibich, V. Paxson and R. Sommer, "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), July 2005.

[24] A. Pasupulati, J. Coit, K. Levitt and F. Wu, "Buttercup: On Network-based Detection of Polymorphic Buffer Overflow Vulnerabilities," In Proc. IEEE/IFIP Network Operation and Management Symposium, May 2004.

[25] U. Payer, P. Teufl and M. Lamberger, “Hybrid Engine for Polymorphic Shellcode Detection,” In Proc. Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), July 2005.
[26] M. Polychronakis, K. G. Anagnostakis and E. P. Markatos, “Network-level Polymorphic Shellcode Detection using Emulation,” In Proc. Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), July 2006.
[27] M. Polychronakis, K. Anagnostakis and E. P. Markatos, “Emulation-Based Detection of Non-self-contained Polymorphic Shellcode.” In Proc. 10th International Symposium on Recent Advances in Intrusion Detection (RAID), Aug. 2007.
[28] K. Borders, A. Prakash and M. Zielinski, “Spector: Automatically Analyzing Shellcode,” In Proc. Annual Computer Security Applications Conference (ACSAC), Dec. 2007.
[29] C. M. Linn, M. Rajagopalan, S. Baker, C. Collberg, S. K. Debray, J. Hartman, “Protecting against Unexpected System Calls,” In Proc. 13th Usenix Security Symposium, Aug. 2005.
[30] M. Shimamura and K. Kono, “Yataglass: Network-Level Code Emulation for Analyzing Memory-Scanning Attacks,” In Proc. Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), July 2009.
[31] G. Varghese, J. A. Fingerhut and F. Bonomi, “Detecting Evasion Attacks at High Speeds without Reassembly,” In Proc. ACM SIGCOMM, Sept. 2006.
[32] G. Antichi, D. Ficara, S. Giordano, G. Procissi and F. Vitucci, “Counting Bloom Filters for Pattern Matching and Anti-Evasion at the Wire Speed,” IEEE Network, vol. 23, issue 1, Jan/Feb 2009.
[33] V. M. Igure and R. D. Williams, “Taxonomies of Attacks and Vulnerabilities in Computer Systems,” IEEE Commun. Surveys Tutorials, vol. 10, issue 1, pp. 6-19, First Quarter 2008.
[34] R. Smith, C. Estan and S. Jha, “Backtracking Algorithmic Complexity Attacks against a NIDS,” In Proc. 22nd Annual Computer Security Applications Conference (ACSAC), Dec. 2006.
[35] S. Dharmapurikar and V. Paxson, “Robust TCP Stream Reassembly In the Presence of Adversaries,” In Proc. USENIX Security Symposium, Aug. 2005.
[36] M. Sachs and D. Ahmad, “Always the Same, Never the Same,” IEEE Security & Privacy, vol. 8, issue 2, pp. 73-75, Mar./Apr. 2010.
[37] L. Juan, C. Kreibich, C. H. Lin and V. Paxson, “A Tool for Offline and Live Testing of Evasion Resilience in Network Intrusion Detection Systems,” Proc. Fifth GI International Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), July 2008.
[38] S. Rubin, S. Jha and B. P. MIller, “Automatic generation and analysis of NIDS attacks,” In Annual Computer Security Conference (ACSAC), Dec. 2004.
[39] M. Vutukuru, H. Balakrishnan and V. Paxson, “Efficient and Robust TCP Stream Normalization,” In Proc. IEEE Symposium on Security and Privacy, May 2008.

Tsung-Huan Cheng received his B.S. degree and M.S. degree in Computer Science from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 2007 and 2009, respectively. His research interests include network security and network network forensics. He is currently a software engineer of MediaTek Company since 2010.

Ying-Dar Lin is Professor of Computer Science at National Chiao Tung University (NCTU) in Taiwan. He received his Ph.D. in Computer Science from UCLA in 1993. He served as the CEO of Telecom Technology Center during 2010–2011 and a visiting scholar at Cisco Systems in San Jose during 2007–2008. Since 2002, he has been the founder and director of Network Benchmarking Lab (NBL, www.nbl.org.tw), which reviews network products with real traffic. He also cofounded L7 Networks Inc. in 2002, which was later acquired by D-Link Corp. He recently, in May 2011, founded Embedded Benchmarking Lab (www.ebl.org.tw) to extend into the review of handheld devices. His research interests include design, analysis, implementation, and benchmarking of network protocols and algorithms, quality of services, network security, deep packet inspection, P2P networking, and embedded hardware/software co-design. His work on “multi-hop cellular” has been cited over 500 times. He is currently on the editorial boards of IEEE Transactions on Computers, IEEE Network, IEEE Communications Magazine Network Testing Series, IEEE Communications Surveys and Tutorials, IEEE Communications Letters, Computer Communications, and Computer Networks. He recently published a textbook “Computer Networks: An Open Source Approach” (www.mhhe.com/lin), with Ren-Hung Hwang and Fred Baker (McGraw-Hill, 2011). It is the first text that interleaves open source implementation examples with protocol design descriptions to bridge the gap between design and implementation.

Yuan-Cheng Lai received the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 1997. In August 2001, he joined the faculty of the Department of Information Management at National Taiwan University of Science and Technology, Taipei, Taiwan, where he has been a professor since February 2008. His research interests include wireless networks, network performance evaluation, network security, and content networking.

Po-Ching Lin received the B.S. degree in computer and information education from National Taiwan Normal University, Taipei, Taiwan, in 1995, and the M.S. and Ph.D. degrees in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2001 and 2008, respectively. He joined the faculty of the Department of Computer and Information Science, National Chung Cheng University (CCU), Chiayi, Taiwan, in August 2009. He is currently an Assistant Professor. His research interests include network security, network traffic analysis, and performance evaluation of network systems. He is also a member of the IEEE.