# Detecting SQL-Injection and Cross-Site Scripting Attacks Using Case-Based Reasoning and SEASALT

Jakob Michael Schoenborn[1,2,3], Klaus-Dieter Althoff[1,2]

[1]University of Hildesheim, Germany
[2]German Research Center for Artificial Intelligence (DFKI), Germany
[3]Exploit Labs GmbH, Germany

**Abstract**

Since the internet offers a rising amount of services and reachable devices, the amount of criminal activities rises as well. Protective measures such as firewalls and intrusion detection systems are being actively developed. We accompany this development by offering a case-based reasoning approach to detect similar attacks based on previous attacks (cases), starting with cross-site scripting (XSS) and SQL-injection (SQLi). With an instantiation of the SEASALT framework, the foundation for expandability towards further attack vectors, such as authentication testing, can easily be established by adding additional topic agents. Additionally, we propose to distinguish between two different views on network traffic: the request itself, and the traffic overall. The latter enables us to detect timed attacks, e. g, authentication testing by brute-force guessing login credentials, and to identify clients with a suspicious large amount of generated traffic. This paper focuses on the request itself to identify XSS and SQLi attacks - two of the most commonly used attack vectors in the last decade according to the open web application security project (OWASP). As we store cases containing these attacks in our casebases, we are able to detect similar cases. Depending on the use-case, we identified up to 16 relevant attributes, predominantly text attributes. However, the similarity assessment needs improvement to reduce the rate of false-positives.

**Keywords**

Case-Based Reasoning, Network Security, SEASALT, Cross-Site Scripting, SQL-Injection

## 1. Introduction

Not only the events of the current and last year but also the growth of the digitalization in general led to a rapid increase in the usage of online technologies [1, 2]. Visiting a website is based on a client requesting the source behind the addressed URL (usually a .HTML file). Additionally, it results in transferring multiple hundreds of packets of the different protocols in the OSI layer model between the client and the server [3]. Per default, this communication contains seemingly harmless information about the participants, such as the server information, e. g., an Apache/2.4.29 (Ubuntu)[1] server. However, a potential attacker can make use of this information to adjust an attack accordingly - increasing the chance of success. As so-called

*LWDA'21: Lernen, Wissen, Daten, Analysen September 01–03, 2021, Munich, Germany*
schoenb@uni-hildesheim.de (J. M. Schoenborn)
0000-0001-9669-8148 (J. M. Schoenborn)


[1]Which has known vulnerabilities. Vulnerabilities and their corresponding risk assessment are maintained at https://www.cvedetails.com/index.php (last validation 08/18/2021)

human *blue-teamers* are constantly monitoring the network traffic and observing malicious changes in their to be protected host computers, it is not feasible to manage the sheer amount of traffic manually. Blue-teamers rely on warnings raised by firewalls and intrusion detection systems (IDS).

To take a step into the direction of a more efficient network analysis, we propose an instantiation of the "***S**hared **E**xperience using an **A**gent based **S**ystem **A**rchitecture **L**ayou**T***" framework (SEASALT) [4]. As a brief overview, SEASALT uses one coordination agent who controls $n$ topic agents. Topic agents are specialized on machine learning approaches, knowledge-based systems, or any other artificial intelligence technology in general. The agents used here are case-based reasoning (CBR) agents, where each agent maintains the four knowledge containers according to Richter [5]: vocabulary, casebase, similarity measurements, and adaptation knowledge. Using this framework provides the benefit of flexible knowledge management and decentralisation, so that, for example, we can instantiate one agent for cross-site scripting (XSS), one agent for SQL injections (SQLi), and more for any other attack vectors.

Given an input field such as a search function, the search keyword will be passed to the database to look for entries matching the search keyword. The SQL search query may look like

```
SELECT * FROM products WHERE item LIKE 'keyword';
```

As a result, a list of products matching the given keyword will be printed to the graphical user interface. The attacker controls the value of `keyword` (excluding the apostrophes). A SQL injection allows the attacker to interact with the database of the server by escaping the given SQL statement, e. g., by commenting the latter part of the SQL statement and inserting own SQL statements - selecting, updating, deleting, or creating other tables. For example, entering as keyword

```
mask' UNION (SELECT username, password FROM users);--
```

the complete SQL statement reads

```
SELECT * FROM products WHERE item LIKE 'mask' UNION (SELECT username,
↪  password FROM users);--';
```

The attacker receives a list of all products matching the keyword mask *and* all usernames and passwords from the table users. The idea for XSS attacks is basically the same, but targets for client-side information, such as authentication tokens or session cookies.

However, this assumes a table named `users` and the column names `username`, `password` exist. Finding and executing a SQLi often requires a trial & error phase to identify the correct identifiers. Automatization tools such as `sqlmap` simplify this process. These products contain wordlists with known SQLi and automatically try them against the target domain and target input field. Rule-based systems can use these wordlists and detect any requests containing words of these wordlists. Since attackers are aware of this, attackers started to create their own wordlists by slightly changing the contents of the wordlists. This is where a case-based reasoning approach can support other defensive approaches by comparing the current traffic with known, similar SQLi. Whenever, for example, the SQLi agent detects an attack, the agent reports to the coordination agent who not only reports the attack to the user of the system,

but may also initiate further meta investigations to detect other patterns of additional attacks originated from the same client.

The next sections are structured as followed: Section 2 provides a brief overview on the current literature regarding network traffic analysis. Afterwards, we present the theoretical foundation of our system in Section 3 with a main focus on how to fill, use, and maintain the knowledge containers used in CBR. Consequently, Section 4 presents the resulting prototype, using myCBR for the CBR interface and other commonly used tools as network traffic data sources. In Section 5, we present our experiments of the prototype and discuss the results. Closing this contribution, we remain with conclusions and future work in the last Section 6.

## 2. Related work

In this section, we briefly present contributions in the IT security domain using CBR to gain an overview on current approaches and application areas.

Kapetanakis et al. [6] built a profile of a person who tries to intrude into a system by gathering their traces. The assumption is that cyber criminals are different from normal criminals in their behaviour and technical characteristics [6]. To evaluate their CBR approach, 87 participants were asked to attack a target system by shutting down the services running on this computer while each attack was regarded as an individual case for the CBR system containing at least 15 attributes and the outcomes as evidence for the solution part, classified by forensic experts [6]. The idea of criminal cyber profiling has been further investigated by Han et al. [7], also including a CBR approach to categorize traces left by professional advanced persistent threat groups. As our approach investigates the attack vectors itself, a higher level view on the attacker per se seems to be unique in the literature. It remains open for further research whether these could be incorporated to our approach as an additional topic agent.

As we motivated the problems with IDS, there is also work on improving the rule-based systems behind these IDS. More precisely, Schwartz et al. [8] implemented a CBR system to improve the rule-based system Snort to overcome its known limitations regarding false-positives and false-negatives. Another approach for anomaly detection has been provided by Micarelli and Sansonetti [9], who investigate the interaction between users and network configuration based on the queries sent to the system. The similarity is assessed by the Earth Mover's Distance between the corresponding feature distributions [9].

## 3. Conceptual model: framework and knowledge containers

Analyzing the network traffic data requires reliable tools to monitor the ongoing network data. We gather the network data by using Wireshark[2] and Burp[3]. Both aim to capture, filter, and adjust network traffic and are standard tools to use in the domain, with slightly different foci. For a more detailed, technical description, we refer to Section 4 and to our prototype demonstrated at the ICCBR 2021 workshop on CBR demonstrations and showcases [10]. For the conceptual model, these tools allow us to generate structured text data (.CSV and .XML), containing all

[2]For more information, see https://www.wireshark.org/ (last validation 08/18/2021) and Section 4

[3]For more information, see https://portswigger.net/burp (last validation 08/18/2021) and Section 4

important information to automatically generate cases. However, before observing the traffic and hoping to gain insights, we differentiate two types of traffic to observe - the request per se and the general traffic patterns:

1.) **the request** (using Burp to gather)
Requests contain various information about the sender, such as the used browser, the location, the source IP address and the request itself. After receiving the request, the server answers with a HTTP response, which contains a header and a body part. This is important, as the header part may be exploitable in various ways as Kettle [11] shows - and thus, request and responses need both to be observed. Additionally, certain flags set in the TCP/IP protocol, such as RST, are possible characteristics for a network scan, which may indicate the preparation of an incoming attack.

2.) **traffic patterns** (using Wireshark to gather)
The observations in 1.) presume the existence of malicious code, such as an XSS or SQLi. However, this does not necessarily need to be the case. In a request by itself, a failed login attempt does not seem to be malicious and would not be detected by inspecting the request. However, a failed login attempt occurring every second for multiple hours is indeed suspicious, possibly a brute-force attempt to guessing a users password. A multi-agent system as the SEASALT framework suggests can be helpful to detect these general patterns of timed attacks.

Having two different types of objectives to observe leads to at least two different casebases. Consequently, we need a possibility to communicate between these casebases to detect anomalies. SEASALT offers the opportunity in the form of multiple topic agents. For example, we can deploy one topic agent focusing on the traffic patterns. This agent may find a regular similar request each 0.4 seconds, which may indicate an automated process and could be worth further investigation. Therefore, the traffic agent forwards the information to the coordination agent, who distributes the suspicious requests to other specified topic agents, such as SQLi-, XSS-, and authentication testing topic agents. Each agent contains specific knowledge containers to successfully classify the requests in question. In the following subsections, we describe the structure and content of the knowledge containers.

## 3.1. Knowledge Container: Vocabulary

The vocabulary knowledge container describes the terms used as part of a knowledge-intensive case based reasoning system [5]. Therefore, we exemplary describe a few chosen attributes used for our cases.

**Source/destination IP address:** While communicating in a network, an individual IP address is assigned to each device (see Figure 1). An IPv4 addresses consist out of four triplets, each having values between 0 and 255, for example 147.172.2.69 represents the unique IP address of the hackers device. IPv6 extends this to eight groups of four hexadecimal digits. However, internal networks such as the home office example are usually managed through network address translation methods (NAT routing). For traffic and network detection, the consequence is, that only the IP address of the router (192.168.0.1) is visible to the internet, thus, we may not

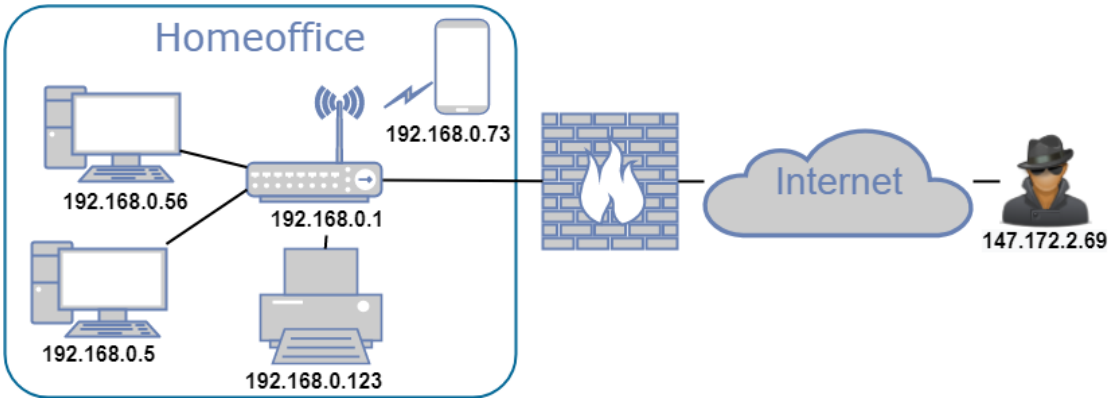


**Figure 1:** Example of a small home office network, connected to the internet through a firewall. On the other side, a malicious user ("hacker") is also connected to the internet and needs to pass the firewall to connect to the internal network.

be able to identify how many attackers are behind the same network[4]. It is important to note that IP addresses may change after 24 hours or after a restart of the router.

**Source/destination port:** While a computer operates under a certain IP address, it may offer different services on different ports. The port is usually directly following after the IP address. For example, 192.168.0.56:443 indicates a secure web server (HTTPS) on port 443. The Internet Assigned Numbers Authority (IANA) defines a global standard list[5] of which services should be operated on which ports and it is heavily encouraged to network administrators to follow these suggestions. This attribute may be used to distinguish between a legitimate request of a web source or an information gathering approach (scanning on any other unexpected ports).

**Protocol:** Similar to the IP address and port, the communication between server and client can be established based on different protocols. For example, a typical visit of a website omits a three-way-handshake to establish a connection, using the TCP protocol. The actual request, asking for a certain document, uses the HTTP protocol. In contrast, UDP is a protocol, which is primarily used to send data. This protocol may be used to circumvent firewalls preventing to scan the target host and thus may indicate an attack.

For further attributes, we remain with a short tabular overview in Table 1.

To summarize the vocabulary: We are basically using 'typical' network traffic data to distinguish between legitimate traffic and suspicious traffic by identifying irregularities in used protocols, port, and other attributes. Using a case-based instead of a rule-based approach here seems to be promising in regards to identifying similar attacks as there are multiple different attributes to consider which increases the complexity of a properly configured rule-based system.

[4]Rest assured, with the compliance of internet service providers, traffic can be pin-pointed to a certain machine and criminals may be identified.

[5]For more information and the complete list of standardized ports, see https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml (last validation 08/18/2021)

| Attribute | Type | Description |
|---|---|---|
| Length | number | Length of the request |
| Timings | symbol | opening a connection every 5 min, 15 sec, 0.4 sec, 10 ms, 5 ms |
| Description | text | additional information added by a maintenance engineer |
| Request/Info | text | content of the request |
| User-Agent | text | used web browser including version number |
| Status | number | server status based on the request (200-OK, 403-Forbidden, ...) |
| Cookie | text | additional information gather about the user |
| Referer | text | URL of the last visited website |
| Time | text | timestamp of the request/response |
| Method | symbol | HTTP verbs which allow to infer an intent of the request |
| Number | number | position in the exported data lists |

**Table 1**
Brief description of attributes used in the case structure.

## 3.2. Knowledge Container: Similarity

The similarity container defines the local similarities of attributes and their contribution to the respective amalgamation function. For text attributes, initial testing showed the best results by using the Levenshtein distance. Numerical attributes performed well by using a polynomial function with factor 5 (see Table 2). Future work will need to revisit on automated learning of optimal weights. We remain with a short tabular overview in Table 2.

In terms of global similarity, which depicts the weight and thus the importance of an attribute, we begin with a distribution based on the experience and estimation of a domain expert. Table 2 provides an overview. In general, we impose a higher similarity for attributes which are more likely to identify an attack. We form the global similarity by using the weighted sum of all local similarities.

## 3.3. Knowledge Container: Case Base

Based on the observation and the need of two different case structures, we consequently define two different kinds of casebases: $CB_{request}$ and $CB_{traffic}$. The former casebase inspects a single request - typically through the HTTP protocol - while the latter observes multiple requests through the network in general. An initiated agent may only have one kind of casebase. As case representation, we use attribute-value-pairs. Regarding automated learning of cases, we do not see a valuable opportunity in automated learning without overfilling the case as of today. It remains the maintenance engineers' decision whether to retain or to deny a new problem case into the case base.

| Attribute | Type | Similarity | Weight |
|---|---|---|---|
| IP address | text | Levenshtein distance | 1 |
| Length | number | Polynomial with 5 | 6 |
| Protocol | text | Levenshtein distance | 1 |
| Description | text | Levenshtein distance | 7 |
| Request/Info | text | Levenshtein distance | 7 |
| User-Agent | text | Levenshtein distance | 1 |
| Status | number | Polynomial with 5 | 4 |
| Cookie | text | Levenshtein distance | 1 |
| Referer | text | Levenshtein distance | 1 |
| Time | text | Polynomial with 5 | 2 |
| Method | symbol | Predefined symbol matrix | 1 |
| Timing | symbol | Predefined symbol matrix | 4 |
| number | number | Polynomial with 5 | 3 |

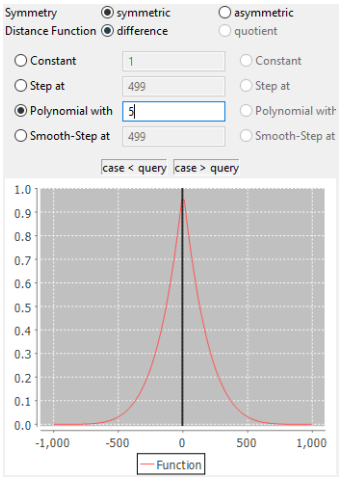


**Table 2**
(left) Local similarities of the attributes and attribute weights for computing the global similarity using the weighted sum. (right) Similarity function polynomial with 5 for attributes of type number. The higher the factor, the earlier the similarity between case and query decreases.

# 4. Development of a prototype

## 4.1. Retrieving data - OWASP Juice Shop, Wireshark, and Burp

Wireshark is the standard tool for network analysis across many different institutions due to its large variety of different observable protocols and continuing development by volunteer contributions.

Per default, seven columns are displayed: No., Time, Source, Destination, Protocol, Length, Info. *No.* iterates over the number of packages received and *Time* is measured in milliseconds since actively tracking (in contrast to timestamps in Burp). By investigating a certain package, we receive further information such as which flags have been set. We use the inbuilt function of exporting the data to CSV format[6]. As initial casebase, we use the retrieved data and treat each package as a case and provide an interface to query the casebase. This allows us to search for 'timed traffic' such as slow brute force attacks or scans which occur, for example, every 15 seconds. One might argue we could just take the attribute *Time* and calculate $\pm$ 15 seconds. Due to wavering latency in the network, we would not be able to identify these timings. However, using CBR and querying the system for similar cases in $\pm$ 15 seconds detects similar traffic.

To inspect one package in more detail, we use Burp. Just as Wireshark, Burp is another standard tool for network analysis and available for free. A request and the corresponding response are compared to each other. Both contain a head- and a body section - separated by an empty line. As one can see, there are a lot of potential attributes for a CBR system to identify similar traffic. In addition to the presentation of the traffic, Burp offers the opportunity to adjust a request and send it again to the server, hoping for an information leak (as an attacker). In most

[6]Unfortunately, the resulting CSV writes all data in one column per row, attributes are separated by commata. We wrote a parser to structure the data.

attacks, only one parameter will be changed to identify the influence of the parameter - and this allows the CBR system to identify this diverging traffic. Nevertheless, to gather training cases, we need a vulnerable website, which can be attacked without legal consequences. This is provided by the OWASP[7] Juice Shop[8], which is a vulnerable web application with a large number of different security issues. Against this web application, we can launch attacks and store them as attack cases in a separate (malicious) casebase.

## 4.2. Implementation in Java using myCBR 3.0

The prototype implementation is a dynamic web project written in Java. By using Java server pages, forwarding via servlets, and creating dynamic content with Java custom tags, any user on the internet may interact with the system (see Figure 2). Since weighting the attributes of a CBR system is an experience-driven process, we enable the users to set their own weights.

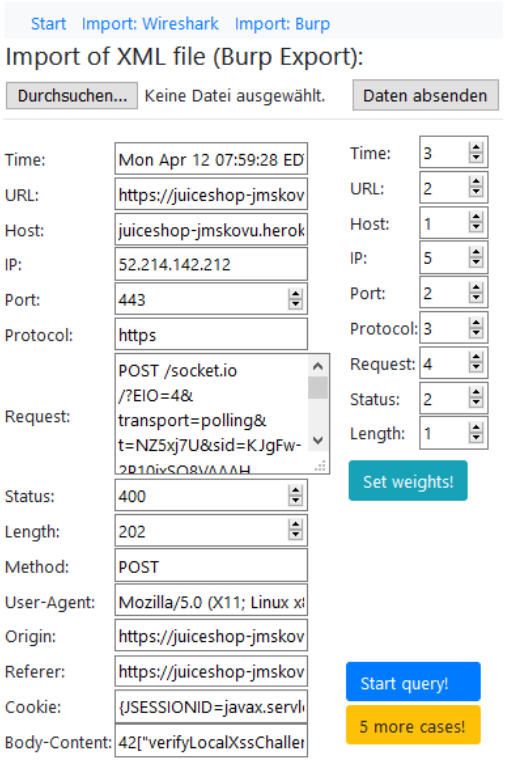


Figure 2: Userinterface to interact with the CBR system. Weights can be set on the right side.

Figure 3 illustrates the general process of the system: Beginning on the index page of the dynamic web project, a user may either upload an XML export file from Burp, or a CSV export file from Wireshark. Either way, an agent will be instantiated. These agents are SEASALT agents as presented in Section 3 and contain their own casebases. These casebases are initiated by adding the content of the imported files to the initially empty casebase. If the user does not choose to update a file, a query can be sent to the communication agent. The user specifies the query by filling a form representing all possible attributes of a case. The communication agent queries topic agents and awaits their response to the users query. Each topic agent presents its case with the highest similarity. If a predefined threshold cannot be reached, the communication agent asks the user for a more detailed query.

Otherwise, the communication agent presents the most similar case to the user. Every agent uses the myCBR[9] 3.0 SDK which is an open-source similarity-based retrieval tool developed by the Competence Centre CBR at DFKI, Germany, and the School of Computing and Technology at the University of West London, UK.

[7]The open web application security project (OWASP) is a community driven project, offering security materials and training for free.

[8]For more information, see https://owasp.org/www-project-juice-shop/ (last validation 08/18/2021).

[9]For more information, see http://mycbr-project.org/ (last validation 08/18/2021)

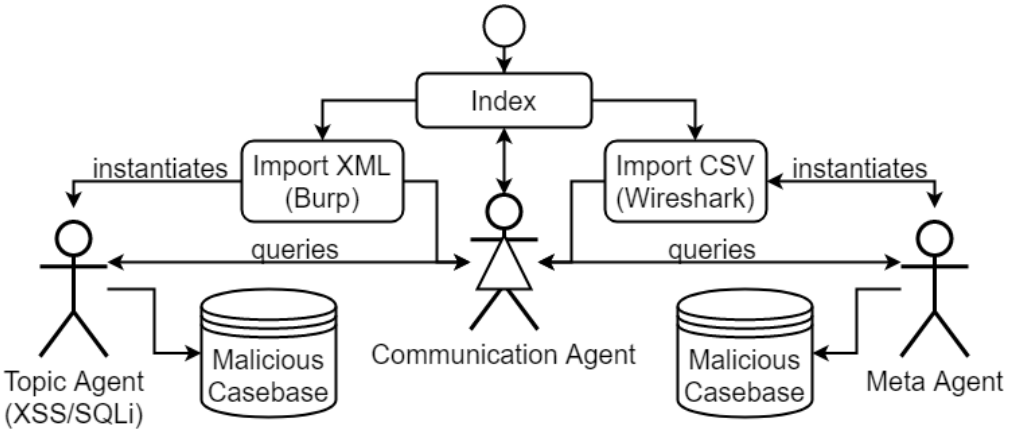


**Figure 3:** Overview of the prototype. The workflow starts at index.jsp. By importing a Wireshark or Burp Export, the corresponding agent will be initialized. Afterwards, the user may send queries to the communication agent.

As an example use-case, we want to investigate the current traffic. To go through each packet manually would be a tedious task due to the flood and volatile data. However, we can begin with investigating the Wireshark import for an overview of the traffic data. This enables the traffic agent to provide a list of the top $n$ loudest[10] clients. In Figure 4, the user retrieves a tabulated and a visual presentation of the traffic, filtered by the IP address of the clients. As one can see, the first client issued almost 60 times more requests than the third client - which makes the first client suspicious.

[10] "loud" measured in total number of requests sent.

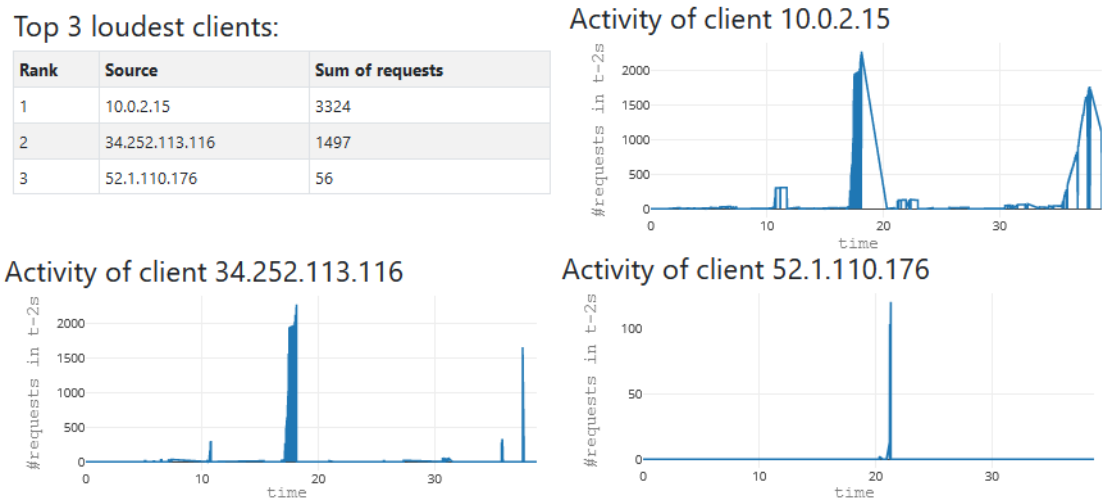


**Figure 4:** Top 3 loudest clients as graphical and tabular overview. The graphs display the accumulated amount of requests during the last two seconds.

This can be communicated to the communication agent and forwarded to the request agent. Now, the request agent can filter for the IP address of the suspicious client and start investigation for possible attacks by a smaller range of clients to test.

## 5. Experiment and Results

We distinguish between a *normal* and a *malicious* casebase. Initially, we fill the *normal* casebase our agents with XML/CSV training imports, which do not contain malicious traffic to establish a model of valid traffic. In a second step, we fill the *malicious* casebases with hand-crafted malicious requests and public lists of known SQLi/XSS attacks ('payloads') to establish a model of malicious traffic.

By using Burp and the OWASP Juice Shop, we are located in a controlled environment without noisy data. In a third step, for training purposes, we import a CSV or a XML file and search for malicious traffic. For each malicious request identified this way by our prototype, we offer the user, e. g., a maintenance engineer to ultimately decide whether to move the case to the *malicious* or to the *normal* casebase.

This way, we can naturally fill our casebase of malicious traffic and further increase the precision of our malicious traffic model. This opens up two opportunities:

- Contrast normal and malicious traffic (i. e. as an explanation)
- Prioritize traffic based on the "top 3 loudest clients" offered by the traffic agent to test against the *malicious* casebase (higher success probability)

Our test is similar to the third step: We upload a XML file with six malicious requests out of a total of 795 requests. These malicious requests are manually created and are not part of the *malicious* casebase. We aim to automatically identify the malicious requests after comparing each of the 795 requests against cases in our *malicious* casebase. Basically, we query the CBR system automatically for each request against the *malicious* casebase which has been trained with malicious cases before. We are in an advantageous situation as a query is usually complete (each attribute has a value assigned), with a few rare exceptions such as an empty cookie. The requests contain different attack vectors such as XSS and SQLi shown in Table 3.

We expect to find similar cases with at least 90 % similarity. Indeed, we can identify an attack containing a similar payload as ID 6 with 90 % similarity:

$$\{\textit{'email':' ' AND INSERT INTO users VALUES(1,2,3,4); --','password':'123'}\}.$$

This attack revolves around guessing the number of columns the queued table contains by adding numbers in the round brackets after VALUES, which is a common testing approach [12]. However, unfortunately, we only found one out of six attacks. Lowering the threshold to at least 85 % similarity sheds light onto the situation: now, we also do find payloads with the ID 1,3,5, which adds up to identifying four out of six attacks. Nevertheless, we also find false-positives in 11/16 cases and one redundant finding.

| ID | Payload | Comment |
|---|---|---|
| 1 | <script>alert('XSS!')</script> | typical XSS alert window |
| 2 | ' or '1' = '1 | basic SQLi check |
| 3 | '-5455%') OR 2598=(SELECT COUNT(*) FROM SYSIBM.SYSTABLES AS T1,SYSIBM.SYSTABLES AS T2,SYSIBM.SYSTABLES AS T3) AND ('%'=' | very specific SQLi targeting IBM databases |
| 4 | */'>alert(1)/* | XSS |
| 5 | <x oncut=alert(1)>A | HTML-Injection paired with XSS |
| 6 | ' AND INSERT INTO Users VALUES(1); – | SQLi to investigate database structure |

**Table 3**
Overview of used XSS/SQLi payloads.

## 6. Conclusion and Future Work

We presented a novel CBR application in the IT security and network domain to detect attacks towards a companies network - regardless of external or internal traffic. As one cannot cover each attack by developing and maintaining a large set of rules, we offered an application to detect similar attacks as soon as possible. We identified possible attributes for a reasonable similarity assessment and gathered the information using the commonly used tools Wireshark and Burp. With an instantiation of the SEASALT framework, we ensured a rational modularization of tasks and guaranteed expandability by simply adding additional topic agents for any new attack vector. The results were two-fold: on the one side, we were able to detect one of six attack vectors by retrieving similar cases, which was the original goal of the prototype. However, on the other side, the precision is too low and does not cover enough cases as initially expected.

It remains for future work to further adjust the local- and global similarity. As we now have a prototype for detecting malicious traffic, we may now focus on the training of the knowledge containers. This should result in a higher precision to find similar cases in our *malicious* casebase and lower the rate of false-positives. Additionally, this work opens up opportunities for explainable case-based reasoning: as soon as the weights have been trained, the user should receive an explanation on how the similarity has been assessed. Especially, since the weights are heavily dependant on the use case.

## References


[1] J. Johnson, Internet usage worldwide - statistics & facts, 2021. URL: https://www.statista.com/topics/1145/internet-usage-worldwide/#dossierSummary, Statista. Website. Last validation: 05/04/2021.

[2] L. LaBerge, C. O'Toole, J. Schneider, K. Smaje, How COVID-19 has pushed companies over the technology tipping point—and transformed business forever, 2020. URL: https://www.mckinsey.com/business-functions/strategy-and-corporate-finance/our-insights/how-covid-19-has-pushed-companies-over-the-technology-tipping-point-and-transformed-business-forever, McKinsey & Company. Website. Last validation: 03/31/2021.

[3] Cloudflare, What is the OSI model?, 2021. URL: https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/, Website. Last validation: 04/06/2021.

[4] K. Bach, Knowledge Acquisition for Case-Based Reasoning Systems, Ph.D. thesis, University of Hildesheim, 2012. URL: http://www.dr.hut-verlag.de/978-3-8439-1357-7.html.

[5] M. M. Richter, R. O. Weber, Case-based Reasoning, in: Handbook of Artificial Intelligence, Springer Heidelberg New York Dordrecht London, 2013. doi:10.1007/978-3-642-40167-1.

[6] S. Kapetanakis, A. Filippoupolitis, G. Loukas, T. Saad Al Murayziq, Profiling cyber attacks using case-based reasoning, in: 19th UK Workshop on Case-Based Reasoning, 2014, pp. 39–48. Conference date: 01-01-2014.

[7] M. L. Han, B. I. Kwak, H. K. Kim, Cbr-based decision support methodology for cybercrime investigation: Focused on the data-driven website defacement analysis, Secur. Commun. Networks 2019 (2019) 1901548:1–1901548:21. URL: https://doi.org/10.1155/2019/1901548. doi:10.1155/2019/1901548.

[8] D. Schwartz, S. Stoecklin, E. Yilmaz, A case-based approach to network intrusion detection, in: Proceedings of the Fifth International Conference on Information Fusion. FUSION 2002. (IEEE Cat.No.02EX5997), volume 2, 2002, pp. 1084–1089 vol.2. doi:10.1109/ICIF.2002.1020933.

[9] A. Micarelli, G. Sansonetti, A case-based approach to anomaly intrusion detection, in: P. Perna (Ed.), Machine Learning and Data Mining in Pattern Recognition, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 434–448.

[10] J. M. Schoenborn, K. Althoff, Prototype application to detect malicious network traffic with case-based reasoning and SEASALT, in: Workshops Proceedings for the Twenty-ninth International Conference on Case-Based Reasoning co-located with the Twenty-ninth International Conference on Case-Based Reasoning (ICCBR 2021), Salamanca, Spain, September 13-16, 2021, CEUR Workshop Proceedings, CEUR-WS.org, 2021.

[11] J. Kettle, Practical HTTP host header attacks: Passwort reset and web-cache poisoning, 2013. URL: https://www.skeletonscribe.net/2013/05/practical-http-host-header-attacks.html, Skeleton Scribe. Website. Last validation: 04/01/2021.

[12] M. Pound, Running an SQL Injection Attack, 2016. Computer Science at the University of Nottingham. Computerphile.