# SMAD: A Configurable and Extensible Low-Level System Monitoring and Anomaly Detection Framework


**Basel Sababa, Karlen Avogian, Ioanna Dionysiou, and Harald Gjermundrod**

## 1 Introduction

The global proliferation of technology has introduced new security challenges that span the devices themselves, their communication channels, and the systems that are connected to. Preventing security breaches in such a heterogeneous and diverse environment is nontrivial, despite the abundance of security products and technologies in the market, as the attack surface is simply too broad. The frequency of cyber attacks is increasing dramatically and organizations from both public and private sectors are struggling to identify and respond to those security breaches. Over the last few years, several instances of security breaches were brought to light with at least one thing in common: the response time was too long. According to the 2019 IBM Cost of a Data Breach report [1], the mean-time-to-identify (MTTI) a breach in 2019 was 206 days and the mean-time-to-contain (MTTC) was 73 days, with a notable 4.9% increase over the 2018 breach lifecycle. Taking into account these facts, one should expect that the security parameter of a system will be penetrated by an unauthorized party at some point, and the goal must be to identify the incident as quickly as possible and respond effectively. It is therefore of paramount importance to provide adequate practical training to the next-generation security experts, preferably using real data from past security incidents.

The identification of the security attacks relies on technological and human factors; the former one being the security tools that are integrated in the organization's network infrastructure and the latter one being the in-house security and system administrators who have the ultimate responsibility of all decision-making. Defense

B. Sababa · K. Avogian · I. Dionysiou (✉) · H. Gjermundrod
Department of Computer Science, School of Sciences and Engineering, University of Nicosia, Nicosia, Cyprus
e-mail: sababa.b@live.unic.ac.cy; avogian.k@live.unic.ac.cy; dionysiou.i@unic.ac.cy; gjermundrod.h@unic.ac.cy



K. Daimi, G. Francia III (eds.), *Innovations in Cybersecurity Education*,
https://doi.org/10.1007/978-3-030-50244-7_2

in depth, a multilayered approach that supports defensive mechanisms on various layers, is a popular approach to protect the network. However, the vast majority of the deployed security technologies focus on external attempts of bypassing the front lines of the system defense plan, without considering internal attacks orchestrated by individuals who are authorized users abusing their system privileges. As far as the human factor is concerned, it is generally argued that people are the *weakest link in the information security chain*. Rather than scanning and exploiting vulnerabilities in the deployed technology, it is comparatively easier to surpass the defenses of the human endpoints in the security chain using low tech but still sophisticated approaches such as phishing and social engineering.

This chapter proposes SMAD, a configurable and extensible **S**ystem **M**onitoring and **A**nomaly **D**etection framework based on Sysdig, which monitors kernel and system resources data (e.g., system calls, network connections, process info) based on user-defined configurations that initiate nonintrusive actions when alerts are triggered. SMAD is envisioned to be used not only by security-enthusiastic students with some technical skills to track their local Linux server health but also by university instructors who want to leverage the practical dimension of their security courses with the introduction of novel low-level system security monitoring tools. Monitoring raw kernel and system resources data is a tedious and error-prone task, if done manually. SMAD eases this overwhelming task by allowing users launching several system monitors via SMAD in a user-intuitive manner, alerting him/her on any unexpected behavior based on user-defined metrics, including CPU usage, directories visited, system errors, commands executed, HTTP requests, IP addresses connected to the system, and files opened. Furthermore, once an alert is triggered, raw data capturing could be initiated for a specified time period, giving the student the opportunity to correlate the alert triggering to the actual activities taking place in the kernel.

The rest of the chapter is organized as follows: Sect. 2 describes related research efforts on system monitoring. Section 3 presents the SMAD framework, followed by its experimental evaluation in Sect. 4. Section 5 discusses the value of SMAD in cybersecurity education. Section 6 concludes with future directions.

## 2 Related Work

System monitoring typically includes installing a surveillance software on a system, usually running as a background process monitoring the system's resources and performance, on the lookout for deviations from normal behavior. It runs concurrently with and independently from other types of system monitoring, such as operating system monitoring. In the event of unexpected behavior, alerts or notifications are sent to the system administrator. It is crucial to support system monitoring in real time while intrusive activities are in progress to minimize and/or contain the damage [2] as it leverages the user ability to control and maintain the monitored system [3].

There exist several academic and open source security-related system monitoring approaches as well as commercial solutions. Starting with the noncommercial approaches, Swatchdog [4] (formally known as Swatch [5]) is a log file monitoring system designed to address the challenges faced by system administrator when monitoring numerous servers continuously and simultaneously. Linux is configured to log security information to a central logging host system. In order to keep the system administrator from being overwhelmed by the size of logged data, Swatchdog monitors the log files and filters out unwanted and redundant data and supports an action system where an action is executed after filtering, specified by the user from a list of possible actions.

Haystack [6], a system designed to detect intrusions in air force computer systems, analyzes audit trail files daily, searching for user activity and comparing it against predefined security constraints and normal behavior models. The application generates a report that summarizes the activities analyzed. These reports can be analyzed by system administrators in order to locate possible intrusions.

Finally, Graph-based Intrusion Detection System (GrIDS) [7] is a system designed to monitor and analyze network activity on TCP/IP networks with thousands of hosts and possibly detect large-scale attacks. The application collects activities of individual computers and the networks among them, which are then aggregated into activity graphs. By analyzing various characteristics of these activity graphs, the application is able to automatically detect and report any network attacks that are happening in near real time.

There are other system monitoring tools that are not security oriented. For example, Benini et al. [8] have designed a system monitoring tool used for supporting dynamic power management in computers with tight power constraints. This is accomplished by monitoring and analyzing power consumption and dissipation on a device. The tool collects data on the use of system resources such as disks, CPU, keyboard, and mouse and analyzes the power consumption.

Commercial solutions also exist that monitor system activity either locally or on a cloud infrastructure. Nagios XI log monitoring system [9] integrates Swatchdog in a commercial enterprise IT infrastructure monitoring solution. Similarly, IBM [10] provides cloud monitoring using Sysdig [11] to collect monitoring data.

## 3 SMAD: System Monitoring and Anomaly Detection Framework

This section presents the design and functionality of SMAD,[1] an extensible framework for monitoring the state and various activities of a Linux server in an intuitive way. The SMAD framework could be perceived as a wrapper to Sysdig, therefore a brief overview of Sysdig is also given.

[1]The source code of the SMAD prototype can be found in Github repository https://github.com/kosnet2/sad

### *Sysdig Overview*

Sysdig [11] is an open source, command-line utility for capturing all system calls residing within the Linux kernel. It could be perceived as the Wireshark for the end system. Every time an installed application performs a privileged operation (e.g., open/read file, open network port, read/write to any device), it invokes a system call that executes the operation on the behalf of the user's process. Capturing all invoked system calls could be viewed as passive sniffing of all the operations performed within the server. A subset of the Linux monitoring and debugging tasks that are bundled by Sysdig is {*strace, tcpdump, netstat, htop, iftop, lsof*}. Additional features of Sysdig include provision of chisels (lightweight Lua scripts) for processing captured system events, provision of simple filtering of output, support of system and application tracing, and support of Linux server attack analysis features for ethical hackers.

The large number of the executed system calls would quickly overwhelm the system, with respect to both processing time and storage. As a countermeasure, Sysdig supports the configuration of filters in order to capture specific system calls or a subset of them. In this way, the filtered events are those of interest to the user. However, on the downside, it is nontrivial to formulate the appropriate filters tailored to the deployment, usage, and threat landscape of the specific server. There are commercial solutions that provide intuitive user interfaces for monitoring large deployments of servers (as well as container deployment) using Sysdig. An example is the Sysdig Monitor Dashboard [12] developed by Sysdig, a commercial product targeted for enterprises deploying applications in cloud infrastructures. Similarly, IBM offers a front end to Sysdig [10] as part of its BlueMadator product [13].

### *SMAD in a Nutshell*

The System Monitoring and Anomaly Detection (SMAD) framework is a modular framework that acts as a front end for Sysdig, utilized by a user possessing the required technical skills to monitor a personal server. SMAD's user-centric approach provides an intuitive environment to configure and operate a set of monitors, including start/stop operations, adding alerts to monitors whenever user-defined conditions are met, and capturing all system events for a specified time duration upon an alert trigger (optional action). The inspection and analysis of the captured log files are nontrivial and could overwhelm the novice system administrator. One of the goals of SMAD is to introduce the amateur user into low-level system events, acquiring and/or expanding their knowledge and skill set on the low-level operation of a system. To this end, a new SMAD component is currently under development for visualizing and/or graphically representing captured system activity. More details on SMAD functionality can be found in [14].

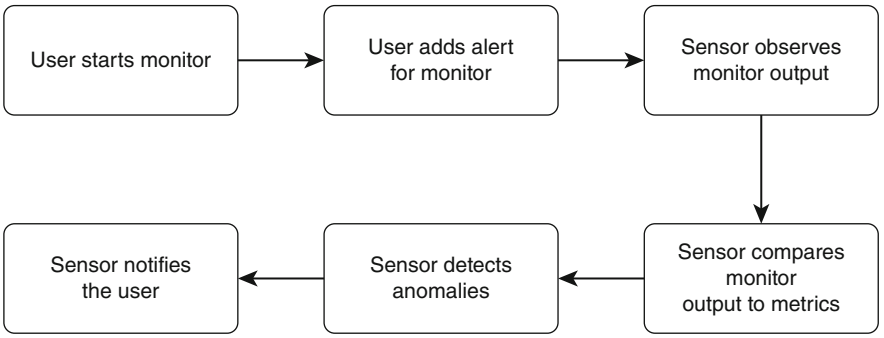



**Fig. 1** SMAD components' workflow

## *SMAD Components*

The SMAD baseline components along with the workflow of the system are illustrated in Fig. 1. The User interacts with the User Interface to issue commands (the command set is discussed in detail in the next subsections). The system executes the actions needed to fulfill the user's request, rendering the graphical interface with the execution output. If the stop/start monitor commands are issued, the User Interface interacts with the Monitoring Sensor that carries out the necessary actions. In the case where an event capture is attached to an alert trigger, the Monitoring Sensor starts storing events to Capture Files, based on the user-defined settings. The logged data is available for further analysis.

SMAD is extensible, supporting the integration of new components to provide additional functionality to the user. For example, one could develop a module that transforms the nonintrusive nature of SMAD into an active one by reacting to the alert trigger using actions other than event capturing, such as kill a process or close a port. The two baseline components, Monitoring Sensor and User Interface, are now presented in more detail.

### Monitoring Sensor Component

The Monitoring Sensor component is a wrapper for Sysdig, executing Sysdig commands with the appropriate arguments that are automatically generated based on the user's preferences and selections set using the User Interface.

As mentioned earlier, SMAD supports a notification mechanism via alerts. An alert is assigned to a particular monitor and its configuration profile includes the metrics to be monitored, the user notification method, and whether or not event capturing upon triggering is required. The cumbersome task of setting the desired metrics and their parameters is alleviated by the user-centric approach of SMAD that allows the specification of metrics to be done in a user-friendly and intuitive manner

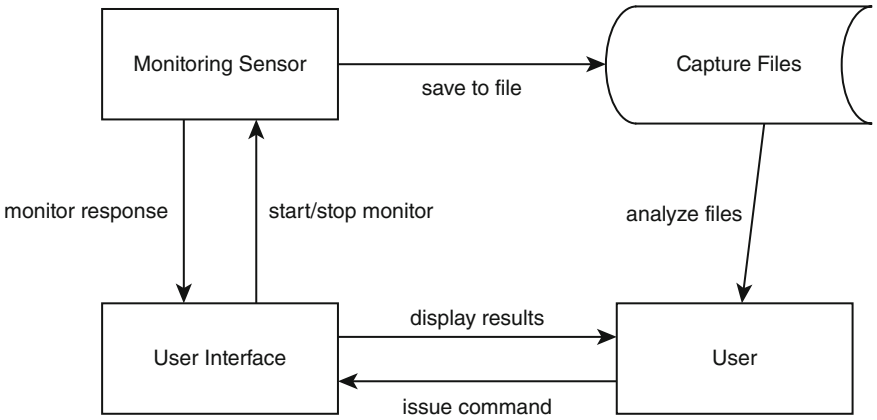



**Fig. 2** Monitoring Sensor component

while converting them to the appropriate Sysdig commands. The Monitoring Sensor maintains a list of running monitors and automatically detects alerts for its running monitors and starts observing the monitor output according to the metrics in the alert configuration profile. Figure 2 illustrates the Monitoring Sensor process.

### User Interface Component

The User Interface module was created using PyQt5, a Python-binding of the cross-platform GUI toolkit Qt [15]. It consists of three primary pages, namely, Monitors, Alerts, and Notifications, as shown in Fig. 3. These pages are now described in more detail.

#### Monitors Page

Monitors is the main SMAD page, responsible for managing monitors. A taxonomy of monitors is inherent in SMAD as it supports five monitor categories (CPU & Processes, Performance & Errors, Network, Security, and Application), and each category is represented in its own tab, as shown in Fig. 3. The monitor taxonomy was devised by taking into consideration usage scenarios for the target SMAD user. An easy-to-use technique to specify arguments is supported that allows monitor customization to meet the user's needs. Additional monitors could be added to each category as this taxonomy serves as a starting point to further extend the system.

Within each category, the user selects the monitors he/she would like to start by clicking on a checkbox. There are two types of monitors, one that requires further user input and another one that is self-contained. In the former type, no alerts are required to be configured and attached to the monitor. Clicking the "Start Monitors"


**Fig. 3** Monitors: CPU & Processes tab

button launches Sysdig instances running as background processes. At the same time, the Running Monitors list is updated to include the newly launched monitors. The "Stop Monitor" button kills the process running the selected monitor and gets removed from the list.

As mentioned earlier, there are five monitor categories. Starting with the CPU & Process monitor category, as shown in Fig. 3, it consists of two monitors. The first monitor observes the output of a specific process, where the user specifies the process to be monitored. Once the monitor starts, the specified process is monitored.

The second one keeps track of the top CPU-consuming processes, requiring no further user input. However, unlike the previous monitor, this monitor will not start immediately the monitoring, but it rather waits for an alert to be triggered to do so. The user must specify a metric for the allowed CPU percentage consumption that a process must exceed in order to trigger the alert (see Fig. 8).

Figure 4 shows the Performance & Errors monitor category, supporting six monitors, as listed below:

1. Monitor to show all failed open file operations by given processes (this monitor requires user input but not an alert specification)
2. Monitor to observe system calls returning the most errors
3. Monitor to show files with the most input/output errors
4. Monitor to observe processes with the most input/output errors

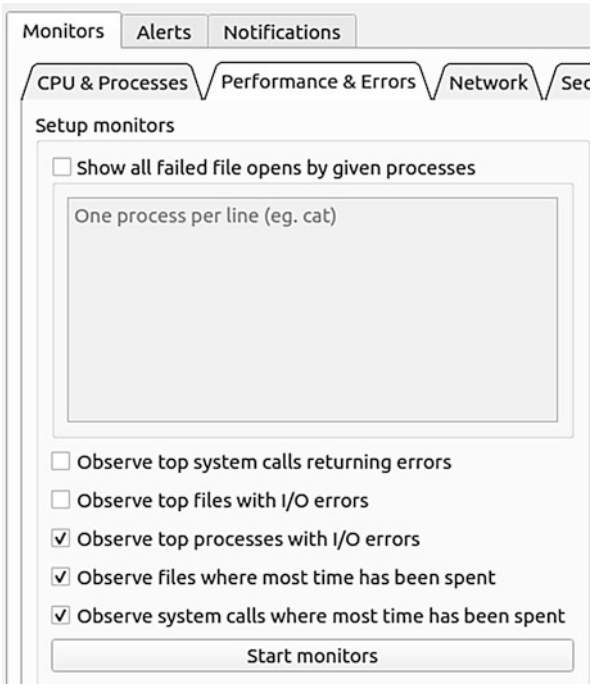



**Fig. 4** Monitors: Performance & Errors tab

5. Monitor to observe files where the system has spent the most time
6. Monitor to observe system calls where the system has spent the most time

These monitors could be used to detect system misbehavior or sluggish performance, allowing to narrow down the root of the problem. Additionally, they could be used to assess whether or not current applications running on the system generate an abnormally high number of faults, probing the user to investigate alternative applications to execute the specific task.

Proceeding with the Network category, shown in Fig. 5, the monitors comprising it are the following:

1. Monitor for network data exchanged for given IP addresses
2. Monitor for the network connections that consume the most bandwidth
3. Monitor for the processes that consume the most bandwidth

The last two monitors require an alert configuration with regards to bandwidth (see Fig. 10), triggering an alert when a threshold is reached.

The Security category is arguably the most important one (shown in Fig. 6). It contains three monitors that do not require alerts:

1. Monitor to show all the commands that are being executed by a user logged into the system
2. Monitor to show all the directories that a user is visiting
3. Monitor to display all file open operations that occur inside the mentioned directories

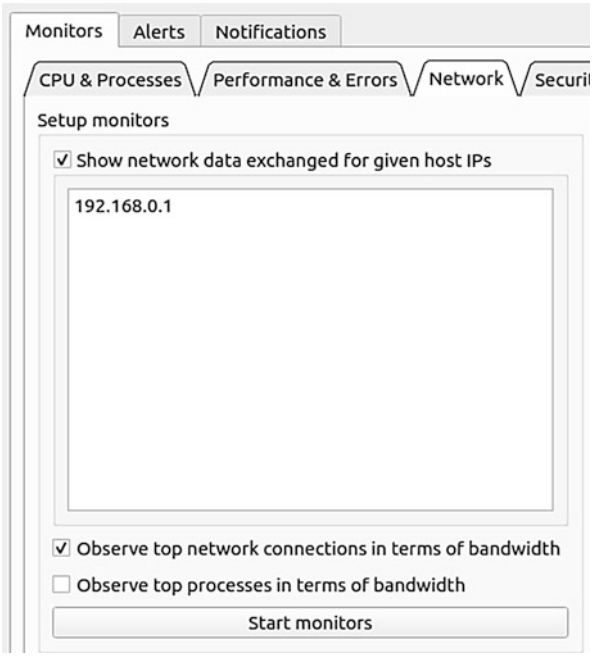



**Fig. 5** Monitors: Network tab

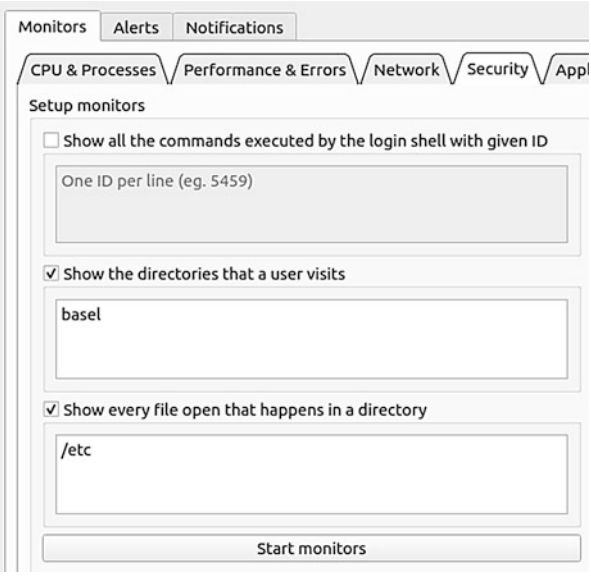



**Fig. 6** Monitors: Security tab

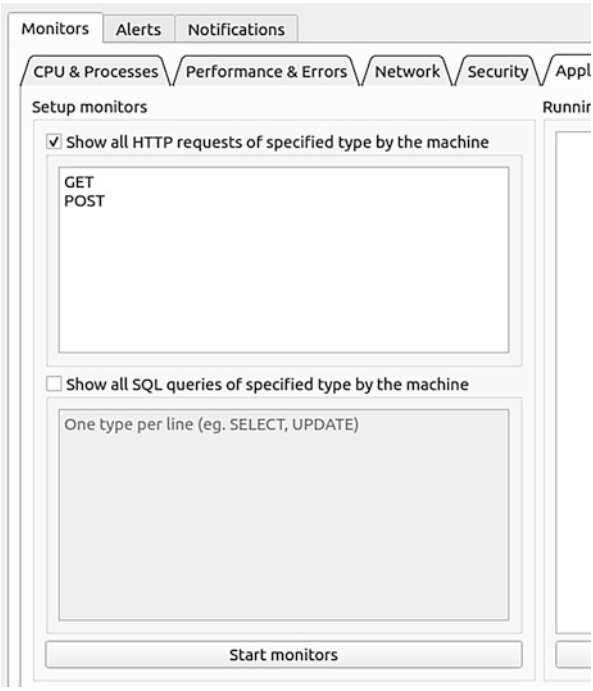



**Fig. 7** Monitors: Application tab

The launch of the security monitors allows a user to monitor the actual operations executed by a running application/service and detect misuse of resources, for example perform read/copy operations on files that it shouldn't need to access or open the file that will access a camera or microphone (if present).

Figure 7 shows the two monitors of the Application category. The first monitor shows all HTTP requests made to the system depending on the type of request that the user wants. The second one displays all SQL queries made to the system of the specified type that the user chooses. The Application monitors complement the Security monitors as they also monitor the activity of an application, but they can also be used to debug and evaluate what and how a specific application is performing its functionality.

### Alerts Page

The Alerts page is responsible for the alert configuration. An alert is a set of parameters attached to a specific monitor already running. Some monitors will not start monitoring events until at least one alert is attached to it since they require the metric from the alert specification profile. A metric violation yields an automatic notification posted on the Notifications page as well as initiating the capturing of events, if capturing is enabled for this alert. The alert parameters are as follows:

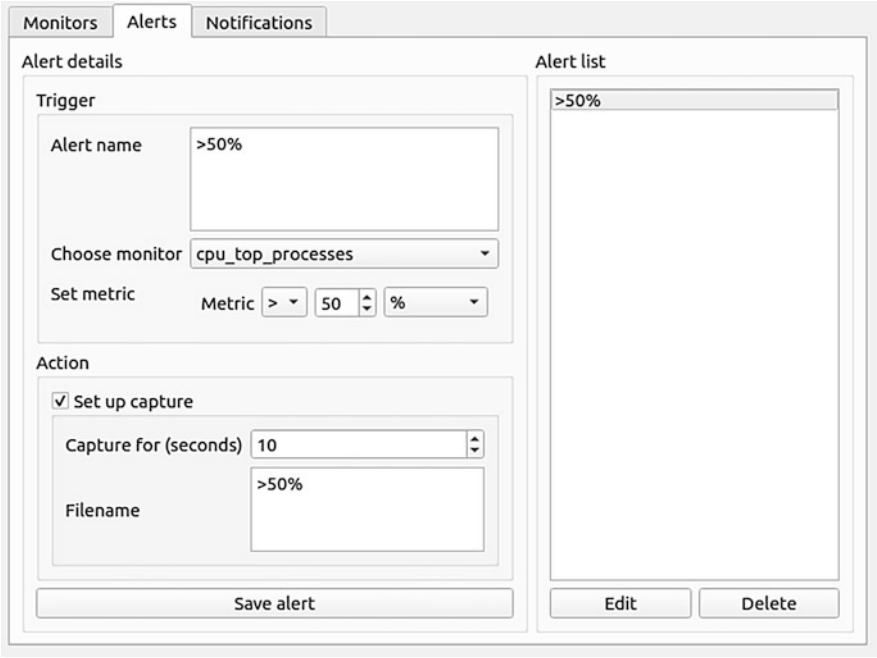



**Fig. 8** Alerts: Number metric

1. Monitor it is assigned to
2. Metric to be monitored (number, time, size)

Figure 8 shows the alert configuration to be attached to the *cpu_top_processes* monitor. This alert utilizes a number metric representing a percentage. It triggers the specific monitor to start observing processes that exceed 50% CPU usage. Additionally, once triggered, it initiates capturing of all events for the duration of 10 s (user-customized setting) and save them to the user-specified file. The file contents could be analyzed at a later time to derive useful information. SMAD only allows the configuration of error-prone alerts as it supplies the metrics along with the permitted operations (e.g. $<$, $>$, $=$) on them as well as checking that the value is within a valid range.

Figure 9 presents an alert that uses the time metric, where time could be specified in four time units. The user specifies the desired time using his/her preferred time unit and the application handles any unit conversion, if needed. The alert is attached to the *errors_files_most_time_spent* monitor and gets triggered when a file has been used for more than 4 $\mu$s. The alert also triggers a 20-s capture session saved to the file *time_spent.*

The third and final type of metric is size, in terms of data. There are four units available, namely, byte, kilobyte, megabyte, and gigabyte. Similar to the time metric, the user is free to use any size unit and the application will handle

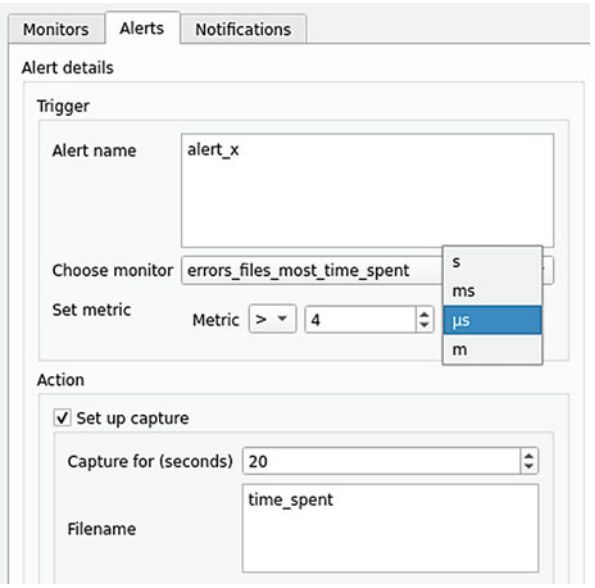



**Fig. 9** Alerts: Time metric

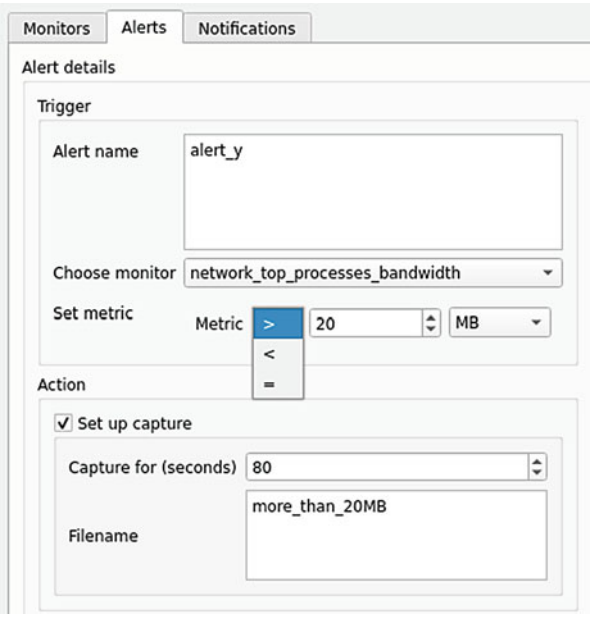
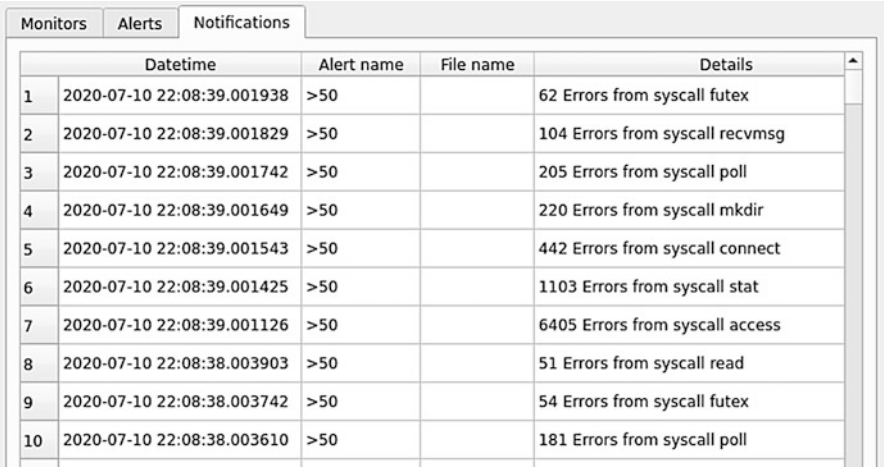



**Fig. 10** Alerts: Size metric

the conversions. In the example shown in Fig. 10 an alert is attached to the *network_top_processes_bandwidth* monitor and gets triggered when the bandwidth exceeds 20 MB. It also starts capturing events for 80 s after the metric violation.




| | Datetime | Alert name | File name | Details |
|---|---|---|---|---|
| 1 | 2020-07-10 22:08:39.001938 | >50 | | 62 Errors from syscall futex |
| 2 | 2020-07-10 22:08:39.001829 | >50 | | 104 Errors from syscall recvmsg |
| 3 | 2020-07-10 22:08:39.001742 | >50 | | 205 Errors from syscall poll |
| 4 | 2020-07-10 22:08:39.001649 | >50 | | 220 Errors from syscall mkdir |
| 5 | 2020-07-10 22:08:39.001543 | >50 | | 442 Errors from syscall connect |
| 6 | 2020-07-10 22:08:39.001425 | >50 | | 1103 Errors from syscall stat |
| 7 | 2020-07-10 22:08:39.001126 | >50 | | 6405 Errors from syscall access |
| 8 | 2020-07-10 22:08:38.003903 | >50 | | 51 Errors from syscall read |
| 9 | 2020-07-10 22:08:38.003742 | >50 | | 54 Errors from syscall futex |
| 10 | 2020-07-10 22:08:38.003610 | >50 | | 181 Errors from syscall poll |

**Fig. 11** Notifications tab

Notifications Page

The Notifications page displays events that have violated an alert's metric. The information shown is the alert responsible for the anomaly event, the date and time of the alert triggering, other information relevant to the event, and the name of the capture file if capturing was enabled for that alert.

In Fig. 11, one monitor is currently running that observes errors resulting from system calls. This monitor has only one alert attached with a metric to alert the user when the number of errors exceeds 50. In this case, the details provided are the number of errors and the system call where the errors originated from. The File name field is empty because capturing was not enabled by the user.

## 4 SMAD Experimental Evaluation

Three different evaluation tests were conducted to assess the SMAD system: stress testing, functionality testing, and vulnerability assessment.

### *Stress Testing*

The stress testing was performed to assess the overhead of SMAD on the machine it runs. SMAD was tested under heavy workload with multiple monitors running, each with multiple configured alerts. The stress testing experiment was performed

**Table 1** SMAD stress testing results

| Number of monitors | CPU usage (%) |
|---|---|
| 1 | 1 |
| 2 | 5 |
| 3 | 12 |
| 5 | 25 |
| 10 | 59 |
| 15 | 91 |
| 20 | 100 |

on an Ubuntu 18.04 operating system running on a VirtualBox virtual machine. The machine running the virtual machine had an x64-based processor Intel Core i5-9300H with 2.40 GHz frequency and 8 GB RAM. The virtual machine was restricted to only 2 GB RAM and 4 out of 8 cores.

Table 1 shows the findings of the experiment and the impact on the machine's CPU usage. Two alerts were attached per running monitor. Based on the findings, at 15 monitors, the application was heavily slowed down but was still functional. At 20 monitors, the application ultimately consumed the full capacity of the CPU. Therefore, it is not recommended to exceed 10 monitors running simultaneously.

## *Functionality Testing*

A test was designed to detect server misuse by an authorized user with limited privileges. The default settings on a Linux server assigns the user group read access (but not write access) to various system files like the *etc/passwd* file in order to perform their assigned tasks. However, if a user manually browses contents of specific files, it may be considered a suspicious activity and could be part of an insider attack.

Monitors were configured to observe high-risk locations and resources for potential misuse by insider attackers, who had already authorized accounts on the server. The following monitors were launched:

1. Observe files where most time has been spent
2. Show network data exchanged with server with a 10-s capturing
3. Show all directories visited by users with a 30-s capturing

At the beginning of the experiment, there were no notable captured events that required further analysis. After some time, an alert was triggered and a post notification shown on the Notifications list indicated that a user accessed the */etc* directory that contains system configuration files that a normal user should not need to access directly. This incident on its own does not indicate malicious activity but prompted a further investigation of the captured events after the alert was triggered. It was discovered that the user read the contents of the *etc/passwd* file using the *cat*

command and this activity could be part of the reconnaissance phase of an attack. Figure 12 shows a snapshot of the captured file that logged the user activity.

## *Vulnerability Assessment*

The goal of the vulnerability assessment was to uncover ways that SMAD could be exploited. It was discovered that SMAD was vulnerable to command injection, giving a malicious user access to the system. In particular, a number of text fields used for monitor configuration are subject to this attack. However, a user must be permitted to enter data into these fields, otherwise the capabilities of SMAD would be limited. For example, a user would not be able to specify a specific process, user, or IP address to monitor.

The countermeasure against command injection is the filtering of special characters from the user input, whose presence could indicate a command injection. If those characters are present, the input is discarded, as shown in the regex below:

```
if re.search('[&|;#$]', line):
    return False
```

```
42022 19:08:26.689217036 1 bash (12548) > rt_sigaction
42023 19:08:26.689217285 1 bash (12548) < rt_sigaction
42024 19:08:26.689217871 1 bash (12548) > rt_sigaction
42025 19:08:26.689218279 1 bash (12548) < rt_sigaction
42026 19:08:26.689233399 1 bash (12548) > execve filename=/bin/cat
42027 19:08:26.689374811 1 cat (12548) < execve res=0 exe=cat args=passwd. tid=12548(cat) pid=125
48(cat) ptid=12265(bash) cwd= fdlimit=1024 pgft_maj=0 pgft_min=30 vm_size=360 vm_rss=4 vm_swap=0
comm=cat cgroups=cpuset=/.cpu=/.cpuacct=/.io=/.memory=/.devices=/user.slice.freezer=/.net_cls=...
 env=CLUTTER_IM_MODULE=xim.LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;3... tty=34816 p
gid=12548(cat) loginuid=1000
42028 19:08:26.689400487 1 cat (12548) > brk addr=0
42029 19:08:26.689401260 1 cat (12548) < brk res=555F2E187000 vm_size=360 vm_rss=4 vm_swap=0
42030 19:08:26.689421413 1 cat (12548) > access mode=0(F_OK)
42031 19:08:26.689425397 1 cat (12548) < access res=-2(ENOENT) name=/etc/ld.so.nohwcap
42032 19:08:26.689427626 1 cat (12548) > access mode=4(R_OK)
42033 19:08:26.689428727 1 cat (12548) < access res=-2(ENOENT) name=/etc/ld.so.preload
42034 19:08:26.689432428 1 cat (12548) > openat
42035 19:08:26.689435954 1 cat (12548) < openat fd=3(<f>/etc/ld.so.cache) dirfd=-100(AT_FDCWD) na
me=/etc/ld.so.cache flags=4097(O_RDONLY|O_CLOEXEC) mode=0 dev=801
42036 19:08:26.689437271 1 cat (12548) > fstat fd=3(<f>/etc/ld.so.cache)
42037 19:08:26.689438649 1 cat (12548) < fstat res=0
42038 19:08:26.689439238 1 cat (12548) > mmap addr=0 length=101367 prot=1(PROT_READ) flags=2(MAP_
PRIVATE) fd=3(<f>/etc/ld.so.cache) offset=0
42039 19:08:26.689441342 1 cat (12548) < mmap res=7F8BEB304000 vm_size=460 vm_rss=4 vm_swap=0
42040 19:08:26.689442074 1 cat (12548) > close fd=3(<f>/etc/ld.so.cache)
42041 19:08:26.689442408 1 cat (12548) < close res=0
42042 19:08:26.689447305 1 cat (12548) > access mode=0(F_OK)
42043 19:08:26.689448171 1 cat (12548) < access res=-2(ENOENT) name=/etc/ld.so.nohwcap
42044 19:08:26.689452905 1 cat (12548) > openat
42045 19:08:26.689455135 1 cat (12548) < openat fd=3(<f>/lib/x86_64-linux-gnu/libc.so.6) dirfd=-1
00(AT_FDCWD) name=/lib/x86_64-linux-gnu/libc.so.6 flags=4097(O_RDONLY|O_CLOEXEC) mode=0 dev=801
42046 19:08:26.689456400 1 cat (12548) > read fd=3(<f>/lib/x86_64-linux-gnu/libc.so.6) size=832
42047 19:08:26.689457922 1 cat (12548) < read res=832 data=.ELF.............>.............@.....
```

**Fig. 12** Snapshot of captured data

## 5 SMAD Role in Cybersecurity Education

SMAD could play a vital role in formal education with its subject-oriented structured curricula, while at the same time SMAD educational benefits could also be perceived in a nonformal education setting via flexible adult self-learning.

### *SMAD in Formal Education*

Technology integration in course curricula could extend learning in powerful ways, demonstrating the application of theoretical concepts in practice. There is a plethora of cybersecurity tools and technologies that could be embedded in university courses, envisioning to develop multiskilled competent security practitioners. The SMAD framework could be an integral part of any security-oriented undergraduate/graduate course that aims in providing students with the sought-after technical knowledge and skills in cybersecurity system monitoring.

If one of the learning objectives of a security program were to also train the students to defend against zero-day vulnerabilities and novel attack techniques, then it would be imperative to comprehend the internal functionality of the security tools, moving aside the black-box practice of using tools and instead delve into the technical specifications of the tool modules. Educators often find it challenging to demonstrate low-level attack vectors and interactions that allow an attacker to circumvent the defense lines of a system to accomplish his/her mission (password cracking, file exfiltration, privilege escalation, backdoor installation, process manipulation, to just name a few). The current practice usually involves demonstration of point-and-click tools (usually GUI-based tools), low-level command-line commands/scripts (potential shell scripts in hex), rules files (for tools that support this), and log files. SMAD bridges the gap of black-box and white-box paradigms by using a point-and-click approach while at the same time uncovering what takes place underneath the hood (in the SMAD case, the kernel). Educators who use SMAD allow their students to gradually move from the point-and-click approach toward script environments, grasping the full technical profile of numerous attack scenarios.

Four SMAD-based scenarios, appropriate for the current laboratory part of a security course, are presented next. Each scenario is assigned a tentative level of difficulty based on the authors' experience teaching security courses, ranging from introductory courses (e.g., computer security, network security) to advanced graduate courses (ethical hacking, cyber warfare). It is strongly recommended that the course instructor prepares virtual machines (depending on the scenario) with known vulnerabilities to allow the students experiment in a secure testing environment, providing protection of operating within a sandbox environment to avoid accidental security incidents originating from the testing environment to an outside host.

**Scenario 1: Capture the Intruder (Level of Difficulty—Easy)**

One of the learning objectives of security courses covering topics related to system defenses and countermeasures against internal and external attacks is to expose the student in various attack methodologies. Ideally, the theoretical aspects of these methodologies should be demonstrated in practice in order for the student to acquire the practical dimension of the aforementioned methodologies.

The course instructor is taking an active role in this scenario and prepares a set of exercises following the capture-the-intruder style, where each exercise clearly states the service/resource/account the student is supposed to monitor using SMAD. The student is responsible for configuring the appropriate SMAD monitors, setting the event capture option on whenever an alert is triggered, and to monitor the assigned service/resource/account. The instructor decides when to launch the attack and executes the attack. Unlike other intrusion detection tools, SMAD logs all system calls once an intrusion is detected, giving the opportunity to the student to investigate the various system interactions once a threat is realized into an attack. The analysis of the captured system call set could yield an attack profile that the student could map into the various phases of an attack methodology.

**Scenario 2: Red/Blue Team Exercise (Level of Difficulty—Medium)**

This scenario is similar to the first one, with the main difference being that the course instructor has an observer/monitor role, who clearly sets the boundaries before the attack exercises commence. Students form red and blue teams, with the red team preparing an attack using penetration testing tools and/or attack frameworks whereas the blue team prepares the lines of defense using SMAD. It is recommended to hold two different exercises, allowing students to assume both red and blue team roles.

**Scenario 3: Dissection of Malware (Level of Difficulty—High)**

The postmortem analysis of an attack offers a useful insight into the attack pathway, allowing the formulation of an attack profile that could be utilized to detect (and perhaps prevent) future attacks based on the same or similar profile. The dissection of malicious code is challenging and nontrivial, going beyond the scope of most security courses. SMAD could be used to introduce the concept of the anatomy of an attack by running a malware in the sandbox test environment and studying the sequence of system calls (including their arguments) that were executed while the malware was running. The student should be able not only to form a timeline of the malware-related system calls but also determine what vulnerability was exploited, and how and when was it exploited. Depending on the level of the course and its learning objectives, the instructor could provide a set of malware, spanning relatively benign ones to more sophisticated ones.

**Scenario 4: Extending SMAD (Level of Difficulty—High)**

In general, graduate-level security courses focus on current trends and new research developments. Students interested in low-level security monitoring could extend SMAD and contribute their modules to the SMAD community. It is up to the instructor's discretion to decide what module could be developed. The instructor could also contact the authors, who could provide a list of potential modules.

## *SMAD in Nonformal Education*

A nonformal education learner discovers and acquires skills and knowledge from nonformal activities, outside the educational institution, usually while being part of the workforce. It is not uncommon for nonstudents with technical skills to experiment at home on their personal Linux servers, trying to synthesize theory and practice on their own. There are several sources of learning cybersecurity in an environment that diminishes the contact of instructor and student, including the massive open online courses offered by online learning platforms such as Udemy, Coursera, and Lynda and the tutorial-style video clips posted on online platforms such as YouTube. Users pursuing nonformal education are interested in obtaining knowledge or enhancing their current knowledge on a specific topic, without being concerned about accreditation and/or certificates of any sort. Below are ways that SMAD could contribute toward nonformal learning in cybersecurity.

**SMAD-Based Security Monitoring Tutorials**

SMAD is an ideal tool to use for developing short tutorials that demonstrate the attack methodology followed to exploit a known vulnerability, while at the same time present the actual technical details throughout the lifetime of the attack. It is challenging to deliver interesting and user-intuitive tutorials using the command-line interface. SMAD's graphical-based configuration of the command-line interface commands creates a learning environment that is more intuitive to the average learner. It is recommended in the tutorial to follow the sequence of steps as shown below:

1. Configure SMAD monitors that are triggered accordingly once the exploit is launched on the SMAD host, making sure to activate the event-capturing option.
2. Run the exploit and observe how/when SMAD alerts are triggered.
3. Use the terminal-style interaction to view the captured log files and determine from the system call sequence when and how the vulnerability was exploited.

### SMAD Interest Communities

Open source software greatly benefits through the community, an ad hoc group of contributors that inspect, modify, and enhance the software. It is environed that SMAD will attract the creation of two different communities to

1. Enhance its source code: Contributors belonging to this community advance the SMAD framework via submission of new modules or improved ones. This is a well-established activity in the open source community, with the source code hosted in a repository and contributions get accepted as long as they comply with the project's guidelines.
2. Enhance its use: Contributors in this community develop/configure VMs with different vulnerabilities or interesting misconfigurations along with guidelines of how an instructor or self-taught learner could use the specific VM in an educational/training session. This community is as equally important as the first one, but unfortunately it is underrepresented and not streamlined as the source-code contribution community. Lecturers/trainers/tutors spend a significant amount of time developing training material, including getting the systems configured/customized in order to demonstrate a particular topic. This work is mostly not shared back with the community. This practice could change, especially now with the rapid uptake of using virtualized environments (including droplets). Freemium models may also be appropriate alternatives for this sharing in order to motivate the contributions.

## 6 Conclusion

This chapter presented SMAD, a novel framework that monitors kernel and system resources data (e.g., system calls, network connections, process info) based on user-defined configurations that initiate nonintrusive actions when alerts are triggered. The user-centric SMAD environment allows the specifications of monitors and alerts to be done in a free-of-errors manner. A prototype system based on the framework was evaluated and its performance was assessed, yielding promising results. The only drawback of the system is that the amount of information that is captured might be overwhelming, making it tedious to browse and analyze the captured data. For example, the average capture file size generated after running a 60-s capture is 9 MB. A new SMAD component is currently under development for visualizing and/or graphically representing captured system activity and integrate this output with the Falco security system [16].

SMAD is also a security educational tool and its intended usage is by educators who want to leverage the practical aspect of their security courses as well as by students who wish to monitor the health of Linux servers. As technology integration in course curricula could extend learning in powerful ways, demonstrating the application of theoretical concepts in practice, SMAD could be part of any

security-related undergraduate and graduate course. The easy-to-configure nature of SMAD makes it an ideal introductory tool to low-level security monitoring, allowing students to experiment with alert configuration based on low-level system commands and properties, view and analyze system activity upon alert triggering, and add new functionality by extending SMAD with new modules.

## References


1. IBM Security and Ponemon Institute LLC, Cost of a Data Breach Report 2019. https://www.ibm.com/. Accessed Jan 2020
2. N. Ye, S. Vilbert, Q. Chen, Computer intrusion detection through EWMA for autocorrelated and uncorrelated data. IEEE Trans. Reliab. **52**(1), 75–82 (2003)
3. J.R. Harrow, F.P. Messinger, *System Monitoring Method and Device Including a Graphical User Interface to View and Manipulate System Information*. US Patent 5,375,199, 20 Dec 1994
4. Swatchdog, Simple Log Watcher. https://sourceforge.net/projects/swatch/les/swatchdog/. Accessed Jan 2020
5. S.E. Hansen, E.T. Atkins, Automated system monitoring and notification with swatch, in *Proceedings of the 7th USENIX Conference on System Administration*, USENIX Association, Monterey, CA, USA, 1993, pp. 145–152
6. S.E. Smaha, Haystack: an intrusion detection system, in *Proceeding of Fourth Aerospace Computer Security Applications*, Orlando, FL, USA, 1988, pp. 37–44
7. S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle, Grids—a graph based intrusion detection system for large networks, in *Proceedings of the 19th National Information Systems Security Conference*, Baltimore, MD, USA, 1996, pp. 361–370
8. L. Benini, A. Bogliolo, S. Cavallucci, B. Ricco, Monitoring system activity for OS-directed dynamic power management, in *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (IEEE Cat. No. 98TH8379)*, Monterey, CA, USA, 1998, pp. 185–190
9. Nagios, Nagios Enterprises Log Monitoring with Swatchdog. https://assets.nagios.com/downloads/nagiosxi/docs/Log-Monitoring-With-Swatch.pdf. Accessed Jan 2020
10. IBM, IBM Cloud Monitoring with Sysdig. https://www.ibm.com/cloud/sysdig. Accessed Jan 2020
11. Sysdig, Sysdig Open Source. https://github.com/draios/sysdig. Accessed Jan 2020
12. Sysdig, Sysdig Monitor Dashboards. https://sysdig.com/products/monitor/dashboarding/. Accessed Jan 2020
13. BlueMatador, Alert Automation for your Cloud Infrastructure. https://www.bluematador.com. Accessed Jan 2020
14. B. Sababa, System monitoring and anomaly detection application. Final Year Project Report, Department of Computer Science, University of Nicosia, 2020
15. Qt, Qt Open Source Widget Toolkit for GUI and Cross-platform Applications. https://www.qt.io. Accessed Jan 2020
16. Sysdig, Sysdig Falco. https://sysdig.com/opensource/falco/. Accessed Jan 2020