VIETNAMESE–GERMAN UNIVERSITY

PROJECT REPORT

A CEP-based SIEM System

Author:
Thien Hoang
Hieu Le
Khuong Lu
Quan Nguyen

Supervisors:
Prof. Dr. Martin Kappes
Manuel Grob

A final report submitted as partial fulfillment of the requirements for the subject Project of Winter Semester 2020/21

in the

Department of Computer Science

November 8, 2020

This page was intentionally left blank.

Declaration of Authorship

We, Thien Hoang, Hieu Le, Khuong Lu, Quan Nguyen, declare that the work presented in this report, titled “A CEP-based SIEM System”, is a product of our own, unless otherwise indicated. This work was done wholly during the course of the subject “Project” of the Winter Semester 2020/21 at Vietnamese–German University. We are aware of the University’s regulations regarding plagiarism, including any disciplinary actions taken against plagiarism.

Included with the report is the source code of the software that we had written. We are aware that the digital version of the document and the source code can be examined for the use of unauthorized work. We agree to have our work enter the university’s database for future examination on other submissions. Any other use of our work must be formally communicated beforehand.

Signature 1:	Thien Hoang
Signature 2:	Hieu Le
Signature 3:	Khuong Lu
Signature 4:	Quan Nguyen
Date:	

This page was intentionally left blank.

VIETNAMESE–GERMAN UNIVERSITY

Abstract

Faculty of Engineering
Department of Computer Science

Project of Winter Semester 2020/21

A CEP-based SIEM System

by Thien Hoang
Hieu Le
Khuong Lu
Quan Nguyen

For any business, monitoring and catching security events is extremely important to maintain a sustainable product and/or service. As the business scales up, the amount of events in the system will increase exponentially and quickly fill up the memory. This results in a need for automatic detection of malicious actions by analyzing the incoming events in real time.

In this report, we would like to present a monitoring software that we have built around a state-of-the-arts paradigm called Complex Event Processing. It is capable of capturing potential attacks against the system and/or the users in the system, and alerting them to the system administrators.

This page was intentionally left blank.

Acknowledgements

We would like to thank Prof. Dr. Martin Kappes for successfully organizing the course “Project” online during the Winter Semester 2020/21, when COVID-19 was changing the way we would study and work. We are also very grateful for Mr. Manuel Grob’s assistance and enthusiasm. Thanks to them, this remote project would forever be a memorable experience in our undergraduate study.

This page was intentionally left blank.

Contents

List of Abbreviations

CEP	**C**omplex **E**vent **P**rocessor/**P**rocessing
SIEM	**S**ecurity **I**nformation and **E**vent **M**anagement
EPL	**E**vent **P**rocessing **L**anguage
TCP	**T**ransmission **C**ontrol **P**rotocol
IP	**I**nternet **P**rotocol
VM	**V**irtual **M**achine

This page was intentionally left blank.

Chapter 1

Introduction

QUAN NGUYEN, THIEN HOANG

In this chapter, we would like to review on the concepts of Complex Event Processing and Security Information and Event Management , upon which this whole report and our product is built. We also briefly described the context of the project and the feature set of the software.

1.1 Prerequisites

1.1.1 Complex Event Processing

Complex Event Processing (CEP) is the use of technology for capturing and identifying data or application traffic as "events" for a period of real time. CEP Engine analyzes relationship among continuously received events and matches these events with patterns defined by *Event Processing Language* (EPL) statements. CEP is typically used for situations which there is the large number of events occurring and latency requirements are very low, typically in milliseconds such as real-time marketing, predictive maintenance, etc. [1]

CEP Engine organize events of the same class into what is known as *event streams*. For a person who is already familiar with database concepts, an event stream is

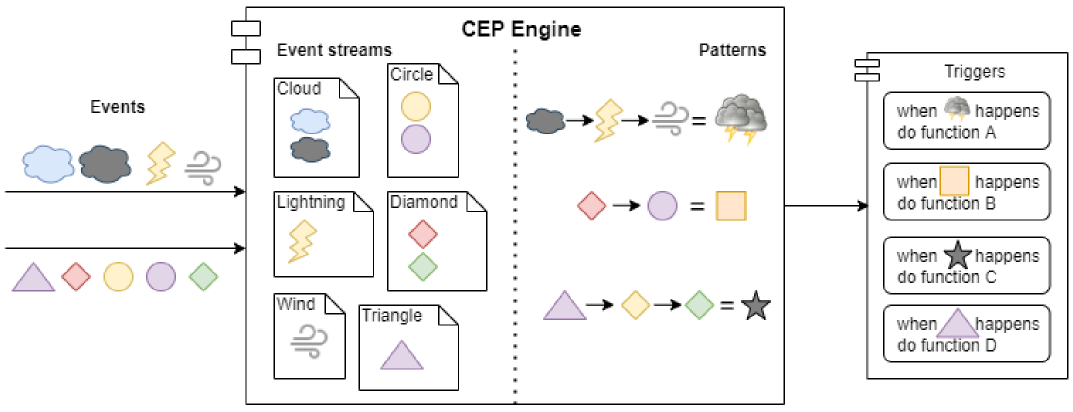

Events
CEP Engine
Event streams
Cloud
Circle
Lightning
Diamond
Wind
Triangle
Patterns
Triggers
when happens do function A
when happens do function B
when happens do function C
when happens do function D

FIGURE 1.1: A 10.000-meter overview of a CEP Engine

equivalent to a *table* (in fact, we would use these two terms interchangably in the report), while an event is equivalent to a row or *record* in the table.

Let's look at Figure 1.1: the CEP engine is receiving data and treats them as the put them neatly into the event streams such as cloud, wind, and lightning. On the right half of the CEP Engine area, we have some predefined patterns such as if there are cloud, wind, lightning coming in that order, an event of type "storm" will be *raised*. This means that a new record will be inserted into a separate event stream for Storm events, which was not shown in the figure for the sake of simplicity. Finally, CEP will trigger some action when certain events happen. One may think of a scenario wherein if a storm is coming, a warning will be sent to the government to evacuate the citizens.

One famous implementation of CEP Engine is Esper. One robust feature of Esper of which we frequently took advantage is the concept of *event window*. To put it simply, one can create a table with an expiration rule to automatically remove events from the table, e.g. only keep the Lightning events in the last 10 minutes, or only keep the last 15 Diamond events.

1.1.2 Security Information and Event Management

Security Information and Event Management (SIEM) is a set of techniques for threat detection and security incident management through collection, identification, and analysis both in near real time and past log events, as well as a wide variety of other events. In a *CEP-based SIEM system* (Figure 1.2), a set of collectors will gather data and feed them to the CEP engine. CEP engine will aggregate the data from the collector, then identifies and categorizes them into event streams, compare them to the predefined patterns, and may trigger certain function (defined externally to the CEP engine) when some event shows up. It can be an alert or a notification to the system administrator, or in more advanced system, a sophisticated algorithm to patch the security hole.

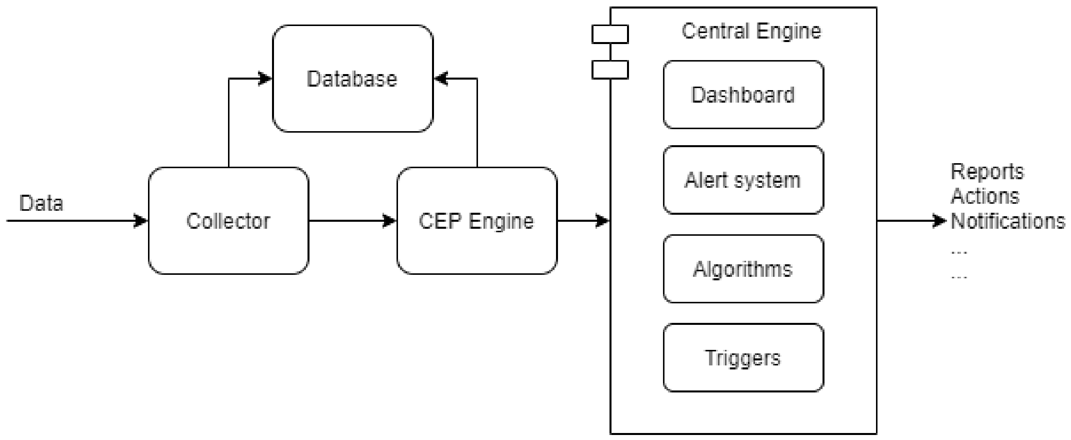


FIGURE 1.2: A typical CEP-based SIEM System

In general, a SIEM system extends the capability of a CEP engine. The CEP engine knows how to analyze the data, but SIEM system takes care of the pre-analysis and post-analysis procedures.

Our application also follows this general architecture in Figure 1.2. One may look at subsection 2.1.2 for an example.

1.2 Project requirements

Per the requirements of the course "Project" of the Winter Semester 2020/21, each group of four students must implement at least three of the following four features:

- Monitor a web server: detect failed logins, denied access to protected pages, analyze status codes returned, etc.
- Detect potential port scans: port scan is a typical action of an attacker wherein he/she was trying to find which ports are open before doing an actual attack on a port.
- SIEM Dashboard: an interface for system administrators to get an overview of what is happening in the system, as well as to control the SIEM system to some extent.
- "Decide yourself": up to creativity.

We have chosen to implement the first three requirements. Our SIEM system (henceforth referred as *Neptune's Watcher*) is capable to catch the some harmful events:

- Brute-force attack
- Dictionary attack
- Vertical port scan against multiple ports on a single machine
- ...

Of course the feature set does not end there, rather, this is to give the readers a general idea of how we use events to make deduction of a malicious activity. Please refer to section 2.2 for a complete set of capabilities of the software.

Lastly, we have designed a dashboard that is able to display all important events and alerts (with different levels of severity) neatly in tables. The thresholds to raise an alert can be controlled without leaving the dashboard as well.

This page was intentionally left blank.

Chapter 2

Application overview

Hieu Le, Thien Hoang

By implementing three requirements stated in the previous chapter, Neptune's Watcher would have three main components–Webserver Monitor, Port Scan Detector, and Dashboard. What are they capable of?

2.1 Architecture

Neptune's Watcher contains three components: a Webserver Monitor for checking improper access, a Port Scan Detector for detecting port scans, and a Dashboard for interacting with the user. The backend consists of Webserver Monitor and Port Scan Detector, working independently of each other, although they can be controlled from the same Dashboard. (Figure 2.1)

In the next subsections, we are going to explain more about the two backend components and how they interact with the Dashboard.

2.1.1 Monitoring a web server

The overall task of the first subsystem is to monitor and report accesses and attacks made by visitors. The Webserver Monitor is able to classify the priority of each observed threat ranging from low to high based on their security impact.

For our approach, we first created a web server that used the open-source Apache `httpd` [2] and PHP. The application is delivered together with a simple website for demonstration. We have programmed the PHP scripts to write log messages into `/var/log/httpd/error.log`. Messages are prefixed with "Neptune: " as a signal so that later on the Webserver Monitor can read `error.log` and take into account only the log entries produced by Neptune's Webserver. In addition, Apache's `httpd` also logs all access requests to any resources in the Webserver, these events go into `/var/log/httpd/access.log`. Neptune's Watcher would read this log file as well to monitor incoming accesses. More on the Webserver in section 3.3.

In short, the Webserver will create two log files that the Webserver Monitor can read and analyze the different kinds of security-related log entries based on them.

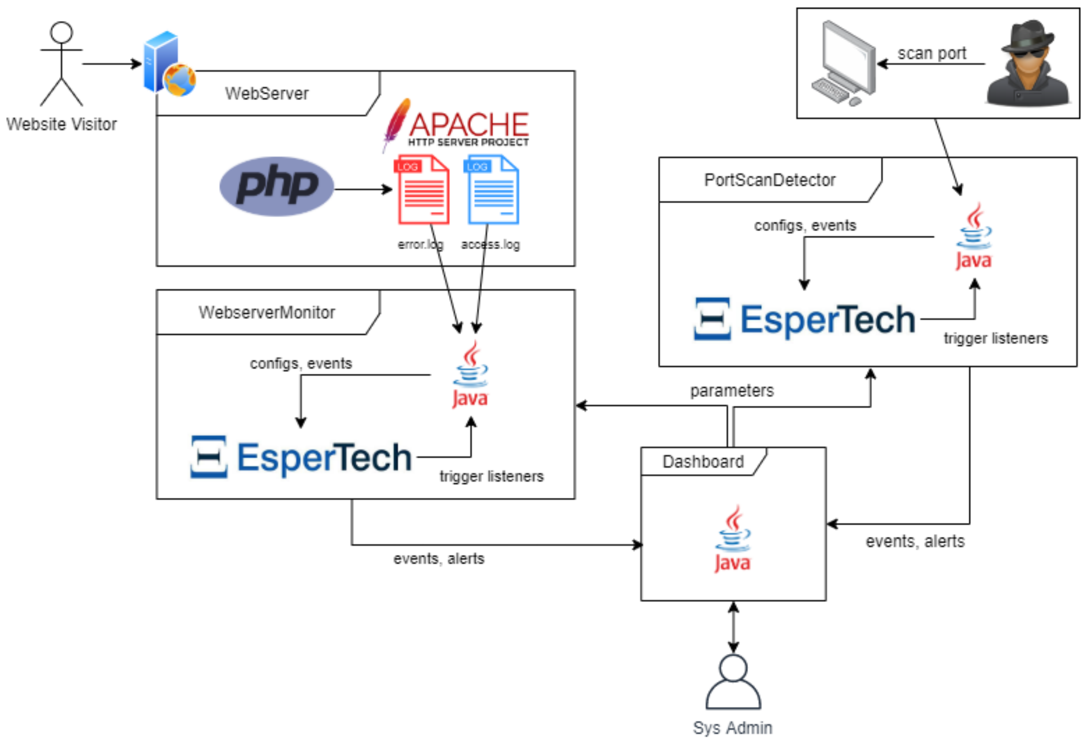


FIGURE 2.1: The whole picture of the architecture

The Webserver Monitor has listeners that will be triggered by the CEP Engine (powered by Esper) whenever the predefined conditions are met based on received events from the Webserver. These triggers are basically functions that display the alerts to the appropriate table in the Dashboard. From the Dashboard, administrators can also set new thresholds for each level of severity. For example, one can set up a rule that if 100 failed logins against one account occur in 10 seconds, a high priority alert of a brute-force attack will pop up in the alert table.

2.1.2 Detect potential port scans

On the second subsystem, Port Scan Detector monitors the network packets transferred between the internet and the intranet, between the computers in the intranet, and alerts users when a port scan is detected.

The network traffic is monitored using `pcap4j` [3]. We filter out all TCP Packets and run them through the CEP Engine. Comparing to Figure 1.2, pcap4j takes the role of a data collector.

The Port Scan Detector also has the same structure as the Webserver Monitor, it has listeners that will be triggered whenever a pattern of a mass port scanning is matched. For example, one can set up a rule that if 500 port scans on a single machine occur in 5 seconds, The printed alerts and the options for modifying the thresholds of the Port Scan Detector are also placed in the same Dashboard as the Webserver Monitor's.

2.2 Features

While discussing the architectures of the two subsystems in section 2.1, we gave a few examples of setting thresholds for certain events. This section will list out an extensive list of events catchable by the Port Scan Detector and Webserver Monitor, as well as showcase more convenient functions in the Dashboard.

2.2.1 Webserver

In order to demonstrate the capabilities of our Webserver Monitor, we have provided a Webserver running in PHP. It is a minimalistic website with basic features such as login, register, and change password. There is no database implemented and all account credentials are hard-coded. One could login using one of the usernames `thien_hoang`, `hieu_le`, `khuong_lu`, or `quan_nguyen` and password `neptune`.

2.2.2 Webserver Monitor

The Webserver Monitor can recognize several types of actions done by webserver visitor:

- Brute force attack: A rather popular type of attack where the attacker attempts to log into an account by trying all possible passwords.
- Dictionary attack: This is another common attack where the attacker will use one simple password such as "123456789" and go through thousands of accounts and hack into those actually having this insecure password.
- User base scan: For special servers such as webmail servers, a spammer will try to check if certain usernames exist on the webserver and later send spam emails to these accounts. Though we did not implement a webmail server, this malicious act can be identified with someone attempting to register too many times.
- Success change password: This simply means that someone has changed their password successfully. We track this event because it is also crucial to know if this change is benign or a result of a successful attack.
- User hacked: If we detect a brute-force attack on an account, followed by a success password change, that is a very confident signal of an account getting hacked.
- Resources missing: Sometimes, while deploying the webserver from development environment to production environment, we lost some files/pages, whether intentionally or accidentally. However, old links to those lost resources would not be fixed anywhere. If there were an excessive amount of 404 error on a file is reported, we would raise an alert to let the admin know and add the corresponding resources or adding redirects to the new ones.

For each type of events, the system administrator can set up a threshold to raise an alert in term of the *occurrence rate*, i.e. number of events in a specified duration. Please refer back to subsection 2.1.1 for an example of the occurrence rate.

2.2.3 Port Scan Detector

In a normal client-server model, in order to communicate, the client must first establish a TCP connection by performing a so-called *TCP Handshake* with the server [4]. In order to do that, the client must specify the IP address and the port of the server to which it wants to connect.

If the port is closed, the server will refuse to do a handshake; if open, the server will be happy to connect. However, in a scenario where the client only wants to check the port status without actually establishing any connection, the client will respond to that happiness with a "reset" packet. This is a signal of a port scan, or more accurately, a *SYN scanning* [5]. There are other types of port scans as well, to name a few: UDP scanning, TCP scanning, etc. However, since we only implemented our Port Scan Detector to detect SYN port scan, it will be henceforth shortly called as *port scan* only.

The Port Scan Detector can recognize 4 types of actions:

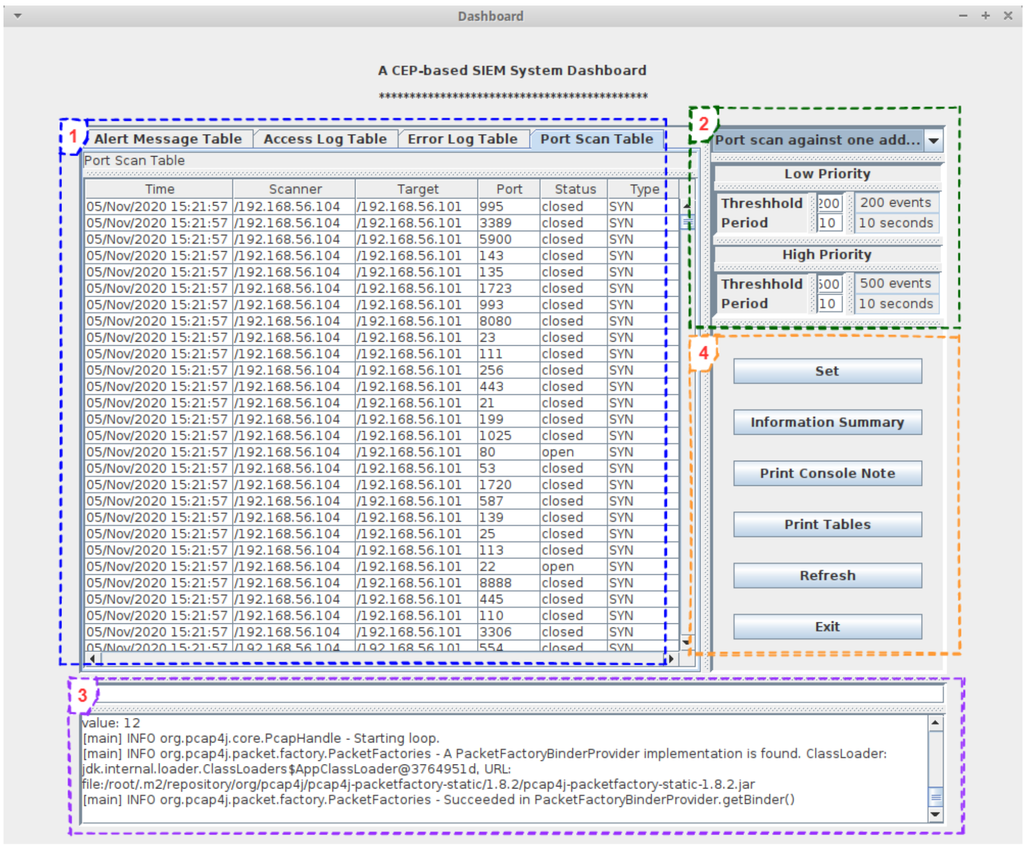
- Single SYN port scan: as described above – whenever a client initiates a TCP Handshake but it was not completed, either the port is closed or the client closed the connection, an event of this type will be raised.
- Vertical port scan: Repeated port scans on a single IP address
- Horizontal port scan: Repeated port scans on a single port in the network
- Block port scan: Repeated port scans on different IP addresses, different ports as well.

Just like the case with Webserver Monitor, one can set up a threshold in occurrence rate for each type of events as well.

2.2.4 Dashboard

The Dashboard is a single-window desktop application intended for a system administrator to monitor incoming events and alerts. There are 4 main areas of functions on the Dashboard, marked on Figure 2.2:

1. Event panel: Consists of 4 tables in total, each table is contained in a separate tab. Every table has the first column being the timestamp of an event/alert. However, they also differ in the structure of the content they present
   - Port scan table: Every time someone scans a port in our network, a row will be added here. The table shows the scanner IP address, the target IP address, the port scanned, its status, and lastly the category of the scan.



Port Scan Table

Time	Scanner	Target	Port	Status	Type
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	995	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	3389	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	5900	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	143	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	135	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	1723	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	993	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	8080	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	23	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	111	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	256	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	443	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	21	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	199	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	1025	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	80	open	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	53	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	1720	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	587	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	139	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	25	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	113	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	22	open	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	8888	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	445	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	110	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	3306	closed	SYN
05/Nov/2020 15:21:57	/192.168.56.104	/192.168.56.101	554	closed	SYN

Low Priority		
Threshold	200	200 events
Period	10	10 seconds
High Priority		
Threshold	500	500 events
Period	10	10 seconds

```
value: 12
[main] INFO org.pcap4j.core.PcapHandle - Starting loop.
[main] INFO org.pcap4j.packet.factory.PacketFactories - A PacketFactoryBinderProvider implementation is found. ClassLoader:
jdk.internal.loader.ClassLoaders$AppClassLoader@3764951d, URL:
file:/root/.m2/repository/org/pcap4j/pcap4j-packetfactory-static/1.8.2/pcap4j-packetfactory-static-1.8.2.jar
[main] INFO org.pcap4j.packet.factory.PacketFactories - Succeeded in PacketFactoryBinderProvider.getBinder()
```

FIGURE 2.2: A screenshot of the dashboard during a port scan

- Error log table: Every time the PHP scripts in our Webserver produce a log entry in `error.log`, this table will be updated. It shows the client address, the URL, and the message included in the log entry.
- Access log table: Every time a person access a file in our Webserver, whether it is a web page or an image, the request will show up here. Basic information of the request is shown: client address, URL, request method, and status code of the response.
- Alert message table: The default-chosen table to display when starting the Dashboard. Each row in this table is an alert produced when repetitive malicious events happen. It informs the severity (priority) of the alert and the inference message that the SIEM system could make out of the observed events.

2. Parameter controller: The system administrator can choose the event to raise alert in the drop-down list and set the threshold and the period (i.e. occurrence rate) beyond which an alert will show up. There are two levels of thresholds: Low priority and High priority. One can change the item in the drop-down list, change the thresholds, click "Set" and change to another item to set up for a different event. The Dashboard "remembers" what was the last set parameters and will show them next time the same item in the list is chosen. The drop-down list includes:

- Bad requests
- Failed logins on one username
- Failed logins on one password
- Failed registrations from one client
- Port scans against one address
- Port scans against one port

3. Console: Redirect all texts from the standard output stream and error stream to this box. The console is robust enough to deal with massive output at once without lagging. Above the big console box is a small search bar where one could enter a keyword and any line in the console containing this word will be filtered and highlighted.

4. Utility buttons: Apart from the "Set" button that has just been introduced, other buttons are also useful for summarizing and extract data:

   - Information summary: Summarize number of records shown in each table and display it in the console at the bottom of the screen.
   - Print console note: Export everything shown in the console to a file.
   - Print tables: Export all three tables to .csv files.
   - Refresh: Clear all the content in all tables and in the console. This has no effect on the set thresholds.
   - Exit: Exit the application gracefully.

Chapter 3

Implementation details

KHUONG LU, HIEU LE, AND THIEN HOANG

We have discussed a lot about the high-level functions of the SIEM system. In this chapter, we will dive into the implementation details of each component to see how it works.

3.1 Dependencies

Our application is programmed in Java 14 [6]. The Webserver is programmed in PHP [7] and deployed with Apache2 [2]. Below is a list of 3rd-party packages that Neptune's Watcher is built upon.

- Esper [8] (`com.espertech`): Esper is an implementation of CEP Engine that we adopt in our project. The artifacts include `esper-common`, `esper-compiler`, and `esper-runtime`.
- Slf4j (`org.slf4j`): One of the most popular logging libraries. The artifacts include `slf4j-api` and `slf4j-simple`. This is actually a dependency of Esper packages, we did not directly use this library.
- Pcap4j [3] (`org.pcap4j`): Pcap4j helps sniff traffic in the network and return the packets to us. The artifacts include `pcap4j-core` and `pcap4j-packetfactory-static`
- FastCSV [9] (`de.siegmar`): We use this robust *io* library to write tables to CSV files. The only artifact is `fastcsv`.

However, as a user, one does not have to install these manually. We use Maven [10] as our build automation tool and all of these dependencies will be taken care while building.

3.2 Project structure

Below is the directory structure of our project. It is a typical skeleton just like most of Java/Maven projects.

```
SIEM-Neptune/
├─ html/
│  ├─ main.php
│  └─ ...
├─ src/
│  ├─ main/
│  │  ├─ java/
│  │  │  ├─ CEP/
│  │  │  │  ├─ PortScanDetector/
│  │  │  │  │  ├─ Detector.java
│  │  │  │  │  └─ ...
│  │  │  │  └─ WebserverMonitor/
│  │  │  │     ├─ Monitor.java
│  │  │  │     └─ ...
│  │  │  ├─ Dashboard/
│  │  │  │  ├─ Dashboard.java
│  │  │  │  └─ ...
│  │  │  └─ Utilities/
│  │  │     ├─ EPAdapter.java
│  │  │     └─ ...
│  │  └─ resources/
│  │     └─ *.epl
│  └─ test/...
└─ pom.xml
```

The folder `html` contains the implementation of our Webserver. The `src/main` folder is the SIEM system, i.e. Neptune's Watcher. The source code is in the sub-folder `java` and the resources (EPL files) are in `resources`. The SIEM system has 4 main packages:

- PortScanDetector
- WebserverMonitor
- `Dashboard`
- Utilities

While the names of the packages are self-explanatory, we would like to underline a few points:

- The two subpackages of `CEP`, PortScanDetector and WebserverMonitor, are responsible for defining the structures of raw events, setting up event streams in the shared environment of Esper and the event listeners. See more at section 3.5 and section 3.4.
- The Utilities package contains adapter classes and utility functions. See more at section 3.6.
- The `Dashboard` package contains implementation of the user interface. Unlike the other three packages listed above, this package does not have a dedicated section in Appendix C because of its complexity and we think that a reference manual for that is rather unnecessary.

3.3 Webserver

Our Webserver is powered by Apache2 [2] and PHP [7].

The landing page of our website is a simple login form so that users can type in their username and password. We also have a register button for new users to join our web server, but for the scope of this project, no new user will be created. For checking the validity of the account, we use PHP to check whether the username exists and also the correctness of the password. Because we don't allow a new account, we have already created 4 default accounts for our team members along with the default password. If the user is successfully logged in, he will be redirected to the welcome page, otherwise, an error message will be raised, this message is written directly to the `error.log` file.

To register a new account, we will check whether the registered username matches any of the 4 default accounts, if it does, an error message will be displayed to tell that that username exists. We also check if the password for the new account is typed and re-typed correctly.

On the welcome page, if you logged in as `thien_hoang`, you will see a link to a secret page. There is also a logout button to end the session and also a change password button. Users who access this link directly without logging in will be redirected to the `require_login` page to enter their credentials. For the secret page, those who are not logged in will also be redirected to the `require_login` page, and if the account that is used to log in is not `thien_hoang`, they will receive a 403 error message.

For the change password page, we will first verify the old password and then check if the new password is typed and re-typed correctly. But for this project, the password won't be changed, it only writes into the error.log file that the password has been changed successfully. This function will be used to check whether the account has been hacked or not in our system.

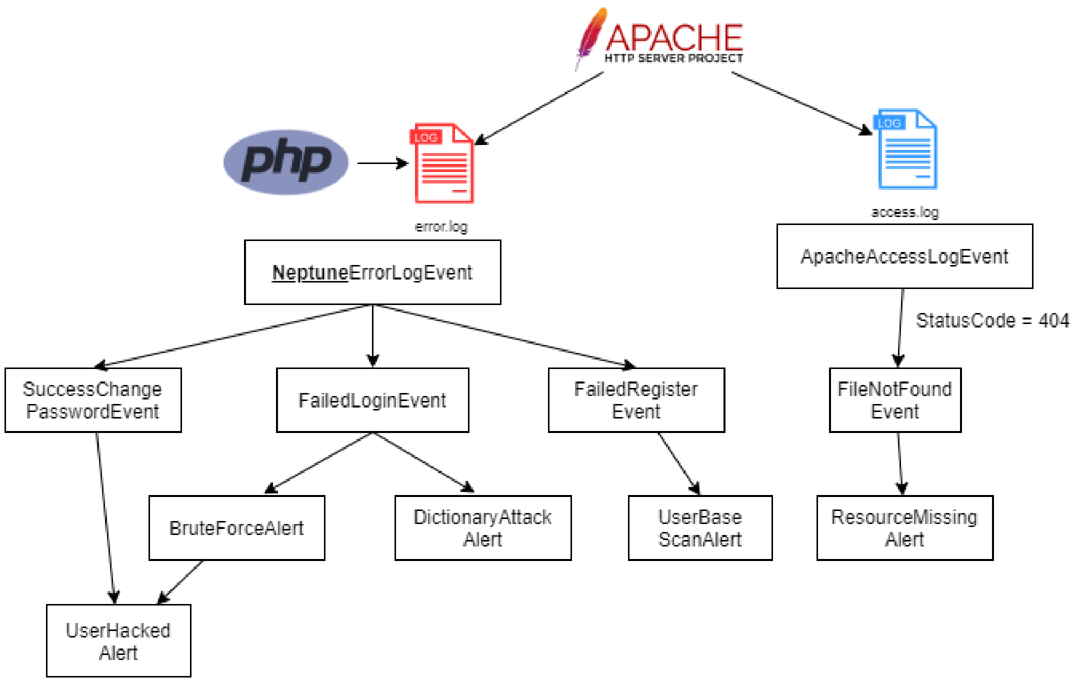


FIGURE 3.1: Event hierarchy of the Webserver Monitor subsystem

3.4 Webserver Monitor

3.4.1 Event collector

Recall from subsection 2.1.1, our Webserver Monitor will read the two log files—`access.log` and `error.log`—and run them through the CEP Engine. In the `access.log` file, we can get the timestamp, request method, HTTP status code, client address, and also the URL of the webserver. And for the `error.log`, we focus on getting the message that the web server returns to us, e.g. when the user types in the wrong password or enters a wrong username. With the help of regular expressions, these tasks are trivial to complete.

We took advantage of the command `tail` in Linux to obtain the last rows of a file, thus getting rid of the responsibility to go through the whole file on our own. When we start the program, we have the current timestamp assigned to a variable called `now`. When we read the first log entry, we compare its timestamp to `now`, only if it is later than `now` do we accept this log entry and send it to the CEP Engine. Furthermore, we will update `now` to be that timestamp and the next log entry must have its timestamp larger than the previous timestamp in order to be accepted.

Of course, we would not read only one line at a time because there may be a lot of events coming in at once, thus reading only the latest event would cause us to miss other events arriving at virtually the same time.

The functions NeptuneErrorLogEvent.nextEvent() and ApacheAccessLogEvent.nextEvent() read a handful of events in their respective log files every time they are called, but only return the earliest-not-considered event[1]. All other events are kept in a queue that will be popped out one by one.

The reading of log entries is done in a separate thread to ensure the concurrency of Webserver Monitor and Port Scan Detector.

3.4.2 Event hierarchy

Each entry in the file `error.log` or `access.log` is considered a raw event, parsed into an object of type NeptuneErrorLogEvent or ApacheAccessLogEvent, respectively (Figure 3.1). Next, we will be investigating each of these events.

ApacheAccessLogEvent

As mentioned earlier, with the help of regular expressions, we split each line of this file into five groups, each group contains different information, such as timestamp, client address, HTTP status code, URL, request method. Then we bundle this information together as an object. Please refer to subsection C.3.2 to see the structure of the event.

FileNotFoundEvent

We filter out any `ApacheAccessLogEvents` that has `httpStatusCode` = 404 and classify them as `FileNotFoundEvents`

ResourceMissingAlert

Based on the given threshold and period, we will create a window of the latest `FileNotFoundEvents`, then if the number of records in that window crosses the threshold, an event `ResourceMissingAlert` will be raised. The event only contains the URL of the resource which has been repeatedly retrieved unsuccessfully.

```
@public CREATE WINDOW FileNotFoundEvent_Latest.win:time($period) AS
    FileNotFoundEvent;
INSERT INTO FileNotFoundEvent_Latest SELECT * FROM FileNotFoundEvent;
INSERT INTO ResourceMissingAlert SELECT url FROM
    FileNotFoundEvent_Latest WHERE count(*) >= $threshold GROUP BY url;
```

and then we can attach a trigger to `ResourceMissingAlert` table by using the EPAdapter class.

```
new EPAdapter().execute("SELECT * FROM ResourceMissingAlert").
    addListener((data, __, ___, ____) -> {
     DashboardAdapter.alertHigh("Likely missing file: " + data[0].get("
    url"));
});
```

[1]an event that has not processed since the start of the program; if there are many, choose the one with earliest timestamp

NeptuneErrorLogEvent

We also use regular expressions to extract information from the second file, which is the `error.log` file, such as timestamp, client address, client port, log message, and the URL of the Webserver. Please refer to subsection C.3.5 to see the structure of the event.

SuccessChangePasswordEvent

Derived from NeptuneErrorLogEvent, those whose log message has the pattern

```
Successfully changed password for <username>.
```

will be pushed into `SuccessChangePasswordEvent`. The event only contains the username who changed the password. We extract the username from the log message by using the utility function parseSuccessChangePassword(String logMsg).

```
@public INSERT INTO SuccessChangePasswordEvent SELECT MessageParser.
    parseSuccessChangePassword(message) as username FROM NELEvent WHERE
     message like 'Successfully changed password for %';
```

FailedLoginEvent

Derived from NeptuneErrorLogEvent, those whose log message has the pattern

```
Successfully changed password for <username>.
```

will be pushed into `SuccessChangePasswordEvent`. The event only contains the username who changed the password. We extract the username from the log message by using the utility function parseSuccessChangePassword(logMsg).

```
@public INSERT INTO SuccessChangePassword SELECT MessageParser.
    parseSuccessChangePassword(message) as username FROM NELEvent WHERE
     message like 'Successfully changed password for %';
```

FailedLoginEvent

Derived from NeptuneErrorLogEvent, those whose log message has the pattern

```
Failed login for username <username> and password (md5 hashed)
<password>
```

will be pushed into `FailedLoginEvent`. The event only contains the username and password (hashed because it is insecure to store passwords in plaintexts) that were entered. We extract the username/password from the log message by using the utility function parseFailedLogin(logMsg, 1) and parseFailedLogin(logMsg, 2), respectively.

```
@public INSERT INTO FailedLoginEvent SELECT MessageParser.
    parseFailedLogin(message, 1) as username, MessageParser.
    parseFailedLogin(message, 2) as password FROM NELEvent WHERE
    message like 'Failed login for username % and password (md5 hashed)
     %';
```

FailedRegisterEvent

Derived from NeptuneErrorLogEvent, those whose log message has the pattern

```
Failed to register for <username>. Account already exists.
```

will be pushed into `FailedRegisterEvent`. The event only contains the username that was entered during the failed registration and the client address who tried to register. We extract the username from the log message by using the utility function parseSuccessChangePassword(logMsg). As for the client address, we already had that information in `NELEvent`.

```
@public INSERT INTO FailedLoginEvent SELECT MessageParser.
    parseFailedRegister(message) as username, clientAddress FROM
    NELEvent WHERE message like 'Failed to register for %. Account
    already exists.';
```

BruteForceAlert

We will keep a window of recent `FailedLoginEvents` (within the set period) and count if the number of attempts on a certain account crosses the set threshold, then we would issue an alert.

```
@public CREATE WINDOW FailedLoginEvent_Latest_1.win:time($period) AS
    FailedLoginEvent;
INSERT INTO FailedLoginEvent_Latest_1 SELECT * FROM FailedLoginEvent;
INSERT INTO BruteForceAlert SELECT username FROM
    FailedLoginEvent_Latest_1 WHERE count(*) >= $threshold GROUP BY
    username;
```

and then we can attach a trigger to `BruteForceAlert` through EPAdapter:

```
new EPAdapter().execute("select * from BruteForceAlert").addListener((
    data, __, ___, ____) -> {
     DashboardAdapter.alertHigh("There seems to be a brute-force attack
    on " + data[0].get("username"));
});
```

We will explain in a moment why we had the suffix `_1` as in `FailedLoginEvent_Latest_1`

DictionaryAttackAlert

We will keep a window of recent `FailedLoginEvents` (within the set period) and count if the number of attempts on a certain password crosses the set threshold, then we would issue an alert.

```
@public CREATE WINDOW FailedLoginEvent_Latest_2.win:time($period) AS
    FailedLoginEvent;
INSERT INTO FailedLoginEvent_Latest_2 SELECT * FROM FailedLoginEvent;
INSERT INTO DictionaryAttackAlert SELECT password FROM
    FailedLoginEvent_Latest_2 WHERE count(*) >= $threshold GROUP BY
    password;
```

and then we can attach a trigger to `DictionaryAttackAlert` through EPAdapter:

```
new EPAdapter().execute("select * from DictionaryAttackAlert").
    addListener((data, __, ___, ____) -> {
     DashboardAdapter.alertHigh("Someone is trying to scan the accounts
    with password = " + data[0].get("password"));
});
```

Note that we used the suffix `_2` in order to separate the windows of `FailedLoginEvent` for two different purposes—`BruteForceAlert` and `DictionaryAttackAlert`. Because each of the two events may require different periods, thus different time windows.

UserBaseScanAlert

We will keep a window of recent `FailedRegisterEvents` (within the set period) and count if the number of attempts from a certain client address crosses the set threshold, then we would issue an alert.

```
@public CREATE WINDOW FailedRegisterEvent_Latest.win:time($period) AS
    FailedRegisterEvent;
INSERT INTO FailedRegisterEvent_Latest SELECT * FROM
    FailedRegisterEvent;
INSERT INTO UserBaseScanAlert SELECT username FROM
    FailedRegisterEvent_Latest WHERE count(*) >= $threshold GROUP BY
    clientAddress;
```

and then we can attach a trigger to `UserBaseScanAlert` through EPAdapter:

```
new EPAdapter().execute("select * from UserBaseScanAlert").addListener
    ((data, __, ___, ____) -> {
     DashboardAdapter.alertHigh(data[0].get("clientAddress") + " may be
    trying to crawl the user base of your platform.");
});
```

UserHackedAlert

This alert will be issued whenever a brute force followed by a change password on the same account. We used a feature of EPL statement called “pattern” to accomplish the goal:

```
@public INSERT INTO UserHackedAlert SELECT A.username as username FROM
    PATTERN[EVERY A=BruteForceAlert -> B=SuccessChangePasswordEvent(
    username=A.username)];
```

and then we can attach a trigger to `UserHackedAlert` through EPAdapter:

```
new EPAdapter().execute("select * from UserHackedAlert").addListener((
    data, __, ___, ____) -> {
    DashboardAdapter.alertHigh(data[0].get("username") + " may have
    been hacked because I detected a brute-force attack earlier and a
    successful password change.");
});
```

3.4.3 Multi-tier alerts

As we have mentioned, the user can set up threshold for low priority alerts and high priority alerts. What we have shown in the subsection 3.4.2 is the settings for high priority alerts. For low priority, we only need to do the same, with different window names, different alert event stream names (e.g. differentiate between `BruteForceAlert_LowPriority` and `BruteForceAlert_HighPriority`), different alert messages. However, we would not dive too deep into that and only presented an overview of our idea in defining the events, actual implementation may vary slightly.

3.4.4 Delete old events

After an alert is issued, we would remove all events that made up the alert in its respective window, for example:

```
ON BruteForceAlert AS A DELETE FROM FailedLoginEvent_Latest_1 AS B
    WHERE A.username=B.username
```

This would prevent a screen full of alerts.

3.5 Port Scan Detector

3.5.1 Event collector

In this project, we use the Java library called Pcap4J [3] in order to capture packets that are sent to our computer. This library also allows us to get information about the packet such as source and destination IP address as well as the port number. We used the IP packet in its library to get the source and destination IP addresses in the IP header, and also used the TCP Packet to get the source and destination port numbers in the TCP header. Moreover, we also take the ACK (Acknowledge), SYN (Synchronize), RST (Reset) flags in the TCP header for detection of port scans.

Firstly, every computer has more than one network interface and at the same time, not all of them are used simultaneously, only some are used to receive packets from the internet. In order to capture the packets, we have to specify which network interface is working at the moment, such as wifi (`enp0s3`, `en0`) or Ethernet, etc. However, for the scope of this project, we decided to use `any`, which is an available choice for all computers, where the traffic of all network interfaces is monitored.

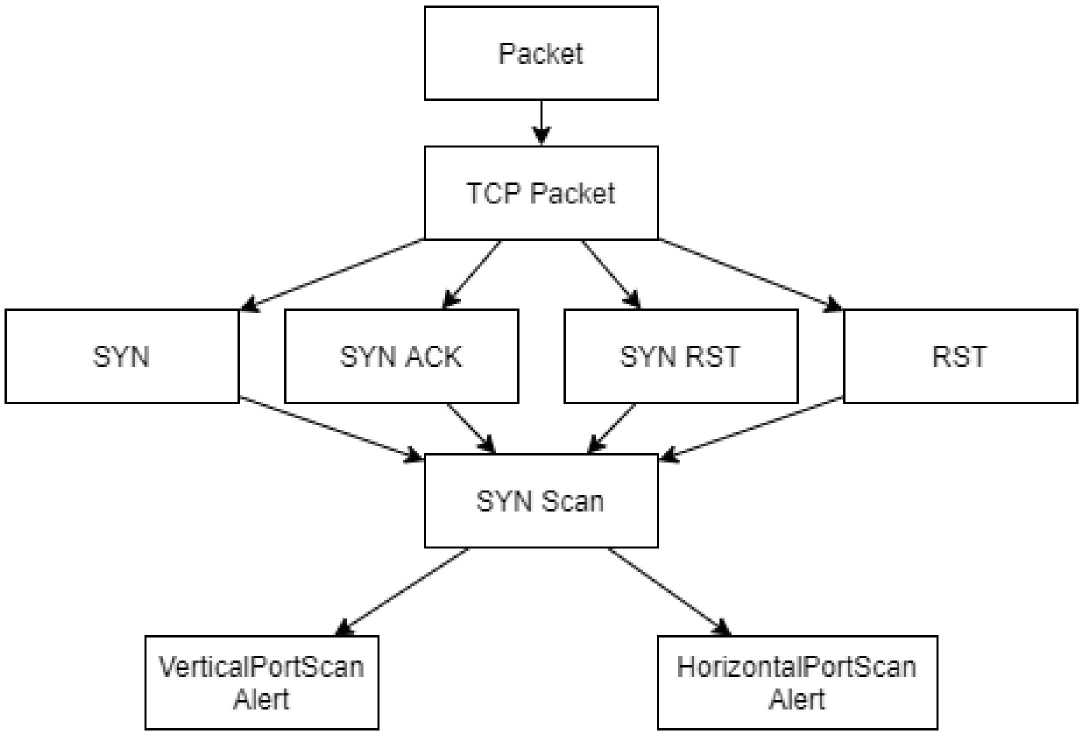


FIGURE 3.2: Event hierarchy in the Port Scan Detector subsystem

To set up a handler to collect packets that are sent to the device, we used `PcapHandle` in the Pcap4J library with Promiscuous Mode, which enabled our computer to capture all network traffic and also filter only TCP packets that are sent in the network. For each packet that we sniffed, we put them into the CEP engine for calculation.

3.5.2 Event hierarchy

Each TCP packet sniffed by Pcap4j will be parsed into an object of type TCPPacketEvent, which includes IP address and Port of sender and receiver, and the four aformentioned flags in the TCP header. Please refer to subsection C.1.4 to see the structure of the event.

SYN, SYN-ACK, SYN-RST, RST Events

Based on the flags of the packet, we will classify them into 4 derived events: SYN, SYN-ACK, SYN-RST, and RST Events.

```
@public insert into SYN_Event select * from TCPPacket_Event(ack=false,
    fin=false,rst=false,syn=true);
@public insert into ACK_SYN_Event select * from TCPPacket_Event(ack=
    true,fin=false,rst=false,syn=true);
@public insert into ACK_RST_Event select * from TCPPacket_Event(ack=
    true,fin=false,rst=true,syn=false);
@public insert into RST_Event select * from TCPPacket_Event(ack=false,
    fin=false,rst=true,syn=false);
```

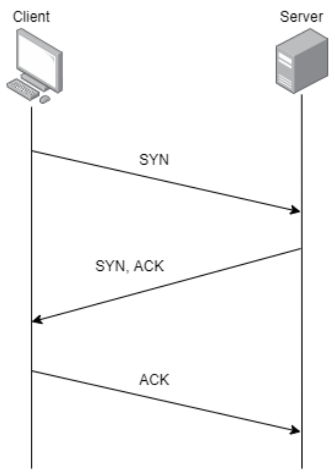


FIGURE 3.3: TCP Handshake

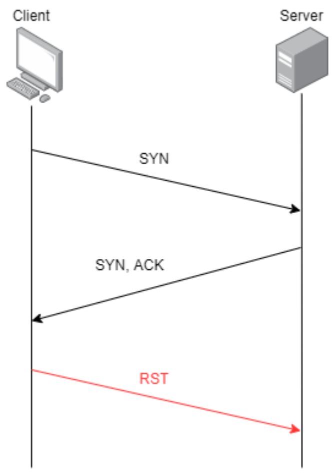


FIGURE 3.4: Port scan when the port is open

SYN Scan

We would like to briefly explain the TCP Handshake (Figure 3.3). It is a common mechanism to establish connection between client and server. The client would send a SYN packet to the server, the server responds with SYN-ACK packet, and the client finalizes with an ACK packet.

However, in a malicious action such as port scan, either the server would reject the client because the port is closed (Figure 3.5), or the client would terminate the connection right after receiving SYN-ACK packet from the server (Figure 3.4).

We would take advantage of these patterns and define our `SYNScanEvent` as follows:

```
@public insert into SYNScanEvent select current_timestamp() as
    timestamp, 'SYN' as type, 'closed' as status, A.dstPort as
    targetPort, A.dstAddress as targetAddress, A.srcAddress as scanner
    from pattern[every A=SYN_Event -> B=ACK_RST_Event(srcAddress=A.
    dstAddress, srcPort=A.dstPort, dstAddress=A.srcAddress, dstPort=A.
    srcPort)].win:time(1 minute);
```

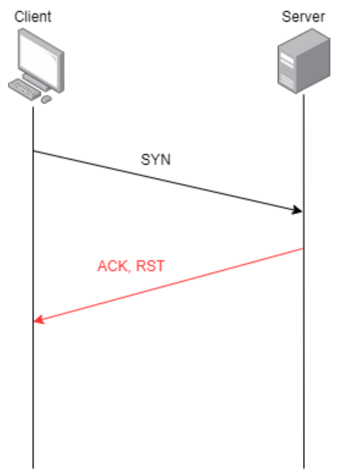


FIGURE 3.5: Port scan when the port is closed

```
@public insert into SYNScanEvent select current_timestamp() as
    timestamp, 'SYN' as type, 'open' as status, A.dstPort as targetPort
    , A.dstAddress as targetAddress, A.srcAddress as scanner from
    pattern[every A=SYN_Event -> B=ACK_SYN_Event(srcAddress=A.
    dstAddress, srcPort=A.dstPort, dstAddress=A.srcAddress, dstPort=A.
    srcPort) -> C=RST_Event(srcAddress=B.dstAddress, srcPort=B.dstPort,
     dstAddress=B.srcAddress, dstPort=B.srcPort)].win:time(1 minute);
```

Our SYNScanEvent will have three properties: timestamp of the scan, target IP address, and target port. We decided not to include the IP address/Port of the attacker because the mass scanning could be done in a distributed manner.

VerticalPortScanAlert

We will keep a window of recent SYNScanEvents (within the set period) and count if the number of attempts on a target IP address crosses the set threshold, then we would issue an alert.

```
@public CREATE WINDOW SYNScanEvent_Latest_1.win:time($period) AS
    SYNScanEvent;
INSERT INTO SYNScanEvent_Latest_1 SELECT * FROM SYNScanEvent;
INSERT INTO VerticalPortScanAlert SELECT targetAddress FROM
    SYNScanEvent_Latest_1 WHERE count(distinct targetPort) >=
    $threshold GROUP BY targetAddress;
```

and then we can attach a trigger to VerticalPortScanAlert through EPAdapter:

```
new EPAdapter().execute("select * from VerticalPortScanAlert").
    addListener((data, __, ___, ____) -> {
     DashboardAdapter.alertHigh(data[0].get("targetAddress") + " is
    under a vertical port scan.");
});
```

HorizontalPortScanAlert

We will keep a window of recent `SYNScanEvents` (within the set period) and count if the number of attempts on a target port crosses the set threshold, then we would issue an alert.

```
@public CREATE WINDOW SYNScanEvent_Latest_2.win:time($period) AS
    SYNScanEvent;
INSERT INTO SYNScanEvent_Latest_2 SELECT * FROM SYNScanEvent;
INSERT INTO HorizontalPortScanAlert SELECT targetAddress FROM
    SYNScanEvent_Latest_2 WHERE count(distinct targetAddress) >=
    $threshold GROUP BY targetPort;
```

and then we can attach a trigger to `HorizontalPortScanAlert` through EPAdapter:

```
new EPAdapter().execute("select * from HorizontalPortScanAlert").
    addListener((data, __, ___, ____) -> {
     DashboardAdapter.alertHigh("Port " + data[0].get("targetPort") + "
    is under a horizontal port scan.");
});
```

3.5.3 Multi-tier alerts and deleting old events

Similar to the Webserver Monitor subsystem, in this subsystem we also implement multi-tier alerts and delete old events in the same manner.

3.6 Dashboard

3.6.1 Display events and alerts

The dashboard can automatically display unlimited alert and raw events in a table format with different tables in different tabs including Alert Message, Access Log, Error Log and Port Scan Tables. The tables will update in real time and scroll to the latest events. You can also edit the size of the columns and the order of columns by drag and drop it to the order you want.

Whenever a new row is added to a table, it will get the maximum length of the table and then trigger the scroll to scroll down to the bottom of the table for displaying the last event. When we click the scroll, it will delete the current position data so we can scroll up to the position we want later.

3.6.2 Set the configuration of the SIEM system

Firstly, you can choose the type of events that you want to change thresholds (Figure 3.6). If the names of the events are too long, it will automatically expand the size of the drop-down box so that you can see the full text. Besides that, if you put too many events to choose, the scroll will also show up to limit the size of the drop-down. We use `BoundsPopupMenuListener` to create these features.

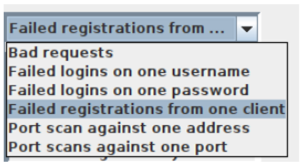

Failed registrations from ...
Bad requests
Failed logins on one username
Failed logins on one password
Failed registrations from one client
Port scan against one address
Port scans against one port

FIGURE 3.6: Choosing event type in the Dashboard

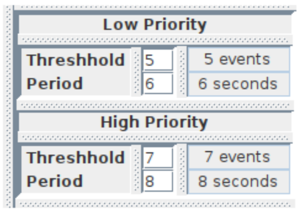


FIGURE 3.7: Set and display properties for high and low priority events in the Dashboard

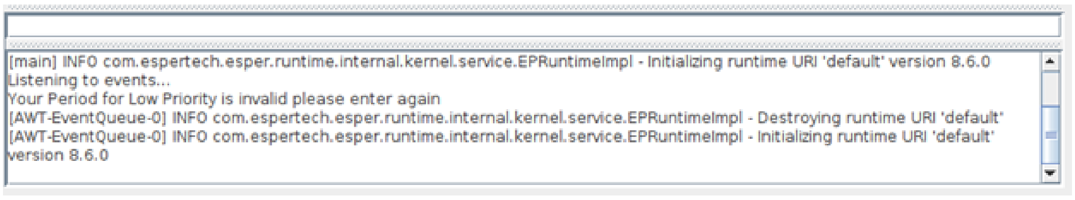
After that, you edit the attributes for displaying low and or priority (Figure 3.7) and click the “Set” Button to change the threshold of the system. The information will be stored and displayed to the users. When you change between events, the systems will receive the index of the drop-down box and use it to take the information from data lists so that the displayed information will be set to its current event. Besides that, it will also send the information to the Webserver Monitor and the Port Scan Detector to change the system’s configuration.

Your input for threshold and period will be all checked when you click the “Set” button by `isNumeric(String str)` function. If you enter null, non-numeric or negative variables, it will trigger the `printEnterAgain(String object, String variable)` function so that the displayed text will not change and an alert will be printed in the console to ask you to enter the input again.

3.6.3 Buttons for other options

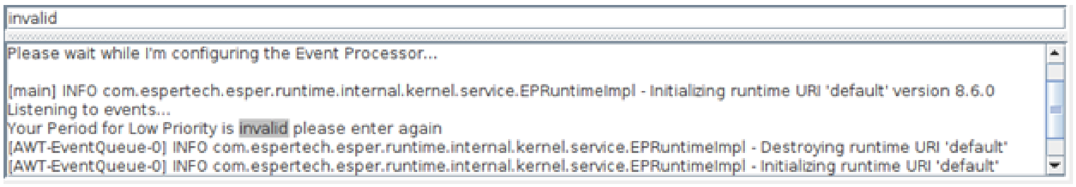
There are total 6 buttons that you can choose from includes:

- The set buttons will set thresholds for the systems.
- The Information Summary button will print the number and period of processed events for each event type in the console by counting the number of rows in each event table for the number of events, and also take the time of the first and last events to determine their period of time.
- The Print Console note will print the note in the console. It will create the file name “console.txt” if the file does not exist or overwrite it in another case.
- The Print Tables will print the all 4 tables in .csv format. We first get the information from the tables and then write it in the csv format in 4 different csv



```
[main] INFO com.espertech.esper.runtime.internal.kernel.service.EPRuntimeImpl - Initializing runtime URI 'default' version 8.6.0
Listening to events...
Your Period for Low Priority is invalid please enter again
[AWT-EventQueue-0] INFO com.espertech.esper.runtime.internal.kernel.service.EPRuntimeImpl - Destroying runtime URI 'default'
[AWT-EventQueue-0] INFO com.espertech.esper.runtime.internal.kernel.service.EPRuntimeImpl - Initializing runtime URI 'default'
version 8.6.0
```

FIGURE 3.8: Search bar and console in the Dashboard



```
invalid
```

```
Please wait while I'm configuring the Event Processor...

[main] INFO com.espertech.esper.runtime.internal.kernel.service.EPRuntimeImpl - Initializing runtime URI 'default' version 8.6.0
Listening to events...
Your Period for Low Priority is invalid please enter again
[AWT-EventQueue-0] INFO com.espertech.esper.runtime.internal.kernel.service.EPRuntimeImpl - Destroying runtime URI 'default'
[AWT-EventQueue-0] INFO com.espertech.esper.runtime.internal.kernel.service.EPRuntimeImpl - Initializing runtime URI 'default'
```

FIGURE 3.9: Searching for texts in console in the Dashboard

files. It will create the file name from “table0.txt” to “table3.txt” with the content of Alert Message Table, Access Log Table, Error Log Table, Port Scan Table respectively if the files do not exist or overwrite them in the remaining case.

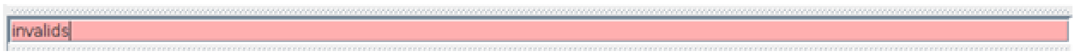
- The Refresh buttons will clear all the information in tables and console.
- The Exit Button will make us exit from the program.

3.6.4 Search bar and console note

In order to make the dashboard more convenient for the users, we have created a search bar for searching text in the console and a console which can operate like a normal console (Figure 3.8). We create a custom output stream, redirect data to the text area, scroll the text area to the end of data when a new line is added, and keep the text up to date. We can also set the limit of lines by using `LimitLinesDocumentListener`. If the new line exceeds the limit, it will remove the oldest line instead.

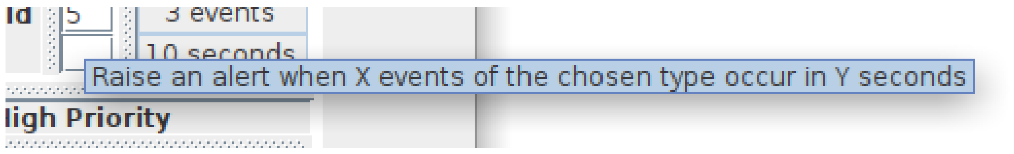
Whenever you type a text that exists in the console, the console will then jump to the text's location and the text will be highlighted. You can then press “ENTER” for searching for the next appearance or press “ESC” to clear all the text in the search bar and begin a new search. (Figure 3.9)

Firstly, it will search for the similar string in the console, jump, highlight it and store the location. When we press “ENTER”, it will then remove the highlight from



```
invalids
```

FIGURE 3.10: Searching text does not exist in the console in the Dashboard



Raise an alert when X events of the chosen type occur in Y seconds

FIGURE 3.11: An example tool tip in the Dashboard

previous words, and do the same function for searching the next word from the end of the previous words.

When we press the "ESC" button on the keyboard, it will trigger the Cancel Action function which clears all the current highlight and text in the search bar.

If you type a text that does not exist in the console, the search bar will change its color to red that alerts you the word does not exist. (Figure 3.10)

The Information can be dragged and dropped between the tables, the search bar and the console. The console can also be edited and printed for taking note but in the newest version, this feature has been turned off due to the lagging of the console.

3.6.5 Tool tips

Most of the components will have tool tips to help the user understand more about some functionalities of the dashboard. (Figure 3.11)

3.7 Utilities

EPAdapter will help executing EPL statements by preparing environment, compiling, and deploying, removing the hassels for the developers. All statements will be executed in the same environment, thus all event stream names must be distinct. Furthermore, as Esper requires each statement to have a name, while it is not a matter for us, we would have a function to generate random names assigned to the statements.

We also have `MessageParser` as an utility class imported to the CEP Engine, so that it can be used to extract username and password from NeptuneErrorLogEvent's log message, as previously shown.

The class `DashboardAdapter` will be responsible for displaying the alerts on the Dashboard. It is by default "disabled" and can be enable by setDisabled(false). It is only enabled when we run `Dashboard.main(String args)`. This is intentional since we may wish to run the Webserver Monitor or Port Scan Detector without initiating a Dashboard.

Chapter 4

Evaluation & Conclusion

4.1 Testing

Because of the limited time, we could not fully roll out a set of unit tests for our project. Most of our testing is done manually, though we do have some unit tests in parsing log entries as well as to test the runnability of the CEP engine (i.e. compile-error-free EPL statements).

4.2 Known limitations

We were not able to test the `HorizontalPortScanAlert` (section 3.5.2) thoroughly because we could not configure the network between the VMs. Ideally, we would implement the CEP Engine to be running on a separate machine where it receives information from various machines in the network, instead of sniffing packets sending to those machines. Instead, we only had all components running all the same machine.

4.3 Extendability

There are two points of improvement that we would make if we had more time:

- Make the system running remotely, receiving events from multiple endpoints, analyzing, and reporting to the Dashboard.
- Implement more types of port scans. Possibilities include TCP Scanning and UDP Scanning.

4.4 Conclusion

Given a period of one month with limited knowledge about Complex Event Processing and Security Information and Event Management , I believe what we have done is a great success. The project was held completely online, communicating may be difficult, but most team members have done a good job in completing their tasks.

This page was intentionally left blank.

Appendix A

Member contribution

We would like to acknowledge the contribution of every team member in this appendix.

A.1 Hieu Le

He had single-handedly built the whole Dashboard without any help from any team member. He was also responsible for any documents related to the Dashboard presented in this report. Though the description of his job was short, the volume of his work was arguably considerable for the success of this project.

A.2 Khuong Lu

Khuong contributed significantly to the application, in both Webserver Monitor and Port Scan Detector subsystems. He did a lot of experiments in configuring CEP Engine, sniffing packets with Pcap4j, etc. That had opened our path to the final solutions.

In terms of documentation, Khuong had written for many sections, most notably the event collector sections (subsection 3.4.1 and subsection 3.5.1).

A.3 Quan Nguyen

Quan had helped writing the log entry parser using regular expression. He also helped in testing the application and writing some introductory sections in our report.

A.4 Thien Hoang

As a leader of the team, he had helped planning the project and assigning tasks. He was responsible for merging commits and merging documentation from individual members to this final report. He was the creator of the package Utilities, which had facilitated the development process, as well as the multi-tier alerts feature for the application.

This page was intentionally left blank.

Appendix B

User manual

THIEN HOANG& QUAN NGUYEN

B.1 Installation

B.1.1 Neptune's Watcher

In the following steps, you will be able to build an executable .jar file from the source code. However, we also provide the executable file as well, just in case you are not able to build it from the source code.

First, install Maven and JRE 14:

```
$ sudo apt update
$ sudo apt install maven
$ sudo apt install openjdk-14-jre
```

Set up JAVA_HOME variable:

```
$ export JAVA_HOME=/usr/lib/jvm/java-14-openjdk-amd64
$ echo $JAVA_HOME # /usr/lib/jvm/java-14-openjdk-amd64
```

Then, navigate to the root folder of SIEM-Neptune and run

```
$ mvn clean install
```

It may takes a few minutes to download all the dependencies and bundle them into a .jar file. After the building succeeds, execute the .jar file:

```
$ sudo java -jar target/SIEM-Neptune-1.0-SNAPSHOT.jar
```

The Dashboard should be displayed on the screen right away

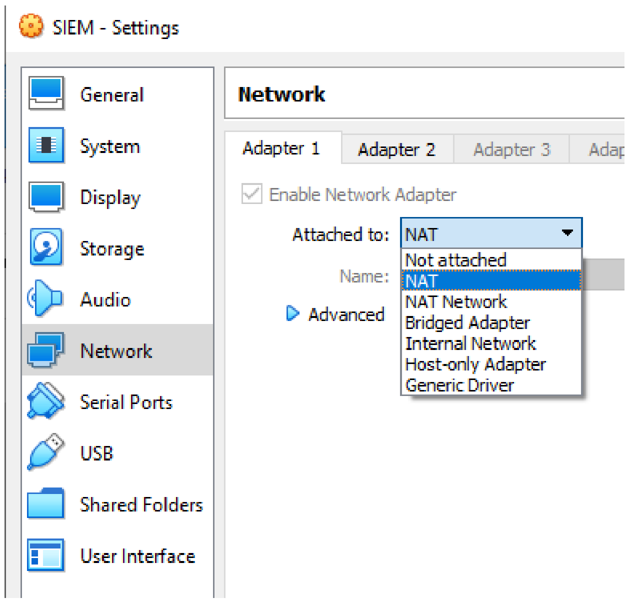
B.1.2 Webserver

The Webserver is built upon PHP and Apache2, thus you need to install them in order to deploy the Webserver:

```
$ sudo apt install apache2
$ sudo apt install php libapache2-mod-php
$ sudo systemctl restart apache2
```

Then, copy the folder html to replace the default html folder of Apache2:

```
$ sudo cp html /var/www/
```



Network

Enable Network Adapter

Attached to: NAT

Name:

Not attached
NAT
NAT Network
Bridged Adapter
Internal Network
Host-only Adapter
Generic Driver

FIGURE B.1: Set up network adapters for a VirtualBox VM

At this point, the Webserver should be up and running. Open your browser and type in `localhost` to see if it's working.

We highly advise the use of virtual machine to run the Neptune's Watcher and Webserver (on the same machine[1]). In this case, you may run `ifconfig` to see the IP address of the VM and configure your VM's networks to be NAT (Figure B.1) and Host-only Adapter (Figure B.2) with Promiscuous Mode = Allow All. We will explain shortly why.

B.1.3 Port scan

You will need to install `nmap` to help you do a port scan.

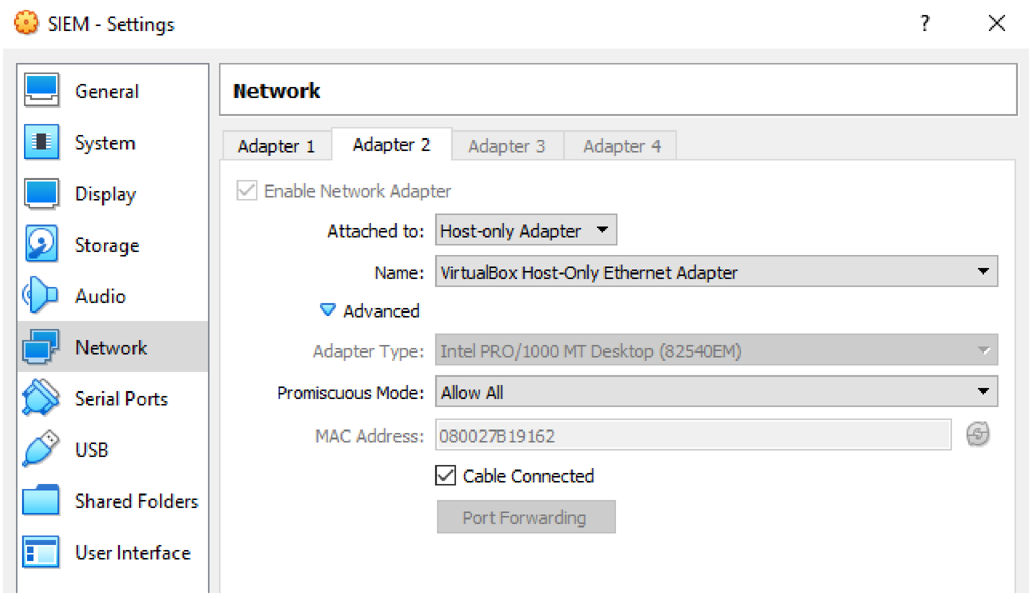
```
sudo apt-get install nmap
```

Ideally this should be installed on a different machine other than the one running Neptune's Watcher. You may start a different VM[2] and run `nmap` from there to scan the ports of A. Machine B's network is also set up as A's. The reason behind this configuration is that we want all VMs to be able to access the Internet via the the host machine, and the Promiscuous Mode turned on to all so that each VM can sniff those packets in the network that are not related to them. This is crucial because we want machine A to be able to detect any port scan to machine B as well.

B.2 Demonstration

Here is a short guide to experience the features of Neptune's Watcher.

[1]In this manual we will refer to this as machine A
[2]in this manual we will refer to this as machine B



Network

Enable Network Adapter ✓

Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

Advanced

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

Promiscuous Mode: Allow All

MAC Address: 080027B19162

Cable Connected ✓

FIGURE B.2: Set up network adapters for a VirtualBox VM

1. Navigate to the main website `localhost/`, if it doesn't work, try to append `main.php` at the end.

2. Observe the Access Log Table, you will see a few rows show up because there are requests to access the web page, the favicon, etc.

3. Intentionally type in an incorrect combination of username and password, you will see the Access Log Tableand Error Log Tableeach has a new row. Recall that you can log in with username being one of our names, e.g. `khuong_lu`, and password being `neptune`.

4. Intentionally fail to login for 5 times with 5 **different** passwords in 10 seconds, you will see two new alerts regarding brute-force attack in the Alert Message Table, one with Low priority—issued after the moment you made the 3rd failed attempt, one with High priority—issued after the 5th time.

5. On the 6th time, enter the correct password, you will see in Error Log Tablea message telling that you have logged in successfully.

6. Click "Change password" and change it. Of course this will not change anything but the system will recognize a successful action. Check in the Alert Message Tableyou will see a High priority alert regarding the possibility of some user got hacked.

You can also try:

- Register for an account with an existing username. Repeated the action with **different** usernames and you will see a few alerts popping up.

- Try to access a non-existent page such as `neptune.php`, you will receive a 404 error. Repeatedly refresh this page will create alerts in the Alert Message Table.

For port scan, first you will need to find the IP address of A via `ifconfig`, suppose for the moment it is `192.168.56.101`. From machine B, run the following command:

```
sudo nmap -sS 192.168.56.101 -p 22
```

This is to scan the port 22 (typical SSH port) to see if it is open. You will see a new entry in the Port Scan Table. The option `-sS` is basically instruct `nmap` to use the SYN scanning technique, which Neptune's Watcher can detect. Using other types of scanning does not guarantee success because we have not programmed that path yet.

You may try to do a vertical scan against machine A, i.e. check over 1000 ports of A and report which ports are open.

```
sudo nmap -sS 192.168.56.101
```

As you do that, the Port Scan Tablewill quickly be filled with new entries. More specifically, 1000 new rows will be inserted (but that would not cause any lagging, so feel free to test). (Figure 2.2)

Appendix C

JavaDoc documentation

C.1 Package CEP.PortScanDetector

Package Contents *Page*

C.1.1 Class Detector

main class of the Port Scan Detector component

Declaration

```
public class Detector
 extends java.lang.Object
```

Constructor summary

Detector()

Method summary

execute() Run the Port Scan Detector

main(String[]) main function if you want to run the Port Scan Detector alone

Constructors

- **Detector**

```
public Detector()
```

Methods

- **execute**

```
public static void execute() throws EPCompileException, java.io.
    IOException, EPDeployException, PcapNativeException,
    NotOpenException, java.lang.InterruptedException,
    ParseException
```

  - **Description**

    Run the Port Scan Detector

  - **Throws**
    - `EPCompileException` –
    - `java.io.IOException` –
    - `EPDeployException` –
    - `PcapNativeException` –
    - `NotOpenException` –
    - `java.lang.InterruptedException` –
    - `ParseException` –

- **main**

```
public static void main(java.lang.String[] args) throws
    EPCompileException, java.io.IOException, EPDeployException,
    PcapNativeException, java.lang.InterruptedException,
    NotOpenException, ParseException
```

  - **Description**

    main function if you want to run the Port Scan Detector alone

  - **Parameters**
    - `args` – command line arguments

  - **Throws**
    - `EPCompileException` –
    - `java.io.IOException` –

  * `EPDeployException` –
  * `PcapNativeException` –
  * `java.lang.InterruptedException` –
  * `NotOpenException` –
  * `ParseException` –

C.1.2 Class HorizontalPortScanCEP

Facade class to set up the CEP Engine to catch Horizontal Port Scan The setup() function must be called after SinglePortScanCEP.setup()

Declaration

```
public class HorizontalPortScanCEP
 extends java.lang.Object
```

Constructor summary

HorizontalPortScanCEP()

Method summary

getPeriod() Get the current periods after which old packets (used to detect Horizontal Port Scan) will expire

getThreshold() Get the current thresholds (number of different machines that got scanned) over which a horizontal port scan alert will be raised

setPeriod(int[]) Set the new periods after which old packets (used to detect Horizontal Port Scan) will expire

setThreshold(int[]) Set the new thresholds (number of different machines that got scanned) over which a horizontal port scan alert will be raised

setup() Set up the event streams in the CEP Engine with EPL Statement and some listeners

Constructors

- **HorizontalPortScanCEP**

```
public HorizontalPortScanCEP()
```

Methods

- **getPeriod**

```
public static int[] getPeriod()
```

  - **Description**
    Get the current periods after which old packets (used to detect Horizontal Port Scan) will expire
  - **Returns** – [period_lowPriority, period_highPriority]

- **getThreshold**

```
public static int[] getThreshold()
```

  - **Description**
    Get the current thresholds (number of different machines that got scanned) over which a horizontal port scan alert will be raised
  - **Returns** – [threshold_lowPriority, threshold_highPriority]

- **setPeriod**

```
public static void setPeriod(int[] period)
```

  - **Description**
    Set the new periods after which old packets (used to detect Horizontal Port Scan) will expire
  - **Parameters**
    * `period` – [period_lowPriority, period_highPriority]

- **setThreshold**

```
public static void setThreshold(int[] threshold)
```

  - **Description**
    Set the new thresholds (number of different machines that got scanned) over which a horizontal port scan alert will be raised
  - **Parameters**
    * `threshold` – [threshold_lowPriority, threshold_highPriority]

- **setup**

```java
public static void setup() throws EPCompileException,
    EPDeployException
```

- **Description**

  Set up the event streams in the CEP Engine with EPL Statement and some listeners

- **Throws**
  * `EPCompileException` –
  * `EPDeployException` –

C.1.3 Class SinglePortScanCEP

Facade class to set up the CEP Engine to analyze and classify the incoming TCP Packet

Declaration

```java
public class SinglePortScanCEP
 extends java.lang.Object
```

Constructor summary

SinglePortScanCEP()

Method summary

setup() Set up the event streams in the CEP Engine with EPL Statement and some listeners

Constructors

- **SinglePortScanCEP**

```java
public SinglePortScanCEP()
```

Methods

- **setup**

```java
public static void setup() throws EPCompileException,
    EPDeployException, java.io.IOException, ParseException
```

- **Description**

  Set up the event streams in the CEP Engine with EPL Statement and some listeners

- **Throws**

  * `EPCompileException` –
  * `EPDeployException` –
  * `java.io.IOException` –
  * `ParseException` –

C.1.4 Class TCPPacketEvent

Represent a packet transferred in the network using TCP

Declaration

```
public class TCPPacketEvent
 extends java.lang.Object
```

Constructor summary

TCPPacketEvent(IpPacket.IpHeader, TcpPacket.TcpHeader) Constructor Extract the information from the IP header and the TCP header of the IP packet and bundle them into this POJO Note that in an IP packet, a TCP packet is wrapped inside

Method summary

getDstAddress() Get IP address of the receiver
getDstPort() Get Port of the receiver
getSrcAddress() Get IP address of the sender
getSrcPort() Get Port of the sender
isAck() Get ACK flag
isFin() Get FIN flag
isRst() Get RST flag
isSyn() Get SYN flag
setAck(boolean) Set ACK flag
setDstAddress(String) Set IP address of the receiver
setDstPort(int) Set Port of the receiver
setFin(boolean) Set FIN flag
setRst(boolean) Set RST flag
setSrcAddress(String) Set IP address of the sender
setSrcPort(int) Set Port of the sender
setSyn(boolean) Set SYN flag
toString() Display the packet nicely

Constructors

- **TCPPacketEvent**

```
public TCPPacketEvent(IpPacket.IpHeader ipHeader,TcpPacket.
    TcpHeader tcpHeader)
```

  - **Description**
    Constructor Extract the information from the IP header and the TCP header of the IP packet and bundle them into this POJO Note that in an IP packet, a TCP packet is wrapped inside
  - **Parameters**
    * `ipHeader` – IP Header of the packet
    * `tcpHeader` – TCP Header of the packet

Methods

- **getDstAddress**

```
public java.lang.String getDstAddress()
```

  - **Description**
    Get IP address of the receiver
  - **Returns** – IP address of the receiver

- **getDstPort**

```
public int getDstPort()
```

  - **Description**
    Get Port of the receiver
  - **Returns** – Port of the receiver

- **getSrcAddress**

```
public java.lang.String getSrcAddress()
```

  - **Description**
    Get IP address of the sender
  - **Returns** – IP address of the sender

- **getSrcPort**

```java
public int getSrcPort()
```

  - **Description**
    Get Port of the sender
  - **Returns** – Port of the sender

- **isAck**

```java
public boolean isAck()
```

  - **Description**
    Get ACK flag
  - **Returns** – true iff ACK=1 in the packet

- **isFin**

```java
public boolean isFin()
```

  - **Description**
    Get FIN flag
  - **Returns** – true iff FIN=1 in the packet

- **isRst**

```java
public boolean isRst()
```

  - **Description**
    Get RST flag
  - **Returns** – true iff RST=1 in the packet

- **isSyn**

```java
public boolean isSyn()
```

  - **Description**
    Get SYN flag
  - **Returns** – true iff SYN=1 in the packet

- **setAck**

```java
public void setAck(boolean ack)
```

  - **Description**
    Set ACK flag
  - **Parameters**
    * `ack` – true iff ACK=1 in the packet

- **setDstAddress**

```java
public void setDstAddress(java.lang.String dstAddress)
```

  - **Description**
    Set IP address of the receiver
  - **Parameters**
    * `dstAddress` – IP address of the receiver

- **setDstPort**

```java
public void setDstPort(int dstPort)
```

  - **Description**
    Set Port of the receiver
  - **Parameters**
    * `dstPort` – Port of the receiver

- **setFin**

```java
public void setFin(boolean fin)
```

  - **Description**
    Set FIN flag
  - **Parameters**
    * `fin` – true iff FIN=1 in the packet

- **setRst**

```java
public void setRst(boolean rst)
```

  - **Description**

    Set RST flag
  - **Parameters**
    - `rst` – true iff RST=1 in the packet

- **setSrcAddress**

```java
public void setSrcAddress(java.lang.String srcAddress)
```

  - **Description**

    Set IP address of the sender
  - **Parameters**
    - `srcAddress` – IP address of the sender

- **setSrcPort**

```java
public void setSrcPort(int srcPort)
```

  - **Description**

    Set Port of the sender
  - **Parameters**
    - `srcPort` – Port of the sender

- **setSyn**

```java
public void setSyn(boolean syn)
```

  - **Description**

    Set SYN flag
  - **Parameters**
    - `syn` – true iff SYN=1 in the packet

- **toString**

```java
public java.lang.String toString()
```

  - **Description**

    Display the packet nicely
  - **Returns** – information of the packet

C.1.5 Class VerticalPortScanCEP

Facade class to set up the CEP Engine to detect Vertical Port Scan

Declaration

```
public class VerticalPortScanCEP
 extends java.lang.Object
```

Constructor summary

VerticalPortScanCEP()

Method summary

getPeriod() Get the current periods after which old packets (used to detect Vertical Port Scan) will expire
getThreshold() Get the current thresholds (number of different ports that got scanned) over which a horizontal port scan alert will be raised
setPeriod(int[]) Set the new periods after which old packets (used to detect Vertical Port Scan) will expire
setThreshold(int[]) Set the new thresholds (number of different ports that got scanned) over which a horizontal port scan alert will be raised
setup() Set up the event streams in the CEP Engine with EPL Statement and some listeners

Constructors

- **VerticalPortScanCEP**

```
public VerticalPortScanCEP()
```

Methods

- **getPeriod**

```
public static int[] getPeriod()
```

  - **Description**
    Get the current periods after which old packets (used to detect Vertical Port Scan) will expire
  - **Returns** – [period_lowPriority, period_highPriority]

- **getThreshold**

```
public static int[] getThreshold()
```

  - **Description**
    Get the current thresholds (number of different ports that got scanned) over which a horizontal port scan alert will be raised
  - **Returns** – [threshold_lowPriority, threshold_highPriority]

- **setPeriod**

```
public static void setPeriod(int[] period)
```

  - **Description**
    Set the new periods after which old packets (used to detect Vertical Port Scan) will expire
  - **Parameters**
    * `period` – [period_lowPriority, period_highPriority]

- **setThreshold**

```
public static void setThreshold(int[] threshold)
```

  - **Description**
    Set the new thresholds (number of different ports that got scanned) over which a horizontal port scan alert will be raised
  - **Parameters**
    * `threshold` –

- **setup**

```
public static void setup() throws EPCompileException,
   EPDeployException
```

  - **Description**
    Set up the event streams in the CEP Engine with EPL Statement and some listeners
  - **Throws**
    * `EPCompileException` –
    * `EPDeployException` –

C.2 Package Utilities

Package Contents *Page*

C.2.1 Class DashboardAdapter

Adapter class that helps deliver the alerts and events from the CEP-triggers to the dashboard

Declaration

```java
public class DashboardAdapter
 extends java.lang.Object
```

Field summary

ACCESS_LOG_TABLE
ALERT_TABLE
ERROR_LOG_TABLE
PORT_SCAN_TABLE

Constructor summary

DashboardAdapter()

Method summary

alertHigh(String)
alertLow(String)
setDisabled(boolean)
writeToTable(EventBean, int)

Fields

- public static final int **ALERT_TABLE**
- public static final int **ACCESS_LOG_TABLE**
- public static final int **ERROR_LOG_TABLE**
- public static final int **PORT_SCAN_TABLE**

Constructors

- **DashboardAdapter**

```
public DashboardAdapter()
```

Methods

- **alertHigh**

```
public static void alertHigh(java.lang.String message)
```

- **alertLow**

```
public static void alertLow(java.lang.String message)
```

- **setDisabled**

```
public static void setDisabled(boolean disabled)
```

- **writeToTable**

```
public static void writeToTable(EventBean o,int i)
```

C.2.2 Class EPAdapter

Adapter class that helps executing EPL statements with less effort

Declaration

```
public class EPAdapter
 extends java.lang.Object
```

Field summary

arguments Singleton arguments
compiler Singleton compiler
configuration Singleton configuration
runtime Singleton runtime

Constructor summary

EPAdapter() Constructor Start creating the environment iff it does not exist

Method summary

addListener(UpdateListener) Attach a listener to the statement belong to this instance.
destroy() Destroy the environment That includes compiler, configuration, runtime, and arguments.
execute(String) Execute an EPL statement in the shared environment
executeFile(String) Execute all statements in a file in the shared environment With this method, you will not be able to attach listener to any of the statements in the file.
quickExecute(String[]) Fire-and-forget execution Execute an EPL statement to which you do not intend to attach any listener
sendEvent(EventType, String) Send a POJO to an event stream in CEP
setup() Prepare the environment That includes compiler, configuration, runtime, and arguments.

Fields

- `public static EPCompiler` **compiler**
  - Singleton compiler
- `public static Configuration` **configuration**
  - Singleton configuration
- `public static CompilerArguments` **arguments**
  - Singleton arguments
- `public static EPRuntime` **runtime**
  - Singleton runtime

Constructors

- **EPAdapter**

```java
public EPAdapter()
```

  - **Description**
    Constructor Start creating the environment iff it does not exist

Methods

- **addListener**

```java
public void addListener(UpdateListener listener)
```

  - **Description**
    Attach a listener to the statement belong to this instance. Normally you would call this immediately after executing a statement. For example: new EPAdapter().execute("SELECT * FROM TCPPacketEvent").addListener(...)
  - **Parameters**
    * `listener` – preferably a lambda function with 4 parameters

- **destroy**

```java
public static void destroy()
```

  - **Description**
    Destroy the environment That includes compiler, configuration, runtime, and arguments.

- **execute**

```java
public EPAdapter execute(java.lang.String statement) throws
    EPCompileException, EPDeployException
```

  - **Description**
    Execute an EPL statement in the shared environment
  - **Parameters**
    * `statement` – the EPL statement
  - **Returns** – an instance of the adapter, which you can use to attach a listener with addListener

- **Throws**
  - EPCompileException –
  - EPDeployException –

- **executeFile**

```java
public static void executeFile(java.lang.String filename) throws
    java.io.IOException, ParseException, EPCompileException,
    EPDeployException
```

- **Description**

  Execute all statements in a file in the shared environment With this method, you will not be able to attach listener to any of the statements in the file.

- **Parameters**
  - filename – the relative path to the file with respect to the resources folder

- **Throws**
  - java.io.IOException –
  - ParseException –
  - EPCompileException –
  - EPDeployException –

- **quickExecute**

```java
public static void quickExecute(java.lang.String[] statements)
    throws EPCompileException, EPDeployException
```

- **Description**

  Fire-and-forget execution Execute an EPL statement to which you do not intend to attach any listener

- **Parameters**
  - statements – the EPL statement

- **sendEvent**

```java
public static void sendEvent(java.lang.Object event,java.lang.
    String eventType)
```

- **Description**

  Send a POJO to an event stream in CEP

- **Parameters**

    * `event` – the POJO
    * `eventType` – the name of the event stream set up in CEP

- **setup**

```
public static void setup()
```

  - **Description**
    Prepare the environment That includes compiler, configuration, runtime, and arguments.

C.2.3 Class MessageParser

Utility class that is used by the CEP to extract the information from the error log's message

Declaration

```
public class MessageParser
 extends java.lang.Object
```

Constructor summary

MessageParser()

Method summary

parseFailedLogin(String, int) Extract username or password from the message of a log entry related to a failed login attempt
parseFailedRegister(String) Extract username from the message of a log entry related to a failed register due to duplication
parseSuccessChangePassword(String) Extract username from the message of a log entry related to a successful change in password

Constructors

- **MessageParser**

```
public MessageParser()
```

Methods

- **parseFailedLogin**

```java
public static java.lang.String parseFailedLogin(java.lang.String
    message,int group)
```

  - **Description**
    Extract username or password from the message of a log entry related to a failed login attempt
  - **Parameters**
    * `message` – the log message
    * `group` – 1 if you want to get username, 2 if you want to get password
  - **Returns** – the requested information

- **parseFailedRegister**

```java
public static java.lang.String parseFailedRegister(java.lang.
    String message)
```

  - **Description**
    Extract username from the message of a log entry related to a failed register due to duplication
  - **Parameters**
    * `message` – the log message
  - **Returns** – the existed username

- **parseSuccessChangePassword**

```java
public static java.lang.String parseSuccessChangePassword(java.
    lang.String message)
```

  - **Description**
    Extract username from the message of a log entry related to a successful change in password
  - **Parameters**
    * `message` – the log message
  - **Returns** – the username who changed password successfully

C.2.4 Class Misc

Miscellaneous utility class

Declaration

```
public class Misc
 extends java.lang.Object
```

Constructor summary

Misc()

Method summary

formatTime(long)

Constructors

- **Misc**

```
public Misc()
```

Methods

- **formatTime**

```
public static java.lang.String formatTime(long timestamp)
```

C.3 Package CEP.WebserverMonitor

Package Contents *Page*

C.3.1 Class ApacheAccessLogCEP

Facade class to set up the CEP Engine to analyze the requests made to the Webserver

Declaration

```
public class ApacheAccessLogCEP
 extends java.lang.Object
```

Constructor summary

ApacheAccessLogCEP()

Method summary

getPeriod() Get the current periods after which old events will expire
getThreshold() Get the current thresholds (number of 404 responses) over which a resource missing alert will be raised
setPeriod(int[]) Set the new periods after which old events will expire
setThreshold(int[]) Set the new thresholds (number of 404 responses) over which a resource missing alert will be raised
setup() Set up the event streams in the CEP Engine with EPL Statement and some listeners

Constructors

- **ApacheAccessLogCEP**

```
public ApacheAccessLogCEP()
```

Methods

- **getPeriod**

```
public static int[] getPeriod()
```

  - **Description**
    Get the current periods after which old events will expire
  - **Returns** – [period_lowPriority, period_highPriority]

- **getThreshold**

```
public static int[] getThreshold()
```

  - **Description**

    Get the current thresholds (number of 404 responses) over which a resource missing alert will be raised
  - **Returns** – [threshold_lowPriority, threshold_highPriority]

- **setPeriod**

```java
public static void setPeriod(int[] period)
```

  - **Description**

    Set the new periods after which old events will expire
  - **Parameters**
    - `period` – [period_lowPriority, period_highPriority]

- **setThreshold**

```java
public static void setThreshold(int[] threshold)
```

  - **Description**

    Set the new thresholds (number of 404 responses) over which a resource missing alert will be raised
  - **Parameters**
    - `threshold` – [threshold_lowPriority, threshold_highPriority]

- **setup**

```java
public static void setup() throws EPCompileException,
    EPDeployException
```

  - **Description**

    Set up the event streams in the CEP Engine with EPL Statement and some listeners
  - **Throws**
    - `EPCompileException` –
    - `EPDeployException` –

C.3.2 Class ApacheAccessLogEvent

Represent a log entry in the file access.log produced by Apache2

Declaration

```java
public class ApacheAccessLogEvent
 extends java.lang.Object
```

Field summary

REGEXP Regular expression to match with the log entry in access.log

Constructor summary

ApacheAccessLogEvent()
ApacheAccessLogEvent(String) Constructor Create an event object by parsing the log entry

Method summary

getClientAddress() Get client's IP address
getHttpStatusCode() Get the status code to the request
getRequestMethod() Get the request method of the request Example: "POST", "GET", ...
getTimestamp() Get timestamp of the event This is the timestamp recorded in the log entry, converted into number of seconds since Epoch
getUrl() Get the requested URL The URL upon which the client made the request
nextEvent() Get an event from the log file access.log The event will always be the latest event that has not been processed.
setClientAddress(String) Set client's IP address
setHttpStatusCode(String) Set the status code to the request
setRequestMethod(String) Set the request method of the request Example: "POST", "GET", ...
setTimestamp(long) Set timestamp of the event This is the timestamp recorded in the log entry, converted into number of seconds since Epoch
setUrl(String) Set the requested URL The URL upon which the client made the request

Fields

- `public static final java.lang.String` **REGEXP**
  - Regular expression to match with the log entry in access.log

Constructors

- **ApacheAccessLogEvent**

```
public ApacheAccessLogEvent()
```

- **ApacheAccessLogEvent**

```
public ApacheAccessLogEvent(java.lang.String logline) throws java.
    lang.Exception
```

  - **Description**

    Constructor Create an event object by parsing the log entry
  - **Parameters**
    - `logline` – the log entry
  - **Throws**
    - `java.lang.Exception` – when the log entry cannot be parsed

Methods

- **getClientAddress**

```
public java.lang.String getClientAddress()
```

  - **Description**

    Get client's IP address
  - **Returns** – client's IP address

- **getHttpStatusCode**

```
public java.lang.String getHttpStatusCode()
```

  - **Description**

    Get the status code to the request
  - **Returns** – the status code

- **getRequestMethod**

```
public java.lang.String getRequestMethod()
```

  - **Description**

    Get the request method of the request Example: "POST", "GET", ...

  - **Returns** – the request method

- **getTimestamp**

```java
public long getTimestamp()
```

  - **Description**

    Get timestamp of the event This is the timestamp recorded in the log entry, converted into number of seconds since Epoch

  - **Returns** – timestamp of the event

- **getUrl**

```java
public java.lang.String getUrl()
```

  - **Description**

    Get the requested URL The URL upon which the client made the request

  - **Returns** – URL of the request

- **nextEvent**

```java
public static ApacheAccessLogEvent nextEvent() throws java.io.
    IOException
```

  - **Description**

    Get an event from the log file access.log The event will always be the latest event that has not been processed. If there are multiple such events, the earliest event is returned. If there are no new events, null will be returned.

  - **Returns** – null or an event in access.log

  - **Throws**

    * `java.io.IOException` – thrown when failed to read the log file

- **setClientAddress**

```java
public void setClientAddress(java.lang.String clientAddress)
```

  - **Description**

    Set client's IP address

  - **Parameters**

    * `clientAddress` – client's IP address

- **setHttpStatusCode**

```java
public void setHttpStatusCode(java.lang.String httpStatusCode)
```

  - **Description**
    Set the status code to the request
  - **Parameters**
    * `httpStatusCode` – the status code

- **setRequestMethod**

```java
public void setRequestMethod(java.lang.String requestMethod)
```

  - **Description**
    Set the request method of the request Example: “POST”, “GET”, ...
  - **Parameters**
    * `requestMethod` – the request method

- **setTimestamp**

```java
public void setTimestamp(long timestamp)
```

  - **Description**
    Set timestamp of the event This is the timestamp recorded in the log entry, converted into number of seconds since Epoch
  - **Parameters**
    * `timestamp` – timestamp of the event

- **setUrl**

```java
public void setUrl(java.lang.String url)
```

  - **Description**
    Set the requested URL The URL upon which the client made the request
  - **Parameters**
    * `url` – URL of the request

C.3.3 Class Monitor

main class of the Webserver Monitor component

Declaration

```
public class Monitor
 extends java.lang.Object
```

Constructor summary

Monitor()

Method summary

execute() Run the Webserver Monitor
main(String[]) main function if you want to run the Webserver Monitor alone

Constructors

- **Monitor**

```
public Monitor()
```

Methods

- **execute**

```
public static void execute() throws EPCompileException,
    EPDeployException
```

  - **Description**
    Run the Webserver Monitor
  - **Throws**
    * `EPCompileException` –
    * `EPDeployException` –

- **main**

```
public static void main(java.lang.String[] args) throws java.lang.
    Exception
```

  - **Description**
    main function if you want to run the Webserver Monitor alone
  - **Parameters**
    * `args` – command line arguments
  - **Throws**
    * `java.lang.Exception` –

C.3.4 Class NeptuneErrorLogCEP

Facade class to set up the CEP Engine to analyze the events recorded by the Webserver

Declaration

```
public class NeptuneErrorLogCEP
 extends java.lang.Object
```

Constructor summary

NeptuneErrorLogCEP()

Method summary

getBruteForce_period() Get the current periods after which old events (used to detect brute force) will expire

getBruteForce_threshold() Get the current thresholds (number of different passwords) over which a brute-force alert will be raised

getDictAttack_period() Get the current periods after which old events (used to detect dictionary attack) will expire

getDictAttack_threshold() Get the current thresholds (number of different usernames) over which a dictionary attack alert will be raised

getUserBaseScan_period() Get the current periods after which old events (used to detect user base scanning) will expire

getUserBaseScan_threshold() Get the current thresholds (number of different usernames) over which a user base scan alert will be raised

setBruteForce_period(int[]) Set the new periods after which old events (used to detect brute force) will expire

setBruteForce_threshold(int[]) Set the new thresholds (number of different passwords) over which a brute-force alert will be raised

setDictAttack_period(int[]) Set the new periods after which old events (used to detect dictionary attack) will expire

setDictAttack_threshold(int[]) Set the new thresholds (number of different usernames) over which a dictionary attack alert will be raised

setup() Set up the event streams in the CEP Engine with EPL Statement and some listeners

setUserBaseScan_period(int[]) Set the new periods after which old events (used to detect user base scan) will expire

setUserBaseScan_threshold(int[]) Set the new thresholds (number of different usernames) over which a user base scan alert will be raised

Constructors

- **NeptuneErrorLogCEP**

```java
public NeptuneErrorLogCEP()
```

Methods

- **getBruteForce_period**

```java
public static int[] getBruteForce_period()
```

  - **Description**
    Get the current periods after which old events (used to detect brute force) will expire
  - **Returns** – [period_lowPriority, period_highPriority]

- **getBruteForce_threshold**

```java
public static int[] getBruteForce_threshold()
```

  - **Description**
    Get the current thresholds (number of different passwords) over which a brute-force alert will be raised
  - **Returns** – [threshold_lowPriority, threshold_highPriority]

- **getDictAttack_period**

```java
public static int[] getDictAttack_period()
```

  - **Description**
    Get the current periods after which old events (used to detect dictionary attack) will expire
  - **Returns** – [period_lowPriority, period_highPriority]

- **getDictAttack_threshold**

```java
public static int[] getDictAttack_threshold()
```

  - **Description**
    Get the current thresholds (number of different usernames) over which a dictionary attack alert will be raised

  - **Returns** – [threshold_lowPriority, threshold_highPriority]

- **getUserBaseScan_period**

```java
public static int[] getUserBaseScan_period()
```

  - **Description**
    Get the current periods after which old events (used to detect user base scanning) will expire
  - **Returns** – [period_lowPriority, period_highPriority]

- **getUserBaseScan_threshold**

```java
public static int[] getUserBaseScan_threshold()
```

  - **Description**
    Get the current thresholds (number of different usernames) over which a user base scan alert will be raised
  - **Returns** – [threshold_lowPriority, threshold_highPriority]

- **setBruteForce_period**

```java
public static void setBruteForce_period(int[] bruteForce_period)
```

  - **Description**
    Set the new periods after which old events (used to detect brute force) will expire
  - **Parameters**
    * `bruteForce_period` – [period_lowPriority, period_highPriority]

- **setBruteForce_threshold**

```java
public static void setBruteForce_threshold(int[]
    bruteForce_threshold)
```

  - **Description**
    Set the new thresholds (number of different passwords) over which a brute-force alert will be raised
  - **Parameters**
    * `bruteForce_threshold` – [threshold_lowPriority, threshold_highPriority]

- **setDictAttack_period**

```java
public static void setDictAttack_period(int[] dictAttack_period)
```

  - **Description**

    Set the new periods after which old events (used to detect dictionary attack) will expire

  - **Parameters**

    * `dictAttack_period` – [period_lowPriority, period_highPriority]

- **setDictAttack_threshold**

```java
public static void setDictAttack_threshold(int[]
    dictAttack_threshold)
```

  - **Description**

    Set the new thresholds (number of different usernames) over which a dictionary attack alert will be raised

  - **Parameters**

    * `dictAttack_threshold` – [threshold_lowPriority, threshold_highPriority]

- **setup**

```java
public static void setup() throws EPCompileException,
    EPDeployException
```

  - **Description**

    Set up the event streams in the CEP Engine with EPL Statement and some listeners

  - **Throws**

    * `EPCompileException` –
    * `EPDeployException` –

- **setUserBaseScan_period**

```java
public static void setUserBaseScan_period(int[]
    userBaseScan_period)
```

  - **Description**

    Set the new periods after which old events (used to detect user base scan) will expire

- **Parameters**
  - userBaseScan_period – [period_lowPriority, period_highPriority]

- **setUserBaseScan_threshold**

```
public static void setUserBaseScan_threshold(int[]
    userBaseScan_threshold)
```

- **Description**

  Set the new thresholds (number of different usernames) over which a user base scan alert will be raised

- **Parameters**
  - userBaseScan_threshold – [threshold_lowPriority, threshold_highPriority]

C.3.5 Class NeptuneErrorLogEvent

Represent a log entry in the file error.log produced by the Webserver (via PHP's error_log(string) function)

Declaration

```
public class NeptuneErrorLogEvent
 extends java.lang.Object
```

Field summary

REGEXP Regular expression to match with the log entry in error.log

Constructor summary

NeptuneErrorLogEvent()
NeptuneErrorLogEvent(String) Constructor Create an event object by parsing the log entry

Method summary

getClientAddress() Get client's IP address
getMessage() Get the message of the Webserver This message is written to the log by PHP's built in function error_log
getTimestamp() Get timestamp of the event This is the timestamp recorded in the log entry, converted into number of seconds since Epoch

getUrl() Get the URL that recorded this event The URL of the PHP script which wrote the log entry

init(long, String, String, String)

nextEvent() Get an event from the log file error.log The event will always be the latest event that has not been processed.

setClientAddress(String) Set client's IP address

setMessage(String) Set the message of the Webserver This message is written to the log by PHP's built in function error_log

setTimestamp(long) Set timestamp of the event This is the timestamp recorded in the log entry, converted into number of seconds since Epoch

setUrl(String) Set the URL that recorded this event The URL of the PHP script which wrote the log entry

Fields

- `public static final java.lang.String` **REGEXP**
  - Regular expression to match with the log entry in error.log

Constructors

- **NeptuneErrorLogEvent**

```
public NeptuneErrorLogEvent()
```

- **NeptuneErrorLogEvent**

```
public NeptuneErrorLogEvent(java.lang.String logline) throws java.
    lang.Exception
```

  - **Description**
    Constructor Create an event object by parsing the log entry
  - **Parameters**
    * `logline` – the log entry
  - **Throws**
    * `java.lang.Exception` – when the log entry cannot be parsed

Methods

- **getClientAddress**

```
public java.lang.String getClientAddress()
```

  - **Description**
    Get client's IP address
  - **Returns** – client's IP address

- **getMessage**

```
public java.lang.String getMessage()
```

  - **Description**
    Get the message of the Webserver This message is written to the log by PHP's built in function error_log
  - **Returns** – the log message

- **getTimestamp**

```
public long getTimestamp()
```

  - **Description**
    Get timestamp of the event This is the timestamp recorded in the log entry, converted into number of seconds since Epoch
  - **Returns** – timestamp of the event

- **getUrl**

```
public java.lang.String getUrl()
```

  - **Description**
    Get the URL that recorded this event The URL of the PHP script which wrote the log entry
  - **Returns** – URL of the logger

- **init**

```
protected void init(long timestamp,java.lang.String clientAddress,
    java.lang.String message,java.lang.String url)
```

- **nextEvent**

```
public static NeptuneErrorLogEvent nextEvent() throws java.io.
    IOException
```

  - **Description**
    Get an event from the log file error.log The event will always be the latest event that has not been processed. If there are multiple such events, the earliest event is returned. If there are no new events, null will be returned.
  - **Returns** – null or an event in error.log
  - **Throws**
    * `java.io.IOException` – thrown when failed to read the log file

- **setClientAddress**

```
public void setClientAddress(java.lang.String clientAddress)
```

  - **Description**
    Set client's IP address
  - **Parameters**
    * `clientAddress` – client's IP address

- **setMessage**

```
public void setMessage(java.lang.String message)
```

  - **Description**
    Set the message of the Webserver This message is written to the log by PHP's built in function error_log
  - **Parameters**
    * `message` – the log message

- **setTimestamp**

```
public void setTimestamp(long timestamp)
```

  - **Description**
    Set timestamp of the event This is the timestamp recorded in the log entry, converted into number of seconds since Epoch
  - **Parameters**
    * `timestamp` – timestamp of the event

- **setUrl**

```
public void setUrl(java.lang.String url)
```

– **Description**

  Set the URL that recorded this event The URL of the PHP script which wrote the log entry

– **Parameters**

  * `url` – URL of the logger

Bibliography

[1] A. Adi, “Complex event processing,” IBM, Tech. Rep., 2006. [Online]. Available: https://www.research.ibm.com/haifa/dept/services/papers/cep_Jan07_nc.pdf

[2] Apache Software Foundation, “Apache http server software.” [Online]. Available: https://httpd.apache.org

[3] K. Yamada, “pcap4j.” [Online]. Available: https://github.com/kaitoy/pcap4j

[4] L. L. Peterson and B. S. Davie, “5 - end-to-end protocols,” in *Computer Networks (Fifth Edition)*, fifth edition ed., ser. The Morgan Kaufmann Series in Networking, L. L. Peterson and B. S. Davie, Eds. Boston: Morgan Kaufmann, 2012, pp. 391 – 476. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123850591000053

[5] J. Erickson, *Hacking: the art of exploitation*, 2nd ed. No starch press, 2008.

[6] O. C. et al, “Java programming language.” [Online]. Available: https://www.java.com/en/

[7] T. P. Group, “Php programming language.” [Online]. Available: https://www.php.net

[8] EsperTech, “Esper.” [Online]. Available: https://www.espertech.com/esper

[9] O. S. et al, “Fastcsv.” [Online]. Available: https://github.com/osiegmar/FastCSV

[10] A. S. Foundation, “Apache maven.” [Online]. Available: https://maven.apache.org/