SVED: Scanning, Vulnerabilities, Exploits and Detection

Hannes Holm
Swedish Defence Research Agency (FOI),
Olaus Magnus väg 42,
Linköping, Sweden
Email: hannes.holm@foi.se

Teodor Sommestad
Swedish Defence Research Agency (FOI),
Olaus Magnus väg 42,
Linköping, Sweden
Email: teodor.sommestad@foi.se

***Abstract*—This paper presents the Scanning, Vulnerabilities, Exploits and Detection tool (SVED). SVED facilitates reliable and repeatable cyber security experiments by providing a means to design, execute and log malicious actions, such as software exploits, as well the alerts provided by intrusion detection systems. Due to its distributed architecture, it is able to support large experiments with thousands of attackers, sensors and targets. SVED is automatically updated with threat intelligence information from various services.**

1. Introduction

Cyber attacks are frequent and a serious problem for organizations and individuals. Numerous models, tools and metrics have been proposed for the purpose of measuring and managing cyber security. In particular, there are a wide range of approaches for assessing cyber vulnerabilities (e.g., using attack graphs, as in MulVAL [1] or CySeMoL [2], [3]) and detecting malicious behavior (e.g., by mapping alerts to an attack model, as in SnIPS [4] and the solution in [5]).

Unfortunately, few of these approaches have been examined using empirical data. For example, to this date, there are few published empirical tests of measurement methods [6], and most tests of solutions for detecting malicious behavior is performed on a criticized dataset produced in the late 1990's [7]. This is primarily due to the difficulty for researchers to obtain observational data from live networks and systems. Such data are typically deemed too sensitive for public research. Furthermore, because it is next to impossible to completely distinguish malicious activity from benign activity in an operational network, such data do not offer a ground truth to run security tests against.

The dire need for more tests is illustrated by the disappointing results of the few studies that examine the reliability of different cyber security models and metrics. For example, [8] found attack graphs inaccurate; [9] found that system-level vulnerability metrics show little correlation with the actual time required to compromise a system; [10] found little correlation between the severity of a software vulnerability and the actual exploitation of it in practice.

These issues have been acknowledged by the scientific community. For example, a roadmap for experimental cybersecurity research is presented in [11]. This roadmap stress, among other things, that the community needs shared, validated models and tools that help researchers to rapidly design meaningful experiments and test environments.

Fortunately, recent advances in virtualization technology and the advent of large-scale laboratory computer environments, often called cyber ranges, offer alternative means of obtaining empirical data and conducting experiments. In cyber ranges, systems and software can be configured similarly to the real world, and attacks can be carried out in a controlled manner without influencing any operational business. Cyber ranges can therefore be used to generate data suitable for testing both methods for vulnerability assessment and methods for detecting malicious behavior. This is demonstrated by a number of published papers that use data generated in cyber ranges, for instance, [8], [9], [12].

However, as the data in cyber ranges is generated synthetically, its characteristics may be different from that observed in an operational environment. This validity problem can be decomposed into the issue of valid cyber environments (e.g., software versions and network configurations), valid background traffic (e.g., users' web browsing habits) and valid cyber attacks (e.g., software exploits and payloads). Advances are needed in all these areas to enforce ecological validity. This paper focuses on the generation of cyber attacks.

A common means of performing security experiments in cyber ranges is to use red teams who attempt to compromise assets (see e.g., [13]). While red teams offer a degree of realism in the sense that their attacks are diverse and intelligently chosen, it is generally expensive to involve them. Also, the authors' experience show that red team members are more interested in compromising assets than recording their actions. Thus, the resulting data may be unreliable and the corresponding experiments difficult to reproduce. As a consequence, researchers and practitioners turn to different methods that aim to simulate attacker actions in a reproducible and reliable manner.

This paper presents the Scanning, Vulnerabilities, Exploits and Detection tool (SVED). SVED enables automated or manual low-effort design of attack plans. It also enables distributed automatic execution of sequences of attack steps as well as recording all the activity by carried out actions.

Finally, SVED has interfaces for automatically interacting with popular intrusion detection tools such as Snort in order to get their alerts in real-time.

2. Related work

As previously noted, attacks in an experiment can be injected manually, e.g., by a red team. However, from an experimental perspective, this has obvious limitations with respect to scalability, reliability, replicability and reuse. Such issues are avoided if attacks are scripted so that they are automatically executed by a computer. This section offers an overview of research on automating attacks for the purpose of security experimentation.

ARENA [14] and RINSE [15] are two platforms that support cyber security experimentation. Their descriptions suggest that they support injections of various types of attacks, allow experiments of scale, specify experiments in a way that makes them replicable and allow both extensions and reuse. However, similar to many other cyber security testbeds (e.g., [16], [17]), both ARENA and RINSE are based on syntehetical simulations of computer networks, and not real systems. This may cause problems with regards to fidelity and realism in cyber security testing as many software vulnerabilities (e.g., buffer overruns) exists because of poorly implemented treatment of system calls. To accurately predict the effects of an attack that targets such a vulnerability, there is a need to include the actual sofware stack.

Experimental platforms based on virtualization technology (e.g., VirtualBox, VMware or Emulab) suffer less from the issue with realism as they allow the execution of the entire software stack with the exception of hardware drivers. In combination with offensive tools, virtualized machines allow realistic testing of most types of cyber security attributes. Available offensive tools include, but are not limited to, Metasploit, w3af, Immunity Canvas, Cain and Abel, and Core Impact. Some of these tools integrate vulnerability scanning capabilities with automated exploit capabilities. This supports component testing and simple injects in cyber security exercises. However, many cyber security attacks involve intelligently chosen sequences of attack steps, e.g., when obtained privileges are used to access additional resources. To the authors' knowledge, none of the available offensive tools support automation of this sort, nor do they offer frameworks for creating complex attack plans involving multiple tools. Also, as cyber attacks attempt to alter the state of the target system, valid consecutive tests of the same system requires restoring it to a previously known stable state (e.g., using virtual snapshots) before the execution of an exploit. To our knowledge, none of the current tools have such functionality.

There are experimental frameworks that integrate offensive tools with cyber range environments to allow realistic and controlled experimentation. DCAFE [18] is such an experimental framework. In DCAFE software agents are deployed on virtual machines and instrumented to run cyber attacks as well as collect data. However, DCAFE is currently only available as an early prototype that seems to lack the extensibility one would expect from a security experimental testbed. For instance, it only supports Windows 7 victim machines and Kali Linux attacker machines. A similar platform is desribed in [19]. This platform has the purpose of producing datasets for intrusion detection system testing and the tool Metasploit is used to execute attacks against machines according to predefined attack plans. While Metasploit contains a wide range of actions and has APIs to several significant third-party tools (e.g., nmap and OpenVAS), far from all relevant kinds of actions are covered by it. Also, [19] is only able to model server-side attacks, which is an issue as many attacks today are conducted against clients. Nevertheless, both DCAFE and the platform presented by [19] illustrate the potential for cyber security experimentation where offensive tools are integrated into cyber ranges. The present work is one step further in that direction.

3. Design and implementation

This chapter describes the design and implementation of the Scanning, Vulnerabilities, Exploits and Detection tool (SVED). SVED consists of five main components: threat intelligence, a designer, an executioner, attacker/sensor agents and a logger. The relations between these components can be seen in Figure 1. These components enable creating well-designed attack plans (Section 3.2), executing these plans (Section 3.4 - 3.5) and logging the result from experiments (Section 3.6). The entire codebase, not counting third-party libraries, consists of 6400 lines of Python code and 6100 lines of Javascript/HTML code.

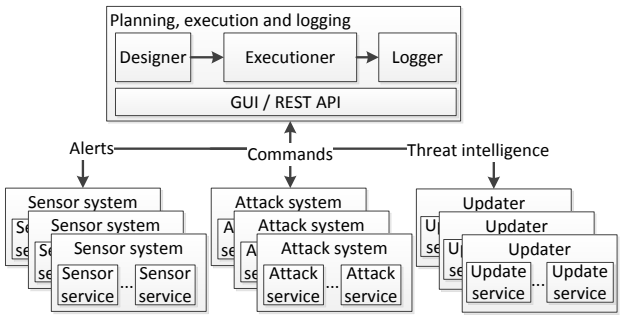

Figure 1: An overview of the architecture of SVED.

SVED is currently used in a cyber range called Cyber Range and Training Environment (CRATE). CRATE is briefly described in Section 3.1.

3.1. Cyber Range and Training Environment

CRATE is a cyber range built and maintained by the Swedish Defence Research Agency (FOI). It has been used for a number of research studies [20]. This section only provides a brief description of CRATE and the interested

reader is referred to [20] for more detailed description. Physically, CRATE consist of approximately 750 servers, a number of switches and various auxiliary equipment (e.g., for remote access). During an experiment these servers are instrumented with between five and twenty VirtualBox machines of various types. Instrumentation is straightforward for any type of operating system and application software that VirtualBox can handle. In addition to the typical enterprise software, simplified versions of social network services, industrial control systems and search engines are available.

Instrumentation is either scripted or managed using a web-based graphical user interface called CRATEWeb. This web interface allows the researcher to configure operating systems, application software, firewall configurations, network topologies, users, windows domains, and more. A set of scripts then deploy virtual machines that match the design to the server park.

Benign user activity can be scripted through bash commands and a Windows-based automation tool called AutoIT. These "bots" use historical user activity from operational systems as a basis for their behaviour.

3.2. Threat intelligence

SVED is updated with general and specific information relevant to cyber security experiments in the cyber range. General information include vulnerabilities (from the US National Vulnerability Database (NVD)), exploits and vulnerable software (from the Exploit Database), network intrusion detection signatures (from Snort Talos and Emerging Threats rule sets), anti-malware signatures (from the Symantec Security Response) and incidents (from the Vocabulary for Event Recording and Incident Sharing (VERIS) database). Specific information about assets in the cyber range (such as network interfaces and user accounts) is provided by CRATEWeb (see Section 3.1). This information is enriched by another application called AutoVAS which autonomously runs parallell authenticated automated vulnerability scans using OpenVAS on systems in the cyber range. SVED polls data from these services periodically with the aid of a time delta that enables extracting revised, new or removed data.

This information fascilitate better manual as well as automated designs of attack graphs (see section 3.3). We are currently working on probabilistic attack designs that employ threat intelligence information to more realistically simulate attacker behaviour (see section 5).

3.3. Designer

The designer is used to create attack graphs. This can be accomplished either through a web-based GUI or scripted through a REST API. The user interface is illustrated in Figure 2. Two canvases constitute the majority of the view.

The left canvas describes assets of interest for an experiment. These assets are either targets, attackers or sensors. *Targets* are subjected to malicious actions from attackers, e.g., a Windows-based workstation or a Debian-based firewall. *Attackers* have services that can be used to inject malicious commands during an experiment. Several types of services are currently supported, such as Metasploit, OpenVAS and a general service that can facilitate a range of commands (e.g., flooding attacks, cracking password hashes, sending email or running VirtualBox commands). The framework has been built to be easily extended with new services and commands. For instance, adding a new command to the general service only requires a few lines of code. *Sensors* have functionality to detect malicious activity carried out against targets. The framework currently only supports Snort. However, as for attack services, the framework was built to be easily extended with new sensors.

The user loads a designated set of assets from the backend database by specifying which targets, attack systems and sensors that should be included in an experiment.

The right canvas describes the defined attack steps and their connections. SVED supports four types of actions: reconnaissance, exploits, shellcode and auxiliary. With the exception of auxiliary actions, these are part of most cyber attacker behaviour models. *Auxiliary* actions are a kind of "glue" that help to facilitate valid execution of actions without generating any activity that could tamper with the experiment result (such as intrusion alerts). Example auxiliary actions include restoring a target to a previous snapshot using the VirtualBox API and querying the database of an attack service regarding any identified vulnerabilities. *Reconnaissance* actions increase the information known about targets through, e.g., vulnerability scanning or port scanning. *Exploit* actions attempt to provide unauthorized access to targets. SVED contains functionality to execute exploits that target both clients (e.g., Adobe Reader) and servers (e.g., Windows SMB), where client-side attacks typically are administrated through e-mail or directly executed on targets using the VirtualBox API. Both local and remote access vectors are supported. *Shellcode* actions run commands on compromised machines, e.g., to increase persistence by disabling anti-malware, to cover the tracks of the attacker, or to extract sensitive data through FTP.

This categorization is, however, purely for usability purposes. From an architecture/database point of view, all action types correspond to a class called *abstract action*. This "base template" of an action is defined through a list of settings, where each setting is responsible for describing its use. Some settings are specific (e.g., CVE-2008-4250 requires the name of the targeted SMB user name); some settings are shared between actions in a category (e.g., scans can either run until they are completed or until a defined amount of time as passed); some settings are shared between all actions. The latter includes different timers, e.g., regarding start and stop. The user is required to describe the states of some settings (e.g., the target address of a server exploit) whereas other settings are voluntary as they have valid default states. The user is able to filter between different types of settings for a selected action. It is possible to define the states of settings for multiple actions simultaneously. Abstract actions are loaded by quering attacker systems that

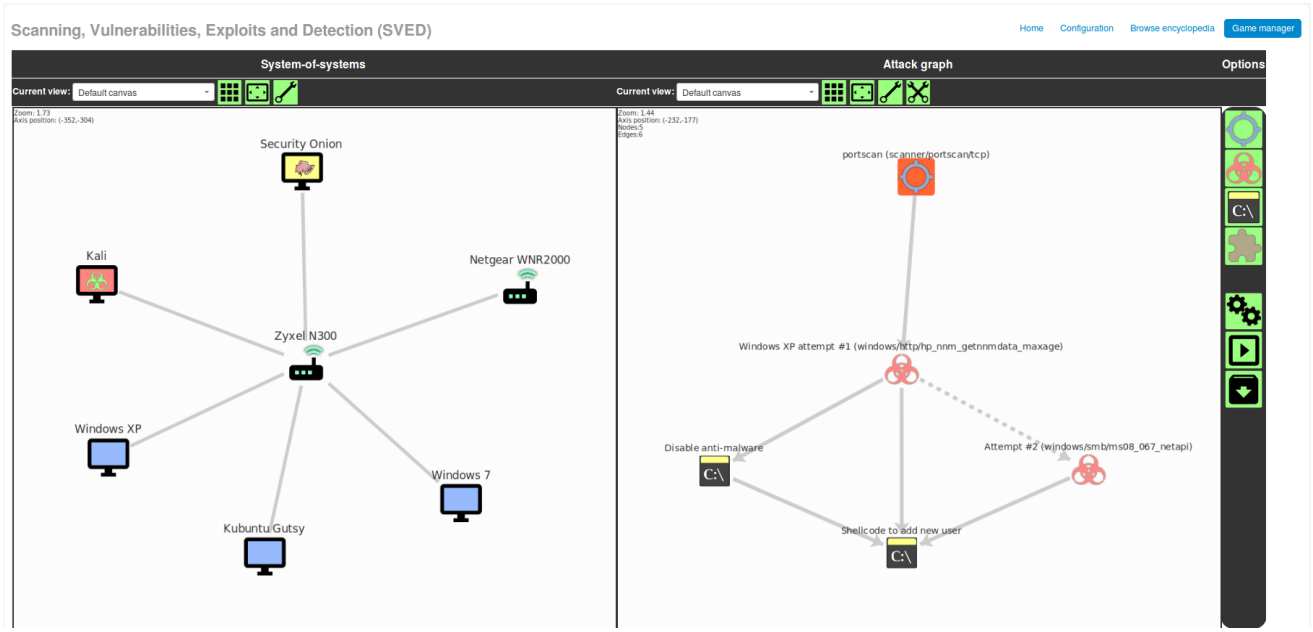

Figure 2: An overview of the designer interface in SVED.

are present in an experiment design. From a user perspective, this is done by pressing a button in the GUI. This means that each attack service is responsible for translating all actions in its arsenal to abstract actions upon request.

An experiment design consists of *action instances*, which are unique instances of abstract actions. When an action instance is created, it keeps a reference to its parent abstract action and clones all settings that can be altered by the user. There are several helper functions that reduce the effort required to create attack graphs:

1) It is possible to query the database for attacks that are relevant to a particular target. This is facilitated by relating software and systems to vulnerabilities and vulnerabilities to actions (CPEs to CVEs, and CVEs to actions).
2) One experiment can be copied and injected into another. Through this method, a user is able to create small experiment stubs, or templates, such as a simulation of a computer worm, which then can easily be loaded into other experiments.
3) There is an automated planning algorithm that maps exploits from a set of attack systems against a set of targets. This planner utilizes a sub-set of the threat intelligence data (see section 3.2) by coupling software vulnerability information to exploits. Test validity is enforced by a set of pre-exploit actions that first restore the target to a known vulnerable snapshot, then verify connectivity, and finally that the target software is in a valid state.

Actions can be connected in three ways (see Figure 2):

- execute if successful (light grey line)
- execute if fail (light grey line with short dashes)
- always execute (dark grey line with long dashes)

Here, success and failure means different things for different actions. For example, an exploit is successful if a session can be obtained on the targeted system, whereas an auxiliary action that checks whether a particular vulnerability has been identified on a specific system is successful if this condition is met. In practice this enables relating actions using AND/OR gates of arbitrary complexity.

In both canvases, the user chooses which elements that should be visible by coupling elements to viewspoints and choosing which viewpoint that should be active.

3.4. Executioner

When a designated attack graph is executed (the play button in Figure 2), the executioner first makes sure that the logger service is running at its designated address. It then builds the complete attack graph, where each attack step is provided directions to the logger as well as directions to its required attack services. Tests are then conducted to ensure that all required attack services are operational. If the attack graph and its dependencies are valid, the executioner continues by checking which included sensor services that are operational. The executioner then notifies each operational sensor that an experiment is beginning and which address that the logger is located at, configuring the logger in an appropriate alert forwarding mode.

The executioner then adds all actions that do no not have preceeding attack steps to the list of currently active attack steps and proceeds to step through the entire attack graph, administrating commands and reading their results. An experiment is considered completed either when a user aborts

it or when all actions that could be attempted have been attempted (which not necessarily means that all branches in a graph have been traversed). When this is the case, all involved sensors are notified, causing them to stop reporting alerts to the logger. At this point, the executioner reports to the logger that the experiment has been completed.

The user is able to view the activity of live and previous experiments in a graphical interface.

3.5. Agents

Each attack system and sensor system has one or more services that are used to run commands or report alerts. Each such service is required to have functionality that enables the service to fulfill its purpose as well as interact with the executioner and the logger. As-is, this functionality is enabled by a combination of Python code and third-party code (such as the Metasploit RPC server). For a sensor service, this functionality includes being able to act on overall game events (e.g., start and stop) and status updates (e.g., if the service is operational), as well as report alerts to the logger. For an attack service, it includes acting on overall game commands and sending status updates (both regarding the status of the service and the status of its currently running actions), executing commands (e.g., run a certain exploit), as well as notify the designer regarding the actions in its arsenal.

3.6. Logger

The logger is a server application that is responsible for receiving and managing incoming reports from attack steps and sensors. Its schema is inspired by IODEF, but revised to handle the additional properties relevant from an attackers perspective. It can easily be modified to support other standards such as STIX, CybOX or CEF.

4. Example experiment

This chapter describes an example experiment with SVED. The experiment is simple by design in order to make it easily understandable. As SVED has a distributed architecture, the performance of the executioner is next to independent of the complexity of the attack graph. In other words, it is possible to run experiment with thousands of unique attackers, sensors and victims.

The example experiment consists of the systems and attack steps presented in Figure 2. A Kali Linux 2.0 system with a Metasploit RPC server and a utility server is used to execute the actions against a Windows XP machine. A Security Onion 12.04 configured with a Snort sensor service (with the latest Emerging Threats rule set) is present to detect attacks and report these to SVED. The attack graph includes a portscan (`scanner/portscan/tcp`), a web server attack (`windows/http/hp_nnm_getnnmdata_maxage`), a shellcode that disables anti-malware protection, a server attack against SMB (`windows/smb/ms08_067_netapi`), and a shellcode that adds a new user on the victim.

The complete result with all events is available for download[1]. Of the five actions, the portscan, the SMB attack and the second shellcode were successful; the web server attack failed and the first shellcode was never executed. A total of 315 events were recorded by the logger. 305 of these events were produced by actions and 10 by alerts from Snort (see Table 1). Three Snort alerts triggered for the SMB attack and seven for the port scan. In other words, no alerts were given for the web server attack or the shellcode.

While this experiment is too small and unrealistic to draw any general conclusions regarding the effectiveness of Snort, it serves to illustrate the use of SVED: Unlike any experiment involving red teams, the actions would be carried out and recorded in exactly the same way every time the experiment is run.

Table 1: Snort alerts and their respective actions.

Time	Priority	Snort ID	Triggered by
15:41:50	2	2010935	`scanner/portscan/tcp`
15:41:52	2	2010936	`scanner/portscan/tcp`
15:42:43	2	2010937	`scanner/portscan/tcp`
15:43:13	2	2010938	`scanner/portscan/tcp`
15:43:45	2	2010939	`scanner/portscan/tcp`
15:43:56	2	2002910	`scanner/portscan/tcp`
15:43:59	2	2002911	`scanner/portscan/tcp`
15:46:16	3	2102465	`windows/smb/ms08_067_netap`
15:46:16	1	2009247	`windows/smb/ms08_067_netap`
15:46:16	1	2009247	`windows/smb/ms08_067_netap`

5. Summary and future work

This paper presented the SVED, a framework that facilitates reliable and repeatable cyber security experiments by providing a means to plan, execute and log malicious actions as well as intrusion detection alerts in real-time.

While SVED is a fully functional tool, there are plenty of work left. In particular, its automated planning capabilities are very crude. One means of enhancing its automated planning capabilities would be to add an interface to an attack graph tool such as MulVAL [1]. However, these kinds of tools do not consider many aspects important to attacker decision making [21]. In particular, they typically do not model reconnaissance or shellcode actions. Thus, such an implementation likely requires first extending the attack graph tool itself. Also, as an attack graph merely contains hypothetical adverserial actions with uncertain outcome [8], they are not perhaps not suited for cyber security experimentation.

An arguably better method would be to let the user decide start- and end-conditions of an experiment, and then allow the executioner to autonomously interact with victims in the system graph in real-time with a designated amount of initial information similar to that of a real attacker. This would create a worm-like behaviour that is operated and controlled by the executioner. For example, a simple

1. ftp://download.iwlab.foi.se/sved/

dynamic execution pattern could be to first run a portscan, then a vulnerability scan, then an exploit and then pivot from the compromised system to new locations. Each of these actions could be connected through database queries to examine discovered host addresses, services, vulnerabilities and gained sessions. This method also requires implementing an algorithm that is able to intelligently choose an action whenever there are multiple options available. In practice this requires a search algorithm believed to well reflect an attacker profile (e.g., depth-first or breadth-first search methods) and a function that can be used to weigh each edge (option) according likelihood of usage. The latter could for example be a combination of the risk of being detected by a sensor in combination with the likelihood of success as well as the value of success in relation to other alternatives. Indicators regarding the exploitation aspects of such a model are given by Hoffman [22], who propose extensions to the Core Impact exploit planner.

A second future work is to extend SVED with the capability to interact with more security tools that execute and detect malicious actions. As SVED was built with this in mind it should be a minor task to add a new tool to its scope. New actions should however preferrably be configured to comply with a standardized exploit framework as this would serve to decrease the overall tool complexity. A reasonable candidate for such a framework is the Metasploit framework (which SVED already is complient with) due to its wide-spread usage and regular updates. The coverage of Metasploit is, however, limited in relation to other public exploit databases such as Exploit DB – there are roughly 3000 modules in Metasploit and 35000 modules in Exploit DB. Research that automatically make actions Metasploit-compatible would thus be valuable.

Last but not least, there is a need to use SVED to produce data that can be used by cyber security researchers and practitioners when evaluating the effectiveness of different metrics, methods and tools that are thought to enhance cyber security by some means. It would be valuable to share such results as well as their experimental designs in some common repository.

Acknowledgments

This project has received funding from the European Unions FP7 research and innovation programme under grant agreement No 603993.

References

[1] X. S. Ou, S. Govindavajhala, and A. Appel, "MulVAL: A logic-based network security analyzer," in *Proceedings of the 14th conference on USENIX Security Symposium-Volume 14*. USENIX Association, 2005, p. 8.

[2] T. Sommestad, M. Ekstedt, and H. Holm, "The Cyber Security Modeling Language – A Tool for Vulnerability Assessments of Enterprise System Architectures," *Systems Journal, IEEE*, vol. 7, no. 3, pp. 363–373, 2013.

[3] H. Holm, K. Shahzad, M. Buschle, and M. Ekstedt, "P^2CySeMoL: Predictive, Probabilistic Cyber Security Modeling Language," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2014.

[4] S. C. Sundaramurthy, L. Zomlot, and X. Ou, "Practical ids alert correlation in the face of dynamic threats," in *Proceedings of the International Conference on Security and Management*, 2011.

[5] X. Qin and W. Lee, "Attack plan recognition and prediction using causal networks," *Proceedings - Annual Computer Security Applications Conference, ACSAC*, pp. 370–379, 2004.

[6] V. Verendel, "Quantified security is a weak hypothesis," in *Proceedings of the 2009 workshop on New security paradigms workshop - NSPW '09*. New York, New York, USA: ACM Press, 2009, pp. 37–49.

[7] J. McHugh, "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*, vol. 3, no. 4, pp. 262–294, nov 2000.

[8] T. Sommestad and F. Sandström, "An empirical test of the accuracy of an attack graph analysis tool," *Information and Computer Security*, vol. 23, no. 5, pp. 516–531, nov 2015.

[9] H. Holm, M. Ekstedt, and D. Andersson, "Empirical Analysis of System-Level Vulnerability Metrics through Actual Attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 6, pp. 825–837, nov 2012.

[10] L. Allodi, W. Shim, and F. Massacci, "Quantitative assessment of risk reduction with cybercrime black market monitoring," in *Security and Privacy Workshops (SPW), 2013 IEEE*. IEEE, 2013, pp. 165–172.

[11] D. Balenson, L. Tinnel, and T. Benzel, "Cybersecurity Experimentation of the Future (CEF): Catalyzing a New Generation of Experimental Cybersecurity Research," SRI International, Tech. Rep., 2015.

[12] J. Mirkovic, P. Reiher, C. Papadopoulos, A. Hussain, M. Shepard, M. Berg, and R. Jung, "Testing a Collaborative DDoS Defense In a Red Team/Blue Team Exercise," *IEEE Transactions on Computers*, vol. 57, no. 8, pp. 1098–1112, aug 2008.

[13] D. Levin, "Lessons learned in using live red teams in IA experiments," in *Proceedings of DARPA Information Survivability Conference and Exposition*, vol. 1. IEEE, 2003, pp. 110–119.

[14] K. C. Costantini, "Development of a cyber attack simulator for network modeling and cyber security analysis," Ph.D. dissertation, Rochester Institute of Technology, 2007.

[15] M. Liljenstam, J. Liu, D. M. Nicol, Y. Yuan, G. Yan, and C. Grier, "Rinse: the real-time immersive network simulation environment for network security exercises (extended version)," *Simulation*, vol. 82, no. 1, pp. 43–59, 2006.

[16] A. Futoransky, F. Miranda, J. Orlicki, and C. Sarraute, "Simulating cyber-attacks for fun and profit," in *proceedings of the 2nd international conference on simulation tools and techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, p. 4.

[17] M. E. Kuhl, J. Kistner, K. Costantini, and M. Sudit, "Cyber attack modeling and simulation for network security analysis," in *Proceedings of the 39th Conference on Winter Simulation: 40 years! The best is yet to come*. IEEE Press, 2007, pp. 1180–1188.

[18] G. Rush, D. R. Tauritz, and A. D. Kent, "Dcafe: A distributed cyber security automation framework for experiments," in *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*. IEEE, 2014, pp. 134–139.

[19] A. Abou El Kalam, M. Gad El Rab, and Y. Deswarte, "A model-driven approach for experimental evaluation of intrusion detection systems," *Security and Communication Networks*, vol. 7, no. 11, pp. 1955–1973, 2014.

[20] T. Sommestad, "Experimentation on operational cyber security in CRATE," in *NATO STO-MP-IST-133 Specialist Meeting*, Copenhagen, Denmark, 2015, pp. 7.1–7.12.

[21] J. Yuen, "Automated Cyber Red Teaming," Cyber and Electronic Warfare Division, Defence Science and Technology Organisation, Edinburgh South Australia, Australia, Tech. Rep., 2015.

[22] J. Hoffmann, "Simulated penetration testing: From" dijkstra" to" turing test++"." in *ICAPS*, 2015, pp. 364–372.