SAIBERSOC: A Methodology and Tool for Experimenting with Security Operation Centers

DOI:
[10.1145/3491266](https://doi.org/10.1145/3491266)

Document status and date:
Published: 01/06/2022

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

SAIBERSOC: A Methodology and Tool for Experimenting with Security Operation Centers

MARTIN ROSSO, MICHELE CAMPOBASSO, GANDUULGA GANKHUYAG, and
LUCA ALLODI, Eindhoven University of Technology

In this article, we introduce SAIBERSOC (Synthetic Attack Injection to Benchmark and Evaluate the Performance of Security Operation Centers), a tool and methodology enabling security researchers and operators to evaluate the performance of deployed and operational Security Operation Centers (SOC)—or any other security monitoring infrastructure. The methodology relies on the MITRE ATT&CK Framework to define a procedure to generate and automatically inject synthetic attacks in an operational SOC to evaluate any output metric of interest (e.g., detection accuracy, time-to-investigation). To evaluate the effectiveness of the proposed methodology, we devise an experiment with $n = 124$ students playing the role of SOC analysts. The experiment relies on a real SOC infrastructure and assigns students to either a BADSOC or a GOODSOC experimental condition. Our results show that the proposed methodology is effective in identifying variations in SOC performance caused by (minimal) changes in SOC configuration. We release the SAIBERSOC tool implementation as free and open source software.

CCS Concepts: • **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**; *Systems security;*

Additional Key Words and Phrases: Cyber security operations center, SOC, performance, evaluation

1 INTRODUCTION

The growing importance of effective security monitoring solutions calls for appropriate measures of their effectiveness. The operation of **Security Operation Centers (SOCs)** is the recommended best practice to which large and medium-size enterprises rely on the detection, notification, and ultimately response to cybersecurity incidents [26, 33]. Yet, the average time for detecting an attack ranges between several weeks to years [58]. In a recent study, Kokulu et al. [26] interviewed security analysts and managers, who explicitly identify the issue of *measuring* SOC performance as one of the main obstacles towards effective detection and response operations.

The problem of measuring security (performance) is a long-standing and difficult one [21]. Measuring the performance of SOCs is no exception. Whereas numerous performance metrics exist (e.g., number of detected incidents, time to detection, and time to response) [49], it is still unclear how to obtain reliable and reproducible measures over those metrics. As incoming attacks are, by definition, unknown, a ground truth on which to base

the measurements cannot be easily defined; on the other hand, performing measurements in *in-vitro* settings limits the representativeness of those measurements in real-world settings [26, 48].

However, the problem of measuring the performance of infrastructures and systems dedicated to the detection of *rare* events of unknown magnitude is not new. For example, the ***Laser Interferometer Gravitational-Wave Observatory*** (***LIGO***) infrastructure has been built with the goal of detecting perturbations in the spacetime continuum caused by "rippling effects" of large-scale events, such as the collapse of a binary blackhole system. Despite the completely different settings and event generation processes, the challenges faced by the LIGO interferometer and SOCs are quite similar: both must detect unknown, unpredictable, and arbitrarily "small" manifestations of an event, and both must know whether the detection procedure is accurate while operating in the absence of a ground-truth on the events that ought to be detected.

This article presents SAIBERSOC, a practical and deployable solution for SOC performance evaluation based on the same working principle as the LIGO infrastructure [1]: to generate and automatically inject "events" (in our case, cyber attacks) into the detection infrastructure at random instants in time, with the purpose of evaluating, in a controlled fashion, the performance of the detection system. The SAIBERSOC —Synthetic Attack Injection to Benchmark and Evaluate the Performance of SOCs — methodology and tool can be employed by researchers and security operatives alike to evaluate the impact of various factors on SOC performance, such as analyst training, experience, and skill composition, or the relative change in effectiveness of the deployed analysis and visualization tools as the SOC configuration and processes change.

SAIBERSOC automatically generates the injected attacks relying on the MITRE ATT&CK Framework [55] and is structured over four components providing automated methods for attack definition, generation, injection, and analysis. The SAIBERSOC tool is publicly available as free and open source software (see Section 7).

To evaluate the effectiveness of the method at the core of SAIBERSOC, we design and run an experiment employing $n = 124$ students enrolled in a security course of a medium-sized technical university in Europe. The experiment relies on a real SOC infrastructure (normally monitoring network events at the M&CS department of the university) where students operate as SOC analysts. In particular, we measure detection accuracy in terms of both attack *identification* and *investigation*, injecting two attacks based on the Mirai [6] and Exim [10] scenarios over two SOC configurations (BADSOC and GOODSOC). Our evaluation shows that the SAIBERSOC methodology correctly identifies and can quantify the relative change in detection accuracy across SOC configurations and, importantly, does not "overshoot" or reveal differences where there should be none.

This paper develops as follows: Section 2 provides a background on SOC operations and discusses relevant literature. Section 3 describes the proposed methodology and Section 4 our experimental validation strategy. Experimental results follow in Section 5, we discuss our results in Section 6, and then present the SAIBERSOC software implementation in detail in Section 7. Finally, Section 8 concludes the article.

2 BACKGROUND AND RELATED WORK

2.1 Attack Frameworks

Defensive operations against increasingly sophisticated attacks require a deep understanding of how adversaries conduct offensive operations, how threats unfold and how security breaches escalate into major incidents. The MITRE PRE-ATT&CK [11], MITRE ATT&CK [55], and Cyber Kill Chain [29] frameworks aim at building a knowledge base of adversary TTPs (Tactics, Techniques, and Procedures) from real-world observations, grouping them in a taxonomy of attack stages to describe the anatomy of any attack. The extensive and detailed enumeration of the attack techniques provided by the frameworks and stages can be, roughly, grouped in four phases: *Reconnaissance*, *Exploitation*, *Delivery,* and *Control* (Table 1). *Reconnaissance* groups data gathering operations performed by the adversary, ranging from subject information for the development of social engineering attacks, to probing the network surface of the infrastructure to derive paths of compromise. During *Exploitation*, the attacker evaluates the measured attack surface (vulnerabilities, misconfigured software, phishing opportunities) and creates or

Table 1. Attack Phases

Phase	Description
Reconnaissance	Techniques to research, identify and select targets using active or passive reconnaissance.
Exploitation	Techniques employed by attackers to gain initial control over (vulnerable) target systems.
Delivery	Techniques resulting in the transmission of the weaponized object to the targeted environment.
Control	Techniques used by attackers to communicate with controlled systems in a target network.

acquires the attacks (vulnerability exploits, phishing artefacts, malware, ...) to perform information leak, lateral movement (e.g., inside the organization), command execution, and/or privilege escalation. *Delivery* covers the vectors adopted from the adversary to drop malicious software that will act as a foothold for further actions. This includes spear phishing and supply chain compromise. *Control* includes the techniques adopted by the adversary to control the system and reliably execute commands on it, compromising system integrity, confidentiality and availability. This can be achieved via remote management interfaces, custom software and web services.

2.2 Automatic Anomaly, Attack and Breach Simulations

Chaos Tests are commonly used by cloud service providers such as Netflix, Google, and Facebook to actively test and train service outages on their production infrastructure [20, 22, 27]. By testing and training disaster recovery, those companies increase their resiliency against selected service failures. Chaos Monkey [34], for example, is a software tool developed by Netflix to simulate exceptional events in their infrastructure—similar to the *LIGO* testing infrastructure mentioned in Section 1.

Similar concepts and tools exist in the domain of security and incidence response teams to simulate partially or fully automated attack and breach events [8]. These tools are usually based on a library of potential attacker activities that implement or map to individual techniques from attack frameworks, such as the MITRE ATT&CK framework. Given access to a system (simulating an initial point of compromise), these software tools try to utilize attacker actions from their library to move through the network and escalate their privilege, often with a specific set of goals. To do so, breach simulation tools come with a control server containing the logic and planning and clients or agents that allow the control server to execute commands and attack activities on (compromised) systems. Compared to automatic breach simulation tools, SAIBERSOC generates reproducible and easily modifiable experiments, while breach simulation tools are built to make ad-hoc decisions. Furthermore, the SAIBERSOC methodology focuses on SOC operations, rather than on threat mitigation and analysis. An example of (open source) repository for attacker activities is the Atomic Red Team repository [44]; examples for two (open source) attack and breach simulation tools are the MITRE CALDERA [31] or Infection Monkey [18]. An overview and comparison of various attack and breach simulation tools is provided in [8].

More broadly, cyber-range experiences and simulation platforms (see e.g., [28, 40, 50]) have been developed recently to foster training and awareness in organizations, as well as to support red and blue teaming exercises, across all domains, including governmental and military [12]. The application domain of cyber-ranges varies from security education [41] to defensive applications such as monitoring and event correlation [57]. SAIBERSOC represents an additional step in the integration of cyber-threat simulations in detection and monitoring processes; by operating directly at the monitoring level, SAIBERSOC allows operators to investigate not only technical capabilities of the monitoring technology (e.g., to tune rules in an intrusion detection system), but processes (e.g., incident investigation, reporting) too.

2.3 Security Monitoring Operations

Tools and procedure to support security analysts in incident response and network monitoring (such as **Network Intrusion Detection Systems** (**NIDS**) and **Security Information and Event Management** (**SIEM**)) are at the core of modern security monitoring in operational settings. SOCs are the center of monitoring operations in medium or large organizations, either internally or outsourced to service providers [26, 33].

SOCs are organized hierarchically in (generally three) tiers [33], where analysts with different skills and expertise monitor the network activity and take action against a threat. Tier 1 is the first frontier where alerts are investigated by analysts, identifying possible threats among the non-significant ones and prioritizing them. Identified threats are escalated to Tier 2, where more qualified analysts with forensics and incident response skills correlate the information with threat intelligence to identify threat actors. Tier 2 is in charge to determine a strategy for containment, remediation and recovery. If the threat targets business critical operations, Tier 3 analysts identify and develop tailored responses to the identified threats and attack patterns.

In general, incident investigation can be split in two phases [33]: *Attack identification*, where Tier 1 analysts evaluate incoming alerts to identify possible attacks; *Attack investigation*, where Tier 1 and Tier 2 analysts investigate the identified attack over its phases (ref. Table 1) to, for example, identify victims, attack timing, payloads, and propagation. Once these two phases are complete, the attack is reported and responded to, depending on the service level agreement at which the SOC operates.

The large amount of alerts and potential security events detected in a SOCs make it impossible, operationally to "investigate everything" [49]: the more relevant are the alerts generated by a SOC (e.g., through accurate use-cases), the fewer "false positives" a (Tier 1) analyst will have to investigate before passing the baton on to higher tiers (i.e., Tier 2 and 3).

However, *measuring* how SOC operations respond to changes in the SOC configuration (e.g., a refined use-case, a different alert investigation process, and new rulesets) remains an open and critical challenge [26], in terms of analyst competences as well as for metric definition and measurement [17, 48, 49, 60].

2.4 The Human Factor in Threat Identification and Incidence Response

Whereas technology and automation takes an important role in security monitoring, human operators are at the core of most follow-up security-related process such as alert investigation and incident reporting/resolution. Thus, it is essential to provide operators with the required resources (including time, and proper training) to efficiently process threat information generated by the automated processes in a SOC. Additionally, processes and software tools have to be designed around human behavior and decision-making to support a human's decision-making process and to anticipate and compensate for typical human flaws and limitations, where possible. Along this line of research, much work focuses on the characteristics and limitations of decision support systems in the context of threat detection.

By nature, every human operator makes mistakes. In the case of SOC analysts, mistakes can result in the omission of evidence, wrong decisions, and potentially increased damage caused by an undetected cyber attack. Dietrich et al. used a survey to investigate administrators' view on security-relevant misconfigurations and how these could potentially be avoided [13]. SAIBERSOC can help to acquire data to identify what type of mistakes are common and what causes them (e.g., lack of knowledge/experience/tools/…).

Other research focused on investigating factors influencing the operational performance of (human) security analysts. For example, Roden presents SOC alerts to the analysts and asks them to either escalate them or classify them as false alarms. His study focuses on the relation between alert frequency and operators performance when confronted with different number of true-positive and false-positive intrusion alerts [45]. Similarly, Wang et al. conducted an experiment asking operators to detect malfunctions and cyber attacks in a simulated industrial control system scenario. Not only does this study present a cognitive decision-making model, but it also investigates into variables that impact human performance, including, for example, the analysts' stress levels [59].

Nyre-Yu uses cognitive task analysis to investigate what tasks are performed and what skills are required, but also how automated systems can support SOC analysts at their work [35]. Andrade and Yoo, as well, investigate which part of the work can be automated to support security analysts [5]. Furthermore, Andrade et al. conducted a literature review to pinpoint ways software tools and process can improve analysts' work [4]. Mullins et al. designed a prototype dashboard and evaluated its usefulness to facilitate situational awareness in a team of security analysts [32].

The SAIBERSOC methodology and software can enable further research into the human aspect of alert identification, analysis, investigation, evaluation, as well as human-machine interaction between SOC analysts and the underlying software systems powering the SOC. SAIBERSOCallows the creation of realistic, reproducible experimental setups and the ability to introduce fine-grained alterations to established scenarios by replacing individual aspects of a (simulated) cyber attack.

Finally, an inadequate working environment can not only support human error but also demotivate employees and cause burnout [51]. To research burnout among SOC analysts, Sundaramurthy et al. embedded five students into two operational SOCs (one corporate SOC and one SOC operating at a University) and collected their observations and experiences in an anthropological study [51]. Among other factors, Sundaramurthy et al. conclude that contradicting expectations for stakeholders and poor performance metrics can create pressure that may lead to a vicious cycle, to frustration and finally burnout. Additional factors maybe be exhausting but repetitive and boring work (i.e., false positive alerts) [51], or inadequate software tools Haber and Bailey. By injection realistic attacks, SAIBERSOC can help to train analysts and assess the current operational performance of the SOC by capturing real-life data from the system.

2.5 SOC Performance Evaluation

Albeit attempts have been made in trying to tackle these problems, the scientific literature proposing methods to measure SOC performance is still limited. Effectiveness of a SOC depends on both the qualities of the tools adopted, their configurations and capabilities, and on skills of the personnel. For what concerns the correlation between human capabilities and a SOC's performance, a few studies have been made [52, 53]. In [52], Sundaramurthy et al., they trained three students to work as SOC analysts and embedded them in three different SOCs. While working as operational components, they were required to produce reports on their observations and meet with an anthropologist involved in the research. By comparing reports and interviews, it was possible to draw conclusions on the aspects that may be more promising to the effectiveness of a SOC and which not, in terms of its architecture, experience of analysts, nature and degree of their interactions, workflow for incident reports and the quality of work itself for the employee. Kokulu et al. [26] investigate through interviews to both SOC analysts and SOC managers about technical and non-technical issues of a SOC; from their interviews emerge how often SOCs fail to provide substantial support against specific types of attacks, overloading analysts with low-quality threat intelligence, long reports and logs, and more. Albeit evidences are supported by interviews, the authors point out that metrics to measure security quantitatively and qualitatively are rudimentary [26]. Jacobs [25] proposes a method to systematically and quantitatively evaluate SOC's maturity by identifying a set of capabilities that a SOC embodies, i.e., log analysis, event correlation, incident management, threat identification and reporting, scoring each of them [24, 25]. The SOC aspects analyzed derive from a number of industrial security management and control frameworks, including ISO 27000 series [23] and SANS Critical Controls and each of them is scored with respect to their maturity; however, the score attribution is left as an expert-driven task, rather than a data-driven measurement. A simulation-optimization approach is proposed by Shah et al. [48], where they identified some of the causes that negatively affect the throughput of a SOC in terms of efficiency. They propose the *Time to Analyze Alert* (**TTA**) metric to evaluate the efficiency of a SOC by measuring the time that goes from the alert generation to its analysis. By monitoring SOC's TTA, Shah et al. are able to generate advice for the SOC Manager, suggesting live corrective actions towards a desired benchmark. In subsequent

Table 2. Architectural Requirements

R-ID	Requirement	Description	AC-1	AC-2	AC-3	AC-4
R-1	Independence	The proposed solution should be independent of the specific SOC implementation and monitored environment.	✓	✓		✓
R-2	Versatility	The framework must work with all types of attack scenarios in scope of the SOC, regardless of their complexity.	✓	✓		
R-3	Realism	The attack injection must be seamlessly integrated to the operative environment of the analyst. This includes IP addresses, local software configurations, and network setups.		✓	✓	
R-4	Traceability	The framework must be able to link all SOC reactions (i.e., a report) to the initial stimulus or attack that caused it.		✓		✓

Table 7 in the Appendix details the mapping to the architectural components.

work Shah et al. [49] also model the problem as a simulation-optimization problem and evaluated it from data derived from a simulated SOC, to obtain a set of metrics that allow the optimization of some of a SOC capabilities; however, a procedure to test the resulting SOC performance in real-world scenarios remains to be identified and tested.

Research gap. Whereas current research has focused mostly on the identification of metrics [49] and procedures [25, 48, 49] to evaluate SOC performance, an empirical method capable of capturing the complexity of a SOC operation (including alert configuration, analyst capabilities, etc.) has yet to be proposed and validated [26].

3 THE SAIBERSOC SOLUTION

To address this gap, we propose SAIBERSOC, a method and solution to perform systematic SOC performance evaluations through automated attack injection. Our solution is composed of four Architectural Components (AC-{1..4}) constituting of a library of (attack) traces (AC-1), and components to generate (AC-2), inject (AC-3), and report (AC-4) the synthetic attacks.

SAIBERSOC can be used to inject attacks into a SOC during operation, or alternatively in a "virtual" SOC testing environment where all network traffic is replaced by pre-recorded background traffic. Once the attack has been detected, the SOC analysts perform the investigation following their normal procedure. The type of injected attack as well as its timing, targets and sources, and velocity are unknown to the SOC analysts. Finally, the analyst's attack report is automatically evaluated against the ground truth obtained from the attack assembly phase. A more detailed description of the SAIBERSOC architecture can be found in Section 3.2, we present our software implementation in Section 7.

3.1 Solution Requirements

To guide the development of the proposed solution, we pose a list of requirements that a real-world SOC performance evaluation framework should fulfill. A summary of the identified requirements is provided in Table 2.

*Independence (*R-1*).* To measure and compare SOC performance, the framework must apply to all SOCs, regardless of how they are implemented. Furthermore, the results must be reproducible, regardless of who uses the framework. Those are fundamental requirements for any measure or assessment [39]. Therefore, a solution must be able to work with various SOC configurations and implementations.
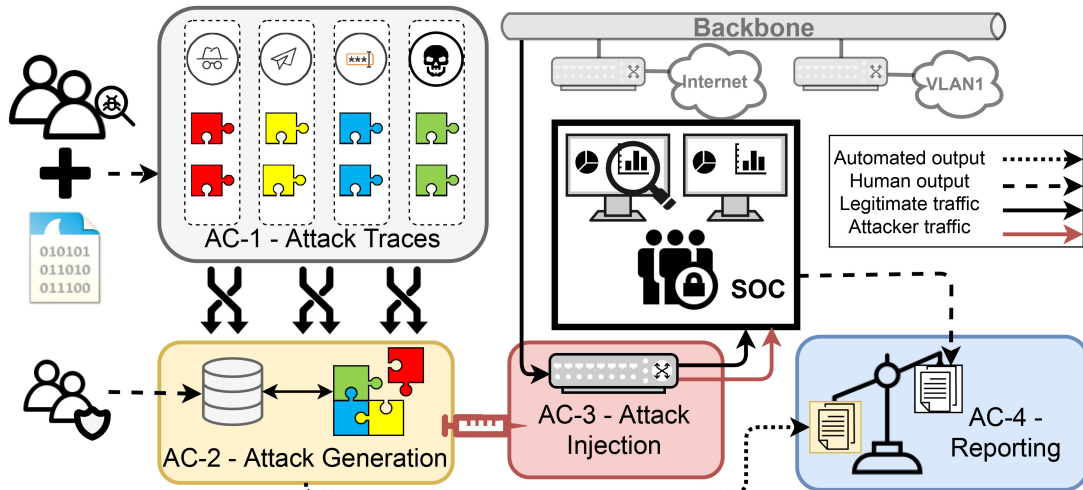
The platform uses (network) traces characterizing the 'building blocks' of an attack (AC-1), and (re-)combines them to generate new attacks (AC-2). The generated attacks are then injected into the network (AC-3). The SOC analysts perform and report on the investigation results, which can be checked (AC-4) against the known ground-truth (derived from the attacks generated in AC-2).

Fig. 1. Schematic depiction of the SAIBERSOC solution.

*Versatility (*R-2*).* To be able to create meaningful measures covering the whole scope of the SOC, all aspects of a potential attack must be covered. Thus, a metric should cover the complete value-space to assure meaningful results reflecting real-world performance [26]. Because attackers continuously adapt their behavior, a solution must be capable of simulating past, current, as well as future attacks in realistic ways to be able to measure real-world security.

*Realism (*R-3*).* Real-world security is influenced by numerous factors, sometimes in unpredictable ways. For example, wrong presumptions about context or human behavior in unforeseen situations leads to alleged security measures and therefore a false sense of security [2, 9]. This can be avoided by assessing real-world performance, i.e., a setup that is as close as possible) to reality. In case of a SOC this means measuring a real SOC (including software and configuration), realistic work environments, and realistic attacks. It is therefore imperative that the evaluation is performed under real or realistic operational conditions (e.g., the analysts do not know the attacks in advance, and the injected attacks are realistic).

*Traceability (*R-4*).* The framework must be able to link SOC reactions i.e., reports, to the attacks that caused those reactions in the SOC. Only if both input and output are known, a meaningful evaluation of the SOC performance can be made by comparing the *expected* output to the realized output.

3.2 The SAIBERSOC Solution Architecture

Figure 1 provides a schematic depiction of the architecture behind the proposed SAIBERSOC solution. Table 2 reports the mapping of the architectural components to the identified requirements. A running example of implementation of the proposed workflow is reported in Table 8 in the appendix and discussed below.

*3.2.1 Attack Traces (*AC-1*).* All attack simulations or scenarios deployed in SAIBERSOC are constructed by combining multiple pre-defined attack traces. An attack trace consists of a set of network packets (e.g., stored in a *pcap* file) that reproduces the network traffic generated by an attack during a specific attack phase. These traces can be extended to non-network events (e.g., *syslog* activity to reproduce events at the host level). To structure

traces in attack phases, we rely on the ATT&CK Framework [55] to create and store a library of different attack traces reflecting different phases of ongoing attacks (see Table 1).

The MITRE CALDERA [31] and Atomic Red Team [44] already maintain some form of an attack-database with attacker actions mapped to the ATT&CK Framework [55].

Impl. example: with reference to the example reported in Table 8, AC-1 generates a set of traces characterizing events in each phase of the ATT&CK framework; for example, *port scan(src_ip, dst_ip)* will reproduce network traffic (to be parameterized with *src_ip, dst_ip* associated to the *Reconnaissance* phase of the MITRE ATT&CK framework (ref. Table 1). Similarly, *expl_cve()* will reproduce traces corresponding to the exploitation of a known vulnerability (e.g., as derived from an exploit on ExploitDB [36].

*3.2.2 Attack Generation (*AC-2*).* Multiple attack traces are combined to generate fully-fledged attack simulations. Attacks are composed following the ATT&CK framework [55] (i.e., reconnaissance happens before exploit delivery, which happens before command and control communication, which happens before data exfiltration).

The CALDERA framework is already capable of automatically generating meaningful attacks from attack traces [31].

In this step, the generic attack traces are matched to the specific environment monitored by the SOC (e.g., IP ranges, exploits vs vulnerable systems). This process can be automated by relying on asset information available to any SOC during normal operation.

The result of this phase is a *ready-to-inject* attack simulation.

The "ground-truth" of the simulation (e.g., attacker's IP addresses(es)) can be extracted from the parameterized attack traces composing the attack. Once re-parameterized (i.e., victim IP addresses matching the range of a monitored network and system functionality), attack scenarios can be reused the SOC or organization's network.

Impl. example: with reference to Table 8, a sequence of (attack) traces is selected and parameterized. In this example, the IP of the attacker and the receiving victim system(s) are identified (*src_ip,dst_ip*), as well as the exploited vulnerability (*cve*), and protocols used for the command and control (*proto*). The parametrized attack traces are then composed following the MITRE ATT&CK framework phases; the parameters of the attack are used to generate the ground truth against which the analysts' reports will be checked in AC-4.

*3.2.3 Attack Injection (*AC-3*).* Attack simulations retrieved from AC-2 are injected in the network traffic monitored by the SOC. This can be either executed on the live infrastructure, or on a virtual "shadow network" separate from the operative environment. In the latter setting, the entire (virtual) network traffic is recorded and later forwarded to the Network Intrusion Detection sensors employed by the SOC. The attack simulation can be merged seamlessly into (real-world) traffic streams either on a SOC sensor or on network infrastructure devices (i.e., network switches or routers). To make simulations as realistic as possible, (optional) background traffic that was previously recorded on site can be mixed with the attack simulation. Attack simulations can be injected, recorded, and replayed at any point in time (i.e., record network traffic from a virtual environment instead of passing to the SOC immediately). The SOC will then raise the respective events and alerts as if the injected traffic was real traffic to (or from) the monitored infrastructure.

Impl. example: In the example in Table 8 traffic is recorded and replayed into a SOC sensor as scheduled by executing the parametrized traces (e.g., *exploit_cve(cve,src_ip,dst_ip)*) (i.e., 12pm). The attack injection raises corresponding SOC alerts, then evaluated by the analyst during incident identification and investigation.

*3.2.4 Reporting (*AC-4*).* The attack reports produced by SOC analysts in response to the attack simulation are matched against the ground truth (obtained from AC-2) in AC-4. Because the exact details of the simulated attacks are known in advance, report(s) that mention specific details unique to the injected attack can be identified automatically. Using simple string matching and the knowledge-base of the SIEM, it is possible to check whether the final SOC report details the incident events and sources correctly.

Impl. example: A SOC analyst investigated the generated SOC alerts and reported the incident through a standardized template (e.g., *src_ip, cve, …*). Comparing the analyst report with the ground truth may indicate that the current SOC configuration lacks detection capabilities (e.g., undetected exploitation), or that raised alerts systematically misdirect the analyst (e.g., due to poorly defined threats or use cases).

4 EXPERIMENTAL VALIDATION

Before presenting the SAIBERSOC tool implementing the described architecture and solution, we validate the methodology behind it by testing it experimentally. To do so, we devise an experiment involving $n = 124$ M.Sc. students attending the "Security in Organizations" course of a medium-sized European Technical University and an operative SOC infrastructure deployed at the Department level.

Goal of the experiment: *to evaluate whether the proposed solution is capable of detecting (small) differences in SOC performance.*

Experimental subjects worked in groups of two and took the role of *Tier 1* security analysts to investigate (injected) attacks. Subjects were randomly assigned to the experimental treatment (consisting of a small tweak in the SOC configuration) and asked to report the results of their analysis by filling in a report template.

4.1 Experimental Infrastructure

The experiment is based on a real SOC used for research and education operating at the Mathematics and Computer Science department of the university. The SOC is operated over open-source technology and based on Security Onion [47] and Elastic Stack [14] for event correlation and analysis. During regular operation the SOC monitors real incoming and outgoing network traffic in the department. The chosen network sensors (NIDS) are Suricata [37] for security monitoring, and Bro/Zeek [38] for network flow logs. The SOC is currently limited to the monitoring of network traffic and is operated with the involvement and cooperation of the security team of the university.

4.2 Experimental Subjects

Subjects were recruited from a security in organization M.Sc. course held at the university operating the SOC. The course is mandatory for all students graduating in the security track of the M.Sc. program in Computer Science and open to students from others tracks (security = 14; computer science = 56; other = 64; total = 124). Students were asked to form groups of two to participate in the experiment and were randomly assigned to the respective treatment groups (next subsection). The final size of the experiment pool is n = 63; the term "experiment subject" in the remainder of the article will refer to the student groups and not to the single students.

4.3 Experimental Variables

Independent variable. To reproduce realistic SOC configurations we set up two analysis environments, namely GOODSOC and BADSOC. To test the effectiveness of the proposed solution, we introduce only a small change between the two configurations. To maintain the change realistic, we (1) define it around the university environment in which the SOC operates, and (2) base it only on the (de-)activation of a set of predefined rules that *do not* match the use-cases of the university.[1] Namely, as the university is an open environment, default rulesets triggering alerts related to violation of generic policies such as use of TOR or **peer-to-peer** services (**p2p**) are out of the scope of the SOC. We therefore define the following two experimental conditions:

BADSOC: baseline SOC configuration consisting of the default set of rules defined by the detection software (Suricata);
GOODSOC: baseline SOC configuration *minus* alerts related to policy violation events.

---

[1]This setup has been checked against a set of use-cases provided by the university for SOC operation.

In the BADSOC configuration 19,731 out of 27,125 present Suricata [37] rules were active by default. By deactivating 2,753 rules (14%), 16,978 active rules remain). The disabled rules[2] inform about policy violations (including the use of chat/instant messaging software and usage of the TOR network [56]).

Outcome variables. In output of the experiment, we evaluate the accuracy of the assessments made by the experiment subjects. We collect analyst output through a survey compiled by all subjects at the end of the experiment. To evaluate the subject's output under different experimental conditions, we consider the following variables: (1) the total number of reports submitted by each group: these may be related to the real injected attacks, or to other suspicious events detected by the subjects; (2) the number of submitted reports dealing with one of the simulated attacks; (3) the correctness of the submitted reports when compared against the ground truth.

4.4 Expected Outcome and Evaluation Criteria

If the proposed solution is effective in detecting changes in SOC performance triggered by (small) changes in the SOC configuration, we would expect that GOODSOC (i.e., the configuration producing fewer alerts not related to the university's use cases), lead to more accurate reports when compared to BADSOC. Reflecting normal SOC operation procedures [33], we split our evaluation over the analysis phases of *Attack identification* and *Attack investigation*.

Outcome expectation. As the modification between BADSOC and GOODSOC should only reduce the number of false alerts displayed to the analyst, leaving alerts relevant to an attack unaffected, we expect the SAIBERSOC method to highlight that:

(1) GOODSOC outperforms BADSOC for the *attack identification* phase;
(2) no significant difference between GOODSOC and BADSOC emerges for the *attack investigation* phase.

To evaluate the accuracy of the *Attack identification*, we evaluate the number of correct entries reported by the subjects in relation to: IP(s) of the attacker, IP(s) of the victims. To evaluate the accuracy of *Attack investigation*, we evaluate reported information on the reconnaissance, exploitation, and delivery and control activities (ref. Table 1). As we are interested in testing the effectiveness of the proposed solution, we are not concerned with quantifying whether our GOODSOC configuration is *significantly better than* our BADSOC configuration; differently, we are interested in evaluating whether the (albeit small) *change in the* GOODSOC *configuration can be spotted by the proposed experimental procedure* in terms of a difference in performance. To evaluate these differences, we employ a mix of non-parametric statistical tests including Fisher's Exact Test (for differences in counts across conditions), and Wilcoxon rank-sum tests (for differences in outcome distributions). The significance level is set at $\alpha = 0.05$.

4.5 Scenario Planning

We define two attack scenarios, namely Mirai and Exim, both reproducing realistic cyber attacks. The Mirai scenario is inspired by the Mirai botnet that spread around (IoT) devices with default passwords in 2016 [6]. It was designed to raise multiple alerts of varying severity (including four high-priority alerts). Due to the amount of alerts reported in the SOC, we assume it to be easier for an analyst to identify this attack (relative to the second attack, that triggers less alerts). The second attack scenario is named after the Exim mail server [15]. In this scenario the attacker uses an exploit against an outdated Exim SMTP server to gain command-line access. This scenario generates only one high-priority alert and is thus easier to overlook (compared to the Mirai scenario).

---

[2]All rules prefixed with either "ET POLICY", "ET INFO", "ET CHAT", "GPL CHAT", "ET TOR", or "SURICATA" were disabled (see "better-soc.sh" in the artifact repository).
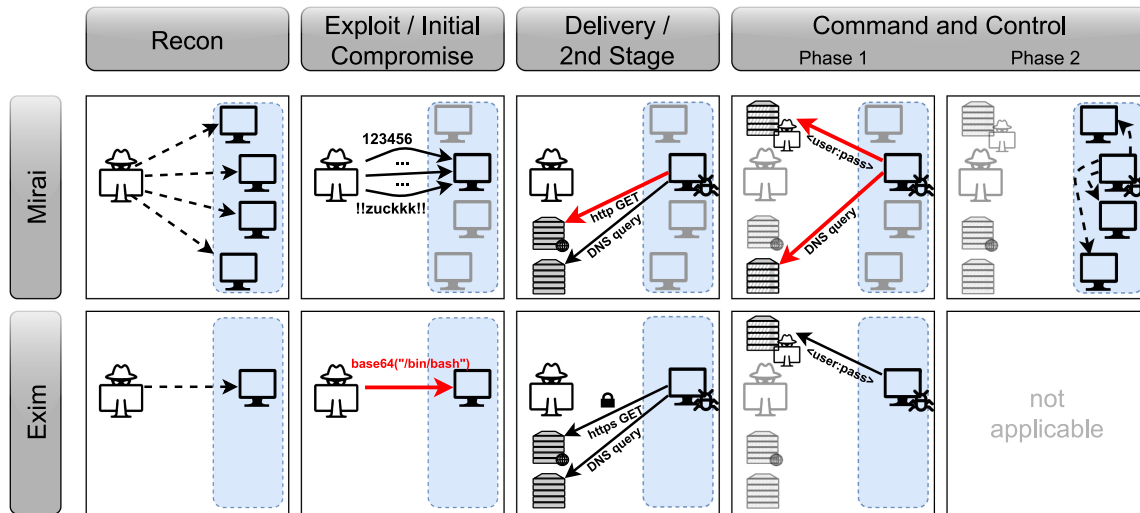
Black dashed lines indicate port scan activity, black solid lines indicate an active connection, red solid lines indicate an active connection that triggers a high-priority alert in the SOC. The blue area identifies the systems monitored by the SOC.

Fig. 2. Comparison and visualization of the Mirai and Exim attacks.

As the GOODSOC and BADSOC configurations (only) differ in the optional triggering of policy violation alerts, high-priority alerts generated for both scenarios are identical in both the experimental setups: when we refer to high-priority alerts triggered during the scenarios execution descriptions, these are generated in both experimental setups. A schematic visualizing both attack scenarios is depicted in Figure 2. The following provides a detailed description of the two attack scenarios.

Mirai Scenario.

Reconnaissance. The Mirai scenario starts with the attacker conducting a port scan on multiple ports against multiple targets for reconnaissance.

Exploitation/Initial compromise. The attacker gains initial access on the victim host by successfully guessing SSH credentials. The password-list used to reproduce this phase is derived from the original password list used by the original Mirai malware.

Delivery/2nd stage. The attacker opens an SSH session on the victim and executes a few commands. Eventually the victim resolves a domain name and then connects to a suspicious pastebin-like web server using a suspicious User-Agent Download Manager.

Command and control. A few minutes later, the victim tries to resolve multiple domain names known to be used by the Mirai botnet. Eventually, the victim uploads host-related data, including a username and password, to a Command and Control server via UDP. Both the IP addresses of the Command and Control server and of the attacker belong to the same class-C subnet.

The scenario ends with the compromised victim conducting a port scan on neighbouring devices in the same network, attempting a lateral movement. As noted before, this activity is captured with a low-priority alert from the SOC.

Mirai scenario alerts and attack visibility. The Mirai scenario is designed to trigger several alerts, compared to the Exim scenario. In total, three different high-priority categories of alerts are raised by the SOC. The most notable are four alerts of the type "ET TROJAN Mirai Botnet Domain Observed", triggered when the victim tries to resolve domain names known to (formerly) host Command and Control servers for the Mirai botnet (cf Command and Control phase in Figure 2). When leaking user credentials during the Command and Control

phase, an alert "ET CNC Shadowserver Reported CnC Server Group 10" is raised because the destination IP address is known to be involved in botnet activities. The alert generated during the Exploit/Initial Compromise phase ET MALWARE User-Agent (User Agent) - Likely Hostile is raised because the HTTP connection uses a suspicious HTTP user agent string. Additional alerts of lower priority, "ET SCAN Potential SSH Scan" and "ET SCAN Potential SSH Scan Outbound", are raised during the port scan activities in the Reconnaissance (attacker – targets) and Command and Control phase (victim – targets), respectively, (cf Figure 2).

Exim Scenario

Reconnaissance. Similarly to the Mirai scenario, the Exim scenario starts with the attacker conducting a stealthy port scan against a single victim.

Exploitation/Initial compromise. The attacker gets initial access by exploiting a vulnerability in the Exim SMTP server, which injects arbitrary base64 encoded shell commands when sending a dummy email via that server.

Delivery /2nd stage. From the remote shell obtained on the victim, the attacker accesses exploit-db.com, a website known to host proof-of-concept implementations of exploits. The HTTP connection is encapsulated in a TLS tunnel.

Command and control. Finally, the victim connects to the command and control server via UDP and uploads device-specific information including username and password of the current user.

Scenario alerts and attack visibility. The Exim scenario generates exactly one (high priority) alert. The alert generated by the exploit is titled "ET EXPLOIT bin bash base64 encoded Remote Code Execution 1". It is triggered by the attacker injecting a base64 encoded shell command into an SMTP email (cf Figure 2). None of the subsequent activities trigger any SOC alerts, thus analysts are required to conduct a manual investigation stating from the generated alert to reconstruct the attack. An investigation of the log files accessible in the SOC reveals the victim connecting to a web server and later on a UDP connection with the Command and Control server containing the user credentials. The rate of the port scan during the recon phase is below the (low-priority) alert threshold and thus does not trigger an alert.

Based on the small alert footprint we expect this attack to be harder to detect than the Mirai one.

4.6 Experiment Preparation

This section details the experiment setup and preparation for the scenario injection, and the attack reporting.

Attack traces. We use the previously identified attack phases in Table 1 for our proof-of-concept implementation, namely *Reconnaissance*, *Exploitation*, and *Delivery* and *Control* phases. For each of these phases we collect or reconstruct the respective network traces from public resources. For the *Reconnaissance* phase we collect data using nmap [30] scans against selected targets; in the *Exploitation* phase we rely on PoC exploits and brute-force attacks available on public resources, such as Exploit-DB [36] and Metasploit [43]. *Delivery* and *Control* include the download of malicious software from both Exploit-DB and suspect IP addresses, and communication with IPs known to be involved in botnet C2C infrastructures.

Attack generation. Using the previously identified attack traces, we build two different attack scenarios, described in Section 4.5; these attacks are built by assembling the previous network captures created and provided with consistency among individual actions by editing the IP addresses involved. Resulting attacks reproduce the *modus operandi* implemented by two real-world attacks (namely, *Mirai* and *Exim*).

Table 3 provides a comparison of the two scenarios across the attack phases defined by MITRE ATT&CK framework, and which phases can be detected by our SOC configuration. Note that the scenarios are unaffected by the GOODSOC or BADSOC configurations, as the GOODSOC configuration only removes "policy violation" rulesets that are not triggered by either scenario. Due to the higher number of alerts related to the Mirai attack than for Exim, we consider the latter to be a more advanced scenario than the former.

Table 3. Comparison between the Injected Attack Scenarios `Mirai` and `Exim`

Scenario phase	Mirai		Exim	
	Phase description	Alert	Phase description	Alert
Reconnaissance (port scan)	Comprehensive port scan; Multiple hosts, ports and services scanned with aggressive techniques.		Probing multiple hosts using a targeted port scan on port TCP/25.	
Exploitation (Initial access)	Successful SSH password guessing and local file inclusion/upload through SSH.		Attacker runs a known exploit against an SMTP server, triggering a red alert.	✓
Delivery (2nd stage)	Victim requests web page with suspicious data via HTTP. Bogus user agent triggers a red alert.	✓	Visited a web page offering public exploits via HTTPS; Encrypted content not readable	
Control (CnC DNS query)	Victim tries to resolve Mirai-related hostnames. Multiple red alerts are triggered.	✓	Nothing - *no DNS resolution.	
Control (CnC comm.)	Victim sends suspicious UDP packets to an IP address known to be involved in CnC activities; triggers a red alert.	✓	Victim sends suspicious UDP packet to a non-blacklisted IP address.	
Control (Pivot port scan)	Comprehensive port scan; Multiple hosts, ports and services scanned with aggressive techniques.		Nothing - *no further port scan.	

For each phase, we report if this activity triggers and high priority alert in the SOC.

Attack injection. The IP address of the attacker was rewritten to match the IP address defined in the scenarios. The attacks were injected in pre-recorded network traffic arriving at the SOC during working hours. While we cannot guarantee the background traffic to be attack-free, we analyzed the most significant alerts observed in the traffic to ensure that generated alerts not related to the injected attacks are not symptomatic of known attacks. We found no evidence of ongoing attacks in the pre-recorded network traffic.

Attack report. Due to infrastructural limitations, we collect reports using an online survey.[3] Each group could report up to five[4] suspicious activities detected during the experiment. Table 4 provides an overview of the questions. The first two questions asked for the attacker and victim IP addresses; as multiple entities can be associated to a role for an attack, we allow to insert more than a single IP address. Furthermore, multiple-choice questions address reconnaissance, vulnerability exploitation, the delivery and control phase of the attack. For each of the selected answers it is required to specify the IP address involved with the selected action. Lastly subjects could report additional free-text comments. The full survey is available in the Appendix. To evaluate whether a report is a response to one of the injected attack simulations, we compare the reported IP addresses to the ground-truth from the scenario definition. If the attacker *and* victim IP addresses of a scenario were both mentioned in the respective field, we mark the report to be a response to one of the corresponding scenarios.

Technical setup. We deployed two instances of the SOC environment and set up each accordingly to the `GOODSOC` and `BADSOC` configurations (ref. Section 4.3). Each VM is powered by a machine with Intel Xeon Bronze

---

[3]A SOC would normally rely on an integrated case management system for incident reporting; Such a tool was not available in the employed SOC setup.

[4]The limit of five activities was chosen after piloting. A discussion of implications on result validity is provided in Section 4.9.

Table 4. Summary of Information Expected in a Report

Analysis param	Info provided for result evaluation
Attacker IP	Framing of the reported scenario
Victim IP	Framing of the reported scenario
Reconnaissance	Detection of reconnaissance actions
Exploitation	Detection of exploitation actions
Delivery & C&C	Detection of delivery & C&C
Comments	Space for further information

3104 CPU with 12 cores at 1.70 GHz each and 64 GB of RAM. During the experiment, the two SOC servers' sensors were isolated from the University's network. Students accessed their respective analysis environments from the classroom on their own PCs.

4.7 Experiment Pilot

Prior to execution, we ran a set of pilots to refine the infrastructure and test the devised experimental procedures. A pilot phase was conducted to evaluate the survey procedure (including question phrasing) employed for the reporting phase. Furthermore, the pilots were used to assess whether the difficulty of the assignment and the pre-experiment training sessions were a good match with students' skills. To this aim, we piloted the whole experiment four times, pooling for volunteers from students with a similar background to our subjects (e.g., enrolled in other security courses at the time, or PhD students in Computer Science).

4.8 Experimental Execution

Student autonomously created groups of two and each group was assigned randomly to the BADSOC (n = 31) or GOODSOC (n = 32) experimental conditions. Students were given half a bonus point valid for the final exam for attending the experiment and if they managed to correctly identify (at least one) attacker in the injected attacks.

The experiment session started with an introduction and training session lasting approximately 30 minutes. After introducing the assignment, we ran a 15 minutes warm-up exercise where students had to identify a simple scenario with small network traffic to familiarize with the interface. The warm-up exercise was run on the BADSOC configuration for all groups.[5] The experiment was run after the warm-up phase and lasted 50 minutes. Subjects were told that at least one attack would be injected but were unaware of attack type and timing. At the end of the experiment, traffic injection was terminated, and the survey used for report collection was made available. Students were asked to submit their reports within 15 minutes but retained access to SOC and questionnaire until all submissions were collected. Figure 3 provides a visual representation of the experiment timeline.

4.9 Ethical and Experimental Considerations

The background noise traffic was captured from the SOC operating at the M&CS department of the host university under the ethical and organizational approval of the university. The traffic raw data was not accessible to the students during the experiment (i.e., they could only see the alerts generated based on the original traffic). The experiment was integrated in the practical educational activities run by the university and performed following guidelines approved by the university for exercises in classroom settings.

Experimental limitations. **(a) Construct validity.** We implicitly assume that students who report the correct IP addresses for both attacker and victim detected the attack and started analyzing it. Using automated checking

---

[5]The warm-up exercise, albeit simpler and shorter than the actual exercise, follows the same structure as the subsequently injected attacks.
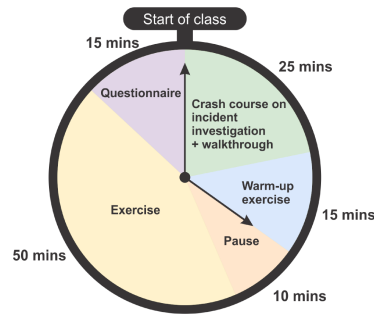
Fig. 3. Experiment phases and duration.

Table 5. Expected Findings/Ground-truth for Both Scenarios

Analysis param	`Mirai`	`Exim`
Attacker IP	199.19.215.23	31.220.56.38
Victim IP	131.155.68.116	131.155.71.27
Reconnaissance	port scan	port scan
Exploit/Vuln.	weak credentials	remote code exec.
Delivery/C&C	data exfiltration	data exfiltration
	HTTP requests	HTTP requests

it is impossible for us to determine whether a group actually understood what happened. **(b) External validity.** Students do not have the same experience and training that SOC analysts do and the training during the introduction section cannot make up for that. In real settings, the experiment should be replicated against multiple analysts and using a portfolio of attacks, as opposed to only one or two attack scenarios, to obtain ample validity across experimental conditions. **(c) Internal validity.** Students were told in advance that there would be at least one attack, creating an expectation to find something. Furthermore, by allowing students to submit up to five incidents and not penalizing "false positive" reporting, our experiment may inflate the number of reports per group. As this limit is the same across all groups and treatments, we do not expect any effect on the likelihood that groups report the "real" injected incidents (i.e., our main experimental outcome).

5 EXPERIMENTAL RESULTS

Whereas students could submit up to five reports, we expected exactly two reports detecting the injected attack scenarios `Mirai` and `Exim`, respectively. Table 5 reports the correct answers related to the `Mirai` and `Exim` scenarios.

5.1 Attack identification

We received 63 submissions (one per group) reporting a total of 162 incidents. Figure 4 reports overall submission rates for all groups in the BADSOC ($m = 2.28$, $sd = 0.99$) and GOODSOC ($m = 2.9$, $sd = 1.3$) configurations. Overall, the 32 groups working on BADSOC submitted 73 reports, the 31 groups on GOODSOC submitted 89 reports in total. A Wilcoxon Rank-Sum test results in BADSOC producing significantly less reports than GOODSOC ($p = 0.02$, $W = 353.5$). This suggests that groups assigned to the GOODSOC condition were able to reconstruct more "suspicious" activities than groups assigned to BADSOC, despite the events related to these activities being available to both treatment groups.
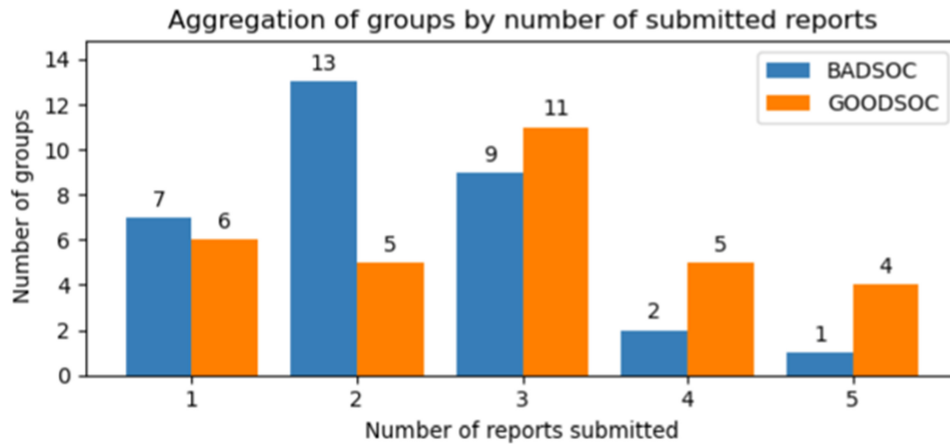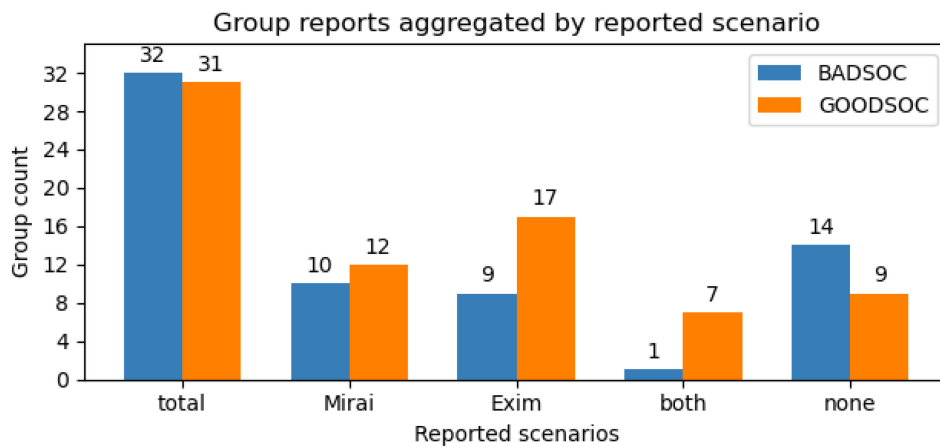
Aggregation of groups by number of submitted reports

Fig. 4. Aggregation of groups by number of submitted reports.

Group reports aggregated by reported scenario

Column 'both' represents the number of groups that reported both scenarios (Mirai *and* Exim), while 'none' represents the number of groups that did not report any of the scenarios.

Fig. 5. *Attack identification* by reported scenario.

Overall, 26% of BADSOC reports (19/73) and 33% of GOODSOC reports (29/89) detailed the injected attack scenarios (i.e., the majority of reports were *not* about the injected attacks). As groups could report up to five incidents, this is unsurprising. To evaluate reporting in more detail, Figure 5 shows the distribution of reports on the injected scenarios. Overall, 57% of groups in BADSOC and 71% of groups in GOODSOC reported at least one of the injected attacks. Whereas no noticeable difference against Mirai reporting rates can be observed ($p = 0.36$, $OR = 1.38$), GOODSOC groups were approximately three times more likely to report the Exim attack ($p = 0.03$, $OR = 3.04$) than the BADSOC groups, and approximately eight times more likely to report both attacks ($p = 0.02$, $OR = 8.49$) overall. However, if one considers only groups that reported at least one of the attacks, the difference between GOODSOC and BADSOC decreases and becomes borderline significant ($p = 0.09$, $OR = 5.56$); this may suggest that, under our treatment conditions, groups that perform well and detect at least one attack are more likely to find both irrespective of the SOC configuration. Similarly, BADSOC groups seem overall more likely ($OR = 1.88$) than GOODSOC to *not* report either of the attack scenarios, albeit the difference is not significant ($p = 0.17$).

Table 6. Report Results of *attack investigation*

		Mirai				Exim			
		Count		% of reports		Count		% of reports	
		BADSOC	GOODSOC	BADSOC	GOODSOC	BADSOC	GOODSOC	BADSOC	GOODSOC
	no. reports	**10**	**12**	**31.3%**	**38.7%**	**9**	**17**	**28.1%**	**54.8%**
Recon	correct	7	9	70.0%	75.0%	1	4	11.1%	23.5%
	wrong[b]	3	3	30.0%	25.0%	8	[b]13	88.9%	[b]76.5%
Expl.	correct	5	5	50.0%	41.7%	8	[c]14	88.9%	[c]82.4%
	wrong	[a]5	7	[a]50.0%	58.3%	1	[c]3	11.1%	[c]17.6%
Deliv. & Cntrl	*one correct* answer selected[x]	6	8	60.0%	66.6%	2	3	22.2%	17.6%
	data exfil	3	0	30.0%	0%	1	1	11.1%	5.9%
	http requests	3	8	30.0%	66.6%	1	2	11.1%	11.8%
	two correct answers selected[x]	0	0	0%	0%	0	0	0%	0%
	wrong	4	4	40.0%	33.3%	7	14	77.8%	82.4%

Number and fraction of correct answers for each correctly identified scenario. Fractions are calculated against the total number of reports for that scenario. [x]: two answers are correct; groups were allowed and expected to select two answers. [a]: one group selected two answers, one right and one wrong—we (only) counted the submission as *correct*. [b]: one group did not make any selection in response to this question, and we count it as wrong. [c]: three groups selected two answers—we counted them as *correct* if at least one of the given answers was right and wrong when both answers were wrong.

These findings provide an initial indication that the proposed solution can be effectively employed to evaluate (relative) SOC performance across SOC configurations (e.g., before and after a major configuration change in a deployed SOC).

5.2   Attack Investigation

We now provide a breakdown over the *reconnaissance*, *exploitation*, and *delivery and control* attack phases across the two injected attack scenarios. Results are reported in Table 6.

Reconnaissance. For both injected attacks, "port scan" was the correct answer, when asked for the observed reconnaissance activity in the questionnaire (see Table 5). For the Mirai attack, the correct answer was given by 70% (7/10) groups operating on BADSOC and 75% (9/12) groups operating on GOODSOC configuration ($p = 0.58$, $OR = 1.27$). The ratio of reports mentioning the port scan for the Exim attack is, overall, lower than for Mirai, albeit no significant difference emerges ($p = 0.42$, $OR = 2.39$): In BADSOC 11.1% (1/9) correct answers, in GOODSOC 23.5% (4/17). In conclusion, GOODSOC did not perform significantly better than BADSOC with in the detection of the port scan in none of the scenarios.

Exploitation. For exploitation activity, Table 6 shows that barely half of the groups selected the correct answer "weak credentials" when asked to report the exploitation vector abused by the attacker in the Mirai attack, i.e., 50% (5/10) in BADSOC and 41.7% (5/12) in GOODSOC. For the Exim attack, Table 6 shows that 88.9% (8/9) of the groups in BADSOC that reported the Exim attack also reported the correct vulnerability "remote code execution" abused by the attacker to gain foothold on the victim. In GOODSOC, 82.4% (14/17) answered correctly. On neither of the two injected attacks GOODSOC performed better than groups working with the BADSOC (Mirai: $p = 0.79$, $OR = 0.73$, Exim: $p = 0.84$, $OR = 0.59$). In fact, BADSOC performed slightly better than GOODSOC in identifying the correct exploitation vector (Mirai: $p = 0.52$, $OR = 1.38$, Exim: $p = 0.57$, $OR = 1.68$). Putting the differences between

BADSOC and GOODSOC aside, all groups were more likely to identify the correct exploitation actions in `Exim` (85%, or 22/26) than in the `Mirai` scenario (45%, or 10/22).

Delivery and Control. We expected groups to select two correct answers in a multiple-choice question about the delivery and control activities observed in context of an attack. The answers "data exfiltration" and "HTTP requests" are considered correct for both simulated attacks (see Table 5). Of the groups who reported on the `Mirai` attack, 66.6% (8/12) using GOODSOC and 60% (6/10) of the groups using BADSOC reported at least one of the two activities ($p = 0.55$, $OR = 1.32$). Looking at the `Exim` attack, 22% (2/9) of the groups reporting the attack on BADSOC and 18% (3/17) on GOODSOC also mentioned one correct delivery/control activity ($p = 0.79$, $OR = 0.76$). No group submitted both correct answers, regardless of their condition or reported attack. Again, there are no significant differences in performance between BADSOC and GOODSOC.

In the `Mirai` scenario option "HTTP requests" was likely to be selected roughly four times more often ($p = 0.099$, $OR = 4.32$) by groups operating GOODSOC. In the BADSOC configuration 30% (3/10) selected it, in GOODSOC 67% (8/12). Only three groups, all using BADSOC, chose the correct answer "data exfiltration" when reporting the `Mirai` attack. For the `Exim` attack, most groups, 56% (5/9) in BADSOC and 71% (12/17) in GOODSOC *wrongly* chose "none of them".

6 DISCUSSION OF RESULTS

The presented results indicate that the SAIBERSOC methodology is capable of highlighting the modifications introduced between SOC configurations. In our experiment, the GOODSOC configuration significantly increases the chances of a correct *attack identification*; however, the BADSOC vs GOODSOC conditions do not significantly affect the chances of correctly investigating the attack across its phases, indicating that the change in configuration leaves *attack investigation* capabilities unaffected.

Overall, in our example application the SAIBERSOC method indicates that the GOODSOC configuration is preferable to the BADSOC configuration as it significantly increases (up to threefold) the likelihood of identification of incidents. In particular, if GOODSOC and BADSOC were alternative environments tested in a SOC environment, using the SAIBERSOC solution the SOC manager of that infrastructure could conclude that analysts working with the GOODSOC configuration were more likely to find one or both of the attacks: analysts on GOODSOC were three times more likely to find the (slightly harder to identify) `Exim` attack ($p = 0.03$, $OR = 3.04$) than analysts on BADSOC. The method results indicate that analysts in the BADSOC configuration are either overwhelmed by the number of alerts to investigate or require more time analyzing individual events and therefore had no time to analyze the second scenario. However, analysts working with GOODSOC produced more false-positive reports, i.e., reports *not* in response to one of the injected attacks. This may indicate that analysts working on GOODSOC either spent less time per analysis and thus could write more reports in the same time, or that they are generally more likely to report false-positives.[6] By contrast, no significant differences between BADSOC and GOODSOC can be observed in the *attack investigation* phase. This provides additional information on the SOC performance related to which aspects of an incident investigations are (positively or negatively) affected by the different configurations under test. The ability to quantify these effects experimentally allows both researchers and practitioners in making informed decisions on technological and process-level solutions for security monitoring application.

Implications for research. The SAIBERSOC method can be used to evaluate the effect on security analysis of virtually any live analysis setting (such as a SOC or other security monitoring or analysis environments), either technical, procedural, or human. For example, SAIBERSOC can be employed to evaluate the impact of factors such as the analysts' experience, skill composition, and training, or the effect of analysis and visualization tools by comparing the performance of two SOC configurations with a known set of selected differences. Similarly, the

---

[6]This effect may be caused by our experiment design; students had the *expectation* to find at least one attack and were rewarded on finding at least one of the injected attacks, which may lead to over-reporting.

proposed method can be employed to evaluate multiple performance variables, including timing (e.g., total time for alert investigation) and throughput measures [48]), and how these vary across different phases of the attack. The extensibility of the proposed method further opens the door to the investigation of more complex effects, such as the relation between attack types and analyst skills/expertise [3], which are still under-researched in the scientific literature. Lastly, SAIBERSOC opens new opportunities for studying the human factor in SOCs and threat identification and and investigation scenarios on general. Besides of anthropological studies like the ones performed by Sundaramurthy et al. [51–53]), we are not aware of any academic publication that (a) focuses on the human aspects of threat investigation and incidence response and (b) was conducted in an operational SOC or cyber security team. Instead the majority of academic publications rely on data collected in artificial interview or experiment sessions, e.g., in form of a quiz featuring SOC alerts and possible responses [32, 35, 45, 59].

In summary, SAIBERSOC can be used to create more sophisticated experimental setups in which analysts have to perform hands-on alert investigation on a real SOC environment as part of the experiment and thus enables new opportunities for academic research in threat identification and alert investigation. The SAIBERSOC implementation presented further down in this section provides a practical tool to deploy experiments based on the proposed method in live or laboratory environments.

Implications for practice. The SAIBERSOC methodology (and tool) is fully flexible in terms of attack procedure, traces, and outcome variables, and can be implemented in operational SOCs to evaluate the relative performance of their configuration over the desired metrics. This includes both technical (e.g., rulesets, use-cases, alert correlation rules) and human-level (e.g., analysis processes, analyst expertise, flat vs hierarchical reporting structures) aspects of SOC operation, both of central importance for effective incident detection and reporting [26, 33]. Examples of concrete use-cases are comparison of SOC performance before and after changes in SOC configuration, or continuous monitoring of SOC performance. Furthermore, the proposed solution can be integrated in analyst training procedures, or as part of a selection procedure for security professional roles.

7   THE SAIBERSOC TOOL

To operationalize the SAIBERSOC methodology, after its validation, we developed the SAIBERSOC tool as an open source software solution, which allows to create and customize a synthetic attack to inject in the network. As such, SAIBERSOC implements all four architectural components outlined in Section 3.2. It is deployed with a web-based frontend (depicted in Figure 7) interacting with a backend. A comprehensive API enables scripted access to all backend functionalities.

7.1   Solution Design

The main goal of the SAIBERSOC tool is to enable SOC operators to run automated experiments in a SOC infrastructure. Experiments are composed, either automatically or manually, by selecting a number of "attack traces" (as defined by the operator) and meaningfully composing them (i.e., such that traces related to subsequent attack phases arrive in the correct order) in a timeline for execution. The operator can define the number of traces to use, the overall duration of the experiment, select traces manually or randomly, and extend/contract the duration of each single trace by operating the timeline. Once the traces are selected, experiments can be built and then injected using the builder and injector component. During the build process, all blocks (*attack traces*) in the experiment are assembled into a single pcap file. During the injection, SAIBERSOC keeps track of the current status of an experiment injection and allows to pause and resume, or abort an injection.

7.1.1   Operating Over SAIBERSOC *Experiments.* When designing the SAIBERSOC tool, we identified the execution flow of an experiment and condensed it in seven possible states. An overview of the states and their transitions is visualized in Figure 6.
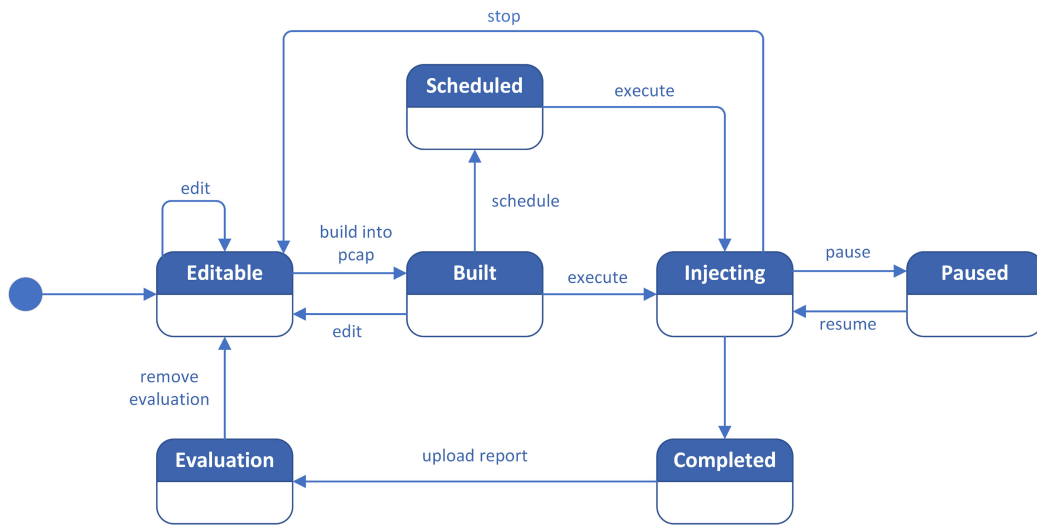
Fig. 6. Internal state visualization of an experiment in the SAIBERSOC application.

Editable. After creating a new experiment, the user is greeted with an empty timeline. A user can then add attack traces from the library to the timeline, remove attack traces from the timeline, or modify traces on the timeline (i.e., in terms of time offset, duration, rewriting IP addresses). The experiment remains in an *editable* state until the user decides to prepare it for injection.

Built. An experiment has to be *built* before it can be injected into the network. The building process is triggered manually by the user, which collects and merges all attack traces in the timeline into a single intermediate pcap file.

Any manual change of an already built experiment will reset the state of the experiment to *editable* and requires the user to build it again. This is necessary to ensure that the attack traces displayed in the timeline are consistent with the intermediate pcap file that is injected into the network.

Injecting. Once an experiment is *built*, the user can decide to run it immediately; when running, the final pcap file is injected into the network according to the specified timeline.

Scheduled. The user can schedule the experiment to automatically run at a later time. In this case, SAIBERSOC will automatically switch to the *injecting* state, starting the injection of the network traffic at the given date/time.

Paused. At any point in time, the user can pause the injection of an experiment, resume its injection or abort the process entirely. To maintain consistency, editing a running or paused experiment is not allowed.

Completed. When the pcap is completely injected into the network, SAIBERSOC will switch to the *completed* state.

Evaluation. Finally, the user can upload a SOC investigation report file that SAIBERSOC will automatically check against the ground-truth extracted from the attack traces that make up the experiment. SAIBERSOC will then provide feedback about the correct and wrong answers given in the uploaded SOC investigation report.[7] An experiment with an uploaded report cannot be modified to avoid inconsistencies between the uploaded result and the ground-truth extracted from the attack traces used to construct the experiment. At this stage, the user is able to delete the generated results/report to reset an experiment's state to *editable* and inject it again. Alternatively a user can duplicate the experiment itself.

---

[7]This feature is currently a proof-of-concept and is not exposed in the graphical frontend, and relies on a backend API for a user to upload a csv file with the experiment results.

7.1.2 Backend. The SAIBERSOC web application is implemented in Python. The (HTTP) API is based on Flask [46] and the Flask-RESTX framework [16] and comply to OpenAPI Specification 2.0.[8] Experiment building is implemented using Python Scapy [7], and the pcap generated file with the attack traces and background noise is injected into the network using tcpreplay [54]. Pause, resume, and abort commands to the injection process are implemented using POSIX signals (SIGUSR1, SIGCONT, SIGTERM) for inter-process communication between SAIBERSOC and tcpreplay. The state and meta information for traces in the attack trace library and experiments are stored in a PostgreSQL [42] database, while the pcaps themselves are simple files on the local file system.

7.1.3 Frontend. The frontend is implemented with TypeScript relying on the vue.js JavaScript framework. The VueX extension is used to manage the current state of the application. The frontend utilizes the API to retrieve data and submit requests to the SAIBERSOC application, i.e., no computation or state-changes are implemented in the frontend. Figure 7 shows the deployed web interface and how architectural components (see Section 3.2) AC-1…AC-3 are implemented in the web interface; report evaluation (AC-4) is not yet accessible through the frontend. On the left side of the screen, UX element **A** provides access to the attack *trace library* (AC-1) showing previously uploaded network traffic recordings. A user can manually add traces onto a timeline **B** in the center of the screen. Convenience features allow to search a specific attack trace in the library or select a random one. From the timeline **B**, the offset and playback speed can be adjusted for each attack trace individually. In the lower section of the screen, **C** provides additional configuration options for the selected attack trace. Here, IP addresses and whole IP address ranges can be rewritten to create a realistic attack scenario for the monitoring environment. Buttons **D**, in the upper right area, provide functionalities to load and save experiments/attacks and to start, stop, or schedule their injections (AC-3). During attack injection (AC-3), state and progress is visualized on the timeline **B** using a vertical red line as progress indicator. The tool is capable of automatically extracting a ground-truth based on the metadata from the attack trace library (AC-1) and the optional IP address rewrites in AC-2. A SOC report, allegorized by a csv file, can be checked against the ground-truth derived from the attack traces (AC-4).

Attack Traces (AC-1): The attack trace library consists of attack traces in form of pcap network traffic recordings linked to metadata stored in an internal database.

Attack Generation (AC-2): Internally, SAIBERSOC uses a list of *blocks* to specify an attack. Each *block* consists of an attack trace (taken from the attack trace library) and additional information specifying time offset, playback speed, and IP address rewriting. Once an attack is fully sketched out, scapy [7] is used to assemble a single intermediate attack pcap.

Attack Injection (AC-3): SAIBERSOC invokes tcpreplay [54] to inject the "intermediate attack pcap" on a network interface. Optionally, background noise can be replayed together with the attack.

Reporting (AC-4): The reporting module extracts and combines the IP address rewriting from AC-2 and attack trace metadata from AC-1 as ground-truth. It then compares selected columns from an uploaded csv file against the extracted ground truth.

7.2 SAIBERSOC in Operational Environments

7.2.1 Use Cases. We envision several use cases for the SAIBERSOC tool. All presented use cases require a tailored attack trace library relevant for the organizational scope and processes.

Extending monitoring capabilities and training. Firstly, we believe that the SAIBERSOC tool is useful for building a SOC or extending its capabilities. SAIBERSOC can give a first glance on what and how different cyber attacks

---
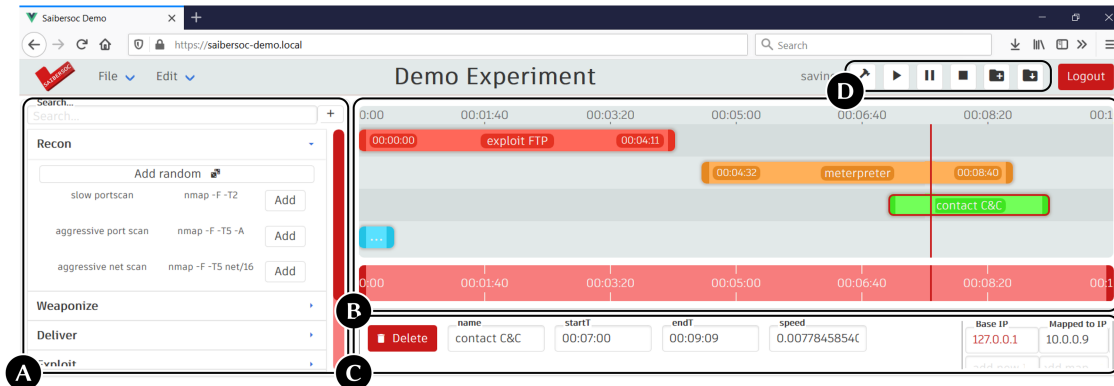
The *attack trace* library (AC-1) is depicted in (A). Attacks are *generated* (AC-2) by adding attack traces to the timeline (B), optionally changing playback speed and IP address rewriting (C). The *attack injection* (AC-3) is controlled using simple buttons. A red marker indicates the progress on the timeline (B).

Fig. 7. Screenshot of the SAIBERSOC web interface and relation to the relevant architectural components.

may be picked up by the SOC. This becomes relevant for example when a the scope of a SOC is extended to capabilities or scope of a SOC has to be extended (e.g., to cover different types of threats or new networks are added to the scope of the SOC).

In addition our tool can be used to train (future) SOC analysts on how to interact with the software, processes and procedures in the organization's SOC, and how to analyze specific events or attacks. Using the SAIBERSOC tool, one can quickly and repeatedly assemble and inject cyber attack scenarios tailored to the SOC and analyst to be trained.

Testing and red-teaming. Secondly, SAIBERSOC and the attack trace library can be used by a red- or purple-team to manually test operational performance of the SOC by simulating a breach. In a hybrid approach, a red-team can focus on the core parts of their attack and simulate common or less important aspects of an attack using SAIBERSOC. Alternatively, SAIBERSOC can replay distractions to hide the actions of the red-team.

Regression testing. Lastly, but importantly, we believe that SAIBERSOC may be used to implement a form of semi-automated regression tests. Major modifications to the SOC (e.g., change of software or change of operational processes) may have an impact on the operational effectiveness of a SOC. In such situations the SAIBERSOC tool allows to compare the detection and analysis capabilities of the SOC before and after the changes. These tests can become part of a change-management procedure process.

Education and research. The SAIBERSOC tool is currently used at our institution for educational and research activities. SAIBERSOC can be integrated in the educational curriculum of Cybersecurity and general Computer Science majors for training activities, as well as to run experiments involving students (or professionals, depending on the educational environment) to, for example, test and configure SOC alert generation to determine alert thresholds and performance of individual SOC components to improve our university's SOC configuration. Thus the SAIBERSOC tool proves to provide a practical solution to real-world problems. Other educational activities can have students extend the catalogue of attack simulations and attack traces by analyzing and replicating cyber attacks.

7.2.2 Publication, Development, and Licensing. The development of SAIBERSOC was supported by a team of BSc students as part of their final graduation project. The SAIBERSOC tool is released under Mozilla Public License (MPL 2.0). Source code, extensive documentation, and supplementary materials, including software design docu-

ments, a list of all authors, and a virtual machine for quick evaluation are available from the artifact repositories at https://gitlab.tue.nl/saibersoc/. Additionally, we present a selection of attack scenarios simulating real-world cyber attacks.

7.3 Limitations

The SAIBERSOC tool (and methodology) are designed to be integrated in virtually any deployed network-based monitoring solution. Being based on network traffic and attack traffic simulation, SAIBERSOC is unable to inject alerts into host-based monitoring systems or simulate breaches of individual hosts (in contrast to breach simulation tools designed for this purpose). That being said, SAIBERSOC is not an off-the-shelf solution but requires the creation of an attack trace library (cf AC-1) implementing individual aspects of cyber attacks to be emulated. For the creation of the attack trace library and assembly of targeted, complex, and meaningful attacks, expert knowledge is required.[9]

Lastly, SAIBERSOC relies on an existing SOC process and infrastructure as it only injects network traffic into a network. All anomaly detection and investigation has to be performed by the deployed monitoring systems. Compared to our solution, the setup by [45] is much more lightweight and allows for easy prototyping, at the price of not being able to measure operational performance of a real SOC, hence affecting the realism of derived measurements.

A further limitation of the current SAIBERSOC implementation tool is that it can only add network events but never alter or suppress background network traffic.

8 CONCLUSION

In this article, we proposed a methodology based on attack injection to systematically measure SOC performance across SOC configurations, analyst expertise, and for any output metric (e.g., accuracy, time-to-report, …). We verified the proposed methodology by conducting an experiment in which 124 students assumed the role of a SOC analysts. Our results show that the proposed methodology is capable of systematically measuring SOC performance and attributing it to differences in configuration, where some is to be expected. Our solution is general and can be implemented in any SOC. In addition, we developed the SAIBERSOC tool to help replicate the experiments and facilitate SOC exercises.

APPENDIX

Data Collection and Analysis

Evaluation for possible typos. In an exploratory data analysis, we observed that some groups reported IP addresses that are very similar to the ones we expected to see as attacker or victim IP address in one of the two scenarios. Based on the difference between the IP addresses in the report and the ground-truth, we conclude that it is very likely that those are typos or copy-paste errors.

In other cases, groups did not make the right selections in the multiple-choice part of the questionnaire, but later mentioned correct details in the additional comment section of the report. For example, group 78 did not indicate that they observed "http requests" through the multiple-choice selection but wrote "[…] this malware made a HTTP request […]" in the additional comments.

To ensure automatic and reproducible processing of the results, we decided to only consider IP addresses that are perfect matches and solely rely on the multiple-choice selection.

Submissions violating the enforced format. Some groups managed to submit reports that violate the constraints enforced through the questionnaire. This mostly includes submissions that ticked more or less checkboxes than

---

[9]We provide a set of exemplary attack simulations designed after real-world cyber attacks in our artifact repository (see Section 7.2.2). Furthermore, we give advice on how to design attack scenarios and populate a custom attack-trace library from scratch.

Table 7. Mapping between Requirements and Architectural Components

R-ID	AC-1	AC-2	AC-3	AC-4
R-1	Attack traces are independent of the monitored environment.	The attack generation allows for the matching of the traces to the environment.		The reporting and comparison relies on the ground truth for the generated scenario.
R-2	The attack traces identify all MITRE ATT&CK 'building blocks'.	The generated attack can suit any final environment and attack scenario.		
R-3		Attacks can be mapped to the target environment using asset information always available at the SOC.	The injection happens in the real network flow monitored by the SOC.	
R-4		The tailoring of the attack generation guarantees the existence of a ground-truth for the examination.		The analyst report can be automatically checked against the attack parameters (AC-2).

we allowed. Group 80 was able to select three options where the questionnaire was supposed to allow no more than two selections. We kept those submissions as they do not interfere with our way of data analysis.

Double-submission. One group submitted twice, i.e., we collected 64 submissions for 63 groups. While both submissions are similar, they are not identical. We decided to merge the submissions. The reports in response to one of the injected attack scenarios are merged by selecting the best parts of both submissions. We further decided to keep the submission time and duration of their first submission.

(Not) Accepting "lateral movement" in Mirai. We initially did not plan to accept "lateral movement" as a correct answer in Mirai scenario. In the Mirai scenario, the victim eventually mirrors the attacker's behavior and starts scanning the local network for vulnerable hosts. While the scenario ended before and therefore does not cover lateral movement, the network scan can be interpreted as a first step of lateral movement. We decided to *not* accept "lateral movement" as a correct answer for the Mirai scenario.

In total 40.0% groups (4/10) in BADSOC and 41.7% groups (5/12) in GOODSOC selected "lateral movement" (either alone or in combination with "http requests").

Questionnaire

Based on the choice on question 9, questions 3–8 can be repeated up to four additional times (i.e., a total of five reports can be submitted). The decision to limit the number of reports per submission was induced by limitations of the survey tool used to collect the results.

(1) **Insert your student ID**
    Multiple short text fields for multiple student IDs. Answers are considered personal data.

Table 8. Implementation Example Across AC-1...AC-4

	AC-1 Trace	AC-2 Attack Generation	AC-3 Injection	AC-4 Report
params	Identification of the needed attack phases	Attacker, victim, and CnC IP(s), URLs, #CVE, protocols, encryption, ...	Setting up pcap injection and schedule	1. SOC report; 2. Attack ground-truth
execution	Gathering or creation of attack traces	**Recon:** *portscan*(src_ip, dst_ip); **Expl:** *exploit_cve*(cve, src_ip, dst_ip); **Del:** *http_get*(src_ip, url, tls); **C&C:** *contact_cnc*(src_ip, dst_ip, prot, enc)	Pcap is injected in the sensor of the SOC according to schedule (e.g, 12pm)	Comparison of SOC report and attack ground-truth
output	**Del.:** *http_get*(); **Expl.:** *exploit_cve*(); **Recon.:** *portscan*(); **C&C:** *contact_cnc*()	1. Pcap of the synthetic attack built from the parametrized phases; 2. Attack ground-truth (expectations)	(At 12pm) pcap on wire is raising alerts in the SOC	e.g., SOC found portscan and CnC activity, but **not** exploitation **and** reported false positive lateral movement.

Each AC is characterized by its parameters (par.), execution (exec.) and output. In AC-1 the attack traces are structured across the MITRE ATT&CK phases; in AC-2 these traces are parameterized for the specific environment, for example, with IP ranges corresponding to the monitored infrastructure; AC-2 produces the traces that will be injected based on the defined parameters and generates the attack by ordering the traces following the ATT&CK framework phases. The parameters of AC-2 also characterize the ground truth of the attack. AC-3 injects the generated attack in the infrastructure, generating the corresponding alerts at the SOC level; finally, AC-4 checks the analyst's evaluations against the ground truth defined by the parametrization in AC-2.

(2) **Which was your SOC name?**
Binary choice between BADSOC and GOODSOC (names changed).
(3) **Insert the IP address(es) of the attacker, one per line. If no IP address is known, write "NA".**
Multi-line text field.
(4) **Insert the IP address(es) of the attacker, one per line. If no IP address is known, write "NA".**
Multi-line text field.
(5) **Did you observe any of the following reconnaissance activities?**
Multiple choice single selection only: "Username enumeration", "Port scan", "SIP scan", "Web application vulnerability scan", "None of them", "Other (small text field to specify)".
(6) **Which vulnerability did the attacker exploit?**
Multiple choice, multiple selections possible: "SQL injection", "Weak credential", "DNS remote command execution", "Poor web server configuration", "Remote coded execution", "None of them", "Other (small text field to specify)".
(7) **Which of the following actions did you observe? (check at most 2 boxes). For each option selected insert IP address of the receiver of such activities (i.e. HTTP requests -> web server's IP; enumerating SMB shares -> SMB server's IP;.**
Multiple choice, at least one, at most two answers can be selected. Except of "No action observed", all answers have a small text field to further specify an IP address: "Data exfiltration", "Enumerating SMB shares", "HTTP requests", "Denial of Service attack", "Web server path traversal", "NTP amplification", "Network lateral movement", "No action observed".

(8) **Anything else to report about this attack?**
Multi-line text field.

(9) **Do you want to report another attack?**
Binary yes/no selector. Subjects could submit up to five reports (questions 3–8), selecting "yes" would allow them to continue, selecting "no" would allow them to submit their results.

(10) **Did you had fun?**
Rating-scale 1 (really boring) – 5 (it was great)

(11) **Do you think the introduction was enough to do the exercise?**
Rating-scale 1 (not at all) – 5 (definitely enough)

(12) **Do you think we should do again this next year?** Binary yes/no selection.

(13) **Did you find anything particularly challenging during the investigation process?**
Multi-line text field.

(14) **Write here any other suggestions you might have in mind.**
Multi-line text field.

REFERENCES

[1] J. Abadie, B. P. Abbott, R. Abbott, T. Accadia, F. Acernese, R. Adhikari, P. Ajith, B. Allen, G. Allen, E. Amador Ceron, R. S. Amin, S. B. Anderson, W. G. Anderson, F. Antonucci, M. A. Arain, M. Araya, K. G. Arun, Y. Aso, P. Astone, P. Aufmuth, C. Aulbert, S. Babak, P. Baker, G. Ballardin, S. Ballmer, D. Barker, F. Barone, B. Barr, P. Barriga, L. Barsotti, M. Barsuglia, M. A. Barton, I. Bartos, R. Bassiri, M. Bastarrika, Th. S. Bauer, B. Behnke, M. G. Beker, A. Belletoile, M. Benacquista, J. Betzwieser, P. T. Beyersdorf, S. Bigotta, I. A. Bilenko, G. Billingsley, S. Birindelli, R. Biswas, M. A. Bizouard, E. Black, J. K. Blackburn, L. Blackburn, D. Blair, B. Bland, M. Blom, C. Boccara, O. Bock, T. P. Bodiya, R. Bondarescu, F. Bondu, L. Bonelli, R. Bonnand, R. Bork, M. Born, S. Bose, L. Bosi, B. Bouhou, S. Braccini, C. Bradaschia, P. R. Brady, V. B. Braginsky, J. E. Brau, J. Breyer, D. O. Bridges, A. Brillet, M. Brinkmann, V. Brisson, M. Britzger, A. F. Brooks, D. A. Brown, R. Budzy.ski, T. Bulik, A. Bullington, H. J. Bulten, A. Buonanno, O. Burmeister, D. Buskulic, C. Buy, R. L. Byer, L. Cadonati, G. Cagnoli, J. Cain, E. Calloni, J. B. Camp, E. Campagna, J. Cannizzo, K. C. Cannon, B. Canuel, J. Cao, C. D. Capano, F. Carbognani, L. Cardenas, S. Caudill, M. Cavaglia, F. Cavalier, R. Cavalieri, G. Cella, C. Cepeda, E. Cesarini, T. Chalermsongsak, E. Chalkley, P. Charlton, E. Chassande-Mottin, S. Chatterji, S. Chelkowski, Y. Chen, A. Chincarini, N. Christensen, S. S. Y. Chua, C. T. Y. Chung, D. Clark, J. Clark, J. H. Clayton, F. Cleva, E. Coccia, C. N. Colacino, J. Colas, A. Colla, M. Colombini, R. Conte, D. Cook, T. R. C. Corbitt, N. Cornish, A. Corsi, J.-P. Coulon, D. Coward, D. C. Coyne, J. D. E. Creighton, T. D. Creighton, A. M. Cruise, R. M. Culter, A. Cumming, L. Cunningham, E. Cuoco, K. Dahl, S. L. Danilishin, S. D'Antonio, K. Danzmann, V. Dattilo, B. Daudert, M. Davier, G. Davies, E. J. Daw, R. Day, T. Dayanga, R. De Rosa, D. DeBra, J. Degallaix, M. del Prete, V. Dergachev, R. DeSalvo, S. Dhurandhar, L. Di Fiore, A. Di Lieto, M. Di Paolo Emilio, A. Di Virgilio, M. Diaz, A. Dietz, F. Donovan, K. L. Dooley, E. E. Doomes, M. Drago, R. W. P. Drever, J. Driggers, J. Dueck, I. Duke, J.-C. Dumas, S. Dwyer, M. Edgar, M. Edwards, A. Effler, P. Ehrens, T. Etzel, M. Evans, T. Evans, V. Fafone, S. Fairhurst, Y. Faltas, Y. Fan, D. Fazi, H. Fehrmann, I. Ferrante, F. Fidecaro, L. S. Finn, I. Fiori, R. Flaminio, K. Flasch, S. Foley, C. Forrest, N. Fotopoulos, J.-D. Fournier, J. Franc, S. Frasca, F. Frasconi, M. Frede, M. Frei, Z. Frei, A. Freise, R. Frey, T. T. Fricke, D. Friedrich, P. Fritschel, V. V. Frolov, P. Fulda, M. Fyffe, M. Galimberti, L. Gammaitoni, J. A. Garofoli, F. Garufi, G. Gemme, E. Genin, A. Gennai, S. Ghosh, J. A. Giaime, S. Giampanis, K. D. Giardina, A. Giazotto, E. Goetz, L. M. Goggin, G. Gonzalez, S. Gosler, R. Gouaty, M. Granata, A. Grant, S. Gras, C. Gray, R. J. S. Greenhalgh, A. M. Gretarsson, C. Greverie, R. Grosso, H. Grote, S. Grunewald, G. M. Guidi, E. K. Gustafson, R.Gustafson, B. Hage, J. M. Hallam, D. Hammer, G. D. Hammond, C. Hanna, J. Hanson, J. Harms, G. M. Harry, I. W. Harry, E. D. Harstad, K. Haughian, K. Hayama, J.-F. Hayau, T. Hayler, J. Heefner, H. Heitmann, P. Hello, I. S. Heng, A. Heptonstall, M. Hewitson, S. Hild, E. Hirose, D. Hoak, K. A. Hodge, K. Holt, D. J. Hosken, J. Hough, E. Howell, D. Hoyland, D. Huet, B. Hughey, S. Husa, S. H. Huttner, D. R. Ingram, T. Isogai, A. Ivanov, P. Jaranowski, W. W. Johnson, D. I. Jones, G. Jones, R. Jones, L. Ju, P. Kalmus, V. Kalogera, S. Kandhasamy, J. Kanner, E. Katsavounidis, K. Kawabe, S. Kawamura, F. Kawazoe, W. Kells, D. G. Keppel, A. Khalaidovski, F. Y. Khalili, R. Khan, E. Khazanov, H. Kim, P. J. King, J. S. Kissel, S. Klimenko, K. Kokeyama, V. Kondrashov, R. Kopparapu, S. Koranda, I. Kowalska, D. Kozak, V. Kringel, B. Krishnan, A. Królak, G. Kuehn, J. Kullman, R. Kumar, P. Kwee, P. K. Lam, M. Landry, M. Lang, B. Lantz, N. Lastzka, A. Lazzarini, P. Leaci, M. Lei, N. Leindecker, I. Leonor, N. Leroy, N. Letendre, T. G. F. Li, H. Lin, P. E. Lindquist, T. B. Littenberg, N. A. Lockerbie, D. Lodhia, M. Lorenzini, V. Loriette, M. Lormand, G. Losurdo, P. Lu, M. Lubinski, A. Lucianetti, H. Lück, A. Lundgren, B. Machenschalk, M. MacInnis, M. Mageswaran, K. Mailand, E. Majorana, C. Mak, I. Maksimovic, N. Man, I. Mandel, V. Mandic, M. Mantovani, F. Marchesoni, F. Marion, S. Márka, Z. Márka, A. Markosyan, J. Markowitz, E. Maros, J. Marque, F. Martelli, I. W. Martin, R. M. Martin, J. N. Marx, K. Mason, A. Masserot, F. Matichard, L. Matone, R. A. Matzner, N. Mavalvala, R. McCarthy, D. E. McClelland, S. C. McGuire, G. McIntyre, D. J. A. McKechan, M. Mehmet, A. Melatos, A. C. Melissinos, G. Mendell, D. F. Menéndez, R. A. Mercer, L. Merill, S. Meshkov, C. Messenger, M. S. Meyer, H. Miao, C. Michel, L. Milano, J. Miller, Y. Minenkov, Y. Mino, S. Mitra, V. P. Mitrofanov, G. Mitselmakher, R. Mittleman, O. Miyakawa, B. Moe, M. Mohan, S. D. Mohanty, S. R. P. Mohapatra, J. Moreau, G. Moreno,

N. Morgado, A. Morgia, K. Mors, S. Mosca, V. Moscatelli, K. Mossavi, B. Mours, C. MowLowry, G. Mueller, S. Mukherjee, A. Mullavey, H. Müller-Ebhardt, J. Munch, P. G. Murray, T. Nash, R. Nawrodt, J. Nelson, I. Neri, G. Newton, E. Nishida, A. Nishizawa, F. Nocera, E. Ochsner, J. O'Dell, G. H. Ogin, R. Oldenburg, B. O'Reilly, R. O'Shaughnessy, D. J. Ottaway, R. S. Ottens, H. Overmier, B. J. Owen, A. Page, G. Pagliaroli, L. Palladino, C. Palomba, Y. Pan, C. Pankow, F. Paoletti, M. A. Papa, S. Pardi, M. Parisi, A. Pasqualetti, R. Passaquieti, D. Passuello, P. Patel, D. Pathak, M. Pedraza, L. Pekowsky, S. Penn, C. Peralta, A. Perreca, G. Persichetti, M. Pichot, M. Pickenpack, F. Piergiovanni, M. Pietka, L. Pinard, I. M. Pinto, M. Pitkin, H. J. Pletsch, M. V. Plissi, R. Poggiani, F. Postiglione, M. Prato, M. Principe, R. Prix, G. A. Prodi, L. Prokhorov, O. Puncken, M. Punturo, P. Puppo, V. Quetschke, F. J. Raab, D. S. Rabeling, D. S. Rabeling, H. Radkins, P. Raffai, Z. Raics, M. Rakhmanov, P. Rapagnani, V. Raymond, V. Re, C. M. Reed, T. Reed, T. Regimbau, H. Rehbein, S. Reid, D. H. Reitze, F. Ricci, R. Riesen, K. Riles, P. Roberts, N. A. Robertson, F. Robinet, C. Robinson, E. L. Robinson, A. Rocchi, S. Roddy, C. Röver, L. Rolland, J. Rollins, J. D. Romano, R. Romano, J. H. Romie, D. Rosińska, S. Rowan, A. Rüdiger, P. Ruggi, K. Ryan, S. Sakata, F. Salemi, L. Sammut, L. Sancho de la Jordana, V. Sandberg, V. Sannibale, L. Santamaría, G. Santostasi, S. Saraf, P. Sarin, B. Sassolas, B. S. Sathyaprakash, S. Sato, M. Satterthwaite, P. R. Saulson, R. Savage, R. Schilling, R. Schnabel, R. Schofield, B. Schulz, B. F. Schutz, P. Schwinberg, J. Scott, S. M. Scott, A. C. Searle, F. Seifert, D. Sellers, A. S. Sengupta, D. Sentenac, A. Sergeev, B. Shapiro, P. Shawhan, D. H. Shoemaker, A. Sibley, X. Siemens, D. Sigg, A. M. Sintes, G. Skelton, B. J. J. Slagmolen, J. Slutsky, J. R. Smith, M. R. Smith, N. D. Smith, K. Somiya, B. Sorazu, L. Sperandio, A. J. Stein, L. C. Stein, S. Steplewski, A. Stochino, R. Stone, K. A. Strain, S. Strigin, A. Stroeer, R. Sturani, A. L. Stuver, T. Z. Summerscales, M. Sung, S. Susmithan, P. J. Sutton, B. Swinkels, G. P. Szokoly, D. Talukder, D. B. Tanner, S. P. Tarabrin, J. R. Taylor, R. Taylor, K. A. Thorne, K. S. Thorne, A. Thüring, C. Titsler, K. V. Tokmakov, A. Toncelli, M. Tonelli, C. Torres, C. I. Torrie, E. Tournefier, F. Travasso, G. Traylor, M. Trias, J. Trummer, L. Turner, D. Ugolini, K. Urbanek, H. Vahlbruch, G. Vajente, M. Vallisneri, J. F. J. van den Brand, C. Van Den Broeck, S. van der Putten, M. V. van der Sluys, S. Vass, R. Vaulin, M. Vavoulidis, A. Vecchio, G. Vedovato, A. A. van Veggel, J. Veitch, P. J. Veitch, C. Veltkamp, D. Verkindt, F. Vetrano, A. Viceré, A. Villar, J.-Y. Vinet, H. Vocca, C. Vorvick, S. P. Vyachanin, S. J. Waldman, L. Wallace, A. Wanner, R. L. Ward, M. Was, P. Wei, M. Weinert, A. J. Weinstein, R. Weiss, L. Wen, S. Wen, P. Wessels, M. West, T. Westphal, K. Wette, J. T. Whelan, S. E. Whitcomb, B. F. Whiting, C. Wilkinson, P. A. Willems, H. R. Williams, L. Williams, B. Willke, I. Wilmut, L. Winkelmann, W. Winkler, C. C. Wipf, A. G. Wiseman, G. Woan, R. Wooley, J. Worden, I. Yakushin, H. Yamamoto, K. Yamamoto, D. Yeaton-Massey, S. Yoshida, M. Yvert, M. Zanolin, L. Zhang, Z. Zhang, C. Zhao, N. Zotov, M. E. Zucker, and J. Zweizig. 2010. All-sky search for gravitational-wave bursts in the first joint LIGO-GEO-Virgo run. *Physical Review D* 81, 10 (May 2010), 102001. DOI : https://doi.org/10.1103/PhysRevD.81.102001

[2] Anne Adams and Ann Blandford. 2005. Bridging the gap between organizational and user perspectives of security in the clinical domain. *International Journal of Human-Computer Studies* 63, 1–2 (2005), 175–202. DOI : https://doi.org/10.1016/j.ijhcs.2005.04.022

[3] Luca Allodi, Marco Cremonini, Fabio Massacci, and Woohyun Shim. 2020. Measuring the accuracy of software vulnerability assessments: Experiments with students and professionals. *Empirical Software Engineering* 25, 2 (2020), 1063–1094. DOI : https://doi.org/10.1007/s10664-019-09797-4

[4] Roberto Andrade, Jenny Torres, and Susana Cadena. 2019. Cognitive security for incident management process. In *Proceedings of the Information Technology and Systems*. Álvaro Rocha, Carlos Ferrás, and Manolo Paredes (Eds.). Springer International Publishing, 612–621.

[5] Roberto O. Andrade and Sang Guun Yoo. 2021. Cognitive security: A comprehensive study of cognitive science in cybersecurity. *Journal of Information Security and Applications* 48, Article 102352 (2021), 102352. DOI : https://doi.org/10.1016/j.jisa.2019.06.008

[6] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the mirai botnet. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 1093–1110. Retrieved 2020-06-13 from https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.

[7] Philippe Biondi. 2005. scapy. Retrieved 2020-06-13 from https://scapy.net/.

[8] Sunders Bruskin, Polina Zilberman, Rami Puzis, and Shay Shwarz. 2021. SoK: A survey of open source threat emulators. arXiv:2003.01518 Retrieved 2021-04-27 from https://arxiv.org/abs/2003.01518.

[9] Sonia Chiasson and P. C. van Oorschot. 2015. Quantifying the security advantage of password expiration policies. *Designs, Codes and Cryptography* 77, 2 (2015), 401–408. DOI : https://doi.org/10.1007/s10623-015-0071-9

[10] Common Vulnerabilities and Exposures (CVE) Database 2018. CVE-2018-6789. Available from MITRE, CVE-ID CVE-2018-6789. Retrieved 2020-06-13 from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-6789.

[11] The MITRE Corporation. [n.d.]. MITRE PRE-ATT&CK Introduction. Retrieved 2020-06-13 from https://attack.mitre.org/resources/pre-introduction/.

[12] Jon Davis and Shane Magrath. 2013. *A Survey of Cyber Ranges and Testbeds*. Technical Report DSTO-GD-0771. Australian Government, Department of Defence, Edinburgh, Australia. Retrieved 2021-05-04 from https://apps.dtic.mil/sti/pdfs/ADA594524.pdf.

[13] Constanze Dietrich, Katharina Krombholz, Kevin Borgolte, and Tobias Fiebig. 2018. Investigating system operators' perspective on security misconfigurations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, New York, NY, 1272–1289. DOI : https://doi.org/10.1145/3243734.3243794

[14] Elasticsearch B.V. [n.d.]. The Elastic Stack. Retrieved 2020-06-13 from https://www.elastic.co/elastic-stack. (Elasticseach and Kibana).

[15] exim-server 1995. Exim. Retrieved 2021-04-16 from https://www.exim.org/.

[16] Flask-RESTX [n.d.]. Flask-RESTX. Retrieved 2020-06-13 from https://github.com/python-restx/flask-restx.

[17] Rajesh Ganesan, Sushil Jajodia, and Hasan Cam. 2017. Optimal scheduling of cybersecurity analysts for minimizing risk. *ACM Transactions on Intelligent Systems and Technology* 8, 4 (2017), 1–32. DOI : https://doi.org/10.1145/2914795

[18] Guardicore. [n.d.]. Infection Monkey. Retrieved 2021-04-27 from https://github.com/guardicore/monkey.

[19] Eben M. Haber and John Bailey. 2007. Design guidelines for system administration tools developed through ethnographic field studies *(CHIMIT'07)*. In *Proceedings of the 1st ACM Symposium on Computer Human Interaction for Management of Information Technology*. Association for Computing Machinery, New York, NY. DOI : https://doi.org/10.1145/1234772.1234774

[20] Dave Hahn. 2018. Incident management at netflix velocity. USENIX Association, Washington, D.C. Retrieved 2021-04-27 from https://www.usenix.org/conference/lisa18/presentation/hahn.

[21] Cormac Herley and Paul C. van Oorschot. 2017. SoK: Science, security and the elusive goal of security as a scientific pursuit. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 99–120. DOI : https://doi.org/10.1109/SP.2017.38

[22] Robert Hof. 2016. Interview: How facebook's project storm heads off data center disasters. *Forbes.com* ([n. d.]). Retrieved 11th November 2021 from https://www.forbes.com/sites/roberthof/2016/09/11/interview-how-facebooks-project-storm-heads-off-data-center-disasters/#3bc1f4f64875.

[23] ISO. 2013. *Information Technology — Security Techniques — Information Security Management Systems — Requirements*. ISO/IEC 27001:2013. International Organization for Standardization, Geneva, Switzerland.

[24] Pierre Jacobs, Alapan Arnab, and Barry Irwin. 2013. Classification of security operation centers. In *Proceedings of the 2013 Information Security for South Africa*. IEEE, 1–7. https://doi.org/10.1109/ISSA.2013.6641054

[25] Pierre Conrad Jacobs. 2014. *Towards a Framework for Building Security Operation Centers*. Master Thesis. Rhodes University. Retrieved 2020-06-13 from https://research.ict.ru.ac.za/SNRG/Theses/Jacobs%202014%20Msc.pdf.

[26] Faris Bugra Kokulu, Ananta Soneji, Tiffany Bao, Yan Shoshitaishvili, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. 2019. Matched and mismatched SOCs: A qualitative study on security operations center issues. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1955–1970. DOI : https://doi.org/10.1145/3319535.3354239

[27] Kripa Krishnan. 2015. 10 years of crashing google. USENIX Association, Washington, D.C. Retrieved 2021-04-27 from https://www.usenix.org/conference/lisa15/conference-program/presentation/krishnan.

[28] Maria Leitner, Maximilian Frank, Wolfgang Hotwagner, Gregor Langner, Oliver Maurhart, Timea Pahi, Lenhard Reuter, Florian Skopik, Paul Smith, and Manuel Warum. 2020. AIT cyber range: Flexible cyber security environment for exercises, training and research. In *Proceedings of the European Interdisciplinary Cybersecurity Conference*. Association for Computing Machinery, New York, NY. DOI : https://doi.org/10.1145/3424954.3424959

[29] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. 2021. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Lockheed Martin Corporation. https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf.

[30] Gordon "Fyodor" Lyon. 1997. The Art of Port Scanning. *Phrack Magazine* 7, 54, Article 11 of 17 (1997). Retrieved 11th November 2021 from http://phrack.org/issues/51/11.html#article.

[31] Doug Miller, Ron Alford, Andy Applebaum, Henry Foster, Caleb Little, and Blake E. Strom. 2018. Automated Adversary Emulation: A Case for Planning and Acting with Unknowns. Retrieved 2020-06-13 from https://www.mitre.org/publications/technical-papers/automated-adversary-emulation-a-case-for-planning-and-acting-with.

[32] Ryan Mullins, Ben Nargi, and Adam Fouse. 2020. Understanding and enabling tactical situational awareness in a security operations center. In *Proceedings of the Advances in Human Factors in Cybersecurity*. Isabella Corradini, Enrico Nardelli, and Tareq Ahram (Eds.). Springer International Publishing, 75–82. DOI : https://doi.org/10.1007/978-3-030-52581-1_10

[33] Joseph Muniz, Gary McIntyre, and Nadhem AlFardan. 2015. *Security Operations Center: Building, Operating, and Maintaining Your SOC*. Cisco Press, Hoboken, NJ.

[34] Netflix. [n.d.]. Chaos Monkey. Retrieved 2021-04-27 from https://github.com/Netflix/chaosmonkey.

[35] Megan Nyre-Yu. 2021. Identifying expertise gaps in cyber incident response: Cyber defender needs vs. Technological Development. In *Proceedings of the 54th Hawaii International Conference on System Sciences*. 1978–1987. DOI : https://doi.org/10.24251/HICSS.2021.242 licensed under Creative Commons CC BY-NC-ND 4.0.

[36] OffSec Services Limited. 2004. Offensive Security's Exploit Database. Retrieved 2020-06-13 from https://www.exploit-db.com/.

[37] Open Security Foundation (OISF). [n.d.]. Suricata. Retrieved 2020-06-13 from https://suricata-ids.org/.

[38] Vern Paxson. 1999. Bro: A system for detecting network intruders in real-time. *Computer Networks* 31, 23–24 (1999), 2435–2463. DOI : https://doi.org/10.1016/S1389-1286(99)00112-7

[39] Shari Pfleeger and Robert Cunningham. 2010. Why measuring security is hard. *IEEE Security & Privacy* 8, 4 (2010), 46–54. DOI : https://doi.org/10.1109/MSP.2010.60

[40] Cuong Pham, Dat Tang, Ken-ichi Chinen, and Razvan Beuran. 2016. CyRIS: A cyber range instantiation system for facilitating security training. In *Proceedings of the 7th Symposium on Information and Communication Technology*. Association for Computing Machinery, New York, NY, 251–258. DOI : https://doi.org/10.1145/3011077.3011087

[41] Cuong Pham, Dat Tang, Ken-ichi Chinen, and Razvan Beuran. 2016. CyRIS: A cyber range instantiation system for facilitating security training. In *Proceedings of the 7th Symposium on Information and Communication Technology*. Association for Computing Machinery, New York, NY, 251–258. DOI : https://doi.org/10.1145/3011077.3011087

[42] PostgreSQL [n.d.]. PostgreSQL Database Management System. Retrieved 2021-09-06 from https://www.postgresql.org/.

[43] H. D. Moore. 2003. Metasploit Framework. Retrieved 2020-06-13 from https://www.metasploit.com/.

[44] Red Canary. 2017. Atomic Red Team. Retrieved 2020-06-13 from https://atomicredteam.io/.

[45] William T. Roden. 2019. Towards a Framework for Building Security Operation Centers. Master Thesis. University of North Carolina Wilmington. Retrieved 2021-04-27 from https://uncw.edu/csb/mscsis/complete/pdf/roden_fall2019.pdf.

[46] Armin Ronacher. [n.d.]. Flask. Retrieved 2020-06-13 from https://palletsprojects.com/p/flask/.

[47] Security Onion Solutions, LLC. 2009. Security Onion. Retrieved 2020-06-13 from https://securityonion.net/.

[48] Ankit Shah, Rajesh Ganesan, Sushil Jajodia, and Hasan Cam. 2018. A methodology to measure and monitor level of operational effectiveness of a CSOC. *International Journal of Information Security* 17, 2 (2018), 121–134. DOI : https://doi.org/10.1007/s10207-017-0365-1

[49] Ankit Shah, Rajesh Ganesan, Sushil Jajodia, and Hasan Cam. 2018. Understanding tradeoffs between throughput, quality, and cost of alert analysis in a CSOC. *IEEE Transactions on Information Forensics and Security* 14, 5 (2018), 1155–1170. DOI : https://doi.org/10.1109/TIFS.2018.2871744

[50] Iason Somarakis, Michail Smyrlis, Konstantinos Fysarakis, and George Spanoudakis. [n.d.]. Model-driven cyber range training: A cyber security assurance perspective. In *Proceedings of the Computer Security*. Apostolos P. Fournaris, Manos Athanatos, Konstantinos Lampropoulos, Sotiris Ioannidis, George Hatzivasilis, Ernesto Damiani, Habtamu Abie, Silvio Ranise, Luca Verderame, Alberto Siena, and Joaquin Garcia-Alfaro (Eds.). Lecture Notes in Computer Science, Vol. 11981. Springer International Publishing, 172–184. DOI : https://doi.org/10.1007/978-3-030-42051-2_12

[51] Sathya Chandran Sundaramurthy, Alexandru G. Bardas, Jacob Case, Xinming Ou, Michael Wesch, John McHugh, and S. Raj Rajagopalan. 2015. A human capital model for mitigating security analyst burnout. In *Proceedings of the 11th Symposium On Usable Privacy and Security*. USENIX Association, 347–359. Retrieved from https://www.usenix.org/conference/soups2015/proceedings/presentation/sundaramurthy.

[52] Sathya Chandran Sundaramurthy, Jacob Case, Tony Truong, Loai Zomlot, and Marcel Hoffmann. 2014. A tale of three security operation centers. In *Proceedings of the 2014 ACM Workshop on Security Information Workers*. ACM, 43–50. DOI : https://doi.org/10.1145/2663887.2663904

[53] Sathya Chandran Sundaramurthy, John McHugh, Xinming Ou, Michael Wesch, Alexandru G. Bardas, and S. Raj Rajagopalan. 2016. Turning contradictions into innovations or: How we learned to stop whining and improve security operations. In *Proceedings of the 12th Symposium on Usable Privacy and Security*. USENIX Association, 237–251. Retrieved 2020-06-13 from https://www.usenix.org/conference/soups2016/technical-sessions/presentation/sundaramurthy.

[54] tcpreplay 1999. Tcpreplay. Retrieved 2020-06-13 from https://tcpreplay.appneta.com/.

[55] The MITRE Corporation. 2013. MITRE ATT&CK. Retrieved 2020-06-13 from https://attack.mitre.org/.

[56] Paul Syverson, Roger Dingledine, and Nick Mathewson. 2002. Tor. Retrieved 11th November 2021 from https://www.torproject.org/.

[57] Zhihong Tian, Yu Cui, Lun An, Shen Su, Xiaoxia Yin, Lihua Yin, and Xiang Cui. 2020. A real-time correlation of host-level events in cyber range service for smart campus. *IEEE Access* 8 (2020). DOI : https://doi.org/10.1109/ACCESS.2018.2846590

[58] Verizon Enterprise Solutions. 2018. *2018 Data Breach Investigation Report*. Technical Report 11th edition. Verizon. Retrieved 2020-06-13 from https://enterprise.verizon.com/resources/reports/DBIR_2018_Report.pdf.

[59] Wei Wang, Francesco Di Maio, and Enrico Zio. 2020. Considering the human operator cognitive process for the interpretation of diagnostic outcomes related to component failures and cyber security attacks. *Reliability Engineering & System Safety* 202, 12 (2020), 107007. DOI : https://doi.org/10.1016/j.ress.2020.107007

[60] Carson Zimmerman. 2014. *Ten Strategies of a World-class Cybersecurity Operations Center*. The MITRE Corporation.