Evaluating Computer Intrusion Detection Systems: A Survey of Common Practices

ALEKSANDAR MILENKOSKI, University of Würzburg
MARCO VIEIRA, University of Coimbra
SAMUEL KOUNEV, University of Würzburg
ALBERTO AVRITZER, Siemens Corporation, Corporate Technology
BRYAN D. PAYNE, Netflix, Inc.

The evaluation of computer intrusion detection systems (which we refer to as intrusion detection systems) is an active research area. In this article, we survey and systematize common practices in the area of evaluation of such systems. For this purpose, we define a design space structured into three parts: workload, metrics, and measurement methodology. We then provide an overview of the common practices in evaluation of intrusion detection systems by surveying evaluation approaches and methods related to each part of the design space. Finally, we discuss open issues and challenges focusing on evaluation methodologies for novel intrusion detection systems.

CCS Concepts: • **General and reference** → **Surveys and overviews**; **Evaluation**; **Experimentation**; • **Security and privacy** → **Intrusion detection systems**; Virtualization and security; Penetration testing

Additional Key Words and Phrases: Computer intrusion detection systems, workload generation, metrics, measurement methodology

ACM Reference Format:
Aleksandar Milenkoski, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Bryan D. Payne. 2015. Evaluating computer intrusion detection systems: A survey of common practices. ACM Comput. Surv. 48, 1, Article 12 (September 2015), 41 pages.
DOI: http://dx.doi.org/10.1145/2808691

1. INTRODUCTION

Intrusion detection is a common cyber security mechanism whose task is to detect malicious activities in host and/or network environments. The detection of malicious activities enables timely reaction to, for example, stop an ongoing attack. Given the importance of intrusion detection, the research and industrial communities have designed and developed a variety of intrusion detection systems (IDSes).

With the increasing variety and complexity of IDSes, the development of IDS evaluation methodologies, techniques, and tools has become a key research topic. The benefits

This research has been supported by the Research Group of the Standard Performance Evaluation Corporation (SPEC; http://www.spec.org, http://research.spec.org).
Authors' addresses: A. Milenkoski and S. Kounev, University of Würzburg, Am Hubland, 97074 Würzburg, Germany; email: {milenkoski, skounev}@acm.org; M. Vieira, University of Coimbra, Polo II—Pinhal de Marrocos, 3030-290 Coimbra, Portugal; email: mvieira@dei.uc.pt; A. Avritzer, Siemens Corporation, Corporate Technology, 755 College Road East, Princeton, NJ 08540; email: alberto.avritzer@siemens.com; B. D. Payne, Netflix Inc., Los Gatos, CA 95032; email: bdpayne@acm.org.

DOI: http://dx.doi.org/10.1145/2808691

of IDS evaluation are manyfold. For instance, one may compare multiple IDSes in terms of their attack detection accuracy to deploy an IDS that operates optimally in a given environment, thus reducing the risks of a security breach. Further, one may tune an already deployed IDS by varying its configuration parameters and investigating their influence through evaluation tests.

The research and industrial communities have performed a broad set of activities related to IDS evaluation, such as generation of attack traces (see Lippmann et al. [2000]), construction of IDS evaluation environments,[1] and analysis and development of IDS evaluation methodologies (see Ranum [2001] and McHugh [2000]). Given the significant amount of existing practical and theoretical work related to IDS evaluation, a structured classification is needed to improve the general understanding of the topic and to provide an overview of the current state of the field. Such an overview would be beneficial for identifying and contrasting advantages and disadvantages of different IDS evaluation methods and practices. It would also help in identifying requirements and best practices for evaluating both current and future IDSes.

In this article, we survey the common practices in IDS evaluation by analyzing existing work related to each of the standard evaluation components—workloads, metrics, and methodology. The presented survey includes 126 references out of which 66 are peer-reviewed research publications and technical reports, and 60 are links to specific tool information sites, relevant data, and similar. The benefits of the presented survey are manyfold. For researchers, the survey provides a comprehensive review of the existing work related to IDS evaluation identifying open research questions and promising directions for future research. IDS evaluation practitioners may use it as a basis for identifying useful approaches in the planning of IDS evaluation studies.

The article is organized as follows. In Section 2, we survey the common practices in IDS evaluation. In Section 3, we discuss open issues and challenges, present guidelines for planning IDS evaluation studies based on the lessons learned, and provide a summary. In Appendix A, we provide an overview of major developments in the area of IDS evaluation in a chronological manner.

1.1. Background: Security Mechanisms and Intrusion Detection Systems

A given system is considered secure if it has the properties of confidentiality, integrity, and availability of its data and services (see Stallings [2002]). Attacks are deliberate attempts to violate the previously mentioned security properties.

Kruegel et al. [2005] take an attack-centric approach to classify security mechanisms used to enforce the previously mentioned properties. They distinguish between attack prevention, attack avoidance, and attack detection mechanisms; the first class includes mechanisms that prevent attackers from reaching, and interacting with, the target, such as access control; the second class includes mechanisms that modify the data stored in the target such that it would be of no use to an attacker, such as data encryption; and the third class includes mechanisms that detect ongoing attacks under the assumption that an attacker can reach the target and interact with it—a security mechanism that belongs to this class is *intrusion detection*.

According to Scarfone and Mell [2007], intrusion detection is "the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices." Given the preceding definition, an IDS can be seen as the software that automates the intrusion detection process.

In Table I, we categorize IDSes into types according to the properties we consider relevant for evaluating different systems: the platform they monitor, the employed

[1]http://www.nsslabs.com/.

Table I. Categorization of Intrusion Detection Systems

Property	IDS Type	Description
Monitored platform	Host based	An IDS that monitors the activities on the system (i.e., the host) where it is deployed to detect *local attacks*—attacks executed by users of the targeted system itself (e.g., Open System SECurity (OSSEC)).[2]
	Network based	An IDS that monitors network traffic to detect *remote attacks*—attacks carried out over a network connection (e.g., Snort [Roesch 1999]).
	Hybrid	An IDS that is a combination of host and network-based IDSes (see Jin et al. [2009]).
Attack detection method	Misuse based	An IDS that evaluates system and/or network activities against a set of signatures of known attacks (e.g., Snort [Roesch 1999]); therefore, it is not able to detect *zero-day attacks*—attacks that exploit vulnerabilities that have not been publicly disclosed before the execution of the attacks.
	Anomaly based	An IDS that uses a baseline profile of regular network and/or system activities as a reference to distinguish between regular and anomalous activities, the latter being treated as attacks (see Avritzer et al. [2010]); therefore, it is able to detect zero day as well as known attacks, yet it may often mislabel regular activities as anomalous, which is its major disadvantage. A typical anomaly-based IDS is trained by monitoring regular activities to construct baseline activity profiles.
	Hybrid	An IDS that uses both misuse-based and anomaly based attack detection methods (see Modi and Patel [2013]).
Deployment architecture	Nondistributed	A noncompound IDS that can be deployed only at a single location (e.g., Snort [Roesch 1999]).
	Distributed	A compound IDS that consists of multiple intrusion detection subsystems that can be deployed at different locations and communicate to exchange intrusion detection-relevant data, such as attack alerts (e.g., OSSEC, which can be configured to operate in a distributed manner). Distributed IDSes can detect coordinated attacks targeting multiple sites in a given time order.

attack detection method, and their deployment architecture. We refer the reader to Debar et al. [1999] for an in-depth categorization of IDSes.

2. INTRUSION DETECTION SYSTEM EVALUATION DESIGN SPACE

In this section, we present an IDS evaluation design space structured into three parts: workloads, metrics, and measurement methodology. Although they have a lot in common when it comes to evaluation, different types of IDSes (see Section 1.1) pose challenges and requirements that apply specifically to each IDS type. When a given category or part of the design space relates closely to a particular IDS type, we stress such a relation in our discussions.

2.1. Workloads

In Figure 1, we depict the workload part of the IDS evaluation design space. To evaluate an IDS, we need both malicious and benign workloads. These can be used separately (e.g., as pure malicious or pure benign workloads) to measure the capacity of an IDS as in Bharadwaja et al. [2011] and Jin et al. [2011], or its attack coverage as in Reeves et al. [2012]. *Pure benign* workloads are workloads that do not contain attacks, whereas *pure malicious* workloads are workloads that contain only attacks. Alternatively, one can use *mixed* workloads (i.e., workloads that are a mixture of pure benign and pure malicious workloads) to subject an IDS under test to realistic attack scenarios as in Yu and Dasgupta [2011], Avritzer et al. [2010], and Sinha et al. [2006].

[2]http://www.ossec.net/.

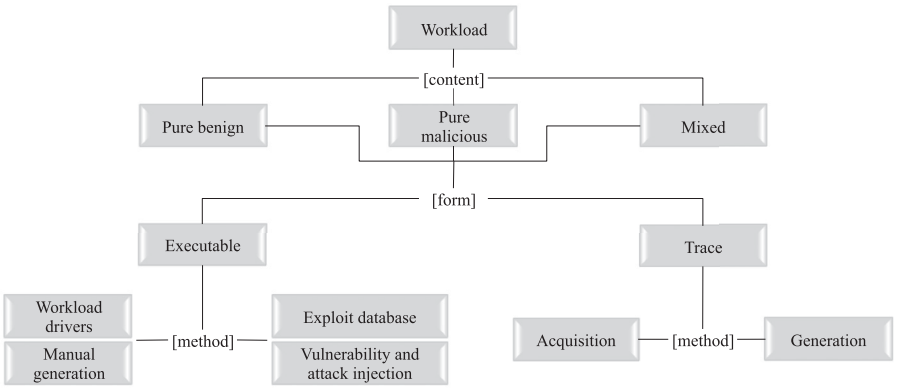


Fig. 1. IDS evaluation design space: workloads. There are three types of workloads with respect to workload content: pure benign (workloads that do not contain attacks), pure malicious (workloads that contain only attacks), and mixed. There are two types of pure benign, pure malicious, or mixed workloads with respect to their form: executable and trace. There are two methods for generating pure benign executable workloads: the use of workload drivers (Section 2.1.1) and manual generation (Section 2.1.2). There are two methods for generating pure malicious executable workloads: the use of an exploit database (Section 2.1.3), and vulnerability and attack injection (Section 2.1.4). There are two methods for generating pure benign, pure malicious, or mixed workloads in trace form: acquisition (Section 2.1.5) and generation (Section 2.1.6).

IDS evaluation workloads normally take an *executable* form for live testing of an IDS, or a *trace* form, generated by recording a live execution of workloads for later replay. The trace replay is performed with tools designed to process trace files—a common combination is the use of the tool tcpdump for capturing network traces for subsequent replay by tcpreplay.[3,4]

A major advantage of using workloads in executable form is that they closely resemble a real workload as monitored by an IDS during operation. However, a malicious workload in executable form requires a specific victim environment that can be expensive and time consuming to set up (see Debar et al. [1998]). In contrast, such an environment is not always required for replaying workload traces. Further, multiple evaluation runs are typically required to ensure statistical significance of the observed system behavior. However, replicating evaluation experiments when using executable malicious workloads is usually challenging, as the execution of attacks might crash the victim environment or render it in an unstable state. Moreover, the process of restoring the environment to an identical state as before the execution of the attacks may be time consuming.

In the following, we first discuss different methods for the generation of benign and malicious workloads in executable form (see Figure 1). We discuss the use of workload drivers and manual generation approaches for generating pure benign workloads. We also discuss the use of an exploit database and vulnerability and attack injection techniques for generating pure malicious workloads. We note that mixed workloads in executable form can be generated by using the previously mentioned methods for generating pure benign and pure malicious workloads in combination. Finally, we discuss methods for obtaining pure benign, pure malicious, or mixed workloads in a trace form, distinguishing between trace acquisition and trace generation.

2.1.1. Workloads → Pure Benign → Executable Form → Workload Drivers. For the purpose of live IDS testing, a common practice is to use benign workload drivers to generate

[3]http://www.tcpdump.org/.
[4]http://tcpreplay.synfin.net/.

artificial pure benign workloads with different characteristics. Some of the commonly used workload drivers are SPEC CPU2000 for CPU-intensive workloads[5]; iozone and Postmark for file I/O (input/output)-intensive workloads[6,7]; httpbench, dkftpbench, and ApacheBench for network-intensive workloads[8,9,10]; and UnixBench for system-wide workloads that exercise not only the hardware but also the operating system.[11] Experiments using the mentioned tools were performed in Griffin et al. [2003], Patil et al. [2004], Chung and Mok [2006], Riley et al. [2008], Zhang et al. [2008], Jin et al. [2009, 2011], Lombardi and Di Pietro [2011], and Reeves et al. [2012]. As expected, the CPU- and file I/O-intensive drivers have been employed mainly for evaluating host-based IDSes, whereas the network-intensive drivers have been employed for evaluating network-based IDSes. We look at the IDS properties typically quantified using these drivers when we discuss IDS evaluation methodologies in Section 2.3.

A major advantage of using benign workload drivers is the ability to customize the workload in terms of its temporal and intensity characteristics. For instance, one may configure a workload driver to gradually increase the workload intensity over time, as typically is done when evaluating the workload processing capacity of an IDS. A disadvantage is that the workloads generated by such drivers often do not closely resemble real-life workloads. In the case when realistic benign workloads are needed (e.g., to be used as background activities mixed with attacks), a reasonable alternative is the manual generation of benign workloads.

2.1.2. Workloads → Pure Benign → Executable Form → Manual Generation. Under manual generation of workloads, we understand the execution of real system users' tasks known to exercise specific system resources, which is typically applied in the context of evaluating host-based IDSes. For example, a common approach is to use file encoding or tracing tasks to emulate CPU-intensive tasks (e.g., Dunlap et al. [2002] perform ray tracing, Srivastava et al. [2008] perform video transcoding, Lombardi and Di Pietro [2011] perform mp3 file encoding), file conversion and copying of large files to emulate file I/O-intensive tasks (e.g., Lombardi and Di Pietro [2011] and Allalouf et al. [2010] use the UNIX command *dd* to perform file copy operations), and kernel compilation to emulate mixed (i.e., both CPU- and file I/O-intensive) tasks (e.g., performed by Wright et al. [2002], Dunlap et al. [2002], Riley et al. [2008], Lombardi and Di Pietro [2011], and Reeves et al. [2012]).

Provided that it is based on a realistic activity model, this approach of benign workload generation enables the generation of workloads with a behavior similar to the one observed by an IDS during regular system operation. Thus, it is suitable when realistic benign workloads are required (e.g., for training and evaluation of anomaly based IDSes). In addition, it is suitable for generation of workload traces capturing realistic workloads executed in a recording testbed, a topic that we discuss later in Section 2.1.6. However, the manual benign workload generation does not support workload customization as workload drivers do and might require a substantial amount of manpower.

In Table II, we provide an overview of the use of the discussed methods for generating pure benign workloads in practice. We also provide stepwise guidelines (see Selection Guidelines section in Table II) for selecting an approach from those presented in Table II

[5]http://www.spec.org/cpu2000/.
[6]http://www.iozone.org/.
[7]http://fsbench.filesystems.org/bench/postmark-1_5.c.
[8]http://freecode.com/projects/httpbench.
[9]http://www.kegel.com/dkftpbench/.
[10]http://httpd.apache.org/docs/2.2/programs/ab.html.
[11]https://github.com/kdlucas/byte-unixbench.

Table II. Practices for Generating Pure Benign Workloads in Executable

Reference	Method/Workload Type/Approach or Workload Driver
Allalouf et al. [2010]	M / I/O intensive / Creating, deleting and truncating files, appending data to files; M / Mixed / Compilation of libraries
Chung et al. [2013]	M / CPU intensive / Compiling Java code; **M / Network intensive / Web surfing**, Telnet sessions; M / I/O intensive / Reading PDF files; **W / CPU-intensive / SPEC CPU2000**
Dunlap et al. [2002]	M / CPU intensive / Ray tracing; **M / Mixed / Kernel compilation**; W / Network intensive / SPECweb99[12]
Griffin et al. [2003]	M / CPU intensive / Building SSH server; **W / I/O intensive / Postmark**
Jin et al. [2011]	**W / I/O intensive / iozone**; **W / Network intensive / Apachebench, dkftpbench**
Jou et al. [2000]	M / Network intensive / Executing `traceroute`
Laureano et al. [2007]	M / CPU intensive / Executing Linux commands (`ps`, `who`); M / Mixed / Executing Linux commands (`find`, `ls`); M / Network intensive / Downloading files
[Meng and Li 2012]	M / Network intensive / Web surfing, transmitting files
Patil et al. [2004]	M / I/O intensive / File read operations; W / CPU intensive / Am-utils[13]; W / I/O intensive / Postmark
Reeves et al. [2012]	M / Mixed / Server and kernel compilation; W / CPU intensive / SPEC CPU2000; W / Mixed / lmbench[14]
Riley et al. [2008]	M / Mixed / Kernel compilation, executing `insmod`; **W / Mixed / Unixbench**; W / Network intensive / Apachebench
[Srivastava et al. 2008]	**M / CPU intensive / Encoding files**; M / I/O intensive / Copying files; M / Mixed / Video file compression and decompression, kernel compilation
[Wright et al. 2002]	M / Mixed / Kernel compilation; W / Network intensive / Webstone[15]; W / Mixed / lmbench
[Zhang et al. 2008]	W / I/O intensive / iozone

Selection Guidelines

1. Select a method for generating workloads—that is, the use of workload drivers ("workload drivers" in Table II) or manual generation—by taking the advantages and disadvantages of the different methods into account (see Sections 2.1.1 and 2.1.2). In Section 2.3, in the context of IDS evaluation methodologies, we present IDS evaluation scenarios where the different methods are applied for evaluating various IDS properties.
2. Select the type of workloads (e.g., CPU- or I/O intensive) that is required for evaluating the considered IDS property. For instance, CPU- and/or I/O-intensive workloads are required for evaluating the performance overhead of a host-based IDS, and network-intensive workloads are required for evaluating any property of a network-based IDS. In Section 2.3, we present IDS evaluation scenarios where workloads of the different types are used for evaluating IDS properties.
3. Depending on the selection made in step 1, select an approach for manually generating workloads or a workload driver. This is normally done based on subjective criteria (e.g., prior experience with using a given workload driver). In an effort to provide general recommendations for selecting an approach for manually generating workloads/a workload driver, we mark the most popular approaches/workload drivers in bold. Based on what is reported in the surveyed work, we argue that the popularity of the workload drivers marked in bold is due to high configurability, representativeness of the workloads they generate, and ease of use.

Form W, workload drivers; M, manual generation.

to apply in a given IDS evaluation study—that is, to evaluate a given IDS property (e.g., workload processing capacity or performance overhead, see Section 2.3).

2.1.3. Workloads → Pure Malicious → Executable Form → Exploit Database. Pure malicious workloads in executable form are used for evaluating the attack detection coverage of IDSes (see Section 2.3). As pure malicious workloads in executable form, security

[12]http://www.spec.org/web99/.
[13]http://www.am-utils.org/.
[14]http://www.bitmover.com/lmbench/.
[15]http://www.mindcraft.com/webstone/.

Table III. Popular Exploit Repositories

Exploit Database	Exploit Verification	Vulnerable Software	Vulnerability Identifiers		
			CVE	OSVDB	BugTraq
Inj3ct0r (http://www.0day.today)	x				
Exploit database (http://www.exploit-db.com/)	x	x	x	x	
Packetstorm (http://packetstormsecurity.com/)			x	x	
SecuriTeam (http://www.securiteam.com/exploits/)			x	x	
Securityfocus (http://www.securityfocus.com/)		○	x		x

Categorization Criteria	
Criteria	Description
Exploit verification	An exploit repository that fulfills this criterion maintains a record for each hosted attack script indicating whether the script has been empirically verified to successfully exploit a specific vulnerability. This helps IDS evaluators to identify attack scripts that they can easily adapt to their specific requirements.
Vulnerable software	An exploit repository that fulfills this criterion provides a download link to the specific vulnerable software that can be exploited by different attack scripts. This helps IDS evaluators to quickly obtain the vulnerable software and use it to experiment with the scripts. ○ marks partial fulfillment of this criterion—an exploit repository that partially fulfills this criterion provides a link to the Web site of the vendor of the vulnerable software instead of a download link to the software, due to which it takes more time for an IDS evaluator to obtain the vulnerable software.
Vulnerability identifiers	An exploit repository that fulfills this criterion can be searched based on standard vulnerability identifiers. This enables IDS evaluators to quickly locate an attack script that exploits a given vulnerability. Common Vulnerabilities and Exposures (CVE),[16] Open Sourced Vulnerability Database (OSVDB),[17] and BugTraq[18] are the defacto standard vulnerability enumeration systems.

researchers typically use an exploit (i.e., an attack script) database. They have a choice of assembling an exploit database by themselves or by using a readily available one.

Exploit database → Manual assembly. A major disadvantage of the manual assembly is the high cost of the attack script collection process. Locating the attack scripts needed for exploiting specific vulnerabilities and obtaining the required vulnerable software typically is time consuming. In addition, once the needed attack scripts are found, they usually have to be adapted to exploit the vulnerabilities of the victim environments, especially when the attack scripts exploit local system vulnerabilities for evaluating host-based IDSes. This includes time-consuming adaptation of employed exploitation techniques and the like.

Depending on the size of a manually assembled exploit database, the previously mentioned activities might require a considerable amount of manpower to be completed in a reasonable time frame. For instance, Mell et al. [2003] report that based on previous experiences, a single attack script requires approximately one person-week to modify the script's code, to test it, and to integrate it in an IDS evaluation environment. Mell et al. also report that in 2001, the average number of attack scripts used for evaluating IDSes was in the range of 9 to 66. We observe that some recent works, such as Lombardi and Di Pietro [2011], use as low as four attack scripts.

To assemble an exploit database, IDS evaluators normally obtain attack scripts from public exploit repositories. In Table III, we list popular exploit repositories

[16]http://cve.mitre.org/.
[17]http://www.osvdb.org/.
[18]http://www.securityfocus.com/archive/1.

characterized according to the criteria exploit verification, vulnerable software, and vulnerability identifiers (see Categorization Criteria section in Table III). Given that an exploit repository hosts a limited number of attack scripts, an IDS evaluator normally does not search only a single repository, but searches as many as it takes until the desired number of attack scripts is obtained. In this process, we recommend that an IDS evaluator prioritizes the exploit repository "Exploit database" (marked in bold in Table III), as it fulfills more criteria than any other repository presented in Table III (see the Categorization Criteria section for an overview of the benefits of a repository fulfilling the exploit verification, vulnerable software, and vulnerability identifiers criteria).

Publicly available attack scripts normally do not feature techniques for evaluating the ability of an IDS to detect evasive attacks—that is, attacks that are specifically crafted such that an IDS might not detect them. Adapting publicly available attack scripts to feature techniques for evaluating the ability of an IDS to detect evasive attacks normally requires an in-depth knowledge of the architecture and inner working mechanisms of the IDS, a topic that we discuss in detail in Section 2.3.1. Such knowledge may be challenging to obtain if the evaluated IDS is a closed source. Thus, IDS evaluators use third-party tools when executing attack scripts, such as Nikto.[19] Cheng et al. [2012] provide an overview of IDS evasion techniques and discuss the use of the previously mentioned and similar tools for evaluating IDSes.

Exploit database → Readily available exploit database. To alleviate the issues mentioned previously, many researchers employ penetration testing tools as a readily available exploit database. We discuss the Metasploit framework in detail, as it is the most popular penetration testing tool used extensively in both past and recent IDS evaluation experiments (e.g., by Nasr, Kalam, and Fraboul [2012]).[20] Some other penetration testing tools are Nikto, w3af,[21] and Nessus.[22]

The interest in Metasploit is not surprising, given that Metasploit enables a customizable and automated platform exploitation by using an exploit database that is maintained up-to-date and is freely available. However, although convenient, penetration testing frameworks have some critical limitations. Gad El Rab [2008] analyzes the Metasploit's database, showing that most of the exploits are executed from remote sources, and therefore they are most useful when evaluating network-based IDSes and are of limited use for evaluating host-based IDSes.

To provide an up-to-date characterization of Metasploit's exploit database, we analyzed the exploit database of Metasploit version 4.7, the most recent release at the time of this writing. In Table IV, we categorize Metasploit's exploits according to the execution source, target platforms, and exploit rank criteria (see the Categorization Criteria section).[23] Similarly to Gad El Rab, we observe that Metasploit's exploit database contains mostly remote exploits, which makes it most useful for evaluating network-based IDSes. We also observe that a big portion of the exploits in the Metasploit's database have a "great" and "excellent" rank. This indicates that an IDS evaluator can use many attack scripts from Metasploit's database without crashing the victim platform(s). Finally, as we can see in Table IV, most of the remote and local exploits exploit vulnerabilities of Windows platforms.

[19]http://cirt.net/nikto2.
[20]http://www.metasploit.com/.
[21]http://w3af.org/.
[22]http://www.tenable.com/products/nessus-vulnerability-scanner.
[23]https://github.com/rapid7/metasploit-framework/wiki/Exploit-Ranking.

Table IV. Characterization of Metasploit's Exploit Database

	Execution Source																	
	Remote													Local				
Target Platforms / Exploit Rank	AIX	IOS	BSDI	Unix	FreeBSD	HP-UX	Irix	Linux	Netware	OS X	Solaris	Windows	Multi-platform	**Total (per rank)**	Linux	OS X	Windows	Multi-platform	**Total (per rank)**
Manual				4				3		1		5	2	**15**					**0**
Low												10		**10**					**0**
Average				1	1			4	1	8	1	120	3	**139**			2		**2**
Normal				2				7		2		251	6	**268**	2				**2**
Good		2						14	1			159	9	**185**					**0**
Great	2		1	1	3			8		1	3	147	17	**183**	2		2		**4**
Excellent		1		70		1	1	30		2	5	89	84	**283**		2	8	1	**11**
Total (per platform)	**2**	**3**	**1**	**78**	**4**	**1**	**1**	**66**	**2**	**14**	**9**	**781**	**121**		**4**	**2**	**12**	**1**	
Total	**1,083**														**19**				

Categorization Criteria

Criteria	Description
Execution source	An exploit can be executed from a *remote* (i.e., a remote exploit) or a *local* source (i.e., a local exploit). Remote exploits are used for evaluating network-based IDSes, whereas local exploits are used for evaluating host-based IDSes. The amount of remote and local exploits in an exploit database indicates its suitability for evaluating network- and host-based IDSes.
Target platforms	Each exploit is designed to exploit a vulnerability of a single or multiple target platforms (i.e., multi-platform exploits), such as Linux, Solaris, and Windows. An exploit database covering a wide range of platforms is beneficial, because, for example, it can be used to evaluate a variety of IDSes for different target platforms.
Exploit rank	Each exploit in Metasploit's database is ranked according to its impact on the target platform as shown below. The use of attack scripts that do not crash the target platform (e.g., scripts ranked as "excellent") significantly reduces the time spent on restoring it (see Section 2.1). *Manual*: An exploit of this rank almost never successfully exploits a vulnerability and nearly always crashes the target platform. *Low*: An exploit of this rank almost never successfully exploits a vulnerability or successfully exploits a vulnerability in less than 50% of the cases if the target platform is popular. *Average*: An exploit of this rank is unreliable and rarely successfully exploits a vulnerability. *Normal*: An exploit of this rank successfully exploits a vulnerability, but only a vulnerability of a specific version of the target platform and cannot reliably autodetect a vulnerable platform. *Good*: An exploit of this rank has a default target platform (i.e., a platform that the exploit almost always successfully exploits), which is popular. *Great*: An exploit of this rank has a default target platform and detects a vulnerable platform. *Excellent*: An exploit of this rank almost always successfully exploits a vulnerability and never crashes the target platform.

2.1.4. Workloads → Pure Malicious → Executable Form → Vulnerability and Attack Injection. An alternative approach to the use of an exploit database is the use of the vulnerability and attack injection technique. Vulnerability and attack injection enables live IDS testing by first artificially injecting exploitable vulnerable code in a target platform and then attacking the platform. Although not yet mature, this technique is useful in cases where collection of attack scripts that exploit vulnerabilities is unfeasible. As the injected vulnerable code may be exploitable remotely or locally, vulnerability and

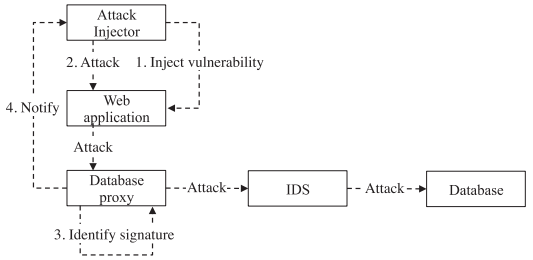


Fig. 2. Use of vulnerability and attack injection to evaluate a network-based IDS.

attack injection is useful for evaluating both host- and network-based IDSes. However, injecting attacks such that the sensors of an IDS under test are exercised may require in-depth knowledge of the architecture and inner working mechanisms of the IDS.

Vulnerability and attack injection relies on the principles of the more general research area of fault injection. Fault injection is an approach for validating specific fault-handling mechanisms and assessing the impact of faults in actual systems. In recent years, the interest in software fault injection has increased providing a basis for many research works on emulation of software faults. A specific application of software fault injection is injection of software faults that represent security vulnerabilities. Fonseca et al. [2014] proposed an approach that enables the automated vulnerability and attack injection of Web applications, which is suitable for evaluating network-based IDSes.

We discuss a scenario where the approach of Fonseca et al. is applied for evaluating a network-based IDS that monitors network traffic to a database, which communicates with a Web application, to detect Standard Query Language (SQL) injection attacks. Fonseca et al. built a Vulnerability Injector, a mechanism that injects vulnerabilities in the source code of Web applications, and an Attack Injector, a mechanism that exploits the injected vulnerabilities. To inject vulnerabilities, the Vulnerability Injector first analyzes the application source code searching for locations where realistic vulnerabilities can be injected by code mutation. The Attack Injector then interacts with the Web application to deliver attack payloads.

In Figure 2, we depict the approach of Fonseca et al. First, the Vulnerability Injector injects a vulnerability in the Web application (*1. Inject vulnerability* in Figure 2), followed by the Attack Injector, which delivers an attack payload with a given signature—that is, an attack identifier (*2. Attack*). The attack payload is targeted at the database (*Attack*). Fonseca et al. developed a database proxy that monitors the communication between the application and the database to identify the presence of attack signatures. When the proxy identifies the signature of the delivered attack payload (*3. Identify signature* in Figure 2), it notifies the Attack Injector that the attack payload has reached the database (*4. Notify*). In this way, the Attack Injector builds a "ground truth" knowledge. Ground truth is information about the attacks used as malicious workloads in a given IDS evaluation study (e.g., time of execution of the attacks). The output of an IDS under test is compared with ground truth information to quantify the attack detection accuracy of the IDS, a topic that we discuss in detail in Section 2.2.

2.1.5. Workloads → Pure Malicious/Pure Benign/Mixed → Trace Form → Trace Acquisition. Under trace acquisition, we understand the process of obtaining real-world production traces from an organization (i.e., nonpublic, proprietary traces) or obtaining publicly available traces that are intended for use in security research.

Trace acquisition → Real-world production traces. Real-world production traces subject an IDS under test to a workload as observed during operation in a real production

environment. However, they are usually difficult to obtain mainly due to the unwillingness of industrial organizations to share operational traces because of privacy concerns. Thus, real-world traces are usually anonymized by using tools for that purpose. Such is the tool tcpmkpub, which anonymizes network traces by modifying recorded network packets at multiple layers of the TCP/IP network stack.[24]

Some organizations are reluctant even toward trace anonymization due to the possibility of information leakages. For instance, trace files may be deanonymized to reveal sensitive internal information (e.g., IP addresses, port numbers, network topologies). Coull et al. [2007] demonstrate the severity of trace deanonymization by revealing anonymized information with 66% to 100% accuracy based on the traces provided by the Lawrence Berkeley National Laboratory (LBNL).[25]

When it comes to the use of anonymization techniques on traces for IDS evaluation, Seeberg and Petrovic [2007] identify the following challenging requirements: (1) during anonymization, the smallest possible amount of intrusion detection relevant data should be removed, and (2) assurance needs to be attained that no private and other sensitive information remains in the trace files after anonymization. For an IDS evaluator, the first requirement is of greatest concern, as an anonymizer might remove data that is relevant for a given IDS under test. Therefore, the provisioning of extensive and accurate metadata on how a given trace file has been anonymized is crucial.

Another challenge is that attacks in real-world production traces are usually not labeled and traces may contain unknown attacks making the construction of the ground truth time consuming since attacks have to be labeled manually. Lack of ground truth information severely limits the usability of trace files in IDS evaluation. For instance, it might be impossible to quantify the false-negative detection rate of an IDS under test (see Section 2.2).

Trace acquisition → Publicly available traces. In contrast to real-world traces, one can obtain publicly available traces without any legal constraints. However, the use of such traces has certain risks. For instance, publicly available traces often contain errors and quickly become outdated after their release since the recorded attacks have limited shelf-life. Consequently, claims on the generalizability of results from IDS evaluation studies based on publicly available traces can often be questioned. An in-depth knowledge about the characteristics of recorded activities in publicly available traces (e.g., types and distributions of recorded attacks) is a requirement for the accurate interpretation of results from IDS evaluation studies based on such traces.

The DARPA and the derived Knowledge Discovery and Data Mining Cup 1999 (KDD-99) datasets are the result of one of the most notable efforts up-to-date to provide publicly available data for security research. In three consecutive years (i.e., 1998, 1999, and 2000), three separate editions of the DARPA datasets have been made available. Since they contain local and remote attacks, the DARPA datasets are suitable for evaluating host- and network-based IDSes. They also contain training data, which makes them useful for evaluating anomaly based IDSes. However, the DARPA and KDD-99 datasets are currently considered outdated and have been often criticized (see Sommer and Paxson [2010] and McHugh [2000]). Despite the criticism, these traces are still used in many recent IDS evaluation experiments (e.g., by Yu and Dasgupta [2011] and Raja et al. [2012]).

In Table V, we provide an overview of popular repositories of publicly available traces categorized according to multiple criteria (see the Categorization Criteria section): the

[24]http://www.icir.org/enterprise-tracing/tcpmkpub.html.
[25]http://www.icir.org/enterprise-tracing/.

Table V. Repositories of Publicly Available Traces

Trace Repository	Content	Activities	Labeled	Realistic	Anonymized	Metadata	Access Restrictions
CAIDA	Mixed	Network	No	Yes	Partially[a]	Yes	Partial[b]
DEFCON	Pure malicious	Network	No	No	No	No	No
DARPA/KDD-99	Mixed	Network/Host	Yes	No	No	Yes	No
ITA	Pure benign	Network	n/a	Yes	Partially[c]	No	No
LBNL/ICSI	Pure benign	Network	n/a	Yes	Yes	Yes	No
MAWILab	Mixed	Network	Yes	Yes	Yes	Yes	No

Categorization Criteria

Criteria	Description
Content	The traces hosted in a given repository may contain only benign activities (pure benign), only attacks (pure malicious), or both benign activities and attacks (mixed, see Figure 1).
Activities	The traces hosted in a given repository may contain network and/or host activities. The former are needed for evaluating network-based IDSes and the latter for evaluating host-based IDSes.
Labeled	The attacks recorded in the traces hosted in a given repository may or may not be labeled. Labeled attacks enable an IDS evaluator to observe whether the IDS under test detects the recorded attacks.
Realistic	The traces hosted in a given repository may or may not be realistic. A trace is considered to be realistic if it has been captured during regular operation of a network or host environment and has not been modified after capturing [Shiravi et al. 2012]. Realistic traces enable the representative evaluation of IDSes.
Anonymized	The traces hosted in a given repository may or may not be anonymized. Anonymized traces may lack information that is crucial for intrusion detection (see Section 2.1.5).
Metadata	Metadata on how activities stored in trace files have been recorded and anonymized may or may not be provided. Metadata is important for accurately interpreting results from IDS evaluation experiments where traces have been used as workloads (see Section 2.1.5).
Access restrictions	The traces hosted in a given repository may be available to the general public or only to certain individuals satisfying specific requirements, such as employment by a research organization or a government agency.

Selection Guidelines

1. Select the trace repositories with the appropriate value of the Activities criterion with respect to the type of the tested IDS (i.e., network- or host-based IDS)—that is, "network" for evaluating network-based IDSes and "host" for evaluating host-based IDSes (see the description of the Activities criterion in the Categorization Criteria section of this table).
2. Select the trace repositories with the appropriate value of the Content criterion with respect to the evaluated IDS property–for example, "pure malicious" for evaluating attack detection coverage, "mixed" for evaluating attack detection accuracy, and so on (see the description of the Content criterion in the Categorization Criteria section of this table). In Section 2.3, we provide details on the use of pure malicious, pure benign, or mixed workloads for evaluating different IDS properties (see Table VIII).
3. Select the repository with values of the Labeled, Realistic, Anonymized, Metadata, and Access restrictions criteria such that the benefit of using the repository is maximal (see the Categorization Criteria section of this table for an overview of the benefits of a trace repository fulfilling or not fulfilling the previously mentioned criteria).

[a]The IP addresses recorded in some trace files are anonymized.
[b]Some trace files are available only to members of CAIDA's membership program.
[c]Some trace files are anonymized such that IP addresses are modified and all packet contents are removed.

Cooperative Association for Internet Data Analysis (CAIDA),[26] the Defense Readiness Condition (DEFCON),[27] the DARPA/KDD–99,[28,29] the Internet Traffic Archive (ITA),[30] the LBNL/ISCI (International Computer Science Institute), and the MAWILab trace repositories.[31] We also provide stepwise guidelines (see the Selection Guidelines section of Table V) for selecting a trace repository from those presented in Table V to use in a given IDS evaluation study—that is, to evaluate a given IDS property (see Section 2.3).

2.1.6. Workloads → Pure Malicious/Pure Benign/Mixed → Trace Form → Trace Generation. Under trace generation, we understand the process of generating traces by the IDS evaluator himself. To avoid the issues with acquiring traces (see Section 2.1.5), researchers generate traces in a testbed environment or deploy a honeypot to capture malicious activities.

Trace generation → Testbed environment. Traces that contain benign and malicious workloads can be generated in a testbed environment by using the previously mentioned methods for generating workloads in executable form and capturing the workloads in trace files. The generation of traces in a testbed environment may be challenging. For instance, the costs of building a testbed that scales to realistic production environments may be high. Further, generating large-scale workloads for recording, especially when performed manually (see Section 2.1.2), may be time consuming and require a considerable amount of manpower. Finally, the method used for trace generation may produce faulty or simplistic workloads. For instance, Sommer and Paxson [2010] warn that activities captured in small testbed environments differ fundamentally from activities in a real-life environment.

An approach to alleviate the issue of generating faulty workloads is to observe the network and/or host activities in a production environment to construct a realistic activity model. The latter can then be used as a basis for the generation of realistic live workloads for the purpose of recording trace files. The DARPA datasets were generated according to the preceding approach. Cunningham et al. [1999] observed the network activities in an Air Force base by deploying network traffic sniffers to record types and amounts of used network services, information that they used as a basis for generating and recording benign workloads.

Although useful, the use of the preceding approach results in one-time datasets (i.e., datasets that resemble the real world only for a given time period after the trace generation). This issue has motivated a current research direction focusing on the generation of traces in a testbed environment in a customizable manner. For instance, Shiravi et al. [2012] proposed the use of workload profiles enabling the specification and customization of malicious and benign network traffic that can be captured in trace files. They introduced α-profiles for the specification of attack scenarios with attack description languages and β-profiles for the specification of mathematical distributions or behaviors of certain entities (e.g., distribution of network packet sizes). The α- and β-profiles support the generation of datasets that can be modified, extended, and reproduced. Using the specifications defined in the profiles, manually or using scripted

[26]http://www.caida.org/data/.
[27]http://cctf.shmoo.com/.
[28]http://www.ll.mit.edu/ideval/data/.
[29]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
[30]http://ita.ee.lbl.gov/html/traces.html.
[31]http://www.fukuda-lab.org/mawilab/.

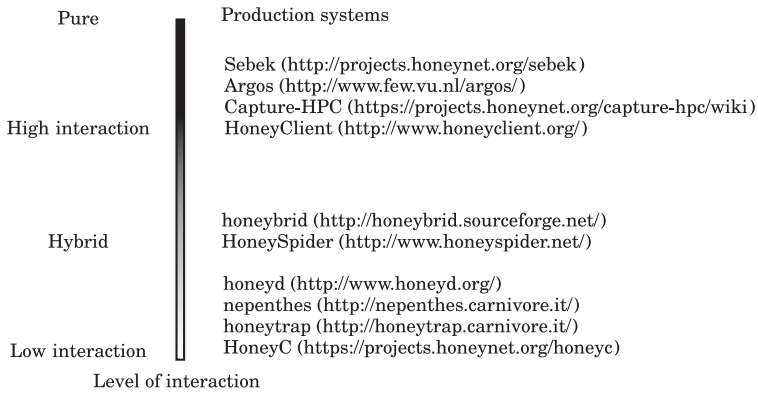


Fig. 3. Honeypots of different levels of interaction.

agents, one can generate both malicious and benign workloads in a testbed environment for recording.

Trace generation → Honeypots. By mimicking real systems and/or vulnerable services, honeypots enable the interaction and recording of host and/or network malicious activities performed by an attacker without revealing their purpose. Since honeypots are usually isolated from production platforms, most of the interactions that they observe are malicious, making honeypots ideal for generation of pure malicious traces. However, given that honeypots interact with real attackers, the outcome of a trace generation campaign performed by using a honeypot (e.g., amount and types of recorded attacks) is uncertain, as it cannot be planned in advance and controlled.

Based on their level of interaction with attackers, honeypots can be categorized into low-interaction, high-interaction, and pure honeypots. Low-interaction honeypots mimic only specific vulnerable services by using scripts, high-interaction honeypots mimic production systems by using actual operating systems, and pure honeypots are full-fledged production systems equipped with logging tools.

Low-interaction honeypots are not flexible, can be easily detected by attackers, and cannot interact with zero-day attacks. However, they are not expensive to maintain and are useful for recording malicious activities that can be easily labeled. High-interaction and pure honeypots are flexible, can interact with zero-day attacks, and are not easily detectable; however, they are expensive to maintain and require time-consuming analysis of recorded activities for the purpose of labeling recorded attacks to construct ground truth. There are also *hybrid* honeypots, which combine the advantages of low- and high-interaction honeypots. In Figure 3, we present commonly used honeypots of the previously mentioned levels of interaction.

2.2. Metrics

In Figure 4, we depict the metrics part of the IDS evaluation design space. We distinguish between two metric categories: performance-related metrics and security-related metrics. Under *performance-related* metrics, we consider metrics that quantify the non-functional properties of an IDS under test, such as capacity (e.g., see Meng and Li [2012]) and resource consumption (e.g., see Sinha et al. [2006]). In this article, we focus on security-related metrics. Under *security-related* metrics, we consider metrics that quantify the attack detection accuracy of an IDS. We distinguish between *basic* and *composite* security-related metrics (see Figure 4). We provide an overview of the most commonly used basic and composite security-related metrics in Table VI. In this table,

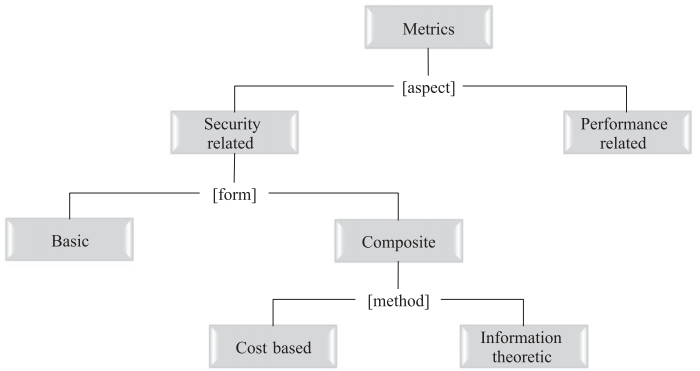


Fig. 4. IDS evaluation design space: metrics. There are two types of metrics with respect to the aspect of IDS behavior they quantify: security related (quantify IDS attack detection accuracy) and performance related (quantify nonfunctional IDS properties). There are two types of security-related metrics with respect to metric form: basic (Section 2.2.1) and composite, which are metrics derived from basic metrics (Section 2.2.2). There are two types of composite metrics with respect to used measurement method (Section 2.2.2): cost based and information theoretic.

we also show the notation, formulas, and value domains of used symbols (including variables).[32]

2.2.1. Metrics → Security Related → Basic. The basic metrics quantify various individual attack detection properties. Although they are quantified individually, these properties need to be analyzed *together* to accurately characterize the attack detection efficiency of an IDS. For instance, the false-negative rate $\beta = P(\neg A|I)$ quantifies the probability that an IDS does not generate an alert when an intrusion occurs; therefore, the true-positive rate $1 - \beta = 1 - P(\neg A|I) = P(A|I)$ quantifies the probability that an alert generated by an IDS is really an intrusion. The false-positive rate $\alpha = P(A|\neg I)$ quantifies the probability that an alert generated by an IDS is not an intrusion but a regular benign activity; therefore, the true-negative rate $1 - \alpha = 1 - P(A|\neg I) = P(\neg A|\neg I)$ quantifies the probability that an IDS does not generate an alert when an intrusion does not occur. In IDS evaluation experiments, the output of the IDS under test is compared with ground truth information to calculate the basic metrics.

Other basic metrics are the positive predictive value (PPV) and the negative predictive value (NPV). The first quantifies the probability that there is an intrusion when an IDS generates an alert, whereas the latter quantifies the probability that there is no intrusion when an IDS does not generate an alert. These metrics are calculated by using the Bayesian theorem for calculating a conditional probability (Table VI). PPV and NPV are interesting from a usability perspective, such as in situations when an intrusion alert triggers an attack response. In these situations, low values of PPV and NPV indicate that the considered IDS is not optimal for deployment. For example, a low value of PPV (therefore a high value of its complement $1 - P(I|A) = P(\neg I|A)$) indicates that the considered IDS may often cause the triggering of attack response actions when no real attacks have actually occurred.

2.2.2. Metrics → Security Related → Composite. Security researchers often combine the basic metrics to analyze relationships between them. Such an analysis is used to discover an optimal IDS operating point (i.e., an IDS configuration that yields optimal values of both the true- and false-positive detection rate) or to compare multiple IDSes.

[32]In Table VI, P and p denote a probability, $\mathbb{R}$ denotes the set of real numbers, $\mathbb{R}^+$ denotes the set of positive real numbers excluding zero, and $\mathbb{R}_0^+$ denotes the set of positive real numbers including zero.

Table VI. Common Metrics for Quantifying IDS Attack Detection Accuracy

Metric Form	Metric	Annotation/Formula
Basic	False-negative rate	$\beta = P(\neg A\|I)$
	True-positive rate	$1 - \beta = 1 - P(\neg A\|I) = P(A\|I)$
	False-positive rate	$\alpha = P(A\|\neg I)$
	True-negative rate	$1 - \alpha = 1 - P(A\|\neg I) = P(\neg A\|\neg I)$
	Positive predictive value	$P(I\|A) = \frac{P(I)P(A\|I)}{P(I)P(A\|I)+P(\neg I)P(A\|\neg I)}$
	Negative predictive value	$P(\neg I\|\neg A) = \frac{P(\neg I)P(\neg A\|\neg I)}{P(\neg I)P(\neg A\|\neg I)+P(I)P(\neg A\|I)}$
Composite	Expected cost	$C_{exp} = Min(C\beta B, (1-\alpha)(1-B)) + Min(C(1-\beta)B, \alpha(1-B))$
	Intrusion detection capability	$C_{ID} = \frac{I(X;Y)}{H(X)}$

Notations and Properties of Used Symbols

Symbol	Meaning	Formula/Value Domain
A	*Alert event*: An IDS generates an attack alert	n/a
I	*Intrusion event*: An attack is performed	n/a
C_α	Cost of an IDS generating an alert when an intrusion has not occurred	$C_\alpha \in \mathbb{R}_0^+$
C_β	Cost of an IDS failing to detect an intrusion	$C_\beta \in \mathbb{R}^+$
C	*Cost ratio*: The ratio between the costs C_α and C_β	$C = C_\beta / C_\alpha : C \in \mathbb{R}_0^+$
B	*Base rate*: Prior probability that an intrusion event occurs	$B = P(I) : B \in \mathbb{R} \rightarrow [0;1]$
X	*IDS input*: Discrete random variable used to model input to an IDS such that $X = 0$ represents a benign activity and $X = 1$ represents a malicious activity (i.e., an intrusion)	$X = 0 \vee X = 1$
Y	*IDS output*: Discrete random variable used to model the generation of alerts by an IDS such that $Y = 0$ represents no alert and $Y = 1$ represents an alert	$Y = 0 \vee Y = 1$
$H(X)$	*Uncertainty of X*: Entropy measure quantifying the uncertainty of the IDS input X	$H(X) = -\sum_x p(x) log p(x) : x = 0 \vee x = 1, p(x) \in \mathbb{R} \rightarrow [0;1]$
$I(X;Y)$	*Mutual information*: The amount of information shared between the random variables X and Y—in other words, the amount of reduction of the uncertainty of the IDS input (X) after the IDS output (Y) is known	$I(X;Y) = \sum_x \sum_y p(x,y) log \frac{p(x,y)}{p(x)p(y)} : x = 0 \vee x = 1, y = 0 \vee y = 1, p(x) \wedge p(y) \wedge p(x,y) \in \mathbb{R} \rightarrow [0;1], 0 \leq I(X;Y) \leq H(X)$

In this section, we focus on comparing the applicability of composite security-related metrics for the purpose of comparing IDSes, which includes the identification of optimal IDS operating points.

A receiver operating characteristic (ROC) curve plots true-positive rate against the corresponding false-positive rate exhibited by a detector. In the context of IDSes, an ROC curve depicts multiple IDS operating points of an IDS under test, and as such, it is useful for identifying an optimal operating point or for comparing multiple IDSes.

An open issue is how to determine a proper unit and measurement granularity for the false-positive and true-positive rates based on which an ROC curve is plotted. Different units of measurement might yield different rates, and therefore the selection of a proper unit is considered as a task that needs to be performed with care. Gu et al. [2006] acknowledge the importance and scope of the preceding issue by referring to it as a "general problem for all the existing [IDS] evaluation metrics." They discuss this issue

Table VII. Values of $1-\beta$, PPV_{ID}, C_{exp}, C_{rec}, and C_{ID} for IDS_1 and IDS_2

		IDS$_1$				IDS$_2$			
α	PPV_{ZRC}	$1-\beta$	PPV_{ID}	$C_{exp/rec}$	C_{ID}	$1-\beta$	PPV_{ID}	$C_{exp/rec}$	C_{ID}
0.005	0,9569	0.9885	0,9565	**0.016**	**0.9159**	0.973	0,9558	0.032	**0.8867**
0.010	0,9174	0.99	0,9167	0.019	0.8807	0.99047	0,9167	0.019	0.8817
0.015	0,8811	0.9909	0,8801	0.022	0.8509	0.99664	0,8807	**0.017**	0.8635

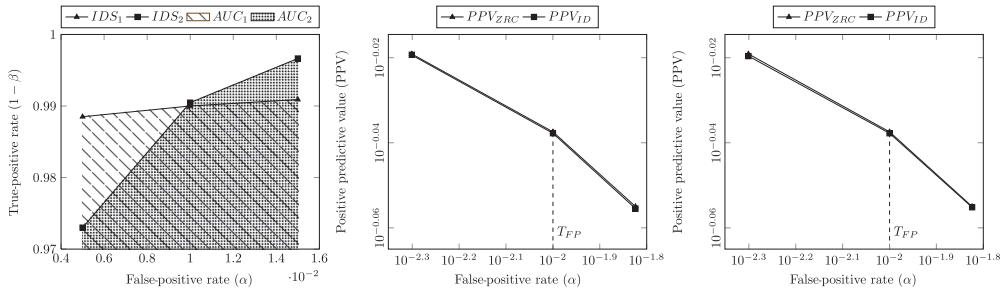


Fig. 5. IDS comparison with ROC curves (a) and the intrusion detection effectiveness metric (b, c).

in the context of the evaluation of network-based IDSes and state that depending on the unit of analysis in a network-based IDS, at least two different units of measurement exist (i.e., a unit of packet and flow), which makes the comparison of IDSes with these units of analysis challenging. Gu et al. [2006] recommend the conversion of different units of measurement to the same unit when possible for a fair and meaningful IDS comparison (e.g., conversion of a packet-level to a flow-level unit by defining a flow as malicious when it contains a malicious packet). Next, we analyze and demonstrate the use of ROC curves and related metrics through a case study scenario.

Case study #1. Let's consider the comparison of two IDSes, IDS_1 and IDS_2, and analyze the relationship between the true-positive $(1-\beta)$ and the false-positive (α) detection rate. We assume that for IDS_1, $1-\beta$ is related to α with a power function (i.e., $1-\beta=\alpha^k$) such that $k=0.002182$. We assume that for IDS_2, $1-\beta$ is related to α with an exponential function (i.e., $1-\beta=1-0.00765e^{-208.32\alpha}$). We obtain the values of k, α, and the coefficients of the exponential function from Gaffney and Ulvila [2001]. We calculate the values of $1-\beta$ for IDS_1 and IDS_2 for $\alpha=\{0.005, 0.010, 0.015\}$. The respective values are shown in Table VII.

In Figure 5(a), we depict the ROC curves that express the relationship between $1-\beta$ and α for IDS_1 and IDS_2. The ROC curves intersect approximately at $1-\beta=0.99$ and $\alpha=0.01$. Thus, the better IDS cannot be identified in a straightforward manner. An IDS is considered as better if it features a higher true-positive detection rate $(1-\beta)$ at all operating points along the ROC curve.

An intuitive solution to the preceding problem, as suggested by Durst et al. [1999], is to compare the area under the ROC curves (i.e., $AUC_1 : \int_{\alpha=0.005}^{\alpha=0.015} \alpha^{0.002182}\mathrm{d}\alpha$ and $AUC_2 : \int_{\alpha=0.005}^{\alpha=0.015}(1-0.00765e^{-208.32\alpha})\mathrm{d}\alpha$). However, Gu et al. [2006] consider such a comparison as unfair, as it is based on all operating points of the compared IDSes, whereas in reality a given IDS is always configured according to a single operating point.

The ROC curves depicted in Figure 5(a) do not express the impact of the rate of occurrence of intrusion events ($B=P(I)$), known as base rate, on α and $1-\beta$. The attack detection performance of an IDS should be assessed with respect to a base

rate measure for such an assessment to be accurate (see Axelsson [2000]). The error occurring when α and $1 - \beta$ are assessed without taking the base rate into account is known as the base-rate fallacy.

To address the preceding issue, Nasr et al. [2012] propose a metric called *intrusion detection effectiveness* (E_{ID}). E_{ID} is calculated based on comparing the ideal and actual performance of an IDS depicted in the form of IDS operation curves called *zero reference curve* (ZRC) and *actual IDS operation curve*, respectively. An IDS operation curve plots PPV, which contains measure of the base rate B (see Table VI), against α. Given a specific value of B, the ZRC plots PPV (denoted by PPV_{ZRC}), calculated assuming an ideal operation of the tested IDS—that is, the IDS does not miss attacks ($1 - \beta = 1$). The actual IDS operation curve plots the actual PPV (denoted by PPV_{ID}) exhibited by the IDS. The value of E_{ID} is the normalized variance between the ZRC and the actual IDS operation curve over the interval $[0, T_{FP}]$, where T_{FP} is the maximum acceptable α exhibited by the IDS—that is, $E_{ID} = \frac{1}{\int_0^{T_{FP}} PPV_{ZRC} \mathrm{d}\alpha}(\int_0^{T_{FP}} PPV_{ZRC} \mathrm{d}\alpha - \int_0^{T_{FP}} PPV_{ID} \mathrm{d}\alpha)$, $E_{ID} \in [0; 1]$, such that the lesser the E_{ID}, the better the performance of the IDS under test.

In Table VII, we present PPV_{ZRC} and PPV_{ID} for IDS_1 and IDS_2, calculated assuming that $B = 0.1$. In Figure 5(b) and (c) (the axes are in logarithmic scale), we depict the ZRC and the actual IDS operation curve for IDS_1 and IDS_2. These curves are very similar due to the high PPVs (i.e., close to the ideal PPV — PPV_{ZRC}) exhibited by IDS_1 and IDS_2. We calculate E_{ID} of 0.0004 for IDS_1 and E_{ID} of 0.0011 for IDS_2 assuming that $T_{FP} = 0.01$, based on which we conclude that IDS_1 performs better.

Although E_{ID} expresses the impact of the base rate on α and $1 - \beta$, it suffers from the same issue as the metric proposed by Durst et al. [1999]—that is, a comparison of IDSes based on E_{ID} may be misguiding, as it is based on multiple operating points of the compared IDSes (see Gu et al. [2006]).

Cost-based and information-theoretic metrics. Due to the issues mentioned previously, researchers have proposed novel metrics that can be classified into two main categories: (1) metrics that use cost-based measurement methods and (2) metrics that use information-theory measurement methods (see Figure 4). In the following, we discuss metrics that belong to these categories, focusing on the *expected cost* and *intrusion detection capability* metrics described in the seminal works of Gaffney and Ulvila [2001] and Gu et al. [2006].

Cost-based metrics. Gaffney and Ulvila [2001] propose the measure of cost as an IDS evaluation parameter. They combine ROC curve analysis with cost estimation by associating an estimated cost with each IDS operating point. The measure of cost is relevant in scenarios where a response that may be costly is taken (e.g., stopping a network service) when an IDS generates an attack alert. Gaffney and Ulvila introduce a cost ratio $C = C_\beta / C_\alpha$, where C_α is the cost of an IDS alert when an intrusion has not occurred, and C_β is the cost of not detecting an intrusion when it has occurred. Gaffney and Ulvila use the cost ratio to calculate the expected cost C_{exp} of an IDS operating at a given operating point (see Table VI). Using C_{exp}, one can compare IDSes by comparing the estimated costs when each IDS operates at its optimal operating point. The IDS that has lower C_{exp} associated with its optimal operating point is considered better. An IDS operating point is considered optimal if it has the lowest C_{exp} associated with it compared to the other operating points.

The formula of C_{exp} (see Table VI) can be obtained by analyzing the decision tree depicted in Figure 6(a). The decision tree shows the costs that may be incurred by an IDS (e.g., C_α and C_β) with respect to the operation of the IDS (i.e., generation of alerts) and the responses that can be taken; uncertain events (e.g., the generation of an alert)

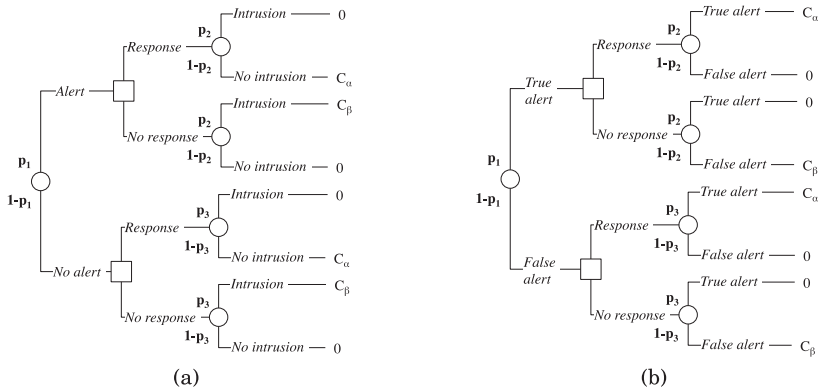


Fig. 6. Decision tree for calculating expected cost (a) and relative expected cost (b).

are depicted by circles and actions are depicted by squares. In Figure 6(a), we depict the probabilities $p_1 = P(A)$, $p_2 = P(I|A) = PPV$, and $p_3 = P(I|\neg A)$ (see Table VI). The formula of C_{exp} is obtained by "rolling back" the tree depicted in Figure 6(a)—that is, from right to left, the expected cost at an event node is the sum of products of probabilities and costs for each branch, and the expected cost at an action node is the minimum of expected costs on its branches. The formula of C_{exp} shown in Table VI can be derived using the basic algebra of probability theory (see Gaffney and Ulvila [2001]).

Meng [2012] proposed a cost-based IDS evaluation metric called *relative expected cost* (C_{rec}). This metric is intended for comparing modern IDSes that use false alert filters (e.g., see Chiu et al. [2010]). A false alert filter detects false alerts generated by an IDS. The response taken when a false alert filter labels an alert as false is filtering out the alert before it is reported by the IDS. C_{rec} is based on the previously discussed expected cost metric. In contrast to C_{exp}, C_{rec} measures cost associated with the accuracy of an IDS's false alert filter at classifying alerts as true or false, which can be used as an IDS comparison parameter. C_{rec} can be associated with each IDS operating point on an ROC curve and can be used for comparing IDSes same as C_{exp}.

The formula of C_{rec} ($C_{rec} = C\beta B + \alpha(1 - B)$) can be obtained in a way similar to obtaining the formula of C_{exp}—that is, by "rolling back" the decision tree depicted in Figure 6(b) (see Meng [2012]). This tree is a modified version of the tree depicted in Figure 6(a). In Figure 6(b), p_1 denotes the probability that the false alert filter reports a true alert, p_2 denotes the conditional probability of true alert given that the filter reports a true alert, and p_3 denotes the conditional probability of true alert given that the filter reports a false alert. In the context of the work of Meng, α and β denote the false-positive and false-negative rate exhibited by a false alert filter. Further, B denotes the prior probability of a false alert, and C_α and C_β denote the cost of classifying a false alert as a true alert, and the cost of classifying and filtering out a true alert as a false alert, respectively. C is the cost ratio C_β/C_α.

Although the discussed cost-based metrics enable straightforward comparison of IDSes, they depend on the cost ratio C. To calculate the cost ratio, one would need a cost-analysis model that can estimate C_α and C_β, which might be difficult to construct in reality. Cost-analysis models take parameters into consideration that might not be easy to measure, or might not be measurable at all (e.g., man-hours, system downtime). Further, C_{exp} and C_{rec} enable the comparison of IDSes based on a subjective measure, making the metrics unsuitable for objective comparisons [Gu et al. 2006]. However, cost-based metrics may be of value when the relationships between the different attack

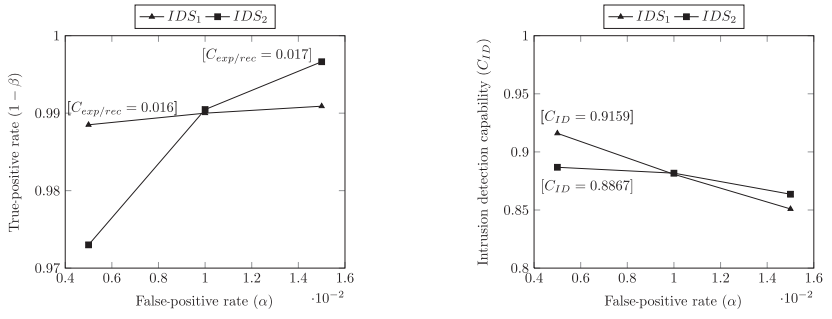


Fig. 7. IDS comparison with the expected cost and relative expected cost metric (a) and the intrusion detection capability metric (b).

detection costs (e.g., cost of missing an attack, cost of a false alert) can be estimated and when such estimations would be considered as sufficiently accurate. For instance, given a statement such as "a false alert is twice as costly as a missed attack," a cost-based metric would be crucial to identify an optimal IDS operating point. Next, we demonstrate the use of the expected cost metric (C_{exp}) and the relative expected cost metric (C_{rec}) for comparing IDSes through a case study scenario.

Case study #2. First, we compare IDS_1 and IDS_2 (see Case study #1) using C_{exp}. The IDS that has lower C_{exp} associated with its optimal operating point (i.e., the point that has the lowest C_{exp} associated with it) is considered better. To determine the optimal operating points of IDS_1 and IDS_2, we calculate C_{exp} for each operating point of the two IDSes. To calculate C_{exp}, we assume that $C = 10$ (i.e., the cost of not responding to an attack is 10 times higher than the cost of responding to a false alert) and $B = 0.10$. We present the values of C_{exp} in Table VII. The optimal operating point of IDS_1 is $(0.005, 0.9885)$ and of IDS_2 is $(0.015, 0.99664)$. Since the minimal C_{exp} of IDS_1 (0.016) is smaller than the minimal C_{exp} of IDS_2 (0.017), we conclude that IDS_1 performs better. In Figure 7(a), we depict the ROC curves annotated with the minimal C_{exp} of IDS_1 and IDS_2.

We now compare IDS_1 and IDS_2 using C_{rec}. The IDS that has lower C_{rec} associated with its optimal operating point (i.e., the point that has the lowest C_{rec} associated with it) is considered better. To calculate C_{rec}, we assume that $C = 10$ and $B = 0.10$, and that IDS_1 and IDS_2 use false alert filters. We present the values of C_{rec} in Table VII. For the sake of simplicity, we assume that α and $1 - \beta$ exhibited by IDS_1 and IDS_2 (see Case study #1 and Table VII) correspond to α and $1 - \beta$ exhibited by the IDS's false alert filters. This results in identical values of C_{rec} and C_{exp}. Using the same approach for comparing IDSes as the one we used when comparing IDSes using C_{exp}, we conclude that IDS_1 performs better than IDS_2 in terms of incurred costs that are associated with the accuracy of the IDS's false alert filters. In Figure 7(a), we depict the ROC curves annotated with the minimal C_{rec} of IDS_1 and IDS_2.

Information-theoretic metrics. Another approach for evaluating the attack detection accuracy of an IDS is the information-theoretic approach. Gu et al. [2006] propose a metric called *intrusion detection capability* (C_{ID}, see Table VI). They model the input to an IDS as a stream of a random variable X ($X = 1$ denotes an intrusion, $X = 0$ denotes benign activity), and the IDS output as a stream of a random variable Y ($Y = 1$ denotes IDS alert, $Y = 0$ denotes no alert). The input and output stream have a certain degree of uncertainty reflected by the entropies $H(X)$ and $H(Y)$, respectively. Thus, Gu et al. model the number of correct guesses of an IDS (i.e., $I(X;Y)$) as mutual shared

information between the random variables X and Y (i.e., $I(X;Y) = H(X) - H(X|Y)$). C_{ID} is obtained by normalizing $I(X;Y)$ with $H(X)$ (see Table VI).

C_{ID} incorporates the uncertainty of the input stream $H(X)$ (i.e., the distribution of intrusions in the IDS input) and the accuracy of an IDS under test $I(X;Y)$—that is, C_{ID} incorporates the base rate B, the true-positive rate $(1 - \beta)$, and the false-positive rate (α). For the definition of the relationship between C_{ID}, on the one hand, and B, $1 - \beta$, and α, on the other hand, we refer the reader to Gu et al. [2006]. Given this relationship, a value of C_{ID} may be assigned to any operating point of an IDS on the ROC curve. With this assignment, one obtains a new curve (i.e., a C_{ID} curve).

The information-theoretic approach is generic and enables the evaluation of IDSes of different designs. For instance, Meng and Kwok [2013] discuss the application of the intrusion detection capability metric for quantifying the attack detection accuracy of IDSes that use false alert filters. They fine-tuned this metric and developed another called *false alarm reduction capability*, $RC_{FA} = \frac{I(X;Y)}{H(X)}$. In the context of the work of Meng and Kwok, X denotes the input to a false alert filter (i.e., alerts generated by an IDS) and Y denotes the filter's output. Meng and Kwok [2013] show that the information-theoretic approach can be applied in practice for evaluating IDSes that use false alert filters by evaluating Snort [Roesch 1999] configured to use a variety of false alert filters. Next we show how one can compare IDSes using the intrusion detection capability metric (C_{ID}) through a case study scenario.

Case study #3. Assuming a base rate of $B = 0.10$, we calculated C_{ID} for the operating points of IDS_1 and IDS_2 (see Case study #1) presented in Table VII. In Figure 7(b), we depict the C_{ID} curves of IDS_1 and IDS_2. A C_{ID} curve provides a straightforward identification of the optimal operating point of an IDS (i.e., the point that marks the highest C_{ID}). One can compare IDSes by analyzing the maximum C_{ID} of each IDS and considering as better performing the IDS whose optimal operating point has higher C_{ID} associated with it. From Table VII, one would consider IDS_1 to perform better since it has greater maximum C_{ID} (0.9159) than the maximum C_{ID} of IDS_2 (0.8867). In contrast to the previously discussed expected cost metric, C_{ID} is not based on subjective measures such as cost, which makes it suitable for objective comparison of IDSes.

2.3. Measurement Methodology

Under measurement methodology, we understand the specification of the *IDS properties* of interest (e.g., attack detection accuracy, capacity) and of the employed workloads and metrics for evaluating the properties.

After examining IDS evaluation experiments covering different types of IDSes, we identified nine IDS properties that are commonly considered in practice. In Table VIII, we present these properties, some of which are grouped into categories. For the sake of clarity, we also present definitions of the IDS properties attack detection accuracy, attack coverage, performance overhead, and workload processing capacity. Further, in Table VIII, we provide an overview of the workload and metric requirements (see Sections 2.1 and 2.2) for evaluating the different properties.

In Table IX, we list references to the surveyed publications where representative studies that investigate the respective properties can be found. Table IX compares the surveyed work in terms of types of tested IDSes (see Table I) and considered IDS properties. This enables the identification of common trends in evaluating IDS properties for different types of IDSes. Next, we discuss such trends and provide recommendations and key best practices, which we identified based on reported benefits of applying the practices. We also present observed quantitative values (e.g., acceptable performance overheads and attack detection speeds) and relevant observations (e.g., evasion

Table VIII. IDS Evaluation Design Space: Measurement Methodology

IDS Property	Workloads [Content]	Metrics [Aspect]	Metrics [Form]
Attack Detection Related			
Attack detection accuracy	Mixed	Security related	Basic, composite
Attack coverage	Pure malicious	Security related	Basic
Resistance to evasion techniques	Pure malicious, mixed	Security related	Basic
Attack detection and reporting speed	Mixed	Performance related	n/a
Resource Consumption Related			
CPU consumption Memory consumption Network consumption	Pure benign	Performance related	n/a
Performance overhead	Pure benign	Performance related	n/a
Workload processing capacity	Pure benign	Performance related	n/a

Definitions of IDS Properties

IDS Property	Definition
Attack detection accuracy	The attack detection accuracy of an IDS in the presence of mixed workloads.
Attack coverage	The attack detection accuracy of an IDS in the presence of attacks without any background benign activity.
Performance overhead	The overhead incurred by an IDS on the system and/or network environment where it is deployed. Under overhead, we understand performance degradation of users' tasks/operations caused by (a) consumption of system resources (e.g., CPU, memory) by the IDS and/or (b) interception and analysis of the workloads of users' tasks/operations (e.g., network packets) by the IDS.
Workload processing capacity	The rate of arrival of workloads to an IDS for processing in relation to the amount of workloads that the IDS discards (i.e., does not manage to process). For instance, in the context of network-based IDSes, capacity is normally measured as the rate of arrival of network packets to an IDS over time in relation to the amount of discarded packets over time. The capacity of an IDS may also be defined as the maximum workload processing rate of the IDS such that there are no discarded workloads.

techniques to which current IDSes are vulnerable) that may serve as reference points for designing and evaluating future IDSes.

Attack detection accuracy/attack coverage. As expected, these properties are evaluated for IDSes of all types. Due to the longevity of their presence on the IDS evaluation scene, at this time the DARPA and the KDD-99 Cup datasets (see Section 2.1.5) represent standard workloads for comparing novel anomaly based IDSes with their past counterparts in terms of their attack detection accuracy. For the sake of representativeness, we recommend the evaluation of a single IDS using not the DARPA and the KDD-99 Cup datasets but workloads that contain current attacks (see Section 2.1). We observed that the attack detection rates of IDSes reported in the surveyed work vary greatly—that is, between 8% and 97%, measures that largely depend on the configurations of the tested IDSes (see Section 2.2) and the applied evaluation methodologies.

Attack detection and reporting speed. This property is typically evaluated for distributed IDSes. Each node of a distributed IDS typically reports an ongoing attack to the rest of the nodes, or to a designated node, when it detects a malicious activity (see Table I). Thus, the attack detection speed of a distributed IDS is best evaluated by

Table IX. Comparison of Practices in Evaluating IDS Properties

Reference	IDS Type			IDS Properties								
	Distributed (D) Nondistributed (N)	Anomaly based (A) Misuse based (M) Hybrid (Hy)	Host based (H) Network based (N) Hybrid (Hy)	Attack detection accuracy	Attack coverage	Resistance to evasion techniques	Attack detection and reporting speed	CPU consumption	Memory consumption	Network consumption	Performance overhead	Workload processing capacity
Avritzer et al. [2010]	N	A	Hy	x								
Ahmed et al. [2011]	N	M	N	x								
Chung and Mok [2006]	N	A	H	x	x						x	
Chung et al. [2013]	D	M	N	x				x			x	x
Garfinkel and Rosenblum [2003]	N	Hy	H	x							x	
Hassanzadeh and Stoleru [2011]	D	M	N	x			x	x	x			
Jin et al. [2009] Jin et al. [2011]	N	Hy	Hy	x							x	x
Jou et al. [2002]	N	Hy	N	x	x							
Kannadiga and Zulkernine [2005]	D	A	N	x						x		
Sinha et al. [2006]	N	M	N	x					x			x
Laureano et al. [2007] Lombardi and De Pietro [2011] Reeves et al. [2012] Roesch [1999]	N	A	H		x						x	
Dehnert [2012] Raja et al. [2012] Riley et al. [2008]	N	A	H	x							x	
Sen et al. [2008]	D	A	N	x			x			x		
Srivastava et al. [2008]	N	A	H	x		x					x	
Sun et al. [2013]	D	A	N	x							x	
Zhang et al. [2008]	N	A	H		x						x	

measuring the time needed for the IDS to converge to a state in which all of its nodes, or the designated nodes, are notified of an ongoing attack (see the work of Hassanzadeh and Stoleru [2011] and Sen et al. [2008], who consider delays up to 3 seconds acceptable). The fast detection and reporting of an attack by an IDS node is important for the timely detection of coordinated attacks targeting multiple sites.

Resistance to evasion techniques. We observe that the evaluation of attack detection accuracy/attack coverage is prioritized over evaluation of resistance to evasion techniques. Sommer and Paxson [2010] confirm this trend, stating that resistance to evasion techniques is of limited importance from a practical perspective since most real-life attacks perform mass exploitation instead of targeting particular IDS flaws. However, a single successful IDS evasion attack poses the danger of a high-impact intrusion; therefore, it is a good practice to consider the resistance to evasion techniques in IDS evaluation studies.

We observed that Metasploit (see Section 2.1.3) is considered the optimal tool for executing IDS evasive attacks, which is required for evaluating resistance to evasion techniques. This is because Metasploit provides a freely available and regularly maintained attack execution environment supporting a wide range of IDS evasive techniques. By analyzing the decision-making processes of the IDSes proposed in the surveyed work for labeling an activity as benign or malicious, we observed that many IDSes are vulnerable to temporally crafted attacks (e.g., short-lived or multistep attacks executed by

delaying the execution of the attack steps, see Srivastava et al. [2008]). The execution of such attacks is supported by Metasploit.

Resource consumption related. These properties are typically evaluated for IDSes deployed in resource-constrained environments. An example is the evaluation of the resource consumption of a distributed IDS operating in wireless ad hoc networks, which enables the measurement of the power consumption of its nodes. This is important since the computing nodes in a wireless ad hoc network typically rely on a battery as a power source delivering a limited amount of power. Since the power consumption of software (i.e., of an IDS node) is difficult to measure, it can be best observed by using a model that estimates power consumption based on resource consumption measurements (see Hassanzadeh and Stoleru [2011]).

The network consumption in particular is often evaluated for distributed IDSes (see Sen et al. [2008]). The nodes of a typical distributed IDS exchange messages that contain information relevant for intrusion detection (see Table I). This may consume a significant amount of the network bandwidth of the environment where the IDS is deployed. For instance, Sen et al. [2008] consider the exchange of 120 messages over 160 seconds as a very high network consumption.

Performance overhead. This property is normally evaluated for host-based IDSes since they are known to cause performance degradation of the tasks running in the system where they are deployed. Performance overhead is evaluated by executing tasks twice, once with the tested IDS being inactive and once with it being active (see Laureano et al. [2007]). The differences between the measured task execution speeds reveal the imposed performance overhead. Performance overhead is normally evaluated using workloads in executable form generated by workload drivers (see Section 2.1.1). Workload drivers enable the straightforward generation of live customized workloads in a repeatable manner. In general, overheads under 10%, relative to the execution time of tasks measured when the tested IDS is inactive, are considered acceptable.

Workload processing capacity. This property is normally evaluated for network-based IDSes that monitor workloads at high rates. For instance, some studies consider workload rates as high as 1 million packets per second and report percentage of discarded packets of around 50%. Workload processing capacity is best evaluated using traces or workloads generated by workload drivers, as they allow for the generation of workloads at user-defined speeds. This enables the accurate measurement of the workload processing capacity of an IDS (e.g., of the particular network traffic speed such that a given network-based IDS under test does not discard packets, see Table VIII). Further, it is a good practice to evaluate the workload processing capacity of an IDS together with its resource consumption (e.g., Sinha et al. [2006] observe the resource consumption of an IDS for various workload intensities). This enables an IDS evaluator to observe how resource consumption scales as workload intensity increases.

Next we survey common approaches for evaluating the IDS properties presented in Table VIII. In addition, we demonstrate through case studies how these approaches and the discussed best practices have been applied in the surveyed work—that is, we round up the IDS evaluation design space by demonstrating the applicability of the different types of workloads and metrics with respect to their inherent characteristics (see Sections 2.1 and 2.2).

2.3.1. Measurement Methodology → Attack Detection-Related Properties. We start by considering the properties attack coverage, attack detection accuracy, and resistance to evasion techniques. Note that we do not discuss the IDS property attack detection and reporting speed. The approach for evaluating this property is almost identical to that

Table X. Attack Coverage of Snort

Targeted Vulnerability (CVE ID)	Platform	Detected
CVE-2011-3192	Apache	x
CVE-2010-1870	Apache Struts	✓
CVE-2012-0391	Apache Struts	x
CVE-2013-2251	Apache Struts	x
CVE-2013-2115/CVE-2013-1966	Apache Struts	✓
CVE-2009-0580	Apache Tomcat	x
CVE-2009-3843	Apache Tomcat	x
CVE-2010-2227	Apache Tomcat	x

✓, detected; x, not detected.

of attack detection accuracy with the only difference being in the used metrics (i.e., performance-related metrics that quantify time instead of security-related metrics).

Attack coverage. The attack coverage of an IDS is typically evaluated with the goal of measuring the ability of the IDS to detect various attacks targeted at the specific system/network environment that it protects [Mell et al. 2003]. Given that the attack coverage of an IDS is its attack detection accuracy in the presence of attacks without any background benign activity (see Table VIII), it is evaluated by using pure malicious workloads. The used pure malicious workloads should not have IDS evasive characteristics, as such workloads are used for evaluating resistance to evasion techniques, which is a separate property. Further, only basic metrics that do not contain measures of false alerts are used (e.g., true-positive rate). Note that an IDS might generate false alerts only in the presence of background benign activity.

Case study #4. We evaluate the attack coverage of the network-based IDS Snort [Roesch 1999], which uses misuse-based attack detection techniques. We consider a scenario where Snort is deployed in, and monitors the network traffic of, a server hosting Web applications using the Apache 2.2.16 server software, extended with the Tomcat 6.0.35 and Struts 1.3.10 frameworks. We tested Snort 2.9.22 using a database of signatures dated July 11, 2013. Given the architecture of the platform protected by Snort, attacks relevant for evaluating the attack coverage of Snort in this scenario are attacks targeting Apache, Tomcat, and Struts.

We used the Metasploit framework to generate pure malicious workloads (see Section 2.1.3). We use Metasploit's exploit database for the sake of convenience—Metasploit provides a readily available exploit database that contains attack scripts targeting the platforms Apache, Tomcat, and Struts. We executed eight attack scripts from a host that we refer to as the attacking host. To eliminate benign network traffic destined for the Web server, we used a firewall to isolate the Web server in a way such that it was reachable only for the attacking host. In Table X, we list the CVE identifiers of the vulnerabilities targeted by the executed attack scripts and the respective target platforms. In Table X, we present the results of the study. Snort detected two attacks, thus exhibiting a true-positive rate of 0.25.

Resistance to evasion techniques. The evaluation of the IDS property resistance to evasion techniques involves the execution of attacks using techniques such that there is a strong possibility that a tested IDS does not detect the attacks. The decision about what evasion techniques should be used in a given study is based on knowledge about the IDS's decision-making process for labeling an activity as benign or malicious. For instance, one may evade an IDS that matches string content of network packets to signatures by modifying the content of malicious packets in a way such that the IDS cannot match them to signatures—a technique known as string obfuscation.

Table XI. Resistance to Evasion Techniques of Snort

Evasion Technique	Targeted Vulnerability (CVE ID)	
	CVE-2010-1870	CVE-2013-2115/CVE-2013-1966
HTTP::uri_use_backslashes	✓	✓
HTTP::uri_fake_end	✓	✓
HTTP::pad_get_params	✓	x
HTTP::uri_fake_params_start	✓	✓
HTTP::uri_encode_mode (u-random; hex-random)	✓	x
HTTP::pad_method_uri_count	✓	✓
HTTP::method_random_valid	✓	x
HTTP::header_folding	✓	✓
HTTP::uri_full_url	✓	✓
HTTP::pad_post_params	✓	x
HTTP::uri_dir_fake_relative	✓	✓
HTTP::pad_uri_version_type (apache; tab)	✓	✓
HTTP::uri_dir_self_reference	✓	✓
HTTP::method_random_case	✓	✓

✓, detected; x, not detected.

As workloads for evaluating resistance to evasion techniques, one may use pure malicious or mixed workloads. Pure malicious workloads are used to determine which evasion techniques can be used for successfully evading an IDS, not taking into account benign activity as a factor. Mixed workloads are used in scenarios where benign activities are important for evading an IDS. For instance, a network-based IDS designed to detect multistep attacks (i.e., attacks that consist of several sequential attacks) may be constrained in the number of network packets that it can buffer for the purpose of attack tracking. Thus, one may evade the IDS by delaying the execution of the sequential attacks and generating benign network traffic between the executions.

The metrics used for quantifying resistance to evasion techniques are basic metrics that do not contain measures of false alerts (e.g., true-positive rate), as the goal is to measure the accuracy of an IDS in detecting only evasive attacks.

Case study #5. Following up on the results of Case study #4 (see Table X), we now investigate whether Snort is still able to detect the attacks targeting the vulnerabilities CVE-2010-1870 and CVE-2013-2115/CVE-2013-1966 when evasion techniques are applied. We used Metasploit to apply IDS evasive techniques to the considered attacks, as it enables the convenient execution of a wide range of IDS evasive attacks (see Section 2.3). Given that Snort matches string content of network packets to signatures [Roesch 1999], we used the string obfuscation evasion techniques provided by Metasploit. The applied IDS evasion techniques modify HTTP request strings stored in malicious network packets. In Table XI, we list the evasion techniques that we applied to each of the executed attacks (for details on the techniques, see Foster [2007]).

In Table XI, we present the detection score of Snort. It can be observed that Snort detected most of the executed evasive attacks and failed to detect the attack targeting the vulnerability CVE-2013-2115/CVE-2013-1966 when the evasive techniques HTTP::pad_get_params, HTTP::uri_encode_mode, HTTP::method_random_valid, and HTTP::pad_post_params were applied. Snort detected 24 out of 28 attack executions, thus exhibiting a true positive rate of 0.85.

Attack detection accuracy. By evaluation of the attack detection accuracy of an IDS, we mean evaluation of the accuracy of the IDS in detecting attacks mixed with benign activities (see Table VIII). Attack detection accuracy is quantified using security-related metrics that include measures of false alerts (see Section 2.2.2). This is important, as an

Table XII. Attack Detection Accuracy of Snort: Basic Metrics (`seconds=120`)

Configuration	Metrics			
	α	$1-\beta$	PPV	NPV
`count=6`	0.0008	0.333	0.9788	0.9310
`count=5`	0.0011	0.416	0.9768	0.9390
`count=4`	0.0013	0.5	0.9771	0.9473
`count=3`	0.0017	0.624	0.9761	0.9598
`count=2`	0.0024	0.833	0.9747	0.9817
Default configuration	0.0026	0.958	0.9762	0.9953

IDS under test might mislabel some benign activities as malicious. We demonstrated quantification of attack detection accuracy in Section 2.2.2.

Case study #6. We evaluate the attack detection accuracy of Snort 2.9.22 using a database of signatures dated July 11, 2013. We used the DARPA datasets as mixed workloads (see Section 2.1.5); we replayed a trace file from the 1998 DARPA datasets with tcpreplay.[33,34] To calculate values of security-related metrics that contain measures of false alerts, we used the ground truth files provided by the Lincoln Laboratory at MIT.[35] These files contain information useful for uniquely identifying each attack recorded in the trace file that we replayed, such as time of execution. We compared the ground truth information with the alerts produced by Snort to calculate the number of detected and missed attacks as well as the number of false alerts. This is required for calculating values of security-related metrics.

With its default configuration enabled, Snort detected almost all attacks. However, Snort also issued false alerts—Snort's rule with ID 1417 led to mislabeling many benign Simple Network Management Protocol (SNMP) packets as malicious. Therefore, we examined the influence of the configuration parameter `threshold` on the attack detection accuracy of Snort. The parameter `threshold` is used for reducing the number of false alerts by suppressing signatures that often mislabel benign activities as malicious. A signature may be suppressed such that it is configured to not generate an alert for a specific number of times (specified with the keyword `count`) during a given time interval (specified with the keyword `seconds`).

The measurement of the attack detection accuracy of an IDS for different configurations of the IDS enables the identification of an optimal operating point (see Section 2.2). We measured the attack detection accuracy of Snort for five different configurations where the signature with ID 1417 was suppressed by setting the value of `count` to 2, 3, 4, 5, and 6, whereas `seconds` was set to 120. We also measured the attack detection accuracy of Snort when its default configuration was used, according to which the signature with ID 1417 is not suppressed.

In Table XII, we present values of basic security-related metrics (see Section 2.2.1). One can observe that the values of α and $1-\beta$ decrease as the value of `count` increases—increasing the value of `count` leads to decreasing the number of generated false alerts, which is manifested by the decreasing values of α. However, increasing the value of `count` also leads to worsening of the true-positive rate $1-\beta$. This is a typical trade-off situation between the true- and the false-positive rate of an IDS. Next, we calculate values of composite security-related metrics.

[33]The trace file that we replayed is available at http://www.ll.mit.edu/ideval/data/1998/testing/week1/monday/tcpdump.gz.

[34]We used tcpreplay in all IDS evaluation studies presented in this article where we replayed traces.

[35]We discussed the importance of ground truth information in Section 2.1.5. The ground truth files that we used are available at http://www.ll.mit.edu/ideval/data/1998/Truth_Week_1.llist.tar.gz.

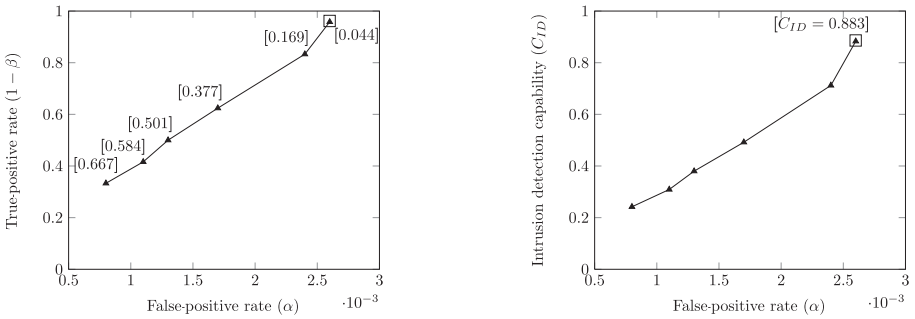


Fig. 8. Attack detection accuracy of Snort: composite metrics. ROC curve and estimated costs (a) and C_{ID} curve (b) (□ marks an optimal operating point).

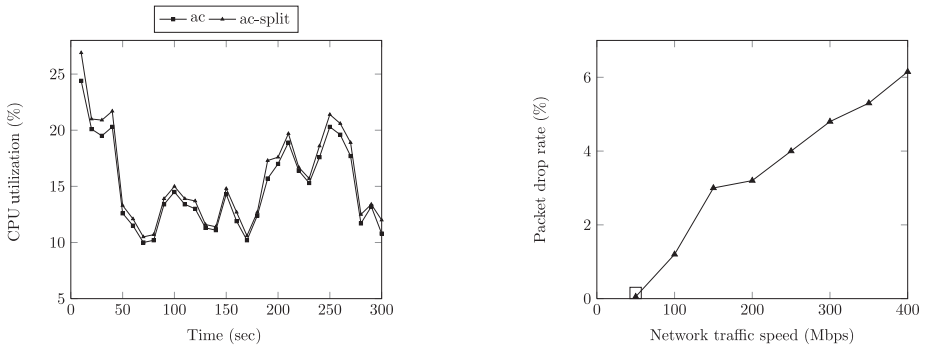
In Figure 8(a), we depict an ROC curve providing an overview of the trade-off mentioned earlier. In addition, in Figure 8(a), we annotate the depicted operating points with the associated estimated costs C_{exp}. The values of the estimated costs are values of the expected cost metric (see Section 2.2.2). To calculate values of the expected cost metric, we assumed a cost ratio C of 10. The base rate B is 0.10.[36] Once we calculated the cost associated with each operating point, we were able to identify the optimal operating point (i.e., the operating point that has the lowest C_{exp} associated with it)—(0.0026, 0.958). Based on our findings, we conclude that Snort operates optimally in terms of cost when configured with its default settings.

In Figure 8(b), we depict the values of the intrusion detection capability metric C_{ID} (see Section 2.2.2) for the considered operating points. The C_{ID} curve depicted in Figure 8(b) enables the identification of the optimal operating point of Snort in terms of intrusion detection capability (i.e., the point that marks the highest C_{ID})—(0.0026, 0.958), which marks a C_{ID} of 0.883. We conclude that Snort operates optimally in terms of intrusion detection capability when configured with its default settings.

2.3.2. Measurement Methodology → Resource Consumption–Related Properties. The resource consumption–related properties are evaluated using pure benign workloads that (1) are considered as regular for the environment where the IDS under test operates or (2) exhibit extreme behavior in terms of their intensity. The first are used for evaluating the resource consumption of an IDS under regular operating conditions, whereas the latter for evaluating the resource consumption of an IDS that processes high-rate workloads. The metrics used for quantifying IDS resource consumption are performance-related metrics that quantify resource utilization (e.g., CPU utilization).

There are mainly two approaches for evaluating IDS resource consumption: black-box and white-box testing. Black-box testing assumes the measuring of the resource consumption of an IDS as resource consumption of the IDS process that is active in the system where the IDS is deployed. This approach is commonly adopted in IDS evaluation experiments; however, it does not provide insight into the resource demands of the individual IDS components. Such insight is normally important for optimizing the IDS configuration. White-box IDS testing usually assumes the use of an IDS model that decomposes the IDS into its components and estimates their individual resource consumption. Dreger et al. [2008] construct an IDS model shown to estimate CPU and memory consumption of an IDS with a relative error of 3.5%. An alternative to

[36]This is an approximate estimation of the actual base rate. Accurate base rate is not relevant for the purpose of this case study.



(a) (b)

Fig. 9. CPU consumption of Snort (a) and packet drop rate of Snort (b) (□ marks the data point whose x value is the network traffic speed that corresponds to the maximum workload processing rate of Snort such that there are no discarded workloads).

modeling is code instrumentation. However, this approach is unfeasible in case the tested IDS is not open source.

Case study #7. We demonstrate the common black-box IDS resource consumption testing. We measure the CPU consumption of Snort 2.9.22 in two scenarios where we configured Snort to use the pattern matcher ac and ac-split, respectively.[37] Although ac-split is known to consume less memory resources than ac,[38] it may increase the CPU consumption of Snort. The goal of this study is to determine whether the use of ac-split over ac is advisable by taking CPU consumption into account. To this end, we deployed Snort in a host with a dual-core CPU, each core operating at 2GHz, 3GB of memory, and a Debian OS. To generate pure benign workloads, we replayed a trace file from the LBNL/ICSI trace repository (see Table V) at the speed of 6Mbps.[39]

In Figure 9(a), we depict the CPU utilization of Snort, which we measured with the tool top.[40] The top tool enables the measurement of the CPU utilization of any active system process, and thus it is an example of a typical tool used for black-box IDS resource consumption testing. We used top to sample the CPU utilization of the Snort process at every 10 seconds for 5 minutes. We repeated the measurements 30 times and averaged the results. In Figure 9(a), one may observe that in the considered scenario, the use of the ac-split pattern matcher does not cause significant increase in the CPU consumption of Snort when compared to the use of ac.

2.3.3. Measurement Methodology → Workload Processing Capacity. The IDS workload processing capacity is evaluated using pure benign workloads that exhibit extreme behavior in terms of intensity. The goal is to observe the rate of arrival of workloads to an IDS under test for processing in relation to the amount of workloads that the IDS discards (see Table VIII). The identification of a maximum workload processing rate of an IDS such that there are no discarded workloads may also be considered. Similar to

[37]The use of a pattern matcher helps to speed up the process of evaluating network packets against Snort signatures by reducing the number of signatures against which each packet is evaluated. When a packet is intercepted by Snort, a pattern matcher evaluates the content of the packet against a set of patterns used to group multiple signatures. If a match is found, the packet is then evaluated only against the signatures that belong to the respective group of rules.
[38]http://manual.snort.org/node16.html.
[39]ftp://ftp.bro-ids.org/enterprise-traces/hdr-traces05/lbl-internal.20041004-1313.port003.dump.anon.
[40]http://linux.die.net/man/1/top.

evaluating resource consumption, an IDS capacity can be evaluated using a black-box or a white-box testing approach.

Wheres black-box capacity testing does not pose significant challenges, white-box testing is more challenging given that multiple evaluation tests that target specific IDS components involved in processing workloads need to be defined and performed. This requires in-depth knowledge on the design of the IDS under test. White-box testing enables the identification of the particular component of an IDS that is a performance bottleneck. Hall and Wiley [2002] propose an approach for white-box IDS capacity testing. They define a methodology consisting of individual tests for measuring the capacity of the components of workload processing mechanisms of network-based IDSes (i.e., the packet flow and capture, the state tracking, and the alert reporting component).

Case study #8. We demonstrate the common black-box IDS capacity testing. We measure the rate of dropped packets by Snort 2.9.22 when it monitors network traffic at speeds in the range of 50Mbps to 400Mbps, in steps of 50Mbps. We deployed Snort in a host with a dual-core CPU, each core operating at the speed of 2GHz, 3GB of memory, and a Debian OS. We generated network traffic by replaying a trace file from the LBNL/ICSI trace repository.[38] We used network workloads in trace form, as the use of traces enables the straightforward generation of network traffic at customized speeds in a repeatable manner (see Section 2.3).

To measure the packet drop rate of Snort, we first started the Snort process, then replayed the network trace file, and finally we stopped the Snort process. When a Snort process is stopped, it displays a set of statistics measured with Snort's built-in performance profiling tools, which includes the packet drop rate averaged over the lifetime of the process. We repeated the measurements 30 times and averaged the results. In Figure 9(b), we depict the packet drop rate of Snort in relation to the speed of the monitored network traffic. Starting at network traffic speed of 50Mbps, the packet drop rate increases almost linearly as the network traffic speed increases. Note that the drop rate of Snort is 0 when it monitors network traffic at the speed of, as well as less than (not depicted in Figure 9(b)), 50Mbps—that is, 50Mbps is the maximum workload processing rate of Snort such that there are no dropped packets.

2.3.4. Measurement Methodology → Performance Overhead. The IDS performance overhead is evaluated using pure benign workloads in executable form that do not exhibit extreme behavior in terms of intensity but are extreme in terms of the exercised set of hardware resources. Depending on the type of workloads that the IDS under test monitors (e.g., network packets, file I/O operations), an overhead evaluation experiment may consist of five independent experiments, each with a workload that is CPU intensive, memory intensive, file I/O intensive, network intensive, or mixed. We provided an overview of such tasks in Sections 2.1.1 and 2.1.2.

The execution of the tasks mentioned earlier is performed twice, once with the IDS under test being inactive and once with it being active. The differences between the measured task execution speeds reveal the performance overhead imposed by the IDS.

Case study #9. We evaluate the performance overhead of the host-based IDS OSSEC 2.8.1. OSSEC performs file integrity monitoring at real time by intercepting and analyzing file I/O operations (e.g., writing to a file). We measure the overhead imposed by OSSEC on file operations reading or writing data of various record sizes.

We deployed OSSEC in a host with an ext3 filesystem. Given that OSSEC monitors file I/O operations, we generated pure benign workloads that are file I/O intensive. We used the workload driver iozone (see Section 2.1.1) to generate workloads consisting

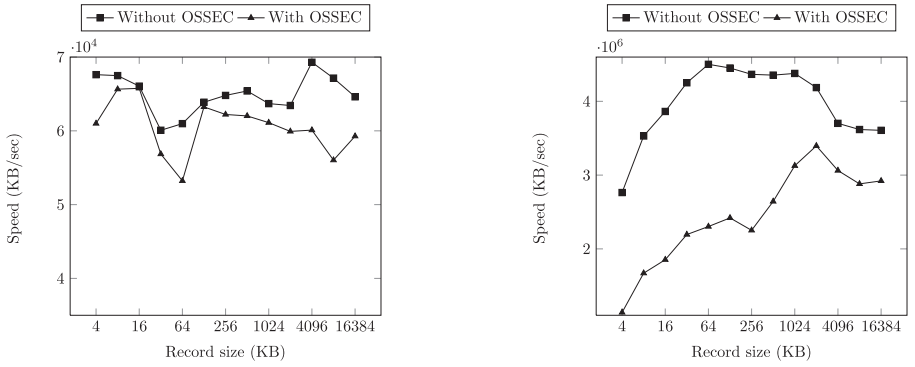


Fig. 10. Performance overhead imposed by OSSEC on file write (a) and read (b) operations.

of file operations that write and read data of record sizes in the range of 4KB to 16MB. For the purpose of this study, we use iozone because it enables the fine-granular customization of file I/O-intensive workloads (see Section 2.3).

In Figure 10(a) and (b), we depict the execution speeds of the generated file write and read operations. We measured the execution speeds twice, once with OSSEC being inactive and once with it being active. We repeated the measurements 30 times and averaged the results. The difference in the speeds shown in the figure reveal the overhead imposed by OSSEC. For instance, the execution speed of the operation reading data of a record size 4KB is 2763641KB/sec when OSSEC is not running and 1138420KB/sec when OSSEC is running.

3. CONCLUDING REMARKS

3.1. Future Topics in IDS Evaluation

We now discuss relevant future topics in the area of IDS evaluation, which includes open issues and challenges that apply to evaluating novel IDSes.

High-speed IDSes. Due to the ever-increasing amount of network traffic, much attention has been given to designing high-speed network-based IDSes (e.g., see Le and Prasanna [2013]). These IDSes use workload processing components specifically designed for efficiently processing high-rate workloads (e.g., pattern matchers, see Section 2.3.2). When it comes to designing high-speed network-based IDSes and evaluating their workload processing capacity, current work is biased in favor of designing novel pattern matchers and evaluating their impact on IDS capacity [Lin and Lee 2013]. Lin and Lee profiled the source code of the IDSes Snort and Bro to identify performance bottlenecks.[41] They discovered that workload processing components used for detecting current IDS evasive attacks (e.g., packet reassembly mechanisms) are often bigger performance bottlenecks than pattern matchers. This indicates that novel white-box IDS evaluation methods and tests targeting the previously mentioned components should be designed to better understand how these components affect the workload processing capacity of IDSes. This is challenging, as it requires in-depth knowledge on the designs of tested IDSes. Further, in contrast to current practice in IDS capacity testing (see

[41]http://www.bro.org/.

Section 2.3.3), the needed tests would involve the generation of workloads that contain modern IDS evasive attacks to exercise the targeted workload processing components.

IDSes for virtualized environments. IDSes for virtualized environments, where a virtual machine monitor (VMM) hosts multiple virtual machines (VMs), are becoming common with the growing proliferation of virtualized data centers. Such IDSes (i.e., VMM-based IDSes) are deployed in the virtualization layer, usually with components inside the VMM and in a designated OS (see Jin et al. [2011]). This enables them to monitor the network and/or host activities of all colocated VMs at the same time. Next we list key issues related to evaluating VMM-based IDSes.

(1) Inaccurate metrics. Existing IDS evaluation metrics (see Section 2.2) are defined with respect to a fixed set of hardware resources available to the IDS [Hall and Wiley 2002]. However, many virtualized environments—cloud environments in particular—have elastic properties. In other words, resources might be provisioned and used by VMs on demand (e.g., the Xen VMM allows for hot plugging virtual CPUs and memory on VMs). This implies that the hardware resources available to an IDS running in a virtualized environment may change over time, which may influence IDS performance. Thus, metrics calculated as a function of an elasticity metric are needed for the accurate measurement of the performance of such an IDS. An elasticity metric would quantify the resource provisioning time and the amount of provisioned hardware resources as the system load changes. The construction of an elasticity metric poses many challenges, such as determining which specific system features the metric should reflect (e.g., scalability, precision/speed of resource provisioning).

(2) Challenging definition of a baseline workload profile. A VMM-based IDS monitors the activities of multiple VMs at the same time. In modern data centers, the number of VMs colocated on a VMM can vary due to VM migration—a new VM may arrive or an existing one may be removed from a VMM due to, for example, load balancing. Thus, one can expect that in a real-world setting, the VMs' activities monitored by a VMM-based IDS would change drastically over time. Given this diversity, we argue that it is challenging to define a baseline workload profile that is representative for "normal" workload for a given virtualized environment, which is needed for IDS training (see Table I). A solution would be the development of a model allowing to estimate the characteristics of the VMs' activities monitored by a VMM-based IDS for different VM deployment scenarios and to identify baseline workload profiles.

(3) Challenging generation of workload traces. A typical VMM-based IDS combines hardware-level information about VMs (e.g., CPU register values) with high-level domain-specific knowledge (e.g., kernel data structures, see Srivastava et al. [2008]) to detect attacks. As a result, we argue that it is a significant challenge to capture workload trace files that contain the exact information required by a VMM-based IDS. Given the complexity of recording procedures, it is expected that replay procedures would also be challenging. Thus, a systematic classification structuring the various types of information used by VMM-based IDSes would be an important contribution. This can be used as a basis to design configurable recording and replay mechanisms.

IDSes for detecting APTs (advanced persistent threats)/zero-day attacks. The detection of APTs is becoming an increasingly important topic due to the escalating number of incidents and severity of APTs. An APT is a carefully executed attack with the objective of intruding a given domain and remaining undetected for an extended period of time. The execution of an APT consists of multiple steps, such as executing a zero-day attack and deploying malware to establish a command and control channel. Therefore, the detection of APTs is performed by composite IDSes consisting of mechanisms for detecting a variety of malicious activities in a coordinated manner, such as host-, anomaly-based IDSes for detecting zero-day attacks (see Table I) and

network-, misuse-based IDSes for discovering command and control channels. An example of such an IDS is Deep Discovery by TrendMicro.[42]

The rigorous and realistic evaluation of IDSes designed to detect APTs/zero-day attacks is an issue that has not been thoroughly addressed so far. Since mechanisms used for detecting APTs are anomaly- or misuse-based IDSes, their attack detection accuracies can be quantified using the discussed security-related metrics (see Section 2.2). However, the generation of IDS evaluation workloads that contain APTs/zero-day attacks is an open issue. One of the first methodologies for evaluating IDSes designed to detect APTs/zero-day attacks has been published only recently by NSS Labs (for more information on NSS Labs, see Appendix A).[43] This methodology involves the use of high-interaction honeypots for capturing zero-day attacks (see Section 2.1.6) that can be executed as part of IDS evaluation workloads.

Given the increasing demand for approaches to evaluate IDSes designed to detect APTs/zero-day attacks, the focus of the IDS evaluation community may shift in the near future toward addressing related issues. For instance, of great importance is the design and evaluation of novel honeypots that capture zero-day attacks such that the attacks can be used as IDS evaluation workloads with minimal investment of time. An example is a high-interaction honeypot that specializes in constructing ground truth—that is, a honeypot that can distinguish different attack sessions to assign a unique identifier to each session, actual zero-day attacks from other activities (e.g., reconnaissance) to record the exact time of start and end of the attacks, and the like. Current honeypots do not support the automatic construction of ground truth, which at this time is performed manually by a human expert and therefore is time consuming and error prone (see Section 2.1.6).

The development of approaches for the generation of representative command and control traffic is also important. With the proliferation of cloud environments, attackers have started using legitimate cloud services (e.g., Google Apps) as command and control channels to evade IDSes—communication with such services is normally part of regular production traffic and is therefore often considered trusted and not a priority for analysis.[44] The development of activity models involving the use of popular cloud services, which can be used as a basis for the generation of command and control traffic to record traces (i.e., IDS evaluation workloads in trace form, see Section 2.1.6) would enable the rigorous evaluation of IDSes designed to detect APTs.

3.2. Summary—Lessons Learned

We now provide guidelines for planning IDS evaluation studies based on what we observed when surveying relevant work in the field. We structure our discussion by focusing on three key and interrelated points in the planning of every evaluation study: *goals* of a study, existing *approaches* to realize the set goals (i.e., approaches for generating workloads and for measuring IDS performance—metrics), and *requirements* that need to be met.

Goals. Under goals of an IDS evaluation study, we understand the IDS properties that one aims to evaluate. The selection of IDS properties for evaluation is normally done by considering the design objectives and the target deployment environment of the IDS under test. In Table XIII, we present the most commonly considered IDS properties in evaluation studies for various IDS types. We also provide a summarizing

[42]http://www.trendmicro.com/us/enterprise/security-risk-management/deep-discovery/.
[43]https://www.nsslabs.com/reports/breach-detection-systems-test-methodology-15.
[44]http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-detecting-apt-activity-with-network-traffic-analysis.pdf.

Table XIII. Summarizing Overview of Common Trends, Recommendations, and Key Best Practices

IDS Property	
Attack detection accuracy Attack coverage	These properties are evaluated for IDSes of all types. • The dated DARPA and KDD-99 Cup datasets represent at this time standard workloads for comparing novel anomaly-based IDSes with their past counterparts. • For the sake of representativeness, evaluate an IDS using not the DARPA or the KDD-99 Cup dataset but workloads that contain current attacks. • Attack detection rates of current IDSes vary greatly—that is, between 8% and 97%, measures that depend on the configurations of the tested IDSes and the applied evaluation methodologies.
Attack detection and reporting speed	This property is normally evaluated for distributed IDSes—it is best evaluated by measuring the time needed for the IDS to converge to a state in which all of its nodes, or the designated nodes, are notified of an ongoing attack. • Attack detection delays up to 3 seconds are considered acceptable.
Resistance to evasion techniques	This property is often not evaluated, as it is considered of limited practical importance. • Consider evaluating this property since a single successful IDS evasion attack poses the danger of a high-impact intrusion. • Metasploit is deemed the optimal tool for executing IDS evasive attacks, which is required for evaluating this property. • Many current IDSes are vulnerable to temporally crafted attacks.
Resource consumption related	These properties are typically evaluated for IDSes deployed in resource-constrained environments. • Network consumption in particular is often evaluated for distributed IDSes. • The resource consumption of a distributed IDS operating in wireless ad hoc networks is typically evaluated to measure the power consumption of its nodes—this is best performed by using a model that estimates power consumption based on resource consumption measurements.
Performance overhead	This property is normally evaluated for host-based IDSes. • Performance overhead is evaluated by executing tasks twice, once with the tested IDS being inactive and once with it being active. • This property is normally evaluated using workloads in executable form generated by workload drivers—workload drivers enable the straightforward generation of live customized workloads in a repeatable manner. • Overheads under 10%, relative to the execution time of tasks measured when the tested IDS is inactive, are generally considered acceptable.
Workload processing capacity	This property is normally evaluated for network-based IDSes that monitor high-rate workloads. • This property is best evaluated using traces or workload drivers, as they allow for the generation of workloads at user-defined speeds. • Evaluate the capacity of an IDS together with its resource consumption—this enables one to observe how resource consumption scales as workload intensity increases.

overview of common trends, recommendations, and key best practices in evaluating IDS properties for different types of IDSes. Finally, we present observed quantitative values (e.g., generally acceptable performance overheads) and relevant observations that may serve as reference points for designing and evaluating future IDSes. For more details, we refer the reader to Section 2.3. For demonstrations on executing IDS tests to evaluate the commonly considered IDS properties, we refer the reader to Sections 2.3.1 through 2.3.4.

Requirements. Before discussing approaches for evaluating IDSes, we identify and systematize the requirements that have to be met for the different approaches:

—*Availability of required resources*: (1) *Financial* resources (e.g., the costs of building a testbed may be significant, see Section 2.1.6); (2) *time* resources (e.g., the manual assembly of an exploit database may be time consuming, see Section 2.1.3); (3) *manpower* (e.g., the amount of available human resources is important for labeling traces in a time-efficient manner, see Section 2.1.5).

Table XIV. Guidelines for Planning IDS Evaluation Studies: Workloads and Metrics

Workloads
Pure benign/mixed/pure malicious →Executable form
Key information: ✓ Generated workloads closely resemble real workloads ○ Multiple evaluation runs are required to ensure statistical significance of IDS behavior ○ Generated malicious workloads require specific victim environments – Replicating experiments when using malicious workloads is challenging due to, for example, crashes of victim environments
(Pure benign)→Workload drivers (Section 2.1.1)
Key information: ✓ Generated workloads can be customized in terms of their temporal and intensity characteristics – Generated workloads do not resemble real-life workloads as closely as those manually generated *Tools:* SPEC CPU2000, iozone, Postmark, httpbench, dkftpbench, ApacheBench, UnixBench (see Table II) *References*: Allalouf et al. [2010], Patil et al. [2004], Dunlap et al. [2002], Garfinkel and Rosenblum [2003], and Jin et al. [2011]
(Pure benign) →Manual generation (Section 2.1.2)
Key information: ✓ Generated workloads are similar to those observed by an IDS during regular system operation ○ Suitable for generation of workload traces capturing realistic workloads executed in a recording testbed – Does not support workload customization *Key requirements*: time resources,[a] manpower[a] *Practices*: File encoding and tracing, file conversion, copying of large files, kernel compilation (see Table II) *References*: Dunlap et al. [2002], Srivastava et al. [2008], Lombardi and Di Pietro [2011], Allalouf et al. [2010]
(Pure malicious) →Exploit database →Manual assembly (Section 2.1.3)
Key information: ✓ Generated workloads are realistic and representative ○ Attack scripts need to be collected and adapted, and a victim environment needs to be set up – Generated workloads are normally of a limited size due to lack of manpower and/or time resources – Publicly available attack scripts normally do not have IDS evasive characteristics *Key requirements*: Time resources, manpower, knowledge about the architecture and inner working mechanisms of the IDS under test[b] *Datasets*: Inj3ct0r, Exploit database, Packetstorm, SecuriTeam, Securityfocus (see Table III) *References*: Puketza et al. [1997], Lombardi and Di Pietro [2011], Debar et al. [1998], and Reeves et al. [2012]
(Pure malicious) →Exploit database →Readily available exploit database (Section 2.1.3)
Key information: ✓ No time spent on collection and adaptation of attack scripts – Some databases have critical limitations (e.g., contain only exploits that can be executed from remote sources) *Key requirements*: Knowledge about the architecture and inner working mechanisms of the IDS under test[b] *Tools*: Metasploit, Nikto, w3af, Nessus *References*: Gad El Rab [2008]
(Pure malicious) →Vulnerability and attack injection (Section 2.1.4)
Key information: ✓ Useful when collection of attack scripts is unfeasible ✓ Remotely and locally exploitable codes can be injected – Not an extensively investigated approach *Key requirements*: Knowledge about the architecture and inner working mechanisms of the IDS under test *Tools*: Vulnerability and Attack Injector Tool (VAIT) [Fonseca et al. 2014] *References*: Fonseca et al. [2014]

(Continued)

Table XIV. Continued

Pure benign/mixed/pure malicious → Trace form
Key information: ✓ A single evaluation run is required to ensure statistical significance of IDS behavior ✓ Replicating evaluation experiments is a straightforward task – Generated workloads closely resemble real workloads only for a short period of time ○ Generated malicious workloads do not require specific victim environments
→ Trace acquisition → Real-world production traces (Section 2.1.5)
Key information: ✓ Highly realistic and possibly large-scale workloads – Normally very difficult to obtain due to privacy concerns and legal issues – Normally anonymized with the risk that intrusion detection relevant data is removed – Normally not labeled, which makes the construction of ground truth challenging *Key requirements*: Time resources, manpower, access to confidential data
→ Trace acquisition → Publicly available traces (Section 2.1.5)
Key information: ✓Can be obtained without any legal constraints ✓Normally labeled – Many contain errors (e.g., unrealistic distributions of attacks) – May be outdated due to limited shelf life of attacks ○ Claims on generalizability of results from IDS tests based on publicly available traces can often be questioned *Key requirements*: Knowledge about the characteristics of the employed workloads *Datasets*: CAIDA, DEFCON, DARPA/KDD-99, ITA, LBNL/ICSI, MAWILab (see Table V) *References*: Alserhani et al. [2010], Yu and Dasgupta [2011], Raja et al. [2012], and Durst et al. [1999]
→Trace generation → Testbed environment (Section 2.1.6)
Key information: ✓ Issues related to trace acquisition not present – The costs of building a testbed that scales to realistic production environments may be high ○ Methods used for trace generation are critical, as they may produce faulty or simplistic workloads *Key requirements*: Financial resources, time resources,[c] manpower[c] *References*: Cunningham et al. [1999], and Shiravi et al. [2012]
→ Trace generation → Honeypots (Section 2.1.6)
Key information: ✓ Enable the generation of traces that contain representative and possibly zero-day attacks – The outcome of a trace generation campaign is uncertain, as it cannot be planned in advance and controlled ○ The attack labeling feasibility depends on the level of interaction of used honeypot(s) *Key requirements*: Time resources,[d] manpower[d] *Tools*: honeyd, nepenthes, mwcollectd, honeytrap, HoneyC, Monkey-Spider, honeybrid, HoneySpider, Sebek, Argos, CaptureHPC, HoneyClient, HoneyMonkey (see Figure 3) *References*: Dumitras and Shou [2011]
Metrics
Security related → Basic (Section 2.2.1)
Key information: ○ Quantify individual attack properties ○ Need to be analyzed together to accurately quantify the attack detection accuracy of an IDS ○ Ground truth information is a requirement for calculating the basic security-related metrics *Key requirements*: Knowledge about the characteristics of the employed workloads *Metrics*: True-positive rate (1-β), false-positive rate (α), false-negative rate (β), true-negative rate (1-α), positive predictive value (PPV), negative predictive value (NPV) (see Table VI) *References*: Mell et al. [2003], Raja et al. [2012], Srivastava et al. [2008], Reeves et al. [2012], and Jou et al. [2000]

(Continued)

Table XIV. Continued

Security related → Composite
Key information:
○ Used for analyzing relationships between the basic security-related metrics
○ Useful for identifying (an) optimal IDS operating point(s) for evaluating a single or comparing multiple IDSes
→ ROC curve (Section 2.2.2)
Key information:
○ The first metric of choice for identifying optimal IDS operating points
○ An open issue is how to determine a proper unit and measurement granularity
– ROC curve analysis may be misleading when used for comparing multiple IDSes
Key requirements: Knowledge about the architecture and inner working mechanisms of the IDS under test,[e] knowledge about the characteristics of the employed workloads
References: Raja et al. [2012] and Durst et al. [1999]
→ Cost based/Information theoretic (Section 2.2.2)
Key information:
✓ Enable the accurate and straightforward comparison of multiple IDSes
✓ Express the impact of the rate of occurrence of intrusion events (i.e., base rate)
○ (Depend/Do not depend) on subjective measures (e.g., cost) and are (unsuitable/suitable) for objective comparisons of IDSes
Key requirements: Knowledge about the characteristics of the employed workloads, knowledge about the implications of different behavior exhibited by the IDS under test (applies for cost-based metrics only)
Metrics: Cost-based—expected cost (see Table VI), relative expected cost [Meng 2012]; information theoretic—intrusion detection capability (see Table VI), false alarm reduction capability [Meng and Kwok 2013]
References: Cost-based metrics: Gaffney and Ulvila [2001], Meng [2012]; information-theoretic metrics: Gu et al. [2006] and Meng and Kwok [2013]

[a]When large-scale workloads are generated (e.g., for recording in a testbed).
[b]In case the IDS property "resistance to evasion techniques" is evaluated.
[c]In particular, when recorded workloads are generated manually.
[d]Depends on the attack labeling feasibility of generated traces.
[e]In particular, knowledge about employed workload processing mechanisms (e.g., units of analysis) to avoid misleading results when comparing IDSes.
✓, advantage; −, disadvantage; ○, neutral.

—*Access to confidential data*: Organizations are often unwilling to share operational traces because of privacy concerns and legal issues (see Section 2.1.5).
—*Availability of knowledge*: (1) The *architecture* and *inner working mechanisms* of the IDS under test (e.g., see the discussions on evaluating resistance to evasion techniques in Section 2.3.1); (2) the *characteristics* of the employed workloads (e.g., information about the attacks used as workloads must be known to calculate any security-related metric, see Section 2.2); (3) the *implications* of different behavior exhibited by the tested IDS (e.g., the cost of missing an attack must be known in order to calculate the expected cost metric, see Section 2.2.2).

We observed that the requirements mentioned earlier often cannot be fully satisfied given the big investment of resources that typically needs to be made. We observed that sacrifices are often made in (1) the *scale* of the employed workloads, an example of which is the typical low number of attack scripts used as workloads (see Section 2.1.3), and (2) the *number* of considered IDS properties (i.e., researchers tend to evaluate only a few IDS properties, see Table IX). We suggest trade-offs made between the quality of evaluations and the invested resources to be clearly stated when reporting results from IDS evaluation studies so that the results can be interpreted in a fair and accurate manner.

Approaches. In Table XIV, we systematize relevant information that facilitates the process of selecting approaches for generating workloads for evaluating IDSes and for

measuring IDS performance (i.e., metrics). In-depth information on what is presented in Table XIV can be found in Sections 2.1 and 2.2, whose subsections correspond to the approaches listed in the table. For each approach, where applicable, we provide *key information* (i.e., advantages, disadvantages, and/or relevant facts); *key requirements* that need to be satisfied, systematized as earlier; example common *practices* or commonly used *tools*, *datasets*, or *metrics*; and selected *references* to relevant publications where studies that use the approach can be found.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

REFERENCES

Abdulbasit Ahmed, Alexei Lisitsa, and Clare Dixon. 2011. A misuse-based network intrusion detection system using temporal logic and stream processing. In *Proceedings of the 5th International Conference on Network and System Security (NSS'11)*. 1–8.

Miriam Allalouf, Muli Ben-Yehuda, Julian Satran, and Itai Segall. 2010. Block storage listener for detecting file-level intrusions. In *Proceedings of the 26th Symposium on Mass Storage Systems and Technologies (MSST'10)*. 1–12.

Faeiz Alserhani, Monis Akhlaq, Irfan U. Awan, Andrea J. Cullen, and Pravin Mirchandani. 2010. MARS: Multi-stage attack recognition system. In *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA'10)*. IEEE, Los Alamitos, CA, 753–759.

Alberto Avritzer, Rajanikanth Tanikella, Kiran James, Robert G. Cole, and Elaine J. Weyuker. 2010. Monitoring for security intrusion using performance signatures. In *Proceedings of the 1st Joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW'10)*. 93–104.

Stefan Axelsson. 2000. The base-rate fallacy and its implications for the difficulty of intrusion detection. *ACM Transactions on Information and Systems Security* 3, 3, 186–205.

Saketh Bharadwaja, Weiqing Sun, Mohammed Niamat, and Fangyang Shen. 2011. Collabra: A Xen hypervisor based collaborative intrusion detection system. In *Proceedings of the 8th International Conference on Information Technology: New Generations (ITNG'11)*. IEEE, Los Alamitos, CA, 695–700.

Tsung-Huan Cheng, Ying-Dar Lin, Yuan-Cheng Lai, and Po-Ching Lin. 2012. Evasion techniques: Sneaking through your intrusion detection/prevention systems. *IEEE Communications Surveys and Tutorials* 14, 4, 1011–1020.

Chien-Yi Chiu, Yuh-Jye Lee, Chien-Chung Chang, Wen-Yang Luo, and Hsiu-Chuan Huang. 2010. Semi-supervised learning for false alarm reduction. In *Advances in Data Mining: Applications and Theoretical Aspects*. Lecture Notes in Computer Science, Vol. 6171. Springer, Berlin, 595–605.

Chun-Jen Chung, Khatkar Pankaj, Tianyi Xing, Jeongkeun Lee, and Dijiang Huang. 2013. NICE: Network intrusion detection and countermeasure selection in virtual network systems. *IEEE Transactions on Dependable and Secure Computing* 10, 4, 198–211.

Simon P. Chung and Aloysius K. Mok. 2006. On random-inspection-based intrusion detection. In *Proceedings of the 8th International Conference on Recent Advances in Intrusion Detection (RAID'06)*. 165–184.

Scott E. Coull, Charles V. Wright, Fabian Monrose, Michael P. Collins, and Michael K. Reiter. 2007. Playing devils advocate: Inferring sensitive information from anonymized network traces. In *Proceedings of the Network and Distributed System Security Symposium*. 35–47.

Robert K. Cunningham, R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendall, S. E. Webster, D. Wyschogrod, and M. A. Zissman. 1999. Evaluating intrusion detection systems without attacking your friends: The 1998 DARPA Intrusion Detection Evaluation. In *Proceedings of the SANS 1999 Workshop on Securing Linux*.

Hervé Debar, Marc Dacier, and Andreas Wespi. 1999. Towards a taxonomy of intrusion-detection systems. *Computer Networks* 31, 9, 805–822.

Hervé Debar, Marc Dacier, Andreas Wespi, and Stefan Lampart. 1998. *An Experimentation Workbench for Intrusion Detection Systems*. Technical Report. IBM Research, Zurich Research Laboratory.

Alex Dehnert. 2012. Intrusion detection using VProbes. *VMware Technical Journal* 1, 2, 28–31.

Holger Dreger, Anja Feldmann, Vern Paxson, and Robin Sommer. 2008. Predicting the resource consumption of network intrusion detection systems. In *Proceedings of the 2008 International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'08)*. ACM, New York, NY, 437–438.

Tudor Dumitras and Darren Shou. 2011. Toward a standard benchmark for computer security research: The worldwide intelligence network environment (WINE). In *Proceedings of the 1st Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. ACM, New York, NY, 89–96.

George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. 2002. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*. ACM, New York, NY, 211–224.

Robert Durst, Terrence Champion, Brian Witten, Eric Miller, and Luigi Spagnuolo. 1999. Testing and evaluating computer intrusion detection systems. *ACM Communications* 42, 7, 53–61.

Y. Frank Jou, Fengmin Gong, Chandru Sargor, Xiaoyong Wu, Shyhtsun F. Wu, Heng-Chia Chang, and Feiyi Wang. 2000. Design and implementation of a scalable intrusion detection system for the protection of network infrastructure. In *Proceedings of DARPA Information Survivability Conference and Exposition*, Vol. 2. 69–83.

Josè Fonseca, Marco Vieira, and Henrique Madeira. 2014. Evaluation of Web security mechanisms using vulnerability and attack injection. *IEEE Transactions on Dependable and Secure Computing* 11, 5, 440–453.

Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. 2010. MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the 6th International Conference on Emerging Networking Experiments and Technologies (CoNEXT'10)*. Article No. 8.

James C. Foster. 2007. *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. Syngress Publishing.

Mohammed Gad El Rab. 2008. *Evaluation des systèmes de détection d'intrusion*. Ph.D. Dissertation. Université Paul Sabatier—Toulouse III.

John E. Gaffney and Jacob W. Ulvila. 2001. Evaluation of intrusion detectors: A decision theory approach. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. 50–61.

Tal Garfinkel and Mendel Rosenblum. 2003. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the Network and Distributed Systems Security Symposium*. 191–206.

John Linwood Griffin, Adam Pennington, John S. Bucy, Deepa Choundappan, Nithya Muralidharan, and Gregory R. Ganger. 2003. *On the Feasibility of Intrusion Detection Inside Workstation Disks*. Research Paper. Carnegie-Mellon University, Pittsburgh, PA.

Guofei Gu, Prahlad Fogla, David Dagon, Wenke Lee, and Boris Skorić. 2006. Measuring intrusion detection capability: An information-theoretic approach. In *Proceedings of the 2006 ACM Symposium on Information, Computer, and Communications Security (ASIACCS'06)*. ACM, New York, NY, 90–101.

Mike Hall and Kevin Wiley. 2002. Capacity verification for high speed network intrusion detection systems. In *Proceedings of the 5th International Conference on Recent Advances in Intrusion Detection (RAID'02)*. Springer-Verlag, Berlin, 239–251.

Amin Hassanzadeh and Radu Stoleru. 2011. Towards optimal monitoring in cooperative IDS for resource constrained wireless networks. In *Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN'11)*. 1–8.

Hai Jin, Guofu Xiang, Feng Zhao, Deqing Zou, Min Li, and Lei Shi. 2009. VMFence: A customized intrusion prevention system in distributed virtual computing environment. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication (ICUIMC'09)*. ACM, New York, NY, 391–399.

Hai Jin, Guofu Xiang, Deqing Zou, Song Wu, Feng Zhao, Min Li, and Weide Zheng. 2011. A VMM-based intrusion prevention system in cloud computing environment. *Journal of Supercomputing* 66, 1133–1151.

Pradeep Kannadiga and Mohammad Zulkernine. 2005. DIDMA: A distributed intrusion detection system using mobile agents. In *Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. 238–245.

Christopher Kruegel, Fredrik Valeur, and Giovanni Vigna. 2005. *Intrusion Detection and Correlation: Challenges and Solutions*. Advances in Information Security, Vol. 14. Springer.

Marcus Laureano, Carlos Maziero, and Edgard Jamhour. 2007. Protecting host-based intrusion detectors through virtual machines. *Computer Networks* 51, 5, 1275–1283.

Hoang Le and Viktor K. Prasanna. 2013. A memory-efficient and modular approach for large-scale string pattern matching. *IEEE Transactions on Computers* 62, 5, 844–857.

Po-Ching Lin and Jia-Hau Lee. 2013. Re-examining the performance bottleneck in a {NIDS} with detailed profiling. *Journal of Network and Computer Applications* 36, 2, 768–780.

Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. 2000. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks* 34, 4, 579–595.

Flavio Lombardi and Roberto Di Pietro. 2011. Secure virtualization for cloud computing. *Journal of Network and Computer Applications* 34, 4, 1113–1122.

John McHugh. 2000. Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security* 3, 4, 262–294.

Peter Mell, Vincent Hu, Richard Lippmann, Josh Haines, and Marc Zissman. 2003. *An Overview of Issues in Testing Intrusion Detection Systems*. NIST Interagency/Internal Report. National Institute of Standards and Technology.

Yuxin Meng. 2012. Measuring intelligent false alarm reduction using an ROC curve-based approach in network intrusion detection. In *Proceedings of the IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA'12)*. 108–113.

Yuxin Meng and Wenjuan Li. 2012. Adaptive character frequency-based exclusive signature matching scheme in distributed intrusion detection environment. In *Proceedings of the IEEE 11th International Conference on Trust, Security, and Privacy in Computing and Communications (TrustCom'12)*. 223–230.

Chirag Modi and Dhiren Patel. 2013. A novel hybrid-network intrusion detection system (H-NIDS) in cloud computing. In *Proceedings of the IEEE Symposium on Computational Intelligence in Cyber Security (CICS'13)*. 23–30.

Khalid Nasr, AnasAbou-El Kalam, and Christian Fraboul. 2012. Performance analysis of wireless intrusion detection systems. In *Internet and Distributed Computing Systems*. Springer, 238–252.

Swapnil Patil, Anand Kashyap, Gopalan Sivathanu, and Erez Zadok. 2004. FS: An in-kernel integrity checker and intrusion detection file system. In *Proceedings of the 18th USENIX Conference on System Administration (LISA'04)*. 67–78.

Nicholas Puketza, Mandy Chung, Ronald A. Olsson, and Biswanath Mukherjee. 1997. A software platform for testing intrusion detection systems. *IEEE Software* 14, 5, 43–51.

Nicholas J. Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, and Ronald A. Olsson. 1996. A methodology for testing intrusion detection systems. *IEEE Transactions on Software Engineering* 22, 10, 719–729.

Kannaiya N. Raja, Srinivasan Arulanandam, and Raja Rajeswari. 2012. Two-level packet inspection using sequential differentiate method. In *Proceedings of the International Conference on Advances in Computing and Communications (ICACC'12)*. 42–45.

Marcus J. Ranum. 2001. *Experiences Benchmarking Intrusion Detection Systems*. White Paper. NFR Security Technical Publications.

Jason Reeves, Ashwin Ramaswamy, Michael Locasto, Sergey Bratus, and Sean Smith. 2012. Intrusion detection for resource-constrained embedded control systems in the power grid. *International Journal of Critical Infrastructure Protection* 5, 2, 74–83.

Ryan Riley, Xuxian Jiang, and Dongyan Xu. 2008. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID'08)*. Springer-Verlag, Berlin, 1–20.

Martin Roesch. 1999. Snort—lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration (LISA'99)*. 229–238.

Karen Scarfone and Peter Mell. 2007. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. Report 900-94. NIST Special Publication.

Vidar Evenrud Seeberg and Slobodan Petrovic. 2007. A new classification scheme for anonymization of real data used in IDS benchmarking. In *Proceedings of the 2nd International Conference on Availability, Reliability, and Security (ARES'07)*. 385–390.

Jaydip Sen, Arijit Ukil, Debasis Bera, and Arpan Pal. 2008. A distributed intrusion detection system for wireless ad hoc networks. In *Proceedings of the 16th IEEE International Conference on Networks (ICON'08)*. 1–6.

Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers and Security* 31, 3, 357–374.

Sushant Sinha, Farnam Jahanian, and Jignesh M. Patel. 2006. WIND: Workload-aware INtrusion Detection. In *Proceedings of the 9th International Conference on Recent Advances in Intrusion Detection (RAID'06)*. Springer-Verlag, Berlin, 290–310.

Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy (SP'10)*. 305–316.

Abhinav Srivastava, Kapil Singh, and Jonathon Giffin. 2008. Secure Observation of Kernel Behavior. Retrieved July 28, 2015, from http://hdl.handle.net/1853/25464.

William Stallings. 2002. *Cryptography and Network Security: Principles and Practice*. Pearson Education.

Bo Sun, Xuemei Shan, Kui Wu, and Yang Xiao. 2013. Anomaly detection based secure in-network aggregation for wireless sensor networks. *IEEE Systems Journal* 7, 1, 13–25.

Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, and Greg Kroah-Hartman. 2002. Linux security modules: General security support for the Linux kernel. In *Proceedings of the 11th USENIX Security Symposium*. 17–31.

Senhua Yu and Dipankar Dasgupta. 2011. An effective network-based intrusion detection using conserved self pattern recognition algorithm augmented with near-deterministic detector generation. In *Proceedings of the 2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS'11)*. 17–24.

Yuxin Meng and Lam-For Kwok. 2013. Towards an information-theoretic approach for measuring intelligent false alarm reduction in intrusion detection. In *Proceedings of the 12th IEEE International Conference on Trust, Security, and Privacy in Computing and Communications (TrustCom'13)*. 241–248.

Youhui Zhang, Hongyi Wang, Yu Gu, and Dongsheng Wang. 2008. IDRS: Combining file-level intrusion detection with block-level data recovery based on iSCSI. In *Proceedings of the 3rd International Conference on Availability, Reliability, and Security (ARES'08)*. 630–635.

Received October 2014; revised March 2015; accepted June 2015

Online Appendix to: Evaluating Computer Intrusion Detection Systems: A Survey of Common Practices

ALEKSANDAR MILENKOSKI, University of Würzburg
MARCO VIEIRA, University of Coimbra
SAMUEL KOUNEV, University of Würzburg
ALBERTO AVRITZER, Siemens Corporation, Corporate Technology
BRYAN D. PAYNE, Netflix, Inc.

A. EVALUATION OF INTRUSION DETECTION SYSTEMS: HISTORICAL OVERVIEW

In Figure 11, we depict chronologically ordered dates that mark major developments in the area of intrusion detection system (IDS) evaluation from its inception until the present date.

The earliest effort on evaluating IDSes in a systematic manner is the work of Puketza et al. [1996, 1997]. These authors presented an approach for evaluating IDSes based on principles of the field of software systems testing. They were the first to develop a framework for evaluating IDSes, which they describe in detail in their work from 1997. They used the framework to evaluate a network-based IDS in terms of attack detection accuracy, resource consumption, and performance under stress.

The years of 1998, 1999, and 2000 mark a major accomplishment in the area of IDS evaluation. The Lincoln Laboratory at the Massachusetts Institute of Technology, sponsored by the Defense Advanced Research Projects Agency (DARPA), evaluated multiple IDSes using generated trace files that contain host and network activities of benign and malicious nature. The latter are commonly known as the DARPA datasets (see Section 2.1). Cunningham et al. [1999] describe the approach taken to generate the DARPA datasets in detail. The DARPA datasets are still extensively used in IDS evaluation studies.

Debar et al. [1998] from the IBM Zurich Research Laboratory developed a workbench for evaluating IDSes. The workbench enabled the execution of attack scripts stored in a database maintained internally at IBM and the generation of regular workloads for training anomaly based IDSes. Debar et al. demonstrated the use of the workbench by evaluating multiple host-based IDSes.

A recent effort to support the rigorous evaluation of IDSes is being driven by Symantec. Dumitras and Shou [2011] presented Symantec's Worldwide Intelligence Network Environment (WINE) datasets,[45] which contain local and remote attacks (see Table I). They also presented an evaluation platform that makes use of the datasets and is available for use by researchers for evaluating security mechanisms. However, since the datasets are captured from real network infrastructures and systems, and therefore contain private user data, they can only be accessed on-site at Symantec to avoid legal issues. The large scale of this project is indicated by the fact that Symantec continuously monitors and records malicious activities using more than 240,000 sensors deployed in 200 countries.

In addition to attacks, which can be used for evaluating IDSes, the WINE datasets contain samples of malware (i.e., malicious software like trojans or viruses), which

[45] http://www.symantec.com/about/profile/universityresearch/sharing.jsp.

DOI: http://dx.doi.org/10.1145/2808691

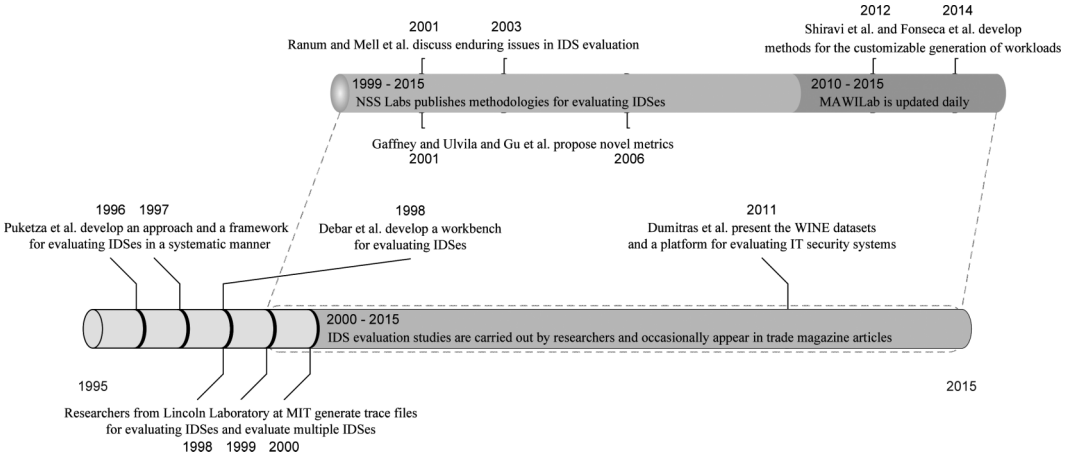


Fig. 11. Timelime showing dates that mark major developments in the area of IDS evaluation.

can be used for evaluating malware detection systems (e.g., antivirus systems). In contrast to IDSes, which are designed to detect ongoing attacks (see Section 1.1), malware detection systems are designed to detect malware running on a given host, whose installation normally takes place after an intrusion (i.e., a successful attack) has occurred. Evaluation of malware detection systems is outside the scope of this work.

There have been many small-scale IDS evaluation efforts between 2000 and today. Articles reviewing and comparing IDSes occasionally appear in trade magazines, such as the IDS evaluation study presented in the *SC* magazine in 2011.[46] Following the rising interest of researchers in intrusion detection since 2000, many IDS evaluation studies have been presented as part of publications proposing novel intrusion detection techniques or IDS evaluation methods.

Several works published between 2000 and today have had long-term impact on the IDS evaluation area: Ranum [2001] and Mell et al. [2003] proposed approaches and gave recommendations toward addressing enduring issues in IDS evaluation (e.g., the use of faulty or unrepresentative workloads, inaccurate interpretation of results from IDS evaluation studies); Gaffney and Ulvila [2001] and Gu et al. [2006] were the first to propose metrics for quantifying IDS attack detection accuracy that use specific measurement methods to address issues in using the conventional metrics at the time, such as the receiver operating characteristic curve (see Section 2.2); and focusing on the issue of using unrepresentative workloads, Shiravi et al. [2012] and Fonseca et al. [2014] developed methods for the customizable generation of IDS evaluation workloads that closely resemble real-world workloads at the time they are generated (see Section 2.1).

In 2010, the Measurement and Analysis on the WIDE Internet (MAWI) Working Group of the Widely Integrated Distributed Environment (WIDE) project announced MAWILab, a repository of publicly available traces intended for use in IDS evaluation studies [Fontugne et al. 2010].[47] This is a significant effort to enable the representative evaluation of modern network-based IDSes. The trace files in MAWILab contain network traffic captured from a trans-Pacific 150Mbps link between Japan and the United States. They contain regular network traffic as well as attacks, which are labeled before

[46]http://www.scmagazine.com/idsips/grouptest/241/#.
[47]http://www.fukuda-lab.org/mawilab/index.html.

the public release of the traces using a variety of attack labeling methods. MAWILab has been updated daily since its release until the present date.

In 1999, NSS Labs, an information security research and testing organization, pioneered third-party testing of IDSes with the publication of the first systematic, criteria-driven methodology for IDS testing. From 1999 until the present date, NSS Labs has been continuously supplying methodologies for testing IDSes to the public following trends in IDS design.[48] These methodologies may serve as guidelines for the rigorous testing of IDSes. For instance, in 2014, NSS Labs published a methodology for testing next-generation IDSes[49]—that is, IDSes designed to detect novel threats, such as advanced persistent threats and social media threats.

[48]Recent methodologies published by NSS Labs can be found at https://www.nsslabs.com/reports/categories/methodologies.

[49]https://www.nsslabs.com/reports/next-generation-intrusion-prevention-systems-ngips-test-methodology-v10.