Anomaly detection in interception proxies

Sami J. Lehtinen

School of Science

Thesis submitted for examination for the degree of
Master of Science in Technology.
Espoo 18.4.2017

Thesis supervisor

Professor Aristides Gionis

Thesis advisor

Lic.Sc. (Tech) Timo Lilja

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE
MASTER'S PROGRAMME IN COMPUTER, COMMUNICATION AND INFORMATION SCIENCES

ABSTRACT OF THE
MASTER'S THESIS

Author: Sami J. Lehtinen

Title: Anomaly detection in interception proxies

Date: 18.4.2017 **Language:** English **Number of pages**: 75

Major: Computer Science **Code:** SCI3042

Supervisor: Professor Aristides Gionis
Advisor: Lic.Sc. (Tech) Timo Lilja

Use of interception proxies is becoming more popular. They are used to audit access and enforce policies and constraints to important servers or whole network segments. The sheer amount of data captured with the devices makes fully manual pruning of the data impractical. Methods to analyze the gathered data to highlight possible attacks or problems would be valuable in freeing up administrator time and resources.

This thesis investigates the use of clustering methods to identify anomalous connections, either by identifying them as outliers or bundling them with other connections which have raised alarm in the past.

The work shows that a practical approach can be implemented with a DBSCAN-based clustering method, but concluded that an unsupervised approach is not enough. As a semisupervised method the system can have value in production environments.

Keywords: intrusion detection, clustering, unsupervised learning

AALTO-YLIOPISTO
PERUSTIETEIDEN KORKEAKOULU
TIETO-, TIETOLIIKENNE- JA INFORMAATIOTEKNIIKAN MAISTERIOHJELMA

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Sami J. Lehtinen

Työn nimi: Poikkeamien havainnointi sieppausvälityspalvelimissa

Päivämäärä: 18.4.2017 **Kieli:** Englanti **Sivumäärä**: 75

Pääaine: Tietotekniikka **Koodi:** SCI3042

Työn valvoja: Professori Aristides Gionis
Työn ohjaaja: TkL Timo Lilja

Sieppausvälityspalvelimien käyttö on yleistymässä. Niitä käytetään käytäntöjen ja rajoitusten täytäntöönpanossa sekä kriittisten palvelimien ja verkon osien käytön valvomisessa. Laitteiden kaappaaman tiedon määrä on niin valtava, että tiedon purkaminen manuaalisesti on epäkäytännöllistä. Menetelmät jotka analysoivat dataa mahdollisten hyökkäysten tai ongelmien esiin nostamiseksi olisivat hyvin arvokkaita vapauttamaan järjestelmänvalvojien aikaa ja resursseja.

Tässä työssä tutkitaan ryhmittelyalgoritmien käyttökelpoisuutta epätavallisten yhteyksien havainnoimisessa joko tunnistamalla ne poikkeaviksi, koska ne eivät kuulu mihinkään ryhmään tai asettamalla ne samaan ryhmään sellaisen yhteyden kanssa joka on todettu hälyttäväksi aiemmin.

Työssä todetaan, että käytännöllinen sovellus järjestelmästä voidaan toteuttaa käyttäen DBSCAN-pohjaista ryhmittelyalgoritmia, mutta täysin valvomattomalla lähestymistavalla ei saada riittävän hyvää tulosta. Osittain valvottuna menetelmästä voi olla hyötyä tuotantojärjestelmien valvonnassa.

Avainsanat: hyökkäysten havaitseminen, ryhmittely, valvomaton oppiminen

Acknowledgements

I would like to thank my supervisor, professor Aristides Gionis, and my advisor, Timo Lilja, for their guidance, ideas, and council in creating this work.

I would like to thank my former advisor, Antti Huima, for his guidance at the beginning of this work. Antti passed away from a long-time illness before seeing the completion. Rest in peace, Antti.

I thank SSH Communications Security Corporation for the chance to work on this thesis. Discussions with insightful colleagues helped in the direction of this work.

I owe special thanks for the efforts of Jussi Valkiainen, Jan Hlinovsky, and Antti Kettunen for proofreading and validation of this work; they helped fix a number of issues in the original manuscript.

Finally, I thank my wife, Daniela, for her patience and understanding. She has also proofread this thesis and countless other texts while I have studied in the then Helsinki University of Technology and now Aalto University. Without her support and encouragement it is doubtful I would have ever graduated.

Espoo, 15.4.2017

Sami J. Lehtinen

Contents

List of Figures

List of Tables

Chapter 1

Problem definition

1.1 Introduction

Interception proxies are systems for collecting auditing data about connections and the actions taken in them. The systems are typically installed in network chokepoints to guard access to mission-critical servers. Interception proxies can and have been used in illegitimate purposes and for spying; the term "man-in-the-middle" (MITM) applies to interception proxies in the cryptographical context. In a MITM attack, the attacker intercepts the protocol messages the participants are sending to each other, trying to fool the parties into thinking they are talking directly with each other, but instead they are communicating with the attacker. The attacker can either relay the protocol messages as is or the messages can be dropped, replaced, or modified.

In this work, we focus on the legitimate use case of interception proxies; auditing access to servers or parts of the private networks that have been walled off from the rest of the network. The auditing tool in question is CryptoAuditor; a system created by SSH Communications Security Corporation. CryptoAuditor mostly just relays the messages between the clients and servers; for access control purposes, it may respond with the appropriate protocol messages to deny the use of some channels. CryptoAuditor can also generate some messages; for example, in Remote Desktop Protocol (RDP),

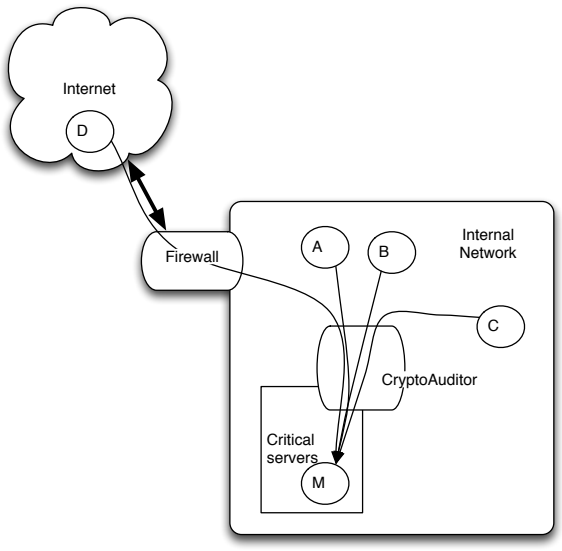

Figure 1.1: A typical CryptoAuditor deployment.

CryptoAuditor can fabricate a protocol exchange to the client to enable additional authentication messages.

The aim of this work is to find a method to help auditors and administrators using CryptoAuditor in their work; to highlight "interesting", problematic or otherwise novel connections which are not common in the gathered data. Administrators should be able to flag existing connections as benign or malicious and have similar connections classified accordingly. From a large number of connections, administrators should be able to review just a small handful of representative connections.

The chosen focus was unsupervised and semi-supervised methods. It is generally assumed, and also it has been observed first-hand, that different organizations use a wide variety of tools to help in automated administration, file transfer, provisioning, and other tasks. A supervised classifier would probably not be able to help a sufficiently large subset of the system's users.

A typical CryptoAuditor installation is shown in Figure 1.1. In the figure, machines A, B, and C in the internal network and D coming from the Internet can only access mission-critical machine M going through CryptoAuditor.

CryptoAuditor can enforce policies, audit all commands executed by the users accessing the machines, and apply an additional layer of authentication by enforcing multi-factor authentication, just to list a few of the more ordinary use cases. CryptoAuditor is just one of the several products in this market segment.

Currently, CryptoAuditor and similar products are mainly used for access control and for later forensics of connections after an incident, as they have few capabilities to automatically tag connections as "good" or "anomalous". The problem is hard because of many factors, including:

lack of real data Actual data from production systems is extremely sensitive, and customers are very unlikely to just hand over the data. Even with access to the production data, it is unlikely that the data set contains actual anomalous connections and even less likely that those connections would be labeled as such.

false positives If the system produces a lot of false positives, the number of alerts from the system will cause the administrators to ignore the alerts from the system or turn it off.

false negatives Failure to detect actual attacks. Categorizing many attacks of the same type as normal connections can also make the system actually harmful, as it can instill the administrators using the system with a false sense of security.

computational complexity Because the number of connections in the system may become very large, in the order of hundreds of thousands, the algorithms applied need to be computationally quite frugal. A quadratic algorithm, where the parameter is the number of connections, will take too much time to be of use in a production system.

Instead of just allowing post-mortem forensics analysis after a breach has happened, the system should be able to detect anomalous connections as they happen and notify the system administrators of such. A successful implementation reduces the workload of the auditors. In this work, we use a

clustering approach for outlier detection. Clustering has been used with some success in research of intrusion detection (see, for example, Leung and Leckie (2005)). An additional goal was to create a system capable of detecting an automated attack against the target servers, and as such pure outlier detection from clustering would have not been enough; our approach is a semi-supervised system with administrator review built in. The clustering algorithm used most in this work is DBSCAN, a time-tested density-based method, which is described in more detail in Section 2.3.2.

The approaches to the problem should allow extending the solution to encompass more protocols in the future. An example of such protocol is Remote Desktop Protocol, or RDP, which is discussed in Section 1.4.3.

This work relied heavily on the `scikit-learn` software package, described in Pedregosa et al. (2011). `scikit-learn` is a machine-learning toolkit, written for the Python language, which contains implementations for a large selection of algorithms used in the field with a consistent interface.

1.2 Classes of anomalies

Chandola et al. (2009) divide anomalies to the following three classes: point anomalies, context anomalies, and collective anomalies. In the context of the SSH protocol, and CryptoAuditor in particular for this project, the following examples apply:

Point anomalies The connection is anomalous by itself. For example, the connection transported a malware binary uuencoded[1] (The IEEE and The Open Group, 2016) in the terminal stream. This is something that never happens in normal connections.

Contextual anomalies The connection would otherwise appear normal, except when examined in context with other connections. An admin

[1] A process where binary data is encoded to a suitable format to be transferred as text. The word stems from "Unix-to-Unix encoding".

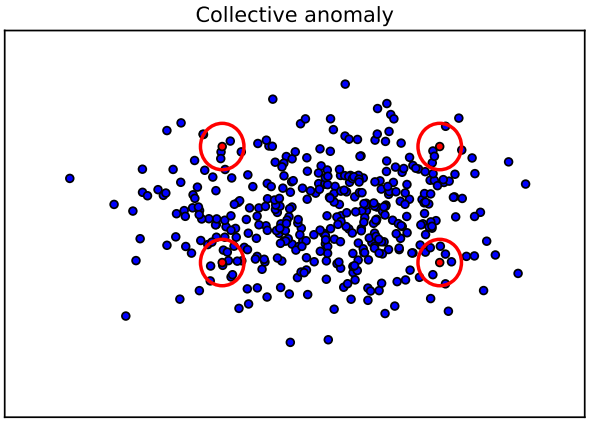

Figure 1.2: An example of connections that form a collective anomaly. The anomalous connections are highlighted.

login from a machine in the middle of a weekend night would be considered anomalous.

Collective anomalies A single connection is not anomalous but a set of connections together are. For example, a situation where a successful password authentication is preceded by a thousand failed attempts against the same username, with varying passwords and millisecond intervals.

It should be noted that the different categories outlined above are not disjoint and an anomalous connection may belong to several categories.

With a clustering approach, point anomalies and contextual anomalies should stand out as outliers or as clusters of problematic connections. Collective anomalies are not the focus of this work, and clustering offers no obvious benefits in uncovering collective anomalies. Collective anomalies can cluster well and blend in with benign connections but together they exfiltrate data or perform malicious activities. An invented example of connections that are collectively anomalous is given in Figure 1.2.

1.3 Approaches in intrusion detection

In machine learning, classification algorithms are generally divided into supervised and unsupervised methods. In unsupervised learning, training data is not necessarily required, and the input data does not have classification labels. In supervised learning, training data with the correct labels is provided, with which a model is created, and data is then classified using the created model. (Leskovec et al., 2014, p. 439)

In semi-supervised learning, we have a few labeled samples, which we can use to help in dealing with the unclassified samples.

Supervised systems are taught with labeled training data. Because such data can be very expensive, and in this case not comprehensive, these methods were not considered for this work. Intuitively, the chosen example of attack connections would be easily classifiable by supervised methods, even by using a simple naive Bayes classifier. The problem here is that in a real-life scenario, we would not be able to train the model with the sample data ahead of time, as the attack connections represent a novel attack against the system. Examples include support-vector machines, decision trees, and Bayesian classifiers.

Unsupervised methods are better in situations where the sample distribution may change over time. For example, real-life intrusion detection systems (IDS) relying on fingerprints of attacks need to be updated as new attacks or variations of old attacks are encountered. A company may introduce a new software system for backing up hosts in the environment, replacing an older system. The organization may introduce some new system for automated system administration, which has previously been handled manually. An example particularly worth mentioning here about the latter is Universal SSH Key Manager – the periodic host scans and key activity scans occur over SSH, as do all the management actions. Any classification system should be able to cope with this dynamic nature of connections with as little extra work for the administrators as possible.

Among other unsupervised methods, alternatives such as self-organizing maps (SOM) could be used. For example, Zanero and Savaresi (2004) use a

SOM-based method for intrusion detection at the TCP-packet level.

Portnoy et al. (2001) introduced an unsupervised system for anomaly detection using clustering; they used a filtered version of the 1999 KDD Cup data set (SIGKDD, 1999) instead of the full set.

Leung and Leckie (2005) again used the 1999 KDD Cup data set in creating a grid and density based clustering algorithm for outlier detection.

One problem that is encountered with unsupervised methods in general, including clustering methods, is that if the attacks form a large part of the data, they show up as the "normal" data over some benign connections. For example, when gathering data for this work, an automated attack run created clusters instead of outliers, even with a relatively small value for the parameter ϵ for DBSCAN. It is a safe assumption that automated attacks getting audited with CryptoAuditor can create clusters, while some will be outliers. This problem is dealt with in Section 2.3.1, which essentially makes the approach semi-supervised.

Our approach in this work will flag all outliers and a sampling from the detected clusters for administrator review, and labeled connections from these samples will be the basis of the classification system.

1.4 CryptoAuditor

CryptoAuditor is an interception proxy; it acts as a man-in-the-middle for decrypting the connections for a variety of protocols, namely SSH, TLS, and RDP. As outlined in Section 1.1 and in Figure 1.1, the main use case for CryptoAuditor is protecting a smaller set of business-critical hosts, access to which is monitored by CryptoAuditor. Access can be restricted in many ways; a lighter approach is configuring the target servers to allow connections only through CryptoAuditor and enforcing the connecting IP address and user credentials to ones contained in the CryptoAuditor Vault. A much more restrictive approach is routing or bridging the traffic through CryptoAuditor and dropping all other network traffic to the hosts; this allows permitting only those connections that match a certain policy and is thus the most secure option. The downside here is that if CryptoAuditor becomes unavailable,

no connections to the target servers are possible. (SSH Communications Security Corp, 2017a)

As the client can and should perform server authentication during the key exchange, introducing CryptoAuditor to the network will usually require configuration for the clients to allow it to intercept connections. If this is not done, well-behaving clients of cryptographic protocols should deny the connections. The user should explicitly allow or force the connections to proceed while the security risks associated with such an action are displayed.

CryptoAuditor architecture is depicted in Figure 1.3. It is noteworthy that in the current architecture the user interface (UI) and the Vault are always in the same machine instance, that is, a single UI governs over a single Vault, and as of this writing, there is only one Vault per CryptoAuditor installation. In the figure, IDS stands for intrusion detection system and DLP for data loss prevention system. The M and N stand for the arbitrary number of Hounds and Vaults in the three-tier architecture; a single Vault can have more than one Hound serving under it.

The Hound is the component performing the interception of the connections. It defers in policy decisions to the configurations in the Vault, either by querying it online or using a cached configuration. The Vault stores the connection trails and can run post-processing of the connections, for example, to classify the connections and alert the administrators of anomalous occurrences.

In the following sections, we describe the most important protocols audited by CryptoAuditor.

1.4.1 Secure Shell

Secure Shell, or SSH for short, is a protocol and software suite created originally for securing remote terminal access to university Unix machines. The SSH use case has traditionally been around command line and administration of remote systems; however, using graphical user interfaces tunneled over a secure SSH connection has always been popular, either with X11 or to lesser extent RDP. SSH allows the use of a multitude of authentication back ends,

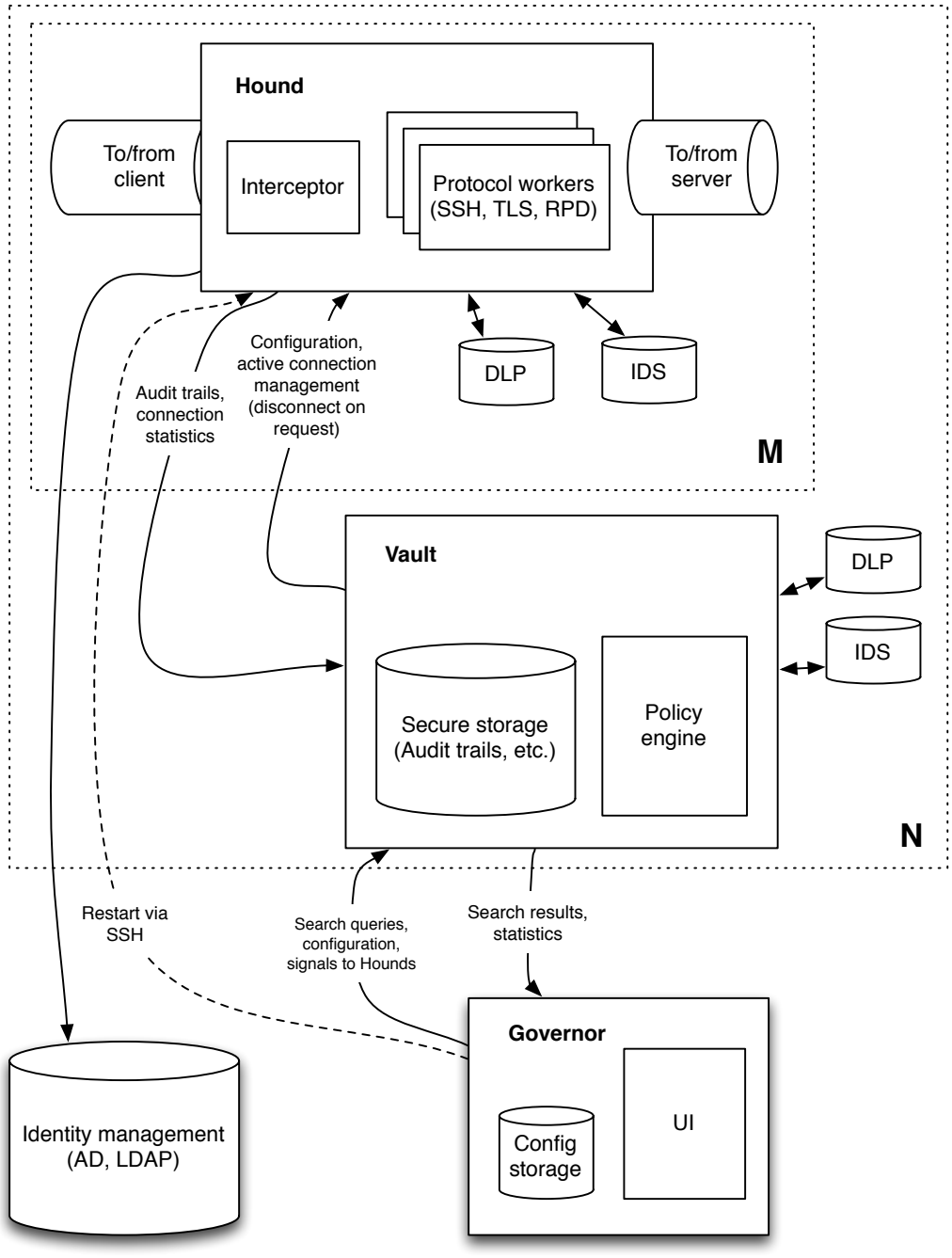

Figure 1.3: CryptoAuditor architecture.

from the standard password authentication to the popular non-interactive use of public key, along with others, such as multi-factor authentication mechanisms and Kerberos.

SSH allows multiple channels over the encrypted link, multiplexing the traffic. The same connections may be used to transfer files, pass communications over terminal or graphical user interfaces, and allow passing public-key authentication tokens towards other hosts from the target server, that is, allow SSH to be used as a single sign-on facility.

SSH software suite has traditionally also contained secure file transfer utilities. As SSH has a facility to tunnel arbitrary traffic, file transfer can be done in many other ways as well, such as directing `tar` output over SSH using command-line pipes or using `rsync` options to enable transport over SSH.

The protocol is an Internet Engineering Task Force (IETF) standard, described in Ylönen and Lonvick (2006a), Ylönen and Lonvick (2006b), Ylönen and Lonvick (2006c), Ylönen and Lonvick (2006d), and Lehtinen and Lonvick (2006).

1.4.2 TLS

TLS, or Transport Layer Security, is a communications protocol allowing two parties to communicate securely over the Internet. TLS is the current version of the protocol that originated as SSL, or Secure Sockets Layer, by Netscape. SSL 3.0 is sufficiently broken to have been formally deprecated by the IETF, with the rationale and method documented in Barnes et al. (2015). SSL 2.0, even more insecure, was deprecated earlier, with the process described in Turner and Polk (2011).

SSH and TLS have similar security characteristics. There are some differences, channel architecture and end-user authentication being the most notable examples. TLS is used as a security layer for application protocols, like HTTPS, whereas SSH also contains a full application protocol for remote administration.

The current version of the protocol, TLS 1.2, is described in Dierks and

Rescorla (2008).

1.4.3 RDP

RDP, or Remote Desktop Protocol, is a protocol typically used in administration of hosts with the Windows operating system. There are also other applications for the protocol; for example, the VirtualBox hypervisor (Oracle Corporation, 2017) can be managed with RDP. In RDP, a graphical user interface is exposed to the user instead of a command line. A drawback with the use of this protocol is that the text seen on the screen consists mostly of bitmaps rendered in the server side, as described in Microsoft Corporation (2016a)[Section 1.3.6, Basic Server Output]. Therefore, when this protocol is used, optical character recognition (OCR) techniques are required to be able to search through the audited textual screen contents. For encrypting the protocol, older versions of RDP used a proprietary scheme, while new versions are secured with TLS, documented by Microsoft Corporation (2016a)[Sections 5.3, Standard RDP Security and 5.4, Enhanced RDP Security]. RDP security has improved over the recent years, and current versions are secure by default, with good measures for mutual authentication.

1.5 Universal SSH Key Manager

Universal SSH Key Manager is a system management tool designed to help large organizations deal with the key-management problem. (SSH Communications Security Corp, 2017b)

This problem arises from a widespread use of SSH public keys, which can still be used to access accounts after the access is no longer needed; for example, after an employee has left the company or the application use in the target server has ended. (Ylönen et al., 2015)

UKM is comprised of a front end, one or multiple back end servers, and possible agents running in the target servers. In “agentless mode”, UKM connects the target hosts using privileged access credentials to run commands directly on the hosts using SSH. This mode is used when Unix machines are

monitored and managed by UKM. For Windows hosts, a separate management agent is run in the target server.

The bulk of the connection data used in this work was generated with UKM by adding and managing hosts.

1.6 Scope of this work

In this work, we focus on the SSH protocol, but the methodology would be very similar for other protocols as well.

The implementation details are described in the following chapter.

1.7 Structure of this thesis

This thesis follows a conventional structure. In this chapter, we introduced the anomaly detection problem and the aim of this study. Next, in Chapter 2, we go through the specifics of the implementation and used algorithms. In Chapter 3, we go through the results and compare the used methods. Finally, in Chapter 4, we introduce further work and the conclusions.

Chapter 2

Implementation

In this chapter, we go through the details of how CryptoAuditor and Universal SSH Key Manager were integrated for data collection and what features were extracted from the gathered connection data. We take a closer look at the selected clustering algorithms and the methods to visualize the data. We introduce Receiver-Operating Characteristics (ROC), the tool we used to analyze classification performance.

2.1 Basics

CryptoAuditor, and specifically a component called Hound, is a termination point for the connection between the client and server; it acts as a server to the client application and as a client to the server side. The Hound component therefore acts as the interception proxy in this context. See Figure 1.3 for details of Hound position in the CryptoAuditor architecture.

The connection data used in classifying the connection are extracted as features from the captured connection trails and the connection metadata.

A connection trail comprises the application data produced by the connection and associated metadata. It contains timestamps when activities have occurred in the connection, so that the connection can later be replayed exactly. Typically, the trail will contain a full protocol dump, with some unnecessary or risky data omitted. The password the user input in

authenticating against the system is an example of such omission. Storing it would only increase the damage should the trail be compromised, without any benefit.

The connection metadata includes connection attributes such as the username in the server, authentication method used, source IP address, destination IP address, and number of channels.

2.2 Gathering data

Considering the use case for CryptoAuditor, where it is likely that the most sensitive data of the organization is accessed through it, it is very hard to obtain data from actual production environments. However, even with a weaker data set it should be possible to use unsupervised and semi-supervised methods to create something that generalizes to a production environment. A weak data set in this sense is one generated automatically from a set of parameters or one obtained by using existing tools to generate connections in a small number of known classes.

Only successfully authenticated connections are considered for this work; the focus is on using the connection features to detect anomalous user behavior. Using clustering for intrusion detection against network level attacks has been considered by Portnoy et al. (2001); in this work we take advantage of the stored trails.

Much of the intrusion detection research uses the 1999 KDD Cup data set (SIGKDD, 1999), as good real-life data is hard to come by. As that data set is very much centered on the network level traffic instead of the application level, and CryptoAuditor is about monitoring the application traffic, we created a sample data set using a CryptoAuditor installation to monitor a set of hosts that were administered with Universal SSH Key Manager. All of the data is from captured connections, that is, the data rows are not artificially generated. We also considered generating data rows artificially from shell `.history` files and similar; however, just running `ssh` towards the hosts was fast and convenient enough, and artificial generation was abandoned.

The bulk of the connection data was generated using Universal SSH Key

Manager. The tool administers hosts registered to it and generates a lot of automated traffic with varying commands checking SSH activity logs, changing authorizations, modifying configurations, and performing other related activities. Some of the connections are "passive" in nature; they just collect data and read logs, while others modify the managed system by changing the SSH server configuration or syslog configuration, for example. An unsupervised clustering approach should be able to classify these connections into different groups.

Administration traffic was generated by logging in to the systems manually and also by using the Fabric software package's `fab` tool (Forcier, 2017). In real-life production environments, `fab` and similar tools are used in larger environments due to the amount of manual work otherwise involved. Administrative or monitoring commands may be initiated also completely automatically by an administration system.

Attack connections were represented by running `fab` against a set of hosts, copying a local root exploit binary to them, and running said binary in the hosts. In this attack scenario, a user's credentials have been compromised, for example, by eavesdropping or by a disclosed password file, which has then been brute forced. A vulnerability in the system is then exploited to elevate the stolen account privileges to those of the administrator user, gaining total control of the affected systems. This represents a whole class of attacks, and the chosen approach to highlight these connections is not sensitive to fingerprinting a particular exploit binary or vulnerability; the unsupervised approach should highlight the connection as an outlier if there are few of these connections or bundle them in a cluster if there are many connections using the same technique.

2.2.1 Feature extraction

Using CryptoAuditor, it is possible to capture the connection fully or in part, so that all the activity is stored in the system. To make it computationally feasible to classify these connections, a set of features should be generated from the data instead of just using the connection data as a bag-of-words or

similar representation.

In addition to normal connection metadata, the possible features we can extract from the trail data include: number of commands, minimum time between commands, average time between commands, command modifies users, command modifies SSH configuration, command modifies SSH keys, command downloads software, command lists system users, and command uses privileged access (`sudo`, `su`). The full list of features extracted from the connection trails is given in Table 2.1.

Extracting features from the channel data sidesteps the issue we would have in treating the packets in a channel as a time series; just studying individual packets would lose the information of the effect the packets have as a whole. We can avoid treating the connection's packets as a time series, as all packets in the channel are tightly linked. This is an advantage in the level where the interception proxy is in the network over a network-level intrusion detection system, where the inter-packet correlation needs to be considered (for example, the rolling packet window in Zanero and Savaresi (2004)). Also, while attacks that are spread over multiple connections and channels certainly exist, usually an attack against a server using SSH or RDP is carried out in a single connection.

If the connections have stored trails, the feature extraction step can be done later or redone if the system is modified to accommodate for new features. Even without the trail storage, the feature extraction can be performed online; the features just cannot be updated later.

The Vault component has a mechanism to post-process the audit trails, which can also access the channel's attributes. This can be used to add the features as attributes, which can later be dumped for analysis with, for example, a CSV dump of the audit trails.

Table 2.1: Extracted features from the connection data.

Feature	Type	Description
has_shell	boolean	The channel is a shell session.
has_sftp	boolean	The channel is an SFTP session.

Continued on next page

Table 2.1 – continued from previous page

Feature	Type	Description
has_exec	boolean	The channel executes a command.
has_tty	boolean	The channel allocates a pseudo-terminal.
has_agent	boolean	The session channel includes an agent authentication tunnel.
has_x11	boolean	The session channel includes an X11 tunnel.
has_root	boolean	The user is very likely a privileged user.
duration	float	The time duration the channel existed.
exit_status	int	Exit status of the command, or -1 if not received.
exit_signal	boolean	Whether the executed command resulted in a signal to be sent.
time_of_day	int	Time in seconds after midnight for the channel start.
weekday	int	Day of week for the channel start.
min_inter_packet_duration	float	Minimum time between channel packets.
max_inter_packet_duration	float	Maximum time between channel packets.
avg_inter_packet_duration	float	Average time between channel packets.
median_inter_packet_duration	float	Median time between channel packets.
min_packet_size	int	Minimum channel packet size.
max_packet_size	int	Maximum channel packet size.
avg_packet_size	float	Average channel packet size.
median_packet_size	int	Median channel packet size.
num_input_packets	int	Number of input packets.
num_output_packets	int	Number of output packets.
bytes_received	int	Bytes received for the channel.
bytes_sent	int	Bytes sent for the channel.
target_port	int	Connection target port (usually 22).
num_conn_log_messages	int	Number of connection-specific log messages received.

Continued on next page

Table 2.1 – continued from previous page

Feature	Type	Description
num_chan_log_messages	int	Number of channel-specific log messages received.
num_streams	int	Number of streams, usually two or three.
File transfer specific dimensions. These are set to 0 if the channel is not a file-transfer channel.		
num_files	float	Number of files transferred.
min_file_sent_bytes	float	Minimum bytes sent per file.
max_file_sent_bytes	float	Maximum bytes sent per file.
avg_file_sent_bytes	float	Average bytes sent per file.
median_file_sent_bytes	float	Median bytes sent per file.
min_file_received_bytes	float	Minimum bytes received per file.
max_file_received_bytes	float	Maximum bytes received per file.
avg_file_received_bytes	float	Average bytes received per file.
median_file_received_bytes	float	Median bytes received per file.
Command execution specific dimensions. These are set to 0 if the channel is not a shell or exec channel.		
num_commands	int	Number of commands executed on the channel. For shell sessions, this is calculated with a terminal emulation heuristic, which assumes one command per line.
has_sudo	boolean	Whether `sudo` was used on the connection. Obtained with a string-search heuristic.
has_su	boolean	Whether `su` was used on the connection. Obtained with a string-search heuristic.

Connections versus channels

A connection in general terms is something where the server and client communicate over a reliable link, usually TCP/IP, and authenticate each other in some way. A connection typically multiplexes multiple channels, which may be of different types. This applies to both SSH and RDP connections.

In the SSH protocol, the commonly used channels can be categorized to three main hierarchies: execution, file transfer, and tunnels. In an execution channel, the client requests the server to execute something, a command or a shell, possibly with an attached pseudo-terminal (tty). The I/O is then passed between the server and client in the channel.

File transfers are a special case of execution; a file-transfer program requests the execution of a file-transfer server binary on the server. The file transfer is usually carried out with the SFTP protocol, the SCP1 protocol, or just transferring `rsync` or `tar` traffic over the channel. In file transfers, the amount of data transferred is generally larger than the execution channels, and a pseudo-terminal is not used. The SFTP protocol, a popular version being described in Ylönen and Lehtinen (2001), is recognized natively by CryptoAuditor. Other means of data transfer can be supported, but they would require additional heuristics.

Tunnels are typically used to pass data between applications and happen more in the background for the user. The different channel types indicate capabilities and requirements for the endpoints; an X11 channel typically requires that the client has an X11 server which is used to open a window by the X11 application launched at the server. The two sides communicate using the tunnel created by the SSH endpoints.

Agent and X11 connections are highly unusual in connections without shell session channels, as normally the endpoints are not accessible without one – the `DISPLAY` and `SSH_AUTH_SOCK` environment variables indicate the socket addresses for the tunnels.

Bundling all the channels in one will not allow distinguishing between different usage patterns – the same user will be creating three connections with one SSH client (OpenSSH) and one connection with three channels with another (Tectia), while performing exactly the same actions. Also, for example, with Tectia SSH Client, it is possible to perform multiple command executions in the context of the same connection.

To deal with the problem, execution channels, notably SFTP and shell, are dealt with as $(\text{connection\_id}, \text{channel\_id})$ tuples. These are treated as "virtual connections" by the classifier, with connection-specific dimensions

being identical for the two channels. This way, multiple terminals in the same session and multiple terminals in separate sessions can be handled by the system identically.

While RDP connections were not used in this work, it is worth noting that also the RDP protocol has different channels, for example, for clipboard, audio, and file access. The channel handling is more specialized than in SSH, and in RDP, separating channels from the connections does not have the same advantages as with SSH. An exception here is the file handling in RDP (Microsoft Corporation, 2016b), which could warrant similar handling to SFTP channels.

Categorical data

Much of the data that is interesting in differentiating the connections is in textual form or does not allow distance comparison efficiently. This categorical data includes source and destination IP addresses, usernames, group names, matched connection rules, and the like. Categorical data are not necessarily strings; for example, the color names "red", "green", and "blue" could be described with the enumeration RED = 1, GREEN = 2, and BLUE = 3, and the fundamental property of categories would remain. In this work, the used metric between data rows, or distance, is Euclidian. With categorical data, using an arbitrary integer to represent a category would cause a distortion; with the above assignments for the labels, the distance between BLUE and RED would be different from the distance between BLUE and GREEN, which is not correct. With categorical data, the distance is usually either zero when the categories are the same or one when the categories are different.

It would be interesting to take into account the usernames and other categorical data used in the connection. Compared to numerical data, it is harder to use this information as the distance function needs extra care. Using some statistic about the username might help; for example, we could use the frequency of a username in place of the string. Similar considerations apply to the source and target IP addresses and the rule used in auditing the

connection, among others.

When testing the clustering, adding an explicit distance function made the classification much too slow with the amount of gathered data – effectively the computation did not end. This was probably due to the function call overhead and the number of calls to the 100000+ channels it generates; effectively each cell in the "virtual sparse matrix" results in a call to the distance function, making the classification actually quadratic, $O(n^2)$, in time. The `scikit-learn` package DBSCAN, using a fast spatial index, should have a computational complexity of $O(n \log(n))$ in time, see Section 2.3.2.

Tags and other administrator-assigned values

Some connections are assigned one or several tags. These are direct user input and as such can be very valuable, as they can be considered the human classification of the connections. On the other hand, some tags might be applied automatically or by a mass assign, which is less precise. In practice, the tags should be used by a mechanism assigning a classification, not as input dimensions for clustering, as it is not a natural property of the connection data itself but an arbitrary manually added feature; a similarity of connections from these should be considered coincidental.

Timestamps

Plain timestamps are not very interesting, as they are strictly increasing, and scaling them will not help. Relevant information that can be used to compare connections can be extracted from the timestamps, such as the time of the day and the day of the week a login occurred. It is anomalous if there are normally no file transfers on weekdays, and then one occurs.

Recognizing commands

Commands are searched from both the input and the output of the channel. Using the output is more reliable if echo has not been turned off, as the server's pseudo-terminal handles the incoming key sequences and performs the necessary operations, such as moving the cursor, clearing the screen, or

inserting characters according to the application – a character code for "`j`", for example, can be a starting character for the command "`jobs`" or, if the terminal is in a `vi`-like mode, cause the cursor to move down.

On the other hand, it is trivial to turn command echoing off with "`stty -echo`", which will cause the input to be missing from the output altogether, so relying on the output only is not wise. Universal SSH Key Manager runs all of its commands without echoing.

2.2.2 Scaling

Scaling or normalization is required to eliminate the bias between the data dimensions. The columns should *a priori* have equal weight in the clustering – a data dimension with only boolean values, ones and zeroes, should contribute as much as bytes transferred in the channel, which can have values from zero to gigabytes. For this purpose, the dimensions are scaled to have a zero mean and unit variance. A detailed method is described in Portnoy et al. (2001); in this work, we used the `StandardScaler` from the `scikit-learn` software package, which achieves the same result.

We used principal component analysis (PCA) to visualize the data, described in Section 2.4.1. Scaling makes the first principal component of the PCA more meaningful. Without scaling the first principal component would be very large and would look like it explained most of the variance in the data.

2.2.3 Environment setup

Universal SSH Key Manager (UKM) manages and monitors hosts. To gather realistic data, a large enough set of hosts was needed. FreeBSD jails (Kamp and Watson, 2000) are a lightweight virtualization alternative allowing thousands of guests with relatively small memory requirements. We ran more than 1000 jail instances in a FreeBSD virtual machine, for which 10 GB of memory was given. The host FreeBSD, itself a guest in VirtualBox, needed to be tweaked with respect to the maximum number of open processes and open file descriptors to allow the jails to run with the required processes: `sshd` and

`syslogd` were required for this experiment, `sshd` to allow administering by the UKM, and `syslogd` to gather the activity which is then disseminated by the UKM.

The network topology of the setup is depicted in Figure 2.1. All traffic to the `potty0`, ..., `potty1015` jail instances from UKM was routed through the CryptoAuditor installation in-between.

Universal SSH Key Manager performs a management-key setup on its first connection to the administered hosts. As transparent public-key authentication is impossible with the standard SSH protocol without access to the private key in CryptoAuditor, additional steps were necessary to be able to manage the hosts with UKM. The details of the procedure are described in Section 2.6. After the key setup was done, the management traffic between the hosts and UKM could be captured by CryptoAuditor.

Universal SSH Key Manager was used to perform administration tasks on the hosts, like changing the SSH configuration. UKM does periodic scans of the hosts, where the key activities to the hosts are checked by scanning the system logs, and the contents of the users' `.ssh/authorized_keys` files are checked for modifications. All of these activities occur over the SSH protocol and were routed through CryptoAuditor, where the connections were stored.

2.3 Performing the clustering

Investigating the projected data, a sample of which can be seen in Figure 2.2, led to the working assumption that a density-based approach, such as DBSCAN, would work better in practice than, for example, k-means++, a cluster centroid based method. A density-based approach allows us to forego specifying the number of clusters, which is a key benefit over k-means++. This is discussed more in Section 3.5.

The `scikit-learn` software package implements both DBSCAN and k-means++ with roughly the same interface, making experimenting easier.

The algorithm for DBSCAN is described in Section 2.3.2, and k-means++ is discussed in Section 2.3.3.

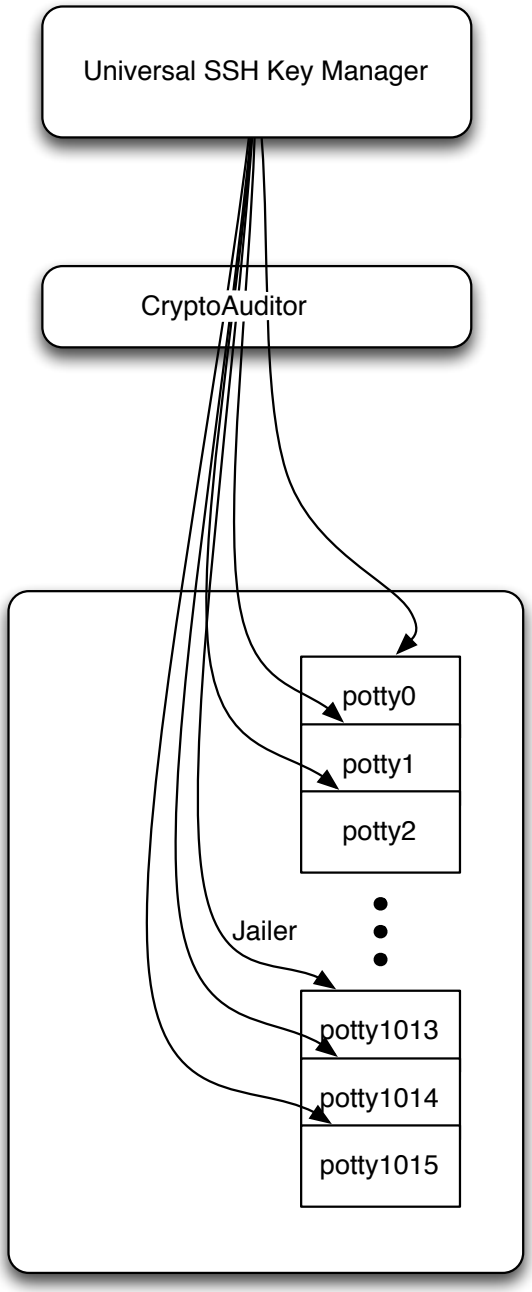

Figure 2.1: Network setup for data gathering.

2.3.1 Supervision

The whole point in automating anomaly and intrusion detection is to reduce the workload of the administrators. Using clustering to highlight the emergent structure in the audited connections and being able to check just a few connections from a cluster of thousands have real benefits for the system administrators.

In this work, we assume that automated attacks with similar connection profiles will get clustered together and that outliers outside of previously categorized connections are dissimilar enough to warrant scrutiny.

Clusters of connections that belong to clusters not already categorized should be checked by the administrator with a sampling from the cluster. The administrator can use the samples to verify the accuracy of the clustering and possibly mark the connections as malicious or benign. Further connections that get assigned to the same clusters as a previous classification will get the same class. A cluster that has connections classified both as benign and malicious should be treated as malicious, so a new connection being assigned to it should be flagged.

As a practical implementation note, new connections flagged by the system as possibly malicious should be raised to the administrators' attention.

The supervision is simulated in this work by flagging a small number of the malicious connections and observing what connections lie in the same clusters.

2.3.2 DBSCAN

In this work, we principally used DBSCAN (Density Based Spatial Clustering of Applications with Noise), originally described in Ester et al. (1996). DBSCAN is a robust algorithm, which received the 2014 SIGKDD Test of Time Award (SIGKDD, 2014). The algorithm is outlined in Algorithm 1, with the most important auxiliary function ExpandCluster shown in Algorithm 2.

The algorithm has two parameters, `minPts` and ϵ. `minPts` indicates how many points are needed to start a new cluster. The distance parameter ϵ is used to find neighbors in existing clusters, in which case the point is added to

the cluster, or if enough points are within ϵ, a new cluster is formed from the point and its neighbors. In this work, `minPts` was kept static at the default, that is, four points. The choice of `minPts` will mostly affect the number of outliers, so it does not affect the comparison between DBSCAN and k-means, described in Section 3.5.

As is evident from the linear nature of DBSCAN, given n points, the algorithm needs to perform at most n region queries. If the region query has a linear time complexity, the clustering algorithm will be quadratic, $O(n^2)$, in time complexity. As discussed in Ester et al. (1996), using an R*-tree has an average time complexity of $O(\log n)$, given that the ϵ-neighborhood of the points is small compared to the whole data-space. A k-d-tree similarly allows a time complexity of $O(\log n)$ when the number of dimensions k is small compared to the number of data points n, as described in Friedman et al. (1977).

If the data cannot be accessed with a spatial index and the distance matrix needs to be computed, the algorithm will necessarily be quadratic. This is too slow for the number of connections seen in practice with CryptoAuditor. `scikit-learn` allowed a choice; in this work, we used the k-d-tree structure. `scikit-learn` does not implement the R*-tree structure, so it could not be tested for this work.

Algorithm 1 DBSCAN

```
procedure DBSCAN(setOfPoints, ε, minPts)
    cId ← nextId(NOISE)
    for all point ∈ setOfPoints do
        if point.cId = UNCLASSIFIED then
            if ExpandCluster(setOfPoints, point, cId, ε, minPts) then
                cId ← nextId(cId)
            end if
        end if
    end for
end procedure
```

Algorithm 2 ExpandCluster

```
function ExpandCluster(setOfPoints, point, clusterId, ε, minPts)   ▷
initially all points have cluster id UNCLASSIFIED
    seeds ← setOfPoints.regionQuery(point, ε)
    if |seeds| < minPts then
        point.cId ← NOISE
    else
        for all pt ∈ seeds do
            pt.cId ← clusterId        ▷ seeds are density-reachable from point
        end for
        seeds ← seeds \ {point}
        while |seeds| > 0 do
            curPoint ← choose one ∈ seeds
            seeds ← seeds \{ curPoint }
            neighbors ← setOfPoints.regionQuery(curPoint, ε)
            if |neighbors| ≥ minPts then
                for all pt ∈ neighbors do
                    if pt.cId = UNCLASSIFIED then
                        seeds ← seeds ∪{ pt }
                    end if
                    if pt.cId ∈ {UNCLASSIFIED, NOISE} then
                        pt.cId ← clusterId
                    end if
                end for
            end if
        end while
    end if
end function
```

2.3.3 k-means and k-means++

In k-means, the execution begins by selecting k starting cluster centroids. The points in the data closest to the cluster centroids are assigned to those clusters, and the centroids are updated accordingly. The algorithm terminates when the point assignments no longer change.

Formally, in k-means, given a set of points $\mathcal{X}$, we are trying to find a set of cluster centroids $\mathcal{C}$ to minimize

$$\phi = \Sigma_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} \|x - c\|^2$$

With any stochastic initialization method for k-means we can rerun the algorithm multiple times and choose the clustering that results in the smallest value for ϕ.

The k-means algorithm is very sensitive to the initial assignment of the cluster centroids. `scikit-learn` implements the k-means++ initialization, which is more efficient than a random assignment (Arthur and Vassilvitskii, 2007).

k-means++ initialization is also stochastic but weights the probability of choosing a point by the distances to already selected centroids. Let $D(x)$ be the distance to the closest centroid from point $x \in \mathcal{X}$. The first centroid is chosen uniformly at random from $\mathcal{X}$, the rest $k-1$ centroids will be chosen with probability $D(x)^2 / \Sigma_{x \in \mathcal{X}} D(x)^2$. The probabilities are updated each time a new centroid is selected.

The k-means algorithm proceeds in the following way.

1. Initialize set $\mathcal{C}$ of cluster centroids either by selecting uniformly at random, using k-means++, or using some other means.

2. Update cluster assignments C_i for every $i \in \{1, \ldots, k\}$ to be the set of points that are closer to c_i than they are to c_j for all $j \neq i$.

3. Update c_i for every $i \in \{1, \ldots, k\}$ to be in the center of points in C_i, so $c_i = \frac{1}{|C_i|} \Sigma_{x \in C_i} x$.

4. Repeat steps 2 and 3 until the cluster centroids no longer change. (Arthur and Vassilvitskii, 2007)

2.4 Visualizing the data

Visualizing the data is important; it allows to quickly verify whether the method being verified is working at all and allows detecting possible improvements or problem points. Visualizing data with high dimensionality is hard, especially when the visualization should work also on the surface of the paper of this thesis. One method is simply to reduce the number of dimensions in the data while still retaining as much of the explanatory power, or variance, of the data as possible. A good tool for this is the principal component analysis.

2.4.1 Principal component analysis (PCA)

PCA is a very popular dimensionality reduction technique. The assumption is that the data dimensions are correlated and there exists a coordinate system with smaller number dimensions that can represent a large portion of the variance in the data.

The computational complexity of the eigen decomposition of the covariance matrix is $O(D^3)$, where D is the number of dimensions in the data, and although this is not particularly efficient in computational complexity, it is not a problem as long as $D \ll n$, where n is the number of connections. The projection operation is linear in time and space after the decomposition. (Barber, 2012)[Section 15.2]

An example of visualizing a clustering result is in Figure 2.2.

2.5 Receiver operating characteristics (ROC)

Given our data, where the number of normal connections greatly outnumbers the anomalous connections, a classifier labeling everything as benign would have a reasonably good accuracy. It is therefore important to be able to observe classifier performance in a more holistic fashion. A practical test for analyzing classifier performance is the ROC curve. (Zweig and Campbell, 1993)

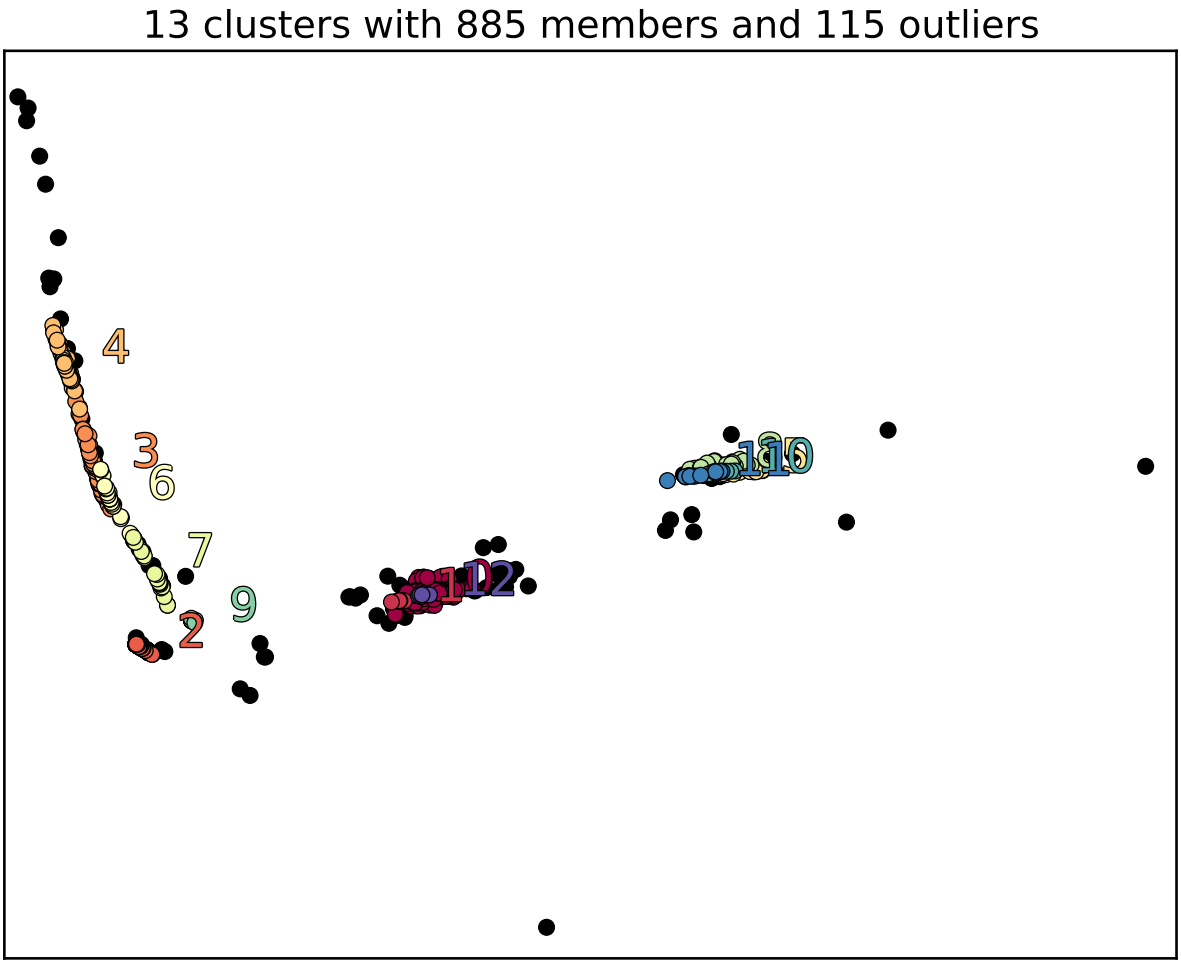

Figure 2.2: A sample clustering with outliers.

In this work, the curves were plotted as a function of the used parameter: ϵ for DBSCAN and the number of clusters k for k-means. In the y-axis, we plot the detection rate, that is, the TP/(TP + FN) ratio, where TP denotes the true positives and FN the false negatives, or detection failures. In the x-axis, we plot the false positive rate, which is the ratio FP/(FP + TN), where FP denotes the false positives and TN the true negatives.

The best behaving classifier would have no false positives and zero detection failures, that is, a detection rate of 100% and a false positive rate of 0%. In the ROC curve this is the top left corner. When interpreting the ROC curves, we are trying to find parameters where we minimize the number of the false positives while trying to maximize the detection rate.

2.6 System implementation

In this section, we discuss implementing a clustering-based anomaly detector for CryptoAuditor. The work included integrating CryptoAuditor with Universal SSH Key Manager to gather data from the UKM-generated connections. Even though the functionality implemented here is not core to this work as such, the integration work was large enough to warrant a description.

The mission was to audit the Universal SSH Key Manager (UKM) traffic by CryptoAuditor. Auditing the traffic might not be very interesting for most customers, but it might be necessary to set up UKM to work through CryptoAuditor because of network topology reasons. Also, a company policy might enforce the auditing of the traffic.

CryptoAuditor can only transparently support password and keyboard-interactive authentications. Later in the text, these will be referred collectively as password authentication, as that is the main use case also for keyboard-interactive. All the challenges and responses are sent through the encrypted channel in clear text and thus allow for transparent relaying in CryptoAuditor.

Public-key authentication cannot be transparently relayed, as the challenge signed with the user's private key contains information on the initial key exchange, the session ID. The session ID is unique for each key exchange,

at least with Diffie-Hellman, and cannot be reproduced, even if the server host's private key was known. This is because the key exchange includes information, a random nonce, from both ends of the negotiation, which is mixed in when the session ID is created. As the man-in-the-middle can only control one side of this to each direction, the session ID will be different for each session.

UKM only uses password authentication on the initial connection to added hosts. During that, the management keys are created. Each host will get its own set of asymmetric keys; thus the problem cannot be solved by just importing one private key to CryptoAuditor. UKM also supports a feature to replace the current management keys, in which case the private key in CryptoAuditor needs to be updated.

2.6.1 Registering the keys

During the discovery phase, the host connection is performed using password authentication, which can be relayed. UKM then creates a management key pair, as usual, and registers the private key to CryptoAuditor with information about the target host. This is done by running a management command in CryptoAuditor and is performed using a secure channel from Universal SSH Key Manager to CryptoAuditor – in practice, this is achieved by a manually set up public-key authentication. After this, the private key can be used to login to the Hound with public-key authentication, and the connection from the Hound to the target server is performed using public-key authentication with the registered private key. Two approaches in CryptoAuditor to handle the public keys were implemented, described in the following sections: through the configuration mechanism and direct storage to the Vault.

Through configuration mechanism

The first attempt was to configure the admin users with the UI configuration mechanism, where a configuration is created for the Hound component. All the public keys were authorized for the user, so that the Hound allowed login

to the admin account. Login to the server host was then done with the user mapping mechanism in CryptoAuditor – the mapping was chosen based on the target server, and the uploaded private key was used to authenticate.

This approach was scrapped, because the configuration update mechanism was not meant to scale to thousands of automatic configuration updates. The configuration update was designed to work through the web UI, and database locking created problems when done from the automatic job; as a worst case, the UI would be unresponsive when the transaction was ongoing. The tested implementation did not fully employ transactions for the configuration updates and it caused problems when running the system with hundreds of end servers concurrently. Even if these problems had been fixed, more work would have been needed in UKM to support the updates in batch, as otherwise the host addition would need to progress one host at a time, disabling any possibility for parallel runs in UKM. This is because the job in UKM that updates the key requires the key to be used in the subsequent authentication to the server, which is done immediately after the key insertion in the same job.

Incoming users stored in the Vault

A working version of the system stores the user mapping information in the storage module, that is, the Vault. The information is inserted and updated without going through the UI configuration version mechanism.

The versioning allows to track and possibly revert changes in the configuration, in a manner similar to the ones used in source code version control systems. The benefit of not using the versioning is scalability, but the version information is lost. This was not considered to be a large problem, as the user information is something that does not necessarily require versioning. The user groups and rules are still versioned, individual users being more dynamic in nature.

Chapter 3

Results

In this chapter, we show how the connection data was visualized in this work, how the connections were classified based on the clustering with different starting parameters, and how the computational performance of the system behaved under different conditions.

3.1 Visualizing the clustering

An early clustering with a number of connections generated with Universal SSH Key Manager was used to initially validate the effectiveness of the method.

The obtained cluster distribution with $\epsilon = 0.3$ is depicted in Figure 3.1. Over half of the connections are in six clusters. Over 90% of the connections are contained in 25 clusters. Outliers are excluded from the figure.

The five largest clusters are shown in Figure 3.2, with clusters 3 and 11, as well as 9 and 16 being very close to each other. Closer inspection of clusters 3 and 11 indicates that they are Universal SSH Key Manager key-activity scan connections, where the command is `cat /var/log/auth.log \| grep ' sshd\[.*\]: ' \| grep 'Found matching .* key: \|Accepted publickey'`.

PCA eigenvalues indicated that a 2-dimensional projection explained around 43% of the variation in the data, and a 3D projection explained around 53%.

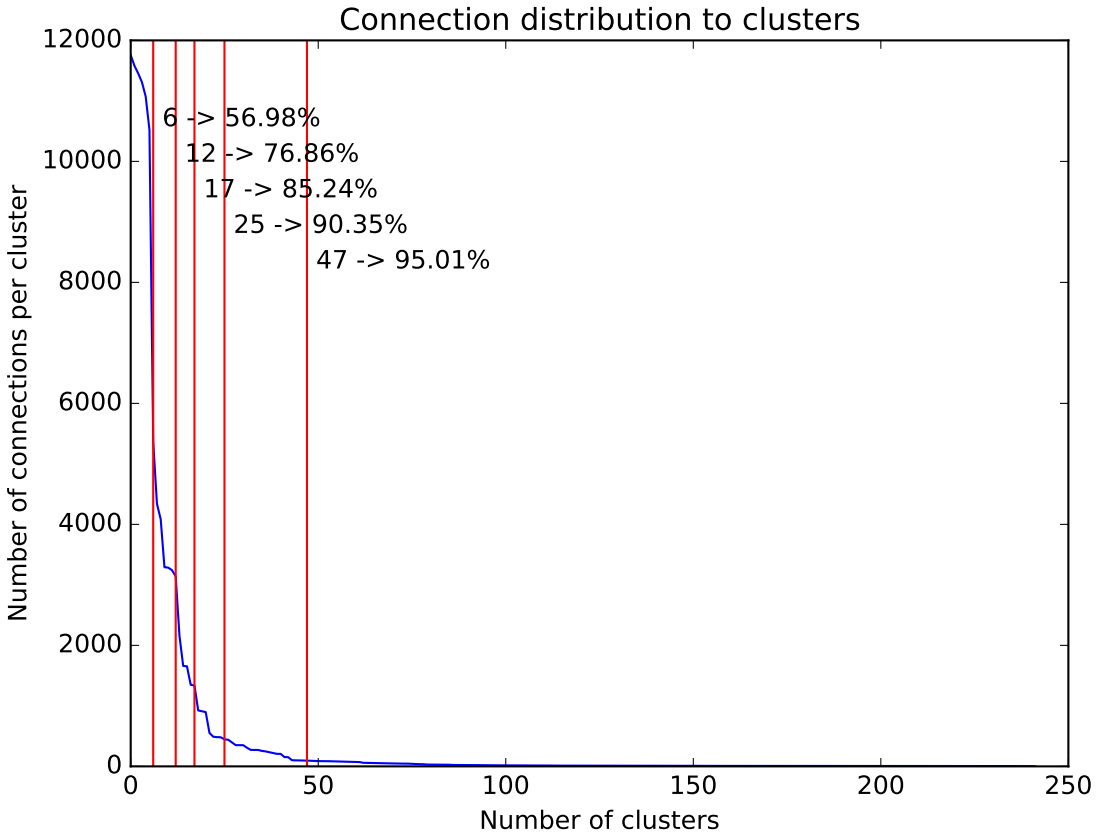

Figure 3.1: The cumulative connection distribution with $\epsilon = 0.3$. Plotted here are the cluster populations with decreasing sizes. Six of the most popular clusters contain over half of the connections.

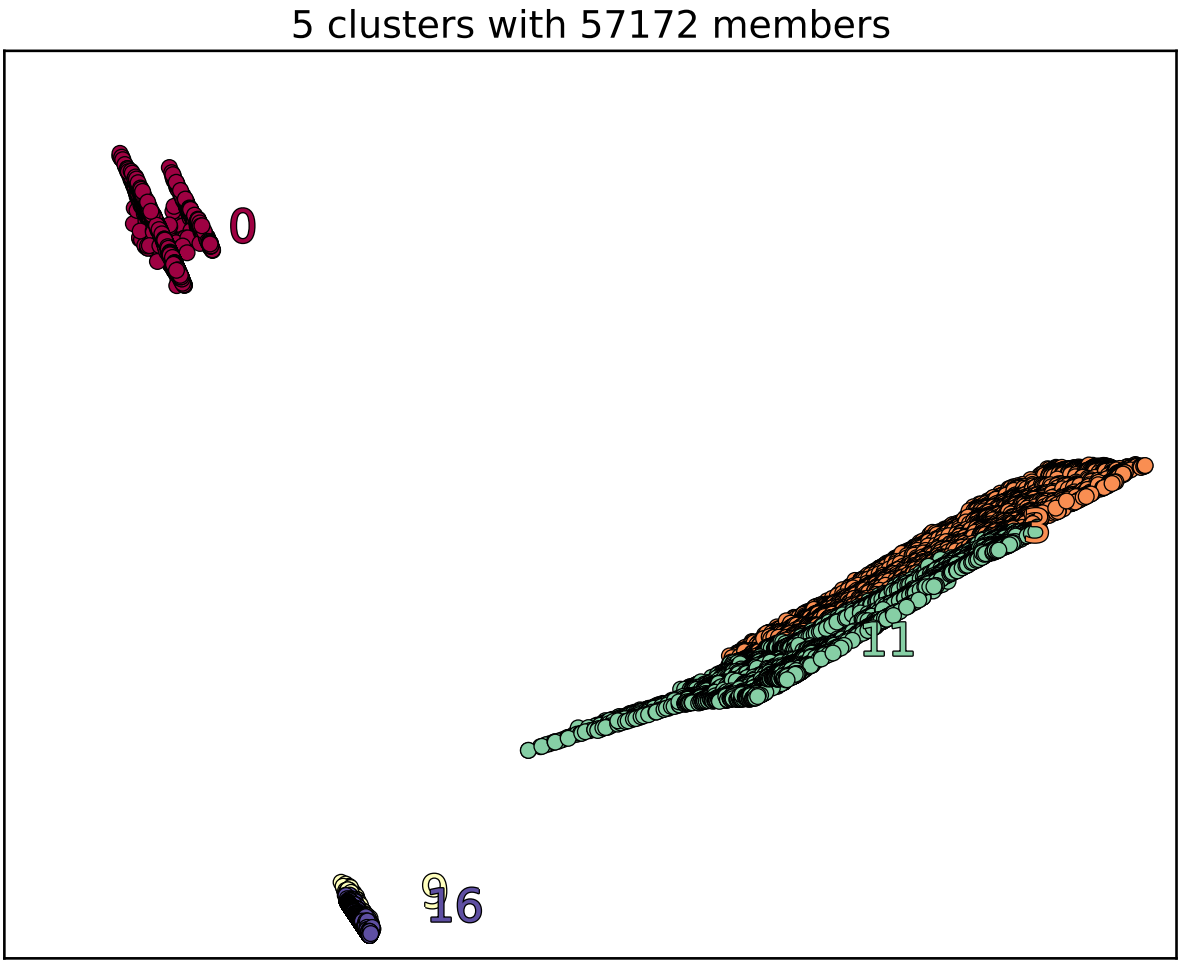

Figure 3.2: Five of the most populated clusters, projected with PCA.

3.1.1 Covariance

The covariance between the dimensions was measured from the data using `scikit-learn ExperimentalCovariance`; a visualization of the resultant matrix is shown in Figure 3.3. From the figure, it can be seen that some dimensions have a strong positive covariance (darker color), such as with `has_shell` and `has_tty`. There are dimensions with a strong negative covariance as shown by the very light color when comparing `has_shell` and `has_exec`. Both of these are as to be expected. From the figure, it can also be noticed that some dimensions do not seem to correlate even with themselves; that is an indication that there is no actual data along that dimension. For example, no files from the hosts have been downloaded, which is indicated by all zeroes in a dimension like `received_bytes` and as the dimension showing a zero covariance throughout in the matrix.

3.2 Clustering performance

Time-measurements during the data collection for this work indicate a quadratic time complexity for the `scikit-learn` DBSCAN implementation. A similar effect was encountered with smaller and larger values for ϵ, depicted in Figure 3.4. This result was disappointing, as the algorithms should allow for $O(n \log n)$ time complexity, where n is the number of connections.

3.3 Validating the classification

Running the classification with different values for ϵ gives a wide range for the numbers of clusters and outliers; not surprisingly, the two correlate heavily. The numbers are shown in Figure 3.5 as a function of ϵ.

Only the attack connections are really true positives, all the rest are considered benign. As there were few attack connections, all of them, excluding the samples used to mark clusters, were used to validate the classification. The data contains lots of singular connections, file transfers, manual command invocations, and the like, a large portion of which do not cluster and

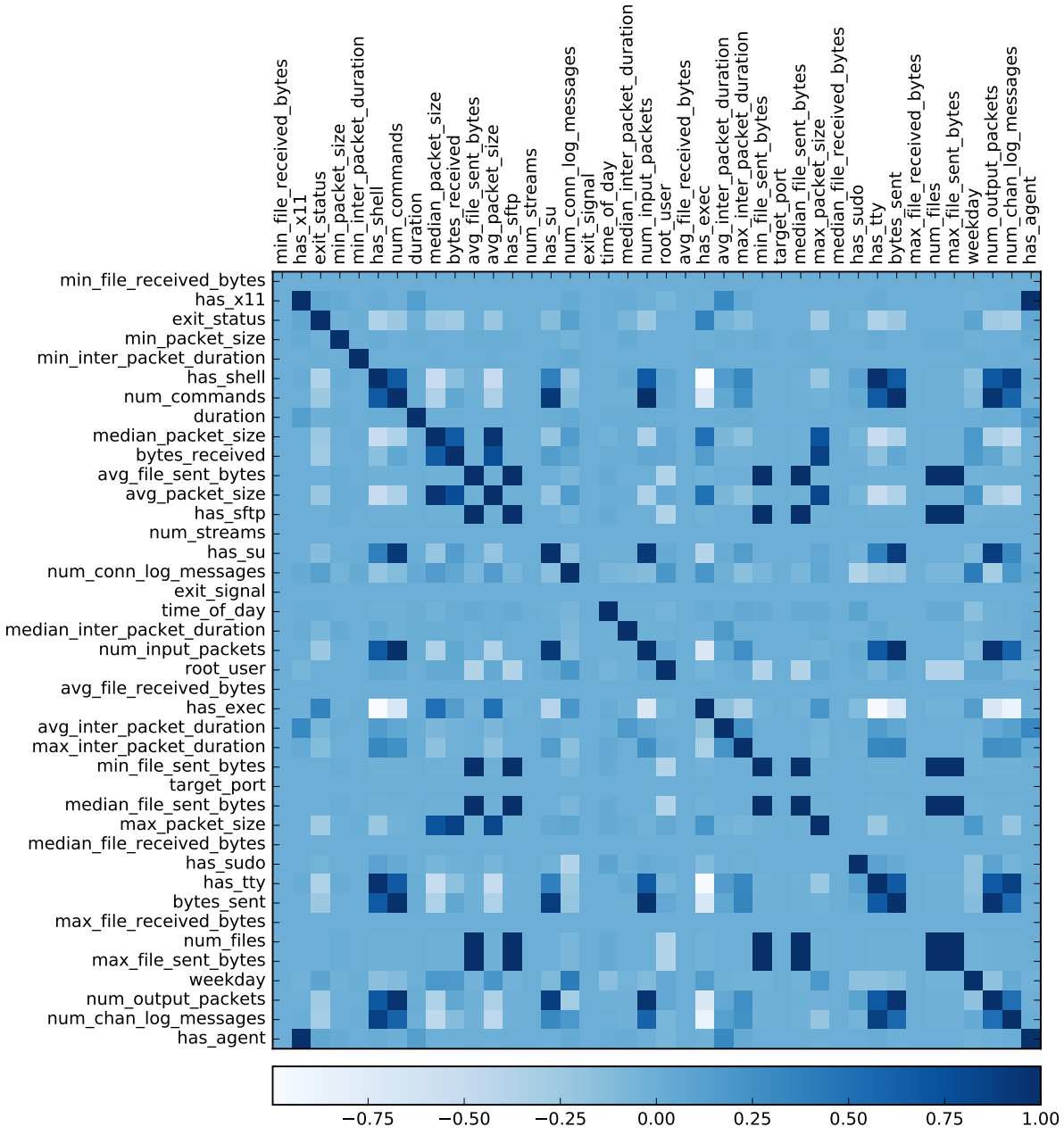

Figure 3.3: The experimental covariance. A lighter color indicates a negative covariance, whereas a darker color indicates a positive covariance. Some dimensions do not seem to correlate even with themselves; indicating that there is no actual data along those dimensions.

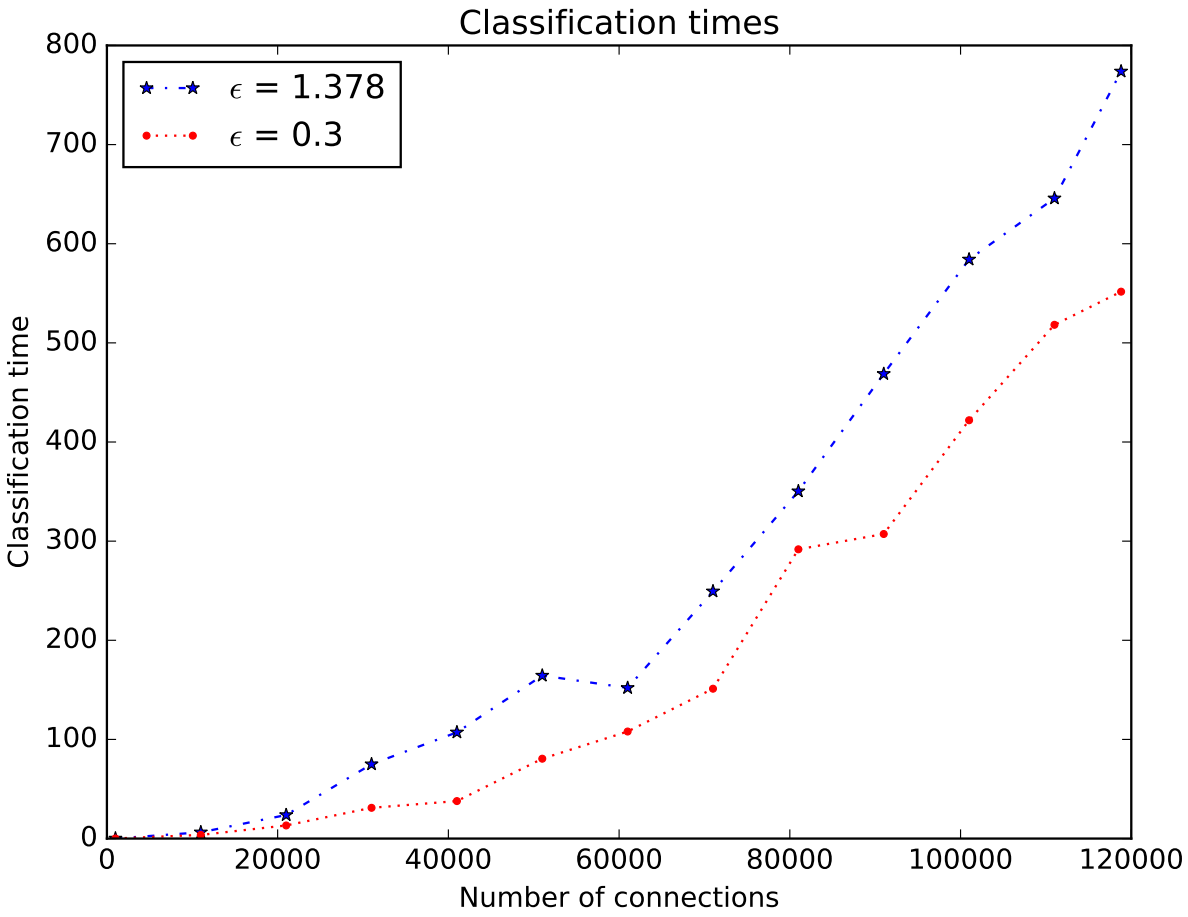

Figure 3.4: The classification performance with different values for ϵ.

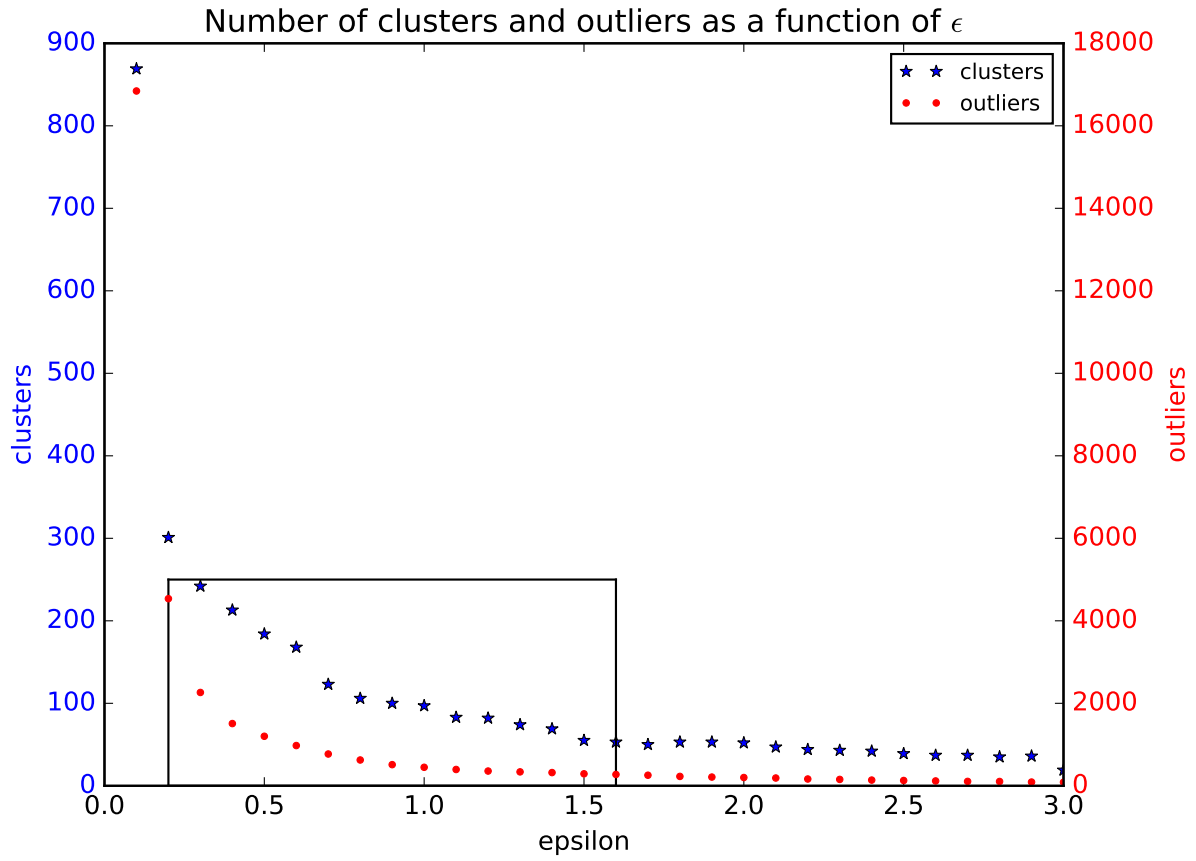

Figure 3.5: The clusters and outliers when ϵ is varied between 0.1 and 3.0. The highlighted area of interest is shown in Figure 3.6. Notice that the right axes get scaled differently between the images.

will be outliers from the clustering and therefore considered false positives.

The sweet spot seems to be somewhere between $\epsilon = 0.3$ and $\epsilon = 1.5$, shown in Figure 3.6. Here the number of clusters and outliers starts to decline less rapidly with increasing value for ϵ.

As all the attack connections should be clustered to one or two clusters, we can treat the rest of the clusters as benign connections. For validation, we know the attack connections, and having tagged one, we should find the rest. Any attacks not in clusters with the tagged one are detection failures or in other words, false negatives.

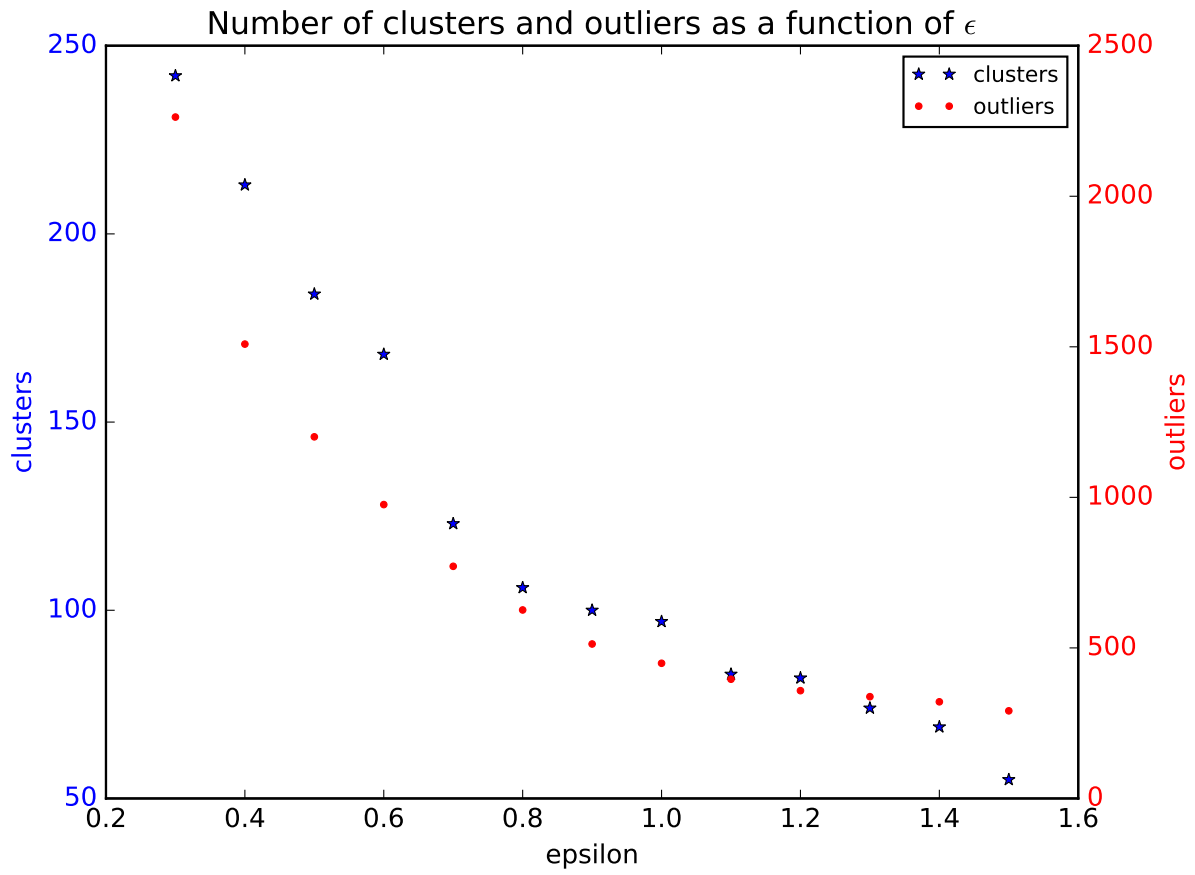

Figure 3.6: The clusters and outliers when ϵ is varied between 0.3 and 1.5. This is a more detailed view from Figure 3.5.

The administrators should always take a closer look at detected outliers. Some of the outliers detected in this work are connections which legitimately should belong to a cluster; those should be treated as false positives. Others are singular events; especially at the early stages of data gathering, sessions were created that failed because of misconfigurations and other such errors. If a unique connection has been successfully authenticated, it should be treated as an anomaly and not as a false positive.

Intuitively, as we increase the value for ϵ, the detection rate increases and the number of outliers goes down; however, also more and more benign connections get grouped in the same clusters as the identified attack connections.

When validating classifiers using supervised learning, the classifier performance can be checked by cross-validating the classifier results; with clustering we do not train a model, and cross-validation is not so useful. Our data contains relatively few attack connections, keeping in line with the assumption that normal traffic greatly outnumbers malicious traffic — in the order of a hundred to one. In these circumstances, as pointed out in Portnoy et al. (2001), a classifier marking everything as normal would have an accuracy of 99%, making it almost useless as a distinguishing metric in our case.

A more interesting metric is how the false positive rate behaves when compared to the detection rate. For this, we use the receiver operating characteristics (ROC) curve, described in Section 2.5. The detection rate is also known as the true positive rate, sensitivity, and recall. The false positive rate is equivalent to $1 -$ specificity. The specificity is also known as the true negative rate, which in this case would mean the rate that the benign connections are correctly identified as benign.

Having chosen a good value for ϵ by studying the ROC curve, we can calculate the accuracy and compare it with the accuracy from other methods.

3.3.1 Single tagged connection

The results in this section were obtained with a single point as training data.

As an example, one of the attack connections, 57646, was picked out.

Ideally, all of the attack connections would be contained in the two clusters that the connection is a part of. There are two connections instead of one because the SSH connection created with the `fab` tool will have two SSH channels: one for the file transfer, another for the command execution. These act to represent an automated attack.

The ROC curve showing the detection rate (DR) against the false positive rate (FPR) is shown in Figure 3.7. Some values from the ROC analysis are shown in Table 3.1, including the detected accuracy (ACC).

A highlighted segment of the ROC curve with the best values for the false positive rate is shown in Figure 3.8. The turning points in the curve, which we can use to decide on the value of ϵ, depending on how we value the detection rate against the false positive rate, are shown in Table 3.2.

With the gathered data, the false positive rate continues to decrease until there are only a few clusters left. Apparently this is because the "normal" connections generated by Universal SSH Key Manager are so different from the attack connections. From Figure 3.5 it can be extrapolated that eventually the clusters would be coalesced and the amount of false positives would therefore be approximately equal to the number of connections N as $N \gg m$, where m is the number of true positives, that is, the number of attack connections. The experimental validation of this claim was unsuccessful because with such large values of ϵ the clustering allocates so much memory that the program is killed by the operating system [1].

3.3.2 Two tagged points

When two points of administrator-classified connections are allowed, the detection rate is much better with very low values of ϵ, as expected. For comparison, the full ROC plot is shown in Figure 3.9, and a drill-down plot of the most pertinent ϵ range is presented in Figure 3.10.

[1] macOS Sierra (10.12.3) running on MacBook Pro, Mid 2015, with 16 GB of RAM.

ϵ	DR	FPR	ACC
0.1	0.475,490	0.142,012	0.857,332
0.2	0.500,000	0.038,254	0.960,953
1.365	0.500,000	0.002,816	0.996,330
1.37	0.500,000	0.002,808	0.996,339
1.375	0.500,000	0.002,808	0.996,339
1.376	0.500,000	0.002,808	0.996,339
1.377	0.500,000	0.002,766	0.996,381
1.378	0.720,588	0.003,592	0.995,935
1.379	0.720,588	0.003,592	0.995,935
1.38	0.720,588	0.003,592	0.995,935
1.9	0.720,588	0.005,885	0.993,645
2.0	0.720,588	0.005,792	0.993,738
2.1	0.941,176	0.005,691	0.994,218
2.2	0.941,176	0.005,489	0.994,420
2.94	0.941,176	0.004,857	0.995,051
2.96	0.941,176	0.004,857	0.995,051
2.98	1.000,000	0.004,848	0.995,160
3.0	1.000,000	0.004,848	0.995,160
3.29	1.000,000	0.004,671	0.995,337
3.295	1.000,000	0.004,671	0.995,337
3.3	1.000,000	0.004,671	0.995,337
3.305	1.000,000	0.004,671	0.995,337

Table 3.1: Selected values from the ROC analysis.

ϵ	DR	FPR
1.377	0.500	0.002,766
1.378	0.721	0.003,592
2.98	1.000	0.004,848
3.285	1.000	0.004,671

Table 3.2: Interesting values for ϵ from the ROC analysis.

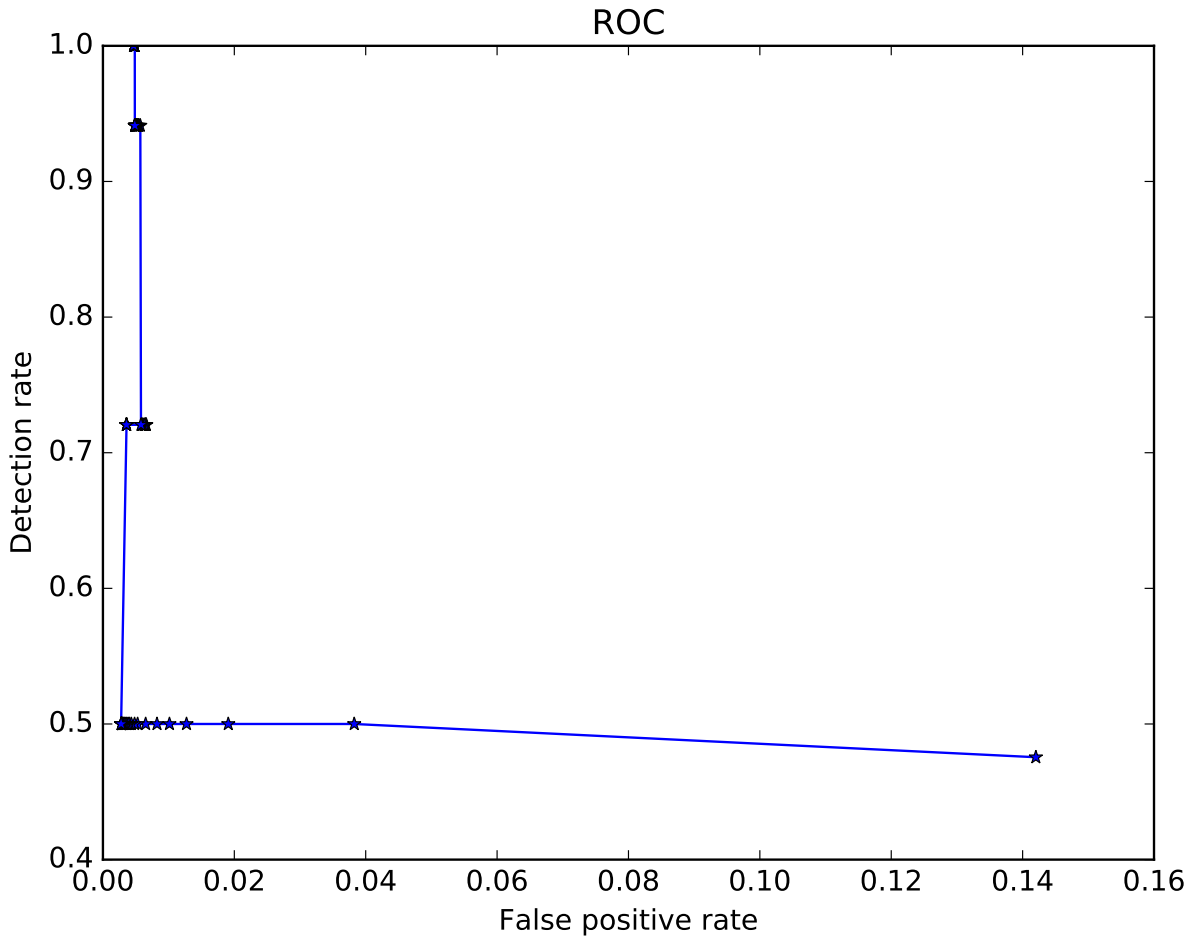

Figure 3.7: The ROC curve for different values of ϵ. This is a parameterized curve as a function of ϵ; interesting points are the ones where the curve turns. A highlight of this plot is shown in Figure 3.8.

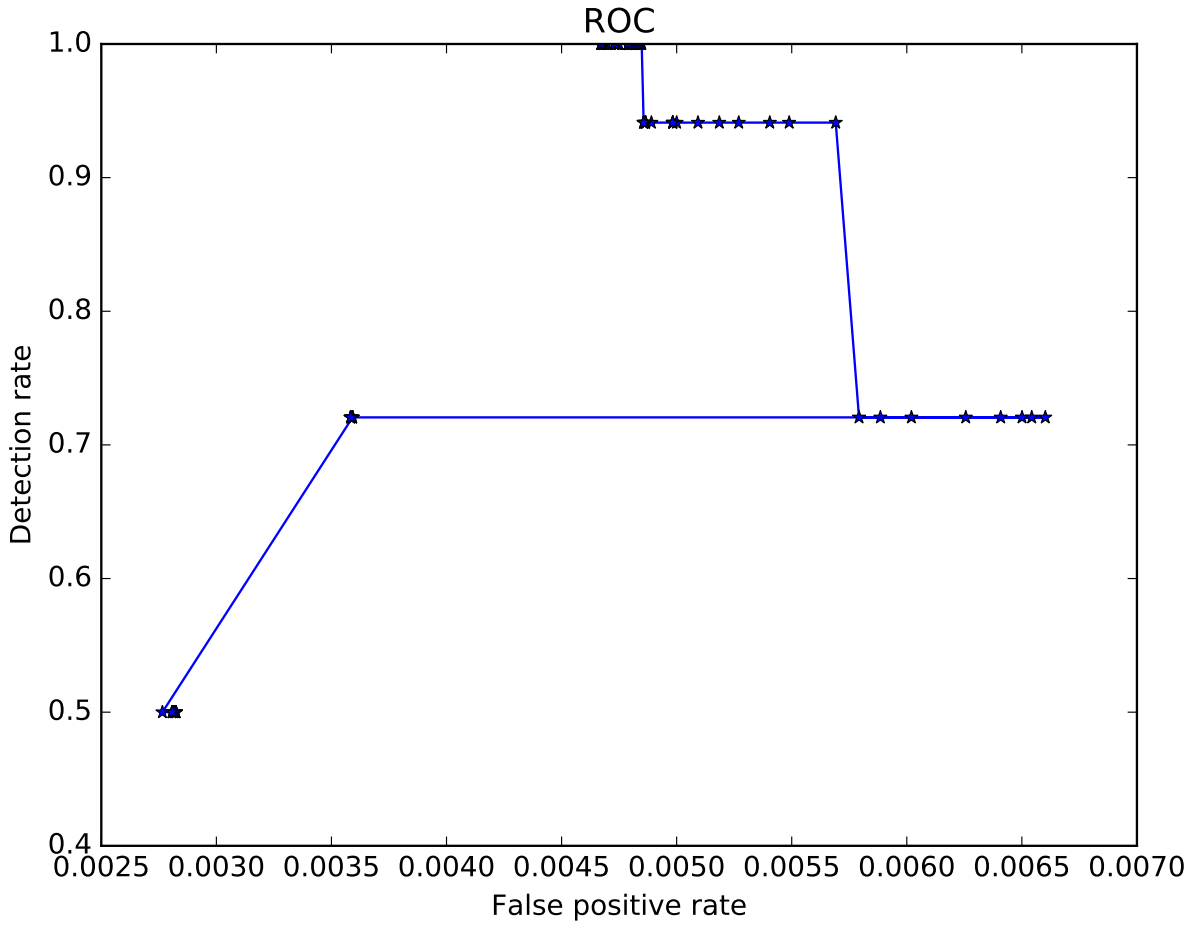

Figure 3.8: The ROC curve for ϵ in the range $[1.35, 4]$. This plot is a highlight of the plot shown in Figure 3.7.

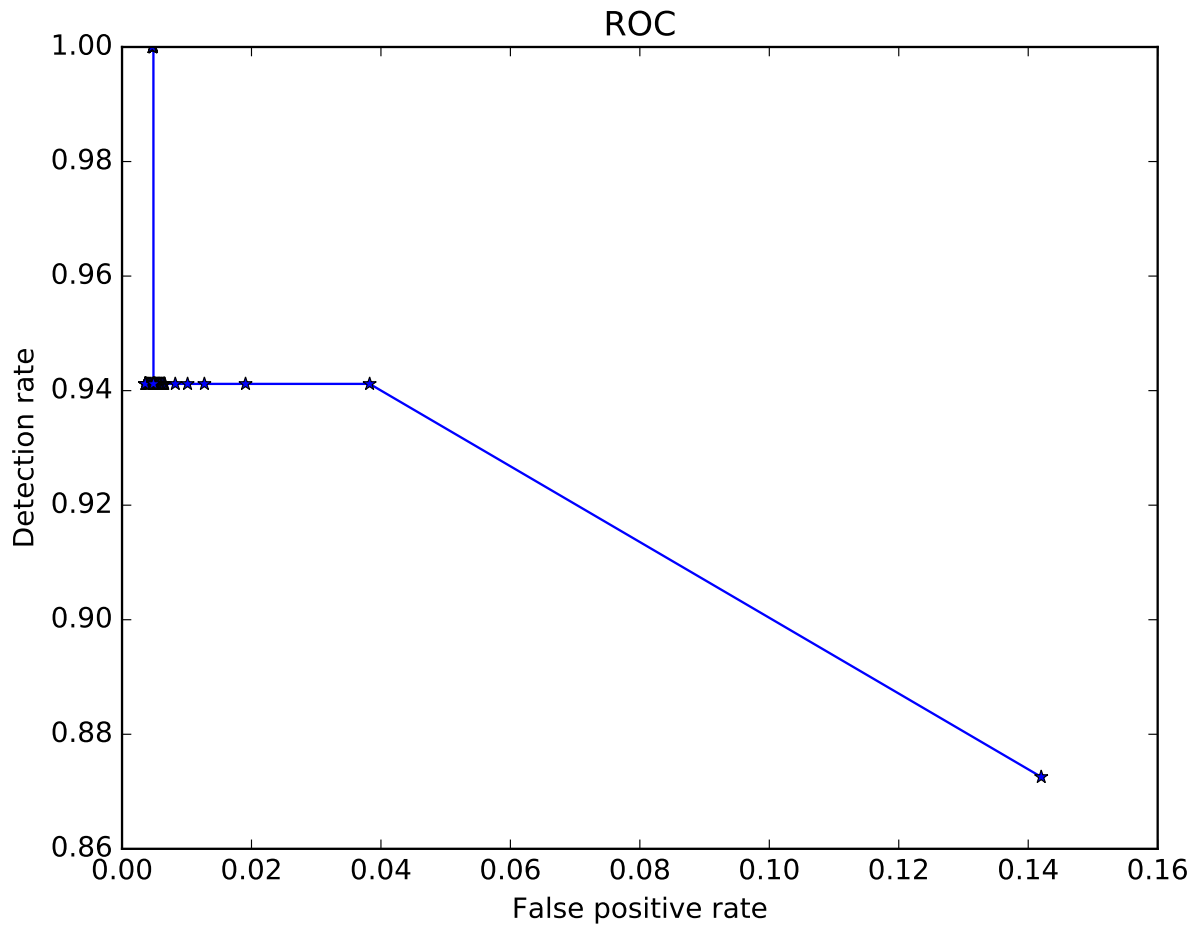

Figure 3.9: The ROC curve for different values of ϵ with two training points. A highlight of this plot is shown in Figure 3.10.

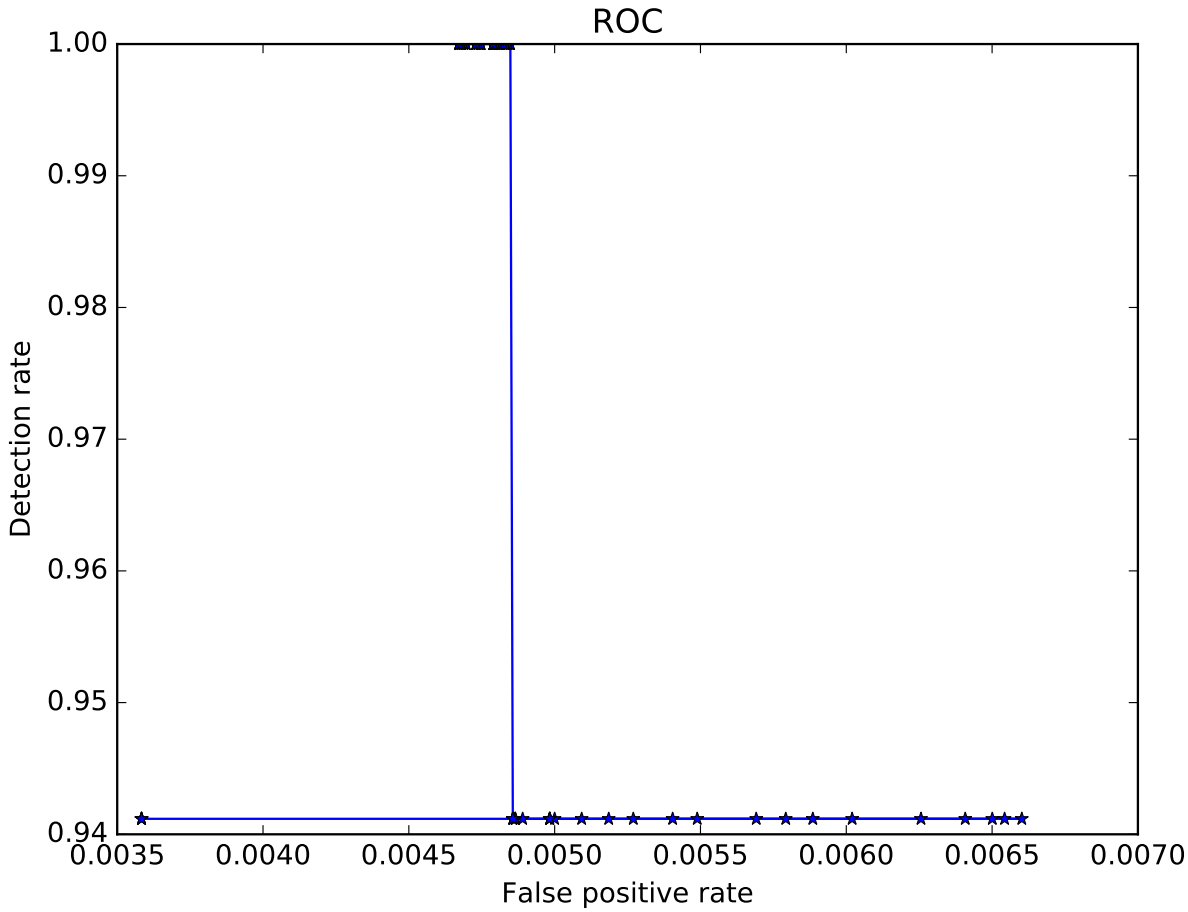

Figure 3.10: The ROC curve as a function of ϵ in the range $[1.39, 4]$ with two training points. This is a highlight of the ROC curve shown in Figure 3.9.

3.4 Number of clusters as a function of connections

Intuitively, it would be beneficial if the number of clusters and outliers increased more slowly than the number of connections – with the assumption that the clusters contain almost exclusively certain kinds of connections and that misclassifications are rare, low numbers of clusters and outliers mean that the administrators need to do less inspections of the results as time passes.

In Figure 3.11, the number of clusters and outliers has been plotted as functions of incoming channels with different values for ϵ. The growth on the number of clusters is manageable, almost linear, whereas the number of outliers remains constant after an initial bump. This is an encouraging result.

3.5 Comparison with k-means++

We measured the performance of the clustering against k-means++. The results are shown in Tables 3.3 and 3.4. The best accuracy (ACC) reported for k-means++ was 0.996 with 12 clusters, whereas for DBSCAN it was 0.999 with, for example, $\epsilon = 0.2$ and $\epsilon = 1.0$.

As k-means++ requires the number of clusters as a parameter, it is not very suitable to our overall use case. The number of clusters should be connected to the number of different types of possible connections in the data, and thus it is something that can change over time. It can be very different depending on which organization has installed the interception proxy and at what point of the network. It is therefore just the kind of parameter that should be detected from the data instead of needing to be fed to the device. Nevertheless, the algorithm was easily accessible in the `scikit-learn` package, and because the algorithm is fast, the experiments could be performed swiftly.

The k-means++ algorithm does not output outliers, and in the compar-

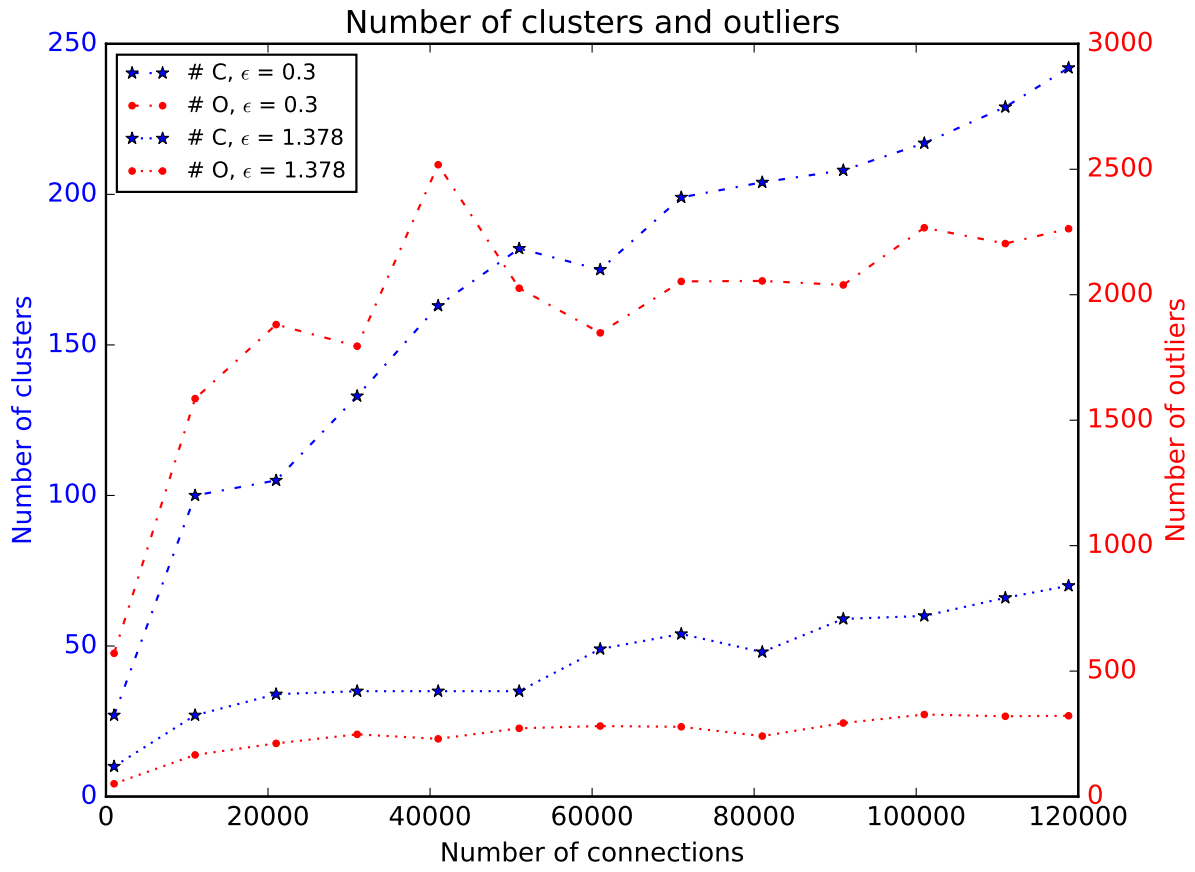

Figure 3.11: The number of clusters and outliers as a function of connections. In the figure legend, "# C, $\epsilon = 0.3$" should be read as "the number of clusters when ϵ equals 0.3".

ϵ	DR	FPR	ACC
0.2	0.500,000	0.000,000	0.999,141
1.0	0.500,000	0.000,051	0.999,091
1.4	0.720,588	0.000,877	0.998,645
3.0	1.000,000	0.004,115	0.995,892

Table 3.3: Notable values from DBSCAN with outliers treated as benign.

clusters	DR	FPR	ACC
2	1.0	0.559,868	0.441,094
3	1.0	0.559,868	0.441,094
4	1.0	0.334,309	0.666,265
5	1.0	0.312,513	0.688,023
6	1.0	0.312,471	0.688,066
7	1.0	0.309,309	0.691,222
8	1.0	0.312,513	0.688,023
9	1.0	0.004,123	0.995,884
10	1.0	0.043,119	0.956,955
11	1.0	0.305,810	0.694,715
12	1.0	0.004,123	0.995,884

Table 3.4: Notable values from k-means++.

ison the outliers detected by DBSCAN were considered benign connections, as if they were in a cluster of their own.

The k-means++ algorithm worked surprisingly well when compared to DBSCAN, but because of the stochastic nature of the initialization in k-means++, the cluster assignment occasionally caused the detection rate to drop and the false positive rate to soar.

In this particular case, k-means++ fared very well all things considered; the algorithm was faster, the false positive rate was comparable, and all attack connections were detected. However, the unstable nature of the clustering should not be overlooked. In DBSCAN, two runs to cluster a set of

points will result in practically the same clusters, with the possible exclusion of border points between two clusters (Ester et al., 1996)[Section 4.2, Section 3, Lemmas 1 and 2]. Also, adding points to the data set will not change the clustering, except by causing the merging of clusters, providing that a dense enough set of points connects two clusters. DBSCAN is thus more consistent than k-means++.

The parameter k for k-means++, the number of clusters, intuitively seems more sensitive than ϵ in DBSCAN, as discussed in Section 2.3.3. Choosing a conservative value for ϵ will just cause us to have more clusters in the result, increasing the screening work of the administrators; if this was the case, ϵ could be increased, even by the end-user administrators, as long as care was taken that the clustering was still explicit enough. If completely different kinds of connections started to be clustered together, it would be an indication that ϵ was too large.

A visualization of the differences between the algorithms is depicted in Figure 3.12.

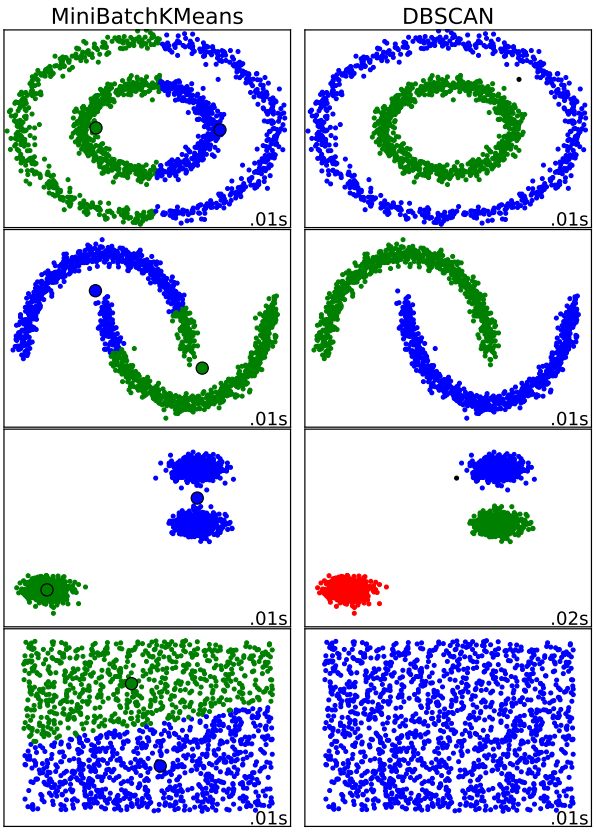

Figure 3.12: A visual comparison between k-means and DBSCAN.

Chapter 4

Discussion

In this chapter, we go through the problems raised during the implementation and data analysis of this work. We discuss some of the future work with anomaly detection in interception proxies in general and CryptoAuditor in particular. We lay out ideas on how to proceed in creating a production-ready system with a detector of anomalous connections for CryptoAuditor. Finally, we present our conclusion.

4.1 Problems

In this section, we discuss some unresolved issues in the approach used in this work. Evasion and mimicry attacks are problems for any method and are largely unresolved in general; existing systems can fight them by obfuscating what they actually measure and keeping the algorithms trade secrets, but these measures require constant updates as the attackers evolve their methods.

4.1.1 Mimicry attacks

Any approach to solve the problem of misuse detection will, at least to some extent, be vulnerable to mimicry attacks. In Wagner and Dean (2001), a mimicry attack is defined as a sequence of system calls taking the system to an insecure state without detection. With the approach in this work, a

bash	`CMD="e.c.h.o.  .f.o.o"; ${CMD//.}`
tr	`` `echo "ech% f%%" | tr % o` ``
python	`` `echo "oof ohce" | python -c 'import sys; print("".join(reversed(sys.stdin.read().strip())))'` ``

Table 4.1: Examples of running `echo foo`.

mimicry attack occurs if an attacker can parrot the actions of real connections well enough to be classified as a non-anomalous connection, that is, to belong to a cluster of benign connections.

4.1.2 Evasion

Most of the evasion tactics, such as the insertion of packets or fooling an IDS with a spoofed three-way handshake, as introduced in Ptacek and Newsham (1998), do not apply to CryptoAuditor, as the system will not initiate the forward connections without a valid underlying TCP connection. However, when considering the captured terminal data, similar techniques apply to a stream of commands. Using shell comments, environment variables to contain strings, and commands such as `tr` to encode and decode the commands to the shell – any one of those methods, and more, can be used to achieve similar results in avoiding a terminal session from being tagged as, for example, a session modifying user keys, or one where administrative privileges are requested. If the server shell is not restricted in the commands it can run, there is practically an infinite number of ways to encode a terminal session with functionally identical command contents. Some ways of running `echo foo` without that string appearing in the output are outlined in Table 4.1.

4.1.3 Correlation between dimensions

Some dimensions might correlate heavily. As we handle the dimensions with equal weights, multiple values "pointing in the same direction" will cause the shape of the data to skew in the direction of the actual data with the

combined weight of the correlating dimensions. Let's consider a made-up example: a column "has-feature" and a mistakenly introduced column "does-not-have-feature". The data has an inverse correlation, so that when the "has-feature" has the value 1, the column for "does-not-have-feature" has the value 0, and vice versa. The combined weight of these columns would skew the data when compared to a dimension without a correlating data column.

This effect can be identified by studying the eigen decomposition in PCA and the covariance matrix of the data. As an example, from 40 dimensions in the original data, the first three dimensions explained over a half of the variation in the data, as reported in Section 3.1. The covariance matrix is discussed in 3.1.1.

4.1.4 Cluster density

DBSCAN assumes the clusters in the data to have similar densities. If some of the clusters are denser than others, depending on the value of ϵ, some of the dense clusters may be coalesced if ϵ is too large, or data points that should belong to a more sparse cluster will be classified as outliers if ϵ is too small.

4.1.5 Mixed clusters

Intermingling clusters will be considered a single cluster with larger values of ϵ, by design. This did not pose a real problem in this work, but the situation could rise in production environments. Consider two sets of connections: one that is represented by good backup connections and another that is comprised of automated attacks transferring data in a similar access pattern, masquerading as the backup application. This scenario is illustrated in Figure 4.1.

Here DBSCAN will combine the clusters, and if one cluster contains flagged connections, the whole combined cluster will be "tainted". The situation is improved somewhat with supervision, as we can mark the whole cluster as problematic, but now the administrator needs to scan through all

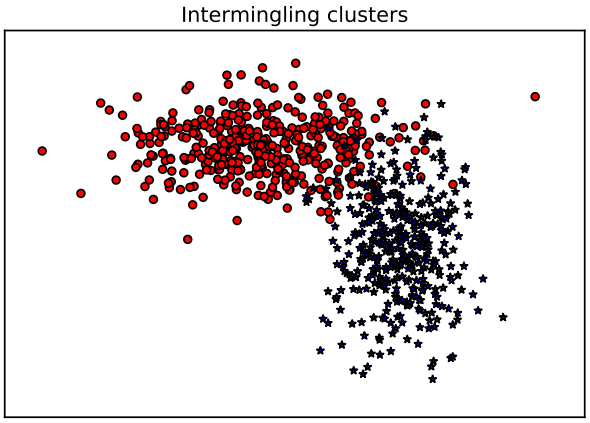

Figure 4.1: An example of intermingling clusters in two dimensions.

the connections in the combined cluster. Clusters with conflicting classification could get special handling, such as rerunning DBSCAN with a smaller value for ϵ for just the points in the problematic cluster, or using an altogether different method for the data points in the problematic area.

A completely unsupervised classification is not achievable with clustering, or perhaps any method, and there will always be some connections in a gray area where judgment calls need to be made.

4.2 Towards a production system

In a production implementation of this system, anomalous connections from clusters and outliers should be tagged differently; mislabelings due to cluster association are more serious than being tagged anomalous because the connection was an outlier. Any novel connection is an outlier at first, so presumably new outliers would receive less scrutiny initially. Different organizations should be able to adopt their own policies to deal with these.

4.2.1 User interface for CryptoAuditor

Because tagging just outliers is not enough, as described in Section 2.3.1, a convenient and fast user interface to go through the connections that the system has raised is required. Without such an interface, the system will have no value.

New outliers and clusters that have not been classified should be raised to the administrators' attention. For clusters, a sampling of the connections as well as aggregate data about them should be available; for example, the cluster size and common features between the connections should be visible. It should be possible to drill down to the connections to allow a closer study of the actions taken.

Tagging outliers and connections in the clusters should be very easy, as should be correcting any misclassifications. Restarting the clustering should be simple, but as it might take a while to obtain the results, the user interface should indicate this.

4.3 Further work

Feature extraction from the connection data was very simplistic in this work. For some features, the detection could be much improved by using a heuristic or a separate classifier. For example, a shell connection used for file transfer by piping the data over could be tagged as such by using a specific classifier over the channel data.

At the time of the experiments, CryptoAuditor had not been in the market very long and there were few customer installations. These days, the product is mature and in production use in a large number of organizations; partnering with select customers to obtain actual production data could be used to gather an anonymized set of connection data, and experiments on clustering and other methods could be performed using that data.

Time complexity for the clustering with the `scikit-learn` DBSCAN implementation was not what was expected from the algorithm descriptions in the literature. An alternative implementation should be used to verify the

result.

4.3.1 Host-based data

The data captured by CryptoAuditor can only partially model what has happened in the target servers. A software component could be added to the target hosts which could give accurate information on the monitored users' activities. This "agent" would run in the background monitoring system activities and perhaps run with elevated privileges to monitor the system calls in the processes the users launch. This data, combined with the monitored data, would allow a far better accuracy on misuse detection. For example, such a system would not need to use heuristics on whether the user executed a command to elevate privileges, mapped to dimension `has_su`.

Similarly, the system could gather general events in the target system and align them with the activities that have occurred on the line. A less intrusive example of this are the system log events, whereas using `ptrace` to capture system calls is much more invasive. An example of a system is described in Hofmeyr et al. (1998), and the classification given by that method could be included, for example, as a separate dimension in the clustering.

4.3.2 Improving the system performance

In this section, we introduce some mechanisms to improve the system performance for a production system, should the performance fall, for example, due to the number of audited connections.

Incremental updates

It would be beneficial if the clustering could be incrementally updated, that is, it would not make any difference to the end result of the clustering whether the data is processed in one pass or if the connections are processed, for example, 1000 connections at a time.

In practice, there are two problems. The scaling to zero mean and unit variance, as described in the previous section, cannot be done incremen-

tally. A rolling variance calculation would be possible, as described in Knuth (1997)[Section 4.2.2, p. 232], but all the old samples would need to be updated after adding samples, resulting in no improvements in the computational complexity. However, when the number of data rows is already large, for example, over a million entries, updating with small numbers of connections does not change the mean or variance that much. This means we can delay the recomputation for the whole set while the updated mean and variance have not deviated too much from the original.

Another problem is updating the clustering. In practice, this would require merging of clusters, which is not supported by the `scikit-learn` DBSCAN implementation. In a production system, this can naturally be avoided, as we can keep persistent state to continue updating the clustering. An incremental version of DBSCAN, described in Ester et al. (1998), can be used here.

It would therefore be feasible to implement incremental updates, recalculating the clustering when the amount of new data would warrant it or based on a time interval, once a week, for example. The recalculation could be done without interfering with the operation of the system by creating an atomic database copy and using different instances of CryptoAuditor to perform the clustering; the copy used could be the backup of the database, the creation of which is already done in production systems.

Sampling

Currently, production installations are reported to have stored connections in the order of hundreds of thousands of trails. The amount of data stored by the trails creates a bottleneck for the system performance, as the data usually needs to be backed up, and the trails need to be indexed and searchable. When these limits are approached, it should still be manageable by a well-behaving implementation with $O(n \log n)$ performance in time and space. If the number of connections surpasses feasible limits for the used clustering algorithms, sampling can be used.

CURE, or Clustering Using Representatives, described in Guha et al.

(1998), itself an algorithm with $O(n^2 \log n)$ complexity, can be used instead of DBSCAN. Similarly, random sampling from the connections could be used in conjunction with DBSCAN, but the connections are not guaranteed to be clustered identically due to the change in density of the clusters, a critical component of DBSCAN. However, intuitively it should not cause overt problems, as the reduced density should only cause an increase in the number of clusters, and perhaps outliers, which would occur in the data anyway with fewer connections.

In practice, this could be implemented by starting to sample after the clustering speed starts to plummet after some large number of connections has been gathered. Samples would be taken from connections that have already been presented to the administrators. They are therefore expected to have received some scrutiny. We would include all admin-labeled connections in the sample. This should still allow reasonably precise clusters. In the unlikely event of too many administrator-labeled connections, we could just use n latest connections, with n set to some sensible upper limit.

4.3.3 Other classes of anomalies

In this work, we have only considered point anomalies. Attackers who have a continued access to user credentials can easily adapt to use multiple connections to avoid detection, moving the detection problem towards collective and contextual anomalies.

4.4 Conclusion

In this work, we have implemented a semi-supervised classification system for detecting anomalous connections from normal ones. We compared two clustering algorithms, DBSCAN and k-means++, for the task. We analyzed DBSCAN's performance in the classification when the parameter ϵ and the number of the connections were varied. We outlined a set of features needed for a practical implementation of the system.

As an unsupervised approach where the accumulated data does not need

to be specifically categorized by the administrators, and only outliers would be treated as anomalous connections, a clustering approach alone would not really work. Very few of the outliers were really anomalous; they were just rare in the collected data. The attack connections were reasonably well clustered.

As a tool to reduce administrator work, this method has merits. Instead of wading through all the connections, the administrators only need to deal with a set of random representatives from the clusters, reducing the amount of manual labor. Instead of checking 50000+ connections, the administrator needs to check a few hundred. This is a marked improvement.

Bibliography

Arthur, D. & Vassilvitskii, S. k-means++: The advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

Barber, D. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, New York, NY, USA, 2012. ISBN 0521518148, 9780521518147.

Barnes, R. & Thomson, M. & Pironti, A. & Langley, A. Deprecating Secure Sockets Layer Version 3.0. RFC 7568, RFC Editor, June 2015. URL: `https://tools.ietf.org/html/rfc7568`.

Chandola, V. & Banerjee, A. & Kumar, V. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

Dierks, T. & Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, RFC Editor, August 2008. URL: `https://www.ietf.org/rfc/rfc5246.txt`.

Ester, M. & Kriegel, H.-P. & Sander, J. & Xu, X. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. *Kdd*, volume 96, pages 226–231, 1996.

Ester, M. & Kriegel, H.-P. & Sander, J. & Wimmer, M. & Xu, X. Incremental clustering for mining in a data warehousing environment. *VLDB*, volume 98, pages 323–333. Citeseer, 1998.

Forcier, J. Fabric – pythonic remote execution. URL: `http://docs.fabfile.org/`, 2017. Accessed: 2017-03-14.

Friedman, J. H. & Bentley, J. L. & Finkel, R. A. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

Guha, S. & Rastogi, R. & Shim, K. Cure: an efficient clustering algorithm for large databases. *ACM Sigmod Record*, volume 27, pages 73–84. ACM, 1998.

Hofmeyr, S. A. & Forrest, S. & Somayaji, A. Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180, 1998.

Kamp, P.-H. & Watson, R. Jails: Confining the omnipotent root. Technical Report, May 2000. URL: `http://phk.freebsd.dk/pubs/sane2000-jail.pdf`. Accessed: 2017-03-14.

Knuth, D. E. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. ISBN 0-201-89684-2.

Lehtinen, S. & Lonvick, C. The Secure Shell (SSH) Protocol Assigned Numbers. RFC 4250, RFC Editor, January 2006. URL: `https://www.rfc-editor.org/rfc/rfc4250.txt`.

Leskovec, J. & Rajaraman, A. & Ullman, J. D. *Mining of massive datasets*. Cambridge University Press, 2014.

Leung, K. & Leckie, C. Unsupervised anomaly detection in network intrusion detection using clusters. *Proceedings of the Twenty-eighth Australasian conference on Computer Science*, volume 38, pages 333–342. Australian Computer Society, Inc., 2005.

Microsoft Corporation. Remote Desktop Protocol: Basic Connectivity and Graphics Remoting. Technical Report, Microsoft Corporation, October 2016a. URL: `https://msdn.microsoft.com/en-us/library/cc240445.aspx`.

Microsoft Corporation. Remote Desktop Protocol: File System Virtual Channel Extension. Technical Report, Microsoft Corporation, July 2016b. URL: https://msdn.microsoft.com/en-us/library/cc241305.aspx.

Oracle Corporation. Test, develop, and demonstrate across multiple platforms on one machine. URL: https://www.oracle.com/virtualization/virtualbox/resources.html, 2017. Accessed: 2017-03-14.

Pedregosa, F. & Varoquaux, G. & Gramfort, A. & Michel, V. & Thirion, B. & Grisel, O. & Blondel, M. & Prettenhofer, P. & Weiss, R. & Dubourg, V. & Vanderplas, J. & Passos, A. & Cournapeau, D. & Brucher, M. & Perrot, M. & Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Portnoy, L. & Eskin, E. & Stolfo, S. Intrusion detection with unlabeled data using clustering. *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMAS-2001)*. Citeseer, 2001.

Ptacek, T. H. & Newsham, T. N. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical Report, DTIC Document, 1998.

SIGKDD. KDD Cup 1999 Data. URL: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999. Accessed: 2017-01-20.

SIGKDD. 2014 SIGKDD Test of time award. URL: http://www.kdd.org/News/view/2014-sigkdd-test-of-time-award, 2014. Accessed: 2017-01-20.

SSH Communications Security Corp. CryptoAuditor. URL: https://www.ssh.com/products/cryptoauditor/, 2017a. Accessed: 2017-03-14.

SSH Communications Security Corp. Universal SSH Key Manager. URL: https://www.ssh.com/products/universal-ssh-key-manager/, 2017b. Accessed: 2017-03-16.

The IEEE and The Open Group. uuencode – The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008, 2016 Edition. URL: http://pubs.opengroup.org/onlinepubs/9699919799/utilities/uuencode.html, 2016. Accessed: 2017-03-13.

Turner, S. & Polk, T. Prohibiting Secure Sockets Layer (SSL) Version 2.0. RFC 6176, RFC Editor, March 2011. URL: https://tools.ietf.org/html/rfc6176.

Wagner, D. & Dean, D. Intrusion detection via static analysis. *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 156–168. IEEE, 2001.

Ylönen, T. & Lehtinen, S. SSH File Transfer Protocol. Internet-draft, RFC Editor, October 2001. URL: https://www.ietf.org/archive/id/draft-ietf-secsh-filexfer-02.txt.

Ylönen, T. & Lonvick, C. The Secure Shell (SSH) Protocol Architecture. RFC 4251, RFC Editor, January 2006a. URL: https://www.rfc-editor.org/rfc/rfc4251.txt.

Ylönen, T. & Lonvick, C. The Secure Shell (SSH) Authentication Protocol. RFC 4252, RFC Editor, January 2006b. URL: https://www.rfc-editor.org/rfc/rfc4252.txt.

Ylönen, T. & Lonvick, C. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, RFC Editor, January 2006c. URL: https://www.rfc-editor.org/rfc/rfc4253.txt.

Ylönen, T. & Lonvick, C. The Secure Shell (SSH) Connection Protocol. RFC 4254, RFC Editor, January 2006d. URL: https://www.rfc-editor.org/rfc/rfc4254.txt.

Ylönen, T. & Turner, P. & Scarfone, K. & Souppaya, M. NISTIR 7966 – Security of Interactive and Automated Access Management Using Secure Shell (SSH). URL: http://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.7966.pdf, 2015. Accessed: 2017-03-16.

Zanero, S. & Savaresi, S. M. Unsupervised learning techniques for an intrusion detection system. *Proceedings of the 2004 ACM symposium on Applied computing*, pages 412–419. ACM, 2004.

Zweig, M. H. & Campbell, G. Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine. *Clinical chemistry*, 39 (4):561–577, 1993.