ERICSSON

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

FH Aachen, Campus Jülich

Faculty 09 - “Medizintechnik und Technomathematik”
“Angewandte Mathematik und Informatik”

Seminar Thesis

Description and Evaluation of Intrusion Detection System Mechanisms for REST-based Applications

Author:
Daniel Bales

Matriculation Number:
3234206

Aachen
December 15, 2021

Statutory Declaration

I hereby assure that this thesis with the title

"Description and Evaluation of Intrusion Detection System Mechanisms for REST-based Applications"

has been produced and written on my own and has not been part of any other study or examination performance. No other means, sources or resources have been used, other than the listed ones. Every part of my work that contains quotations is marked accordingly and can be found in the bibliography.
I vow to keep a copy of this thesis for five years and to hand it out on request by the examination office of the faculty "Medizintechnik und Technomathematik".

Location, Date	Daniel Bales

This thesis was supervised by

Prof. Dr. Andreas Terstegge	Volker Kleinfeld

Aachen, December 2021

Abstract

During the last decades, the process of digitalization became omnipresent in our everyday life. Smart devices with a variety of new, stunning features appear in all parts of life. While this development offers numerous advantages and additional possibilities, it also contains multiple risks, threats and challenges as a consequence of the ongoing digitalization. This is in line with nearly all technical evolutions. These risks and threats include hacker attacks and data breaches, whose frequency has significantly increased in recent years.

Currently, the only way to fully prevent these attacks requires to disconnect all the servers and other devices from the Internet. This, however, is rather impossible as it would make the existence of these devices superfluous. Therefore, administrators of systems and networks need to secure their systems against various types of attacks in the best possible way. As it is infeasible to completely prevent all attacks, tools need to be available which raise alarms and inform the administrator when they discover an intrusion in the system or network. In fact, this is the main purpose of intrusion detection systems (IDS).

This thesis introduces the background of intrusion detection systems and discusses components, mechanisms and approaches which can be used to discover attacks in detail. Furthermore, the structure and principles of REST-based applications are explained. A description of commonly used attacks against this type of application, such as denial-of-service attacks, is also provided. Additionally, the applicability of intrusion detection to the thematized attack types is evaluated with regard to the described IDS mechanisms. Finally, the thesis concludes with a summary of the discussed topics and the result that intrusion detection can be applied to REST-based applications only to a certain extent, as there are some limitations.

Contents

1 Introduction

“My message for companies that think they haven’t been attacked is:
‘You’re not looking hard enough’ ”

- James Snook (Office for Cyber Security, UK Cabinet Office) [1]

This quote from the deputy director of business, crime and skills in the Office for Cyber Security of the United Kingdom emphasizes a problem more and more companies have to deal with nowadays: Cybercrime is a great threat in today’s digital world.
Many companies and institutions are targets of hacker attacks and data breaches every day, which can easily be verified by taking a look at the latest facts and figures. In the first half of 2020 alone, 36 billion records were exposed by data breaches all over the world. [2] In general, the number of data breaches increased by 11 % between 2018 and 2019, and by 67 % from 2014 to 2019. [3] As a consequence, the costs of cybercrime for companies have massively risen from $3 trillion in 2015 up to an estimated cost of $10.5 trillion annually worldwide by 2025. [4]

Apart from the enormous increase of cyber attacks, another problem is that it often takes a long time until the victims of a hacker attack or data breach detect the intrusion in their system. According to a study conducted in 2020, companies required 207 days on average to identify such a breach. [5] Furthermore, there are a lot of attacks and intrusions that are not even noticed by the administrators of the affected systems.
The consequence of this development is that administrators of networks and systems have to take two points into consideration: How can the systems be secured most efficiently and how can successful attacks or intrusions, which occurred despite the security precautions, be detected as quickly as possible?

This thesis focuses on the second point. One popular option to discover attacks or intrusions is the so-called intrusion detection system (IDS). The main purpose of these systems is to raise alarms and inform administrators when an attack or intrusion is currently taking place or happened recently in their networks or systems.

The remainder of this thesis is composed as follows. In Chapter 2, the basic theory and background of intrusion detection systems are provided. Chapter 3 displays different mechanisms and components that can be used for attack discovery. Chapter 4 thematizes the principles and architecture of REST-based applications and also describes typical attacks that are commonly used to corrupt this type of application. In Chapter 5, the applicability of intrusion detection to the presented attacks is evaluated with regard to the described intrusion detection system mechanisms. Finally, Chapter 6 provides a conclusion of how applicable intrusion detection is, in the context of REST-based applications.

2 IDS Background

2.1 Definition

In their book, Rebecca Bace and Peter Mell define intrusion detection systems as "software or hardware systems that automate the process of monitoring the events occurring in a computer system or network, analyzing them for signs of security problems." [6, p. 5] Such a security problem is called intrusion and can be defined as a violation of the system's security policies. In most cases, an intrusion is performed by an attacker from outside the network or computer system. Typically, attackers aim to steal sensitive data, implant malicious software or make the whole system unavailable. [7]

2.2 History

The idea of intrusion detection goes back to the early days of data processing and computer security. J. P. Anderson published the first seminal work in intrusion detection in 1980, where a concept for the automation of the review of security audit trails was proposed. In 1986 the computer scientist Dorothy Denning presented a model for intrusion detection, which was the basis for the first prototype of an intrusion detection system called *Intrusion Detection Export System (IDES)*. Many similar prototype systems were developed in the following years, for example, *Haystack*, *Network Anomaly Detection and Intrusion Reporter (NADIR)*, *Information Security Officer's Assistant (ISOA)* and *ComputerWatch*. [7]
Nowadays, there are several systems available, free and open-source software as well as commercial and proprietary software. Some of the most common IDS tools are listed below.

- Snort
- SolarWinds Security Event Manager
- Open Source HIDS Security (OSSEC)
- Samhain
- Suricata
- McAfee Network Security Platform

2.3 Types

Intrusion detection systems can be categorized into different types, which mainly differ in the kind of data that is monitored and analyzed by the IDS to detect intrusions. The different types are explained in the following sections.

2.3.1 Host-based

The oldest type of intrusion detection system is the host-based IDS (HIDS). This type of IDS runs directly on the host which is observed and analyzes events that occur on this host. The HIDS monitors data such as application and system logs, file system changes, system calls and running processes. As the host-based IDS is running directly on the host, it is able to create a detailed report about the type and progress of an occurring attack, including, for example, files that were changed or commands that were executed by the intruder. A disadvantage of this IDS type is that the attacker can try to modify or corrupt the HIDS itself, preventing it from raising an alarm and detecting the intrusion. Common examples of host-based intrusion detection systems are *Samhain* and *OSSEC*. [8][9]

2.3.2 Network-based

Network-based intrusion detection systems (NIDS) monitor and analyze network traffic in the whole network or a certain network segment. In most cases, a NIDS is running on a dedicated device placed in between the internet and the intranet of a company or institution. Therefore, it is able to provide nearly instant detection of network intrusions. However, this approach has also disadvantages. It is often difficult to detect intrusions if the monitored packets are encrypted. Furthermore, NIDS devices may limit the network's throughput and latency parameters because all traffic has to pass the NIDS. *Snort* and *Suricata* are common examples of a network-based intrusion detection system. [8][10]

2.3.3 Hybrid Systems

As both HIDS and NIDS monitor totally different kinds of data and both have advantages and disadvantages in different areas, a logical approach is to combine both IDS types to achieve a more effective way of intrusion detection. Therefore, systems were developed that offer functionality which would traditionally be attributed to a host-based IDS as well as features that would be attributed to a network-based IDS. Such systems are often implemented with a centralized logging and event management engine. This engine processes data from multiple host- and network-based IDS located somewhere in the network, serving as sensors for the central component. A commonly used hybrid IDS is the *SolarWinds Security Event Manager*. [8][11]

2.4 Response Options

When an intrusion detection system has successfully detected an intrusion, it is necessary that a response follows to avoid further damage to the system or network.

2.4.1 Passive Response

As the IDS itself is not capable of taking appropriate actions against the attack, it just provides the detection results to the administrators so they can consider the actions to take. This kind of response is called passive response and can be divided into two categories, alarms and reports.

- **Alarm:** An alarm is an instant notification, which is generated when an intrusion was just detected. The alarm can be submitted to the user in various ways, for example, as a message in an IDS control panel, as email or as an alarm on a dedicated pager.
- **Report:** A report is a collection of detection results, whose creation is not immediately triggered by a detected intrusion, but invoked on a periodic basis. A report is typically available in an IDS control panel or is sent via email to the administrator. In most cases, it contains a list of the security-related events that have occurred since the previous report creation.

2.4.2 Active Response

Active response means that the IDS itself is capable of taking measures against an ongoing attack, such as blocking all traffic from the intruder's IP address. Classical IDS do not have the capabilities for an active response. A system that contains this functionality is called intrusion prevention system (IPS), but IPS techniques are not further discussed in this thesis as it focuses on the detection process. [7]

As the basic theory of intrusion detection systems has been provided above, the next chapter explains methods and mechanisms that are commonly used for the intrusion detection process.

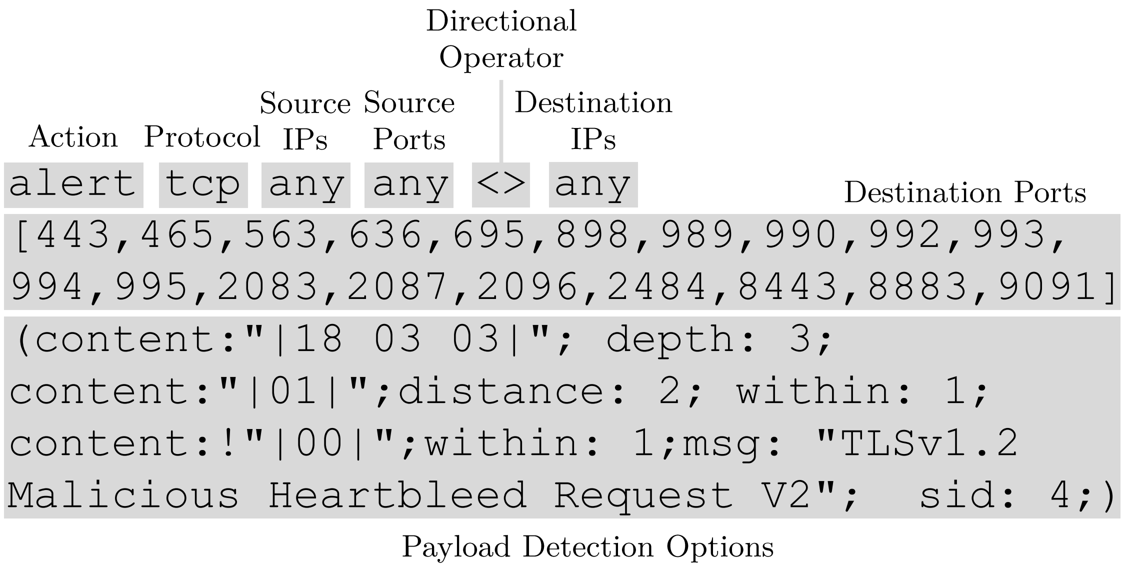
3 IDS Mechanisms

3.1 Detection Methods

In general, there are two different classes of detection methods that are used in IDS tools: signature-based detection and anomaly-based detection. In fact, many IDS tools use both approaches to improve their detection precision. The two different methods are discussed in the following sections.

3.1.1 Signature-based Detection

In the context of intrusion detection systems, the term "signature" can be defined as a pattern that matches an already known attack or vulnerability. An IDS that uses signature-based detection has access to a database of known attack signatures, that is continuously expanded with patterns of recently detected attacks. In order to detect intrusions, the IDS compares the currently processed data, for example, a log entry or a TCP packet, to the signatures stored in the database. If there is a match, the IDS uses its configured response option to notify the administrator about the detected intrusion. In literature, this approach is also referred to as "misuse detection". [8][10]
Figure 3.1 shows an example of an attack signature for the IDS *Snort*. The signature was published by the FBI in 2014 to detect intrusions that exploit the "Heartbleed" vulnerability in the OpenSSL library.



```
alert tcp any any <> any
[443,465,563,636,695,898,989,990,992,993,
994,995,2083,2087,2096,2484,8443,8883,9091]
(content:"|18 03 03|"; depth: 3;
content:"|01|";distance: 2; within: 1;
content:!"|00|";within: 1;msg: "TLSv1.2
Malicious Heartbleed Request V2";  sid: 4;)
```

Figure 3.1: Snort Signature for Heartbleed Attack [12]

Without discussing the signature and the characteristics of the vulnerability in detail, the most important points are the following. As *Snort* is an NIDS, its signatures contain patterns that match a certain network packet. There are various options available for the signature definition, such as the transport protocol (TCP/UDP), the source and destination IP addresses and ports, the traffic direction and a pattern for the packet's payload. If *Snort* detects a packet that matches the signature, it performs the specified action. In this case, it invokes an alert.

Other IDS tools may have a different syntax for defining signatures. Obviously, the signature definition also differs if the analyzed data is not a network packet but, for example, a log entry. Nevertheless, the example signature is suitable for describing the principles of this approach.

Signature-based detection can detect already known attacks very efficiently, but unknown attacks, such as zero-day exploits or even variations of existing attacks, cannot be detected because the IDS has no signature that matches them. That's why the signature-based detection method produces many so-called "false negatives", which means that the IDS does not raise an alarm, although there was an intrusion in the system or network. [8]

The anomaly-based detection method is an approach that tries to solve this problem.

3.1.2 Anomaly-based Detection

An anomaly can be defined as a divergence of the current behavior to the normal behavior. IDS tools with anomaly-based detection search for anomalies in their input data to detect intrusions. In this approach, the IDS needs knowledge of the normal behavior of a network or system to identify discrepancies, but prior knowledge of known threats is not needed. Each deviation from the normal behavior is classified as an intrusion. To accomplish this, the IDS creates profiles of the normal behavior during a dedicated training period, which are then used for anomaly detection. These profiles can either be static or dynamic. Static profiles are never changed after their creation, whereas dynamic profiles are adapted continuously. The main disadvantage of a static profile is that the normal behavior of a system or network may change over time and therefore the profile becomes inexact. Dynamic profiles avoid this problem, but they are vulnerable to manipulation by attackers. If an attacker performs some malevolent activity sporadically and then slowly increases the amount, the IDS could classify this behavior as normal, integrate it in the profile and fail to detect the attacker's real intrusion later.

Anomaly-based detection is capable of detecting previously unknown threats, such as zero-day exploits, but there are also problems with this approach. As each deviation from normal behavior is considered as an intrusion, irregularly happening valid activities also cause an alarm. Therefore the anomaly-based detection method produces many so-called "false positives", which means that the IDS raises an alarm, although the observed activity was not an intrusion. [8][10]

3.2 Components

In addition to the IDS components and mechanisms, such as analysis of system or application logs and single TCP or UDP packets, that have already been mentioned in previous sections, there are some more mechanisms that are very common. These mechanisms are described in the following sections.

3.2.1 File Integrity Check

File integrity checking is an intrusion detection system mechanism that is typically used in HIDS tools. Its main purpose is to verify that certain files have not been compromised and to detect unauthorized changes to the file system of a host. Nowadays, this mechanism is essential for intrusion detection systems, because over time attackers have developed different methods to cover their tracks when attacking a system. Without file integrity checking, an intrusion that has just happened may not be detected and files that have been changed by the attacker cannot be identified by the administrators of a system.

In general, there are two necessary steps for file integrity checking. Initially, a database with information about every file that should be observed has to be created. The database entries contain a hash of the file content and file attributes such as the inode number, file size, file owner and group, file permissions and the modification dates. Once the database is created, the IDS periodically compares the file content hash and the attributes of the actual files with the information stored in the database. If there is a discrepancy, the IDS treats that as an intrusion. Whenever an authorized user changes some of the observed files, the database entries for these files must be recreated because the authorized changes are classified as an intrusion otherwise. Additionally, the database itself can be cryptographically signed to prevent attackers from compromising it to manipulate the IDS. [13]

File integrity checking cannot be categorized clearly as a signature- or anomaly-based mechanism because there is neither a check for matching predefined attack patterns nor a comparison of current system behavior to normal behavior.

3.2.2 System Call Trace Analysis

A system call can be defined as a request for a service of the operating system's kernel that is performed by a running process. In general, system calls are the interface between applications and the system kernel. Using system calls for intrusion detection is a mechanism of host-based IDS tools. The idea is to analyze system call traces to find certain call sequences that could indicate potential intrusions. These traces are collected using tracing tools such as *strace* or *SystemTap*. A typical system call sequence on Linux systems could be the following:

> ...open, read, mmap, mmap, open, getrlimit, mmap, close...

The definition of the system calls shown above can be looked up in the Linux manual.
System call trace analysis can either be signature- or anomaly-based. In the signature-based approach, the IDS uses datasets with known system call sequences for certain attacks and compares the current sequences with the dataset entries. Commonly used datasets are provided by the UNM and the DARPA. In the anomaly-based approach, the IDS searches for uncommon system call traces by comparing the actual system calls of running processes to profiles that contain the normal system call sequences for these processes. This database is created by the IDS itself during an initial training phase.
System call trace analysis is a common IDS mechanism because system calls directly interact with the kernel of the operating system and are therefore very hard to manipulate. [10]

3.2.3 Stateful Protocol Analysis

Stateful protocol analysis is a signature-based mechanism in which the IDS analyzes the behavior in a certain state of a specific transport or application protocol. To infer the current protocol state, the IDS observes the traffic and message flow between the two communicating applications or devices and matches request with response messages. To detect potential intrusions, the IDS has predefined datasets with normal and suspicious practices and behavior for each protocol in each state. If suspicious behavior is detected, it is classified as an intrusion. Examples of deviations from the expected behavior are commands that are untypical for the current protocol state or command arguments that are far shorter or longer than expected. Figure 3.2 shows an example of expected and suspicious behavior in different states of the stateful protocol FTP.

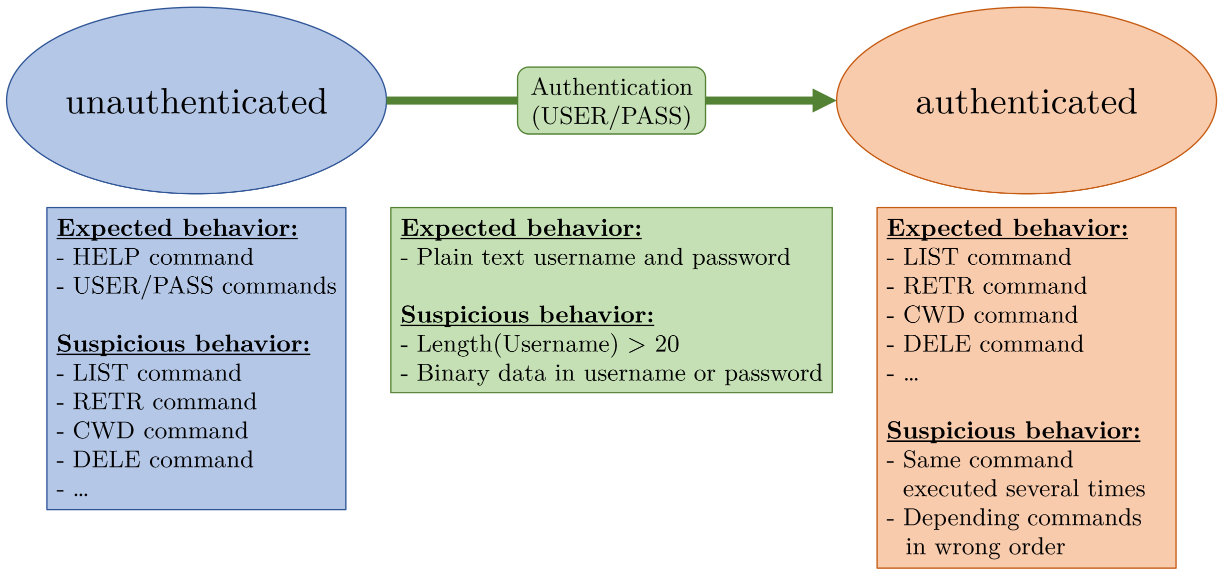


Figure 3.2: Stateful Protocol Analysis in FTP

The initial state of the FTP protocol is unauthenticated, which means that no credentials have been passed yet. In this state, users are expected to perform actions such as executing the help command or providing a username and password to authenticate themselves. The execution of other commands, for example, to retrieve files or list directory contents, is suspicious. When the user provides credentials to get into the authenticated state, the username should typically not exceed a length of 20 characters and neither username nor password should contain binary data. Each deviation from this behavior is considered suspicious. When the user has reached the authenticated state, all commands that were unusual in the unauthenticated state, are now valid. In this state, it is considered peculiar if the same command is executed several times consecutively or if commands that depend on each other are executed in the wrong order. [8]

After certain intrusion detection system mechanisms have been discussed above, the next chapter introduces REST-based applications as a specific context to which intrusion detection can be applied.

4 REST-based Applications

4.1 Theory

The term REST is an acronym for Representational State Transfer and defines a software architecture that follows certain principles and guidelines. Roy T. Fielding introduced the term REST in his famous dissertation ([14]) and defined all constraints and principles that REST-based applications must fulfill. However, as the REST architecture is universal, the dissertation does not specify any implementation technology.
A core concept of REST is the definition of a resource. According to Roy T. Fielding, a resource is defined as follows: "Any information that can be named can be a resource: a document or image, a temporal service [...], a collection of other resources, [...] and so on." [14, p. 88] In general, resources can be understood as the objects on which operations are performed. [15]
The following section lists the five principles of REST-based applications.

4.1.1 REST Principles

1. **Client-Server:** The server provides a service, the client requests it.
2. **Stateless:** Every message to the server contains all relevant information to process the request. No information about a state is stored on the server.
3. **Cacheable:** Caching shall be applied to all cacheable resources, which must be marked as such.
4. **Uniform Interface:** Each resource has a uniform unique address, can have multiple representations and can be accessed using standard methods.

5. **Layered System:** Multiple layers shall be used so that the whole application can be split to several servers and single components can easily be replaced or extended.

[14]
In practice and for this thesis, the only relevant REST implementation technology is the web technology using the stateless client-server protocols HTTP and HTTPS. The next section shortly describes how the concept of REST is realized using the web technology.

4.1.2 RESTful HTTP

As described above, the REST architecture requires a unique address for each resource. The web already contains URIs as a concept for uniform and unique identifiers, so URIs are also used to address REST resources. REST also requires a defined set of standard methods to access the resources. HTTP has the so-called “verbs”, which are used for that purpose. Accessing a resource with a certain HTTP verb causes a specific operation on the resource. Table 4.1 shows the mapping of the four main HTTP verbs to the respective operations. [15]

HTTP Verb	**Operation**
GET	Retrieve a resource
POST	Create a new resource
PUT	Update a whole resource
DELETE	Delete a resource

Table 4.1: Mapping of HTTP verbs to operations

The following section introduces some typical attacks that are commonly used against REST-based applications, only focussing on the described REST implementation that uses the web technology. From this section onwards, the terms REST-based application and REST API are used interchangeably.

4.2 Typical Attacks

According to an article about the top security threats for REST APIs ([16]), the attack types “Denial of Service” and “Injection” have been chosen as the examples to be displayed in this thesis. Both are discussed in the following sections.

4.2.1 Denial of Service

When performing a denial-of-service (DoS) attack, the attacker floods a targeted server with an enormous number of requests with the aim of bringing the system to a collapse. There are different types of DoS attacks that act on different layers of the OSI model:

- **Ping Flood:** Ping flooding is a type of DoS attack on layer three of the OSI model where the attacker sends dozens of echo requests (“ping”) to the targeted server using the protocol ICMP. The server then wastes all of its resources while answering the requests and the system or network may be hugely limited in its accessibility or even collapse. [17]
- **SYN Flood:** This type of DoS attack operates on layer four of the OSI model and misuses the three-way handshake of TCP. The attacker sends a huge number of requests for a new TCP connection (SYN packet) with spoofed source IP addresses to the targeted server. The server then tries to send answer packets (SYN-ACK packet) to the spoofed IP addresses, which are unreachable in most cases. As a consequence, half-open TCP connections occupy the whole connection pool of the targeted server, and new legitimate connection attempts are discarded. [17][18][19]
- **HTTP Flood:** An HTTP flooding attack acts on layer seven of the OSI model. When performing this type of attack, the attacker sends a large amount of valid HTTP GET or POST requests to the targeted system. These requests deliberately target pages or services that are very resource-consuming to overload the server and prevent legitimate users from accessing it. [18]

Nowadays, a common variant of the classical DoS attack is the distributed denial-of-service (DDoS) attack. Instead of sending all requests from a single computer, the attacker uses multiple machines, often about 100 - 1000, to perform the attack. This is more effective than the classical approach because multiple machines can generate more traffic and it is more difficult to take adequate steps against it. [17][20]
This problem is further discussed in Chapter 5.

4.2.2 Injection

The term "Injection" refers to a class of several attacks that have a similar functioning. IBM defines injection attacks as follows: "Injection attacks allow an attacker to inject code into a program or query or inject malware onto a computer to execute remote commands." [21]
All REST APIs that do not properly validate and sanitize input data are vulnerable to injection attacks. According to a list of the top ten security risks for web applications, published by the Open Web Application Security Project (OWASP) every few years, injection attacks have been the biggest threat to web applications in 2013 and 2017 and are currently on third place in the 2021 top ten list.
There are various types of injection attacks. Two of the most common injection types using HTTP are the following:

- **SQL Injection:** REST APIs often use SQL databases as data store. When an API route is called, the API performs a specific SQL query depending on the functionality the route provides. In most cases, some of these SQL queries include input data from the user. An attacker could try to inject a whole SQL statement into the input to perform malicious activities. If the API does not properly validate the input and blindly integrates it into the query, the attacker may be able to extract sensitive data from the database, modify information or drop database tables to damage the application. [22][23]

Figure 4.1 shows an example of an SQL injection where a semicolon is used to terminate the preceding command and a new SQL statement, which drops a certain database table, is injected.

API route definition:
```
GET http://www.example.com/api/products/:productId
```

Resulting SQL query:
```
SELECT * FROM products WHERE product_id = $productId;
```

Expected usage:
```
GET http://www.example.com/api/products/39825
```

Resulting SQL query:
```
SELECT * FROM products WHERE product_id = 39825;
```

SQL injection:
```
GET http://www.example.com/api/products/123;DROP%20TABLE%20products
```

Resulting SQL query:
```
SELECT * FROM products WHERE product_id = 123;DROP TABLE products;
```

Figure 4.1: Example of an SQL Injection

- **OS Command Injection:** If the API internally uses calls to routines of the operating system where user input is included, an attacker can inject malicious code into these calls. This is referred to as "OS command injection". Similar to the SQL injection example, the attacker could terminate the preceding command with a semicolon (Linux) or an ampersand (Windows) and append a self-chosen command. Depending on the privileges of the user account which is running the API process on the server, the attacker could trigger the execution of any command on the server's operating system to perform malevolent activities or could even compromise the whole server by planting malicious software. [24]

After REST-based applications have been introduced above, the next chapter evaluates the applicability of intrusion detection to the described attacks, taking the IDS mechanisms discussed in Chapter 3 into account.

5 Evaluation

The first attack type discussed in this chapter is the denial-of-service attack (Section 4.2.1). As DoS is a network-based attack, an IDS which monitors the network traffic is needed in order to detect it. A first idea is to use an attack signature that matches single network packets, as mentioned in Section 3.1.1. However, as the single packets of a DoS attack are generally not malformed and do not have certain characteristics that allow to distinguish them from packets of valid traffic, this approach is impractical.

A more suitable approach is an anomaly-based detection method, as described in Section 3.1.2. In the simplest scenario of a DoS attack, attackers use only one machine and their own IP address to send a large number of requests. An IDS with anomaly-based detection and a profile of the normal network behavior would classify it abnormal if a single IP address suddenly sent thousands of requests within a short time period. Therefore it is very easy to detect the attack and identify the attacker's IP address in this case.

A more complex scenario is a distributed denial-of-service attack, where attackers use multiple machines with many different IP addresses to perform the attack. In this case, the IDS would also detect an anomaly in the network because of the unexpectedly high amount of requests. In this scenario, the main problem is that a high number of requests does not necessarily indicate a running DDoS attack. A sudden unexpected increase of valid traffic, a so-called “flash event”, is also possible. [25]

For the IDS it is very hard to differentiate between valid and malicious traffic. If the attacker performs an HTTP flooding attack, which operates on layer seven of the OSI model, it is even impossible to determine on TCP level whether the traffic is malicious or valid. Nevertheless, an IDS which is capable of interpreting OSI layer seven can take HTTP-specific criteria, such as status codes or header fields, into account. Additionally, an IDS may be able to distinguish between a flash event and a DDoS attack based on certain general criteria, such as the location of the IP addresses. If the website of a local government in Germany suddenly receives thousands of requests per second from Russian IP addresses, it is very likely that this is a DDoS attack and not a flash event. However, this scenario cannot be generalized because attackers can spoof the source IP addresses or perform the attack using hacked computers from several regions with different IP addresses. In summary, it is infeasible to find a single solution that effectively detects all types of DoS attacks with an accuracy of 100 %, but there are approaches which detect them at least to a certain extent. Moreover, there are some DoS detection approaches that work for applications deployed in specific environments, for example, cloud ([26]) or software-defined networks ([27]), or for specific DoS attack types, such as TCP-based flooding attacks ([28]). Nevertheless, none of these approaches can generally be applied to all possible deployment scenarios of REST-based applications in practice. Furthermore, attackers often adapt their attack techniques whenever a new detection approach becomes public. [25]

Another type of attack which is commonly used against REST-based applications is the injection attack (Section 4.2.2). In the context of REST APIs, the most common type of injection attack is SQL injection. One approach to detect this attack is using signatures that match the injected SQL keywords or statements in an URL or in the payload of a TCP packet, similar to the example described in Section 3.1.1. This approach can detect many SQL injection attacks, but as the SQL language provides dozens of ways to rewrite the same statement, signatures are not capable of detecting all possible statement variations. Attackers can try to slightly modify their injection statement to evade the detection by the IDS, so this approach does not guarantee a detection accuracy of 100 %. Another way of detecting SQL injections is to use an anomaly-based detection method that processes the access logs of the web server which hosts the REST API. If the IDS has a profile of normal calls to certain API routes, it will very likely detect an anomaly in the route parameters, when an attacker tries to inject an SQL statement. However, as in most cases the access logs only contain the called URL and not the body parameters, this approach can only detect injection statements located in the URL. [29]

The second type of injection attack presented in Section 4.2.2 is OS command injection. The two approaches to detect SQL injections can also be applied to OS command injection, but as the possible commands which can be injected depend on the specific operating system and installed packages of the web server, it is more difficult for the IDS to differentiate between valid and malicious input without deep knowledge of the input expected by the respective API route.
An approach to detect possible damage caused by the OS command injection is to use file integrity checking (Section 3.2.1). If all relevant files on the web server are observed, the IDS will detect when an attacker changes some of the files or replaces them with malicious ones. For this approach, it is necessary that the administrator carefully chooses the files to be observed by the IDS. On the one hand, all system critical files and files that normally never change should be observed. On Linux systems, these are, for example, the command binaries (/bin) or the user definition file (/etc/passwd). On the other hand, it does not make sense to observe files that frequently change during the regular execution of the operating system as it would lead to a huge amount of false alarms.

In the context of hardening the application itself, some of the discussed attacks can also be prevented by considering certain points for the API design. If all user input is carefully validated and sanitized, it becomes nearly impossible for attackers to perform injection attacks. Furthermore, there is also an approach to reduce the risk for some DoS attacks. By limiting the number of allowed requests for a certain resource or IP address to a specific amount per second, the probability of DoS attacks can be decreased.

This chapter has evaluated the use and relevance of intrusion detection for the attacks described. The following chapter provides a conclusion about the topics discussed in this thesis.

6 Conclusion

In summary, this thesis has provided an introduction to the area of intrusion detection systems by presenting the necessary background (Chapter 2) and explaining several methods, mechanisms and approaches commonly used in the intrusion detection process, such as anomaly- and signature-based detection, file integrity checking and stateful protocol analysis (Chapter 3). With the aim of introducing a context to which intrusion detection can be applied, the theory of REST-based applications and RESTful HTTP have been described as well as the attack types Denial of Service and Injection, which are commonly used against REST APIs (Chapter 4). Finally, the applicability of intrusion detection to the described attacks has been evaluated with regard to the intrusion detection system mechanisms explained beforehand (Chapter 5).

Examining the question to what extent intrusion detection can be applied to REST-based applications, it becomes clear that there is not a single IDS type or detection approach which can detect all the thematized attacks. As each attack type has individual characteristics, some attack types can be detected more effectively by a host-based IDS while others require a network-based IDS for effective detection. Likewise, the question of whether a signature-based or anomaly-based detection approach is more suitable depends on the concrete attack type. However, as described in the evaluation chapter, intrusion detection can be applied to REST-based applications only to a certain extent. There are approaches which can detect some of the described attack types under certain conditions, but a detection accuracy of 100 % cannot be achieved. Accordingly, intrusion detection systems alone cannot guarantee the complete protection of a REST API. As already mentioned in Chapter 5, many intrusions can be prevented completely by paying attention to certain aspects for the design of the application itself, for example, carefully validating and sanitizing all user input before

using it. Furthermore, additional security components in the network, such as firewalls, are required to ensure higher protection against different kinds of intrusions.
In general, having multiple IDS devices with several detection techniques in a network or system is useful to cover a broad range of attacks. Additionally, a centralized logging and event management engine, as described in Section 2.3.3, is helpful to reduce the number of applications from which an administrator receives warnings and alerts to just one.
Another important point for effective protection against intrusions is that a successful attack detection is always followed by adequate steps taken against the detected attack. Computers can automatically perform many actions to defeat an attack such as blocking the IP address of an identified intruder. Humans, in comparison, can only manually prevent these attacks which takes much more time. Therefore, in practice, this functionality is often integrated into IDS devices. This has been mentioned briefly in Section 2.4.2.
All these things considered and applied, it is possible to achieve good protection of REST-based applications against intrusions. Nevertheless, administrators must be aware that it is in fact impossible to be completely safe from cybercrime, as attackers are continuously adapting, improving and developing attacks to perform even more malevolent activities.

In future, this thesis can serve as a basis for any further work related to intrusion detection systems, especially in the context of REST-based applications. In particular, it is considered as preparatory work for a bachelor thesis within Ericsson in the area of intrusion detection systems and cloud-native applications.

A Abbreviations

API	Application Programming Interface
DARPA	Defense Advanced Research Projects Agency
DoS	Denial of Service
DDoS	Distributed Denial of Service
FBI	Federal Bureau of Investigation
FTP	File Transfer Protocol
HIDS	Host-based Intrusion Detection System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
NIDS	Network-based Intrusion Detection System
OSI	Open Systems Interconnection
OWASP	Open Web Application Security Project
REST	Representational State Transfer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UNM	University of New Mexico
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

B Bibliography

[1] Emily Spaven. *Q&A: James Snook talks the UK government's cybersecurity strategy.* Apr. 12, 2016. URL: https://www.uktech.news/news/qa-james-snook-talks-the-uk-governments-cybersecurity-strategy-20160418 (visited on 10/11/2021).

[2] *2020 Q3 Report.* Tech. rep. Risk Based Security, 2020. URL: https://pages.riskbasedsecurity.com/hubfs/Reports/2020/2020%20Q3%20Data%20Breach%20QuickView%20Report.pdf.

[3] Kelly Bissell, Ryan M. Lasalle, and Paolo Dal Cin. *Ninth Annual Cost of Cybercrime Study.* Mar. 6, 2019. URL: https://www.accenture.com/us-en/insights/security/cost-cybercrime-study (visited on 10/11/2021).

[4] Embroker Team. *2021 Must-Know Cyber Attack Statistics and Trends.* Aug. 11, 2021. URL: https://www.embroker.com/blog/cyber-attack-statistics/ (visited on 10/11/2021).

[5] *Cost of a Data Breach Report.* Tech. rep. IBM Security, Aug. 2020.

[6] Rebecca Gurley Bace and Peter Mell. *Intrusion Detection Systems.* 2001. URL: http://purl.access.gpo.gov/GPO/LPS72073.

[7] Rebecca Gurley Bace. "Intrusion Detection and Intrusion Prevention Devices". In: *Computer Security Handbook.* Ed. by Seymour Bosworth, Michel E. Kabay, and Eric Whyne. 6th ed. Vol. 1. John Wiley & Sons, Inc., Jan. 2012. Chap. 27, pp. 27.1–27.18. DOI: 10.1002/9781118851678.ch27.

[8] Karen Scarfone and Peter Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS).* Tech. rep. 2007. DOI: 10.6028/nist.sp.800-94.

[9] Raj Kishore and Anamika Chauhan. "Intrusion Detection System a Need". In: *2020 IEEE International Conference for Innovation in Technology (INOCON).* 2020. DOI: 10.1109/INOCON50539.2020.9298398.

[10] Ming Liu et al. "Host-Based Intrusion Detection System with System Calls: Review and Future Trends". In: *ACM Comput. Surv.* 51.5 (Nov. 2018). ISSN: 0360-0300. DOI: 10.1145/3214304.

[11] Bharanidharan Shanmugam and Norbik Bashah Idris. "Hybrid Intrusion Detection Systems (HIDS) using Fuzzy Logic". In: *Intrusion Detection Systems.* Ed. by Pawel Skrobanek. InTech, Mar. 2011. Chap. 8, pp. 135–154. DOI: 10.5772/14130.

[12] FBI Cyber Division. *Private Industry Notification: Snort Signatures for Mitigation Against Open Secure Socket Layer Heartbeat Extension Vulnerability.* Apr. 10, 2014. URL: https://us-cert.cisa.gov/sites/default/files/documents/FBI%20Private%20Industry%20Notice-140410-001.pdf (visited on 10/26/2021).

[13] Del Armstrong. *An Introduction To File Integrity Checking On Unix Systems*. 2003. URL: https://www.giac.org/paper/gcux/188/introduction-file-integrity-checking-unix-systems/104739.

[14] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". PhD dissertation. University of California, 2000.

[15] Stefan Tilkov. "A brief introduction to REST". In: *InfoQ* (Dec. 10, 2007). URL: https://www.infoq.com/articles/rest-introduction/.

[16] Guy Levin. *TOP 7 REST API Security Threats*. Jan. 9, 2019. URL: https://blog.restcase.com/top-7-rest-api-security-threats/ (visited on 11/15/2021).

[17] Bundesamt für Sicherheit in der Informationstechnik. *DoS- und DDoS-Attacken*. URL: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Cyber-Sicherheitslage/Methoden-der-Cyber-Kriminalitaet/DoS-Denial-of-Service/dos-denial-of-service_node.html (visited on 11/16/2021).

[18] Rizgar R. Zebari, Subhi R. M. Zeebaree, and Karwan Jacksi. "Impact Analysis of HTTP and SYN Flood DDoS Attacks on Apache 2 and IIS 10.0 Web Servers". In: *2018 International Conference on Advanced Science and Engineering (ICOASE)*. 2018. DOI: 10.1109/ICOASE.2018.8548783.

[19] Christoph L. Schuba et al. "Analysis of a denial of service attack on TCP". In: *1997 IEEE Symposium on Security and Privacy*. 1997. DOI: 10.1109/SECPRI.1997.601338.

[20] Felix Lau et al. "Distributed denial of service attacks". In: *SMC 2000 IEEE International Conference on Systems, Man and Cybernetics*. 2000. DOI: 10.1109/ICSMC.2000.886455.

[21] IBM Corporation. *IBM Security Network Intrusion Prevention System 4.6.2. Injection attacks*. 2014. URL: https://www.ibm.com/docs/en/snips/4.6.2?topic=categories-injection-attacks (visited on 11/16/2021).

[22] Ian Muscat. *What Are Injection Attacks*. Apr. 18, 2019. URL: https://www.acunetix.com/blog/articles/injection-attacks/ (visited on 11/17/2021).

[23] Admir Dizdar. *SQL Injection Attack: Real Life Attacks and Code Examples*. Feb. 22, 2021. URL: https://www.neuralegion.com/blog/sql-injection-attack/ (visited on 11/17/2021).

[24] Tomasz Andrzej Nidecki. *What Is OS Command Injection*. July 1, 2019. URL: https://www.acunetix.com/blog/web-security-zone/os-command-injection/ (visited on 11/17/2021).

[25] Glenn Carl et al. "Denial-of-Service Attack-Detection Techniques". In: *IEEE Internet Computing* 10.1 (2006). Ed. by Siobhán Clarke, pp. 82–89. DOI: 10.1109/mic.2006.5.

[26] S. Velliangiri and Jayapaul Premalatha. "Intrusion detection of distributed denial of service attack in cloud". In: *Cluster Computing* 22 (Sept. 2019). DOI: 10.1007/s10586-017-1149-0.

[27] Lohit Barki et al. “Detection of Distributed Denial of Service Attacks in Software Defined Networks”. In: *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2016, pp. 2576–2581. DOI: 10.1109/ICACCI.2016.7732445.

[28] Mohammed Alenezi and Martin J. Reed. “Denial of Service Detection Through TCP Congestion Window Analysis”. In: *World Congress on Internet Security (WorldCIS-2013)*. IEEE, Dec. 2013, pp. 145–150. DOI: 10.1109/worldcis.2013.6751036.

[29] Brad Warneck. *Defeating SQL Injection IDS Evasion*. 2007. URL: https://www.giac.org/paper/gcia/1231/defeating-sql-injection-ids-evasion/109964.