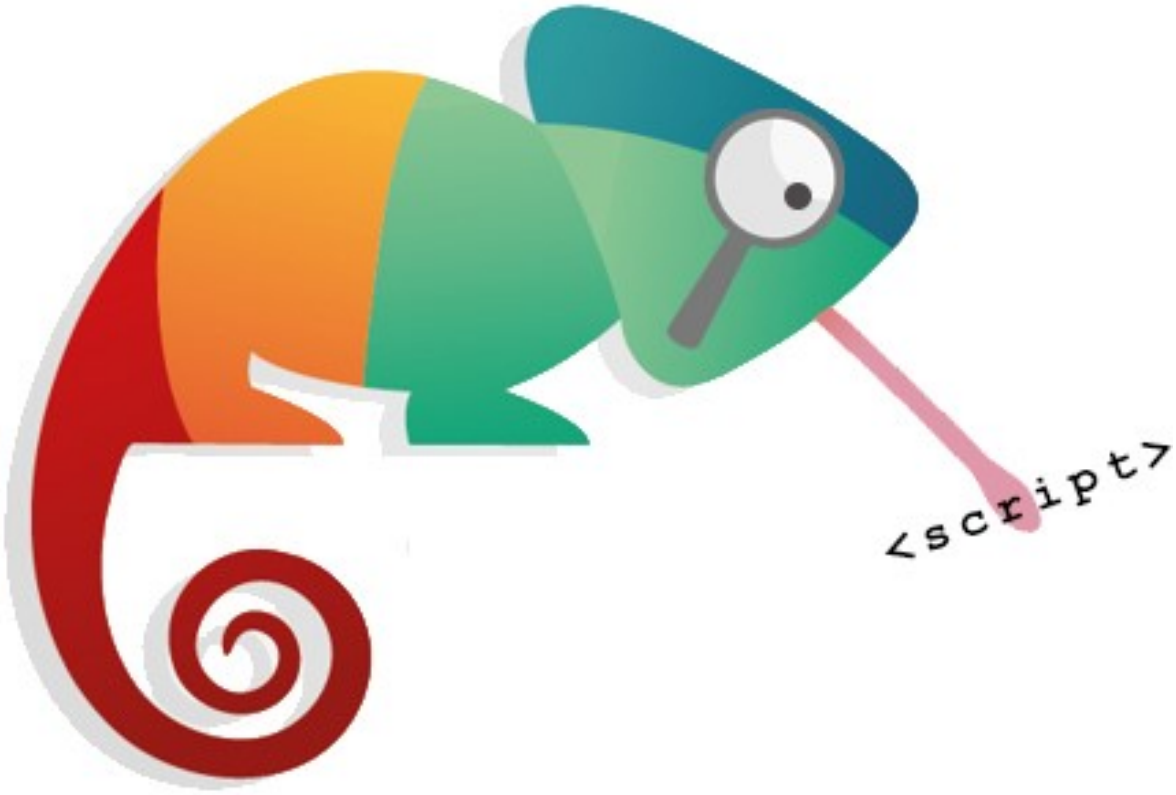
NOVEMBER 2, 2016



SELF-ADAPTATION TO CONCEPT DRIFT IN WEB-BASED ANOMALY DETECTION

MASTER THESIS

JELTE ORIJ
UNIVERSITY OF TWENTE

ABSTRACT

Attacks on web applications that utilize the HTTP request line and request body as attack vectors are amongst the most prevailing web-based attacks in the wild. For anomaly-based detection systems, which compare traffic to a model of normal behavior in order to detect attacks, a major challenge is to cope with "concept drift", which are legitimate changes in the monitored traffic caused by changes in the application to which the traffic belongs. This research proposes an anomaly-based detection system that is specifically designed to cope with this challenge. The system is based on different state-of-the-art techniques in web-based anomaly detection, as well as on the concept of "trusted clients". When clients have a history of trusted behavior, this is considered in the retraining process of the anomaly detection models, with which we aim to decrease the overall false positive rate of the system, especially during instances of concept drift.

CONTENTS

1. INTRODUCTION

When it comes to web application development and deployment, the security of the application and the infrastructure behind it has become an important factor. Attacks on applications and their infrastructure may cause disruptions in the service, as well as unauthorized disclosure and modification of data. From a business perspective, this may deter users which results in financial losses. There may also be legal consequences when, for example, privacy related data is insufficiently being secured. Therefore, it is important that appropriate security mechanisms are in place in order to prevent these scenarios from occurring. This research will focus on a relatively novel but promising type of system within the body of security mechanisms, namely an A-NIDS (Anomaly based Network Intrusion Detection Systems). More specifically, we will investigate certain improvements related to such a system as a way to enhance its overall effectiveness.

In this report we will first narrow our scope within the broad field of “security mechanisms” in chapter 2. Then we will provide more detail on the possible “attacks” in chapter 3 and in chapter 4 we further narrow down the scope within our chosen “security mechanisms”, discuss how they deal with these attacks and provide an overview of the state of the art techniques that are used in these systems. In chapter 5 the research questions are presented. In order to answer these questions, we will propose and test a custom built self-learning detection system, which we named *Scandax*. In order to determine the most optimal setup for *Scandax* we will analyze the available literature on the topic in chapter 6. In chapter 7 we will propose a system based on this analysis, as well as provide an overview of the implementation and in chapter 8 we will put the system to the test. Chapter 9 outlines the conclusions with the answers to the research questions and the report will end with a discussion about the possibilities for future work.

2. SECURITY MECHANISMS - SCOPE AND TERMINOLOGY

In the field of computer security, the level of security that is achieved is often measured as the extent to which confidentiality, integrity, and availability (CIA) are ensured. Confidentiality represents the rate at which unauthorized access of sensitive information is prevented, while integrity stands for the prevention of unwanted modification of data, and availability represents the rate at which systems are guaranteed to be available, i.e. able to perform their full functionality at all times.

CIA can be achieved by protecting the IT infrastructure against attacks or other malicious events that pose a threat against the assurance of CIA. This protection can be implemented on different levels. For simplicity we will take one server as the scope of what we want to secure. On this server, a wide range of applications may be running, some of which may be reachable from the outside via open ports. Now we can first make a distinction between measures related to security that are taken inside the application and the underlying infrastructure and measures related to automatic detection and prevention of the system as a whole.

Security measures inside the application may consist of secure coding practices and the automatic reporting of anomalies from within a certain application and security in the underlying infrastructure may be that the OS and running services are kept up to date.

When we look at the security of the system as a whole, different types of security systems can be applied. In general, at the highest level a distinction can be made between:

- Physical security; physical controls, such as security cameras, security guards, motion sensors.
- Host based security; software installed on a system in order to monitor and alert on the OS and application activity within that same system and possibly autonomously performs preventive actions.
- Network based security; software to identify anomalous behavior based solely on network traffic. A network based security system can be placed separately from other systems in the IT infrastructure, at least if it is ensured that the packets that reach the system that should be protected will also reach this security system.

A common principle when implementing security is to use different layers. In addition to the fact that it is probably impossible to prevent and/or detect all possible threats using one layer, using multiple layers can also aid in detecting a single kind of threat. For example, a malicious packet may go through two subsequent intrusion detection systems that are both inspecting the packet's contents and that may both be configured to detect certain types of attacks. One detection system may lack the capability to detect one specific variant of such an attack, while the other in its turn may be able to detect it. Different layers also apply to different types of intrusion detection and prevention systems. Although in this report we will handle one specific type of security system, combining the different types of security systems that we listed before usually aids in the overall security of the infrastructure.

For this research we will focus on network based security systems. Within the types of security systems there are systems of which the purpose is to either detect attacks, prevent attacks or both. In this research we will handle network based security systems for the purpose of detecting attacks. This type of system is usually referred to as a Network based Intrusion Detection System (NIDS).

A general purpose NIDS can be used to detect attacks in a wide range of protocols. For this research we will solely focus on the detection of attacks within the HTTP(S) protocol, i.e. web traffic. More and more businesses are using the web as a way to reach potential customers by serving a public website, or allowing the remote management of core business processes by existing employees, suppliers, and customers with, for example, a web based ERP system. For some businesses, such as web shops, a website is at the core of their business. From the web based systems in these examples it can be inferred that web based applications may deal with highly confidential information which should not become public and/or be altered by unauthorized persons. Also, availability of the application can be of vital importance to the organization. Another aspect to consider is that the systems that serve these web based applications may be connected to other systems within the organization, in a way that when the server which hosts the web based application is compromised by an attacker, this may provide the attacker with easier access to other systems within the

organization. Given these observations we conclude that it is essential for businesses to have proper security mechanisms in place that detect web-related attacks when serving a web-based application.

Because on a network based web-application level the systems for either detection and prevention are somewhat closely related, we will end this chapter by briefly discussing the main difference between network based systems related to detection and network based systems related to prevention, before we move on to the next chapter where an analysis on the possible attacks and the threats that come with those attacks are outlined.

With regard to prevention on a network based level, a commonly used tool is a firewall. We will briefly discuss a certain type of firewall that is used to prevent malicious events on a web application level and which is closely related to a NIDS, which in its turn is used for detection. This type of firewall is commonly referred to as a web application firewall (WAF), which can be used to restrict access based on whether the web server requests comply with the policy of the WAF.

The main difference between a NIDS and a WAF is that a WAF is mostly used for analyzing the 7th OSI layer, which is something most NIDS's are not able to do and which makes the WAF especially useful for securing web applications [16], [17]. On the other hand, although most NIDS's operate in higher layers of the OSI model, they are able to analyze many different protocols, while a WAF usually analyzes the HTTP(S) protocol. This makes the NIDS and the WAF supplementary layers of security, but there may be some overlapping functionalities. Both the NIDS and the WAF have in common that they detect anomalies based on a predefined set of rules. A rule, for example, may contain a request URL that is often used in a certain type of cross-site scripting attack (signature based detection). Another example of a rule is one that specifies that an alert should be created whenever the amount of connections suddenly increases with a certain rate (anomaly based detection). It is important that rules are carefully configured, because they are the main factors in the detection of attacks. When configuring the rules one should take into account the acceptable level of false positives and false negatives that the system generates with these rules. Proper configuration of the system and the rules can aid in lowering the level of false positives, while keeping a fixed level of false negatives. For NIDS's and WAFs there are both commercial and free open source products available. It may also be the case that the open source system is free but there are pre-configured rule-sets which can be purchased. Most of the times these commercial solutions are able to achieve an improved balance of false positives and negatives, as discussed before, because they have been carefully developed by trained professionals.

3. WEB-BASED ATTACKS

Web based attacks are considered by security experts to be one of the greatest risks related to confidentiality, availability, and integrity [43]. Web based attacks focus on an application itself and functions

on layer 7 of the OSI. In web applications, data is sent from the client and interpreted by the server, which may generate a response to the client. The data that is sent from the client is in the form of HTTP requests.

3.1 Anatomy of an HTTP request

An HTTP request is a collection of text lines (separated by a CRLF) sent to a web server and includes a *request line*, *request header fields* and the *body of the request* [44]. These are also the parts where the attack payload will be present in case of an attack, and thus which are interesting from the point of view of intrusion detection.

The *request line* has three parts, separated by spaces. The first part is designated for the method name which must be applied, the majority of HTTP requests being of the GET method, but others exist, such as POST or HEAD. In or report we limit the analysis to the most common methods, being GET and POST. Following the method, comes the resource path (URI) and an optional query component. The path is a hierarchically structured sequence of string segments that usually represents a file, a directory in the file system, or a combination of both. The "?" character in the URI introduces the query component of a path and parameters are supposed to be in field-value pairs, in which the pairs are separated by an "&" character, and the field and value are separated by an "=" character, but real-world implementations tend to break this convention. This is because expressive path names in URLs are preferred by developers and users, which is done in *URL Rewriting* [45]. However, for simplicity we assume in this report that we are dealing with field-value pairs of parameters. Moving on to the last part of the *request line*, this final part indicates the version of the protocol used by the client (generally HTTP/1.0 or 1.1). An example of a request line is the following:

```
GET   /path/to/file/view_employee.html?id=10   HTTP/1.1
```

Following the initial request line in a HTTP request, there are the *request header fields*, which provide information about the request. They contain data from the client in an unordered field-value structure. For example, headers inform the server which kind of content and encoding is understood by the client. The best example for client data in a header field that is processed by the web application is the so-called cookie. Attacks that have payloads in HTTP request header fields usually target the web server software that is hosting the web application, instead of the application itself. In our research we focus on attacks that target the web application and therefore we ignore the request header fields in our analysis. Note that we then also ignore the attacks on web application that do indeed take place via the header fields, such as via the Cookie field, but they are less common [44].

Finally, there is the request body, which basically contains the data bytes transmitted immediately after the headers. A typical GET request can transport parameters in the URI path, but the size of the query part is restricted by the server's implementation. For high-volume transmissions or forms, the POST method allows query-style or MIME-encoded data in the request entity content. An example of a MIME-type of content that is often used for non-binary and small-sized data is the *application/x-www-form-urlencoded* MIME-type,

which is specified in the Content-Type header field. For this content type, a query-style string is included in the request body, such as:

name=John&gender=M&age=24

For simplicity we will assume this Content-Type throughout the report for POST requests. However, we will also handle the other common request body Content-Types later in this report, such as *multipart/form-data*, *application/xml* and *application/json*.

3.2 Attacks in the HTTP request line and request body

Wrong handling of client data in any function of the web application can introduce a security weakness which may be exploited by an attacker. Most attacks on web applications make use of the HTTP request line, specifically the URI string and the request body to insert the payload. The Open Web Application Security Project (OWASP) has tracked and studied the trends of web vulnerabilities throughout the years, and periodically publishes their findings on this subject. Among their publications is a list of the 10 most critical web application security risks, a top 10 list of which an up-to-date version is periodically published. The latest version is the OWASP Top 10 from 2013 [46].

When we look at the top 10, we see that for many of the attack types the attack payloads contain valid data, where valid means that the attack cannot be distinguished from normal user behavior solely based on the payload. For example, with *Broken Authentication and Session Management*, which is placed 2nd in the list, an attacker uses legitimate authentication data to trick a badly implemented authentication mechanism into believing that the attacker is some known valid user, possibly one that is already logged in, such that the attacker is provided with unauthorized access. This kind of an attack makes use of valid request data. For example, one scenario that is described in the OWASP document is a GET request with a valid session identifier in the URI. In order to detect such an attack, the detection system would have to evaluate more than just the request line and request body of the HTTP request. Another example is number 4 on the list, *Insecure Direct Object References*. For this attack the attacker "*changes a parameter value that directly refers to a system object to another object the user isn't authorized for*". In this case the object is still a valid object, so this attack cannot be detected by solely looking at the request URI and body. The same counts for the other types of attacks, except for numbers 1, 3, 4, and 10, which are the types of attacks which our research focuses on. They are based on an abnormal payload in the request URI and/or the request body, and they are among the most prevalent attacks given that two of these kinds of attacks are in the top 3 of the OWASP Top 10. Actually we could also state that number 9, *Using Components with Known Vulnerabilities*, is also partly covered by our research, because this type is linked to every kind of attack.

Number 10 on the list, *Unvalidated Redirects and Forwards*, is a type of attack where there is a certain web resource that handles redirects in the application. When the location to which the web resource will redirect is retrieved from a parameter value in the URI query string, an attacker could forge a request with a malicious or unauthorized target to which the page will redirect, by changing the parameter value. In this case the

parameter value usually differs from the “normal” range of parameter values and the attack can thus be distinguished by only looking at the URI query string.

Where number 10 is mostly used to redirect other users to malicious web-based locations, number 4, i.e. *Insecure Direct Object References*, will mainly be used by an attacker to personally gain access to local unauthorized resources. This is another example of an instance where malicious user input can lead to unwanted effects when the input is not properly validated. A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without proper sanitization of user input, attackers will be able to manipulate these references to access unauthorized data.

Number 3, Cross-Site Scripting (XSS), occurs when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. An attacker can send malicious scripts to these specific parts of the application, exploiting the interpreter of the client’s browser, which will execute these scripts. The scripts can be crafted to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user’s browser using malware, etc. Although XSS is the most prevalent web application security flaw, it is not ranked 1st in the list because its impact is limited to the client side of the application, not affecting the server of the web application itself. In order to send the data to the application, the attacker will generally use the request URI and request body, because these are the parts of the request from which the application usually retrieves the incoming data, not counting the exceptions such as the Cookie header. Because the XSS payloads contain script-related special characters, they can usually be distinguished from normal data by inspecting the request URI and request body.

Injection tops the OWASP Top 10 list. Injection flaws occur when an application sends untrusted data to a server-side interpreter. A common example is SQL injection, which is a form of injection where user supplied data is included in an SQL command without properly validating or escaping that content. As opposed to XSS, which exploited a client-side interpreter, the SQL command is interpreted by the back end (in this case the database management system) of the application. This type of injection can result in the attacker having access and control over the entire database, which can cause data loss or corruption. Other types of injection, such as OS commands injection, can even cause the takeover of a complete host. As was also the case for XSS, the attacker will generally use the request URI and request body to send the payload, the content of which can usually be distinguished from normal data.

4. WEB-BASED NIDS TYPES - FURTHER NARROWING DOWN THE SCOPE

Although we have stated in the previous chapter that the payload for some of the most prevalent attacks that use the request URI and request body can usually be distinguished from normal data, there still is no

flawless solution for the automation of a proper detection process that is able to distinguish these attacks from normal data. For state-of-the-art web-based NIDS's there currently exists a trade-off between the detection of real attacks, called "true positives", and the unwanted behavior of classifying legitimate data as attacks, called "false positives". Different approaches to intrusion detection try to find an optimal trade-off, in which the rate of true positives is maximized and the rate of false positives is minimized. In the remaining part of this chapter we will discuss the most well-known approaches in web-based Network Intrusion Detection.

The CIDF ("Common Intrusion Detection Framework") is a working group created by DARPA in 1998, mainly oriented towards creating a common framework in the IDS field. The group has defined a general IDS architecture based on four types of modules:

- E-blocks ("Event-boxes"). Consists out of sensor systems that monitor a system, acquiring information and sending relevant information to the other blocks.
- D-blocks ("Database-boxes"). The purpose of these blocks is to store information that was acquired by the E-blocks before the information is further processed in the other blocks.
- A-blocks ("Analysis-boxes"). These blocks process the information that is available from the D-blocks, trying to detect potential hostile events and raising alarms when necessary.
- R-blocks ("Response-boxes"). When the IDS also incorporates prevention in addition to detection, these blocks are responsible for executing the appropriate action in response to certain malicious events that were detected by the A-blocks.

This structure is visualized in the following figure [1].

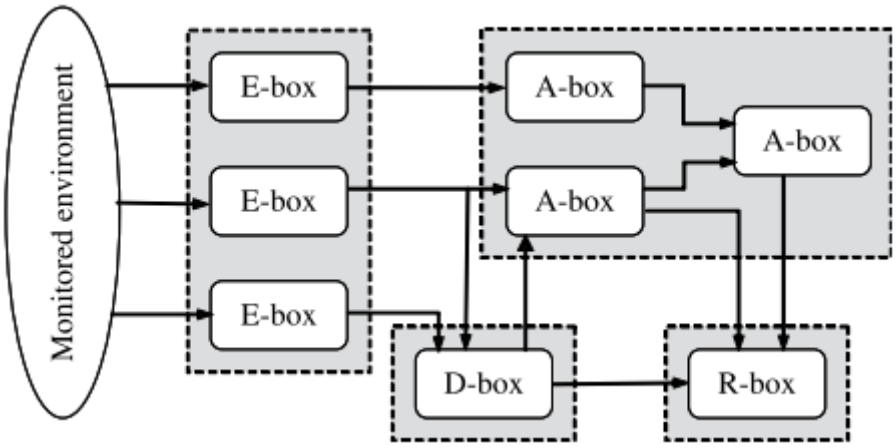


Figure 1 – IDS architecture as described by the CIDF

In chapter 2 we differentiated between host-based and network based systems when it comes to software based security systems. In the general IDS architecture, the distinction between these two types of systems can be seen from the different types of sensors that are used in the E-blocks. For host based systems, sensors usually analyze process identifiers and system calls, mainly related to OS information. On the other hand, for

network based systems the sensors are of a different kind, capturing and dissecting network traffic into useful information on, for example, the payload, the IP address, and the protocol.

We already mentioned that we will handle the network based systems in this report, more specifically those that operate on the HTTP(S) protocol. Now when we look at the Analysis-boxes for this type of system, we can further make a distinction between the ways in which information is analyzed in order to detect malicious events. A distinction is made here between signature-based and anomaly-based systems.

4.1 Signature based IDS

Signature-based systems try to find certain patterns ("signatures") in the traffic. The signatures specify certain filters, usually on a per-packet basis. These filters consist of, for example, the destination port number, certain hexadecimal or ASCII codes in the packet payload, or a certain value in a packet header field. The signatures represent malicious behavior; in essence the total set of possible malicious behavior is specified and stored beforehand and the live traffic is analyzed and scanned for malicious behavior based on this model. The total set of possible malicious behavior is hard to define at a certain point in time. Moreover, the set is ever expanding as new attacks are crafted based on, for example, newly discovered security vulnerabilities in a certain application or protocol. Therefore the signatures usually represent the attacks that are currently known in the industry and in general a signature based IDS relies on these signature sets, which are either publicly available or can be purchased from certain parties. Since these signature sets are created based on all attacks that are known in the wild, the properties of a specific network are not taken into consideration. When implementing a signature based system in a network, it usually requires tweaking or disabling part of the signatures in order to make the system more effective for that specific network.

Because the creation of new signatures is not automated based on newly discovered attacks which occur in the wild, there is a delay between the time a new kind of attack is discovered and the time the (publicly) available signatures are updated. Research has shown that signature updates are typically available later than software patching releases. Moreover, signature updates are generally released within the first 100 days from a vulnerability disclosure [2]. This means that for signature based systems, there can be quite a long period of time during which the IDS is incapable of detecting an attack which is already known to the public.

Most of the time a signature is specified for (part of) a single attack, such as a certain shellcode payload or buffer overflow attack. In such cases, signatures perform well when it comes to detecting these specific attacks. However, a shortcoming is that in general an attacker only has to create a tiny variation for such an attack in order to already be able to circumvent the IDS. For example, the content of the attack could be scrambled by using some very simple form of encryption.

In addition, there are also signatures which cover a range of possible attacks, such as for SQL injection and XSS attacks. It has been shown that the default Snort rules against SQL injection are good but not comprehensive [3][5]. For attack types such as SQL injection and XSS attacks a signature will filter the payload of the packet, generally by using a regular expression (pcre). In [4] it is described that it is infeasible to cover

the entire set of possible malicious traffic patterns related to SQL injection using this signature based regular expression approach, mainly because the set of signatures will become so large that the processing time will become unacceptable. Note that for a signature based system such as Snort, traffic will be compared to each available signature, such that a larger set of available signatures will increase the processing time for each packet. This may become a serious burden on large-volume networks.

Based on this analysis we can conclude that although signature based systems perform well in detecting a wide range of attacks, it is hard to detect the more sophisticated attacks. In addition, a signature based IDS will not be able to detect zero-day attacks, although these occur less frequent in the web-application area which we are focusing on[1].

4.2 Anomaly based IDS

In contrast to signature based systems, an anomaly based IDS compares traffic with a model of "normal" behavior and raises an alert whenever the traffic does not correspond to the model. Another important difference compared to signature based systems is that the model is created autonomously, in a period during which live traffic is observed. This is different for signature based systems, which usually rely on pre-defined signatures. Although these signatures may be disabled or tweaked to a certain extent and in some cases new signatures may be added based on the monitoring preferences on a specific network, this will all involve manual work. If not, such as a system which is able to identify normal behavior, but creates a signature for all behavior that does not correspond to this model (creates an anti-model), would actually be an anomaly based system. In such cases the system may raise an alarm whenever the difference between the observed behavior and the model falls below a given limit [1]. In this research we will focus on anomaly based systems which create a model that corresponds to normal behavior, as opposed to a model of anomalous behavior.

In addition to detecting attacks on the monitored system itself, an anomaly based IDS can also be useful to detect other kinds of abnormalities occurring in the applications that run on the system. For example, on a web application level a bank may be interested in knowing whether an anomalous value was entered in the "amount" field of an HTML form that is used to create a transaction. Although this sanitation seems more like a responsibility for the part of the application that handles the form processing, the implementation of these security checks within the application can be overlooked. In case an attacker performs such an attack on the application, the anomalous value does not necessarily contain attack-specific characteristics, such as special characters or SQL keywords. Instead, the submitted value could for example be an exceptionally big number. For a signature based system it would be harder to detect such an "attack" than for an anomaly based system, the latter of which could have a model wherein it is specified that the "amount", which represents a

[1] Note that the term "zero-day attacks" here refers to unknown SQL/XSS attacks which are not based on a vulnerability in the web-application itself (such as improper SQL escaping), but rather a vulnerability in, for example, the SQL engine or Javascript parser.

monetary value, contains a number of 1 to 7 digits, optionally followed by a dot and 1 to 2 digits for the decimal value.

4.2.1 SCOPE WITHIN ANOMALY BASED IDS

Within the field of anomaly based intrusion detection systems a certain IDS can further be classified based on several characteristics. Estevez-Tapiador et al. provide three criteria with a corresponding classification tree [7]. We will briefly describe the proposed taxonomy and based on this we will classify the type of system that is subject to this research.

4.2.1.1 Network feature analyzed

A certain network can be studied from several points of view. The unique network related property that is analyzed by intrusion detection systems is the traffic within a network. Methods that create models from observed traffic can be divided into two groups, which are flow analysis and protocol analysis. Flow analysis is concerned with observing the variability of certain measures over time. For example, the number of IP/TCP/UDP/ICMP packets sent/received by a certain host during a fixed time interval can serve as a measure for flow analysis. This research is not about flow analysis, but instead we will focus on protocol analysis. Within protocol analysis the classification in [7] somewhat relates to the OSI model. Protocol analysis can be performed on a data link level (ethernet), network level (IP), transport/control level (TCP, UDP, ICMP, etc.), and application level (HTTP, DNS, FTP, SSH, etc.). The system in this research performs analysis on the application level, specifically focusing on the HTTP protocol.

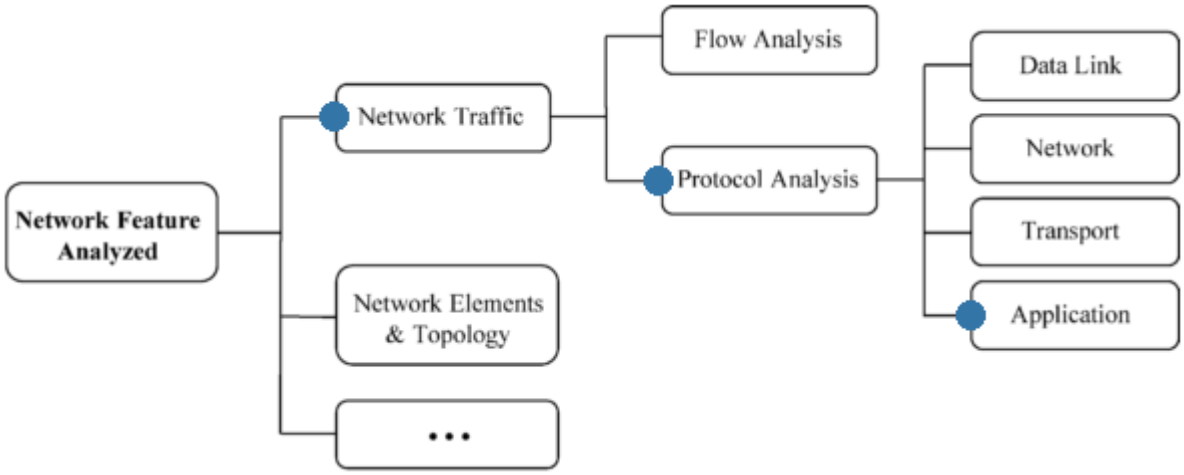


Figure 2 - IDS scope: Network feature analyzed

4.2.1.2 Analysis scale

Because we have already specified that we will focus on protocol analysis, we will limit the notion of the analysis scale provided in [7] to this form of analysis. An important reason for classifying an IDS based on the scale of analysis has to do with the fact that certain anomalies/attacks are only observable at certain scales. The “land” attack, for example, is characterized by an IP packet in which both the source and destination

addresses are equal and can easily be detected by inspecting each packet separately [8]. On the other hand, detection of certain forms of DDoS attacks usually requires a certain correlation or aggregation mechanism among different sources and connections, since the anomaly will not be able to be inferred from the inspection of individual packets.

On a level of protocol analysis, the microscale entails the analysis of individual packets. On the mesoscale connections or packet streams are analyzed and finally there is the macroscale in which analysis of several connections is performed simultaneously and event correlation is done within the whole network. The HTTP protocol analysis in this research is carried out on a microscale, i.e. the inspection of individual packets.

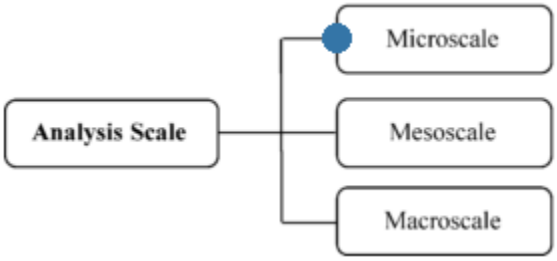


Figure 3 - IDS scope: Analysis scale

4.2.1.2 *Behavior model*

The model of normal behavior lies at the center of an anomaly based IDS. In general there are two main approaches for the construction of such a model. The first approach is based on self-learning techniques in order to automatically obtain a representation of normal behavior from the analysis of network traffic. The second approach is one in which the specifications for normal behavior are provided manually. Specification-based approaches are especially useful when the number of entities that is modeled as well as the specifications of these entities remain relatively fixed and are not highly susceptible to change. For example, the description of the inner workings of many protocols are, in general, described in IETF RFC's. When network traffic is to be analyzed for only a small and fixed number of protocols, these RFC's could be translated into models of normal behavior. However, it has been reported that creating models from specifications is difficult and time-consuming [1][7]. Moreover, in environments in which new models should frequently be created and adapted, specification based modeling is often infeasible. In such cases learning-based techniques are preferred. Because in [1] a more elaborate overview of the different learning techniques is provided when compared to the taxonomy in [7], we have extended the tree from [7] with the learning techniques from [1].

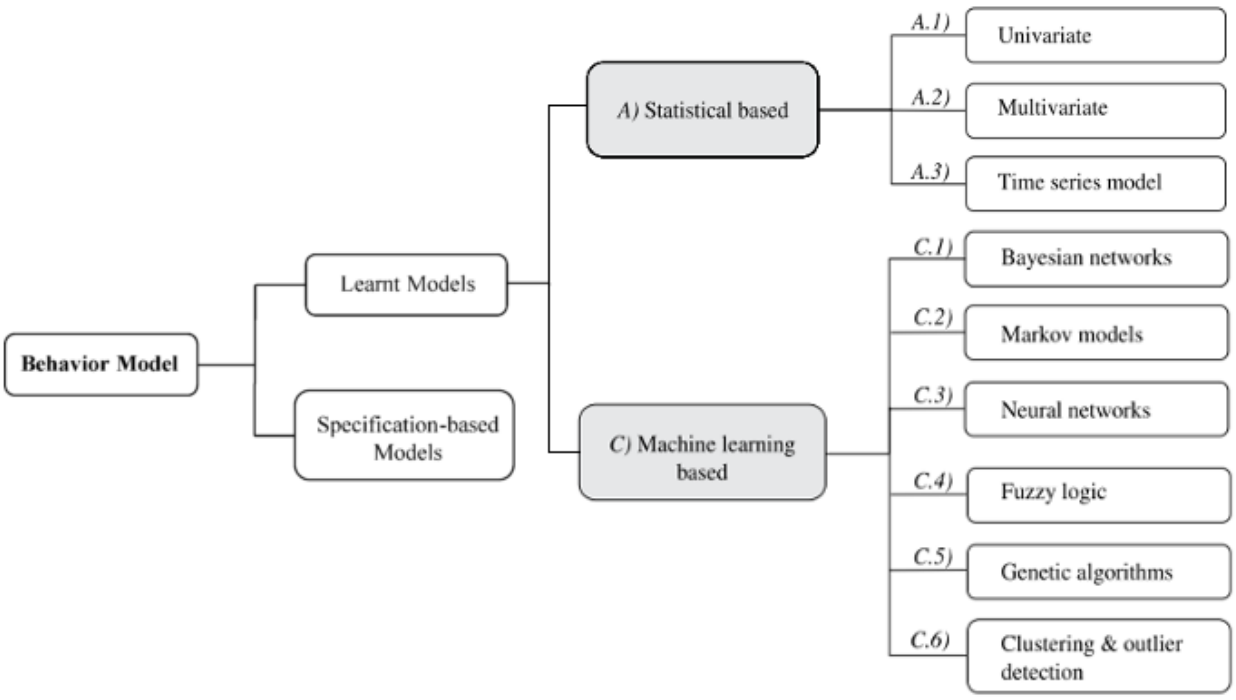


Figure 4 - IDS scope: Behavior model

Within the learning techniques a distinction is made between statistical and machine learning based techniques. However, this distinction is only subtle. With statistical based methods, it is assumed that the "normal" behavior can be modeled with a certain statistical distribution. Statistical based anomaly detection techniques use statistical properties (e.g., mean and variance) of normal activities to build a statistical based normal profile and employ statistical tests to determine whether observed activities deviate significantly from the normal profile, which in its turn represents a statistical distribution. The IDS will assign a score to an anomalous activity and as soon as this score becomes greater than a certain threshold, the IDS will generate an alarm [10]. Machine learning approaches also are able to autonomously create a model from observed traffic, but as opposed to statistical based methods, a machine learning IDS has the ability to change its execution strategy as it acquires new information. This makes the machine learning approaches especially suitable in situations where the normal behavior changes over time, given that the rate of change is below a certain threshold, because otherwise the behavior would be flagged as anomalous. Note that this change has to do with the changes in the normal behavior in the monitored environment, when the initial model has already been created. This notion therefore applies to the retraining stage as seen in Figure 5. In contrast, statistical methods need accurate statistical distributions, but not all behaviors can be modeled using purely statistical methods. Moreover, a majority of the statistical anomaly detection techniques require the assumption of a quasi-stationary process, which is not able to adapt to legitimate changes in a user's behavior, which in its turn cannot be assumed for most data processed by anomaly detection systems [9][10][12]. Although the self-adapting feature could make it desirable to use machine learning schemes for all situations, the major drawback is their resource expensive nature [1][11].

4.2.2 CREATING AND UPDATING THE MODEL

In order to generate the model that lies at the center of any anomaly based IDS, a certain model creation algorithm is used. This algorithm is usually fed with a simplified version of the data that is monitored, consisting only out of the characteristics that are of interest and which can serve to represent normal behavior. The process of converting the monitored data to a simplified, generic representation that can be included in the model is often called “dimensionality reduction” or “feature extraction”. As soon as the initial model has been created from observed traffic, the detection can start. For different model creation algorithms, there are different ways of determining at what moment the model has reached an acceptable level of accuracy in order to be able to start the detection phase with this model without triggering an unacceptable false positive rate due to the inaccuracy of the model. Finally, because the monitored environment is often subject to change, the model should also adapt to this change. This means that either the existing model is used and is retrained based on newly observed legitimate behavior, or a new model is created. This process is further described in chapter 6. An overview of the different phases that can apply to the model is in the figure below. Note that this schema applies to the situation when a model is constructed based on network traffic. As we will see in chapter 6, it is also possible that a model is created based on the inner workings of the software that is served to the user. For this purpose, the set of range of possible legitimate interactions that the user can perform with the software is derived from the application’s source code.

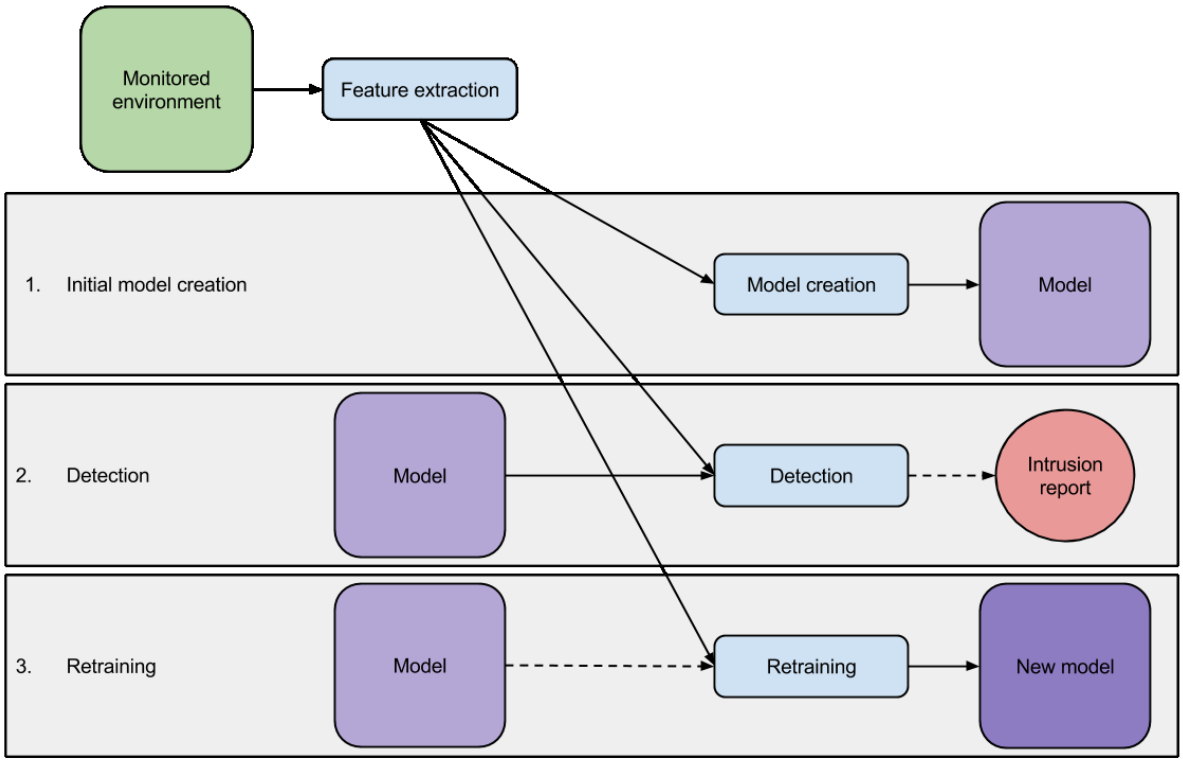


Figure 5 - Anomaly based IDS architecture

In Figure 5, the different stages that apply to the life-cycle of the model are displayed. All three stages make use of features that are extracted from events that take place in the monitored environment. In the first phase, these features are used to create a model. In this stage it is often preferable that the rate of attacks that is present in the environment is low, such that the model will not be polluted with this data. In the detection phase, the extracted features are compared with the model. Possibly, the features from the monitored event do not adhere to the model, such that an alarm or intrusion report will be generated. The third phase is optional; not every anomaly based IDS does incorporate this step. This retraining step is triggered in a response to newly observed legitimate behavior in the monitored environment which is not yet present in the model, and thus generates false positives. For the retraining phase either the existing model can be updated, or an entirely new model is constructed.

We conclude this chapter with the definition of an "autonomic" anomaly based IDS as described in [34], as it largely applies to the notions of model creation, updating and retraining. The authors describe that an autonomic IDS should have the following abilities:

- Self-labeling: automatically identifying the anomalies in unlabeled data streams;
- Self-updating: continuously updating the detection model by incorporating incoming labeled normal data in order to maintain an accurate detection model over time;
- Self-adapting: adapting the detection model to behavioral changes by re-building the model as soon as a change is detected in data streams.

The self-updating ability consists in updating the detection model to take into account the normal variability of the data. On the opposite, self-adapting consists in rebuilding the detection model in case of behavioral changes.

4.2.3 STATE OF THE ART IN WEB-BASED A-NIDS

As was said earlier, this research focuses on an anomaly based NIDS, specifically for detecting web-based attacks. When it comes to this type of attacks, a requested web resource and its attributes can be used by an attacker to perform attacks as SQL injections, buffer overflows and directory traversal attacks. In addition, a HTTP request message contains header fields, which define the operating parameters of an HTTP transaction. These fields usually contain information about the user agent, preferred response languages, connection type, referrer, and additional meta-data. An attacker can inject malicious code to these fields to construct various kinds of attacks based on HTTP response splitting or malicious redirects [13]. For detecting anomalies in HTTP requests, usually the web resources, attribute values, and HTTP header fields from the requests are analyzed. In addition, time-based statistics may serve to detect scanning and brute-forcing attacks. When we look the current techniques that are used in web-based A-NIDS, there is a distinction between techniques that are used for feature extraction and techniques that are used for model creation (refer to Figure 5 for both of these steps). We will handle the state of the art of the techniques that are used in these steps in turn. For a visual overview, refer to Attachment A.

4.2.3.1 Feature extraction techniques

With an anomaly based system we will compare traffic with an existing model that represents “normal” traffic. In general, this model has been created from data which does not necessarily contain the range of all possible legitimate behavior for the environment that is being monitored. With respect to the data that has been gathered during the data acquisition phase, it is therefore often desired to make a generalized representation of this data when creating the model, such that the system will not only allow the specific values that have been observed, but also all values within the range of the general representation. For example, when observing multiple requests that contain the attribute “id” for a certain web resource, we may acquire the following attribute values: 100, 540, and 9281. After observing more and more different values we would like our model not only to allow these specific values to be considered valid, but rather a certain range of integers, or simply all integers. This way of generalization allows us to reduce the dimensionality of the problem significantly without losing relevant information needed for the intrusion detection. The flexibility that is included in this part determines the rate of false positives and false negatives; allowing less flexibility will result in more false positives, while more flexibility will increase the number of false negatives. Therefore, it is important that a well substantiated decision is made with respect to this flexibility. In order to increase the flexibility of the model relative to the situation where one would only allow the values that were observed during the training phase, we will try to generalize the values by extracting certain features that are of interest. Different features can be chosen as a way to generalize the data. Because our research is focused on web-based detection, we will limit the analysis of feature extraction methods to those that are applicable to web traffic.

One possible feature that is often used in web-based detection is that of overlapping character sequences (an n-gram model). This approach is, for example, used in [13] and [14]. Here the characters of the data strings first go through an abstraction function. Since, when making code injections, attackers use specific combinations of non-alphanumeric symbols, the usage of those symbols is of the most interest. Therefore, in order to concentrate on them, the abstraction function that is used in the aforementioned research will consider all alphanumeric characters as the same character. The abstracted strings can then be analyzed using n-grams. In [13], n-grams are created for the entire request string, including the web resource and attribute values. The authors argued here that this would cause the final model to become less complex. However, in [14], which is later work of the same authors, the web resources and attribute values are analyzed separately. In addition, [14] proposes three extensions to the n-gram model, incorporating the frequency of occurrences of the n-gram in the attribute value, the frequency of occurrences of the n-gram in the entire training set and the frequency of occurrences of symbols from the n-gram in the training set.

In [37] several possible features are described that can be used in HTTP anomaly detection for web applications that include CGI scripts. However, some of these features can also be used for HTTP anomaly detection in general. First off the request length is named as a possible feature. This is because of the notion that buffer overflows and cross-site scripting attacks tend to be longer than regular traffic. A second feature that is proposed is the character distribution of a request. This is because of the notion that buffer overflow

attacks, cross-site scripting attacks and path traversal attacks in general have a different character distribution than legitimate requests. This specific feature can also be regarded as a 1-gram method.

In addition to detecting attacks that are based on a single HTTP request, [14] also considers the detection of scanning and brute-forcing attacks by creating a model for network traffic with respect to time. In this method a time interval is split into parts of equal length, called “time bins”. Each request is characterized by a set of parameters that may serve in detecting certain attacks. The parameters include the user’s IP address, requested web resource with its attributes, the amount of transferred bytes and server’s response code. The parameters of a request will be put in a specific time bin that corresponds to the point in time at which the request took place. As a final feature the authors calculate what they call the “sample entropy” as a way to capture the degree of dispersal or concentration of the parameters’ distributions.

It is important to note that the features that have been extracted in the phase that has been described in this paragraph in itself do not directly compose the entire model for “normal” behavior. In order to create the final model against which we can classify newly observed requests, the features that were extracted during the training phase will be processed using certain data-mining techniques. We will handle these in the next paragraph.

4.2.3.2 Model creation techniques

Generally the literature about model creation techniques for web-based A-NIDS’s is about machine learning based techniques and less about statistical based techniques. This seems to be related to the advantages of machine learning based systems when it comes to changes in normal behavior in the monitored environment, which were discussed in paragraph 4.2.1.2. A large part of the literature in which these techniques are proposed, also explicitly mention the advantages with respect to the self-adaptation capabilities of the model.

In [13] the combination of using n-grams with DBSCAN for the entire URI gave optimal results relative to other often used methods, although [14] is later work from the same authors and here the web resources and attribute values are treated separately. In this paper they also compare different classification methods, namely SVDD, K-means, DBSCAN, SOM and LOF when applied to web resources, attribute values and user agents. They have found that their method, which is using SVDD to classify web resources, K-means for attribute values and DBSCAN for user agents performs optimal with respect to the detection accuracy. It can therefore be assumed that this combination of techniques gives relatively good results. However, the advantages and disadvantages with respect to retraining also need to be taken into account. As was described earlier, with the approaches in [13] and [14] the retraining can be done on the fly, which makes them suitable for self-adaptation.

In [15] a well-known neural network technique named SOM (Self Organizing Map) is used for one of the steps in the classification phase. A SOM is a neural network model based on unsupervised learning. It was proposed by Kohonen for analyzing and visualizing high dimensional data into a smaller dimensional space. It does this by detecting inherent structures in the high-dimensional data and subsequently mapping this data onto a

two-dimensional representation space, containing a grid of neurons. In a SOM the vectors that are closely related are located near each other in the neuron grid. The structure of the SOM automatically adapts as new inputs are provided. In [15] the SOM is used to determine the number of the neuron that most closely corresponds the payload, i.e. the "winning" neuron, which in its turn corresponds to a certain model that was created during the training phase. This model is subsequently used as the reference value in a classification algorithm called PAYL, which will eventually determine whether the observed request is anomalous based on an n-gram analysis. The authors substantiate the use of a SOM by stating that it is better able to deal with high-dimensional data than, for example, K-means and K-medoids. Actually the SOM and PAYL combination is only used for parameters that contain a certain minimal amount of special characters. Up to some amount of special characters, the researchers propose a method that is able to infer regular expressions from a series of example input values. The regular expressions are automatically adapted so that they also allow "similar" values. One argument in favor of using regexes is that they are easy to understand and tweak by a system administrator.

In [16], the authors put forward two main drawbacks of using a SOM. They mention that the size and dimensionality of the basic SOM model is required to be fixed prior to the training process and there is no systematic method for identifying an optimal SOM configuration. In addition, a traditional SOM cannot represent hierarchical relations that might be present in the data. A Growing Hierarchical SOM (GHSOM) would overcome these issues. A GHSOM consists of several SOM's structured in layers, the number of which together with the number of neurons in each map and maps in each layer are determined during the unsupervised learning process. Thus, as with a SOM, the structure of the GHSOM is automatically adapted to the structure of the data.

In [17] the limitations of the SOM are tackled by using a DGSOT (Dynamically Growing Self Organizing Tree), which is also hierarchical by nature. This clustering algorithm is augmented with a combination of SVM (Support Vector Machines) as a way to save computation overhead when creating the tree. Unfortunately, no empirical research was performed with this technique, so it is hard to compare its advantages to, for example, the GHSOM approach that was proposed in [16]. This issue also applies to the tree structure used in [18].

In [19], a somewhat different approach is adopted. Here, the occurrence probability of each character in a payload is considered to model the traffic, the payload being the URI string. Following this idea, the authors extend the model by calculating the conditional probability, i.e. the probability of a character occurring given the previous one. The authors make clear that certain patterns exist with respect to this conditional probability matrices, which can be used to differentiate between normal and abnormal traffic. In order to use the feature of the conditional probability as a way of classification, they make use of a Markov chain. The transition matrix is represented by the conditional probabilities and the initial probability vector consists of the probability of a given character being the first character in the payload. In order to determine whether an incoming request is anomalous with respect to the current Markov chain, a formula called LogMAP is introduced that calculates the probability that the observed sequence has been generated by the existing

chain. In addition, the authors test some extensions to the algorithm and in the end they are able to achieve a 5% false positive ratio. However, even lower false positive ratios have been reported in [13] and [14]. In [13], first all unique n-grams and their frequencies of occurrence in the training set are determined. Subsequently it is argued that for a certain request, the ratios of the frequency of appearance of abnormal n-grams in that request and those n-grams in the training set can be used to classify the request either as "normal" or "abnormal". However, some attacks may contain many different n-grams with a low frequency of occurrence, which would lower the ratio, classifying the request closer to normal behavior. In order to overcome this issue, the authors have proposed a formula in which a large number of different, yet abnormal n-grams are also classified as abnormal. With this formula the "distances" of each request with respect to normal behavior can be defined. Once this has been done, this data is classified using a clustering technique called DBSCAN. This algorithm will create the final model against which outliers can be detected. The authors argue that this method is more simplistic and performs better than using K-nearest neighbor, N-gram in combination with GHSOM and N-gram in combination with Diffusion Maps with regard to the true positive rate, the false positive rate, the accuracy (i.e. the ratio of the total number of correctly detected requests to the total number of requests in the testing set) and the precision (the ratio of the number of correctly detected intrusions to the number of requests classified as intrusions).

For classification of the header fields the authors in [13] use the same technique as for the requested URI's, but a simplified version that does not include the DBSCAN clustering step. They only use the custom formula that calculates the distance between the observed request and normal behavior. If at least for one header type the distance is greater than zero, the query is classified as intrusive, otherwise it is considered as legitimate.

In [14], the authors use SVDD (Support Vector Data Description) to classify the web resources. Support vector data description (SVDD) by is a method to find the boundary around a data set. SVDD has been successfully applied in a wide variety of application domains such as handwritten digit recognition, face recognition, pattern-denoising and anomaly detection [20]. The algorithm tries to define a spherical boundary, in feature space, around the data. The volume of this hypersphere is minimized, to minimize the effect of incorporating outliers in the solution. This as opposed to the related SVM (Supported Vector Machine) method, which creates an optimal hyperplane instead of a sphere. There is a lack of studies that compare the two methods and it is unclear in which situations one would perform better than the other. In [21] it is argued that both methods are different versions of the USVM (Unified SVM), an algorithm which the authors have proposed in the paper and which seems to perform better than when using SVDD or SVM separately. Future research could investigate whether USVM outperforms SVDD in the area of HTTP anomaly detection.

The researchers that originally invented SVDD are Tax and Duin [22]. In their paper they use a method called PCA (Principal Component Analysis) to reduce the dimensionality of the data before processing it with SVDD. In [14] the authors adopt this approach. The feature vectors containing the n-gram data will be mapped to a new coordinate system where the number axis, called principal components, is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component

represents the largest possible variance among any projection of the data set (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance. In essence, with PCA one tries to fit an n-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. A small axis in the ellipse will mean that the variance along that axis is also small. When one would omit that axis and its corresponding principal component from the representation of the dataset, only a small amount of information would be lost. In order to create this structure, the eigenvectors of the covariance matrix of the feature vectors are required.

For classifying query attributes the researchers in [14] use a simpler method, namely K-means. This is an unsupervised partitioning technique that classifies a dataset into clusters. The number of clusters is provided beforehand and the algorithm tries to minimize the sum of distances between each feature vector and the mean of the cluster which the vector belongs to. For classifying the header fields, DBSCAN is used, the same way as it was used in [13] to classify the web resources and query attributes. Only the User Agent field is considered. Finally, time bin statistics are classified by applying z-score normalization to the time bin entropies that were obtained during the feature extraction phase and for the detection phase the authors propose a threshold formula in order to differentiate between anomalous sequences of requests. In the end the authors claim to have reached a 0.8% false positive ratio for web resources, a 0% false positive ratio for attribute values and a 2.5% false positive ratio for User agent strings.

5. UPDATING THE MODEL - THE SUBJECT OF RESEARCH

When a web-application is changed by its developers, this may cause noticeable changes in HTTP requests that are sent to the server. Web resources, attribute names and attribute values are therefore subject to change. Because of the fact that a model of normal behavior is at the basis of the anomaly detection engine, this model should be able to adapt to these changes in the environment. The notion of self-adaptation of a detection system to legitimate changes in the environment was already a subject of research several decades ago [38]. However, self-adaptation for web-based detection systems is a more recent topic, and the availability of research on this topic is still quite scarce. Most papers that mention the concept of self-adaptation, actually refer to the fact that during the training phase the model is able to adapt autonomously to different inputs, which is a criterion that is often used in favor of certain machine learning based systems, such as the GHSOM in [16]. Several papers do not even mention the important aspect of self-adaptation and others, such as [13] and [16], simply suggest to retrain the system after a certain period of time, without further specifying the specifics related to this timespan and the retraining method.

It does not require an explanation that the updating, or self-adaptation of the system is a very important aspect, since it greatly affects the number of false positives and the overall usability of the system. Because there is ambiguity surrounding this area, it would be valuable to have a more specifically defined method that can be used for this purpose. In [13] it is proposed that the system needs to be retrained after a certain

period of time or after processing a certain number of requests. In order to fine-tune this definition so that we can come up with a better defined method for self-adaptation, it would be valuable to investigate the specifics regarding the number of requests or the time that passes before the system needs to be retrained. From a usability perspective it would be preferred to have the system determine this automatically. This could, for example, be achieved with historic data about changes in the application which could be inferred from HTTP requests by looking at changes in parameter names and values. A metric should be defined then that is able to classify a change in parameter value or name as being either a change in the application or a true positive, for example when requests that include a new web resource, attribute name or attribute value is requested for a certain number of times from different IP addresses. However, it is important that an attacker is not able to abuse this part of the system. Therefore, there is a trade-off between security and usability and the proper balance need to be found when creating this "legitimate change logic". In the described case, an attacker could have control over different IP addresses and then initiate an attack. An additional measure may therefore be to only consider IP addresses of users that can be considered legitimate.

This research will focus on this trade-off between security and usability while aiming to define a more specific method for the self-adaptation of a web-based ANIDS to legitimate changes in the environment. In a more general way this is reflected in the main research question, which is the following:

HOW CAN A WEB-BASED ANIDS AUTONOMOUSLY ADAPT TO LEGITIMATE CHANGES IN THE MONITORED WEB APPLICATION?

As we have described before, two important aspects in the self-adaptation process are distinguishing legitimate changes as well as retraining the model. From the research that is available on these topics, we will select the methods that apply optimally to our area of interest, which are web-based intrusion detection systems. The following two sub questions reflect these matters:

WHEN CAN AN OBSERVED CHANGE BE CONSIDERED TO BE LEGITIMATE?

–

WHEN A LEGITIMATE CHANGE HAS BEEN DISTINGUISHED, HOW TO UPDATE THE MODEL?

Given these sub questions, in the next two chapters we will investigate methods for determining when a detected change in the monitored environment is actually legitimate (i.e. let the IDS be able to autonomously prevent false positives), as well as methods to update the existing model so that the new model will account for the changes in the monitored environment. As will become apparent in chapter 6, the methods that are described in the existing research on these topics make use of different model creation techniques. The optimal method for distinguishing legitimate changes, as well as for retraining the model may be more

compatible with certain model creation algorithms, and incompatible with others. This brings us to the final sub question:

ARE CERTAIN TRAINING ALGORITHMS MORE PREFERABLE THAN OTHERS WHEN IT COMES TO THE PROPOSED METHODS FOR DISTINGUISHING LEGITIMATE CHANGES AND RETRAINING THE MODEL?

Different techniques related to the research questions will be described in the next chapters, and we will propose and test our own method for self-adaptation based on a selection of techniques that will be combined into one system. In chapter 9 we will conclude this research by answering the research questions based on whether our proposed system proves to be successful.

6. SELF-ADAPTATION

In case of a change in system behavior, the base profile must be updated with the corresponding change so that it does not give any false positives alarms in future. This means that the system either continuously incorporates the live traffic into the model in real-time, or otherwise uses a mechanism for deciding whether to update the model. If the system tries to make a change to the base profile every time it sees a deviation, there is a potential danger of incorporating intrusive activities into the profile. The IDS must be able to adapt to these changes while still recognizing abnormal activities. If both intrusive behavior and a change in normal behavior occur during a particular time interval, the problem becomes more complicated. There are also additional issues that need to be addressed in case of updating. The system should adapt to rapid changes as well as gradual changes in system behavior. Selecting the time interval at which the update should take place is also an important issue. If the interval is too long, the system may miss rapid changes or short-term attacks. If the interval is too short, the system may miss some long-term changes. Different approaches to self-adaptation will be discussed in this chapter. First, we will handle those systems that are able to determine the point in time when a legitimate change has been observed, which serves as a trigger for updating the model and which is therefore more efficient than the methods in paragraph 6.2, which continuously update the model based on live traffic. A system that uses a combination of the advantages of both approaches is discussed in paragraph 6.3. In addition to describing the method to determine the threshold for the systems in paragraph 6.1 and 6.3, we will also discuss the approaches for retraining that can be used for these systems. For systems that continuously update the model, the description of the retraining approach is already interlaced in the description of the system and therefore there is no separate section about retraining in paragraph 6.2. We end this chapter with paragraph 6.4, which includes general notes on retraining.

6.1 Detecting abrupt and gradual changes in observed traffic: threshold-based systems

Threshold-based systems make use of a certain threshold that determines when to evaluate whether recent observed changes are legitimate or illegitimate. In case of legitimate behavior, the model is usually re-created or updated so that it incorporates the new legitimate behavior. For illegitimate behavior, an alert will usually be triggered when this has not already been done during the individual detection of the outliers. In general these systems use a threshold that represents the rate and/or amount of new outliers, but simple time-based thresholds are also used. In this chapter we have made a distinction between systems that detect changes from observed traffic (paragraph 6.1.1) and systems that detect changes from the underlying system (paragraph 6.1.2). In paragraph 6.1.3 we include a few remarks on the retraining of the models for this type of systems.

6.1.1 SYSTEMS THAT DETECT LEGITIMATE CHANGES FROM OBSERVED TRAFFIC

For distinguishing legitimate from illegitimate changes in the observed data, the authors in [24] investigate a host-based approach where they continuously measure the similarity between each day's activity and the profile of system calls executed by users and utilize this similarity trace. If the similarity stays above a threshold level, then the profile is taken to be a correct reflection of the current activities. If the similarity goes down below the threshold level, then there can be two possibilities: either the behavioral patterns are changing or the system is under attack. In order to distinguish between these two possibilities, the rate of change in the similarity is measured. If an abrupt change is encountered, it is interpreted as an intrusion, and that time window will not be used to update the profile. If a gradual negative change is encountered, then that time window will be used to update the profile. It is assumed that behavioral change occurs gradually, not abruptly. This is illustrated in Figure 6, taken from [24].

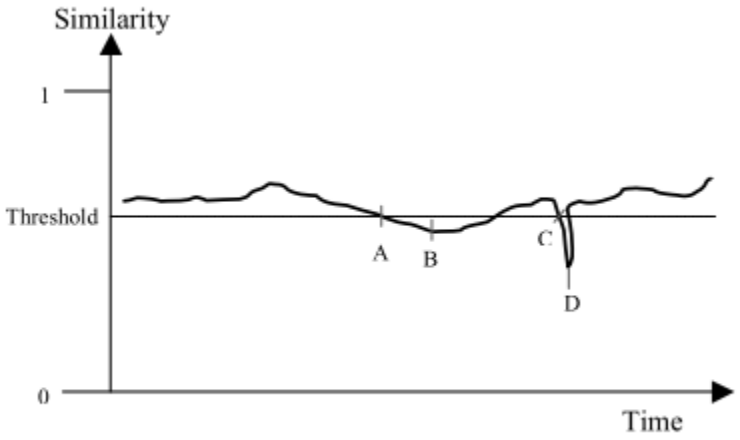


Figure 6 - Gradual vs abrupt change

The activities before point A are considered to be normal and the profile does not need any update. Between points A and B, the patterns represent some behavioral change and the profile needs to be updated. Between points C and D, the patterns represent intrusive behavior and no update is made. The authors suggest using a sliding window technique such that only the monitored actions from day n-1 are considered when updating the model, so that computational efficiency is increased and the old data, which would less represent the current normal behavior, is discarded.

The threshold-based method that was described was used as a host-based detection method and will only detect attacks after a certain period of time, because it will even take some time to distinguish an abrupt change. For host based systems, it is quite usual that an attack will cause anomalous traffic over a certain time period. For example, a system intrusion, virus or worm will usually account for several subsequent anomalous system calls. Having only the ability to detect changes in a less-than-immediate time window may not always be desired behavior. This especially applies to network-based detection systems, where a single anomalous packet, such as one that contains an SQL injection string, can already have substantial effects. In case of such an event, it is therefore usually desired to have the detection system raise an alarm immediately. Therefore, in such cases the threshold that will be used to track gradual, legitimate changes, is different from the threshold that is used to detect immediate changes, which is the anomaly-threshold for which the detection system will raise an alarm whenever the anomaly-score of an incoming data packet exceeds it.

A threshold-based method that makes use of this kind of a separation of threshold values is presented in [25]. Here, the monitored data is divided into two adjoining sliding windows with respect to the time interval. The authors use changepoint detection to detect changes between the data sets that were recorded during both intervals. In changepoint detection, the instances at which the probability distribution of a stochastic process or time series changes is determined. For this purpose, the data in both sliding windows is modeled using SVDD (as described in 4.2.3.2) and the radii of both hyperspheres are compared. When the difference exceeds a predefined threshold, which is different from the default anomaly-threshold, it is concluded that the behavior in the last sliding window contains a legitimate change. Here it is assumed that a sudden burst of large scale attacks causes abrupt changes in the data flow, while concept drift accounts for gradual changes. The width of the sliding window should be adjusted in such a way that it is not too large such that the model creation becomes computationally infeasible, while also being of sufficient size so that abrupt changes, which correspond to attacks, are not considered legitimate changes in normal behavior. Also, the threshold that is used to determine when a certain change accounts for concept drift should be correctly tuned. The authors suggest a per-network procedure based on trial and error in order to determine these parameters.

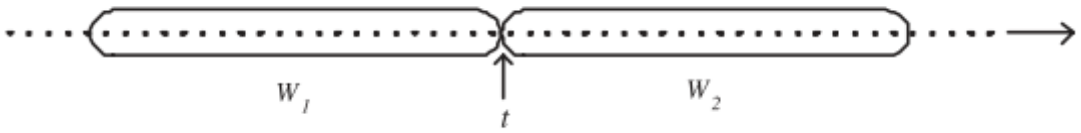


Figure 7 - Sliding window approach

A more extensive way of detecting concept drift from changes in the observed traffic is provided in [31]. Here, a method consisting of a combination of technical analysis tools which are typically used in predicting and verifying financial market data is used to guide the IDS by providing information on when a legitimate change in traffic occurs. The method assumes that the IDS system is able to assign a certain anomaly score to a packet (for anomaly detection the researchers use the statistical properties of the relative byte frequency as a feature, making it possible to assign a score to the statistical deviation). Each packet will be mapped to an anomaly score and with this data a time series graph is constructed that can be used for statistical analysis, specifically to detect changes in the statistical behavior of the anomaly scores. In order to detect changes in a "trend", the authors make a distinction between identifying the point in time when a change has been observed during a short time interval, and identifying the point in time at which this rapid change has evolved to a new trend. The latter analysis that involves the determination of the long-term change will only be initiated as soon as a short-term change has been observed.

In order to define a threshold for the short-term change, first linear regression (LR) is performed on the last n packets. Subsequently, the correlation of the score of each new packet with the value predicted from the LR is tracked. The correlation will measure how well the prediction from the LR matches the actual observed value. It is assumed that for traffic in which no concept drift occurs, in general the scores of incoming packets are not very correlated with the predictions from the LR. This is because the scores will usually be randomly distributed within the anomaly threshold bounds, which are the thresholds that define when the IDS should raise an alert. Because in case of an absence of concept drift the correlation remains low, when a concept drift occurs this will be visible as a rise in the correlation. This is because for multiple subsequent new packets, the score of those packets will be shifted by a certain amount relatively to the scores in the period preceding the concept drift event. The predictions from the LR will now converge towards these shifted scores, making the correlation value rise for some time, which is a situation that is unique for concept drift. Now to identify short-term change, the authors simply suggest to look for a single rise in the correlation. This denotes the event of a "potential upcoming change".

Only identifying this short-term change is not enough to identify a trend break. Confirmation of the break to a new trend is the second part of the algorithm. Here the authors state it is important that the confirmation line is quick and agile; the sooner the trend break is identified and verified, the sooner the system can isolate important packets for retraining and start the retraining process. With respect to determining this long-term trend change, the authors use a combination of two methods. The first method will check whether the rise in correlation from the short-term change will continue during a certain interval. This will show that the

direction of the LR predictions is tracking in the same direction as the incoming scores. As a measure, the percentage of rising during the interval is examined. When this percentage exceeds a certain threshold, this first part of the trend-confirmation algorithm will have determined a potential change in the trend. In order to completely confirm the trend break, we will look at the second part of the confirmation algorithm. This part analyzes the "reverse exponential moving average" (REMA). A moving average (MA) is a series of averages from a data-set over a fixed window size. The exponential moving average (EMA) will assign more importance to recent values. Reversing this property will emphasize the older scores inside the sample window. Slowing down the reaction of the moving average this way gives the ability to smoothly determine changes within the algorithm. Actual attacks and false positives that intermittently occur will have less of an immediate impact. The algorithm will use the running standard deviation as a representation of the boundaries of known scores. In the financial technical analysis world, this is known as the Bollinger Band. By identifying how the score falls relative to this band, changes in score patterns or trends can be tracked. Coupling standard deviation with the slower moving REMA will allow normal score-range trend changes to pass through the system without raising an alarm or triggering the self-update mechanism. Due to the slow nature of REMA, incoming scores that land outside the bounds become points of interest, but have little immediate effect on the REMA line. However, by calculating the percentage of scores over a given sample size that lies inside the standard deviation bounds, it can be determined whether if those scores are moving to a new trend. When only a small percentage of scores lies inside the bounds, it is more likely that concept drift has occurred.

The authors show that it is required to use the conjunction of the verdicts of both parts of the confirmation algorithm in order to be able to accurately define a long-term trend break at a certain point in time. The primary tuning of the algorithm is done by specifying the REMA's window size and the LR's window size to control the reaction speed of the MA and LR. A third window size, called the confirmation window size, is also used. This tunable window size acts as a boundary for the two parameters that are involved in the long-term change analysis; the percentage of rising correlation and the percentage of scores within a standard deviation bounds of the REMA. A shorter window size and more tight bounds on the parameters will cause the algorithm to be less averse in concluding that there was concept drift.

The figure below was taken from [31] and it shows the results for the parameter configuration which performed best in the tests (i.e. with this combination the lowest rate of false positives and the highest rate of true positives was achieved). In the two multi-plots the same configuration of parameters was used, but with a different data-set. The first data-set includes a sharp change in data and the second includes a gradual change. However, both of the data-sets contain actual concept-drift. In the upper part of the multi-plots the x-axis represents the packet number and the y-axis represents the anomaly score. The small black dots are the observed packet scores, the "Upper STD" is the upper standard deviation bound for the REMA and the "Threshold" represents the anomaly score threshold before the IDS will raise its default alarm for the detection of an anomaly. In the lower part of the multi-plots the value of the correlation from the LR with the packet scores is shown. The configuration of parameters involves a large REMA window size, a small LR window size, a rising correlation threshold of 75%, a REMA standard deviation threshold of 90%, and a

confirmation window size of 30 packets. This means that the REMA is less volatile to changes and the LR is more volatile. Also, for 75% of the correlation measurements within the confirmation window size the correlation of packet scores with the LR values has to be larger than the previous correlation and less than 90% of packets should lie within the REMA standard deviation bounds before the algorithm will assume that concept drift has occurred. The blue vertical line indicates the point in time when a short-term change was observed, which is the case when there is a rise in correlation between the LR values and actual scores. This will trigger the long-term change analysis, which terminates after the confirmation window size of 30 packets, represented by the green vertical line. At this point in time the long-term change has either been confirmed, which is the case when within the confirmation window size there is a sufficient rate of rising correlation measurements as well as a sufficient amount of packet scores outside the REMA standard deviation bound, or concept-drift is rejected when these requirements are not met. In this case concept drift was successfully detected by the algorithm.

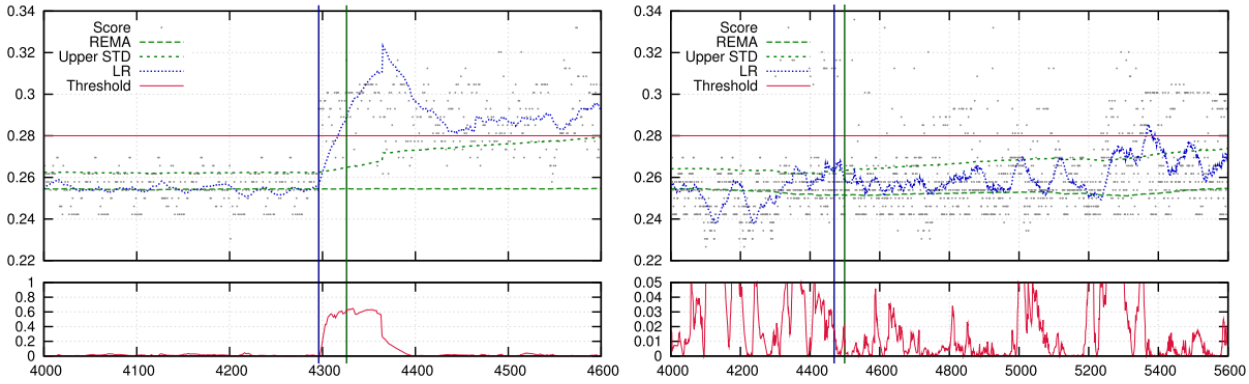


Figure 8 - Identifying trend breaks by using statistical methods

6.1.1.1 A classification feature for concept-drift detection: legitimate clients

In paragraph 4.2.3.1 we already discussed several features that can be used for web-based anomaly detection in order to classify incoming requests. However, certain features can also be used to distinguish concept drift. One such a feature is described in [33], in which a cluster-based modeling technique is used together with a classification technique which also takes the “reliability” of the clients into account when classifying requests to web applications. The idea behind this is that less frequent access patterns may actually be considered “normal” when they originate from clients that have been proven to be non-malicious based on past requests. Note that the proposed method is specifically designed for the purpose of creating an initial model from scratch based on an existing set of training data, such as a database containing a large number of HTTP requests for an application. When using cluster-based techniques to create such models, normally the less frequently occurring patterns would be grouped in relatively small clusters, which are flagged as “anomalous” and discarded from the final model. However, in the method that was proposed

these clusters would not be considered anomalous when they contain a requests that originate from hosts that have a certain level of reliability. This measure is calculated as follows[2].

DEFINITION OF THE VARIABLES USED IN THIS SECTION.

p^j:	popularity index of the j-th cluster
m^j:	total number of requests in the j-th cluster
n^j:	total number of hosts in the j-th cluster
m_i^j:	number of requests from host i in the j-th cluster
q_i^j:	percentage of requests from host i in the j-th cluster
l:	maximum percentage of requests from a single host in a cluster
k:	total number of clusters
w_i:	confidence index of host i
H^j:	set of hosts in the j-th cluster
c^j:	confidence index of the j-th cluster
r^j:	reputation index of the j-th cluster
rd^j:	reputation degree of the j-th cluster

Algoritmo 1 Compute the distribution of requests in the j-th cluster.

$q_i^j = \emptyset$; z= true;
while z do
  z = false
  for i=1; $i \leq n^j$; i = i+1 do
    if $q_i^j > 0$ or $q_j^j = \emptyset$ then $q_i^j = m_i^j / m^j$
    if $q_i^j > l$ then $q_i^j = 0$; $m^j = m^j - m_i^j$; $z = \text{true}$
  end for
end while
return $(q_1^j, \ldots, q_{n_j}^j)$

$$p^j = \frac{1}{n^j} \sum_{i=1}^{n^j} a \quad a = \begin{cases} 0, \text{ if } q_i^j = 0 \\ 1, \text{ if } q_i^j > 0 \end{cases} \tag{2}$$

$$w_i = \frac{v_i}{\sum_{j=1}^{k} p^j \cdot \max\ v_i} \tag{3}$$

$$and: \ v_i = \sum_{j=1}^{k} a \mid a = \begin{cases} 0 \text{if } i \notin H^j \\ p^j \text{if } i \in H^j \text{ and } q_i^j > 0 \end{cases}$$

$$c^j = \frac{1}{n^j} \sum_{i \in H^j} w_i \tag{4}$$

$$r^j = \alpha \cdot\ p\ + \beta \cdot c^j \text{ where : } \alpha + \beta = 4 \tag{5}$$

[2] The variables and descriptions of the algorithms have directly been obtained from [33].

First, the request rate per host is calculated in *Algorithm 1*. In order to mitigate the influence of DoS attacks, a threshold is defined to penalize hosts that have generated an exceptional number of requests. Then in (2) the popularity index of each cluster is calculated; a value between zero and one based on the diversity of the hosts within the cluster. A higher number of different hosts that contribute to the requests in the cluster will increase the popularity of that cluster. After the popularity index of each cluster has been defined, the confidence index of the hosts will be determined in (3). This is done by looking at the popularity of the clusters for which the hosts have any requests. Subsequently, in (4) the confidence index of each cluster is calculated based on the confidence index of the hosts that have requests in the cluster. The final step in the pre-detection phase is to calculate the reputation index of each cluster by means of a weighted sum of the popularity index and the confidence index of the cluster, which is visible in (5).

Note that step (4) and (5), in which the confidence and reputation index of a cluster is calculated, are crucial. An attacker could easily forge a profile with a high confidence index by making sure that he has executed a large number of requests to popular clusters. When one would then merely accept an outlier as legitimate when it is only linked to one request of one legitimate client, an attacker can easily exploit this method. By calculating the reputation based on a combination of the sum of the confidence indexes of the hosts, i.e. the confidence index of the cluster, and the diversity of the hosts, i.e. the popularity of the cluster, with a proper threshold on the reputation index one is able to mitigate such an attack. Note that this threshold should be carefully chosen, as a sophisticated attacker could have control over multiple clients that are considered trustworthy. With a sufficient number of different legitimate clients under his control, the attacker will be able to fool the system into classifying a malicious request as being legitimate. This is very dangerous, because it has a severe impact on the effectiveness of the IDS when this method is feasible for an attacker.

This reputation index of a cluster now represents the legitimacy of the requests in the cluster based on the hosts from which the requests originate; when the hosts themselves have proven to contribute to a sufficient part of legitimate requests within the application, the reputation will be higher. In addition to the standard criteria of cluster size, this reputation index can now also be considered when determining which clusters are outliers and should not be included in the final model. Clusters with a higher reputation index may now be included in the final model, even when the cluster size is below the threshold for that property. This is especially useful for coping with concept drift, because in such a case the clusters that will be formed from new incoming data will initially be small for the drifted traffic pattern and might otherwise be regarded as anomalous and thus creating false positives.

6.1.2 SYSTEMS THAT DETECT LEGITIMATE CHANGES BY OBSERVING THE UNDERLYING SYSTEM

In [29] the authors try to tackle the challenge of self-adaptation by looking at it from a different perspective. In the methods that were described earlier legitimate changes were identified in the monitored environment, which represents either the traffic or the system calls generated by users depending on whether the method was applied for network-based or host-based detection respectively. However, in [29] the underlying cause

of the changes in the monitored environment itself is analyzed, which in this case is the software that is running on a system, which is responsible for changes in the monitored system call behavior when software patches are applied. The authors propose a method in which they use a tool to discover the differences in the possible control flow graph in terms of system calls that can be executed by the program by looking at the binary difference between the source code of two programs, which are in this case the old version of the program and the newer version of that program. It may be clear that this approach can only be used for host-based detection.

In [30] it is also the underlying application that is considered for analysis in order to detect legitimate changes instead of observing user actions. The authors consider three entities that need to be monitored for changes as they are internal factors that determine the behavior of the system: file system, databases and software patches. It is assumed that the changes in these three entities are non-malicious (other security mechanisms might be necessary) and that the monitoring system has direct access to them, and in some cases, can add information to them. For the file system and databases case, the authors introduce a monitoring system that notifies the anomaly detection system of any changes that appear in the two entities. If the anomaly detector system models the data that resides in a system at a granular level, then a section that is changed implies only a small change in the overall model. For patches the analysis is separated from the file systems and databases, because there is a very distinctive difference between the two approaches: for patches the changes are in the code section while for file systems and databases they appear in the data section. The method that is proposed for patching is similar to the approach in [29], in which the source code of the old program and the patched program are compared in changes in their control flow. Although, as in [29], in [30] the examination with respect to source code comparison is limited to the host-based model, the authors suggest that analysis of the impact that patches have on n-gram based network content models is an interesting area for future work.

As was described before, the authors in [30] also monitor changes in the database and file system, in this case specifically for the self-adaptation of an HTTP-based A-NIDS. In their research it is assumed that the granularity of model creation is at least on a web resource (file) basis. Different methods are proposed for static and dynamic web pages. For static web pages, it is proposed that the model of the application maps each possible URL request with some model-representation of the correspondent file that resides in the file system and is returned by the web server. This can, for example, be an md5 hash of the file's contents and path. When changes are made in the file or the file system, only the models of the changed files are altered. If a file is removed than the corresponding model is removed. If a file is changed, its model is updated. Otherwise, if a file is added a new model is built for this file and mapped to the right URL request. For dynamic responses, models corresponding to a particular script file (e.g. index.php) are generated for both the HTTP requests and HTTP replies. The initial models will be created using some training algorithm and when a new request is sent to the web server, it will first be correlated with the SQL query that is generated by the HTTP server, to ensure that non-anomalous information is queried from the SQL server. Another correlation is performed between the SQL reply and the HTTP reply based on the assumption that information returned by the database has to be reflected in the HTTP reply. When new data is introduced to

the file system or to the database, the changes have to be reflected in the two types of models by altering only them accordingly. The researchers speculate that even if the content is dynamically generated there will be common content between pages generated by the same script with different parameters, while part of the different content will be related to the data returned by the database.

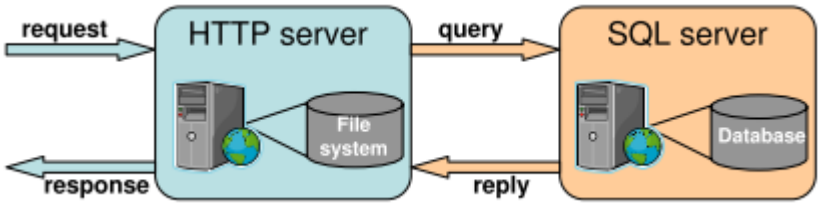


Figure 9 - Environment for detecting changes in the file system and database

For detecting changes in the file-system, the authors use *inotify-tools* on Linux. For the database, a method that uses MySQL triggers in order to detect changes in the database tables is proposed, in which changes are monitored in the table structure as well as in the data that is contained in the tables. However, the empirical study is limited to detecting changes in the file-system for static HTML files and the methods related to monitoring the database traffic were not put to the test.

The research in [30] does not provide any details on how to link the HTTP requests to the SQL queries that are triggered by those requests. For web applications that only have a small number of requests, an option could be to link both actions based on their time of execution. However, especially for more busy services this approach becomes infeasible. In [32] and [36] an approach is used that is applicable to web applications that have a login functionality for users. For each web server session a separate light-weight virtual environment is created and all interaction on a HTTP and SQL level will take place within the container. Now the problem of linking HTTP requests to SQL queries becomes more feasible. A visual representation of the approach is given in the figure below. Note that this type of detection is solely focused on detecting malicious database requests; HTTP query parameter values are normalized by default so as to build a mapping model based on the structures of HTTP requests and SQL queries. Once the malicious user inputs are normalized, the system will not be able to detect attacks hidden in the values, such as XSS attacks.

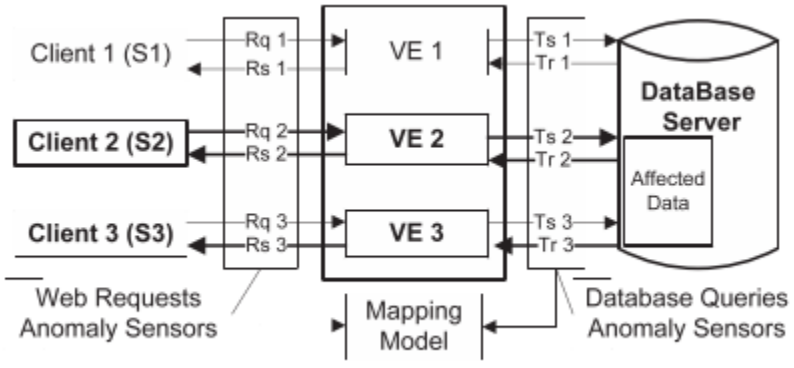


Figure 10 - Detection of malicious database requests by using virtual environments

In [23] a similar approach is used for an HTTP-based A-NIDS. However, instead of observing the differences in the application's source code, file system or database, the HTTP responses that the server generates are analyzed. In case of a web application, the entire structure that will be presented to the user (after it has been parsed and rendered by a browser) will be contained in these responses in programming languages such as HTML, Javascript and CSS. The authors in [23] use a method in which they leverage the fact that for HTML it is quite feasible to identify from this structure those entities which are responsible for the legitimate requests that are sent from the user, such as form fields and hyperlinks. In their method, HTTP responses are scanned for input fields and links by parsing the HTML code in the response message. Assuming that a response is always valid, changes in input fields and links represent valid changes that are to be considered in the model.

6.1.3 RETRAINING FOR THRESHOLD-BASED SYSTEMS

For threshold-based systems the model will be retrained as soon as a legitimate change has been observed in the observed traffic or in the underlying application. Changes in the underlying application will happen instantly and when this is detected there will immediately be sufficient data available with which to update the model, such as when a new form field is discovered in a HTTP response. For concept-drift detection methods that monitor traffic instead of the underlying system, this is somewhat different. When one packet which contains concept-drift is incorporated in the model, it usually has a minor effect. In order to make the model shift towards the concept-drift, a large number of packets with drifted values has to be collected.

In [25] legitimate changes were identified by comparing the differences between the data observed in two subsequent sliding windows, which was described in the previous paragraph. As soon as a legitimate change has been detected, the data in the second sliding window is used to create an entirely new model. This seems to be a more sensible approach than just including random packets that were collected during the complete timespan between retraining periods as was done in [13], because now the retraining data will surely contain the packets that account for concept drift. Note that it may also be needed to continue the retraining with live data for a while, because the sliding window may not contain sufficient packets to assimilate the concept-drift into the model. It may be clear that this depends on the specifics of the modeling technique that is used.

6.2 Naive algorithms: continuous learning based on observed traffic

In addition to systems that incorporate the logic to detect legitimate changes in order to identify suitable points in time when to update the model, there also are systems that contain a less sophisticated self-adaptation method in which the model is continuously updated based on live traffic, possibly filtering out part of the attacks. In [26], all traffic that is not considered anomalous by the IDS with a certain model, created by the SOM and PAYL algorithms, is used to create a new model after some time. In [27], it is shown that some HTTP model creation algorithms outperform others with respect to their self-adaptation

capabilities and the extent to which false positives are reduced with continuous learning. It is concluded that especially DFA outperforms N-gram and Mahalanobis distance algorithms when it comes to the effectiveness of self-learning. In this research the authors update the DFA model from the traffic that was rejected by the current DFA, but exceeds a certain similarity level compared to the current DFA. The authors argue that in general, algorithms that create a more over-generalized model benefit more from regular self-adaptation than algorithms that create a more under-generalized model. This seems plausible, because over-generalized models are less restrictive on the traffic compared to under-generalized models and are therefore less susceptible to false positives when the normal behavior is subject to change. Note that detection systems which employ a procedure of continuous-learning will only adapt to long-term gradual changes, which occur even more gradually than the gradual changes that were described in paragraph 6.1. This is because for the threshold-based systems that were described in that paragraph, it is possible that traffic that exceeds the default anomaly-threshold is also incorporated in the training data that is used to update the model. This makes these systems account for more rigorous changes than the systems that incorporate continuous learning, in which the training data will solely consist out of traffic that lies within the anomaly-threshold.

In [28] an approach to continuous learning is used in which the data is also autonomously sanitized before incorporating it in the final model. At the base of this system lies the ability to self-sanitize the training data. At first, a concept version of the model is created, consisting out of multiple “micro-models” which are based on traffic that has been observed in fixed subsequent time intervals. After that, either the training data that was used to create the micro-models is sanitized using the micro-models, which the authors call “introspection”, or the micro-model testing is applied to a second set of initially available traffic. This sanitization is achieved using a weighted sum of the verdict (anomalous or normal traffic, represented as a 1 or a 0) of each micro-model over all micro-models. The weight for each micro-model is optional and can be based on the number of packets that was used to train a micro-model. After the training data has been sanitized, a final model can be constructed. At the center of this method lies the assumption that attacks and abnormalities are a minority compared to the entire set of training data, certainly for training sets that span a long period of time. Given this notion that each distinct attack will be concentrated in (or around) a certain time period, affecting only a small fraction of the micro-models, the weighted sum method can be used to filter out anomalous traffic. In order to account for concept drift, the authors in [28] propose an approach in which the sequence of micro-models, which essentially acts as a filter that sanitizes the live traffic using “introspection”, a process that was described before, is constantly updated by adding a new micro model based on live traffic and removing the oldest one. The following figure was taken from [28] and visualizes this process:

Figure 11 - Micro-models and introspection

Concept drift appears at different time scales and since the micro-models span a particular period of time, the detection of concept drift is limited to observing the drift that occurs at scales that are larger than the time window that is covered by the micro-models. Any changes that appear inside the time window have a chance of being filtered out in the sanitization phase. The authors also point out that the method cannot distinguish between a legitimate change and a long-lasting attack that slowly pollutes the majority of micro-models.

6.2.1 SEMI-CONTINUOUS LEARNING FROM OBSERVED TRAFFIC: UPDATE THE MODEL AT A CERTAIN TIME INTERVAL

In addition to systems that employ continuous updating of the model, there also are systems that perform an update of the model at a certain time interval. For example, in [26] no method is used to detect legitimate changes. Instead, all processed traffic is flagged as “attack free” based on the current model of the IDS, which in this case consists out of a combination of SOM and PAYL (refer to paragraph 4.2.3.2). After a certain period of time the model is replaced with a new model that is based on the attack free traffic that was captured during the last interval. Therefore, this method only adapts to slight and very gradual changes in normal behavior, because otherwise the traffic would not be labeled as “attack free” by the IDS in the first place, and would therefore not be included in the new model.

The same principle of time-based retraining without a threshold to detect concept-drift is applied in [13]. Here it is suggested that during the retraining stage, several requests from the training set will be replaced with requests received during the detection stage using the first-in-first-out strategy. After that all the variables in the model need to be updated, because they depend on each other. For example, the classification factor in [13] considers the frequency of occurrence of n-grams in the entire training set, which will now include the new data. In addition, they argue that countermeasures are necessary against attackers who try to affect the training set by flooding the web-server with a large number of intrusions, for example by allowing a client to replace a configurable number of HTTP requests in the training set per time slot.

6.3 A combination of continuous and threshold-based learning from observed traffic

Combining continuous as well as threshold-based learning seems like an intuitive approach. Continuous learning will account for minor long-term drift, while threshold-based learning will cope with the short-term changes. In [34] such an approach is employed, in which a cluster based technique is used for creating the model. Updating and retraining the model from real time data (recall the properties of self-updating and self-adaptation from paragraph 4.2.2) is also taken into account. In the proposed framework, the detection model is a set of clusters of normal audit data items. A data item is an object defined to detect whether it is normal or anomalous. A network connection or a HTTP request is an example of a data item. Any incoming data item that deviates much from the current detection model is considered as a suspicious item and suspected to be an attack. A suspicious item can be a variant of normal data items due to concept drift or an attack. To refine

the diagnosis, three states are defined for a data item: normal, suspicious and anomalous. If a suspicious item is identified, it is then put into a reservoir. Otherwise, for normal data the detection model is updated with the current incoming data and for anomalous data an alarm is raised. A suspicious item is considered as real anomalous if it represents a suspicious cluster center or it belongs to a suspicious cluster after the model is rebuilt. This rule can also be interpreted as that "a suspicious item is considered as real anomalous if it is marked as suspicious again after the model is rebuilt".

The detection model is rebuilt as soon as a behavioral change is detected. In order to define a behavioral change, the authors have specified three criteria that trigger the rebuilding process. The model will be rebuilt as soon as any of the following three criteria is met:

- The number of incoming suspicious items exceeds a pre-defined threshold.
- The time window length after the latest clustering exceeds a pre-defined threshold.
- The percentage of suspicious items since the latest clustering exceeds a pre-defined threshold.

The first two criteria indicate gradual change of the detection models while the third describes a sudden change. Large percentage of suspicious items means that there are many suspicious items in a short time and this indicates a sudden change in the audit data. An overview of the detection framework is provided in the figure below, which was taken from [34].

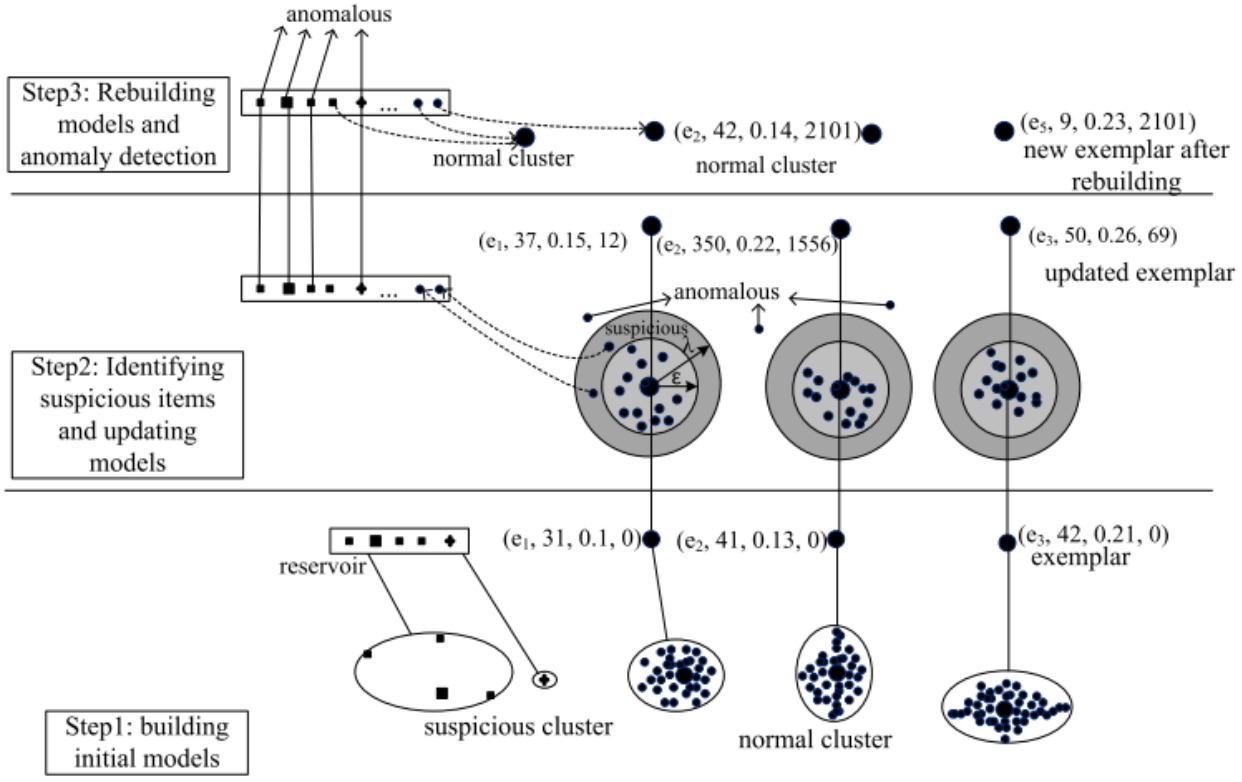


Figure 12 - Retraining approach with clusters and suspicious items

In step 1 the initial clustering takes place. Suspicious items are detected by looking at the size and the sparseness of each cluster. The size of a cluster represents the number of items that are associated to an exemplar, i.e. the data item in the center

of the cluster. The sparseness of the cluster is defined as the mean distance between an exemplar and all its corresponding items. When a cluster is very small or sparse, all the items in the cluster are marked as suspicious. The suspicious items are then put into a reservoir for further investigation. A suspicious item may be regarded as normal or as anomalous later. In step 2 incoming data is observed during real-time detection. When an incoming item is within range ε, it is considered "normal" and the cluster is updated on the fly with this data item. When the item is out of the range ε, but is within the range λ, it is marked as "suspicious" and moved to a separate reservoir. An item that is outside λ will trigger an alarm. When one of the three rebuilding criteria that were described earlier in this report is triggered, step 3 is initiated. Here the clusters are re-built using the current exemplars and the items in the reservoir. Some items may be incorporated in existing legitimate clusters, while others may form their own cluster. These new clusters are classified as either a "normal" or "suspicious" cluster. In case of "suspicious" clusters, all items in the cluster will be seen as anomalous. After processing the reservoir, the normal detection process of step 2 can be resumed. Note that there is no need to pause the detection of step 2; step 3 can be performed in parallel, identifying the anomalous items from the reservoir and "feeding" all other items from the reservoir to the detection model; new clusters would just be added as new clusters, while all other items, which are the items within ε in the existing clusters, would be incorporated in the corresponding existing clusters.

6.3.1 RETRAINING THE CLUSTER-BASED MODEL

We have discussed the clustering-based detection method that was used in [34], which distinguishes "suspicious" behavior in addition to the regular classifications of "normal" and "abnormal" behavior. "Suspicious" behavior is collected in a reservoir, with which the entire model is periodically updated. The modeling technique itself is extensively described in [35]. There it is claimed that the clustering technique on which the proposed method is based, namely DBSCAN, is less suitable for the purpose of detecting concept-drift, because it assumes a static environment and is not suited for real-time updating based on a continuous data stream. The modeling technique is based on Affinity Propagation, which is an algorithm that specifies a set of clusters in which the centers of the clusters represent real data items (they are called "exemplars") and in which the summed distance between the data items in each cluster and the exemplar in each cluster is minimized. A parameter can be set to control the number of different clusters that is created. We will omit the description of all of the underlying mathematics of the method. However, we will briefly handle the way in which the model is updated. The updating consists out of two parts. First, there is the continuous updating from real-time data. This applies to the situation when new data flows in and is flagged as "normal" (the data is within the ε-range). The data will belong to a certain cluster, represented by an exemplar. As the new data item comes in, the following will happen to the exemplar and its properties:

$$e_i = e_i \quad \text{(the exemplar remains the same)}$$
$$n_i = n_i \times \left(\frac{\triangle}{\triangle + (t - t_i)} + \frac{1}{n_i + 1} \right) \quad \text{(the number of items increases)}$$
$$\mu_i = \frac{\mu_i \times n_i + d(e, e_i)}{n_i + 1} \times \left(\frac{\triangle}{\triangle + (t - t_i)} \right) \quad \text{(update the mean distance)}$$
$$t_i = t \quad \text{(update the time when the model is last updated)}$$

The fractional term is named the "forgetting factor" and is used to assign more weight to recent data while reducing the weight of past data in order to stay up to date with a changing environment. n_i is especially important, because it has an effect in the second part of the model-updating mechanism. In chapter 7 we already discussed that the retraining phase of the model would be triggered as soon as one of the rebuilding-

criteria was met. The retraining is performed by using the Affinity Propagation based method on the existing exemplars as well as all the items in the reservoir. Existing exemplars will have an additional preference to be selected as an exemplar in the new model based on n_i. When more data-items were within the ε-range of an exemplar during the period of real-time detection, the exemplar will gain importance, which is expressed in the higher value of n_i.

It may be noted that in fact the model itself is not updated in real-time; the position of each exemplar remains the same and the cluster will not change exemplars; only the properties of the cluster, which is represented by its exemplar, are updated. Also, the incoming items in the real-time data are immediately discarded as soon as the cluster's properties have been updated. Another method would be to update the (position of) the exemplar itself. In [35] it is also suggested that "normal" items could be stored and used with the retraining to increase accuracy. A significant drift can be caught by the change detection and rebuilding method which is triggered by one of the rebuilding criteria. A minor drift within ε could then be coped with by retraining the current exemplar as well as adapting the corresponding model parameters. This means that an exemplar may be subject to a change in position or may even be entirely replaced by another exemplar before the "major" retraining takes place, which incorporates the items in the reservoir. The notion that this real-time updating of exemplars in addition to the periodic update that incorporates the reservoir is not required to maintain a high detection accuracy is proven in [35]. A visual representation of this principle is in the figure below.

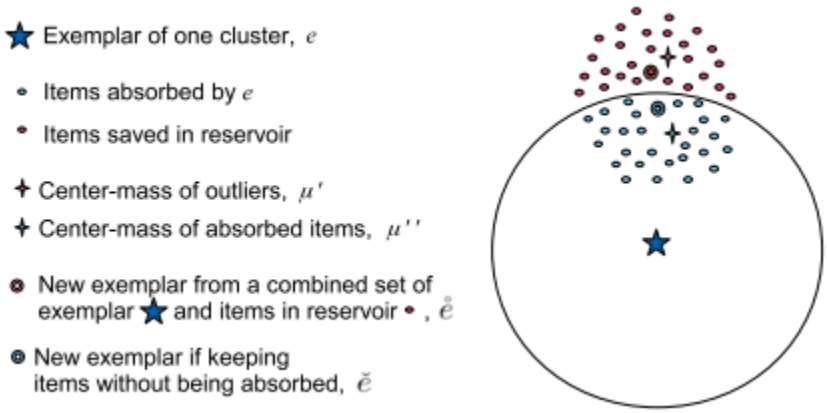


Figure 13 - Classification of legitimate items after clustering

One option would be to retain incoming data items, and frequently update the model to cope with minor drifts in addition to the full-fledged reservoir-based retraining. The new exemplar to which the current exemplar would drift in such a case is displayed as the larger blue circle. It is computationally less efficient to store all incoming data items and perform frequent retraining to cope with those minor drifts. A more efficient method would be to discard all incoming items after updating only some of the exemplar's properties, such as the number of items associated to the exemplar, which will later have an effect in the reservoir-based retraining. The new exemplar that would then be picked after the reservoir-based retraining for this example situation is displayed as the larger red circle. Although the blue circle exemplar would be optimal with respect to the adaptation to the live environment, the fact that, in general, the blue circle and red circle exemplars are not significantly far

apart is proven in [35]. Therefore, it is desirable to use the computationally more efficient method, which discards incoming items immediately after they have been processed, and which is the method that is used in [34].

6.4 General remarks on retraining

In paragraph 4.2 we noted that because the monitored environment is often subject to change, the model should also adapt to this change. In this chapter we have seen several approaches in which the model is updated or retrained in order to account for concept-drift. We have made a distinction between continuous learning and threshold based systems. This separation is also used in [30], where a distinction is made between complete retraining, gradual retraining and spot retraining. Gradual retraining represents our notion of continuous learning, where new data is continuously incorporated into the model as a way to adapt gradually to legitimate changes in the environment. Spot retraining makes the process of retraining more efficient; the model is updated only when necessary by identifying when a legitimate change occurs. This is similar to the threshold-based learning we described. In addition, the complete retraining approach represents a method in which the entire model is rebuilt. For this approach, the system may be re-learning data that was already considered "normal" and this approach can therefore be considered to be less efficient than the other two. Complete retraining has similarities with our notion of semi-continuous learning which was described in paragraph 6.2.1.

In addition to this higher level distinction for retraining methods, which applies to the entirety of the model, we can also look at the structure of the model itself, which can be composed out of several sub-models. For these sub-models the decision whether either the existing model is re-used and is retrained based on newly observed legitimate behavior, or an entire new model is created, is independent from the retraining approach that is used on the higher level. For example, when continuous learning is applied to the entire model, some of its sub-models may require complete re-training. Whether sub-models require complete retraining or otherwise can retain some of the historic data largely depends on the level of granularity that is considered for the observed environment with respect to the sub-models. As an example, consider the HTTP requests for a website as the subject of monitoring. We can either decide to use a low level of granularity and create one model for all of the possible requests, use a moderate level of granularity and create a model for each requested web resource, or use a high level of granularity and create a model for each parameter in the request for each web resource. Although there are many aspects involved in the adaptation of models when it comes to legitimate changes in the monitored environment, for now simply assume that in the described situation there is a legitimate change in a certain parameter value and the underlying model should be updated based on this observation. In this case, with a low level of granularity it would not be efficient to create an entire new model for all the possible web requests, when the change only applies to a parameter of a certain web resource. In such cases, when the model creation algorithm allows for it, retraining the existing model would take much less time than creating a new model. However, with the higher level of granularity, for which a model exists for each individual parameter, creating a completely new model will usually take less time and may improve the accuracy of the model. The improved accuracy that is achieved by replacing the existing model could then be the determining factor to choose for this option. Whether the model is updated

or replaced will also determine the amount of historic information that will be lost. Replacing a model and completely eliminating the previous pattern can result in having to constantly retrain the system in a case where cyclical traffic pattern changes occur. On the other hand, incorporating every observed traffic pattern can lead to an oversize database that suffers from a higher false negative rate since its pattern database is too general. It is therefore important to keep in mind that the level of granularity of the models has an effect on the optimal retraining behavior.

6.4.1 ALGORITHMS SUITABLE FOR REAL-TIME UPDATING

In this chapter we have already discussed some approaches that use continuous learning based on live traffic. Especially for these situations it is important that the algorithm is capable of updating on-the-fly. We have seen that there are several modeling techniques that are capable of real-time updating, such as the cluster-based methods (DBSCAN or the method described in paragraph 6.3), self-organizing maps with self-adaptive extensions (such as the GHSOM) and SVDD. Although there is a lack of research on comparisons between these methods, they can be considered state-of-the-art in modeling algorithms, outperforming the more static algorithms like the SOM, SVM and k-means with respect to their autonomicity for self-adaptation.

6.4.2 OBSERVING TRAFFIC VERSUS OBSERVING THE UNDERLYING SYSTEM

We have also observed a difference between systems that detect changes in the underlying system and systems that detect changes in observed traffic. Re-generating or retraining an existing model based on observed traffic in the monitored environment is time consuming, costly and attack-prone. This is a benefit of detecting changes in the underlying system; the model can directly be updated without any training phase, because changes are directly known. However, detecting changes in the underlying system is usually considered to be a complex task. On the other hand, the delay that is caused before concept drift is detected in live traffic and incorporated in the model can be a real issue, especially when the concept drift is abrupt and the packet scores occur above the anomaly-detection threshold, such that false positives are generated.

6.4.3 DETERMINING THE STABILIZATION POINT AFTER CONCEPT-DRIFT

For some retraining approaches that do not employ continuous learning, in addition to determining the point in time when concept drift is occurring in observed traffic, it may also be desirable to identify the point in time when the concept drift has stopped and the input is stable again. As we have seen in this chapter, for systems that identify concept drift by observing live traffic, this process takes some time, as in all of the methods it requires several packets with drifted values before concept drift will be assumed. When concept drift has then been determined, usually the packets in the sliding window before that point in time will be used as the data for retraining. However, it may also be useful to incorporate new incoming packets in that specific training set or in a new training set and then to perform a second retraining. This is because the concept drift may still be going on. Consider, for example, the situation when the granularity of the models is on an HTTP request parameter-specific level and one parameter value that is modeled as being either the integer zero or one. During the process of live detection, suddenly different numbers and alphabetical characters are occurring as values for this parameter. After several packets concept drift is assumed and the

model is retrained with the data that has been observed up until that point in time. After this retraining live detection will resume while the complete representation of the concept drift, which could also have introduced a certain other characters in the value, has not yet been observed. This may be observed soon after the retraining, although these changes may not be sufficient to let the system assume another instance of concept drift. In this case the model has not achieved the optimal accuracy. It may therefore be desirable to detect when the process of concept drift has stabilized. An example of such an approach is given in [28]. There a method is proposed in which that estimates the likelihood of the system seeing new n-grams, and therefore new content, in the immediate future based on the characteristics of the traffic seen so far. The calculation for this property makes use of the rate of unique n-grams that is observed. Now the system will know the point in time that represents the upper time-bound for which to incorporate packets in the training data, with which it is ensured that the micro-models are well-trained is based on the rate at which new content appears in the training data.

7. PROPOSED METHOD

In this chapter we will propose a self-adapting IDS for HTTP anomaly detection. We have seen that identifying concept drift as well as retraining the model are important aspects when it comes to the self-adaptation capabilities of an IDS. In paragraph 6.1.2 we have discussed systems that are able to predict changes in traffic by analyzing changes in the underlying system and in paragraph 6.4.2 we have named the advantage of such an approach, which is that the model can instantly be updated, so that there is no delay during which false positives may occur. However, there are several limitations to this approach, especially for web applications.

7.1 Limitations of detecting concept-drift from changes in the underlying system

One approach was to infer the changes in the range of legitimate user actions from changes in the source code of programs. However, this approach was focused on host-based systems. For web applications the possible behavior is much more ambiguous and possibly infeasible to derive from the source code. Another approach was to analyze the HTML in web responses. A major limitation of this existing method is that Javascript will not be considered, as it is too hard and it may even be infeasible to infer from Javascript code which elements can be created on the fly, which GET/POST requests can be executed on the fly and which web resources are linked to those requests. For one, there are many different Javascript frameworks which have a different syntax for creating such entities. Although with respect to the dynamic creation of elements and links you might be able to derive patterns from the native code, which for example can be obtained by using a tool such as Google's V8 [39], it still remains hard to link these entities to web resources, which is required for our model. For example, we may observe that a certain Javascript function creates an input element, but we may not be able to infer the web resource that will be requested when the data that is

entered in the field by a user is sent to the server. Therefore, for dynamic applications that heavily rely on Javascript with respect to dynamically created content, the HTML parsing option is quite limited. As more and more applications make use of these Javascript functionalities, we conclude that the existing approach is too limited to incorporate it in our system.

An extension could be to trigger valid requests by observing and navigating through the web application from the outside, as to reproduce the environment in which a valid user operates and which therefore represents the environment from which valid requests will be generated. This can be done by crawling the web application. There are existing HTML and AJAX crawling tools that could prove to be of value here, such as Crawljax [41]. However, these techniques often require an authorized session in order to replicate a user that is logged in. Now when (random) elements are clicked or forms are submitted, this may cause changes in the database, while changes in the database should, in most cases, only be able to be triggered by human users. Also, there may be pre-conditions related to the session that affect the set of possible states, such as the fact that different users may (partly) be served by a different application based on their properties. For example, a user with a certain privilege level in the application may be able to access some parts in the application to which other users do not have access, indicating that there will be differences in the set of all possible DOM states for these users. A possible solution could be to execute Crawljax for all existing users. However, for more advanced applications, identifying the total set of states may become infeasible. There can be several pre-conditions leading to a certain state, such as a certain sequence of elements that the user clicks on, or a certain combination of input in a form. For example, there may be a multi-step form, where the second form is only displayed when certain values have been entered in the first form. Trying out all possible values in the first form is often infeasible and although Crawljax gives you the possibility to manually specify these properties, such as values that are enter in a form, this set of specifications quickly becomes so large that is infeasible to manage. Moreover, when it is required that the system administrator adjusts these specifications, this is contradictory with our initial goal of automated self-adaptation of the detection system.

Considering these difficulties in detecting concept-drift from analyzing the underlying system, we will refrain from using this kind of concept-drift detection.

7.2 The setup of the self-adapting IDS

Based on the analysis up until now, we have observed several aspects that compose an IDS that has self-adaptation capabilities:

- The level of granularity that is used in the model.
- The feature extraction method that is used to reduce the dimensionality of the data.
- The method that is used to model the extracted features.
- The retraining method (time-based, continuous, threshold-based).

We will now handle each of these parts in turn and we will aim to create a setup that is optimal for self-adaptation based on the literary research and our analysis in the previous chapters. After this we will put the

system to the test and report the results in the next chapter. We will then be able to provide a sufficiently substantiated answer to the research questions from chapter 5.

7.2.1 THE LEVEL OF GRANULARITY THAT IS USED IN THE MODEL

In paragraph 6.4 we have discussed that levels of granularity that are commonly used for modeling HTTP requests are either the complete range of possible URI's, the complete parameter string per web resource or parameter-specific models per web resource. We have discussed that the decision on whether to retrain or replace a model is based on this level of granularity. Modeling the complete range of possible URI's is the lowest level of granularity and creates an over-generalized model. In paragraph 6.2 we have seen that this approach benefits more from regular self-adaptation than algorithms that create a more under-generalized model. However, over-generalized models are less restrictive on the traffic compared to under-generalized models and therefore there is a risk that the model is less effective in detecting attacks. Consider, for example, a web application that has many different web pages and input fields. When there is a total of 500 input fields, 499 of which are alphanumeric characters and one that can also contain non-alphanumeric, the low granularity model will allow special characters to be submitted in every input field. With a high level of granularity, the model for the single input with which non-alphanumeric characters can be submitted, such as an input field for an email address, could consist out of a regular expression, while for the other fields only alphanumeric characters are allowed. Now it is possible to be much more restrictive on the parameter values. This is especially useful for restricting the use of non-alphanumeric characters, which is important because the presence of these characters can indicate the presence of an attack payload [15]. Another benefit of using models with a low granularity, especially on a parameter-specific level, was pointed out in in the beginning of paragraph 4.2, where we discussed that in some situations, such as for a bank, it may be interesting to know whether an anomalous value was entered a certain field, independent of whether this value would include non-alphanumeric characters. Looking at the granularity of the model from this perspective, we argue that by using a lower granularity one is better able to be restrictive on non-alphanumeric characters and thus prevent attacks as well as create context with respect to the web application. Therefore, we have chosen to use a modeling granularity in which models are created for every parameter for every web resource. However, as was described earlier, using such an under-generalizing method may result in having to retrain the models. We argue that for our setup this is not problematic, as we will see later in this chapter.

7.2.2 THE FEATURE EXTRACTION METHOD THAT IS USED TO REDUCE THE DIMENSIONALITY OF THE DATA

In paragraph 4.2.3.1 we handled several feature extraction methods which are generally used for HTTP anomaly detection. We conclude that there is no direct link between the methods that were described there and the self-adaptation capability of an IDS. However, in 6.1.1.1 one feature was described that can indeed be used to improve the process of identifying concept drift, and thus the self-adaptation capability of a threshold-based system. We will therefore also incorporate this method in our system. It should be noted

that this feature should serve as an additional and not a stand-alone metric. Purely classifying packets based on whether they originate from trusted clients would consider all traffic from other clients anomalous. Therefore, it is one of the general feature extraction methods that we will use as a basis.

Before we select this method, it is important that we first point out that we will make a higher level distinction between two types of parameter values, using an approach we have adapted from [15]. Based on the notion that attack payloads usually contain non-alphanumeric characters, a distinction is made between values that contain only a few non-alphanumeric characters, called “regular” parameters, and values that contain more instances of these characters, called “irregular” parameters. This distinction is made because for regular parameters it is easy to be very restrictive on the non-alphanumeric characters, improving detection accuracy and context, as was described in the previous paragraph. Being very restrictive here means that, for example, values can be specified to have a certain amount of alphanumerical characters, while the non-alphanumerical characters are specifically defined on a per-character basis. As features we will therefore select those that are used in [15], where the value is divided in groups of alphanumeric characters separated by sets of zero or more non-alphanumeric characters. To cope with the “irregular” parameters, we will analyze the character (byte) distribution, a feature extraction method that is often used for dimensionality reduction of HTTP request parameter values [15][31][34].

7.2.3 THE MODELING METHOD

In chapter 6 we have made a distinction between threshold-based systems and systems that continuously learn based on live data. Both approaches have different advantages compared to one another. Continuously learning systems immediately incorporate the incoming data into the model, making the model adapt to gradual changes in real-time. For threshold-based systems gradual changes make the model drift away from the actual representation of the environment and after a certain delay a potentially costly retraining process is started. On the other hand, continuously incorporating live data into the model also results in a penalty on efficiency. With one of our research questions we aimed to discover whether some modeling algorithms would be more preferable than others when it comes to the self-update method that we propose. We argue based on the analysis in this report that the algorithm should at least be capable of autonomous training, updating the old model with the features that are extracted from new data. One of the requirements for this is that the algorithm does not have any parameters that should be configured manually and that relate to properties of the training data that are unknown beforehand. One such a modeling technique is the basic SOM, which requires parameters that are set based on a trial and error process that iterates through the training data. In order to support continuous updating this behavior is undesirable because it is less efficient. However, nearly all of the state-of-the-art modeling techniques that were named in this report do not have this limitation or could otherwise be adapted easily to allow for continuous retraining. There is a small advantage when using cluster based models, because the clusters can be visualized, which creates a representation of the relevant data which is understandable from the point of view of a human operator and this can aid in activities that optimize the system, such as determining the concept-drift and anomaly-thresholds.

A quite intuitive approach that combines the advantages of both the continuous updating and threshold-based updating is the clustering technique that was described in paragraph 6.3 and more extensively in [34] and [35]. At the base of this technique lies the capability of being able to autonomously account for concept-drift and it is this method that we will use to model the "irregular" parameters, which were defined in the previous paragraph.

As an extension we will use the feature of legitimate clients that we described [33] and in paragraph 6.1.1.1. Here the reputation for clusters was used to prevent legitimate outliers from being filtered out of a set of training data that was modeled using a cluster-based approach. We will use a slightly modified version of this method for detecting legitimate changes in live data streams. We propose a system in which the observed outliers that are out of the range of the anomaly threshold will not raise alarms when they originate from legitimate clients. Instead, these outliers will be added to the reservoir such that the model will be retrained based on these changes. In [34] it was already mentioned as a limitation of the clustering based method that the current detection accuracy still needs to be improved and one solution would be to use more features of the data. As was said before, our unique proposal is to use the feature of the reputation index of outliers, which directly contributes to the adaptation to concept-drift in live data. In short, where items (in live detection) and clusters (in retraining) would be considered to be anomalous in [34], we could consider them to be valid based on the reputation index. The reputation index will be calculated per cluster after retraining. In addition to only storing the suspicious items in the "reservoir", as in [34], we also store the anomalous items during live detection and flag them as "suspicious-anomalous". We also refrain from directly triggering an anomaly-alert when observing a suspicious-anomalous item, but instead issue a low priority suspicious-anomaly alert, while the retraining process will be responsible for triggering a high priority anomaly alert for clusters that are considered to be anomalous after retraining. During retraining we treat the items that have exceeded the *anomaly* threshold differently from items that only exceeded the *suspicious* threshold. For items that have only exceeded the *suspicious* threshold, as a basis we have adopted the same approach as in [34], by checking the size and sparseness of each cluster. Our extension is that when the cluster fails to meet one of these criteria, a sufficient reputation index of the cluster will still result in the cluster to be considered valid. Note that a sufficient reputation index is not a requirement. When the size and sparseness criteria are met, the cluster is considered to be valid regardless of whether the reputation index is sufficient. When it comes to *suspicious* items, our extension is therefore similar to the original method, but less strict towards clusters that would otherwise considered to be anomalous. On the other hand, for items that have exceeded the *anomaly* threshold, in addition to the size and sparseness criteria, a sufficient reputation of the cluster is a *requirement*. This makes the model stricter towards including *suspicious-anomalous* items compared to including regular *suspicious* items.

Note that we only regard it as an anomaly when a *suspicious(-anomalous) item* becomes an *exemplar of a suspicious cluster*. Because clusters may become sparse when non-suspicious existing exemplars can be complemented with infinitely many suspicious items, sparseness has to be regulated in the AP algorithm. In short, we want to make sure that clusters do not become very sparse during rebuilding, which implies that

we have to ensure that the AP algorithm is sufficiently strict towards including outliers, i.e. the suspicious items, in existing “valid” clusters.

For the “regular” parameters we use a more simplistic approach. It can be deduced from the extracted features we mentioned for this type of parameters that these can well be represented in the form of regular expressions, an approach that has directly been adapted from [15]. Details regarding the retraining of this model will be described in the next paragraph.

7.2.4 THE METHODS TO DETECT LEGITIMATE CHANGE AND RETRAIN THE MODEL

With our selected level of granularity, in which the models separately model the web resource and each of the GET and POST parameter values, the following legitimate changes that are caused in web requests are of interest, because they may generate false positives:

- There is a new web resource with corresponding GET and/or POST parameters.
- There is a new GET or POST parameter for a known web resource.
- A known GET or POST parameter value for a known web resource changes.

In the first two situations, there will be no model available. The system will therefore have to decide whether to create a new model for the unknown web resource and/or parameter(s), which can be done based on whether the request is deemed valid. In this report we have described several methods to detect legitimate change. Some of these, such as the sliding-window approaches that were described in the beginning of paragraph 6.1, track the rate of change of observed traffic compared with the existing model, an approach that clearly cannot be applied in the described situations because there is no model available. In addition, there are the methods that detect changes in the underlying system. In the beginning of this chapter we have pointed out the limitations of these systems, especially when it comes to web applications, and therefore we have omitted these methods from our system. However, an approach that can be used for our system is the one that uses the concept of legitimate clients, which was described in paragraph 6.1.1.1 and extensively in [33]. However, there the method is used on an existing set of training data, which has been modeled using a cluster-based approach. We will work with live data and as was mentioned in paragraph 6.1.1.1, to make the system more resistant against attackers the calculation of the reputation index of a cluster includes the number of clients that performed requests in the cluster, as well as the confidence index of each of these clients. When we would use a reputation threshold that is sufficient to withstand most attackers, we would have to evaluate more than just one request for a cluster in order to be able reach a reputation that exceeds the threshold, after which the cluster will be seen as legitimate.

We therefore introduce an “alert delay”, which is a period after the first observation of a deviating entity, the entity being either a web resource, parameter name or parameter value. During this delay there may be more requests to this entity, of which the data is stored. At the end of the period this data is evaluated for each deviating entity. The data that has been collected during this period provides a more solid foundation to calculate the reputation index of the new entity, which eventually determines whether the new entities can be included in the model or that an alert should be generated. This decision is based on whether the

reputation index exceeds a certain threshold. In that case the changes are considered legitimate. By increasing the threshold, the attacker will have to control a larger set of legitimate clients, which is less likely to occur and will therefore increase the robustness of the concept-drift detection. On the other hand, increasing the threshold will increase the delay before the system is updated and therefore increases the number of false positives of the system. In order to mitigate this limitation, an additional feature of the system will be that alarms get labeled with a low priority when at least one legitimate client has requested the new entities. Finally, in order to determine the threshold, it should be taken into account that for some websites the probability that a certain connection is from an unknown and thus less trustworthy client is higher than for other websites. For instance, a new client accessing a well-known World Wide Web portal is less surprising than a new client accessing a particular internal machine that is typically accessed only by a handful of trusted client machines. Therefore, the function that determines the trustworthiness of a client based on the number of past requests should also consider this "surprise factor" [40].

This approach can therefore be used to determine whether a new model should be created for a web resource or parameter name. For the situation in which a known GET or POST parameter value for a known web resource changes, we propose different approaches for the "regular" and "irregular" parameters. For "irregular" parameters we earlier decided to use the cluster-based approach as a modeling method, which incorporates the concept of continuous learning with threshold-based learning. We will stick to an identical retraining approach as the one that is used for this cluster-based method in [34] and that was extensively described in paragraph 6.3.1. This means that we will discard all incoming data after updating only some of the exemplar's properties based on this data, such as the number of items associated to the exemplar, which will later have an effect in the periodic threshold-based (reservoir-based) retraining. This updating from real-time data will then act as the continuous learning, although it is less rigorous compared to a situation where the exemplars would continuously drift positional-wise. This drift now only occurs during the threshold-based retraining, which takes place every now and then. In the previous paragraph we have also proposed an extension to this method, using the concept of legitimate clients to account for changes that occur outside the anomaly threshold.

For "regular" parameters we decided to make use of regular expressions to model the values. The system will be created in such a way that the regular expressions can be updated on the fly based on new training data. Again using the concept of legitimate clients in order to detect legitimate changes, we will incorporate the legitimate changes in the regular expression as soon as they have been detected, by feeding the new values to the part of the system that updates the regular expression. In order to maintain accuracy of the model, we want to be able to detect whether the previous model still applies, because the newly observed parameter values will have made the model less restrictive and we want the regular expression to be as restrictive as possible in order to have an optimal detection rate. Therefore, the system will also analyze all new traffic with the old model. When the old model will frequently flag the traffic as anomalous, that model will be considered to be outdated and the complete model is re-created based on all of the observed values after the point in time when legitimate change was detected.

Finally, we want to make a small outline of how we have used and adapted the trusted-client approach from [33] to be able to cope with live data, specifically about the calculations that are required to determine the reputation index of a cluster. Note that in our system, a "cluster" is represented by the requests for a newly observed web resource, parameter name or changing parameter value that were observed during the "alert delay". Recall the steps described in paragraph 6.1.1.1 that are required to calculate the reputation index of a cluster:

1. Calculate the popularity index per cluster
2. Calculate the confidence index per client
3. Calculate the confidence index per cluster
4. Calculate the reputation index per cluster

In [33], all steps apply to the same set of training data. In our system however, we have to deal with a set of existing models and live incoming data. We want to base the confidence index of a client on all the requests that have been applicable to the existing models: a client is more reliable when it has a track record of many valid requests to the application over a long period of time. We don't want to determine the confidence index of the clients solely based on the requests that are related to a newly observed entity, i.e. during the "alert delay". Therefore, we do not execute all four steps directly. The system will periodically execute step 1 and 2 to update the confidence indexes of all known clients based on (aggregates of) all requests that have been performed over the past *per web resource*. At the end of the "alert delay", that occurs as soon as a new entity has been observed, steps 3 and 4 will be executed *per entity* (i.e. web resource, parameter name, regular/irregular parameter value group) based on the requests that were received during the "alert delay", as well as based on the known confidence indexes that have already been calculated beforehand in step 1 and 2.

This is somewhat different from the approach in [33], but this imposes no problems because for step 3 it actually does not matter on which the client confidence indexes were based. We are free to calculate the client confidence indexes in any way, but we have chosen to stick to the algorithms for step 1 and 2 from [33]. We consider the requests on a *per web resource* basis, because we argue that there is no additional benefit in looking at requests per parameter name/value. However, there is one small caveat that is applicable in step 4 of the calculation and which is caused by our separation of steps. Recall that the popularity index of a cluster/entity represents the diversity of the clients that have requested the entity. As in [33], in step 4 the reputation is derived from a weighted sum of the popularity and the confidence index of the entity:

$$r^j = \alpha \cdot \ p \ + \beta \cdot c^j \ \text{where} : \alpha + \beta = 4$$

Because in this calculation the popularity p applies to the same entity as the entity for which the reputation index is calculated, we would have to calculate the popularity of the new entity based on the requests during the "alert delay", which is different from [33] where the popularity from step 1 is used. This difference is a result of separating the execution of the first two steps, which is necessary for reasons described earlier. In

step 1 the popularity of each existing web resource is calculated by using a threshold that filters out clients that have more than a certain percentage of requests, as a way to counteract the effect of DoS attacks. When calculating the popularity of the new entity in step 4, we need a different threshold, because the average diversity of clients over the requests for newly observed entities during the "alert delay" will be much lower than the diversity of clients for the complete history of requests for all existing entities, which is the data that is used in step 1. This makes it preferable to be less strict in filtering out clients with high request ratios. Therefore, in our implementation, a much higher threshold is used for the calculation of the popularity in step 4, more precisely 0.5 × (1 + threshold).

In addition, because [33] only makes use of training data and our calculation of the client confidence indexes is based on all requests that have ever been performed for the application, we filter out the large number of "uninteresting" clients and web resources in the confidence index calculation (step 1 and 2) by introducing a threshold in order to improve the efficiency of the system. As we stated before, we have the freedom to adapt the client confidence index calculation and this threshold is similar to the threshold that filters out DoS attacks, but in this case the clients that have too few requests, such as automated scanners, are filtered out. But the filter is especially useful for the large number of clients that would otherwise have very low confidence indexes and that will now be filtered out. The reputation index calculation is only slightly affected by this because the missing confidence indexes will be considered to be zero in step 3. However, otherwise they would be close to zero, which is not much of a difference given that the threshold is not too high. In addition, the reputation threshold itself can be adjusted. The reputation indexes may in general be slightly lower because of the filter we introduced, but the reputation threshold can be adjusted in such a way that the system still has the same accuracy.

7.3 Points of attention when it comes to the initial training

For attribute values it is often necessary to have a training phase during which a large number of requests is monitored before a model can be constructed that sufficiently represents normal traffic, i.e. a model that not generates an unacceptable number of false positives. When initiating a training phase using live traffic it may be the case that attacks are included in that traffic. For the clustering algorithm, which is in the core of the machine learning part of the system that is used to handle the "irregular" parameters, this does not have to be problematic, as long as the part of the traffic that consists of attacks is relatively low. This is because the parameters of the clustering algorithm can be tweaked so that small clusters further away from the center will be excluded from the final model. In order to ensure that the percentage of attacks in the training data does not exceed a certain threshold, one option may be to filter the traffic through a signature based IDS or Web Application Firewall beforehand.

For the regular expression part of the system attacks in the training set are dangerous, because they will directly affect the final model. Every incoming string will be processed and merged with the existing regular expression such that the new regular expression will also match the new value, which can be an attack. In order to prevent this, the training data can first be filtered using the machine learning part, which in its turn

may also use a signature based IDS or WAF to filter the data beforehand. After this filtering we can be quite certain that the training data will not include attacks, so we can safely process it in the regular expression system. Note that the purpose of this part of the system is to have a resource efficient, but mostly a user-friendly way (given the regular expressions which are visible and modifiable for the user) of detecting malicious traffic.

What remains is that the system should be able to determine when sufficient traffic has been captured in order to ascertain that the model is complete, in the sense that it will not generate an unacceptable number of false positives. This can be done by specifying an upper limit on the rate of change of the model over a certain amount of traffic that is processed during the training phase. For example, it can be specified that the model is complete as soon as 100 new requests will not change the model more than 5%. A detailed measure of the rate of change depends on whether this is measured for the regular expression system or the clustering system. For regular expressions it can be measured by difference in strings (regular expressions) and for the machine learning part the set of clusters and the clusters themselves have certain properties that can be compared, such as the distance between clusters and the reachability between points within clusters.

7.4 Summary of the implementation – Scandax

Based on the ideas that were outlined in the previous paragraphs of this chapter, we have created a system which we named “Scandax”. Summarizing the information from the previous paragraphs, the system could be seen as a combination of methods from Bolzoni [15], Pereira et al. [33], and Wang et al. [34], together with our own additions and adjustments that we have created in order to “glue” these methods together into one coherent system.

From Bolzoni [15] we have adopted the differentiation between *regular* and *irregular* parameters, as well as the model-granularity on a parameter level. We assume that in [15] the parameter models were stored per web resource per parameter name per request method. However, in [15] we could not find information about how a request with an unknown web resource or parameter name is handled. We assume that this would just raise an anomaly alert and based on this assumption we “whitelist” the known web resources and parameter names in our system.

For *irregular* parameters we have used the *state-of-the-art* clustering method from Wang et al [34], including continuous and threshold-based updates of the model. Based on this paper have also applied the concept of *suspicious* items to unknown web resources, parameter names and *regular* parameter values, as well as the criteria to initiate the evaluation of the *suspicious* items. Some modifications were made in the criteria in order for us to be able to apply it to the aforementioned entities. These modifications are described in detail in paragraph 7.4.4.

The major part of our own contribution is the trusted client approach, the foundation of which was adopted from Pereira et al. [33]. We have used the reputation of the collections of requests for *suspicious* items as a criterion in the validation of these *suspicious* items.

In order to substantiate the effectiveness of this approach, we will put it to the test to analyze its performance in chapter 8. The remaining part of the current chapter and paragraph include a high level overview of the core parts of Scandax. The overview is mainly based on the flowcharts in Appendix B. The different items in the flowcharts have been numbered and those numbers will be referred to by using brackets around the number that is referred to. We will also refer to the different variables that are used in the system, the descriptions of which can be found in Appendix C. For variables we use a notation in the form of _A_, _B_, _C_, etc.

7.4.1 MAIN SYSTEM - APPENDIX B.1

This is the core part of the system, which is responsible for evaluating live incoming data. Incoming HTTP traffic is sniffed and parsed in (1) and (2) and a suitable data-structure is constructed in (3). Certain features, such as the requested web resource, requesting client, time, and parameter values for the HTTP request are stored in a MySQL database in (4). This data is stored, because it will be used later by other parts of the system that run asynchronously. However, the main system also acts on an incoming request directly. What kind of action the system will undertake first depends on whether the system is in *training mode*, which is evaluated in (5).

When in *training mode*, the system is in a state in which it will create models for everything that passes by. The requested web resource is immediately whitelisted in (6), as well as each requested parameter name (GET as well as POST) in (7). Note that when we refer to parameter names, the actual representation in the system is a combination of the request method together with the parameter name, in order to avoid collisions between identical GET and POST parameter names. For parameter values, it is first determined whether the parameter is known to be *irregular*. This is the case if this information has been stored in the database in (4) for some previous request(s). If the parameter is known to be *irregular*, the *cluster check* in (9) will determine whether the value deviates from the existing model to such an extent that the value should be flagged as *suspicious* in (10). More specifically, in (9) it is checked whether the Euclidean distance between the byte distribution of the parameter value and the nearest exemplar in the model exceeds threshold _B_. When the threshold is exceeded, the value is flagged as *suspicious* in (10), so that the *training monitor* (paragraph 7.4.2) can periodically check whether the training phase is completed based on whether there are still any *suspicious* values, among other criteria.

When the parameter was not known to be *irregular* in (8), in (11) it is checked whether there are more than _A_ special characters in the parameter value. In that case the parameter value model will directly be converted from a *regular* model to an *irregular* model, and the parameter will permanently be flagged as *irregular*. The new *irregular* model is created based on all the requests that have been received for this parameter value up until that point in time in the training period. On the other hand, if the parameter value is considered to be *regular* in (11), the *regular* model is simply created or updated in (13), depending on whether the model already exists.

Going back to (5), when the system is in *live mode* instead of *training mode*, in (14) it is checked whether the web resource has been whitelisted. If not, a *low priority alert* is raised in (15) and the web resource and parameter names are stored as *suspicious* in (16) and (17). *Suspicious* entities stored in *live mode* are periodically evaluated by the *suspicious entity monitor* (paragraph 7.4.4). In case there is an existing entry on the whitelist for the web resource in (14), in (18) it is checked whether each of the parameter names is on the whitelist. For each parameter name that has not been whitelisted, a *low priority alert* is raised in (19) and the parameter name is stored as *suspicious* in (17). For each parameter name that passes the whitelist check, it is determined whether the parameter is known to be *irregular* in (20). If not, the parameter value is evaluated for validity in (21) by using the regular expression that has been stored for the parameter. In the other case, when the parameter is known to be *irregular*, the *cluster check*, similar to (9), will evaluate the parameter value for validity. When the value is considered to be valid, the cluster model is updated in order to maintain the accuracy of the model. Note that for the *regular* model this continuous updating is not applicable, because of the *all-or-nothing* kind of matching that is done with regular expressions, as opposed to the *cluster check* that works with a threshold. For *regular* parameter values a *low priority alert* is raised when the parameter value does not match the model (21, 23). In addition, the parameter value will then be stored as *suspicious* in (22). For *irregular* parameter values the validity of the value is evaluated in (24). When the value is considered to be valid, the model is updated in (25) according to the continuous-updating methodology that was adopted from [34]. Otherwise, the parameter value is stored as *suspicious*. In case the value exceeded the threshold to identify *anomalous* items, which is checked in (27), the value is flagged as *suspicious-anomalous* in (28), so that the retraining process will be able to differentiate between these items and "regular" suspicious items (refer to paragraph 7.3.2). Low priority alerts will be generated based on whether the parameter value was considered to be *suspicious* (23) or *suspicious-anomalous* (29).

This concludes the general outline of the *main system*. Note that the system utilizes and combines the different aspects from the regular expression approach from [15], the clustering method from [34] and the trusted client approach from [33]. For instance, the difference between *regular* and *irregular* parameters, which was adopted from [15], can be seen in (8), (11), (20) and (21), while the clustering method from [34] is used for *irregular* parameters in (9), (22) and (23). The first part of our custom utilization of the trusted-clients concept from [33] is reflected in the fact that *low priority* alerts are generated instead of "normal" anomaly alerts in case of deviating entities, as well as the fact that the deviating entities are stored as *suspicious* for later evaluation after the *alert delay*.

7.4.2 TRAINING MONITOR - APPENDIX B.2

When Scandax is started for the first time, the system is automatically put in *training mode*. The system is also able to automatically switch from *training mode* to *live mode*, depending on certain criteria. In paragraph 7.3 we have discussed several aspects related to the training and to determining when the training can be regarded as being completed. Based on this analysis the *training monitor* has been created. However, the *training monitor* is still quite simplistic with regard to the aspects that were proposed in paragraph 7.3. For example, initial filtering of the data and rate-based model completeness checks are missing and could still be

added to improve the system. Especially the initial filtering is an important aspect that is missing. The current *training monitor* assumes that the data has already been filtered beforehand to make it free from attacks.

In the *training monitor*, it is periodically checked whether there are any (pending) changes to any models. Its sole purpose is to determine whether the current models are sufficiently in line with newly incoming data, in which case the system can be put into *live mode*. In (1) the system checks for changes in *irregular* parameter value models and in (2) this is checked for web resources, parameter names and *regular* parameter value models. When there still is activity in the creation and updating of the models, the entire set of models still insufficiently represent the monitored environment, so that it would be better to keep the system in *training mode* at that time. Therefore, if any changes have been observed that occurred since the last run of the *training monitor*, the evaluation simply ends. The system will still remain in *training mode* and the current time is registered as a reference to the point in time of the most recent evaluation in (3). However, when no changes have been observed in the period since the most recent evaluation, the system will be put in *live mode* in (7). In addition, the models for the *irregular* parameter values will be created. As opposed to the *regular* parameter value models, which are continuously updated during the training, we will want to execute the clustering algorithm afterward on the complete set of training data, which is also done in [34]. These models are constructed in (4), (5), and (6).

7.4.3 TRUSTED CLIENT MONITOR - APPENDIX B.3

This part of the system is a representation of one component of our trusted client approach, the concept of which was adopted from [33]. In essence, the *trusted client monitor* periodically executes step 1 and 2 as was described at the end of paragraph 7.2.4. Its purpose is to make sure that there is an up to date database of clients and corresponding measures of "trust", represented by the *confidence index* for each individual client. First the "uninteresting" clients and web resources are filtered out using threshold _E_ and then the excessive request ratios are filtered out by using threshold _F_. Subsequently step 1 is executed, i.e. the popularity index per web resource is calculated and finally in step 2 the confidence index per client is calculated and stored so that they can be used in the *suspicious entity monitor.*

7.4.4 SUSPICIOUS ENTITY MONITOR - APPENDIX B.4.1, B 4.2

The *suspicious entity monitor* is the main embodiment of our trusted client approach for self-adaptation. As our *main system* does not directly consider anomalies as "real" anomalies, but rather stores them as "suspicious" items and generates only a low priority alert, these "suspicious" items have to be evaluated later in order to determine whether they represent real anomalies. This is the purpose of the *suspicious entity monitor*, that evaluates the suspicious entities in the database when the evaluation is triggered, and checks whether they are legitimate based on the reputation index of the entity. For this purpose, step 4 and 5 (refer to paragraph 7.2.4) are executed by using the confidence indexes in the database, which are periodically updated by the *trusted client monitor*, as was described in the previous paragraph. We will first describe the processes that handle the suspicious web resources, parameter names and regular parameter values

(Appendix B 4.1) and after that we will outline the process that handles the suspicious irregular parameter values, which differs somewhat from the way in which the former entities are handled (Appendix B 4.2).

In [34] the rebuilding of a cluster is triggered when since the latest rebuilding operation either the number of items exceeds a threshold, the time interval exceeds a threshold, or the percentage of suspicious items exceeds a threshold. Note that because of the second condition, the rebuilding operation is periodically executed by default. Even when there are no suspicious items this is useful to refresh the model, filtering out outdated exemplars. For whitelisted web resources and parameter names, as well as for regular parameter value models this periodic rebuilding is not applicable. Therefore, we changed the condition "since the latest rebuilding operation" to "since the first suspicious item that was observed for the entity" in order to be able to apply the trigger conditions to this part of our system. For suspicious web resources and parameter names this means that the reference time will be the first request to a certain web resource or parameter name. For regular parameter values this means that the reference time will be the first request to a suspicious value for a certain parameter. Note that the rate-based trigger does not apply to entities for which there are no models yet, because the rate is calculated by dividing the number of suspicious items for a model with the number of valid items for a model, and valid items are not yet available when the model does not yet exist. We also noted that there is missing information in [33] about the percentage-related condition. When the rebuilding would be triggered whenever the percentage of suspicious items since the latest clustering exceeds a threshold, if in the first few items after a rebuilding operation there is a suspicious item, rebuilding will immediately be triggered. While the suspicious item may actually be valid, there certainly will not be sufficient suspicious items yet to create a new valid cluster. Therefore, we assume that the authors have used a threshold that specifies the minimum amount of suspicious items that was observed as a prerequisite in order to be able to apply the percentage-based criteria, which in our system is represented as _M2_.

For suspicious web resources, parameter names and regular parameter value collections (i.e. all suspicious requested values for a regular parameter) the evaluation process and the trigger for this process are shown in Appendix B 4.1. The trigger uses thresholds _H_ and _I_ for the time-criteria, threshold _L_ for the amount-criteria, and thresholds _M_ and _M2_ for the rate-criteria, the latter only for regular value collections for reasons mentioned earlier. Whenever the evaluation process is triggered for an entity, first the popularity and confidence index of the entity is calculated and finally the reputation index is determined in (2). In case the reputation is greater than or equal to threshold _J_, the entity is considered valid in (3). Otherwise a high priority alert is raised in (4). When a suspicious web resource is considered to be valid, the web resource is simply whitelisted in (5). For a new valid parameter name, the parameter name is whitelisted in (7) and a new regular or irregular model is created from the related parameter values that were requested during the *alert delay* in (8). The irregularity of the parameter is determined in a similar fashion compared to the way it is determined in step (11) of the *main system*. If any of the requested parameter values contains more than _A_ special characters, the parameter is considered to be irregular. For regular parameter value groups, it is first determined whether the existing model still applies in (10) so as to maintain the accuracy of the model (refer to paragraph 7.2.4). For this purpose, threshold _K_ is used. When at least _K_ of the total requests during the *alert delay* are considered to be valid by the existing regular model, the existing regular

parameter value model will be updated instead of replaced. Note that the *alert delay* is equal to _I_ in case the evaluation was triggered by the time-criteria and equal to the time of the first suspicious value request for the parameter in case the evaluation was triggered by the amount-based or rate-based criteria. The model that is created in (11) will be either regular or irregular based on the same conditions as in (8).

For suspicious irregular parameter values (Appendix B 4.2), the trigger for the rebuilding process uses threshold _N_ for the time-criteria, which is different from the thresholds that were used in the trigger for the first part of the *suspicious entity monitor*. This is because there is a different reference time, which was necessary for reasons that were discussed earlier in this paragraph. For the other two criteria we did use the same thresholds as in the first part of the *suspicious entity monitor*. This is because the amount and the rate of new suspicious items, provided that we use threshold _M2_, is not affected by the different reference times. We leave it up to the reader to verify the validity of this statement. This is also done in (1), using thresholds _L_ and _M_ respectively. When one of the criteria is met, just like in [34] the model is rebuilt while incorporating the suspicious items in (2). As was described in paragraph 7.2.3, we only regard it as an anomaly when a *suspicious(-anomalous) item* becomes an *exemplar of a suspicious cluster*. Therefore, in (3) we check for each cluster whether the exemplar of the cluster belongs to the suspicious items that were stored in the database. If not, we only generate a notice when the cluster contains any suspicious items. This is checked in (12), and in case there are any suspicious items in the cluster, a notice is generated in (13), announcing the inclusion of new valid parameter values that were formerly flagged as being suspicious.

When in (3) it is determined that the exemplar is among the collection of suspicious items, we evaluate additional criteria in order to validate the cluster. In (4) it is checked whether the exemplar is a special case of a suspicious item, namely a *suspicious-anomalous* item. If the exemplar is a “regular” suspicious item, the size and sparseness of the cluster are evaluated in (5). When the number of items that is associated to an exemplar of an irregular parameter is equal to or greater than _O_, the cluster is considered to be of sufficient size. Also, when the mean distance between an exemplar and its items is equal to or smaller than _B_, the cluster is considered to be of sufficient density. In case those criteria are met, the cluster is immediately regarded as valid. Otherwise the reputation of the cluster is determined in (6). Normally, i.e. in [34], a high priority “anomaly” alert would be created for all the requests that belong to clusters that do not meet the size and sparseness criteria. However, our trusted client approach introduces an extension here, by allowing a cluster that would otherwise be seen as anomalous, to be considered valid in case the reputation index of the cluster is high enough. Only when the reputation is below threshold _J_, which is checked in (7), a high priority anomaly alert is raised in (8). However, in case the reputation is greater than or equal to _J_, the cluster is considered to be valid and is included in the updated model for the parameter.

We are stricter towards allowing *suspicious-anomalous* items as exemplars. If in (9) the cluster would be considered too small or sparse, a high priority anomaly alert is immediately raised in (12). In case the cluster meets these criteria, the cluster should also have a sufficient reputation, which is calculated in (10) and evaluated in (11). Only when the size, density and reputation of the cluster are sufficient, the cluster of a *suspicious-anomalous* exemplar is considered to be valid.

In order to calculate the reputation for a cluster after the rebuilding operation, first the reputation index of the suspicious items in the cluster is determined. However, the updated cluster may also contain one or more of the existing exemplars. We also want to assign a reputation index to those items, because they can in general be seen as "trusted", as they were already incorporated in the existing model. We argue that the reputation index of an existing exemplar depends on the number of items that is associated to the exemplar. An exemplar that already exists for a substantial period of time will have a large number of items associated to it, because the system guarantees that the number of items is updated from real-time data and outdated exemplars are automatically reset. Therefore, we also calculate the reputation index for the existing exemplars that are present in the new cluster, by calculating a weighted sum of the number of items of each existing exemplar with weight _P_. We add this result to the reputation index of the suspicious entities in the cluster to obtain the final reputation index of the new cluster.

7.4.5 FORGETTING WINDOW MONITOR - APPENDIX B.5

This part of the system makes sure that outdated clusters in irregular parameter value models are periodically reset. Here "outdated" means that the exemplar representing the cluster has not been updated for a certain amount of time. Usually, exemplars are updated in real time based on incoming requests (refer to paragraph 6.3.1). The way in which the exemplar is "reset" is described in paragraph 3.3 of [34]: *"In order to avoid the impact of outdated normal behavior, if an exemplar e_i has never been assigned with a single item in a time window Δ, the exemplar is simply reset as a common item: $e_i = e_i$; $n_i = 1$; $\mu_i = 0$"*. In our system the time window threshold is represented by _Q_. For each exemplar of each irregular parameter value model t is first checked in (1) whether the model has not been updated during _Q_. When this is the case, the exemplar is reset by using the method that was described before. This way, the exemplar will have less weight assigned to it in subsequent re-clustering operations, making the item less likely to be re-chosen as an exemplar, which in its turn ensures that the model is less likely to incorporate outdated behavior.

7.4.6 SYSTEM VARIABLES – APPENDIX C

In the previous paragraphs we have mentioned several of the variables that play an important role in the effectiveness of Scandax. A complete list of the variables is included in Appendix C. In this paragraph we will briefly discuss the reasoning behind the values that we decided to assign to the variables for our practical tests.

The value of _A_ has simply been adopted from [15]. To determine an optimal initial value for _B_, we have consulted [34]. There the parameter is labeled as ε, and the authors note that "*The results presented in this paper are mainly obtained based on the adjustment of threshold ε*". From their results we have concluded that a value of 0.1 provides a good starting point for _B_. The value of _C_ is identical to the value used in [34]. For _D_ it is not specified in [34] which value is used, only that the value should be "a small constant". In [42] it is stated that the variable "*should be set to a common value - this value can be varied to produce different numbers of clusters. The shared value could be the median of the input similarities (resulting in a moderate number of clusters) or their minimum (resulting in a small number of clusters)*". Our first tests have

shown that using an initial value of -0.1 for _D_ generated a reasonable amount of clusters for different distributions, and therefore this is the initial value that we will use in our system. With _E_ we filter out the “uninteresting” clients, which only have a few requests to the application, as well as “uninteresting” web resources, having only a few requests. This is done in order to make the operation of the confidence index calculation more efficient. We use 3 as the initial value. For _F_ no guidance is provided in [33]. We have decided to use an initial value of 0.5. However, values for _G_ are provided in [33] and we adopted these. _H_ and _I_ are related to our trusted client approach and in essence they determine the *alert delay* for suspicious web resources and parameter names/regular values respectively. We have used *alert delays* of 12 and 24 hours for _H_ and _I_, but the values mainly depend on how many visitors the application has. If sufficient trust indexes can be gathered within a short amount of time, it would be better to decrease the *alert delay* to make the system more rapidly respond to threats. For _J_, the value has been directly adopted from [33]. _K_ is a variable that we introduced and for which we have estimated a proper value. The values for _L_, _M_, _N_, _O_ and _Q_ are similar to those that have been used in [34]. _P_ is a variable that we introduced for reasons described at the end of paragraph 7.4.4. The value for this variable should be based on the average number of items of existing exemplars and since we have no indication of this number beforehand we will start with using the dummy value of 0.001. Finally, _R_ should be chosen based on the rate and variety of requests that exists for an application: for an infrequently visited application it would be preferable to use a higher value of _R_, because training data is scarce. As can be seen later, we have omitted this functionality in our tests, because in our tests the training will be executed instantly on a pre-defined dataset.

7.4.7 MISCELLANEOUS SYSTEM DETAILS

The system was built in *Python* and persistent data is stored in a *MySQL* database. For sniffing the network, the *scapy* library is used. The *sklearn* library provides tools for the Affinity Propagation clustering calculations. As was mentioned in paragraph 7.4.1, the actual representation of a parameter in the system is a combination of the request method together with the parameter name. Web resources as well as this parameter representation is stored in the form of an integer hash for quick database lookup, calculated by using the efficient *xxHash* algorithm. *Jsonpickle* is used to store and retrieve models in and from the database.

7.4.7.1 Notes on byte distribution as a feature for irregular parameters

In paragraph 7.4.1 it was already mentioned that for an irregular parameter value, the Euclidean distance between the byte distribution of the parameter value and the nearest exemplar in the model is used as a measure of the extent to which the parameter value deviates from the existing model. In [34], the byte distribution is represented by a vector of 95 dimensions, representing the character distribution. The authors argue that "*There are 256 types of ASCII code in total but only 95 types of ASCII code (between 33 and 127) appear in the HTTP requests (unprintable characters are not allowed)*". However, we don’t agree with this statement, or at least is does not apply to our situation. First off, we will always decode the URL encoded parameter values, because we would like to, for example, distinguish the special character "<" in the incoming parameter value "%3Cscript%3E", rather than only regarding the percentage (%) sign. With this

approach, if we would only have a model based on the ASCII characters 33 until 127, then we would instantly disregard lots of other characters in the Unicode range, such as Arabic languages, but also characters like é, ä, etc. Identifying whether a character appears in the set of all possible alphabetic and numeric characters is achieved by using Python's native *isalnum()* method. From our point of view, there would be no reason to only include the English alphabet in the range of possible characters in the model, while disregarding other characters. However, creating a model with a vector that is as large as the range of all Unicode characters is infeasible. We have therefore implemented this in such a way that the feature vector only includes a dimension for each character that has been observed in any of the values, except for all "alphabetic" and "numeric" characters, for which there is only a single dimension in the vector. "Alphabetic" characters also include foreign alphabets and alphabet-based characters, such as é, ä, etc. Each character's Unicode code is registered and is associated with the vector of the character. Now the parameter value's feature, i.e. the vector, properly represents the extent to which the value contains non-alphabetic and non-numeric values, which is especially useful to detect web-based attacks. One could argue that the system could be simplified in a way where all characters outside the ASCII 33-127 range and all alphanumeric characters inside that range are put in one dimension, because we are especially interested in the special characters in the 33-127 range and not in alphanumeric characters within that range or special characters outside that range, because they will not be used in the most common web-based types of attacks. However, we have decided to let the system be able to identify it when "out of the ordinary" special characters are used, taking into account that there could be specially crafted attacks that include special characters outside the ASCII range. This overcomes the limitations of [34], which disregarded alphabets and alphabet-based characters other than the English alphabet in the model, as well as disregarded the special characters outside the 33-127 ASCII range.

As an example of how our system generates a byte distribution of a parameter value, we take the value "<script>". In our system, this value would be represented as a vector of [6/8, 1/8, 1/8] for Unicode codes [0, 60, 62], where code 0 corresponds to the dimension for the alphabet, alphabet-related, and numeric characters. When this byte distribution would be compared to the byte distribution of another value or an existing exemplar, the vectors usually are of different sizes and the dimensions usually correspond to different Unicode characters. In order to be able to calculate the Euclidean distance between the vectors, the vectors are first normalized so that they are of equal size and so that the dimensions correspond to the same characters.

7.5 Review and comparison to state-of-the-art systems

Because in the previous paragraphs a large amount of detailed information was provided about the inner workings of Scandax, we have included this final paragraph to serve as a summarizing review of the system's properties, as well as to give an outline of the unique properties of Scandax compared to the state-of-the-art systems which it is based on.

As was stated in the beginning of paragraph 7.4, Scandax combines methods for feature extraction, modeling, and retraining from Bolzoni [15], Pereira et al. [33], and Wang et al. [34] in order to achieve an optimal configuration for the purpose of self-adaptation. Together with certain custom additions and adjustments that were necessary to connect the different processes, these methods now work seamlessly together in one coherent system.

7.5.1 INITIAL TRAINING AND MODEL CREATION

The initial training Scandax requires a data-set that contains as least attacks as possible. Because sanitization of input data without a-priori knowledge of the data is not the main focus of this research, we have partly adopted the method that was used in [15], and extended this so that we were able to sanitize the data to an acceptable level. Because we do not regard this to be an intrinsic part of the system, we have placed the description of this method in paragraph 8.1.

With respect to the granularity of the models, Scandax whitelists web resources, whitelists corresponding parameter names and creates a separate model for every GET and POST parameter, each of which acts as a representation of the values that are considered to be valid for that parameter. This approach has (partly) been adopted from [15][3], and was preferred over coarser granularity levels, such as in [33] and [34], in which one cluster model is created for the entire application. As a logical consequence of this difference in granularity, the part(s) of the request from which features are extracted to create the model(s) also differ. It may be clear from the granularity level of the models, that Scandax extracts features from parameter values, which is different from [15] and [34][4]. The features that are extracted in Scandax are different for regular parameters and irregular parameters, a distinction that has also been adopted from [15]. For a regular parameter, its values are modeled by means of a regular expression, which is also the case in [15]. For an irregular parameter, the clustering algorithm from [34] is used to model its values, this algorithm being more suitable for self-adaptation than other clustering algorithms, such as those that are used in [15] and [33] (refer to paragraph 6.4.1). Recall that in [34] there is one cluster model for all requests to the application, while in Scandax there will be one cluster-model per parameter, with the purpose to improve the accuracy of the models as a whole. All in all, we distinguish the following unique properties of Scandax:

- Whitelisting of web resources and parameter names.
- Use Weighted Affinity Propagation (WAP) clustering [34] to model irregular parameters.
- Create a model for each irregular parameter.

[3] There are minor differences compared to [15]. For irregular parameters in [15] one model is created for all of these parameters and this remains the case even when more data has been gathered for the parameters. Also, in [15] there is no mention of whitelisting of web resources and parameter names.

[4] In [33], 7 different fields in the HTTP request are used as the basis for feature extraction, namely the URI path, URI query, Host, User-agent, Cookie, Referrer and Content. The extracted feature is a three dimensional vector, representing the amount of alphabetic characters, the amount of numeric characters and amount of non-alphanumeric characters. In [34], the only extracted feature comes from the URI (including the path and the query in one string), namely the byte distribution of ASCII characters in this string.

7.5.2 RETRAINING - TRIGGER AND PROCESS

The retraining mechanism of Scandax is primarily based on [34] in combination with an aspect from [33]. The retraining method in [34] has been designed to work with a cluster-based modeling technique. In Scandax web resources and parameter names are whitelisted, and regular expressions are created for regular parameters, which means that no clustering is involved in these models. In [15] a time-based threshold is proposed to update the regular expressions based on the observed traffic, provided that the observed traffic has been cleared from attacks by using (as was described in [15]) Snort as well as manual inspection. Instead, we use the time, amount, and rate-based (amount / time) threshold from [34] to trigger the retraining. In the retraining process, we utilize a feature on the suspicious items that has been adopted from [33], i.e. the observed anomalous traffic that is stored for deferred processing, in order to let the system be able to autonomously identify new legitimate behavior that may be incorporated in the existing model. In [33], clients are assigned with a confidence index based on their historic rate and diversity of requests. This feature is then used in [33] to identify attacks in the initial training of a cluster model. In Scandax, we use this feature to identify attacks in the collections of suspicious items during the retraining operation for web resources, parameter names, and regular parameters. This avoids the necessity for offline sanitization of the data using a signature-based IDS or manual inspection, which is a requirement in [15]. For the batch of all requests belonging to a suspicious web resource, parameter name or parameter values collection, the reputation index[5] is calculated and will determine whether the suspicious item(s) are valid, in which case the new entity will be whitelisted (in case of a web resource or parameter name) or the collection will be used to update the model (in case of a regular parameter value collection). Note that this functionality is solely focused on the retraining process. In the detection process, a low priority alert will always be generated when a suspicious web resource, parameter name or regular parameter value is observed.

Because as opposed to the aforementioned entities we do use a cluster-based modeling technique for irregular parameters, we have applied the entire retraining process from [34] to these entities, including an extension that also incorporates rigorous changes in the retraining process, as well as the client confidence index feature from [33] to identify legitimate behavior. Valid requests directly update the model so that the model adapts to gradual changes (adopted from [34]). Evaluation of the collection of stored suspicious items is triggered based on time-based, amount-based, and rate-based (amount / time) thresholds (adopted from [34]). In [34], requests that deviate from the model will not raise an alert and will be stored as suspicious items for later evaluation and retraining, and requests that deviate more than the anomaly threshold will raise an alert and will not be considered in the retraining process. The first extension in Scandax is that these anomalous items will be stored as *suspicious-anomalous* items for later evaluation and retraining, so that more rigorous legitimate changes will also be considered in the retraining process. Also, for “normal” suspicious requests as well as suspicious-anomalous items, a low priority alert will be raised upon detection.

[5] The reputation index is a term that has been adopted from [33] and that represents a measure of the diversity and the confidence indexes of the clients that contributed to the requests.

After the retraining operation, when a suspicious(-anomalous) item becomes the center of a suspicious cluster, a high priority alert will be generated (adopted from [34]).

Another extension is that Scandax incorporates the confidence index of the clients as an additional feature to identify legitimate data in the retraining process. Normally (i.e. in [34]) when a suspicious item has become a center of a suspicious cluster after retraining, a suspicious cluster being a cluster that is too small or too sparse, the cluster center and its items are flagged as anomalous. In Scandax the reputation index of a suspicious cluster is used as a fallback criterion which could result in a cluster to be considered valid even though it does not meet the criteria for size and density. As was mentioned before, just like regular suspicious items the suspicious-anomalous items are also evaluated in the retraining process. However, Scandax will use a stricter method of evaluating these entities compared to regular suspicious items. For suspicious-anomalous items that become the center of a cluster, a sufficient reputation index is a requirement instead of a fallback criterion. The following aspects can be considered to be unique for Scandax when it comes to the retraining process:

- Because the whitelisting of web resources and parameter names is unique, any form of retraining for these entities is unique. For the threshold/triggering mechanism the approach from [34] is used and for retraining the reputation index feature from [33] is used. This also applies to the retraining of regular parameters, which differs from the time-based threshold and the sanitization of the retraining data in [15].
- The retraining related to the WAP clustering algorithm has not been deployed yet on such a fine level of granularity (irregular parameter values per parameter instead of URI strings per application).
- The reputation index based extension to the WAP cluster retraining is unique, as well as the distinction between suspicious and suspicious-anomalous items.

7.5.3 OVERVIEW OF UNIQUE PROPERTIES

In the table below we provide an outline of the most significant unique properties of Scandax for three subjects that are part of the general concept of a web-based A-NIDS, as well as the corresponding properties of a specific “basic” state-of-the-art system in which the extensions of Scandax are not included. This overview is based on the summary that was provided in the previous two paragraphs.

Table 1 - Unique properties of Scandax

Subject	Basic system	Scandax
Irregular parameter model granularity	One model for all parameter values, like in [15].	One model per parameter.
Retraining for deviating web resources, parameter names, and regular parameter values	Time-based trigger for retraining with Snort and manual inspection for data sanitization, like in [15].	Time-based, amount-based and rate-based (amount / time) trigger for retraining, and reputation-index based sanitization.
Retraining for deviating irregular parameter values	Basic retraining system, like in [34].	Additional reputation-index feature in sanitization, and the distinction of suspicious-anomalous items.

8. RESULTS

When measuring the performance of an IDS system, the rate of false positives (FP) and the rate of false negatives (FN) are often used as the main evaluation criteria [47]. We will also use these metrics as a basis for our performance analysis. Note that in Scandax there is a distinction between low priority alerts, which are generated whenever a deviation from the model is observed during live detection, and deferred high priority alerts, which are generated after the retraining operation and when the system has decided that an item is anomalous (as opposed to suspicious). Because the low priority alerts are not a valid representation of the final verdict of the system, we will use the deferred high priority alerts as a basis to determine the FP and FN rates. However, this will cause the results to be biased when comparing them to performance results that are described in papers of other web-based A-NIDS's, because the majority of these systems will only generate alerts directly during the detection phase. Since we have introduced a delay in the generation of alerts, which in itself is an unfavorable property, the duration of this delay has to be taken into account when making an overall judgement about the performance of Scandax. Because of the fact that in addition to a time-based threshold Scandax also uses amount-based and rate-based thresholds to determine the optimal point in time for evaluation of a specific entity, the alert delays vary across different suspicious entities. For example, for a suspicious web resource there may only be one request over a large period of time, such that the evaluation of this suspicious entity will be triggered by the time-based threshold. In other occasions, such as a brute-force attack, the evaluation may be triggered much more rapidly, which results in a shorter alert delay in case of an anomalous entity. The purpose of our performance analysis is not to determine the optimal values of the parameters that relate to the alert delay, i.e. most of the parameters for the *SuspiciousEntityMonitor* that are listed in Appendix C. Instead, we will make sure that during a run of the *SuspiciousEntityMonitor*, it will evaluate all known suspicious items that are known at that time. We will only execute a single run of the *SuspiciousEntityMonitor* and based on the output of this process we will determine the results. Although this approach prevents a detailed analysis of the alert delay, we will still be able to get an indication of how the duration of the alert delay affects the FP and FN results by executing the *SuspiciousEntityMonitor* at different points in time. This will at least provide us with an estimation of the effects of the alert delay, which is an improvement compared to [34], in which the unfavorable side effect of the alert delay is completely disregarded. One of the reasons of using this approach is that with our test setup it is hard to properly obtain detailed measurements of the effects of the alert delay. As will be pointed out in paragraph 8.1, we will use pre-captured and prepared data-sets that will be used as input to the system. Because of time-constraints we have decided that every packet must be processed instantaneously, and that there is no delay between the processing of packets, which makes it hard to simulate the real time intervals after which the *SuspiciousEntityMonitor* would run.

As a final note before moving on to the description of the tests we would like to point out that the Affinity Propagation clustering algorithm that is used to model irregular parameter values also has several associated parameters that can be used to alter the behavior of the algorithm. Parameters include the number of

iterations of the algorithm, the damping factor, and the self-preference. For a more detailed explanation of these parameters, we refer to [48]. It proved to be a quite complicated matter to determine the optimal values for these parameters in order to achieve a proper behavior of the clustering operation, but in the end we were able to work towards a suitable configuration. In addition, we discovered that we had to slightly randomize the values in the similarity matrix that we pass to the algorithm in order to prevent unwanted output [49]. Because the configuration of AP-related parameters is application-independent and is not the main subject of this report we have excluded these variables from the list of parameters in Appendix C.

8.1 Preparation of the tests

In order for us to properly measure the performance of Scandax, a sufficiently large and diverse data-set is required, diverse meaning that the data-set should consist out of requests for a substantial number of different web resources, parameters, and clients. For our research we had the opportunity to use a server that monitors incoming and outgoing traffic to and from the campus network of the University of Twente. After inspection of the monitored traffic we found that there was one server in particular that processed a relatively high volume of HTTP traffic, namely a server that hosts many student related websites. The monitored traffic for this server was used for our tests, and we will refer to this server as *UT-Server*.

One server can host different web-applications. It is important that the system modules of Scandax work on a per-application basis, and that the models are linked to their corresponding application, the second notion being a logical consequence of the first one. This is a requirement for correct behavior of *TrustedClientHandler,* because otherwise this would result in faulty reputation index calculations. For example, there may be one small and several large applications running on the server. A highly active client for the small application should actually have a high reputation index for that application. However, when the module would not make a distinction between different the applications, it would consider the entire set of models and popularity data of these models across all applications. Because the client has visited a *relatively* small portion of the entire set of models, the reputation index will be low when it will be calculated based on all applications. In short, it is preferable to have a granularity of the client trust indexes on the level of an application. For this purpose, Scandax was built to be deployed on one server, but to be able to apply the modules on a per-application basis.

Because of this distinction between applications, in addition to selecting a server on which Scandax will be tested, we will also select a specific application for our analysis based on the criteria of traffic volume and diversity. In order to determine how well an application meets these criteria, we extended Scandax so that it can provide an overview of these statistics. This was the preferred solution compared to creating a separate tool for this purpose, because in Scandax the statistics related to the number of requests, different web resources, clients, etc. will already be available in the database as soon as Scandax has processed the traffic. An example of the statistics for a specific application is included in Appendix D. The application that seemed to be most suitable for our tests was the application corresponding to the HTTP Host *wesp.snt.utwente.nl*. Multiple HTTP Host definitions may correspond to a single application, and Scandax provides a manual

configuration for this mapping, because it cannot automatically be determined from observed traffic. After manual inspection we could not identify any other HTTP Host that corresponds to the same application. However, given that the traffic related to *wesp.snt.utwente.nl* is of sufficient size and diversity, as can be seen in paragraph 8.2.1, this is not an issue.

Now that the testing environment has been determined, the system should be trained so that an initial representation of the application can be determined in the form of web resources, parameter names, parameter values, and clients. Earlier in this report we have mentioned that the training process of Scandax requires the data to be as sanitized (cleared from attacks) as possible in order to make the model of legitimate behavior as accurate as possible. In order to sanitize the data we have adopted the approach from [15], in which Snort (a signature-based IDS) is used. We downloaded all regular Snort rules, the Community rule-set, the IP Blacklists, as well as the Emerging Threats rule-set. Subsequently we enabled all rules relating to HTTP traffic and IP Reputation / Blacklisting, and we disabled all the other rules in order to reduce the large amount of memory that would otherwise be consumed by the process. We will create different sets of training data to observe whether the size and diversity of the data-set affects the performance of the system, as well as to have an indication of the relation between time and increased diversity of the data-set. Details on the sanitized sets of data will be provided in paragraph 8.2.1.

In order to test a trained system, live data is required. As our goal is to determine the FP and FN rates, we will need to know beforehand which requests in the live data-set correspond to attacks. Because the data-sets that we use are too large for manual inspection, we also decided to use Snort for this purpose. However, properly identifying attacks by using Snort proved to be too complicated. This is because a large number of rules has been enabled for Snort, which results in many false positives, for example for requests relating to crawling bots, as well as a large number of false positives related to possible “information leaks”. Because of the size of the data-sets it is infeasible to do post-processing by means of manual inspection, or otherwise getting to know which rules have to be disabled, and therefore we eventually used a different strategy. Instead of labeling attacks in the pre-captured live data-sets, we will completely sanitize the live data from attacks, similar to the way the training data was prepared. All alerts that Scandax will generate for this set can be considered to be FP's. Subsequently we will manually execute several different OWASP Top 10 related attacks on the system. When an alert is not generated for an attack, this is a FN. Also, several legitimate changes to the application will be simulated, such as a new web resource that is requested by several trusted clients, and will observe whether the system creates FP's in these instances. All live requests will be stored in the database. Because Scandax generates deferred anomaly alerts, the system has been extended so that a link between the suspicious items and the live requests is stored. The system will then be able to mark those requests in the database for which deferred anomaly alerts are generated. From this data the FP rate can directly be inferred. For the simulated attacks and occurrences of concept drift we will manually determine the FN and FP rates.

8.2 Description of the tests

In this paragraph we will outline the tests that will be performed. First the different configurations will be described, relating to the training data, and subsequently the test cases themselves will be described, related to the data that is used as input for the trained system.

8.2.1 CONFIGURATIONS THAT WILL BE TESTED

It has become clear from Chapter 6 that proper retraining techniques are a crucial factor in order to successfully adapt to concept drift. However, the initial training phase also remains an important aspect, because with insufficient training the system will start with generating a high number of false positives after the initial training has ended. In order to determine the effects of this variable we will compare the system's performance with two different sets of initial training data. Normally we could have used parameter _R_ (Appendix C) to adjust the completeness-requirement of the initial training. A higher value of _R_ will result in a longer training period. However, with this method we cannot exactly pre-determine the duration that the training period will take. Therefore, we have decided to omit the use of this parameter for our tests, and instead vary the size of the prepared training data sets in order to simulate different training periods. The first set of training data of size *n* will be similar to the first *n* packets of the second set of training data. In addition, the second set of training data will contain a number of packets that were captured up to a certain point in time. A detailed description of both sets of training data is below. A graphical overview of some of the most requested entities per training data set can be found in Appendix D.

Table 2 - Properties of training data sets

	Training data set 1	**Training data set 2**
Application	wesp.snt.utwente.nl	wesp.snt.utwente.nl
Period	2 weeks	1 month
Requests	175,126	439,449
Clients	1,451	4,202
Web resources	936	1,665
Parameters (R)	673	1,353
Parameters (IR)	20	61

8.2.2 TEST CASES

When the system has been trained, we will execute the test cases by using certain data as input to the system, and monitoring the system's response. The input will consist out of a simulation of live data, as well as manually crafted attacks.

8.2.2.1 Simulation of live data

In order to get an indication of the FP rate of the system, a pre-captured set of live data will be used as input for the system configurations that were described in paragraph 8.2.1. As was described in paragraph 8.1

these sets of live data have been sanitized, so with only a slight degree of uncertainty they can be considered to be free from attacks, and thus representing only legitimate traffic. Therefore, the rate of anomaly alerts that is generated for this set of data provides a reasonably accurate indication of the FP rate of Scandax. For each instance during which a set of live data will be used as input for a trained version of the system, the live data corresponds to data that was captured (shortly) after the data for the corresponding training set was gathered. This way we are able to properly reproduce the first data that would be processed after the training phase has ended. The following sets of captured live data will be used.

Table 3 - Properties of live data sets

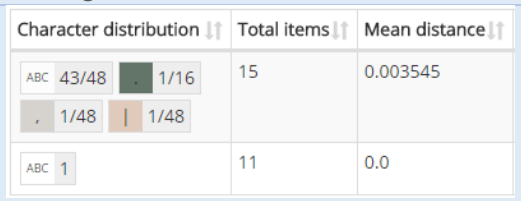
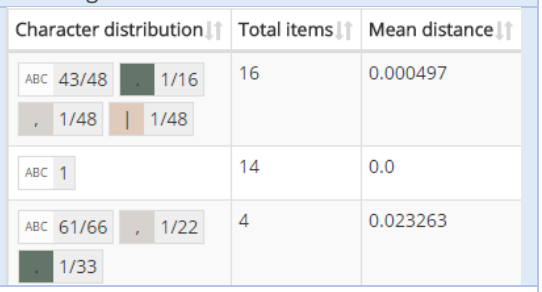
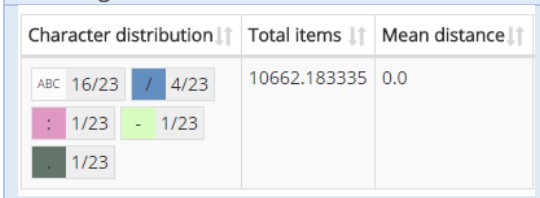
	Live data set 1	**Live data set 2**
Application	wesp.snt.utwente.nl	wesp.snt.utwente.nl
Period	5 days	19 days
Requests	91,110	214,902
Clients	646	2,074
Web resources	783	1,162
Parameters (R)	458	767
Parameters (IR)	19	39

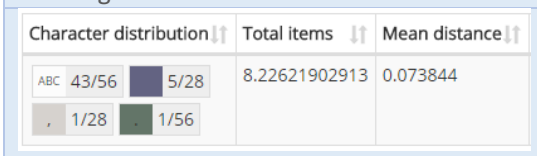
8.2.2.2 Manual attacks and concept drift simulation

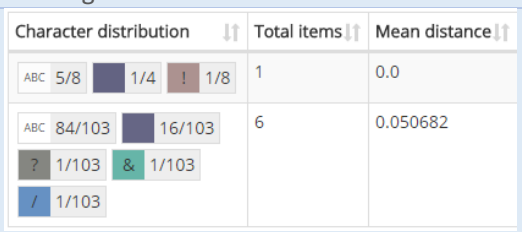
After we have obtained an indication of the FP rate of the system by observing its response on the live data, we will look at whether the system will be able to correctly identify attacks, i.e. whether the system generates TP's instead of FN's when applicable. For the attacks we have used the OWASP Top 10 as a basis, and more precisely the types of attacks that were described in paragraph 3.2, i.e. those that related to user input (GET/POST request data). Moreover, in order to test the trusted-client self-adaptation method more thoroughly, we will simulate several instances of concept drift. For this we will request different entities (web resource, parameter name, etc.) which are unknown to the system, and for which we will use different configurations of well-trusted and less-trusted clients that will request these entities. Below is an overview of the attacks and concept drift simulations that will be used in our tests. Each attack will be executed by a single unknown (untrusted) client, and for each concept drift simulation the configuration of clients is described. For the concept drift simulations, the clients will execute at least _E_ requests, so that they are considered in the reputation index calculations.

Table 4 - Manual attacks and concept drift simulations

Attack 1	**SQL Injection attack in whitelisted regular parameter**	
Details	Web resource: *~alpha/prlo/index.php* HTTP Method: *POST* Parameter: *option*	
	Training data set 1	Training data set 2
	Regex: *(com_u)(([sre]))*+*	Regex: *(com_users)*
Payload	t1' OR 1=1	

<table>
<tr><td>Attack 2</td><td colspan="2">XSS attack in whitelisted irregular parameter with strict model</td></tr>
<tr><td>Details</td><td colspan="2">Web resource: ~scienceontour/load.php
HTTP Method: GET
Parameter: modules</td></tr>
<tr><td></td><td>Training data set 1</td><td>Training data set 2</td></tr>
<tr><td></td><td> 
</td><td> 
</td></tr>
<tr><td>Payload</td><td colspan="2"><script>alert(“test”);</script></td></tr>
<tr><td>Attack 2.1</td><td colspan="2">Extended version of Attack 2</td></tr>
<tr><td>Details</td><td colspan="2">Similar to Attack 2, but with more than _O_ requests. Now when the suspicious cluster is accepted because of sufficient items (at least _O_ items), the cluster should still be considered to be anomalous, because of an insufficient reputation index.</td></tr>
<tr><td>Attack 3</td><td colspan="2">XSS attack in whitelisted irregular parameter with tolerant model</td></tr>
<tr><td>Details</td><td colspan="2">Web resource: ~isstt/wp-login.php
HTTP Method: POST
Parameter: pwd</td></tr>
<tr><td></td><td>Training data set 1</td><td>Training data set 2</td></tr>
<tr><td></td><td>20 different clusters, containing distributions of subsets of the following characters, as well as alphanumeric characters:
! _ @ . - , * > = & + # " ? < ^ \ '] / $</td><td>110 different clusters, containing distributions of subsets of the following characters, as well as alphanumeric characters:
! . - * $ ' [_ # ` ? \ / @ % ( ) , ~ { } & + =] ; : < > ^ "</td></tr>
<tr><td>Payload</td><td colspan="2"><script>alert(“test”);</script></td></tr>
<tr><td>Attack 4</td><td colspan="2">Directory traversal attack in whitelisted irregular parameter with tolerant model</td></tr>
<tr><td>Details</td><td colspan="2">Web resource: ~isstt/wp-login.php
HTTP Method: POST
Parameter: redirect_to</td></tr>
<tr><td></td><td>Training data set 1</td><td>Training data set 2</td></tr>
<tr><td></td><td> 
</td><td>Identical to model for “Training data set 1”. Total items: 34,288</td></tr>
<tr><td>Payload</td><td colspan="2">../../../etc/passwd</td></tr>
<tr><td>Attack 5</td><td colspan="2">Vulnerability scanner on non-whitelisted directory/app (PHPMyAdmin)</td></tr>
<tr><td>Details</td><td colspan="2">Web resource: /phpmyadmin/scripts/setup.php</td></tr>
</table>

<table>
<tr><td></td><td>HTTP Method: GET</td></tr>
<tr><td>Payload</td><td>-</td></tr>
<tr><td>Concept Drift 1</td><td>New web resource requested by several well-trusted clients</td></tr>
<tr><td>Details</td><td>New web resource: /new/web_resource.php
HTTP Method: GET</td></tr>
<tr><td>Clients</td><td>3 top 20% trusted clients, 3 new clients</td></tr>
<tr><td>Concept Drift 2</td><td>New web resource requested by several moderately-trusted clients</td></tr>
<tr><td>Details</td><td>New web resource: /new/web_resource_two.php
HTTP Method: GET</td></tr>
<tr><td>Clients</td><td>3 trusted clients between top 30-50%, 3 new clients</td></tr>
<tr><td>Concept Drift 3</td><td>New web resource mostly requested by non-trusted clients</td></tr>
<tr><td>Details</td><td>New web resource: /new/web_resource_three.php
HTTP Method: GET</td></tr>
<tr><td>Clients</td><td>2 trusted clients in lowest 20% range, 3 new clients</td></tr>
<tr><td>Concept Drift 4</td><td>New parameter for existing web resource requested by several well-trusted clients</td></tr>
<tr><td>Details</td><td>Web resource: ~alpha/index.php
Parameter: new_parameter
HTTP Method: GET</td></tr>
<tr><td>Clients</td><td>3 top 20% trusted clients, 6 new clients</td></tr>
<tr><td>Concept Drift 5</td><td>New parameter values for existing parameter requested by several well-trusted clients</td></tr>
<tr><td>Details</td><td>Web resource: ~tartaros/wordpress/wp-admin/admin-ajax.php
Parameter (irregular): yourmessage
HTTP Method: POST
Training data set 1
 

Training data set 2
 
</td></tr>
<tr><td>Clients</td><td>4 top 20% trusted clients, 4 new clients</td></tr>
<tr><td>Values</td><td>Values that mostly contain the special characters (between quotes) “$#*”, as well as a few alphanumerical characters</td></tr>
</table>

There is a difference between the set of models that is generated for both Training data set 1 and 2. This already became apparent from the differences in entity totals (web resources, parameters) in paragraph 8.2.1. Deviating parameter value models are visible in most of the models that were described in the manual attack and concept drift simulation overview. For example, the parameter that is subject to Attack 1 has a

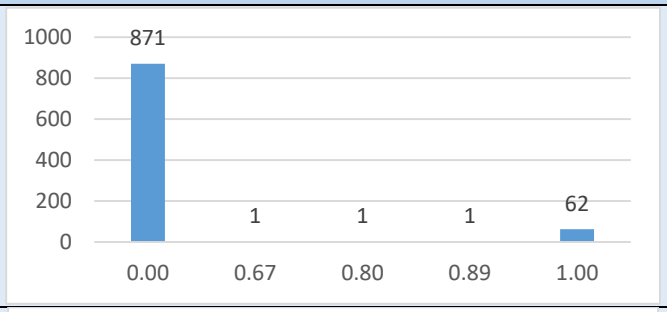
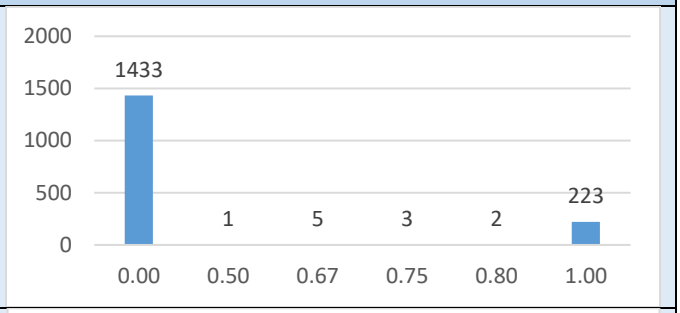
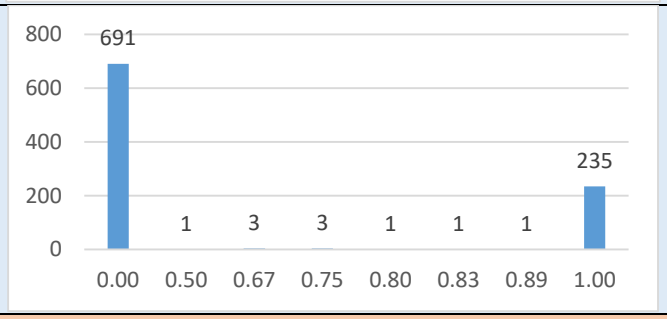
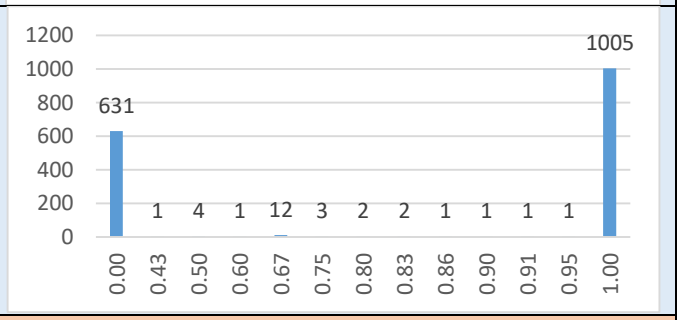
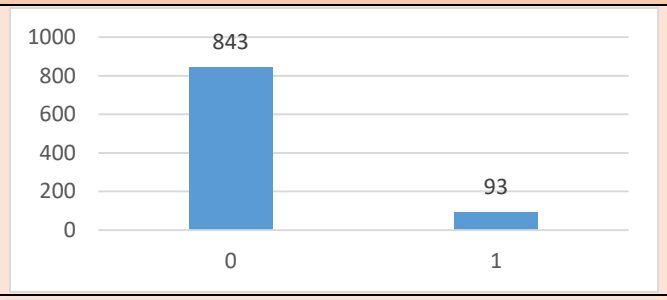
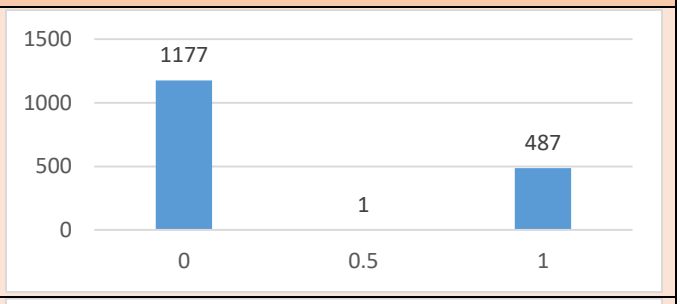
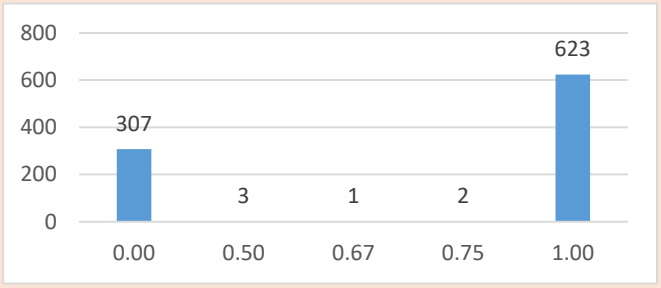
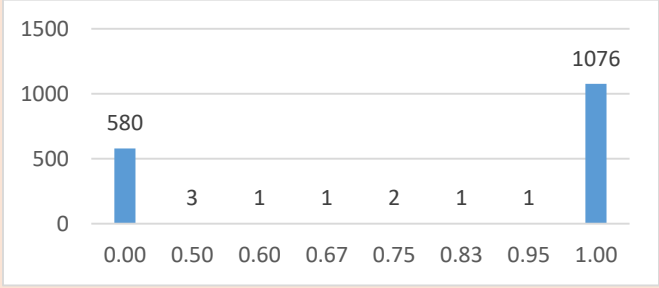
slightly different regular expression for both Training data sets, because in Training data set 1 there were two requests with “com_u” as a parameter value instead of “com_users”, which could have to do with a slow connection causing the received POST data to be incomplete. Another example is the difference in clusters for the models related to Concept Drift 5. Note that for the parameter that is subject to Attack 3, there is a major difference in the number of clusters, and the number of clusters generated from Training data set 2 is exceptionally large. As we will see later this is due to a brute-force attack that was not completely filtered out by Snort.

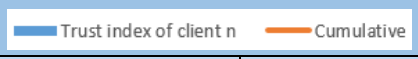
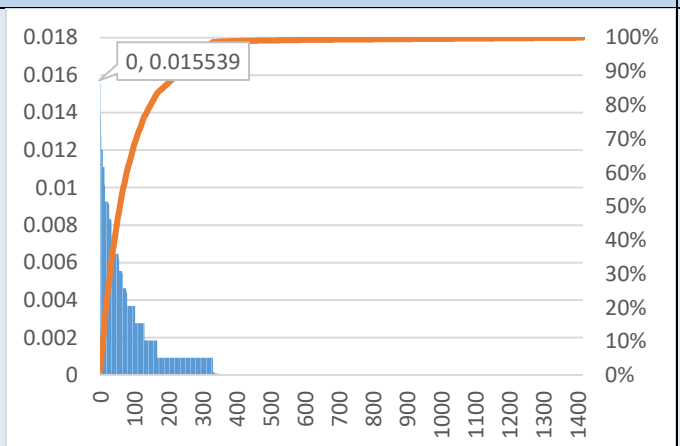
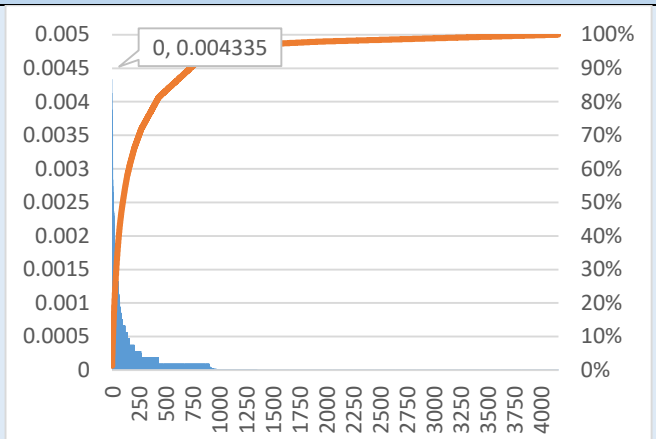
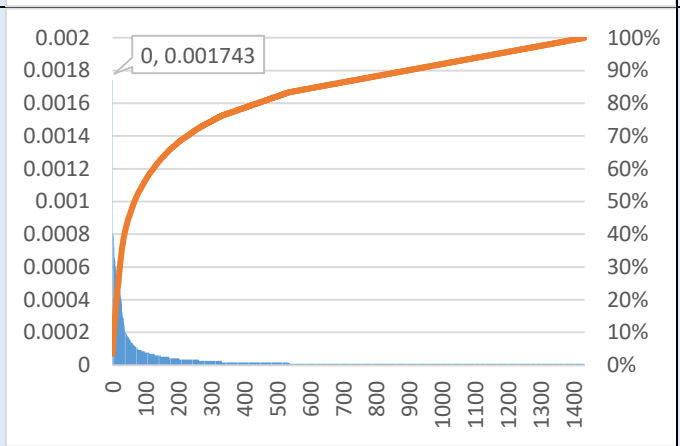
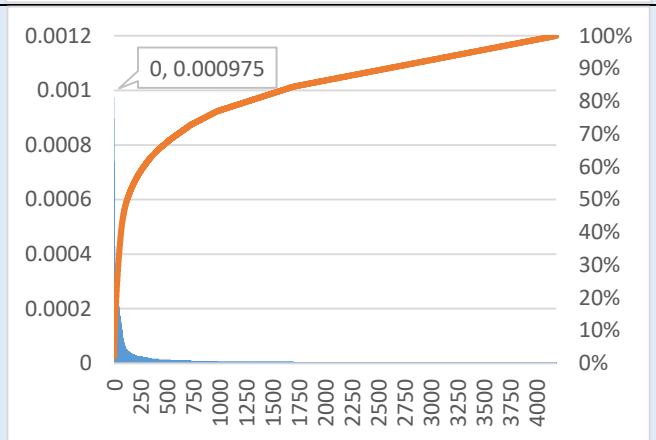
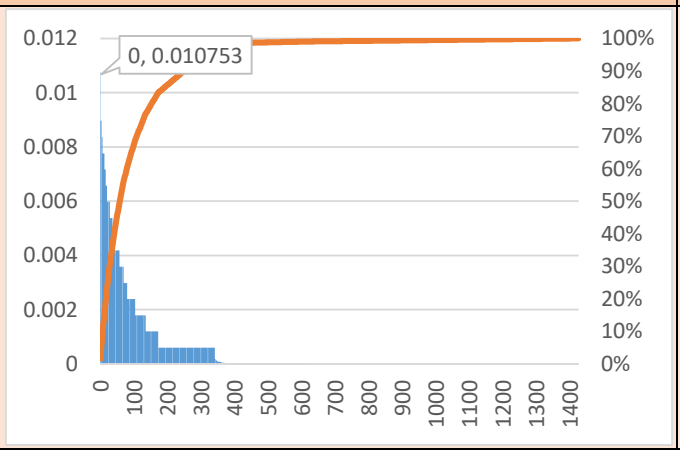
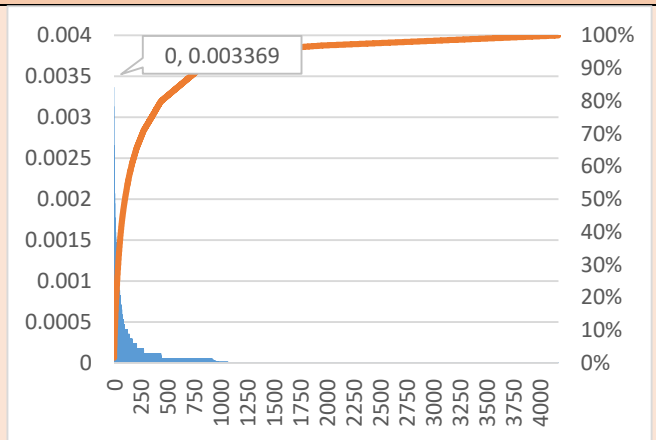
8.2.3 DETERMINING PARAMETER VALUES FOR THE TRUSTED CLIENT MECHANISM

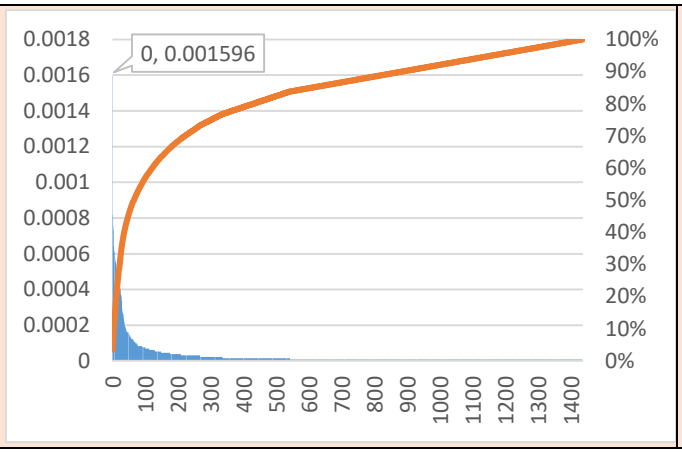
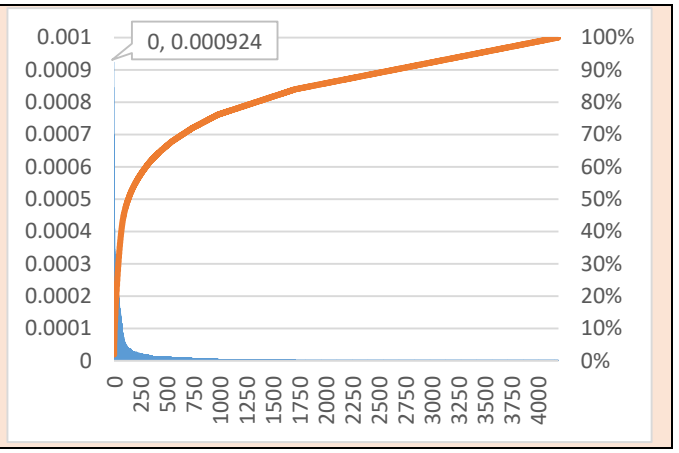
The trusted client mechanism forms the basis of the detection and retraining processes, and therefore the adjustment of the parameter values that are related to this mechanism has a substantial impact on the overall performance of the IDS. The parameters that are involved in the process that assigns confidence indexes to the application’s clients (i.e. TrustedClientMonitor) are _E_, _F_, and _G_. Of these three variables, the values for _E_ and _F_ are partly application-dependent, while _G_ is not application-dependent. More specifically, _E_ and _F_ should be adjusted based on the diversity of clients and their requests within the application. Recall that with _E_ we filter out the “uninteresting” clients, which only have a few requests to the application, as well as “uninteresting” web resources, having only a few requests. When all clients have a similar amount of requests for the entities within the application, then specifying _E_ is not necessary, because there will be no “uninteresting” clients. On the other hand, when a large portion of clients have only a few requests among the entire application, which is often the case, then setting _E_ is useful to improve the efficiency of the trust index calculations, because the “uninteresting” clients and web resources will not be considered in the calculations. However, a value of _E_ that is too high may filter out clients that should actually be assigned with a certain trustworthiness (i.e. confidence index). Parameter _F_ is used to filter out DoS attacks by putting a maximum on the fraction of requests to a web resource that a client is allowed to have, in order for that client to be considered in the popularity index calculation of that web resource. DoS attacks can be filtered out by using this principle, because these attacks will often result in particular client(s) having a high fraction of requests to the web resource. However, for applications that have only a small number of clients, it is more common for one client to contribute a high fraction of requests to a web resource, since there may only be a handful of clients that requested the web resource. Unwanted behavior now occurs when many of the clients are filtered out for most of the web resources, resulting in a popularity index of 0 for most web resources, which in its turn results in inaccurate confidence indexes for the clients. Finally, parameter _G_ defines the overall policy with respect to the relative importance of the popularity index and the confidence indexes in the calculation of the reputation index of an entity. In other words, it defines the relative importance of the number of clients for an entity versus the sum of the confidence indexes of the clients for an entity. This policy does not depend on the size or diversity of clients in an application and as was described in paragraph 7.4.6 we have adopted the values from [33], namely 1 and 3 for the weight of the popularity index and the confidence index of an entity respectively.

In order to determine proper values for _E_ and _F_ for our tests, we have performed an analysis on the effect of varying _E_ and _F_ on the popularity indexes of web resources and the confidence indexes of clients. This was done for instances of the system immediately after the training period, for both training data set 1 and training data set 2. The results of this analysis are displayed below.

Table 5 - Effects of varying trusted client parameter values

<table>
<tr><th colspan="3">Web resource popularity index
x-axis: popularity index, y-axis: number of web resources</th></tr>
<tr><td>_F_
0.5</td><td>Training data set 1</td><td>Training data set 2</td></tr>
<tr><td>_E_
3</td><td> 
</td><td> 
</td></tr>
<tr><td>_E_
1</td><td> 
</td><td> 
</td></tr>
<tr><td>_F_
0.75</td><td>Training data set 1</td><td>Training data set 2</td></tr>
<tr><td>_E_
3</td><td> 
</td><td> 
</td></tr>
<tr><td>_E_
1</td><td> 
</td><td> 
</td></tr>
</table>

<table>
<tr><td colspan="3">Client confidence index
  </td></tr>
<tr><td>_F_
0.5</td><td>Training data set 1</td><td>Training data set 2</td></tr>
<tr><td>_E_
3</td><td>  </td><td>  </td></tr>
<tr><td>_E_
1</td><td>  </td><td>  </td></tr>
<tr><td>_F_
0.75</td><td>Training data set 1</td><td>Training data set 2</td></tr>
<tr><td>_E_
3</td><td>  </td><td>  </td></tr>
</table>

_E_ 1		

As can be seen from the results, varying _F_ only has a slight effect on the web resource popularity indexes and the confidence indexes of the clients (with the exception of the web resource popularity indexes for training data set 1 and _E_ = 1, which has more deviating results). On the other hand, changing _E_ has a profound effect on the number of web resources that is assigned with a popularity index, as well as the overall magnitude and distribution of the client confidence indexes. Given the number of web resources with a zero popularity index, and given the fact that changing _F_ from 0.5 to 0.75 does not profoundly increase the number of web resources with a nonzero popularity index, this suggests that there is a large portion of web resources that have only 1 request. This is because for these web resources the request fraction will be 1 for the client that initiated the request, such that the client is not included in the popularity index calculations for the web resource, and thus resulting in a popularity index of 0. Statistics support this notion, as training data set 1 contains 248 web resources with only a single request, and training data set 2 contains a total of 437 of one-time requested web resources. This is roughly equal to the number of web resources with a popularity index of zero for _F_ = 0.75 and _E_ = 1, the remainder being a result of web resources that have requests from two clients, of which one client contributes to more than 75% of the requests. In that case that client as well as the requests for that client for that web resource are disregarded, resulting in a 100% contribution of requests for the other remaining client, which in its turn is disregarded, resulting in a popularity index of zero for the web resource (refer to paragraph 6.1.1.1 for the algorithm).

The fact that the different values for _E_ have such a profound effect on the popularity indexes is because when _E_ changes from 3 to 1, the number of clients that is considered in the calculations increases significantly, resulting in smaller fractions of requests per client per web resource, such that they will not be filtered out by _F_. This eventually leads to more nonzero popularity indexes. When more web resources have a nonzero popularity index, overall the v_i in part (3) of the algorithm will increase (refer to paragraph 6.1.1.1), i.e. the sum of all popularity indexes that clients visited will be higher. As can also be observed in part (3) of the algorithm, the divisor of the confidence index calculation involves a multiplication with the maximum of all v_i. Since, this maximum is now a larger value, the overall confidence indexes will be lower. For example, for Training data set 2 and _F_ = 0.75, max(v_i) is 57 for _E_ = 3, while max(v_i) is 368 for _E_ = 1.

We do not have a sound explanation of why the authors of [33] decided to include max(v_i) in the divisor here, but it is apparent that it causes deviations in the order of magnitude of the confidence indexes of clients.

The data also shows that when more clients are included in the calculations, the curve of the cumulative total in the Pareto charts will be less steep. We argue that a steeper curve for the confidence indexes is more favorable, because then more weight is assigned to clients that have requested multiple web resources and less weight to the large amount of clients that have only requested 1 or 2 web resources. We have therefore chosen to perform our tests with a value of 3 for _E_. We have seen that for the different values for _F_ there is not a substantial difference in popularity and confidence indexes. For _F_ we will stick to a value of 0.5.

The final parameter that is related to the trusted client mechanism is parameter _J_, which imposes a threshold on the weighted sum of the popularity index and the confidence index (i.e. the reputation index) of a suspicious entity in order for the entity to be considered as legitimate in the SuspiciousEntityMonitor process. Thus, with _J_ the policy with respect to regarding suspicious behavior as legitimate (based on the reputation index) can be defined, a higher value of _J_ representing a stricter policy, while a lower value for _J_ represents a more flexible policy. A suitable value for _J_ therefore depends on the preferred policy, but after trial and error we have chosen to use a value of 1.01 for our tests.

8.3 Test results

In the following two paragraphs the test results for training data sets 1 and 2 are provided. This includes the FP rates for the live data sets, as well as the verdicts of the system related to the manual attacks and the instances of concept drift. Paragraph 8.3.3 gives a brief summary of the output of Scandax during the different stages of operation. Finally, in paragraph 8.4 the results will be discussed.

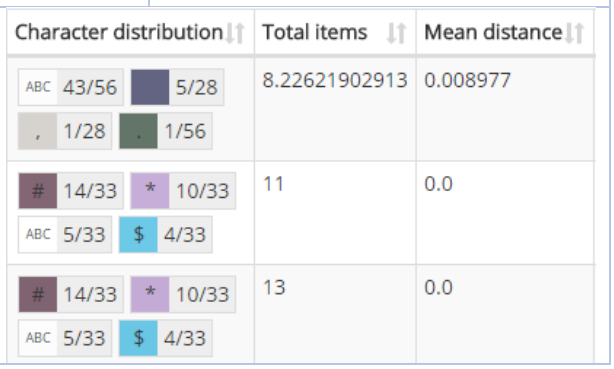
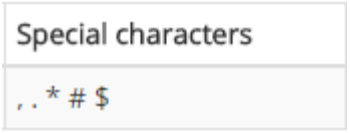
8.3.1 TRAINING DATA SET 1

Table 6 – Training data set 1 - Live data sets: False Positive rate analysis

	Requests	Alerts	Alerts grouped by request	FP rate
Live data set 1	51,360	363	132	0.3%
Live data set 2	214,902	902	678	0.3%

Table 7 – Training data set 1 - Manual attacks: False negative rate analysis

		Detection	SuspiciousEntityMonitor
Attack 1	TP	Value does not match regex	Anomalous, because the reputation index is insufficient (1.0 < 1.01)
Attack 2	TP	Parameter value's *nearest exemplar distance* for GET parameter "modules" is greater or equal to *irregular parameter* ***anomalous*** *threshold* (0.2589 >= 0.2)	Anomalous, because the exemplar is a suspicious item and the cluster contains insufficient items (1 < 4)

Attack 2.1	TP	(Identical to Attack 2)	Anomalous, because the exemplar is a suspicious-anomalous item and the reputation index of the cluster is insufficient (1.0 < 1.01)
Attack 3	FN	Parameter value's *nearest exemplar distance* for POST parameter "pwd" is greater or equal to *irregular parameter **suspicious** threshold* (0.1873 >= 0.1)	Legitimate. The model for irregular POST parameter "pwd" has been rebuilt and contains one or more valid clusters.
Attack 4	TP	Parameter value's *nearest exemplar distance* for POST parameter "redirect_to" is greater or equal to *irregular parameter **anomalous** threshold* (0.3585 >= 0.2)	Anomalous, because the exemplar is a suspicious item and the cluster contains insufficient items (1 < 4)
Attack 5	TP	A suspicious web resource was detected: "phpmyadmin/scripts/setup.php"	Anomalous, because the reputation index is insufficient (1.0 < 1.01)
Concept Drift 1		Suspicious web resource was detected: "new/web_resource.php"	**Valid**, because its reputation index is sufficient (1.0524 >= 1.01)
Concept Drift 2		Suspicious web resource was detected: "new/web_resource_two.php"	**Invalid**, because its reputation index is insufficient (1.0055 < 1.01)
Concept Drift 3		Suspicious web resource was detected: "new/web_resource_three.php"	**Invalid**, because its reputation index is insufficient (1.0011 < 1.01)
Concept Drift 4		Suspicious parameter name detected for web resource "~alpha/index.php": GET parameter "new_parameter"	**Valid**, because its reputation index is sufficient (1.0212 >= 1.01)
Concept Drift 5		Parameter value's *nearest exemplar distance* for POST parameter "yourmessage" is greater or equal to *irregular parameter **anomalous** threshold* (0.8366 >= 0.1)	**Two new valid clusters** with suspicious items as exemplars, because reputation index is sufficient (1.016 >= 1.01 for each cluster)
		Properties of updated model (clusters on the right, special character sequence below)	
			

8.3.2 TRAINING DATA SET 2

Table 8 - Training data set 2 - Live data sets: False Positive rate analysis

	Requests	Alerts	Alerts grouped by request	FP rate
Live data set 1	51,360	56	29	0.06%
Live data set 2	214,902	24	24	0.01%

Table 9 - Training data set 2 - Manual attacks: False negative rate analysis

<table>
<tr><th></th><th></th><th>Detection</th><th>SuspiciousEntityMonitor</th></tr>
<tr><td>Attack 1</td><td>TP</td><td colspan="2" rowspan="3">Identical to Training data set 1 results</td></tr>
<tr><td>Attack 2</td><td>TP</td></tr>
<tr><td>Attack 2.1</td><td>TP</td></tr>
<tr><td>Attack 3</td><td>FN</td><td>Similar to Training data set 1 results
(0.1650 >= 0.1)</td><td>Identical to Training data set 1 results</td></tr>
<tr><td>Attack 4</td><td>TP</td><td colspan="2" rowspan="2">Identical to Training data set 1 results</td></tr>
<tr><td>Attack 5</td><td>TP</td></tr>
<tr><td colspan="2">Concept Drift 1</td><td colspan="2" rowspan="4">Identical to Training data set 1 results</td><td>Invalid, because its reputation index is insufficient (1.0026 < 1.01)</td></tr>
<tr><td colspan="2">Concept Drift 2</td><td>Similar to Training data set 1 results
(1.0004 < 1.01)</td></tr>
<tr><td colspan="2">Concept Drift 3</td><td>Similar to Training data set 1 results
(1.0001 < 1.01)</td></tr>
<tr><td colspan="2">Concept Drift 4</td><td>Invalid, because its reputation index is insufficient (1.0011 < 1.01)</td></tr>
<tr><td colspan="2">Concept Drift 5</td><td colspan="2">Similar to Training data set 1 results
(0.7673 >= 0.1)</td><td>Invalid cluster, because its reputation index is insufficient (1.0029 < 1.01)</td></tr>
</table>

8.3.3 TRAINING & LIVE DATA SETS – ANALYSIS OF IDS OUTPUT

During the training phase, as well as in the period during which the live data is processed, and eventually during the run of SuspiciousEntityMonitor, different output is generated by Scandax. In chapters 7 and 8 we have made a distinction between low priority alerts and (deferred) high priority alerts, the latter of which are used for our TP and FN calculations. In addition to these alerts, Scandax also generates output for less urgent events, such as new models that are created and cluster models that are updated based on legitimate live traffic (refer to paragraph 7.4.1). Below is a high level overview of the output that was generated during the different stages of one of the tests. More specifically, these stages consisted of the training period with Training data set 1, the live data period with Live data set 2, and the subsequent run of SuspiciousEntityMonitor after the live data had been processed.

Table 10 - IDS output for Training data set 1 and Live data set 2

Period	Alert identifier	Type	Amount
Training	new_whitelisted_web_resource	success	936
	new_whitelisted_parameter_name	success	693
	new_regular_parameter_value_model	success	686
	new_irregular_parameter_value_model	success	20
	updated_regular_parameter_value_model	success	852
	updated_irregular_parameter_value_model	success	23
	minor_update_irregular_model_from_legitimate_value	success	344
	training_completed	info	1
	invalid_request_observed	alert	3
Live data	suspicious_parameter_value	warning	1,181
	suspicious_web_resource	warning	1,330
	suspicious_anomalous_parameter_value	warning	40
	suspicious_parameter_name	warning	599
	minor_update_irregular_model_from_legitimate_value	success	22,496
Suspicious Entity Monitor	new_valid_web_resource	success	15
	new_valid_parameter	success	18
	new_valid_regular_parameter_values	success	14
	irregular_model_updated_with_suspicious_items	success	3
	invalid_web_resource	alert	385
	invalid_parameter	alert	387
	invalid_regular_parameter_values	alert	121
	anomalous_cluster_too_small	alert	9

This overview clearly shows that during the training period all the initial models are created and updated based on new data that is used as input to the system during this training period. There are a few alerts for *invalid_request_observed*, which are generated for requests that of certain types that are currently not supported by the system. In paragraph 8.4 we will discuss these unsupported request types. The single notification of type *info* is generated by the system whenever the training for a certain application is completed. For an overview of the criteria that are used to determine whether the training is completed we refer back to paragraph 7.4.2.

During the processing of the live data set, a substantial amount of data was seen as suspicious. This means that either the training period of 2 weeks was not sufficient to make an accurate representation of the application at that time, or the application has changed in the meantime, the latter of which is less likely, but possible. Also, there is a large number of *success* notifications for irregular parameter value models that were updated based on values that only slightly deviated from the model. However, approximately 90% (20,084) of these notifications were generated for two irregular POST parameters belonging to the *~isstt/wp-login.php*

web resource, specifically the *pwd* and *redirect_to* parameters. This suggests the presence of a brute-force Wordpress login attack in the live data set.

The SuspiciousEntityMonitor module generates notifications whenever a new model has been created from certain suspicious items, as well as when anomalous items have been detected among the suspicious items. The number of notifications generated by SuspiciousEntityMonitor is substantially lower than the number of notifications related to the detection of suspicious items, i.e. during the processing of the live data (952 versus 3,110). This is because different low-priority (warning) notifications that are generated for suspicious items during the live data processing can correspond to a single web resource, parameter name, or parameter value collection, and SuspiciousEntityMonitor will only generate one notification for a single entity.

When we observe the differences in the number of model-related entities between the training data and the live data, Scandax should have detected at least 1162 - 936 = 226 deviating web resources and at least (767 + 39) - (673 + 20) = 113 deviating parameter names. The actual number of deviating web resources and parameter names was (15 + 385) = 400 and (18 + 387) = 405 respectively, indicating that a substantial part of the entities in the live data set were not present in the training data set.

8.4 Discussion

The results have shown that for Training data set 1 the FP rate is higher than for Training data set 2. For both training data sets the system is able to correctly identify the manually executed attacks, except for the third attack, which was an XSS attack on an irregular parameter with a tolerant model. Specifically, the POST parameter *pwd* of web resource *~isstt/wp-login.php* was targeted. As could be observed in the description of the manual attacks, the model is tolerant (especially for Training data set 2) because of the size of the special characters sequence and large amount of clusters in the model. In the previous paragraph we noted that there were signs of a brute-force attack on this parameter (among others), and further analysis has supported this notion.

For the simulations of concept drift the difference in results is more significant. For 3 out of 5 instances the verdict varies between both training data sets. In paragraph 8.2.2.3 we have observed that the overall size of the confidence indexes of the clients is different for both training data sets. To be more precise, the average size of the confidence indexes differs by a factor of 10. This is also the case for the top 20% of confidence indexes. This means that for the similar value of _J_ that was used, there should have been more trusted clients that requested the new entity in the concept drift instance in order to raise the reputation index to a sufficient level.

8.4.1 LIMITATIONS

Before moving on towards the conclusions of this research, we first want to point out some of the shortcomings of the system as used in our tests. The first one is that the system currently does not support every type of HTTP request. For example, for HTTP requests with a “POST” request method, only requests

with a content-type of “application/x-www-form-urlencoded” will be processed. This means that, for example, XML and multipart/form-data requests are ignored. The system could be extended so that it will also include those requests in the analysis. Decisions with respect to the granularity of the model will have to be made here. For example, for an XML requests its elements could be modeled individually (high granularity), or the complete XML document could be treated as an irregular value (low granularity). For simplicity we have decided to keep this as future work. Therefore, some packets were ignored in our tests, and for the application that was tested in this research the rate of processed packets was about 80%. The grand totals of requests that were mentioned in the previous paragraphs represent this 80%, i.e. all non-ignored requests.

Second, a limitation of the itself is that it is only able to detect a specific type of web attacks, namely inserting malicious content in the request url. A web attack such as a DoS attack would be harder to detect. As we have seen in the results, a DoS brute-force attack on a parameter value may be detected by the system administrator when he or she notices that there is a substantial amount of invalid parameter value alerts within a short period of time, but the system will not be able to make the distinction of a DoS attack by itself.

Another limitation of our research is that due to time constraints we were only able to perform our tests on one application. Although we have carefully chosen this application based on size and diversity of requests, we may have observed different behavior of the system for other applications. On the other hand, differences in applications are mostly represented by a different number of web resources and parameters, and a different magnitude and distribution of clients and their requests, and this is also tested in our research as we observe the system’s results with different sets of training data.

Finally, there are limitations in the tests themselves. For one, for the FP rate analysis we have assumed that Snort was able to completely sanitize the data-sets that served as the input for the system for this analysis. However, there may be attacks that could not be detected by Snort and that were or were not detected by our system, which could unknowingly be reflected in the FP rate. Also, due to time constraints we were only able to test the system’s behavior on a handful of manual attacks. Although we have carefully chosen these attacks based on the most common web based attacks, the results only give a rough indication of the TP rate.

8.5 Practical usability

In order to determine the practical usability of the system, we consider two important aspects that can serve as criteria for the decision on whether to use the system in a production environment. First off, the system should be able to handle large amounts of traffic without excessively using the CPU and memory of the server on which the system is running. Although the analysis of this aspect was not part of our research, we have observed that on our testing server the system was able to process approximately 50 requests per second during the live detection phase. Our testing server has 16 GB of memory and 12 2.50GHz processors. The live detection process of our system is currently single threaded and this could be improved to utilize multi-processor environments. However, this process requires a relatively low amount of resources, because of the simplicity of this routine, which at most includes a handful of database lookup and write operations, a

regular expression matching operation for regular parameters, or a Euclidean distance calculation for irregular parameters (refer to Appendix B1). Other processes that are part of the system and which are more resource intensive, such as *SuspiciousEntityMonitor*, run in separate threads. Based on these notions and the experiences that we have obtained during our tests we conclude that for low to medium-volume applications our system should be able to run without using excessive resources, and that there are several ways to improve the efficiency of the system so that it can more easily cope with high-volume applications.

Another important aspect that determines the practical usability of the system is the ease with which the system could be deployed in a certain production environment. As became apparent in chapter 7, many parameters are involved in the configuration of Scandax. Since our tests were performed for a single application, this may give rise to skepticism, in such a way that one may suspect that the configuration of parameter values as we have described them is over-fitted and would solely be applicable to our specific tests. Our aim is to remove this suspicion by convincing the reader that our system can well be applied to other applications while only having to slightly adjust the system so that the required parameters are automatically configured by the system, instead of having to manually specify the values of all the parameters that are listed in Appendix C. For this purpose, for each of the parameters we will describe whether there are application-specific properties that possibly affect the optimal value of the parameter, and whether the value could be automatically inferred by the system itself.

_A_	Using the total number of special characters in a value in order to define an irregular parameter, is a way to limit the complexity of the regular expressions *in general* and therefore we can simply conclude that this parameter is not application-dependent.
_B_	Because _A_ is not application-dependent, the concept of an irregular parameter is identical across applications. Deviations of irregular parameter values from the parameter's model are calculated using a fixed Euclidean distance based calculation and since the deviation from the model is a general notion among applications, the value of this parameter is not application-dependent.
_C_	Similar to the reasoning of _B_.
_D_	This is a parameter that is specific to the Affinity Propagation clustering algorithm, and is used to control the diversity of clusters that the algorithm produces in general. independent of specific properties of the data itself, and therefore independent of the application for which the algorithm is used.
_E_	This parameter was used in our tests to make the resulting data more compact and thus easier to analyze. Ignoring this parameter will slightly increase the resource-usage of the confidence-index related calculations, but it has no effect on the accuracy of the calculations themselves. Keeping the parameter's value at 3 or otherwise ignoring it when using the system for different applications will therefore not affect the detection capability of the system.
_F_	When we would use a fixed value for this variable among different applications, we would not be able to utilize the DoS-mitigation function that was partly the purpose of this parameter. This is because DoS attacks will have different rates of requests for a web resource among different applications, because for one application the average number of requests per web resource will be higher than for another application, such that an identical DoS attack will not have an identical request rate for that web resource. However, we argue that this has a negligible effect on the overall performance of the system. A single DoS/bruteforce attack usually only request a single

	web resource. This is also what we observed during our research (refer to paragraph 8.3.3). In such a case the corresponding client will be assigned with a low confidence index, and therefore this client will not be able to affect the overall security of the system. As for the main purpose of this parameter, i.e. imposing a restriction on the minimum number of clients that is required for a web resource to be able to obtain a popularity index of 1, the parameter value can safely be fixed among different applications, as long as the following is considered: the reputation-index system will only start to mark new items as valid when sufficient training data is available to assign at least a portion of the web resources with a nonzero popularity index. This means that when the value is set to 0.1, as in [33], there should be sufficient training data such that at least a portion of the web resources has requests from approximately 10 different clients or more, because otherwise no web resources will have a nonzero popularity index, such that each client will have a confidence index of zero. This results in all reputation indexes being below the threshold, and every potential concept-drift will be flagged as anomalous, resulting in more false positives. Fortunately, most applications will rapidly have sufficient training data available for this purpose.
_G_	This parameter defines the general policy with respect to considering the variety (popularity index) and trust (confidence index) in the reputation index calculations of newly observed entities, and does not depend on any specific properties of an application.
_H_	In paragraph 7.4.4 we stated that a different threshold than _N_ (which applies to irregular parameter values) was required for the time-based trigger for the evaluation of suspicious web resources, parameter names, and regular parameter values. The main reasoning behind this was that periodic rebuilding of the model to filter out outdated behavior is not applicable to the entities other than irregular parameters. However, the purpose of _N_ is also to impose an upper limit on the time between the subsequent evaluation of suspicious items for a model. With this in mind, we can use _N_ as a basis for this upper limit, and base _H_ and _I_ on this value in order to automate the configuration of this parameter. In other words, _N_ can be seen as an interval after which suspicious items will be evaluated for an existing irregular parameter model, and _H_ and _I_ will be based on _N_ and apply to suspicious items for not-yet-existing models. In general, each of _N_, _H_, and _I_ can be seen as an "upper limit on the interval after which suspicious items are evaluated". When using this approach, _I_ and _N_ can be assigned with equal values because they both relate to parameters, and _H_ could have a smaller value because we have assumed that parameters are requested less often than web resources (we have set _H_ at 0.5 × _I_).
_I_	This value is based on _N_, as we have described in the reasoning of _H_.
_J_	In paragraph 8.2.3 we stated that in the equation that calculates the confidence index of a client, the multiplication with $\max(v_i)$ in the divisor results in different orders of magnitude for client confidence indexes based on the number of web resources with a nonzero popularity index that clients have requested, which usually depends on the number of web resources with a nonzero popularity index for a specific application. This means that for different applications with different numbers of web resources, the confidence indexes of clients will have different orders of magnitude, and since this imbalance is not accounted for in the overall reputation index calculation, the threshold _J_ would have to be adjusted to account for these differences. In other words, when confidence indexes for one application are generally lower than for the other, a lower _J_ would need to be used in order to apply a similar security policy. Since this requirement is not optimal considering that we strive for an autonomous system, we propose to omit the multiplication with $\max(v_i)$ from the equation. It is not only that any explanation on this multiplication is missing from [33], but also that including this operation cannot be substantiated by us. For calculating the confidence index, or level of trust, of a client, it should be sufficient to

	measure the sum of popularity of the web resources that was visited by a client, compared to the sum of popularity of all web resources. We therefore conclude that _J_ can be used to specify a security policy regardless of the application that is being monitored, given that the trust index calculation is properly modified so that the multiplication with $\max(v_i)$ is excluded from step (3) of the algorithm.
_K_	This parameter is used to determine whether an existing regular model still applies, and defines a general policy/rate that does not depend on application-specific properties.
_L_	This parameter should be set to the minimum value for which in general accurate results can be obtained, which means that in general sufficient data is available for accurate reputation index calculations. Provided that this has been done, the same value can then be used for different applications. A consequence of this is that for low-volume applications this amount-based trigger may never be applicable, because the time-based trigger _N_ will always apply since not enough suspicious items (less than _L_) will flow in during this period. We argue that for such low-volume applications _N_ and (less likely) _M_ can act as fallback triggers without affecting the detection accuracy for this kind of an application.
_M_	This parameter defines a rate for the rate-based trigger for the evaluation of suspicious values for existing parameter value models. Because this is a rate, it does not depend on the number of items that flows in per second for a specific application. It can therefore be defined independent of the application. Moreover, since we have suggested to set _L_ to the minimum value for which it is likely that there is sufficient data for accurate reputation index calculations, _L_ (i.e. the amount-based trigger) will already be responsible for identifying short term change. When we also would include the rate based trigger using _M_, we would also have to specify _M2_, but _M2_ will have the same requirements as _L_, because sufficient data should be available in order for the reputation-index calculations to be effective. When _M2_ has the same value as _L_, the rate-based trigger is redundant, because it will prompt at the same time as the amount-based trigger. This means that when setting _L_ in our proposed way so that the amount-based trigger can be used among different applications, the rate-based trigger will become redundant, and values for _M_ and _M2_ will not have to be specified.
_M2_	Since we have concluded that _M_ is redundant given the requirements that we put on _L_ in order to specify a value for use in different applications, _M2_ is also redundant as does not have to be specified.
_N_	This parameter relates to the time-based trigger, which is a fallback for the amount-based trigger in order to make sure that suspicious items are evaluated every once in a while. It serves as an upper limit on the interval after which suspicious items are evaluated, and can be used among different applications. The only caveat is that for low-volume applications the number of suspicious items for an entity may be so small, that in such cases one would always prefer to regard these items as anomalous. For the cluster-based calculation with *suspicious-anomalous* items this usually holds, as it will probably result in a cluster that is too small, such that the cluster is rejected. However, in other cases a high importance or weight should be assigned to a nonzero popularity index in the calculation of the reputation index, because a nonzero popularity index indicates that the entity has been requested by at least a certain number of clients.
_O_	When specifying a value for this parameter, one can ask the following question: given that some new irregular value-batch is observed, how many similar values should there at least be in order for the values to be able to be considered legitimate? Since the answer of this question is independent of the size of traffic volume of an application, the same value for _O_ can well be used for different applications.

_P_	The value of 0.001 that we specified for this parameter was suitable for our specific tests, but when we would use this value among different applications, this can lead to incorrect results. Especially when we consider the difference in traffic volume between applications, it can be concluded that for high-volume applications the reputation index contribution of existing exemplars will become too high, because the reputation index increase per item is linear, as the number of items is simply multiplied by the value of this parameter. Note that existing exemplars can have many items, as they (i.e. their *number of items* property) are updated in real time based on all legitimate traffic. Therefore, in order to be able to use this parameter for different applications without having to manually specify a value for each application, it is required that a limit is imposed on the maximum value that this parameter can have. However, we still would like to base the height of the value on the number of items in the existing exemplar, because that property is a proper representation of the level of trust of the exemplar. A method that meets these requirements is to calculate _P_ automatically by determining the fraction of items in the exemplar compared to the total number of items in the cluster.
_Q_	This parameter can simply be assigned with a value that is large enough so that even for low volume applications it can be asserted that when an exemplar has not been assigned with a value, the exemplar can be seen as "outdated". For higher-volume applications this value will then always be sufficient to filter out "outdated" items, so this value can then be used among different applications.
_R_	For this value the same principle applies as for _Q_, i.e. a value should be set that is large enough so that even for low-volume applications the training data will be a proper representation of the application when no model changes have been observed during _R_ seconds. Note that it may be wise to impose an upper duration on the overall training period, as a large _R_ may otherwise cause an excessively long training period for high volume applications, because when there is a lot of traffic, there is a high chance that there will be a (small) change during _R_.

As we can infer from this evaluation, only minor changes to the system are required in order to make it widely applicable among applications that have different numbers of clients and entities, and have different distributions of client-originating requests. Extensive tests among many different applications are a prerequisite to fine-tune the system in order to achieve more optimal results, but we argue that in this paragraph we have proposed a theoretically sound configuration to make the system practically usable for different kinds of applications.

9. CONCLUSIONS AND FUTURE WORK

Starting with our introduction into web based ANIDS's in the first chapters, we subsequently have distinguished several challenges that are involved in optimizing the performance of this kind of a system, by means of formulating research questions for which we aimed to find answers by means of literary research, as well as composing an innovative system that copes with these challenges using a combination of state-of-the-art techniques, and finally examining the performance of this system in order to determine the effectiveness of our approach. In this final chapter we will reach a verdict on whether our proposed methods

are indeed suitable to cope with the aforementioned challenges. This, backed by the knowledge that we have acquired during this research, leads to definitive answers on our research questions. What then remains is an overview of the future work that has resulted from our research.

9.1 Conclusions

Let us recapitulate the research questions. The main research question is as follows.

HOW CAN A WEB-BASED ANIDS AUTONOMOUSLY ADAPT TO LEGITIMATE CHANGES IN THE MONITORED WEB APPLICATION?

In chapter 5 we have distinguished three sub questions, which together serve in answering the main research question. These sub questions are the following.

WHEN CAN AN OBSERVED CHANGE BE CONSIDERED TO BE LEGITIMATE?

–

WHEN A LEGITIMATE CHANGE HAS BEEN DISTINGUISHED, HOW TO UPDATE THE MODEL?

–

ARE CERTAIN TRAINING ALGORITHMS MORE PREFERABLE THAN OTHERS WHEN IT COMES TO THE PROPOSED METHODS FOR DISTINGUISHING LEGITIMATE CHANGES AND RETRAINING THE MODEL?

Based on the promising results that were shown in chapter 8, we can conclude that our method can serve as an effective way to cope with the challenges that are involved in web-based anomaly detection. However, the method has its limitations. In addition to the shortcomings that we described in paragraph 8.4.1, an important fact that has to be taken in to account when one would consider to deploy our method in a production environment is that there are several parameters that should be manually adjusted based on a specific environment or application. This notion has especially become apparent in paragraph 8.2.2.3, and it contradicts the desired goal that was described in paragraph 7.2.3, namely to have a system that does not require any parameters that should be configured manually and that relate to properties of the training data that are unknown beforehand. On the other hand, it could be possible to automate the adjustment of these parameters. With this in mind, the system as is can be seen as a solid foundation that lends itself for extensions that include even more extensive self-learning capabilities. A brief overview of such extensions is included in the next and last paragraph of this report, which is about the future work that can be done to improve the current system and the opportunities to research remaining uncertainties.

This brings us to the answers on the research questions. Since our system, despite its limitations, proved to be a well performing method to cope with the challenges of legitimate change detection and retraining, and given the uniqueness of our approach, we argue that we can advocate that our system can offer an

acceptable solution to these challenges in general. With this we do not imply that our system is the embodiment of the most optimal solution in web-based anomaly detection. Instead, our method is *a* solution of which we have proved that it can be effective under certain circumstances. Being based on several methods that are individually able to cope with some of the challenges related to anomaly-based intrusion detection, we have designed a system that would theoretically be superior to using each single method independently: the whole is greater than the sum of its parts. Even though the system that was created for this research lacks certain features that, in case they would have been implemented, would have improved the effectivity with respect to the challenges, we believe we can make the well-founded assertion that our system, given its current results and its opportunities for optimization, can serve as *a solution*, which enables us to answer the research questions based on this solution. The answers will then be as follows.

WHEN CAN AN OBSERVED CHANGE BE CONSIDERED TO BE LEGITIMATE?

The method with which to determine whether a change is legitimate, depends on the entity that is subject to this assessment. For *whitelisted web resources, parameter names, and regular parameter values,* an observed change can be considered to be legitimate when the newly observed entity that represents the change has a sufficient reputation index after a certain period of time. The reputation index is based on the variety of clients as well as the confidence indexes of the clients that accessed the new entity in the monitored environment. The confidence index of a client is a measure for how well a client can be trusted, i.e. to what extent the client can considered to be non-malicious, and depends on the variety of entities that was accessed by the client in the past.

For *irregular parameter values*, there are different gradations of change, which are specified by pre-defined thresholds. Minor change is automatically considered to be legitimate and incorporated in the model on a per-request basis, directly after detection. Instances of moderate change (suspicious items) and major change (suspicious-anomalous items) are evaluated at certain intervals by using the clustering algorithm that was described in this report, and in the following situations an entity will be considered to be legitimate:

- The requests belong to a cluster that has an exemplar that was already legitimate, or:
    - The exemplar is *suspicious* and the cluster is either large and dense enough *or* has a sufficient reputation index.
    - The exemplar is *suspicious-anomalous* and the cluster is large and dense enough *and* has a sufficient reputation index.

WHEN A LEGITIMATE CHANGE HAS BEEN DISTINGUISHED, HOW TO UPDATE THE MODEL?

There are different types of models. For *whitelisted web resources* the new web resource is simply whitelisted. For new *parameter names*, the parameter name itself is whitelisted, and the corresponding values that were monitored are either used to create a *regular parameter* values model or an *irregular parameter* values model, depending on the number of different special characters in the batch of values. For new *regular parameter values* that belong to existing parameter names it is first determined whether the existing regular model should either be updated or replaced, based on the fraction of newly observed requests that are considered to be valid by the existing model. When the new requests deviate too much from the existing model, the model is replaced instead of updated. Finally, for new *irregular parameter values* belonging to existing parameter names, the clustering algorithm will output a new set of clusters, of which the illegitimate clusters will be removed based on the criteria that were listed earlier.

ARE CERTAIN TRAINING ALGORITHMS MORE PREFERABLE THAN OTHERS WHEN IT COMES TO THE PROPOSED METHODS FOR DISTINGUISHING LEGITIMATE CHANGES AND RETRAINING THE MODEL?

When we talk about *training algorithms in our proposed methods*, this is mainly about the calculations that are performed in the trusted client system and the clustering method. For the trusted client system we are not aware of any methods that could outperform our current method with respect to the properties that we deem important in calculating “trust”, and the ability with which the relative importance of these properties can be adjusted by means of tweaking the variables of the algorithm.

For the clustering algorithm however, we have pointed out that the choice of using the *Affinity Propagation (AP)* algorithm as opposed to for example the DBSCAN algorithm, was a deliberate decision, because *AP* is more suitable for adapting to a changing environment (refer to paragraph 6.3.1).

Finally, we have noted the importance of the system’s autonomicity. The algorithms that we introduced can certainly be improved, in such a way that they would be able to automatically adjust the system’s parameter values for different applications, or for a certain application that grows in time, with respect to the number of different entities and clients. In such a way, these more autonomous algorithms would be more preferable.

HOW CAN A WEB-BASED ANIDS AUTONOMOUSLY ADAPT TO LEGITIMATE CHANGES IN THE MONITORED WEB APPLICATION?

Our proposed method, based on which we have created the web-based ANIDS named *Scandax*, proved to be fairly able to autonomously adapt to legitimate changes in the monitored web application. Improvements are possible to make the system even more autonomous.

9.2 Future work

When we consider the system's limitations that were described in paragraph 8.4.1, and the possibilities for improvement that were described in the previous paragraph, we can deduce that there exist several opportunities for future work that builds on our research. In this final paragraph we will handle some of the aspects that were described in the aforementioned paragraphs, as well as other insights that we have gained during our research and which lend themselves for further research.

9.2.1 AUTOMATIC DEDUCTION OF SYSTEM PARAMETER VALUES

We would like to start off by suggesting a possibility to improve the system itself. As we have seen in paragraph 8.2.2.3, the optimal values of the variables that are related to the trusted client system are dependent on the application's size and diversity (i.e. number of entities and clients), and they can therefore be different among different applications, as well as vary over time for a specific application. Currently these values have to be manually specified, and although we were able to provide a guideline for the values based on the application and training data sets that we used, it would be more convenient when the number of required manual operations is minimized. In order to make the system autonomous in such a way that it is able to automatically deduce the values of the variables from the state of the application, it is first required that the relation between the variables and the system's size and diversity is determined. This relation can for example be a linear function, that can subsequently be applied to let the system automatically calculate the parameter values, making it more autonomous. Therefore, as future work this relationship can be determined and can be translated to a self-update algorithm.

9.2.2 NEGATIVE CONFIDENCE INDEXES

Another opportunity to improve the system has to do with the assignment of confidence indexes to clients. Our system only assigns a confidence index to a client when it has determined that the client can be trusted up to a certain level, with at the lowest level those clients that are inactive or those that are new in the system. However, there currently is no means to identify the clients that should be even less trusted than these kinds of clients. As such, it is currently not possible to distinguish a client that has previously attacked the system, i.e. a client that has been requesting suspicious items in the past that were not subsequently marked as legitimate, but instead seen as anomalous. In these cases, it may be desirable to be able to distinguish these clients, and also incorporate this data in the reputation index calculations. One way to achieve this is to impose a penalty on the confidence index of a client whenever the client requested anomalous items. In addition, it should then become possible for a client to be associated with a negative confidence index. In addition to making the reputation index calculations more precise and effective, an additional benefit of this approach is that it could be further extended by blocking certain IP addresses when their confidence index drops below a certain threshold, although this process would probably have to take place under the supervision of a system administrator.

9.2.3 FURTHER SANITIZING THE TRAINING DATA

A third aspect that could be improved in order to optimize the system's effectivity has to do with the sanitization of the training data. The trusted client approach, described in paragraph 6.1.1.1, is used in [33] to sanitize training data. On the other hand, we use this approach only for self-adaptation in production-mode. To mitigate the effect of attacks polluting the training data, the method could also be used in our system immediately after the training has been completed, so that the training data is sanitized further. For our whitelisted model structure, i.e. the whitelisted web resources and corresponding parameter names, we could simply create a cluster from all the requests that were made to a certain web resource or parameter name and then use the reputation index to determine whether to include the entity in the final model structure. However, attacks can also be apparent in parameter values, such as SQL injection in a valid input field. Our regex models for regular parameters are built from clusters of "changes in parameter values", consisting of the requests to a known parameter name, with possibly different parameter values that were observed during the "alert delay". Therefore, the reputation index for this cluster will apply to all parameter values in the cluster. This is the reason why the regex model is constructed from all requests from the cluster when the cluster has been labelled as legitimate. This makes the regex approach not suitable for sanitizing the training data by using the trusted-clients approach, because instead of having a reputation index for all values, there should be different indexes for the attack-related values and the legitimate values. One approach that is suitable here is a cluster based approach, such as the one we use for the irregular parameters and which is described in paragraph 6.3. This is because the deviating values from the attacks would create small clusters, such that a low reputation index is assigned only to this batch of malicious requests, such that they can be filtered out.

9.2.4 EXPANDING THE INTER AND INTRA-PROTOCOL SCOPE

One of the system's limitations that we have described is that the system does not support every type of HTTP request. For HTTP requests with a "POST" request method, only requests with a content-type of "application/x-www-form-urlencoded" are processed, while for example XML and binary encoded multipart/form-data requests are ignored. The system could be extended so that it will be able to handle the available content-types within the HTTP protocol. However, one could also think of modifying our system so that it is able to work for different protocols. Most likely, it will mainly be the clustering method that will be applicable to model the characteristics of traffic from other protocols, although the regex approach could be used to introduce stricter models on certain values that correspond to packet header information. Future research could be performed that investigates the applicability of our proposed system to other protocols in the application layer, as well as in different network layers.

9.2.5 EFFICIENT TRAINING

We end this report by mentioning a modification of the AP clustering algorithm, which we found during the last stages of our research in a recently published paper [50]. In our system, for the initial training phase the AP algorithm will at a certain point in time (i.e. when the training period has ended) process all requests per

irregular parameter that were observed during the complete training period. This is a rather resource-intensive operation, especially for large volume applications. The paper that we mentioned proposes a modification to the AP algorithm which would make it more efficient, especially in situations where large amounts of data need to be clustered. We leave a detailed explanation of these improvements and an analysis of their practicality for our proposed system for future work.

REFERENCES

[1] García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security, 28(1-2), 18–28. doi:10.1016/j.cose.2008.08.003

[2] Gascon, H., Orfila, A., & Blasco, J. (2011). Analysis of update delays in signature-based network intrusion detection systems. Computers and Security, 30(8), 613–624. doi:10.1016/j.cose.2011.08.010

[3] AlNabulsi, H., Alsmadi, I., & Al Jarrah, M. (2013). Textual Manipulation for SQL Injection Attacks. International Journal of Computer Network and Information Security, 6(1), 26–33. doi:10.5815/ijcnis.2014.01.04

[4] Warneck, B. (2007). Defeating SQL Injection IDS Evasion, SANS Institute.

[5] Sadeghian, A., Zamani, M., & Ibrahim, S. (2013). SQL injection is still alive: A Study on SQL injection signature evasion techniques. Proceedings - 2013 International Conference on Informatics and Creative Multimedia, ICICM 2013, 265–268. doi:10.1109/ICICM.2013.52

[6] Richard, M. (2001). “Intrusion Detection FAQ: Are there limitations of Intrusion Signatures?”. https://www.sans.org/security-resources/idfaq/limitations.php

[7] Estevez-Tapiador, J. M., Garcia-Teodoro, P., & Diaz-Verdejo, J. E. (2004). Anomaly detection methods in wired networks: A survey and taxonomy. Computer Communications, 27, 1569–1584. doi:10.1016/j.comcom.2004.07.002

[8] R. M, "What is Land attack," 2014. [Online]. Available: http://itzecurity.blogspot.nl/2014/05/what-is-land-attack.html.

[9] Demertzis, K., & Iliadis, L. (2014). A Hybrid Network Anomaly and Intrusion Detection Approach Based on Evolving Spiking Neural Network Classification, 441, 11–23. doi:10.1007/978-3-319-11710-2

[10] Kumar, S. (2007). Survey of Current Network Intrusion Detection Techniques Abstract :, 1–18.

[11] Anyanwu, L. O., Keengwe, J., & Arome, G. A. (2010). Dynamically Self-adapting and Growing Intrusion Detection System. Retrieved February 26, 2015, from http://www.sersc.org/journals/IJMUE/vol5_no3_2010/2.pdf

[12] Gyanchandani, M., Rana, J., & Yadav, R. (2012). Taxonomy of Anomaly Based Intrusion Detection System: A Review. Neural Networks, 2(12), 1–13. Retrieved from http://www.ijsrp.org/research-paper-1212/ijsrp-p1232.pdf

[13] Zolotukhin, M., & Hämäläinen, T. (2013). Detection of anomalous HTTP requests based on advanced N-gram model and clustering techniques. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8121 LNCS, 371–382. doi:10.1007/978-3-642-40316-3_33

[14] Zolotukhin, M., Hämäläinen, T., Kokkonen, T., & Siltanen, J. (2014). Analysis of HTTP Requests for Anomaly Detection of Web Attacks. 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, 406–411. doi:10.1109/DASC.2014.79

[15] Bolzoni, D. (2009). Revisiting Anomaly-based Network Intrusion Detection Systems (pp. 1–141). doi:10.3990/1.9789036528535

[16] Zolotukhin, M., Hämäläinen, T., & Juvonen, A. (2012). Online anomaly detection by using N-gram model and growing hierarchical self-organizing maps. IWCMC 2012 - 8th International Wireless Communications and Mobile Computing Conference, 47–52. doi:10.1109/IWCMC.2012.6314176

[17] Anyanwu, L. O., Keengwe, J., & Arome, G. A. (2010). Dynamically Self-adapting and Growing Intrusion Detection System.

[18] Sivatha Sindhu, S. S., Geetha, S., & Kannan, A. (2012). Decision tree based light weight intrusion detection using a wrapper approach. Expert Systems with Applications, 39, 129–141. doi:10.1016/j.eswa.2011.06.013

[19] Estévez-Tapiador, J. M., García-Teodoro, P., & Díaz-Verdejo, J. E. (2004). Measuring normality in HTTP traffic for anomaly-based intrusion detection. Computer Networks, 45, 175–193. doi:10.1016/j.comnet.2003.12.016

[20] Chang, W., Lee, C., & Lin, C. (2004). A Revisit to Support Vector Data Description (SVDD).

[21] Le, T., Tran, D., Ma, W., & Sharma, D. (2012). A Unified Model for Support Vector Machine and Support Vector Data Description, 10–15.

[22] Tax, D. M. J., & Duin, R. P. W. (2004). Support Vector Data Description. Machine Learning, 54, 45–66. doi:10.1023/B:MACH.0000008084.60811.49

[23] Maggi, F., Robertson, W., Kruegel, C., & Vigna, G. (2009). Protecting a moving target: Addressing web application concept drift. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5758 LNCS, 21–40. doi:10.1007/978-3-642-04342-0_2

[24] Hossain, M., & Bridges, S. M. (2001). a Framework for an Adaptive Intrusion, (Graham 1999).

[25] Yang, M., Zhang, H., Fu, J., & Yan, F. (2004). A Framework for Adaptive Anomaly Detection. Network and Parallel Computing, (90104005), 443–450. doi:10.1007/978-3-540-30141-7_62

[26] Vliet, F. Van. (2006). Turnover Poseidon: Incremental Learning in Clustering Methods for Anomaly based Intrusion Detection. Proc. of the 4th Twente Student Conference on IT.

[27] Leroy, K., & Iii, I. (2007). Anomaly Detection for HTTP Intrusion Detection : Algorithm Comparisons and the Effect of Generalization on Accuracy, (May).

[28] Cretu-Ciocarlie, G. F., Stavrou, A., Locasto, M. E., & Stolfo, S. J. (2009). Adaptive anomaly detection via self-calibration and dynamic updating. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5758 LNCS, 41–60. doi:10.1007/978-3-642-04342-0_3

[29] Li, P., Gao, D., & Reiter, M. K. (2009). Automatically adapting a trained anomaly detector to software patches. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5758 LNCS, 142–160. doi:10.1007/978-3-642-04342-0_8

[30] Stavrou, A., Cretu-Ciocarlie, G. F., Locasto, M. E., & Stolfo, S. J. (2009). Keep your friends close: the necessity for updating an anomaly sensor with legitimate environment changes. Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence, 39–46. doi:10.1145/1654988.1655000

[31] Kim, S. I., Nwanze, N., & Kintner, J. (2010). Towards dynamic self-tuning for intrusion detection systems. Conference Proceedings of the IEEE International Performance, Computing, and Communications Conference, 17–24. doi:10.1109/PCCC.2010.5682339

[32] Le, M., Stavrou, A., & Kang, B. B. (2012). DoubleGuard: Detecting intrusions in multitier web applications. IEEE Transactions on Dependable and Secure Computing, 9(4), 512–525. doi:10.1109/TDSC.2011.59

[33] Pereira, H., & Jamhour, E. (2013). A clustering-based method for intrusion detection in web servers. 2013 20th International Conference on Telecommunications, ICT 2013. doi:10.1109/ICTEL.2013.6632070

[34] Wang, W., Guyet, T., Quiniou, R., Cordier, M. O., Masseglia, F., & Zhang, X. (2014). Autonomic intrusion detection: Adaptively detecting anomalies over unlabeled audit data streams in computer networks. Knowledge-Based Systems, 70, 103–117. doi:10.1016/j.knosys.2014.06.018

[35] Zhang, X., Furtlehner, C., Germain-Renaud, C., & Sebag, M. (2014). Data stream clustering with affinity propagation. IEEE Transactions on Knowledge and Data Engineering, 26(7), 1644–1656. doi:10.1109/TKDE.2013.146

[36] Jagale, R. S., & Naoghare, P. M. M. (2015). Intrusion Detection System for Multitier Web Based Application, 4(1), 374–382.

[37] Leroy, K., & Iii, I. (2007). Anomaly Detection for HTTP Intrusion Detection : Algorithm Comparisons and the Effect of Generalization on Accuracy, (May).

[38] Denning, D. E. 1987. An intrusion detection model. IEEE Trans. Softw. Eng. 13, 2, 222–232.

[39] Chrome V8. Obtained from https://developers.google.com/v8/ (2015).

[40] Panda, D. K., & Science, I. (2002). NIC-based intrusion detection: A feasibility study.

[41] Crawljax. Crawling ajax-based web applications. Obtained from http://crawljax.com/ (2015).

[42] Brusco, M. J., & Köhn, H.-F. (2008). Comment on "Clustering by passing messages between data points". Science (New York, N.Y.), 319(5864), 726; author reply 726. http://doi.org/10.1126/science.1151268

[43] Justin Crist (2007), Web Based Attacks, SANS Institute, as part of the Information Security Reading Room, http://www.sans.org/reading_room/whitepapers/application/web-based- attacks_2053

[44] Kruegel, C. and Vigna, G. 2003. Anomaly detection of Web-based attacks. In Proceedings of the 10th ACM Conference on Computer and Communications Security. ACM Press, 251–261.

[45] Lampesberger, H., Winter, P., Zeilinger, M., & Hermann, E. (2012). An on-line learning statistical model to detect malicious web requests. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 96 LNICST, 19–38. http://doi.org/10.1007/978-3-642-31909-9_2

[46] OWASP, T. (2013). Top 10–2013. The Ten Most Critical Web Application Security Risks.

[47] Gu, G., Fogla, P., Dagon, D., & Skori, B. (2006). Measuring Intrusion Detection Capability: An Information-Theoretic Approach. Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, 90–101.

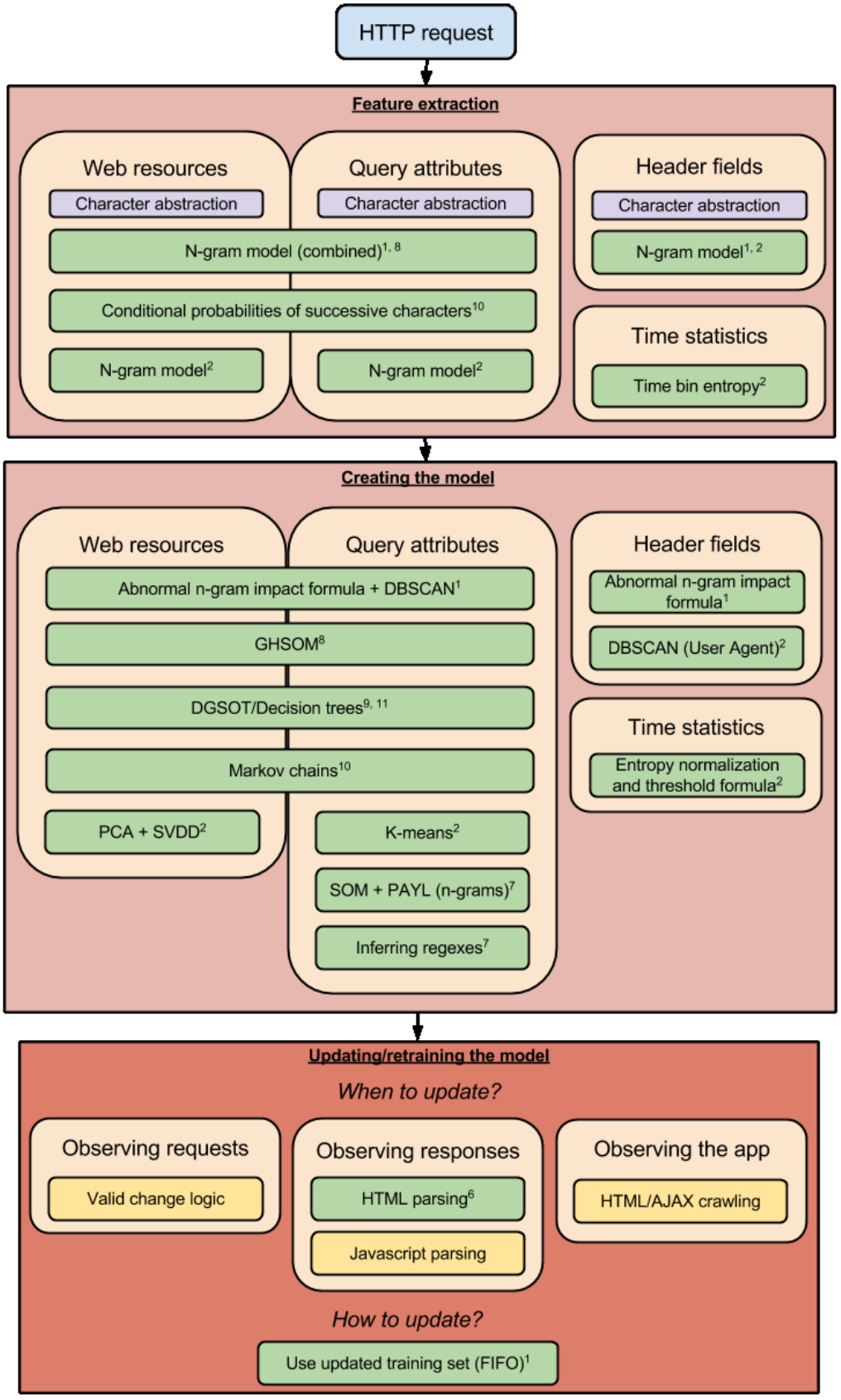
[48] sklearn AffinityPropagation Algorithm documentation (2016). http://scikit-learn.org/stable/modules/generated/sklearn.cluster.AffinityPropagation.html

[49] stackoverflow discussion: "Affinity Propagation (sklearn) - strange behavior" (2015). http://stackoverflow.com/a/30849681

[50] Serdah, A. M., & Ashour, W. M. (2016). Clustering Large-Scale Data Based on Modified Affinity Propagation Algorithm, 6(1), 23–33.

APPENDIX A

Overview of state-of-the-art web based A-NIDS techniques for feature extraction and model creation, as well as proposed methods for updating the model. The numbers in superscript can be ignored.

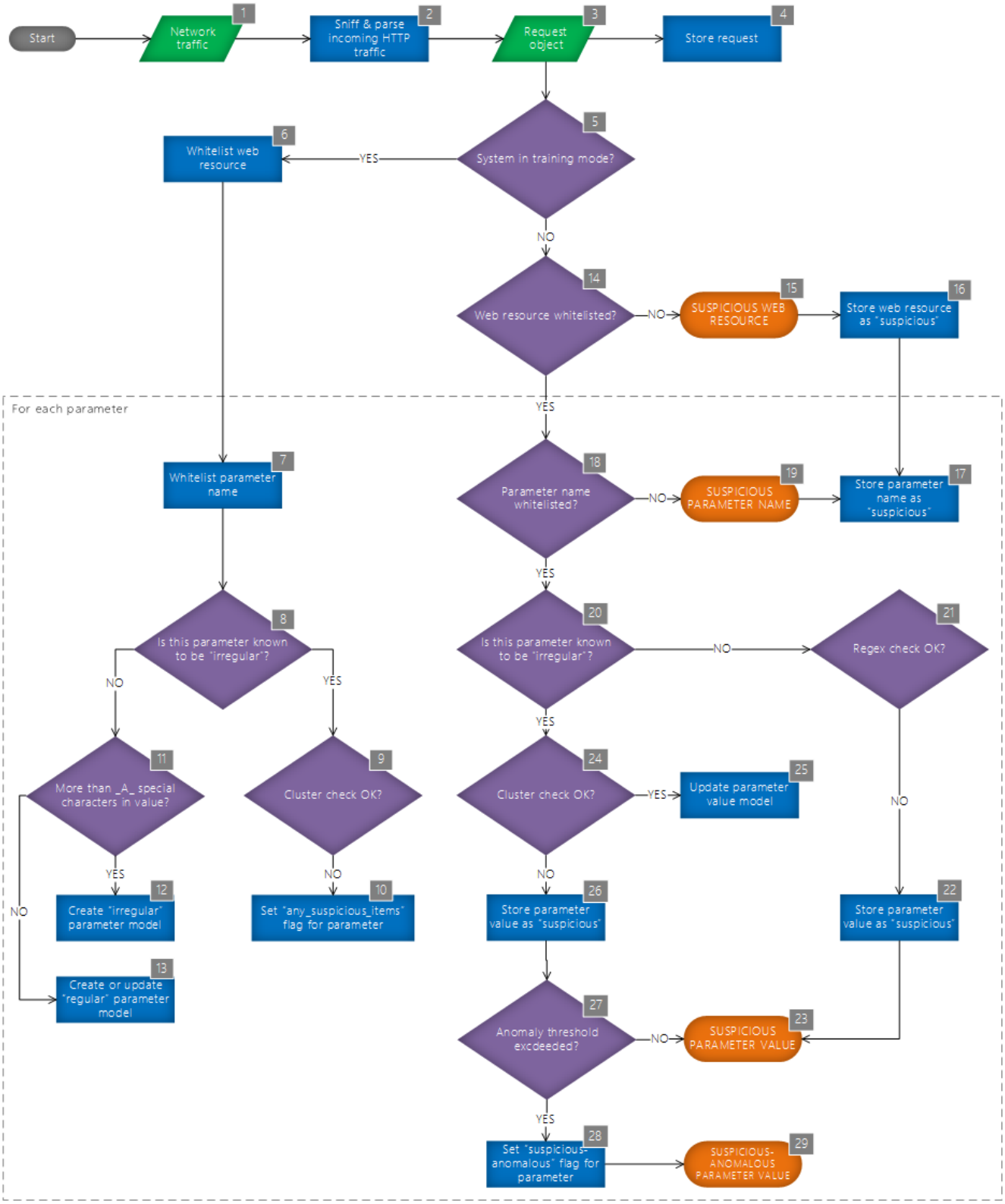


APPENDIX B

Flowcharts of the core system components of Scandax.

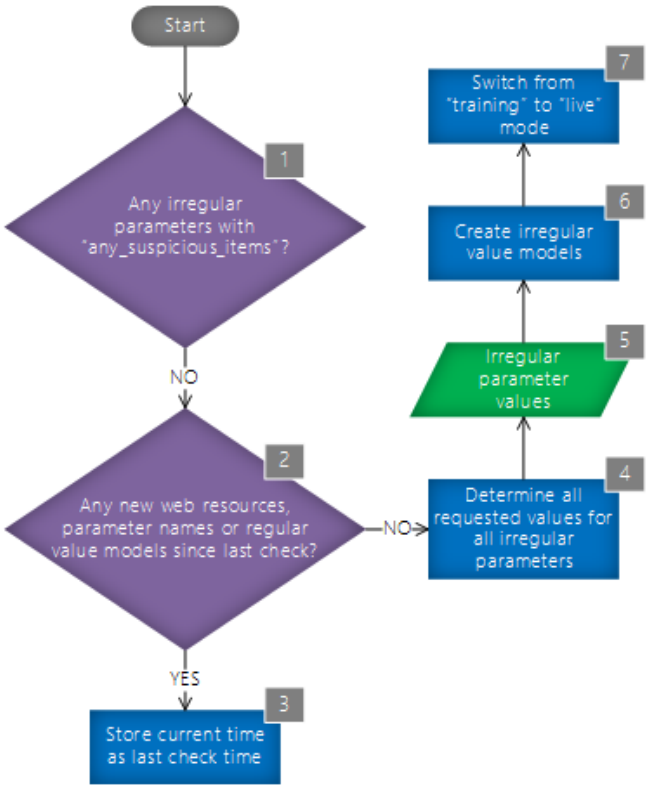
B.1 Main system

The main training and detection system.



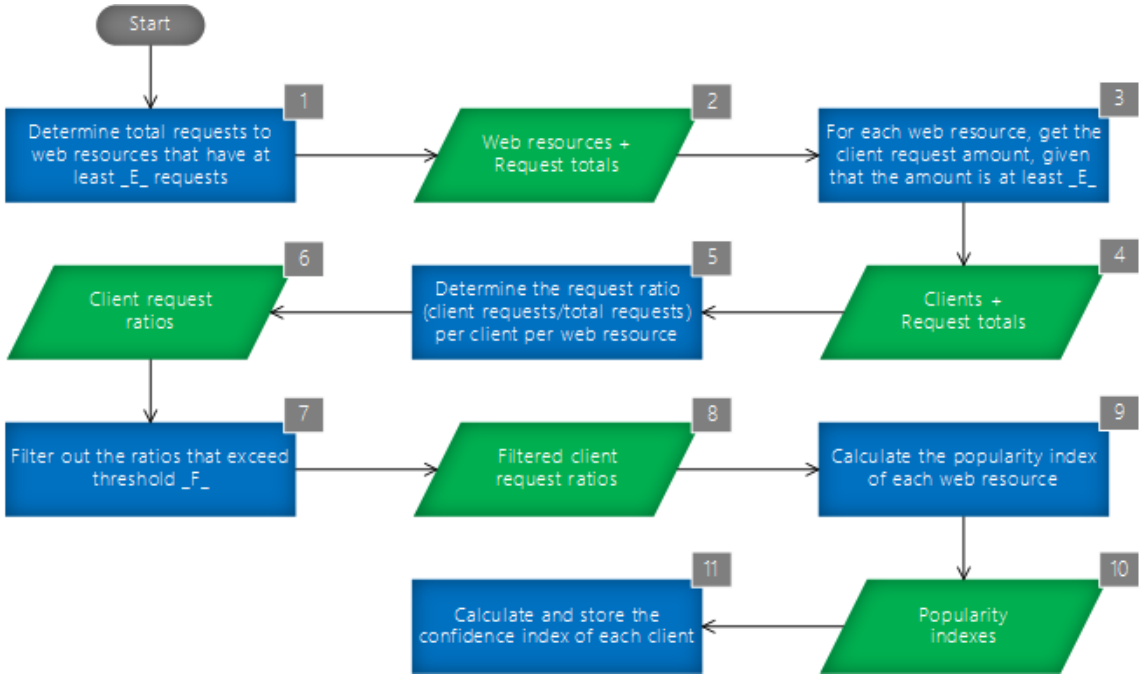
B.2 Training monitor

The part of the system that is responsible for determining whether the initial training is completed and that creates the final models for the irregular parameter values.



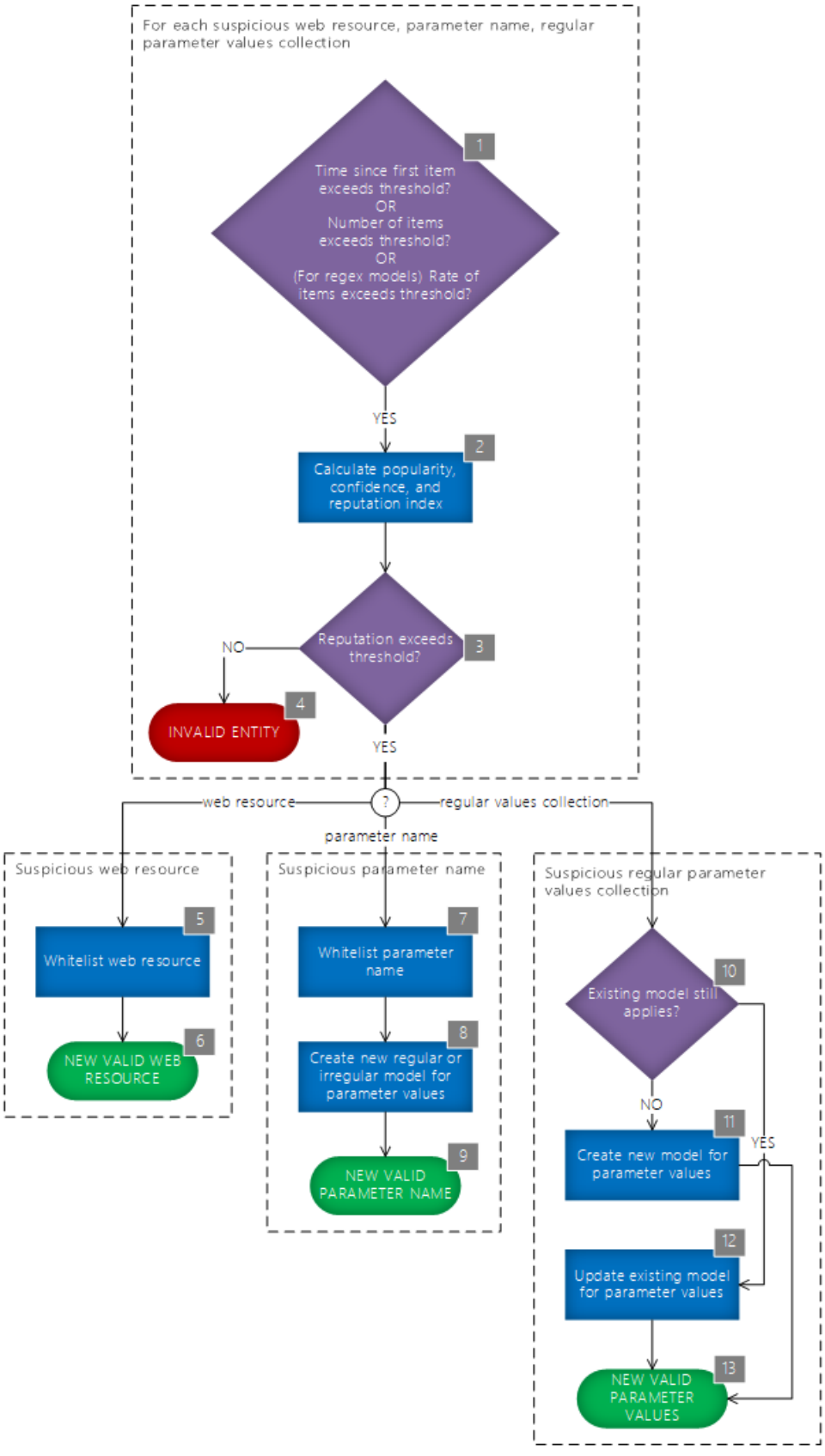
B.3 Trusted client monitor

The part of the system that periodically updates the confidence index of each client.



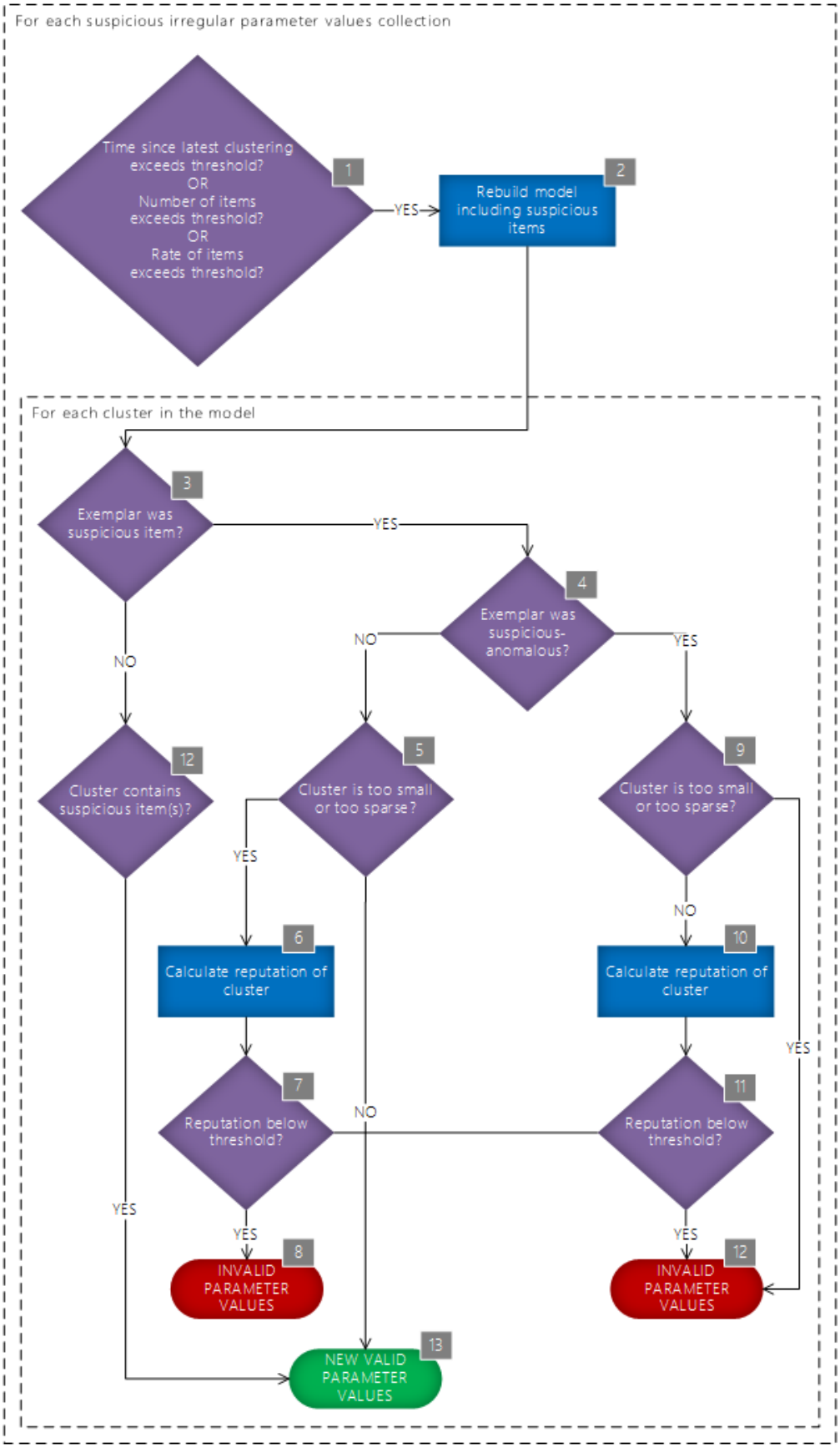
B 4.1 Suspicious entity monitor (Part I)

The first part of the system that evaluates the suspicious entities that have been observed. The entities in this part consist of web resources, parameter names and regular parameter values.



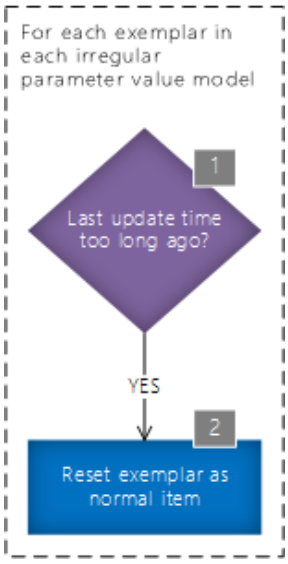
B 4.2 Suspicious entity monitor (Part II)

The second part of the system that evaluates the suspicious entities that have been observed. The entities in this part consist of irregular parameter values.



B.5 Forgetting window monitor

The part of the system that periodically resets outdated cluster exemplars.



APPENDIX C

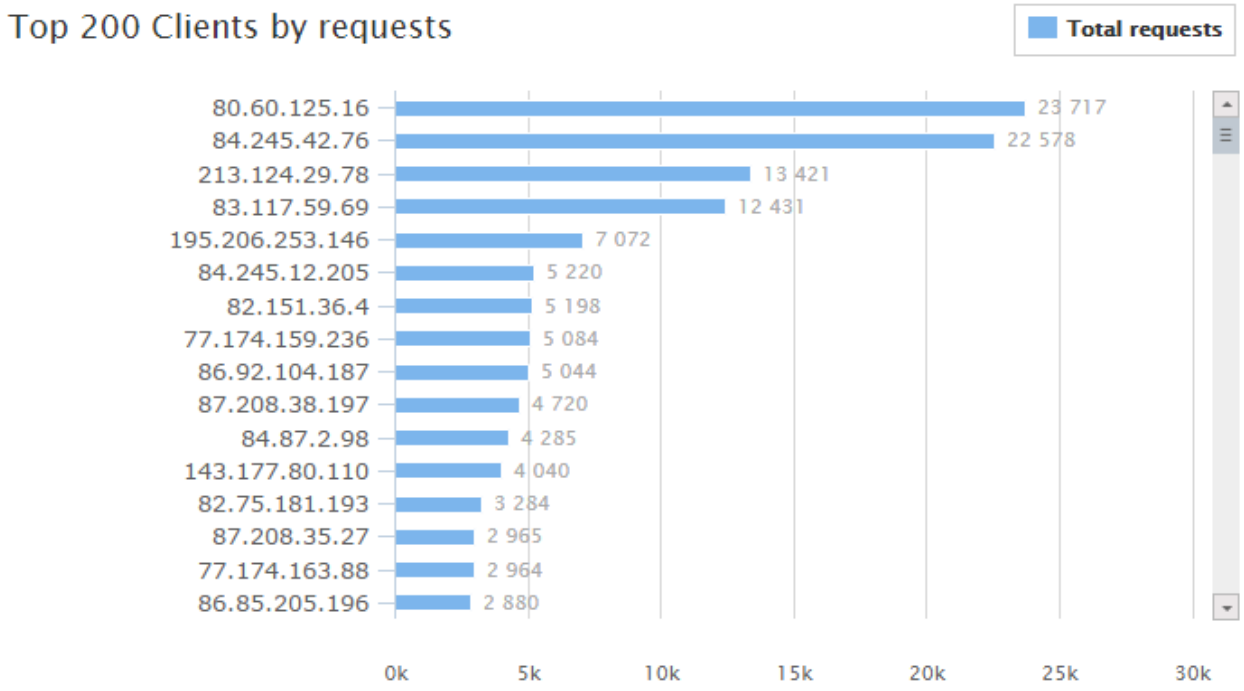
Custom variables for Scandax, including explanations as well as the values that were used for generating the results in this report.

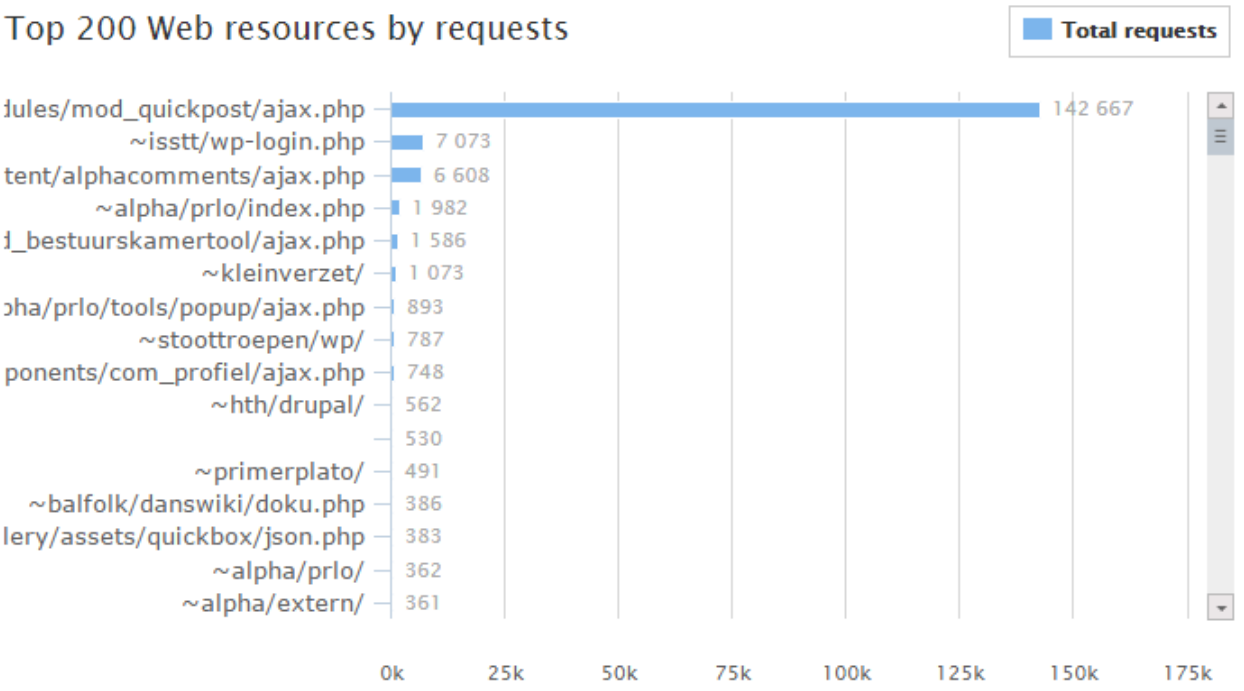
Variable	Description	Value
Main system variables		
_A_	The minimum number of different special characters that should be in the parameter value before the parameter is seen as *irregular*, i.e. when the clustering algorithm will be used to construct the model instead of the regex approach, which is used for *regular* parameter values. Refer to [15].	6
_B_	The minimum distance between an incoming item and the nearest exemplar in order for the item to be labeled as "suspicious". The threshold is also used to post-process the re-clustering operation. When the mean distance between an exemplar and its items is greater than this amount, the cluster of the exemplar of the irregular parameter is flagged as "suspicious" after the re-clustering operation Refer to [34].	0.1
_C_	The minimum distance between an item and the exemplar of a cluster in order for the item to be labeled as "anomalous". This variable is only applicable when running the system in *plain* mode, i.e. without the trusted client approach for self-adaptation to concept drift. Refer to [34].	0.2
_D_	The default preference that an item is chosen as an exemplar. This variable is used when constructing the similarity matrix for the Affinity Propagation clustering algorithm. Refer to [34].	-5
TrustedClientMonitor variables		
_E_	Minimum number of requests for a web resource from a client before the requests are included in the trust index calculations.	3
_F_	Clients with a request ratio higher than this bound will be filtered out (to filter out DoS attacks). Refer to [33]. In addition to mitigating the effect of DoS attacks on the confidence index calculations, this parameter can actually also be used to impose a restriction on the minimum number of clients that is required for a web resource to obtain a popularity index of 1. For example, setting this parameter to 0.1 means that at least 10 clients need to request a web resource before the popularity of the web resource reaches 1. At least two clients need to request the web resource in order for the web resource to be able to have a popularity index at all. However, with two clients there is only a small chance that the web resource will have a nonzero popularity index. This is only the case when one of the clients has a request rate of 0.9 or higher, which is relatively unlikely.	0.5
_G_	An array containing the weights assigned to the entity's popularity and the entity's confidence index respectively. This is used when computing the reputation index for an entity. Note that the sum of these weights should always be 4. Refer to [33].	[1, 3]
SuspiciousEntityMonitor variables		

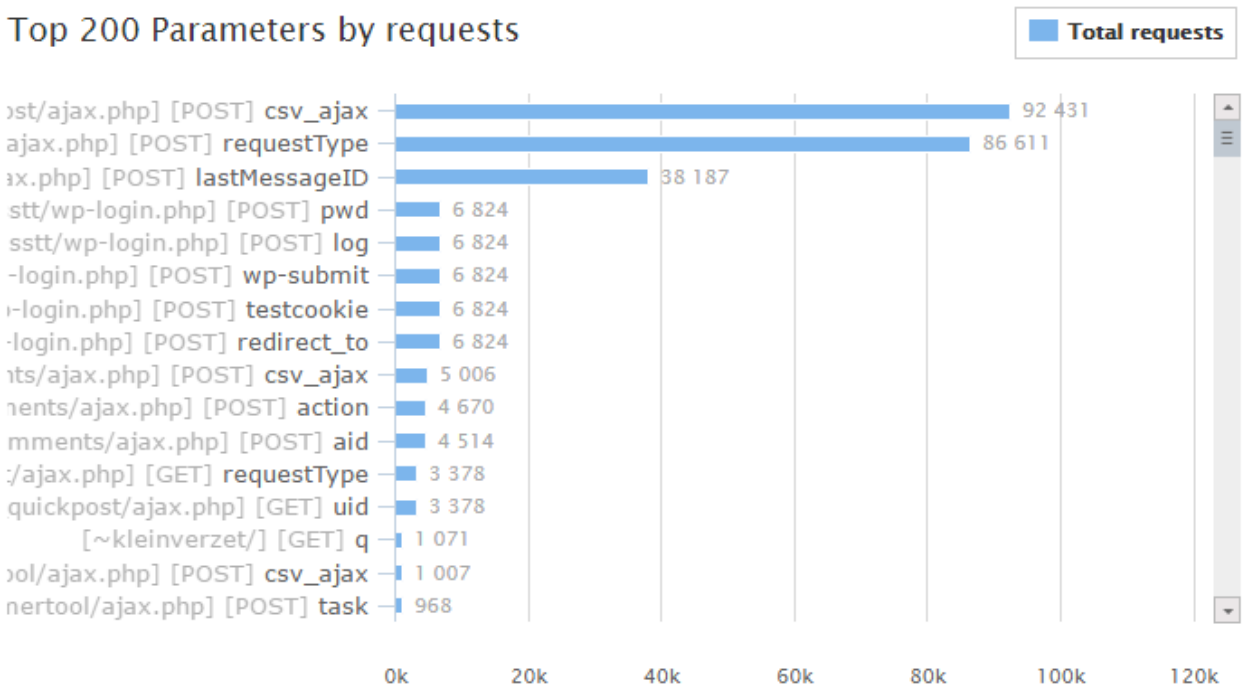
_H_	Time in seconds after which a newly observed web resource is evaluated for validity, i.e. after which the web resource is whitelisted in case it is deemed valid or an alert is raised when the web resource is deemed invalid.	43,200
_I_	Similar to _H_, but for parameter names and regular parameter values. For parameter names and regular parameter values a larger delay is used than for web resources. It is assumed that, in general, parameters are requested less often than web resources.	86,400
_J_	The reputation index of a newly observed entity (web resource, parameter name/values) should be at least this value in order for the new entity to be considered valid. Refer to [34].	1.01
_K_	At least this ratio of the total requests during the *alert delay* should be considered to be valid by the existing regular model in order for the existing regular parameter value model to be updated instead of replaced.	0.2
_L_	When the number of suspicious items for an entity exceeds this threshold, evaluation of the suspicious entities (re-clustering in case of irregular parameter values) is forced. Refer to [34].	300
_M_	When the rate of (suspicious items)/(total items since the reference time) for a regular/irregular parameter value collection exceeds this threshold, evaluation or re-clustering is forced, depending on whether the parameter is regular or irregular. This is only applicable when the total items since the reference time is at least _R_. Refer to [34].	0.6
_M2_	Minimum number of total items before the suspicious items rate (refer to _M_) may be calculated.	20
_N_	When this time window length is exceeded after the latest clustering for an irregular parameter, re-clustering is forced. Refer to [34].	2,000
_O_	When the number of items that is associated to an exemplar of an irregular parameter is less than this amount, the cluster of the exemplar is flagged as "suspicious" after the clustering operation. Refer to [34].	4
_P_	When already existing exemplars are part of a new cluster, the number of items associated to the exemplar will contribute this amount to the reputation index of the new cluster.	0.001
ForgettingWindowMonitor variables		
_Q_	Parameter for the forgetting mechanism of irregular parameter exemplars. If an exemplar has never been assigned with a single item in this time window (expressed as a number of incoming requests), the exemplar is simply reset as a common item.	5,000
TrainingMonitor variables		
_R_	When no model changes have been observed during this interval (seconds), the training is considered to be complete. Model changes are checked among all entities, i.e. whitelisted web resources, whitelisted parameter names, regular parameter values, and irregular parameter values.	3,600

APPENDIX D

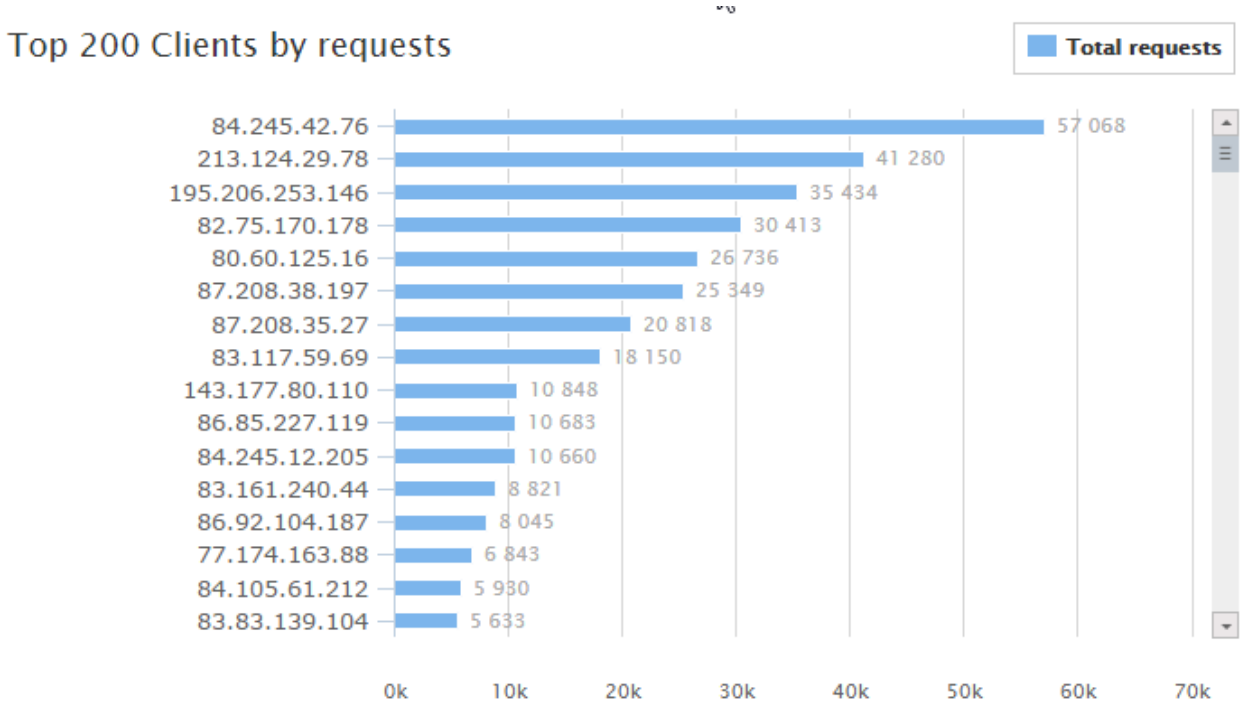
D.1 Training data set 1



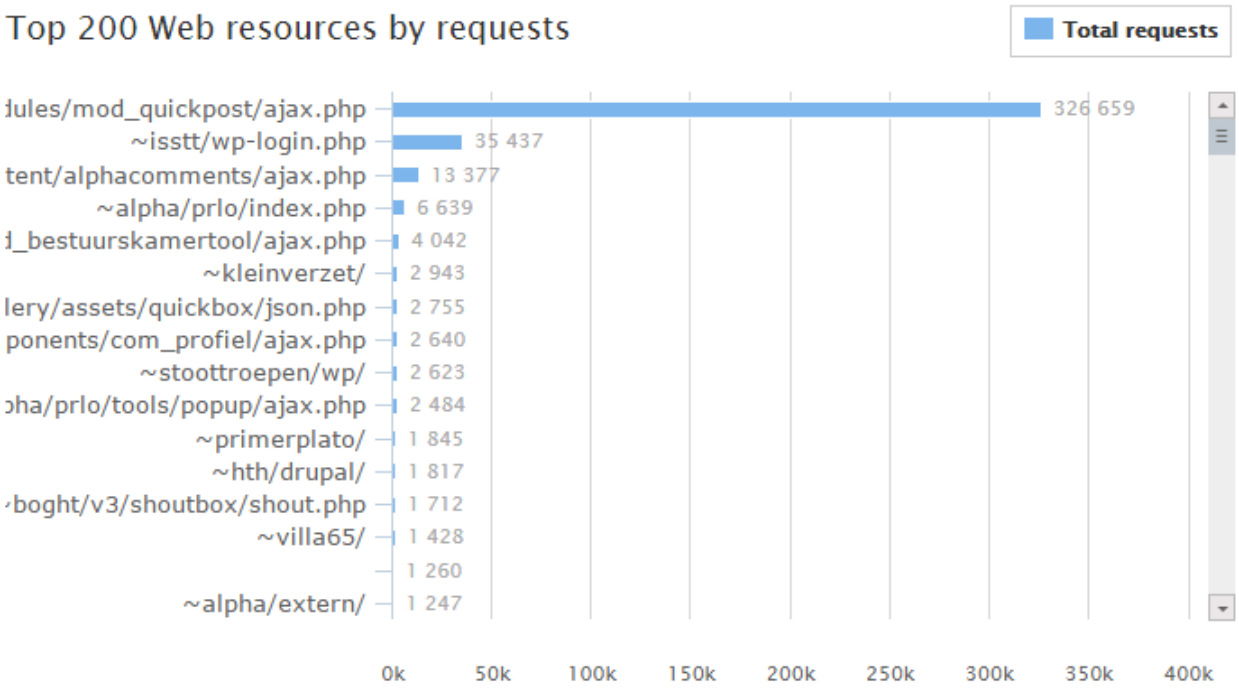




D.2 Training data set 2



Top 200 Web resources by requests



Top 200 Parameters by requests