Chapter 18
Software and System Security

Aleksandar Milenkoski and Samuel Kounev

Evaluation of computer security mechanisms is an active research area with many unresolved issues. The research community has produced many results that contribute towards addressing these issues. In this chapter, we systematize the accumulated knowledge and current practices in the area of evaluating computer security mechanisms. We define a design space structured into three parts: workload, metrics, and measurement methodology. We provide an overview of the current practices by surveying and comparing evaluation approaches and methods related to each part of the design space.

Computer security mechanisms—referred to as security mechanisms—are crucial for enforcing the properties of confidentiality, integrity, and availability of system data and services. A common security mechanism is an intrusion detection system (IDS). IDSes monitor on-going activities in the protected networks or hosts, detecting potentially malicious activities. The detection of malicious activities enables the timely reaction in order to stop an on-going attack or to mitigate the impact of a security breach. Other common security mechanisms include firewalls and access control (AC) systems.

To minimize the risk of security breaches, methods and techniques for evaluating security mechanisms in a realistic and reliable manner are needed. The benefits of evaluating security mechanisms are manifold. For instance, in the case of IDSes, one may compare different IDSes in terms of their attack detection accuracy in order 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 security mechanism by varying its configuration parameters and investigating their influence through evaluation tests. This enables a comparison of the evaluation results with respect to the configuration space of the mechanism and can help to identify an optimal configuration.

The evaluation of security mechanisms is of interest to many different types of users and professionals in the field of information security. This includes researchers, who typically evaluate novel security solutions; industrial software architects, who typically evaluate security mechanisms by carrying out internationally standardized large-scale tests; and IT security officers, who evaluate security mechanisms in order to select a mechanism that is optimal for protecting a given environment, or to optimize the configuration of an already deployed mechanism.

S. Kounev et al., *Systems Benchmarking*, https://doi.org/10.1007/978-3-030-41705-5_18

In this chapter, we survey existing knowledge on the evaluation of security mechanisms by defining an evaluation design space that puts existing work into a common context. Given the significant amount of existing practical and theoretical work, the presented systematization is beneficial for improving the general understanding of the topic by providing an overview of the current state of the field. The evaluation design space that we present is structured into three parts, that is, workload, metrics, and measurement methodology—the standard components of any system evaluation scenario. The discussions in this chapter are relevant for the evaluation of a wide spectrum of security mechanisms, such as firewalls and AC systems.

This chapter is structured as follows: in Section 18.1, we provide the background knowledge essential for understanding the topic of evaluating security mechanisms; in Section 18.1.1, we discuss different types of attacks and put the different security mechanisms into a common context; in Section 18.1.2, we demonstrate the wide applicability of evaluation of security mechanisms; and in Sections 18.2.1–18.2.3, we compare multiple approaches and methods that evaluation practitioners can employ.

The chapter is a compact summary of Milenkoski, Vieira, et al. (2015), Milenkoski, Payne, et al. (2015), and Milenkoski (2016). These publications provide more details on the topics discussed in this chapter.

18.1 Essential Background

We start with some background relevant for understanding the context of the content presented in the rest of the chapter. We first introduce attacks and common security mechanisms used to protect against them. Following this, we describe real-life practical scenarios where techniques for evaluating security mechanisms are needed, demonstrating the wide applicability of such techniques and their broad relevance.

18.1.1 Attacks and Common Security Mechanisms

A given system (i.e., a host) is considered secure if it has the properties of confidentiality, integrity, and availability of its data and services (Stallings, 2002). *Confidentiality* means the protection of data against its release to unauthorized parties. *Integrity* means the protection of data or services against modifications by unauthorized parties. Finally, *availability* means the protection of services such that they are ready to be used when needed. Attacks are deliberate attempts to violate the previously mentioned security properties (Shirey, 1999).

There are many security mechanisms used to enforce the properties of confidentiality, integrity, and availability of system data and services. Kruegel et al. (2005) classify security mechanisms by taking an attack-centric approach distinguishing between attack prevention, attack avoidance, and attack detection mechanisms. Based

on this classification, we put the different security mechanisms into a common context, as depicted in Figure 18.1.

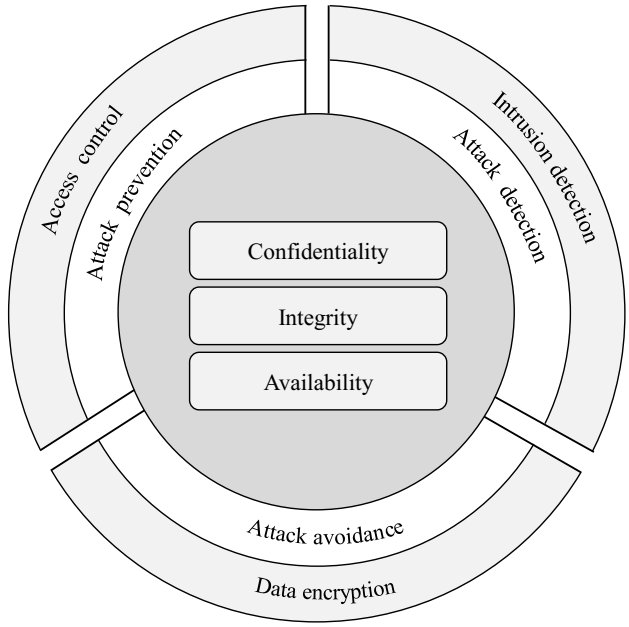

Fig. 18.1: Common security mechanisms

The attack prevention class includes security mechanisms that prevent attackers from reaching, or gaining access to, the targeted system. A representative mechanism that belongs to this class is access control, which uses the concept of identity to distinguish between authorized and unauthorized parties. For instance, firewalls distinguish between different parties trying to reach a given system over a network connection based, for example, on their IP addresses. According to access control policies, firewalls may allow or deny access to the system.

The attack avoidance class includes security mechanisms that modify the data stored in the targeted system such that it would be of no use to an attacker in case of an intrusion. A representative mechanism that belongs to this class is data encryption, which is typically implemented using encryption algorithms, such as RSA (Rivest–Shamir–Adleman) and DES (Data Encryption Standard).

The attack detection class includes security mechanisms that detect on-going attacks under the assumption that an attacker can reach, or gain access to, the targeted system and interact with it. A representative security mechanism that belongs to this class is intrusion detection. There are several different types of IDSes. For example, according to the target platform that IDSes monitor, they can be categorized into *host-based* (IDSes that monitor the activities of the users of the host where they are deployed), *network-based* (IDSes that monitor the network traffic that is destined

for, and/or originates from, a single host or a set of hosts that constitute a network environment), or *hybrid* IDSes. According to the employed attack detection method, IDSes can be categorized into *misuse-based* (IDSes that evaluate system and/or network activities against a set of signatures of known attacks), *anomaly-based* (trained IDSes that use a profile of regular network and/or system activities as a reference to distinguish between regular activities and anomalous activities, the latter being treated as attacks), or *hybrid* IDSes.

This chapter surveys existing knowledge on the evaluation of security mechanisms that belong to the attack prevention and attack detection class. It treats the topic of IDS evaluation as a single sub-domain of evaluation of security mechanisms.

18.1.2 Application Scenarios

We now present various application scenarios of evaluation of security mechanisms in order to demonstrate its wide applicability and broad relevance. This evaluation helps to determine how well a security mechanism performs and how well it performs when compared to other mechanisms. The answer to this question is of interest to many different types of professionals in the field of information security. These include designers of security mechanisms, both researchers and industrial software architects, as well as users of security mechanisms, such as IT security officers.

Researchers design novel security mechanisms. They typically focus on designing mechanisms that are superior in terms of given properties that are subject of research, for example, attack detection accuracy or workload processing capacity. To demonstrate the value of the research outcome, researchers typically perform small-scale evaluation studies comparing the proposed security mechanisms with other mechanisms in terms of the considered properties. For instance, Meng and Li (2012) measure workload processing throughput, Mohammed et al. (2011) measure power consumption, and Sinha et al. (2006) measure memory consumption. Further, in order to demonstrate that the proposed security mechanisms are practically useful, researchers also evaluate properties that are not necessarily in the focus of their research but are relevant from a practical perspective. For example, Lombardi and Di Pietro (2011) measure the performance overhead incurred by the IDS they propose.

Industrial software architects design security mechanisms with an extensive set of features according to their demand on the market. Security mechanisms, in this context, are typically evaluated by carrying out tests of a large scale. The latter are part of regular quality assurance procedures. They normally use internationally standardized tests for evaluating security mechanisms in a standard and comprehensive manner. For instance, Microsoft's Internet Security and Acceleration (ISA) Server 2004 has been evaluated according to the *Common Criteria* international standard for evaluating IT security products.[1] Standardized tests are performed in

[1] https://www.iso.org/standard/50341.html

strictly controlled environments and normally by independent testing laboratories, such as NSS Labs,[2] to ensure credibility of the results.

In contrast to evaluation studies performed by researchers, evaluation studies in industry normally include the evaluation of mechanism properties that are relevant from a marketing perspective. An example of such a property is the financial cost of deploying and maintaining an IDS or a firewall, which is evaluated as part of the tests performed by NSS Labs (NSS Labs, 2010).

IT security officers use security mechanisms to protect environments of which they are in charge from malicious activities. They may evaluate mechanisms, for example, when designing security architectures in order to select a mechanism that is considered optimal for protecting a given environment. Further, if a security architecture is already in place, an IT security officer may evaluate the performance of the selected mechanism for different configurations in order to identify its optimal configuration. The performance is typically very sensitive to the way the mechanism is configured.

In addition to security and performance-related aspects, as part of evaluation studies, further usability-related aspects may also be considered. This is to be expected since IT security officers deal with security mechanisms on a daily basis. For instance, security officers in charge of protecting large-scale environments may be cognitively overloaded by the output produced by the deployed security mechanisms (Komlodi et al., 2004). Thus, the ability to produce structured output that can be analyzed efficiently is an important property often considered when evaluating security mechanisms.

18.2 Current State

In this section, we put the existing practical and research work related to the evaluation of security mechanisms into a common context. Since such an evaluation is a highly complex task, any evaluation experiment requires careful planning in terms of the selection of workloads, tools, metrics, and measurement methodology. We provide a comprehensive systematization of knowledge in the respective areas providing a basis for the efficient and accurate planning of evaluation studies. We define an evaluation design space structured into three parts: workloads, metrics, and measurement methodology, considered to be standard components of any evaluation experiment.

The proposed design space structures the evaluation components and features they may possess with respect to different properties expressed as variability points in the design space. Note that we do not claim complete coverage of all variability points in the design space. We instead focus on the typical variability points of evaluation approaches putting existing work related to the evaluation of security mechanisms

[2] https://www.nsslabs.com

into a common context. We illustrate the defined design space categories by referring to evaluation experiments that fit each of the considered categories.

18.2.1 Workloads

In Figure 18.2, we depict the workload part of the design space. In order to evaluate a security mechanism, one needs both malicious and benign workloads. One can use them separately, for example, as *pure malicious* and *pure benign workloads* for measuring the capacity of the mechanism (Bharadwaja et al., 2011; Jin, Xiang, Zou, et al., 2013) or its attack coverage. Alternatively, one can use *mixed* workloads to subject the mechanism to realistic attack scenarios. A more detailed overview of typical use cases of different workload forms is provided in Section 18.2.3 in the context of measurement methodologies.

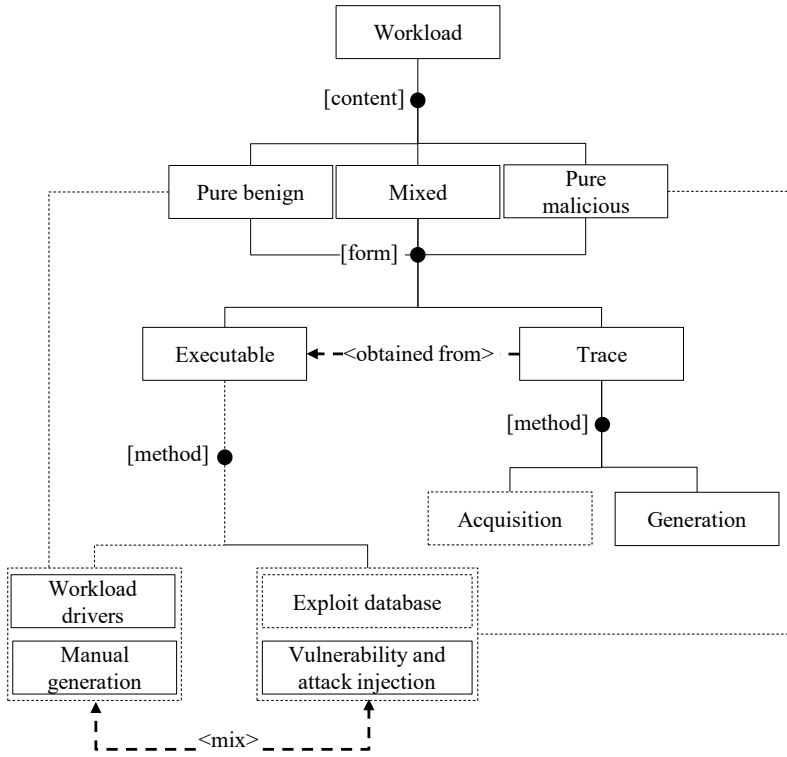

Fig. 18.2: Design space—workloads

Workloads for evaluating security mechanisms normally take an *executable* form for live testing, or a recorded form (i.e., a *trace*) generated by recording a live execution of workloads for later replay. A major advantage of using workloads in executable form is that they closely resemble a real workload as monitored by a security mechanism during operation. However, a malicious workload in executable form requires a specific victim environment, which can be expensive and time-consuming to setup.[3] In contrast, such an environment is not always required for replaying workload traces. Further, replicating evaluation experiments when using executable malicious workloads is usually a challenge since the execution of attack scripts might crash the victim environment or render it in an unstable state. The process of restoring the environment to an identical state as before the execution of the attack scripts may be time-consuming. At the same time, multiple evaluation runs would be typically required to ensure statistical significance of the observed system behavior. We refer the reader to Mell et al. (2003) for further comparison of workloads in executable and trace form. In the following, we discuss different methods for generating benign and malicious workloads in executable form: use of *workload drivers* and *manual generation* approaches for generation of pure benign workloads and use of an *exploit database* and *vulnerability and attack injection* techniques for generating pure malicious workloads (Figure 18.2).

18.2.1.1 Workload Drivers

For the purpose of live testing, a common practice is to use benign workload drivers in order to generate pure benign workloads with different characteristics. We surveyed evaluation experiments (e.g., Jin, Xiang, Zhao, et al. (2009), Lombardi and Di Pietro (2011), Jin, Xiang, Zou, et al. (2013), Griffin et al. (2003), Patil et al. (2004), Riley et al. (2008), Reeves et al. (2012), and Zhang et al. (2008)) concluding that some of the commonly used workload drivers are the following (and alike): SPEC CPU2000[4] for generation of CPU-intensive workloads; IOzone[5] and Postmark (Katcher, 1997) for generation of file I/O-intensive workloads; httpbench,[6] dkftpbench,[7] and ApacheBench for generation of network-intensive workloads; and UnixBench[8] for generation of system-wide workloads that exercise not only the hardware but also the operating system. 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

[3] While setting up their workbench for evaluation of IDSes, Debar et al. (1998) concluded that "*Transforming exploit scripts found in our database into attack scripts requires some work but setting up a reliable and vulnerable server has also proved to be a difficult task!*"

[4] https://www.spec.org/cpu2000

[5] http://www.iozone.org

[6] http://freecode.com/projects/httpbench

[7] http://www.kegel.com/dkftpbench

[8] http://code.google.com/p/byte-unixbench

increase the workload intensity over time, as typically done when evaluating the capacity of a security mechanism.

18.2.1.2 Manual Generation

An alternative approach to using workload drivers is to manually execute tasks that are known to exercise specific system resources. 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, while Lombardi and Di Pietro (2011) perform encoding of a .mp3 file); 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-intensive and file I/O-intensive) tasks (e.g., performed by Wright et al. (2002), Lombardi and Di Pietro (2011), Riley et al. (2008), Reeves et al. (2012), and Dunlap et al. (2002)). This approach of benign workload generation enables the generation of workloads with behavior as observed by the security mechanism under test during regular system operation; however, it does not support workload customization and might require substantial human effort.

18.2.1.3 Exploit Database

As pure malicious workloads in executable form, security researchers typically use an exploit database. They have a choice of assembling an exploit database by themselves or using a readily available one.

A major disadvantage of the *manual assembly* is the high cost of the attack script collection process. For instance, when collecting publicly available attack scripts, the latter typically have to be adapted to exploit vulnerabilities of a specific victim environment. This includes modification of shell codes, adaptation of employed buffer overflow techniques, and similar. Depending on the number of collected attack scripts, this process may be extremely time-consuming. Mell et al. (2003) report that in 2001 the average number of attack scripts in common exploit databases was in the range of 9–66, whereas some later works, such as the one of Lombardi and Di Pietro (2011), use as low as four attack scripts as a malicious workload.

To alleviate the above-mentioned issues, many researchers employ penetration testing tools to use a *readily available* exploit database. The Metasploit framework (Maynor et al., 2007) is a popular penetration testing tool that has been used in evaluation experiments (Görnitz et al., 2009). The interest of security researchers in Metasploit (and in penetration testing tools in general) is not surprising since Metasploit enables customizable and automated platform exploitation by using an exploit database that is maintained up-to-date and is freely available. Metasploit is very well accepted by the security community not only due to the large exploit database it provides but also because it enables rapid development of new exploits. However, although penetration testing frameworks might seem like an ideal solution

for generating malicious workloads, they have some critical limitations. Gad El Rab (2008) analyzes the Metasploit's exploit database to discover that most of the exploits are executed from remote sources and exploit only implementation and design vulnerabilities, neglecting operation and management vulnerabilities (Shirey, 1999). Such characteristics are common for many penetration testing tools, which indicates their limited usefulness in evaluating security mechanisms.

An effort to provide an extensive collection of exploits to security researchers has been driven by Symantec. Dumitras and Shou (2011) present Symantec's WINE datasets, which contain a collection of malware samples that exploit various novel vulnerabilities. The large scale of this project is indicated by the fact that Symantec's sensors continuously collect malware samples from 240,000 sensors deployed in 200 countries worldwide. Due to the continuous nature of the malware collection process, this malware database is useful not only as a basis for generating extensive malicious workloads but also for providing an up-to-date overview of the security threat landscape. However, since the malware samples are collected from real platforms and contain user data, Symantec's malware samples can be accessed only on-site at the Symantec Research Lab to avoid legal issues.

18.2.1.4 Vulnerability and Attack Injection

An alternative approach to the use of an exploit database is the use of vulnerability and attack injection techniques.

Vulnerability injection enables live testing by artificially injecting exploitable vulnerable code in a target platform. Thus, this technique is useful in cases where the collection of attack scripts that exploit vulnerabilities is unfeasible. However, this method for generation of malicious workloads is still in an early phase of research and development. Vulnerability injection relies on the basic principles of the more general research area of fault injection. Since it enables estimation of fault-tolerant system measures (e.g., fault coverage, error latency) (Arlat et al., 1993), fault injection is an attractive approach to validate specific fault handling mechanisms and to assess the impact of faults in actual systems. In the past decades, research on fault injection has been focused on the emulation of hardware faults. Carreira et al. (1998) and Rodríguez et al. (1999) have shown that it is possible to emulate these faults in a realistic manner. The interest in software fault injection has been increasing and has been a foundation for many research works on the emulation of software faults (e.g., Durães and Madeira (2003)). In practice, software fault injection deliberately introduces faults into a software system in a way that emulates real software faults. A reference technique, proposed by Durães and Madeira (2006), is G-SWFI (Generic Software Fault Injection Technique), which enables injection of realistic software faults using educated code mutation. The injected faults are specified in a library derived from an extensive field study aimed at identifying the types of bugs that are usually found in many software systems.

A specific application of software fault injection is a security assessment in which of central importance are software faults that represent security vulnerabili-

ties. Fonseca and Vieira (2008) analyzed 655 security patches of 6 web applications to discover that only 12 generic software faults are responsible for all security problems of the applications. This finding has motivated further research in software fault injection as a method for security evaluation. Fonseca, Vieira, and Madeira (2009) proposed a procedure that enables automatic vulnerability injection and attack of web applications. To accurately emulate real-world web vulnerabilities, this work relies on results obtained from a field study on real security vulnerabilities (Fonseca and Vieira, 2008). Fonseca, Vieira, and Madeira (2009) built a Vulnerability and Attack Injector, a mechanism that automatically exploits injected vulnerabilities. In order to inject vulnerabilities in the source code of web applications, first the application source code is analyzed searching for locations where vulnerabilities can be injected. Once a possible location is found, a vulnerability is injected by performing a code mutation. The code mutation is performed by vulnerability operators that leverage a realistic field data of vulnerable code segments. For more details on the vulnerability injection procedure, we refer the reader to Fonseca, Vieira, and Madeira (2009). Aware of the injected vulnerability, the Attack Injector interacts with the web application in order to deliver attack payloads.

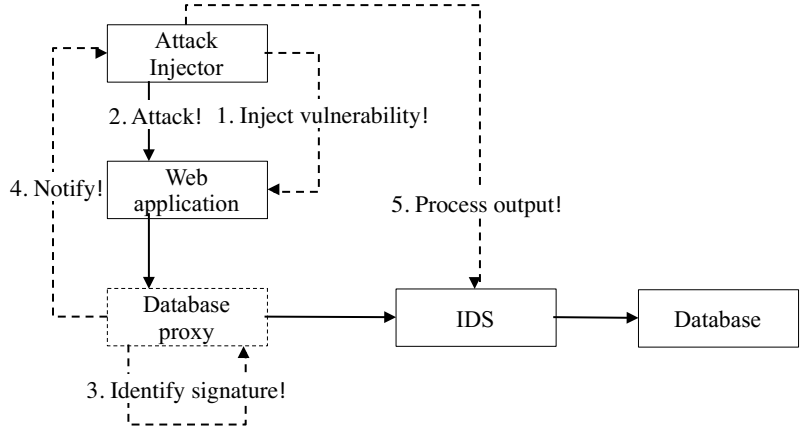

Fig. 18.3: Use of vulnerability injection to evaluate a security mechanism (an IDS)

Fonseca, Vieira, and Madeira (2009) also demonstrated a preliminary approach for automated (i.e., without human intervention) evaluation of a security mechanism that detects SQL (Structured Query Language) injection attacks. We depict a procedure that follows this approach in Figure 18.3. First, the Vulnerability Injector injects a vulnerability in the web application, followed by the Attack Injector that delivers an attack payload with a given signature, that is, an attack identifier. Fonseca, Vieira, and Madeira (2009) developed a database proxy that monitors the communication between the application and the database in order to identify the presence of an attack signature. In case it identifies such signature, it notifies the Attack Injector that the injected vulnerability is successfully exploited. In this way, the Attack Injector

builds a ground truth knowledge. Given that Fonseca, Vieira, and Madeira (2009) customized the Attack Injector to process the output of the security mechanism that monitors the traffic to the database, the Attack Injector can automatically calculate values of attack detection accuracy metrics (see Section 18.2.2).

Attack injection, as an approach separate from vulnerability injection, enables the generation of workloads for evaluating security mechanisms that contain benign and malicious activities such that attacks, crafted with respect to representative attack models, are injected during regular operation of a given system. Same as vulnerability injection, this technique is useful in cases where the collection of attack scripts that exploit vulnerabilities is unfeasible.

In Milenkoski, Payne, et al. (2015), we proposed an approach for the accurate, rigorous, and representative evaluation of hypercall security mechanisms designed to mitigate or detect hypercall attacks. Hypercalls are software traps from the kernel of a virtual machine (VM) to the hypervisor. For instance, the execution of an attack triggering a vulnerability of a hypervisor's hypercall handler may lead to a crash of the hypervisor or to altering the hypervisor's memory. The latter may enable the execution of malicious code with hypervisor privilege. In Milenkoski, Payne, et al. (2015), we presented *HInjector*, a customizable framework for injecting hypercall attacks during regular operation of a guest VM in a Xen-based environment.[9] The goal of HInjector is to exercise the sensors of a security mechanism that monitors the execution of hypercalls. The attacks injected by *HInjector* conform to attack models based on existing Xen vulnerabilities. We distinguish the following attack models:

- Invoking hypercalls from irregular call sites. Some hypercall security mechanisms (e.g., Bharadwaja et al. (2011)) may consider hypercalls invoked from call sites unknown to them, for example, an attacker's loadable kernel module (LKM), as malicious.
- Invoking hypercalls with anomalous parameter values (a) outside the valid value domains or (b) crafted for exploiting specific vulnerabilities not necessarily outside the valid value domains. This attack model is based on the Xen vulnerabilities described in CVE-2008-3687, CVE-2012-3516, CVE-2012-5513, and CVE-2012-6035.
- Invoking a series of hypercalls in irregular order, including repetitive execution of a single or multiple hypercalls. This attack model is based on the Xen vulnerability described in CVE-2013-1920. The repetitive execution of hypercalls, for example, requesting system resources, is an easily feasible attack that may lead to resource exhaustion of collocated VMs.

In Figure 18.4, we depict the architecture of HInjector, which consists of the components *Injector*, *LKM*, *Identificator*, *Configuration*, and *Logs*. We refer to the VM injecting hypercall attacks as malicious VM (MVM). The security mechanism under test (an IDS) is deployed in a secured VM (SVM) collocated with MVM.

The *Injector*, deployed in the hypercall interface of MVM's kernel, intercepts hypercalls invoked by the kernel during regular operation and modifies hypercall pa-

[9] https://xenproject.org

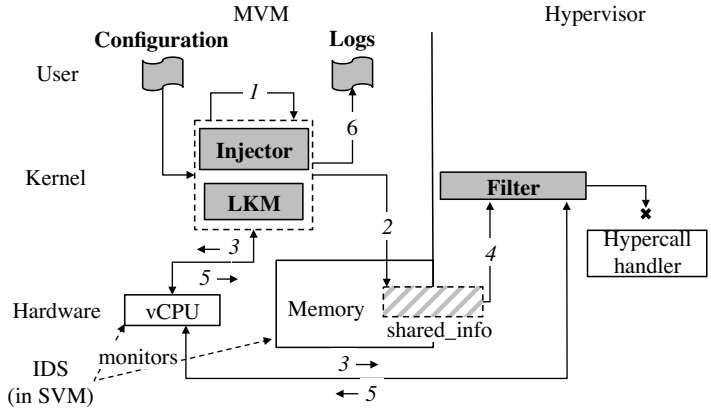

Fig. 18.4: Architecture of HInjector

rameter values on-the-fly, making them anomalous. The Injector is used for injecting hypercalls invoked from a regular call site.

The loadable kernel module (*LKM*), a module of MVM's kernel, invokes regular hypercalls, hypercalls with anomalous parameter values, or hypercalls in irregular order. The LKM is used for injecting hypercalls invoked from an irregular call site.

The *Identificator*, deployed in Xen's hypercall interrupt handler (i.e., 0x82 interrupt), identifies hypercalls injected by the Injector or the LKM, blocks their execution, and returns a valid error code. The latter is important for preventing MVM crashes by allowing the control flow of MVM's kernel to handle failed hypercalls that have been invoked by it. The Identificator blocks the execution of Xen's hypercall handlers to prevent Xen crashes. The Identificator identifies injected hypercalls based on information stored by the Injector/LKM in the *shared_info* structure, a memory region shared between a guest VM and Xen. To this end, we extended *shared_info* with a string field named *hid* (hypercall identification).

The *configuration* is a set of user files containing configuration parameters for managing the operation of the Injector and the LKM. Currently, it allows for specifying the duration of an injection campaign, valid parameter value domains and/or specifically crafted parameter values for a given hypercall (relevant to the Injector and the LKM), and valid order of a series of hypercalls (relevant to the LKM).

The *logs* are user files containing records about injected hypercalls—that is, hypercall IDs (hypercall identification numbers assigned by Xen) and parameter values as well as timestamps. The logged data serves as reference data (i.e., as "ground truth") used for calculating attack detection accuracy metrics.

In Figure 18.4, we depict the steps involved in injecting a single hypercall by the Injector/LKM. An illustrative example of the Injector injecting a hypercall with a parameter value outside of its valid domain is as follows: (1) The Injector intercepts a hypercall invoked by MVM's kernel and replaces the value, for example, of the first parameter, with a generated value outside the parameter's valid value domain

specified in the configuration; (2) The Injector stores the ID of the hypercall, the number of the parameter with anomalous value (i.e., one), and the parameter value itself in *hid*; (3) The Injector passes the hypercall to MVM's virtual CPU, which then issues a 0x82 interrupt and passes control to Xen; (4) The Identificator, using the data stored in *hid*, identifies the injected hypercall when it arrives at Xen's 0x82 interrupt handler; (5) The Identificator returns a valid error code without invoking the hypercall's handler; and (6) After the return code arrives at MVM's kernel, the Injector stores in the log files, the ID and parameter values of the injected hypercall, and a timestamp.

We now discuss methods for obtaining pure benign, pure malicious, or mixed workloads in trace form. We distinguish between trace *acquisition* and trace *generation*.

18.2.1.5 Trace Acquisition

Under trace acquisition, we understand the process of obtaining trace files from an industrial organization, that is, real-world traces, or obtaining publicly available traces.

Real-world traces subject a security mechanism under test to a workload as observed during operation in a real deployment environment. However, they are usually very difficult to obtain mainly due to the unwillingness of industrial organizations to share operational traces with security researchers because of privacy and similar legal concerns. Thus, real-world traces are usually anonymized by using various techniques, which are known to introduce inconsistencies in the anonymized trace files. Another challenge is that the attacks in real-world traces are usually not labeled and may contain unknown attacks, making the construction of the "ground truth" challenging. Lack of ground truth information severely limits the usability of trace files; for example, one could not quantify the false negative detection rate. To quote from Sommer and Paxson (2010): "*If one cannot find a sound way to obtain ground-truth for the evaluation, then it becomes questionable to pursue the work at all, even if it otherwise appears on a solid foundation.*" Among many other things, the labeling accuracy and feasibility depend on the capturing method used to record the traces, that is, on the richness of the information regarding the recorded activity itself (e.g., network packet timestamps, system call arguments). For instance, Sperotto et al. (2009) argue that when it comes to evaluating a network-based IDS that differentiates between network flows, traces captured in honeypots enable much more efficient labeling than traces captured in real-world environments, since honeypots are able to record relevant activity information that is not usually provided with real-world production traces. An interesting derived observation is that one may tend to prioritize trace generation in an isolated and a specialized environment (e.g., a honeypot). This is due to the increased feasibility of trace labeling, overusing real-world traces that are representative of the real world, even in the hardly achievable case when

real-world production traces are available.[10] We discuss more on honeypots and on trace generation approaches in general in Section 18.2.1.6.

In contrast to proprietary 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 they quickly become outdated after their public release; that is, attacks have limited shelf-life, and further, the characteristics of the benign background activities and the mix of malicious and benign activities change significantly over time. Since such activities are recorded permanently in trace files for later reuse, traces lose on representativeness over time. Some of the most frequently used publicly available traces include the DARPA (MIT Lincoln Laboratory, 1999) and the KDD Cup'99 (University of California, 1998) datasets, which are currently considered outdated (Sommer and Paxson, 2010).[11] However, these traces have been used in many evaluation experiments over the last two decades (e.g., Alserhani et al. (2010), Yu and Dasgupta (2011), and Raja et al. (2012)). We also conclude that the trend of overusing these datasets continues up to the current date despite the past criticism of their poor representativeness. For instance, Sommer and Paxson (2010) referred to the DARPA dataset as "*no longer adequate for any current study*" and "*wholly uninteresting if a network-based IDS detects the attacks it [the DARPA dataset] contains,*" stressing the overuse of these datasets due to lack of alternative publicly available datasets for security research. The DARPA and the KDD Cup'99 datasets have also been extensively criticized. For instance, McHugh (2000) criticizes the DARPA dataset for unrealistic distribution of probe/surveillance attacks in the benign background activity. The KDD Cup'99 dataset is known for lack of precise temporal information on the attacks recorded in the trace files, which is crucial for attack detection to many IDSes. Despite all criticism, some researchers are still looking for usage scenarios of these datasets. For instance, Engen et al. (2011) identify the KDD Cup'99 dataset as useful in the evaluation of the learning process of anomaly-based IDSes (e.g., learning from a very large dataset, incremental learning, and similar).

18.2.1.6 Trace Generation

Under trace generation, we understand the process of generating traces by the evaluator himself. In order to avoid the previously mentioned issues related to acquiring

[10] A worthy point to mention is that real-world traces of a "small" size may still be labeled in a reasonable time; however, such traces would contain a small amount of attacks. To quote from Sperotto et al. (2009): "*... labeling is a time-consuming process: it could easily be achieved on short traces, but these traces could present only a limited amount of security events.*" The amount of (human) resources that one has available for labeling plays a central role in determining the acceptable size of real-world traces that can be labeled in a time-efficient manner.

[11] We focus on the DARPA and the KDD Cup'99 datasets because of their popularity. However, we stress that the risk of a dataset to become outdated soon after its public release is *not* a characteristic of the DARPA and the KDD Cup'99 datasets in particular, but to the contrary, of all datasets in general.

traces, researchers generate traces in a testbed environment or deploy a honeypot in order to capture malicious activities.

The generation of traces in a testbed environment is challenged by several concerns. For instance, the cost of the resources needed to build a testbed that scales to realistic production environment may be high. Further, the approach for the generation of traces may produce faulty or simplistic workloads. For instance, Sommer and Paxson (2010) warn that activities captured in small testbed environments may differ fundamentally from activities in a real-world platform. Finally, the methods used to generate traces are not flexible enough to timely follow the current attack and benign activity trends. This issue, in particular, has motivated one of the major current research directions that deals with the generation of traces in a testbed environment in a customizable and scientifically rigorous manner. Such research is mainly motivated by the fact that the characteristics of attacks and of benign workloads are rapidly changing over time, making the one-time datasets inappropriate for evaluation on a long-term basis. To this end, Shiravi et al. (2012) proposed the use of workload profiles that enable customization of both malicious and benign network traffic. This includes customization of the distribution of network traffic from specific applications and protocols as well as customization of intrusive activities.

Honeypots enable recording of malicious activities performed by an attacker without revealing their purpose. By mimicking real operating systems and vulnerable services, honeypots record the interaction between the attack target and the attack itself. Security researchers often use honeyd,[12] a low-interaction honeypot that can emulate a network of an arbitrary number of hosts, where each host may run multiple services. Honeyd is attractive to security researchers since it is open-source and is well equipped with many logging and log processing utilities. Maybe the most extensive deployment of honeyd up-to-date is in the frame of the Leurre.com project,[13] which at the time of writing consists of 50 active honeyd instances in 30 countries worldwide. Since honeypots are usually isolated from production platforms, almost all of the interactions that they observe are malicious, making honeypots ideal for generation of pure malicious traces. However, since low-interaction honeypots use complex scripts to interact with attacks, they are often unable to interact with and record zero-day attacks. Under a zero-day attack, we understand an attack that exploits a vulnerability that has not been publicly disclosed before the execution of the attack. The notion "zero-day" indicates that such an attack occurs on "day zero" of public awareness of the exploited vulnerability. A promising solution of this issue is the work of Leita et al. (2006) where they incorporate unsupervised learning mechanism in the interaction state machine of ScriptGen, a framework for automatic generation of honeyd scripts with the benefit to capture zero-day attacks. ScriptGen was later enhanced and implemented in the honeyd instances of the Leurre.com project.

[12] http://www.honeyd.org

[13] http://www.leurrecom.org

18.2.2 Metrics

In Figure 18.5, we depict the metrics part of the design space. We distinguish between two metric categories: (1) performance-related and (2) security-related.

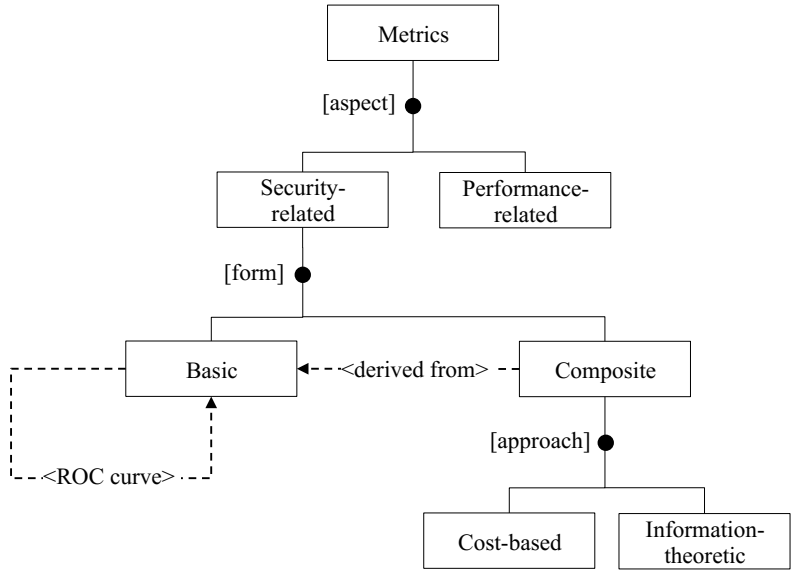

Fig. 18.5: Design space—metrics

Under performance-related metrics, we consider metrics that quantify the non-functional properties of a security mechanism under test, such as capacity, performance overhead, resource consumption, and similar. The metrics that apply to these properties, such as processing throughput and CPU utilization, are typical for traditional performance benchmarks. The practice in the area of evaluating security mechanisms has shown that they are also applicable to security evaluation. For instance, Meng and Li (2012) measure workload processing throughput, Lombardi and Di Pietro (2011) measure performance overhead, Mohammed et al. (2011) measure energy consumption, and Sinha et al. (2006) measure memory consumption. In the context of this chapter, we focus on the systematization and analysis of security-related metrics.

Under security-related metrics, we assume metrics that are used exclusively in security evaluation. In this chapter, we focus on metrics that quantify attack detection accuracy. These are relevant for evaluating security mechanisms that feature attack detection and issue attack alerts, such as IDSes. We distinguish between *basic* and *composite* security-related metrics. We provide an overview of these metrics in Table 18.1, where we annotate an attack (or an intrusion) with I and an attack alert with A.

Table 18.1: Security-related metrics

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

Basic Security-Related Metrics The basic metrics are most common, and they quantify various individual attack detection properties. For instance, the true positive rate $P(A|I)$ quantifies the probability that an alert is really an intrusion. The false positive rate $P(A|\neg I)$ quantifies the probability that an alert is not an intrusion but a regular benign activity. Alternatively, one can use the respective complementary metrics, that is, the true negative rate $P(\neg A|\neg I)$ and the false negative rate $P(\neg A|I)$. In evaluation experiments, the output of the security mechanism under test is compared with a ground truth information in order to calculate the above-mentioned probabilities. Other basic metrics are the positive predictive value (PPV), $P(I|A)$, and the negative predictive value (NPV), $P(\neg I|\neg A)$. The former quantifies the probability that there is an intrusion when an alert is generated, whereas the latter quantifies the probability that there is no intrusion when an alert is not generated. These metrics are normally calculated once one has already calculated $P(A|I)$, $P(A|\neg I)$, $P(\neg A|\neg I)$, and $P(\neg A|I)$ by using the Bayesian theorem for calculating the conditional probability (Table 18.1). Thus, PPV and NPV are also known as Bayesian positive detection rate and Bayesian negative detection rate, respectively. PPV and NPV are useful from a usability perspective, for example, in situations when an alert automatically triggers an attack response. In such situations, low values of PPV and NPV indicate that the considered security mechanism 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.

Composite Security-Related Metrics Security researchers often combine the above presented basic metrics in order to analyze relationships between them. Such analysis is used to discover an optimal operating point (e.g., a configuration of the mechanism under test that yields optimal values of both the true and the false positive detection rate) or to compare multiple security mechanisms. It is a common practice to use a Receiver Operating Characteristic (ROC) curve in order to investigate the relationship between the true positive and the false positive detection rate. However, some argue that a ROC curve analysis is often misleading (e.g., Gu et al. (2006), Gaffney and Ulvila (2001), and Stolfo et al. (2000)) and propose alternative approaches based on (1) metrics that use cost-based measurement methods or (2) metrics that use information-theory measurement methods. In the following, we briefly analyze two of the most prominent metrics that belong to these categories—that is, the expected cost metric (Gaffney and Ulvila, 2001) and the intrusion detection capability metric (Gu et al., 2006)—presented in Table 18.1. We focus on comparing the applicability of ROC curves and these metrics for the purpose of comparison of multiple IDSes. We assume as a goal the comparison of two IDSes: IDS_1 and IDS_2. For that purpose, we analyze the relationship between the true positive and the false positive detection rate denoted by $1 - \beta$ and α, respectively (Table 18.1). We assume that for IDS_1, $1 - \beta$ is related to α with a power function; that is, $1 - \beta = \alpha^k$ such that $k = 0.002182$. Further, we assume that for IDS_2, $1 - \beta$ is related to α with an exponential function; that is, $1 - \beta = 1 - 0.00765e^{-208.32\alpha}$. We take the values of k, α, and the coefficients of the exponential function from Gaffney and Ulvila (2001).

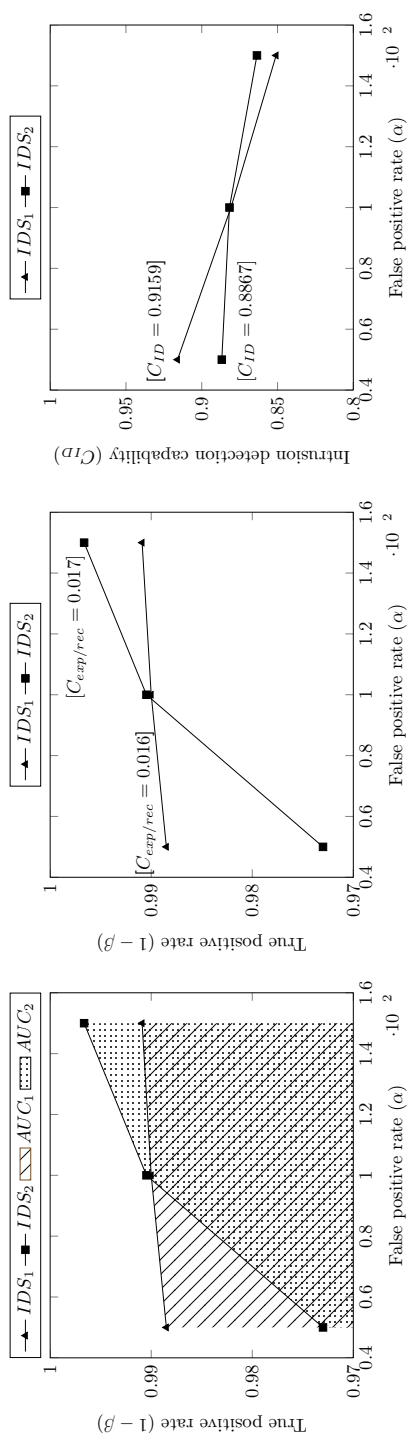

Fig. 18.6: IDS comparison with: (a) ROC curves, (b) expected cost metric, and (c) intrusion detection capability metric

We calculate the values of $1-\beta$ for IDS_1 and IDS_2 for $\alpha = \{0.005, 0.010, 0.015\}$. We depict the values of $1-\beta$ for IDS_1 and IDS_2 in Table 18.2.

Table 18.2: Values of $1-\beta$, C_{exp}, and C_{ID} for IDS_1 and IDS_2

	IDS_1			IDS_2		
α	$1-\beta$	C_{exp}	C_{ID}	$1-\beta$	C_{exp}	C_{ID}
0.005	0.9885	**0.016**	**0.9159**	0.973	0.032	**0.8867**
0.010	0.99	0.019	0.8807	0.99047	0.019	0.8817
0.015	0.9909	0.022	0.8509	0.99664	**0.017**	0.8635

In Figure 18.6a, we depict the ROC curves that express the relationship between $1-\beta$ and α for IDS_1 and IDS_2. One may notice that the ROC curves intersect approximately at $1-\beta = 0.99$ and $\alpha = 0.01$. Thus, one could not identify the better IDS in a straightforward manner. Note that an IDS is considered better than another if it features higher positive detection rate $(1-\beta)$ than the other IDS at all operating points along the ROC curve. An intuitive solution to this problem, as suggested by Durst et al. (1999), is to compare the area under the ROC curves, that is, $AUC_1 : \int_{\alpha=0.005}^{\alpha=0.015} \alpha^{0.002182} \mathrm{d}\alpha$ 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, since it is based on all operating points of the compared IDSes, while in reality, a given IDS is configured according to a single operating point. Moreover, ROC curves do not express the impact of the rate of occurrence of intrusion events ($B = P(I)$), known as the base rate, on α and $1-\beta$. As suggested by Axelsson (2000), the attack detection performance of an IDS should be interpreted with respect to a base rate measure due to the base-rate fallacy phenomenon.

In order to overcome the above-mentioned issues related to ROC curve analysis, Gaffney and Ulvila (2001) propose the measure of cost as an additional comparison parameter. They combine ROC curve analysis with cost estimation by associating an estimated cost with an operating point (i.e., with a measure of false negative and false positive rates); that is, they introduce a cost ratio $C = C_\beta / C_\alpha$, where C_α is the cost of an alert when an attack has not occurred, and C_β is the cost of not detecting an attack when it has occurred. Gaffney and Ulvila (2001) use the cost ratio to calculate the expected cost C_{exp} of a security mechanism operating at a given operating point (see Table 18.1). For further explanation of the analytical formula of C_{exp}, we refer the reader to Gaffney and Ulvila (2001). By using C_{exp}, one can determine which mechanism performs better by comparing the estimated costs when each mechanism operates at its optimal operating point. The mechanism that has lower C_{exp} associated with its optimal operating point is considered to be better. A given operating point of a single mechanism is considered optimal if it has the lowest C_{exp} associated with it when compared with the other operating points.

To determine the optimal operating points of IDS_1 and IDS_2, we calculate the values of C_{exp} for each operating point of the two IDSes. Note that C_{exp} depends on the base rate B as well as the cost ratio C (Table 18.1). Thus, to calculate the values of C_{exp}, we assume that $C = 10$, that is, 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 18.2. One may conclude that the optimal operating point of IDS_1 is $(0.005, 0.9885)$ and of IDS_2 is $(0.015, 0.99664)$; that is, the associated expected cost with these points is minimal. Since the minimal C_{exp} of IDS_1 (0.016) is smaller than the minimal C_{exp} of IDS_2 (0.017), one may conclude that IDS_1 performs better. We depict the ROC curves annotated with the minimal C_{exp} of IDS_1 and of IDS_2 in Figure 18.6b.

Although the discussed cost-based metric enables straightforward comparison of multiple security mechanisms, it strongly depends 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. We argue that in reality, it might be extremely difficult to construct such a model. Cost-analysis models normally take multiple parameters into consideration that often might not be easy to measure or might not be measurable at all (e.g., man-hours, system downtime, and similar). Further, if the cost model is not precise, the calculation of C_{exp} would be inaccurate. Finally, C_{exp} provides a comparison of security mechanisms based on a strongly subjective measure (i.e., cost), making the metric unsuitable for objective comparisons. This issue is also acknowledged by Gu et al. (2006). We argue that the above-mentioned issues apply not only to the considered cost-based metric but also to all metrics of similar nature.

Another approach for quantification of attack detection performance is the information-theoretic approach. In this direction, Gu et al. (2006) propose a metric called intrusion detection capability (denoted by C_{ID}, Table 18.1). Gu et al. (2006) 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). It is assumed that both the input and the output stream have a certain degree of uncertainty reflected by the entropies $H(X)$ and $H(Y)$, respectively. Thus, Gu et al. (2006) model the number of correct guesses by an IDS (i.e., $I(X;Y)$) as a piece of mutually shared information between the random variables X and Y; that is, $I(X;Y) = H(X) - H(X|Y)$. An alternative interpretation is that the accuracy of an IDS is modeled as a reduction of the uncertainty of the IDS input, $H(X)$, after the IDS output Y is known. Finally, by normalizing the shared information $I(X;Y)$ with the entropy of the input variable $H(X)$, the intrusion detection capability metric C_{ID} is obtained (Table 18.1). Note that 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 $I(X;Y)$. Thus, one may conclude that C_{ID} incorporates the base rate B and many basic metrics, such as the true positive rate $(1 - \beta)$, the false positive rate (α), and similar. For the definition of the relationship between C_{ID} and B, $1 - \beta$, and α, 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 in a ROC curve. With this assignment, one obtains a new curve, that is, a C_{ID} curve. Assuming a base rate of $B = 0.10$, we calculated C_{ID}

for various operating points of IDS_1 and IDS_2 (Table 18.2). In Figure 18.6c, 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, that is, the point that marks the highest C_{ID}. Further, one can compare IDSes by comparing the maximum C_{ID} of each IDS. An IDS is considered to perform better if its optimal operating point has a higher C_{ID} associated with it. From Table 18.2, one would consider the IDS_1 as a better performing IDS since it has greater maximum C_{ID} (0.9159) than the maximum C_{ID} of IDS_2 (0.8867).

Note that, in contrast to the expected cost metric, the intrusion detection capability metric is not based on subjective measures such as cost, which makes it suitable for objective comparisons. However, this also implies that this metric lacks expressiveness with respect to subjective measures such as the cost of not detecting an attack, which may also be of interest. For instance, the IDS evaluation methodology of NSS labs (NSS Labs, 2010) advocates the comparison of IDSes by taking into account the costs associated with IDS operation at a given operating point.

18.2.3 Measurement Methodology

Under measurement methodology, we understand the specification of the security mechanism properties that are of interest (e.g., attack detection accuracy, capacity) as well as the specification of the employed workloads and metrics for evaluating a given property. In Sections 18.2.1 and 18.2.2, we presented a workload and metric systematization with respect to their characteristics (e.g., workload content, metric aspect, metric form). In this section, we systematize different, commonly evaluated security mechanism properties. We also indicate the applicability of different workload and metric types with respect to their inherent characteristics. Thus, we round up and finalize the evaluation design space.

We identify the following security mechanism properties as most commonly evaluated in studies: attack detection, resource consumption, capacity, and performance overhead. In Table 18.3, we provide a more fine-granular systematization of these properties. Next, we briefly discuss current methodologies for evaluating the properties listed in Table 18.3.

18.2.3.1 Attack Detection

This property is relevant for evaluating IDSes since these security mechanisms feature attack detection. We classify the attack detection property of IDSes into four relevant categories: (1) attack detection accuracy (attack detection accuracy under normal working conditions, that is, in presence of mixed workloads); (2) attack coverage (attack detection accuracy under ideal conditions, that is, in the presence of attacks without any benign background activity); (3) resistance to evasion techniques; and (4) attack detection speed. In Table 18.3, we provide an overview of the workload

Table 18.3: Design space—measurement methodology

Measurement methodology	Workloads	Metrics	
Variability point	Variability point	Variability point	
[Property]	[Content]	[Aspect]	[Form]
Attack detection			
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, composite
Attack detection speed	Mixed	Performance related	/
Resource consumption			
CPU consumption Memory consumption Network consumption Hard disk consumption	Pure benign	Performance-related	/
Performance overhead	Pure benign	Performance-related	/
Capacity			
Workload processing capacity	Pure benign	Performance-related	/

and metric requirements for evaluating these properties. For instance, in contrast to the case of evaluating the attack detection accuracy, if one is interested in evaluating the attack coverage of an IDS, only pure malicious workloads and (basic) metrics that do not contain measures of false alerts would be required.

When it comes to evaluating the attack detection ability of an IDS, the detection of novel, unseen attacks is of central interest. Thus, the security research community has invested efforts in designing various anomaly-based detection techniques, a process that is still underway (e.g., Avritzer et al. (2010) in 2010 designed system for intrusion detection that uses performance signatures, Raja et al. (2012) in 2012 leveraged statistical patterns for detection of network traffic abnormalities). Since it is practically unfeasible to execute a workload that contains unseen attacks in order to train anomaly-based IDSes, in such cases, researchers use benign workloads that are considered normal whereby any deviation from such workloads is assumed as malicious. This assumption, denoted as "closed world" assumption, is considered unrealistic by Witten et al. (2011). They argue that real-life situations rarely involve "closed worlds." Thus, measurement methodologies that follow this assumption might yield unrealistic results. Furthermore, the prioritization of the attack detection properties of IDSes, with respect to their importance in case of limited available resources, is gaining increasing attention. Due to limited resources (e.g., lack of various malicious workloads), security researchers currently tend to evaluate only one or two of the attack detection properties (Table 18.3). Thus, to obtain the highest benefits from evaluation efforts, a proper prioritization of these properties is in order.

Sommer and Paxson (2010) provide an interesting insight on this matter, stating that resistance to evasion techniques is a stimulating research topic, but of limited importance from a practical perspective since most of the real-life attacks perform mass exploitation instead of targeting particular IDS flaws in handpicked target platforms. We tend to agree with this statement, which is also supported by many reports. As an example, an IBM X-Force report, that is, the 2012 Mid-Year Trend and Risk Report (IBM, 2012), states that the greatest portion of system exploitations are due to automated SQL injection attacks. Thus, similarly to Sommer and Paxson (2010), we argue that a representative workload on a global scale would contain a very small amount of evasive attacks, which decreases the priority of evaluating resistance to evasion techniques in case of limited resources.

18.2.3.2 Resource Consumption

Resource consumption is evaluated by using workloads that are considered normal for the environment in which the evaluated security mechanism is deployed; that is, such workloads should not exhibit extreme behavior in terms of intensity or in terms of the exercised hardware components (i.e., CPU-intensive, memory-intensive). Dreger et al. (2008) show that the resource consumption of many IDSes is often sensitive to the workload behavior. Thus, in order to avoid unrealistic and irrelevant resource consumption observations, one must be assured that the workloads used in evaluation experiments are representative of the target deployment environment.

There are mainly two approaches for evaluating resource consumption: black-box testing and white-box testing. The black-box testing is fairly simplistic since one measures the resource consumption of the evaluated security mechanism as resource consumption of a single entity that operates in a given environment (e.g., the resource consumption of the process of a host-based IDS in an operating system). Although practical, this approach does not provide insight into the resource consumption of the individual components of the security mechanism under test. Such insight is important for optimizing the configuration of the mechanism. To the contrary, the white-box testing usually assumes the use of a model that decomposes the mechanism under test; that is, it abstracts individual mechanism components and estimates the respective resource consumption. Dreger et al. (2008) construct an IDS model that can estimate CPU and memory consumption of an IDS with a relative error of 3.5%. Maybe the greatest benefit of the model-based white-box testing approach is that it can be used to predict resource consumption for varying workloads. The model of Dreger et al. (2008) assumes orthogonal IDS decomposition; that is, it does not model the relations between individual IDS components. Although it would be of great scientific interest to devise a model of a security mechanism that supports inter-component relations, it would require extensive modeling due to the great architectural complexity of modern mechanisms. Alternatively, one may opt for instrumentation of the code of the evaluated security mechanism, making it possible to capture the resource demands of the individual mechanism components. However,

this approach might be unfeasible in case the mechanism under test is not open-source or if it has a complex codebase.

18.2.3.3 Performance Overhead

Performance overhead is evaluated by using workloads that do not exhibit extreme behavior in terms of intensity but are extreme in terms of the exercised set of hardware resources; that is, depending on the evaluated security mechanism, an overhead evaluation experiment may consist of five independent experiments, where in each experiment, one executes a task whose workload is CPU-intensive, memory-intensive, disk I/O-intensive, network-intensive, or mixed. We provided an overview of such tasks in Section 18.2.1. The execution of these tasks is performed twice, once with the mechanism under test being inactive and once with it being active. The differences between the measured task execution times reveal the performance overhead caused by the operation of the mechanism.

18.2.3.4 Capacity

Workload processing capacity is evaluated by using workloads that exhibit extreme behavior in terms of intensity; that is, their intensity increases over time. The goal is to identify a specific workload intensity after which the workload processing performance of the evaluated security mechanism degrades. Similar to resource consumption, capacity may be evaluated using a black-box or a white-box testing approach. With white-box testing, typically multiple live tests that target specific components of the evaluated security mechanism are used. Hall and Wiley (2002) propose a methodology consisting of individual tests for measuring the packet flow, the packet capture, the state tracking, and the alert reporting components of network IDSes. Although such tests enable identification of workload processing bottlenecks in a security mechanism, they require time-consuming experimentation.

In addition to investigating the individual properties of security mechanisms listed in Table 18.3, security researchers are often interested in evaluating trade-offs between these properties. For instance, Hassanzadeh and Stoleru (2011) propose an IDS for resource-constrained wireless networks with a focus on achieving an optimal trade-off between network performance, power consumption, and attack detection effectiveness. Also, Doddapaneni et al. (2012) analyze the trade-off between the attack detection efficiency and the energy consumption of security mechanisms for wireless networks. Due to the increasing complexity and the enhanced detection abilities of modern security mechanisms, the set of requirements considered to be crucial for an effective mechanism operation (e.g., low resource consumption, low performance overhead) is also growing. Thus, simple measurement methodologies, such as the evaluation of a single mechanism property in isolation, are normally insufficient. We observe that currently many research efforts, such as the ones that we previously mentioned, are focusing on evaluating relationships between a small

set of properties. This trend is justified given the various evaluation requirements and the great number of (often insurmountable) challenges to satisfy them.

18.3 Concluding Remarks

There are three inter-related points in the planning of every evaluation study: (1) *goals* of the study; (2) existing *approaches* to realize the set goals (i.e., approaches for generating workloads and for measuring performance metrics, see Section 18.2); and (3) *requirements* that need to be met. Under the goals of an evaluation study, we understand the properties of a security mechanism that one aims to evaluate. Besides the desired extensiveness of an evaluation study, the selection of mechanism properties for evaluation is normally done by considering the *design objectives* and the *target deployment environments* of the mechanisms under test.

We now discuss the requirements that have to be met for the different approaches for evaluating security mechanisms discussed in Section 18.2. We emphasize that the ability of an evaluator to satisfy the requirements that we present significantly affects both the planning and the execution of an evaluation experiment. We systematize requirements for evaluating security mechanisms as follows:

- *Availability of required resources*, mainly related to the generation or adaptation of workloads. There are three major types of resources:
  - *Financial resources*: For instance, the financial costs of building a testbed that scales to realistic production environments are typically significant (Section 18.2.1.6);
  - *Time resources*: For instance, when an exploit database is manually assembled, the attack script collection process and the adaptation of collected attack scripts to exploit vulnerabilities of the target victim environment may be very time-consuming (Section 18.2.1.3); and
  - *Human resources*: For instance, the amount of human resources that one has available for labeling attacks in traces is a key deciding factor whether the traces can be labeled in a time-efficient manner (Section 18.2.1.5).
- *Access to confidential data:* This requirement applies when real-world production traces are used. Organizations are often unwilling to share operational traces with security researchers, or with the public in general, because of privacy concerns and legal issues (Section 18.2.1.5).
- *Availability of knowledge* about:
  - *The architecture and inner working mechanisms of the security mechanism under test*: For instance, when the IDS property resistance to evasion techniques is evaluated, the decision about which evasion techniques should be applied is based on knowledge of the workload processing mechanism of the tested IDS and of the decision-making process of the IDS for labeling an activity as benign or malicious (Section 18.2.3.1);

- *The characteristics of the employed workloads*: For instance, information about the attacks used as malicious workloads (e.g., time of execution of the attacks) must be known in order to calculate any security-related metric (Section 18.2.2); and
- *The implications of different behavior exhibited by the security mechanism under test*: For instance, the cost of the IDS missing an attack must be known in order to calculate the expected cost metric (Section 18.2.2).

The requirements mentioned above often cannot be fully satisfied. This is understandable given the big investment of resources that typically needs to be made. We observed that in case of limited resources, sacrifices are often made in:

- *The representativeness or scale of the employed workloads*: An example is the typically low number of attack scripts used in evaluation studies (Section 18.2.1.3) and
- *The number of evaluated properties of security mechanisms* (Section 18.2.3).

Trade-offs made between the quality of evaluations and the invested resources should be clearly stated when reporting results from evaluation studies so that the results can be interpreted in a fair and accurate manner. We emphasize that robust techniques for evaluating security mechanisms are essential not only to evaluate specific mechanisms but also as a driver of innovation in the field of computer security by enabling the identification of issues and the improvement of existing security mechanisms.

References

Allalouf, M., Ben-Yehuda, M., Satran, J., and Segall, I. (2010). "Block Storage Listener for Detecting File-Level Intrusions". In: *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST 2010)*. (Incline Village, NV, USA). IEEE Computer Society: Washington, DC, USA, pp. 1–12 (cited on p. 396).

Alserhani, F., Akhlaq, M., Awan, I. U., Cullen, A. J., and Mirchandani, P. (2010). "MARS: Multi-stage Attack Recognition System". In: *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010)*. (Perth, WA, Australia). IEEE Computer Society: Washington, DC, USA, pp. 753–759 (cited on p. 402).

Arlat, J., Costes, A., Crouzet, Y., Laprie, J.-C., and Powell, D. (1993). "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems". *IEEE Transactions on Computers*, 42(8). IEEE: Piscataway, New Jersey, USA, pp. 913–923 (cited on p. 397).

Avritzer, A., Tanikella, R., James, K., Cole, R. G., and Weyuker, E. J. (2010). "Monitoring for Security Intrusion using Performance Signatures". In: *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering (ICPE 2010)*. (San Jose, CA, USA). ACM: New York, NY, USA, pp. 93–104 (cited on p. 411).

Axelsson, S. (2000). "The Base-Rate Fallacy and the Difficulty of Intrusion Detection". *ACM Transactions on Information and System Security*, 3(3). ACM: New York, NY, USA, pp. 186–205 (cited on p. 408).

Bharadwaja, S., Sun, W., Niamat, M., and Shen, F. (2011). "Collabra: A Xen Hypervisor Based Collaborative Intrusion Detection System". In: *Proceedings of the 2011 Eighth International Conference on Information Technology: New Generations (ITNG 2011)*. (Las Vegas, NV, USA). IEEE Computer Society: Washington, DC, USA, pp. 695–700 (cited on pp. 394, 399).

Carreira, J., Madeira, H., and Silva, J. G. (1998). "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers". *IEEE Transactions on Software Engineering*, 24(2). IEEE Computer Society: Washington, DC, USA, pp. 125–136 (cited on p. 397).

Debar, H., Dacier, M., Wespi, A., and Lampart, S. I. (1998). *An Experimentation Workbench for Intrusion Detection Systems*. Tech. rep. RZ 2998. IBM T.J. Watson Research Center (cited on p. 395).

Doddapaneni, K., Ever, E., Gemikonakli, O., Mostarda, L., and Navarra, A. (2012). "Effects of IDSs on the WSNs Lifetime: Evidence of the Need of New Approaches". In: *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2012)*. (Liverpool, UK). IEEE: Piscataway, New Jersey, USA, pp. 907–912 (cited on p. 413).

Dreger, H., Feldmann, A., Paxson, V., and Sommer, R. (2008). "Predicting the Resource Consumption of Network Intrusion Detection Systems". In: *Proceedings of the 2008 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2008)*. (Annapolis, MD, USA). Vol. 36. ACM SIGMETRICS Performance Evaluation Review. ACM: New York, NY, USA, pp. 437–438 (cited on p. 412).

Dumitras, T. and Shou, D. (2011). "Toward a Standard Benchmark for Computer Security Research: The Worldwide Intelligence Network Environment (WINE)". In: *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS 2011)*. (Salzburg, Austria). ACM: New York, NY, USA, pp. 89–96 (cited on p. 397).

Dunlap, G. W., King, S. T., Cinar, S., Basrai, M. A., and Chen, P. M. (2002). "ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay". In: *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*. (Boston, Massachusetts). Vol. 36. ACM SIGOPS

Operating Systems Review. ACM: New York, NY, USA, pp. 211–224 (cited on p. 396).

Durães, J. A. and Madeira, H. (2003). "Definition of Software Fault Emulation Operators: A Field Data Study". In: *Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN 2003)*. (San Francisco, CA, USA). IEEE Computer Society: Washington, DC, USA, pp. 105–114 (cited on p. 397).

– (2006). "Emulation of Software Faults: A Field Data Study and a Practical Approach". *IEEE Transactions on Software Engineering*, 32(11). IEEE Computer Society: Washington, DC, USA, pp. 849–867 (cited on p. 397).

Durst, R., Champion, T., Witten, B., Miller, E., and Spagnuolo, L. (1999). "Testing and Evaluating Computer Intrusion Detection Systems". *Communications of the ACM*, 42(7). ACM: New York, NY, USA, pp. 53–61 (cited on p. 408).

Engen, V., Vincent, J., and Phalp, K. (2011). "Exploring Discrepancies in Findings Obtained with the KDD Cup'99 Data Set". *Intelligent Data Analysis*, 15(2). IOS Press: Amsterdam, The Netherlands, pp. 251–276 (cited on p. 402).

Fonseca, J. and Vieira, M. (2008). "Mapping Software Faults with Web Security Vulnerabilities". In: *Proceedings of the 2008 IEEE International Conference on Dependable Systems and Networks (DSN 2008)*. (Anchorage, AK, USA). IEEE Computer Society: Washington, DC, USA, pp. 257–266 (cited on p. 398).

Fonseca, J., Vieira, M., and Madeira, H. (2009). "Vulnerability and Attack Injection for Web Applications". In: *Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009)*. (Lisbon, Portugal). IEEE Computer Society: Washington, DC, USA, pp. 93–102 (cited on pp. 398, 399).

Gad El Rab, M. (2008). "Evaluation des systèmes de détection d'intrusion". PhD thesis. Toulouse, France: Université Paul Sabatier - Toulouse III (cited on p. 397).

Gaffney, J. and Ulvila, J. (2001). "Evaluation of Intrusion Detectors: A Decision Theory Approach". In: *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. (Oakland, CA, USA). IEEE: Piscataway, New Jersey, USA, pp. 50–61 (cited on pp. 406, 408).

Görnitz, N., Kloft, M., Rieck, K., and Brefeld, U. (2009). "Active Learning for Network Intrusion Detection". In: *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence (AISec 2009)*. (Chicago, Illinois, USA). ACM: New York, NY, USA, pp. 47–54 (cited on p. 396).

Griffin, J. L., Pennington, A., Bucy, J. S., Choundappan, D., Muralidharan, N., and Ganger, G. R. (2003). *On the Feasibility of Intrusion Detection inside Workstation Disks*. Tech. rep. CMU-PDL-03-106. Pittsburgh, PA, USA: Parallel Data Laboratory, Carnegie Mellon University (cited on p. 395).

Gu, G., Fogla, P., Dagon, D., Lee, W., and Skorić, B. (2006). "Measuring Intrusion Detection Capability: An Information-Theoretic Approach". In: *Proceedings of*

the 2006 ACM Symposium on Information, Computer and Communications Security (ASIACCS 2006). (Taipei, Taiwan). ACM: New York, NY, USA, pp. 90–101 (cited on pp. 406, 408, 409).

Hall, M. and Wiley, K. (2002). "Capacity Verification for High Speed Network Intrusion Detection Systems". In: *Recent Advances in Intrusion Detection—5th International Symposium, RAID 2002—Proceedings*. (Zurich, Switzerland). Ed. by A. Wespi, G. Vigna, and L. Deri. Vol. 2516. Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, pp. 239–251 (cited on p. 413).

Hassanzadeh, A. and Stoleru, R. (2011). "Towards Optimal Monitoring in Cooperative IDS for Resource Constrained Wireless Networks". In: *Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN 2011)*. (Maui, HI, USA). IEEE: Piscataway, New Jersey, USA, pp. 1–8 (cited on p. 413).

IBM (2012). *IBM X-Force 2012 Mid-Year Trend and Risk Report* (cited on p. 412).

Jin, H., Xiang, G., Zhao, F., Zou, D., Li, M., and Shi, L. (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 2009)*. (Suwon, Korea). ACM: New York, NY, USA, pp. 391–399 (cited on p. 395).

Jin, H., Xiang, G., Zou, D., Wu, S., Zhao, F., Li, M., and Zheng, W. (2013). "A VMM-based Intrusion Prevention System in Cloud Computing Environment". *The Journal of Supercomputing*, 66(3). Springer US: New York, NY, USA, pp. 1133–1151 (cited on pp. 394, 395).

Katcher, J. (1997). *PostMark: A New File System Benchmark*. Tech. rep. TR3022. Sunnyvale, USA: Network Appliance (cited on p. 395).

Komlodi, A., Goodall, J. R., and Lutters, W. G. (2004). "An Information Visualization Framework for Intrusion Detection". In: *CHI'04 Extended Abstracts on Human Factors in Computing Systems*. (Vienna, Austria). ACM: New York, NY, USA, p. 1743 (cited on p. 393).

Kruegel, C., Valeur, F., and Vigna, G. (2005). *Intrusion Detection and Correlation—Challenges and Solutions*. Vol. 14. Advances in Information Security. Springer US: New York, NY, USA (cited on p. 390).

Leita, C., Dacier, M., and Massicotte, F. (2006). "Automatic Handling of Protocol Dependencies and Reaction to 0-day Attacks with ScriptGen Based Honeypots". In: *Recent Advances in Intrusion Detection—9th International Symposium, RAID 2006, Proceedings*. Ed. by D. Zamboni and C. Kruegel. Vol. 4219. Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, pp. 185–205 (cited on p. 403).

Lombardi, F. and Di Pietro, R. (2011). "Secure Virtualization for Cloud Computing". *Journal of Network and Computer Applications*, 34(4). Academic Press Ltd.: London, UK, pp. 1113–1122 (cited on pp. 392, 395, 396, 404).

Maynor, D., Mookhey, K. K., Cervini, J., Roslan, F., and Beaver, K. (2007). *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. Syngress Publishing: Rockland, MA, USA (cited on p. 396).

McHugh, J. (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). ACM: New York, NY, USA, pp. 262–294 (cited on p. 402).

Mell, P., Hu, V., Lippmann, R., Haines, J., and Zissman, M. (2003). *An Overview of Issues in Testing Intrusion Detection Systems*. NIST Interagency/Internal Report (NISTIR) 7007. Gaithersburg, MD, USA: National Institute of Standards and Technology (NIST) (cited on pp. 395, 396).

Meng, Y. and Li, W. (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 2012)*. (Liverpool, UK). IEEE: Piscataway, New Jersey, USA, pp. 223–230 (cited on pp. 392, 404).

Milenkoski, A. (2016). "Evaluation of Intrusion Detection Systems in Virtualized Environments". PhD thesis. Würzburg, Germany: University of Würzburg (cited on p. 390).

Milenkoski, A., Payne, B. D., Antunes, N., Vieira, M., Kounev, S., Avritzer, A., and Luft, M. (2015). "Evaluation of Intrusion Detection Systems in Virtualized Environments Using Attack Injection". In: *Research in Attacks, Intrusions, and Defenses—18th International Symposium, RAID 2015—Proceedings*. (Kyoto, Japan). Ed. by H. Bos, F. Monrose, and G. Blanc. Vol. 9404. Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg (cited on pp. 390, 399).

Milenkoski, A., Vieira, M., Kounev, S., Avritzer, A., and Payne, B. D. (2015). "Evaluating Computer Intrusion Detection Systems: A Survey of Common Practices". *ACM Computing Surveys*, 48(1). ACM: New York, NY, USA, 12:1–12:41 (cited on p. 390).

MIT Lincoln Laboratory (1999). *1999 DARPA Intrusion Detection Evaluation Dataset*. URL: https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset (cited on p. 402).

Mohammed, N., Otrok, H., Wang, L., Debbabi, M., and Bhattacharya, P. (2011). "Mechanism Design-Based Secure Leader Election Model for Intrusion Detection in MANET". *IEEE Transactions on Dependable and Secure Computing*, 8(1). IEEE Computer Society: Washington, DC, USA, pp. 89–103 (cited on pp. 392, 404).

NSS Labs (2010). *Network Intrusion Prevention System Test Methodology v.6.1*. http://www.nsslabs.com/assets/Methodologies/nss2010 (cited on pp. 393, 410).

Patil, S., Kashyap, A., Sivathanu, G., and Zadok, E. (2004). "FS: An In-Kernel Integrity Checker and Intrusion Detection File System". In: *Proceedings of the 18th USENIX Conference on System Administration (LISA 2004)*. (Atlanta, GA). USENIX Association: Berkeley, CA, USA, pp. 67–78 (cited on p. 395).

Raja, N., Arulanandam, K., and Rajeswari, B. (2012). "Two-Level Packet Inspection Using Sequential Differentiate Method". In: *Proceedings of the Intl. Conference on Advances in Computing and Communications (ICACC 2012)*. (Cochin, Kerala, India). IEEE: Piscataway, New Jersey, USA, pp. 42–45 (cited on pp. 402, 411).

Reeves, J., Ramaswamy, A., Locasto, M., Bratus, S., and Smith, S. (2012). "Intrusion Detection for Resource-Constrained Embedded Control Systems in the Power Grid". *International Journal of Critical Infrastructure Protection*, 5(2). Elsevier Science: Amsterdam, The Netherlands, pp. 74–83 (cited on pp. 395, 396).

Riley, R., Jiang, X., and Xu, D. (2008). "Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing". In: *Recent Advances in Intrusion Detection—11th International Symposium, RAID 2008—Proceedings*. (Cambridge, MA, USA). Ed. by R. Lippmann, E. Kirda, and A. Trachtenberg. Vol. 5230. Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, pp. 1–20 (cited on pp. 395, 396).

Rodríguez, M., Salles, F., Fabre, J.-C., and Arlat, J. (1999). "MAFALDA: Microkernel Assessment by Fault Injection and Design Aid". In: *Dependable Computing—EDCC-3—Third European Dependable Computing Conference—Proceedings*. Ed. by J. Hlavička, E. Maehle, and A. Pataricza. Vol. 1667. Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, pp. 143–160 (cited on p. 397).

Shiravi, A., Shiravi, H., Tavallaee, M., and Ghorbani, A. A. (2012). "Toward developing a systematic approach to generate benchmark datasets for intrusion detection". *Computers and Security*, 31(3). Elsevier Science: Amsterdam, The Netherlands, pp. 357–374 (cited on p. 403).

Shirey, R. (1999). *Internet Security Glossary*. Internet Engineering Task Force, RFC 2828, http://tools.ietf.org/html/draft-shirey-security-glossary-01 (cited on pp. 390, 397).

Sinha, S., Jahanian, F., and Patel, J. M. (2006). "WIND: Workload-aware INtrusion Detection". In: *Recent Advances in Intrusion Detection—9th International Symposium, RAID 2006—Proceedings*. (Hamburg, Germany). Ed. by D. Zamboni and C. Kruegel. Vol. 4219. Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, pp. 290–310 (cited on pp. 392, 404).

Sommer, R. and Paxson, V. (2010). "Outside the Closed World: On Using Machine Learning For Network Intrusion Detection". In: *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. (Oakland, California). IEEE Computer Society: Washington, DC, USA, pp. 305–316 (cited on pp. 401–403, 412).

Sperotto, A., Sadre, R., Vliet, F., and Pras, A. (2009). "A Labeled Data Set for Flow-Based Intrusion Detection". In: *IP Operations and Management—9th IEEE International Workshop, IPOM 2009—Proceedings*. (Venice, Italy). Ed. by G. Nunzi, C. Scoglio, and X. Li. Vol. 5843. Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, pp. 39–50 (cited on pp. 401, 402).

Stallings, W. (2002). *Cryptography and Network Security: Principles and Practice*. 3rd edition. Pearson Education: London, UK (cited on p. 390).

Stolfo, S., Fan, W., Lee, W., Prodromidis, A., and Chan, P. (2000). "Cost-Based Modeling for Fraud and Intrusion Detection: Results from the JAM Project". In: *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*. (Hilton Head, SC, USA). Vol. 2. IEEE: Piscataway, New Jersey, USA, pp. 130–144 (cited on p. 406).

University of California (1998). *KDD Cup 1999 Data: Data Set Used for The Third International Knowledge Discovery and Data Mining Tools Competition*. URL: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (cited on p. 402).

Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. 3rd edition. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann: Burlington, MA, USA (cited on p. 411).

Wright, C., Cowan, C., Smalley, S., Morris, J., and Kroah-Hartman, G. (2002). "Linux Security Modules: General Security Support for the Linux Kernel". In: *Proceedings of the 11th USENIX Security Symposium*. (San Francisco, California). USENIX Association: Berkeley, CA, USA, pp. 17–31 (cited on p. 396).

Yu, S. and Dasgupta, D. (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 2011)*. (Paris, France). IEEE: Piscataway, New Jersey, USA, pp. 17–24 (cited on p. 402).

Zhang, Y., Wang, H., Gu, Y., and Wang, D. (2008). "IDRS: Combining File-level Intrusion Detection with Block-level Data Recovery based on iSCSI". In: *Proceedings of the Third International Conference on Availability, Reliability and Security (ARES 2008)*. (Barcelona, Spain). IEEE Computer Society: Washington, DC, USA, pp. 630–635 (cited on p. 395).