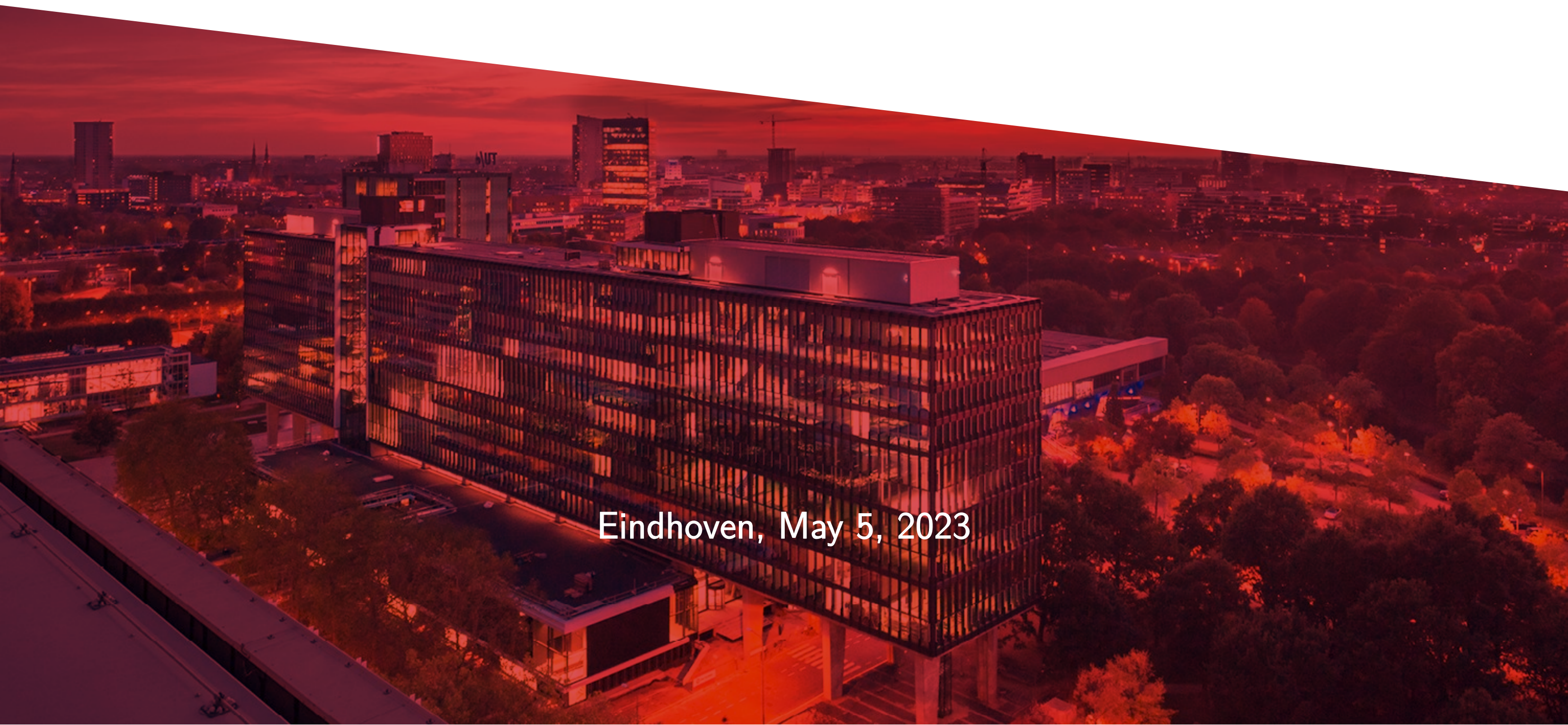
Anomaly Based Detection of IMAP Host Logs

2IMS40,
Intrusion Detection Laboratory - Q3 (2023)

Group 7

Full Name	**Student ID**
Ruben Biskupec	1878638
Joni Bimbashi	1415891
Adriana Radu	1418548
Tarik Doganer	1864165
Andrei Oprea	1768905



Eindhoven, May 5, 2023

Contents

1 | Introduction

This report describes the workings of group 7 for the final project for the course Intrusion Detection Laboratory. The aim of the project is to develop a fully-fledged intrusion detection system (IDS) for a given dataset, either signature or anomaly-based. Our group was assigned to create an anomaly-based IDS for host logs from IMAP servers, gathered from the TU/e's network.

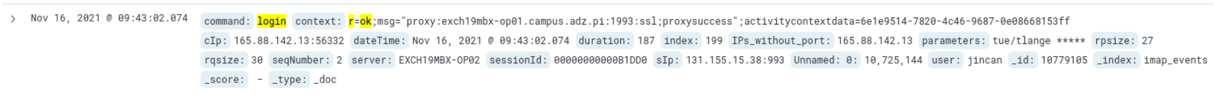
1.1 | IMAP host logs

The gathered data ranges from the 9th of November 2021 to the 16th of November 2021 and describes IMAP client commands. The main commands of higher relevance for this project are the login commands along with the select and fetch commands. The dataset contains 10,779,175 events, out of which 2,278,031 logs are login commands (9,929 are successful and 2,268,102 are failed attempts). From the start, by examining the notable dissimilarity between the number of login attempts that have succeeded versus those that have failed, we can deduce that the malicious activity involves attempting to gain unauthorized access to IMAP accounts. The exact steps we followed to analyze and formulate the attack are described in Section 2.

1.1.1 | Login command

When a client wants to connect to an IMAP server, it sends a **Login command** by providing the username and password of the user mailbox they want to access. The provided credentials are checked against the server's user database and, if they are correct, the server authenticates the client. Once the authentication is successful, a session is opened that associates the client with the respective user's mailbox.

After a successful login command, the server responds with an "OK" message. In an IMAP log, the response of the command can be seen in the context field as seen in 1.1. The session is then opened which allows the client to perform other commands such as the fetch command.



```
> Nov 16, 2021 @ 09:43:02.074   command: login context: r=ok;msg="proxy:exch19mbx-op01.campus.adz.pi:1993:ssl;proxysuccess";activitycontextdata=6e1e9514-7820-4c46-9687-0e08668153ff
                                cIp: 165.88.142.13:56332 dateTime: Nov 16, 2021 @ 09:43:02.074 duration: 187 index: 199 IPs_without_port: 165.88.142.13 parameters: tue/tlange ***** rpsize: 27
                                rqsize: 30 seqNumber: 2 server: EXCH19MBX-OP02 sessionId: 00000000000B1DD0 sIp: 131.155.15.38:993 Unnamed: 0: 10,725,144 user: jincan _id: 10779105 _index: imap_events
                                _score: - _type: _doc
```

Figure 1.1: Successful login event

1.1.2 | Select command

The **Select command** is used by a client to select a mailbox from their account. Once a select command is issued, the server responds with metadata about that specific mailbox and places the client in the picked folder. The client then has access to the messages contained in the mailbox and can perform various commands on them such as fetching or deleting.

1.1.3 | Fetch command

The **Fetch command** is used to retrieve one or more emails or specific parts of the message body from the mailbox. The fetch command uses the sequence number to identify the email to be retrieved. This number is assigned by the server and it can change depending on modifications in the mailbox such as new messages added, messages deleted, or moved.

For more reliable identifying and tracking messages, the user can perform the **uid + fetch command** which retrieves emails based on their unique identifier which remains constant no matter the changes made to the mailbox.

Throughout the report, we will see that the fetch commands are used maliciously to exfiltrate data from the user's accounts that the attacker managed to successfully log in to.

1.1.4 | Login errors

Besides successful logins, the data contains a high number of **failed login attempts**. When a client fails to log in, the IMAP server responds with an error message indicating that the login has failed and the reason for the failure. The returned error message varies depending on the exact cause of the login failure. In the IMAP logs, these error messages are present in the context field.
We have identified four different types of login errors.

The first login error is the **proxy error** and it appears 7140 times in the data. An example of this server response can be seen in Listing 1.

Listing 1: Proxy error message

```
r="3 no login failed.";msg=proxy:exch19mbxop01.campus.adz.pi:1993:ssl;
errmsg=proxynotauthenticated
```

The *msg=* field of the server response indicates that the error took place while trying to connect to the IMAP servers through a proxy server, and the *errmsg=* field specifies that the proxy was not authenticated, which may have caused the failed login attempt. In general, this message indicates that the server was not able to authenticate the login attempt.

The next login error is the **pre-authentication timeout error** which occurs only 14 times. The context for this error looks as follows:

Listing 2: Pre-authentication timeout error

```
r="xm001 no login failed.";msg=logonfailed:logondenied;
errmsg=logonfailed:logondenied:preauthtimeout
```

This server response specifies that the login attempt failed due to a pre-authentication timeout. Pre-authentication requires the client to be authenticated on the server before being able to access its resources. The alphanumeric code `xm001` represents the tag generated by the client for this specific command and acts as an identifier. According to IMAP protocol standards, for each command, a different unique tag should be generated by the client. However, through further investigation of the data it was noticed that specific tags repeat for a vast number of commands. This discovery is further explained in Subsection 2.1.

The third login error message is the **active directory sites error** which appears 3542 times. Due to the response containing a lot of details, the shown error in Listing 3 is truncated.

Listing 3: Active directory site error

```
r="z no [error="microsoft.exchange.data.storage.wrongserverexception:
the user and the mailbox are in different active directory sites. -->
microsoft.mapi.mapiexceptionmailboxintransit:mapiexceptionmailboxintransit:
detected site violation (hr=0x0, ec=1292)" authresult=0] login failed.";
msg="user:legacydn, recipienttype: usermailbox,
recipienttypedetails: monitoringmailbox,
selected mailbox: display name: healthmailbox-exch19mbx-op02-o365,
mailbox guid: 2dce92a3-aa1b-4488-b8c1-b64f85708dab,
database: 698c385b-9b9a-41e8-b780-798add39bcf8,
location: serverfqdn: exch19mbx-op01.campus.adz.pi,
serverversion: 1942127514, databasename: o365;
exstk=h4siaaaaaaaeao1x32/boax+p2d/
```

As the name of the error suggests, the login failed due to the client and the mailbox being in different active directory sites, thus causing a site violation. Additionally, the server response contains auxiliary information such as the exceptions encountered which are specified in the *error=* field, the mailbox the client wants to access which is shown in the *selected mailbox:* field, the mailbox database, the server version, and more. The error message ends with a string, composed of hexadecimal characters, which represents a stack trace and is specified in the field *exstk=*.

The last identified login error is the **failed authentication** error. This response message appears 2,257,405 times and is the most prominent server response for failed login attempts. An example of this error message is seen in Listing 4.

Listing 4: Authentication failure

```
r="xm001 no login failed.";msg=logonfailed:logondenied;errmsg=logonfailed:logondenied
```

The response indicates that the login attempt has failed due to incorrect login credentials. As mentioned before, the prefix `xm001` is used to identify the command and is linked to a specific user.

2 | Analysis & Identification

2.1 | Analysis process

A general view of the dataset shows that on average, a single IP address is used by around unique 10 users, while a single user is accessed on average by approximately 153 unique IP addresses. However, there are 52144 unique IP addresses and only 3786 users in the dataset. This discrepancy in the ratio between users and IPs could potentially be explained by the use of portable devices like mobile phones, which may connect to different networks and generate new IP addresses.

Our first step in exploring the data is to plot the count of commands issued over time, allowing us to gain insights into the dataset at a general level. This plot can be seen in Figure 2.1.

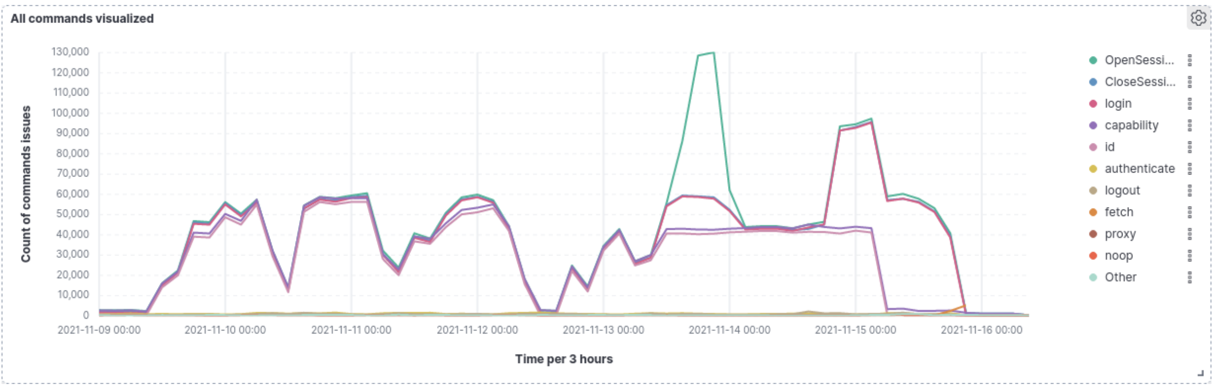


Figure 2.1: Commands issued over time

Upon initial analysis, we observe a spike in OpenSessions commands on November 13th between 6:00 pm and 9:00 pm. We take a closer look at the peak in Figure 2.2.

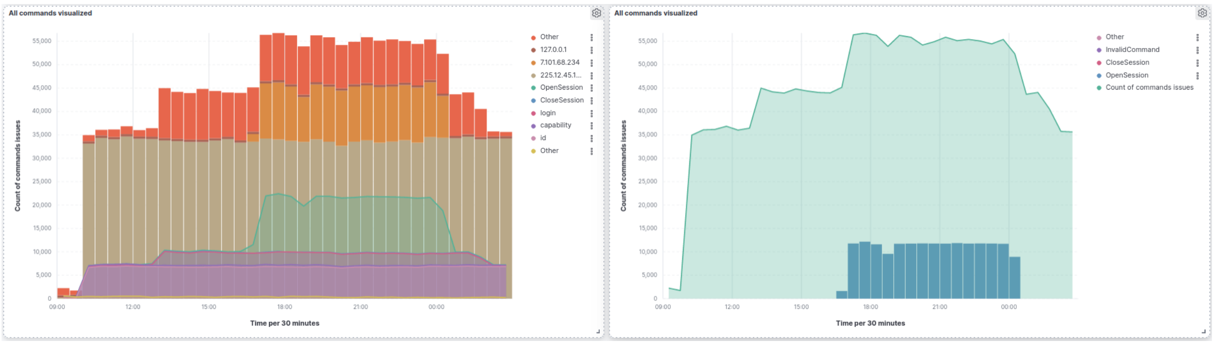


Figure 2.2: Overview of the Peak from Figure 2.1

The left graph in Figure 2.2 displays the IPs that accessed the IMAP server during the 6:00 pm - 9:00 pm timeslot. Among these IPs, one that stands out is `225.12.45.153`, depicted in beige color. This IP shows a constant number of requests, and it is associated with an entity that accessed the user `striddle` throughout the IMAP logs. Later in this section, we will delve into its excessive behavior in the IMAP server. Another IP that draws attention is `7.101.68.234`, represented in light orange color, which only has activity in this timeframe and issued solely OpenSessions commands, as shown in the right graph of the same figure. The peak in the right graph corresponds to this IP. The dark orange portion in the bar chart represents other IPs with insufficient data to draw any conclusion. Our initial hypothesis is that `225.12.45.153` could be an attacker attempting to create noise in the dataset, making it harder to identify any real attacks.

We continue with our analysis by plotting the type of commands issued by users as shown in Figure 2.3. We observe that a user named `striddle` issued the highest number of commands, accounting for 72.13% of all entries with a user field in the dataset. We also notice that this user issued an equal number of CloseSession and login commands, with a CloseSession command following every login command. To improve clarity, we filter these entries, resulting in percentage entries depicted in the following two graphs. The percentages remain approximately the same.

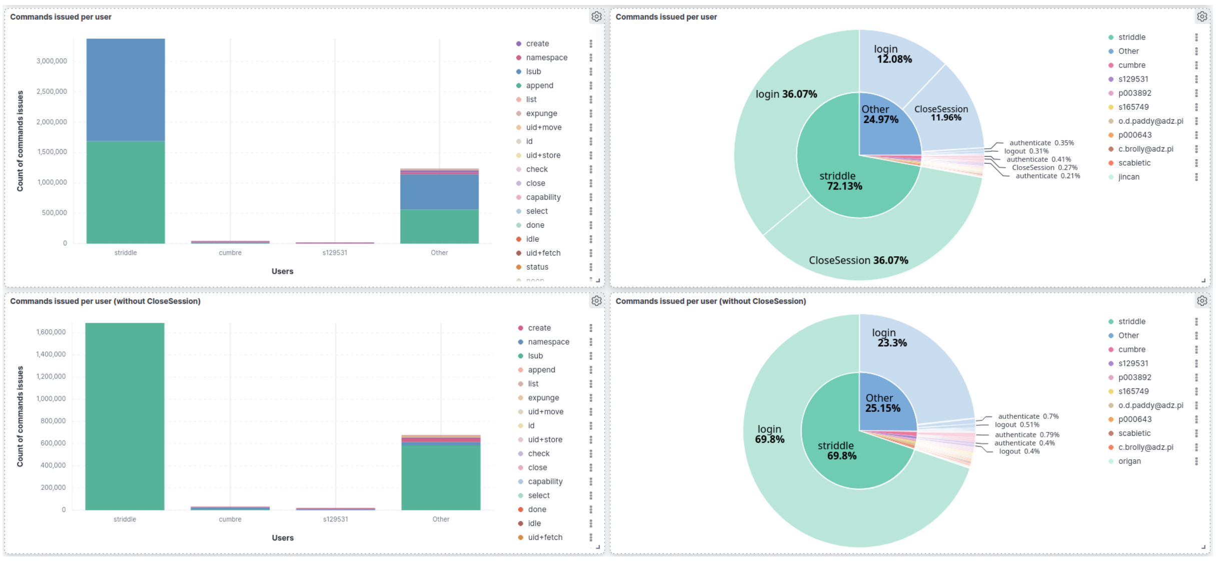


Figure 2.3: Commands issued by user (with and without CloseSession)

Although these observations indicate the possibility of an intruder or malicious actor, their high presence in the data overshadows other information. As a result, we have decided to temporarily exclude the user `striddle` from the data in order to focus on investigating the other aspects of it. In the following graph 2.4, we present the same visualization with the user `striddle` being excluded from it. This shows more insights into the *Other* field of the previous pie chart seen in Figure 2.3.

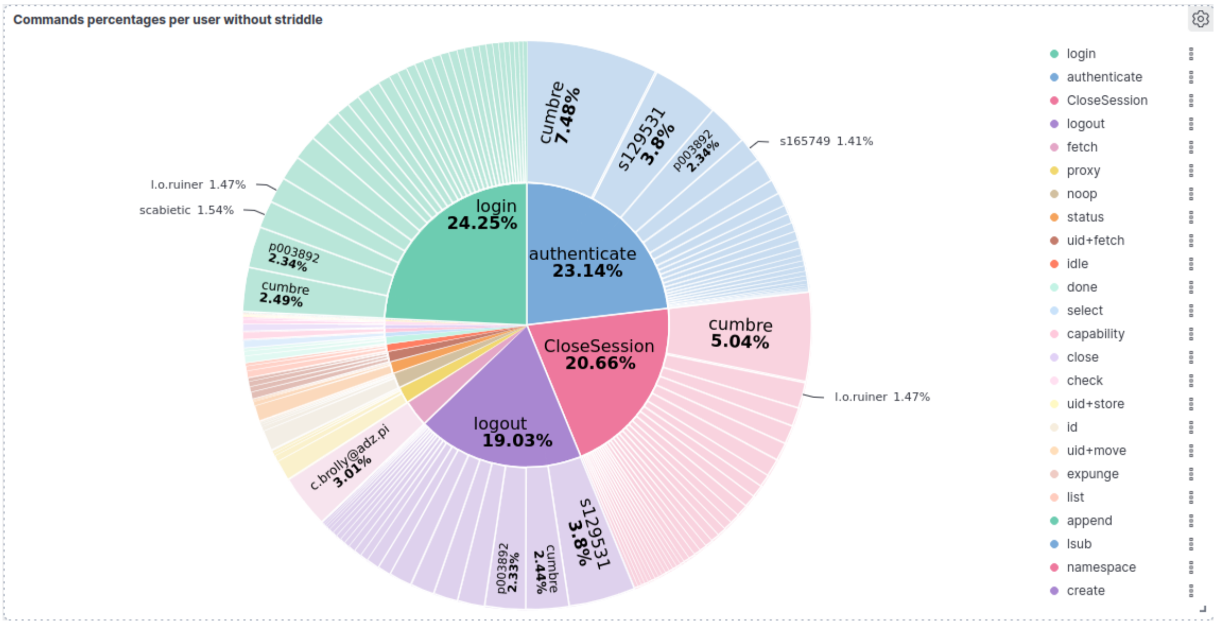
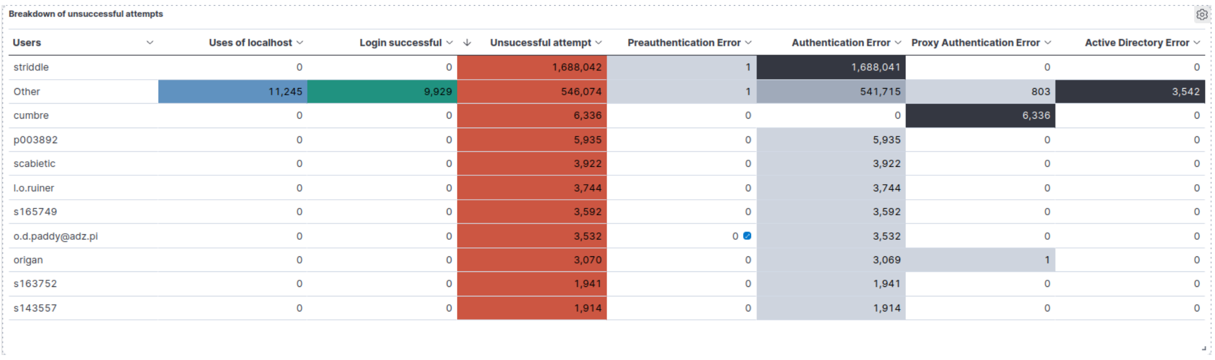


Figure 2.4: Commands issued by users without the user "striddle"

Upon analyzing the data, we notice that the user `cumbre` frequently issues login, authentication, and CloseSessions commands. However, we are unable to draw any significant conclusions from this data, prompting us to examine other possible inconsistencies in the dataset. To do this, we use a table that displays a breakdown of successful and unsuccessful login attempts to the server which can be seen in Figure 2.5 and Figure 2.7. These tables contain only the top 10 users that have such behavior.

From table 2.5, we note that the user `striddle` has the highest number of login errors, with all of them being authentication failures 4, indicating incorrect credentials, and a single pre-authentication timeout error 2. Furthermore, the user **`cumbre`** appears to share only proxy errors 1. This behavior is highly unusual and therefore requires further investigation. It is to be noted that the table shows the sum of commands, and drawing conclusions based solely on these numbers may be misleading. As a final aspect, we examine the rate at which these commands are issued. If the rate is consistent and high, it may suggest that an automated script is attempting to crack the password for the user **`striddle`**.



Breakdown of unsuccessful attempts

Users	Uses of localhost	Login successful	Unsucessful attempt	Preauthentication Error	Authentication Error	Proxy Authentication Error	Active Directory Error
striddle	0	0	1,688,042	1	1,688,041	0	0
Other	11,245	9,929	546,074	1	541,715	803	3,542
cumbre	0	0	6,336	0	0	6,336	0
p003892	0	0	5,935	0	5,935	0	0
scabietic	0	0	3,922	0	3,922	0	0
l.o.ruiner	0	0	3,744	0	3,744	0	0
s165749	0	0	3,592	0	3,592	0	0
o.d.paddy@adz.pi	0	0	3,532	0	3,532	0	0
origan	0	0	3,070	0	3,069	1	0
s163752	0	0	1,941	0	1,941	0	0
s143557	0	0	1,914	0	1,914	0	0

Figure 2.5: Unsuccessful login attempts

When we plot these 10 users on failed login attempts by the hour, we notice some intriguing patterns. In Figure 2.6, we present two plots of these users where the top graph includes the user **`striddle`** while the bottom one excludes this user. The black line indicates the total number of commands issued by these users together. In the top graph, the user **`striddle`**, represented with a green line, counts for the most part of the commands. Furthermore, **`striddle`** seems to issue a consistent number of commands at specific intervals. After November 14th, failed login attempts become more pronounced and follow a regular pattern, as evidenced by the spikes for users **`origan`** and **`sciabietic`**. In contrast, **`cumbre`**'s login attempts are sporadic, occurring in large volumes followed by periods of drops before another wave of high-volume login attempts.

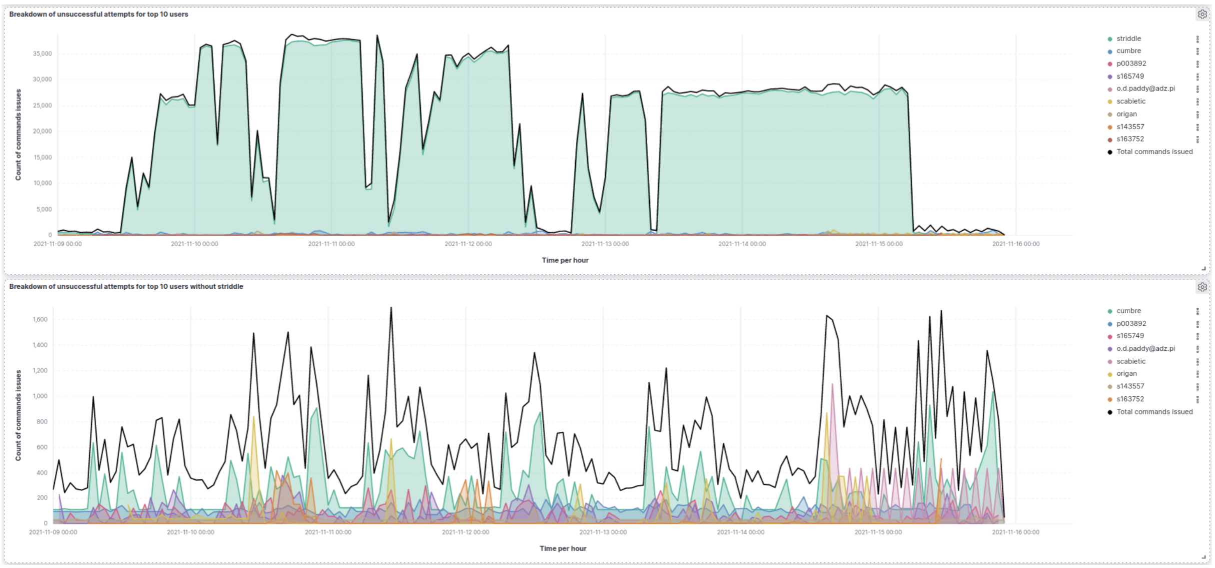
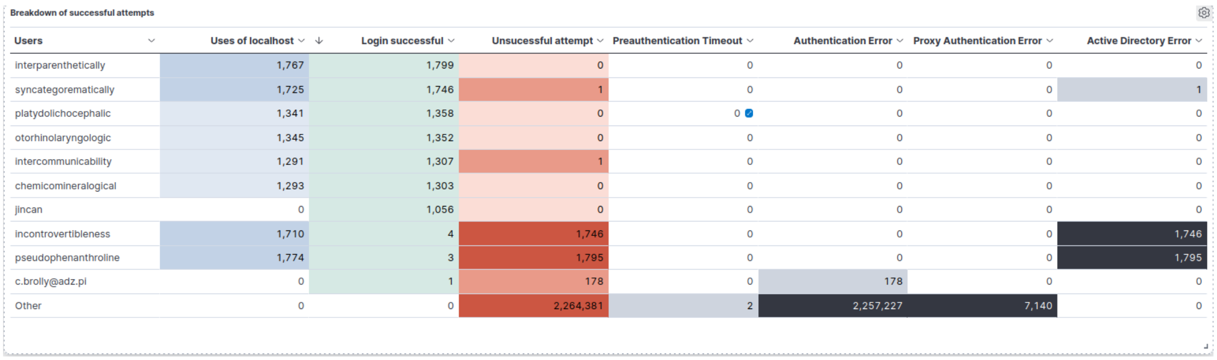


Figure 2.6: Login attempts for top 10 users per hour

Looking at Figure 2.7, we can see the top 10 users with the most successful login attempts. Out of the entire dataset, only 10 users were able to log in successfully. It is interesting to note that eight out of these 10 users appeared to be accessing the IMAP server locally, with six of them having at most 1 unsuccessful attempt. The remaining two local access users have limited login success, only managing to log in a maximum of 4 times after numerous failed attempts. However, these failed attempts arise from Active Directory errors instead of denied login errors (such as invalid credentials). It is possible that these accounts belong to a system admin for testing and configurations.



Breakdown of successful attempts

Users	Uses of localhost	Login successful	Unsucessful attempt	Preauthentication Timeout	Authentication Error	Proxy Authentication Error	Active Directory Error
interparenthetically	1,767	1,799	0	0	0	0	0
syncategorematically	1,725	1,746	1	0	0	0	1
platydolichocephalic	1,341	1,358	0	0	0	0	0
otorhinolaryngologic	1,345	1,352	0	0	0	0	0
intercommunicability	1,291	1,307	1	0	0	0	0
chemicomineralogical	1,293	1,303	0	0	0	0	0
jincan	0	1,056	0	0	0	0	0
incontrovertibleness	1,710	4	1,746	0	0	0	1,746
pseudophenanthroline	1,774	3	1,795	0	0	0	1,795
c.brolly@adz.pi	0	1	178	0	178	0	0
Other	0	0	2,264,381	2	2,257,227	7,140	0

Figure 2.7: Successful login attempts

The remaining two users who do not show any local access are `jincan`, who has no failed login attempts, and `c.brolly@adz.pi`, who only manages one successful login attempt after 178 unsuccessful attempts. We will refer to this user as `c.brolly` for the rest of the report.

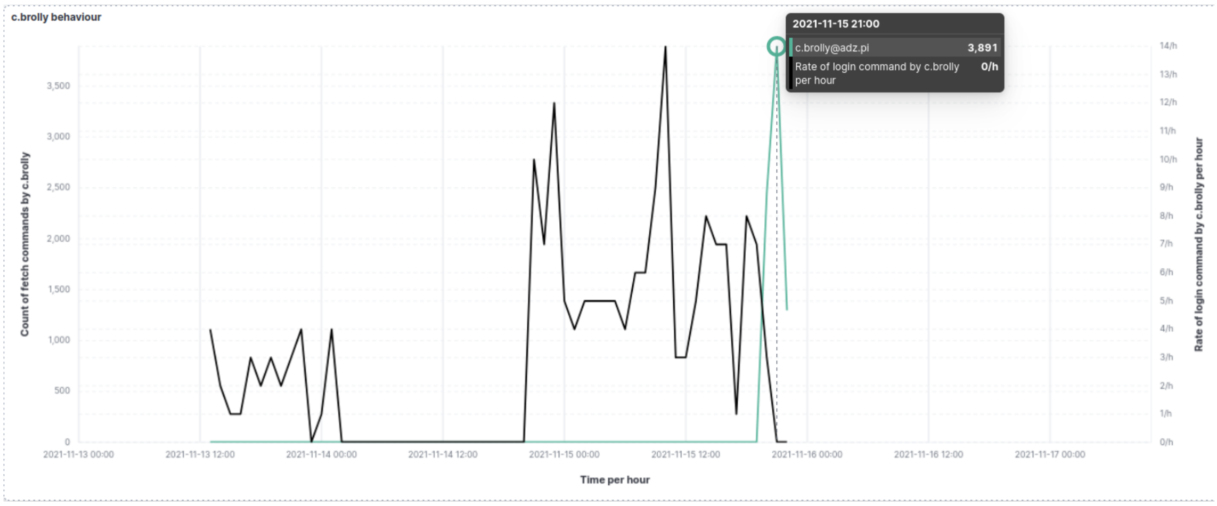
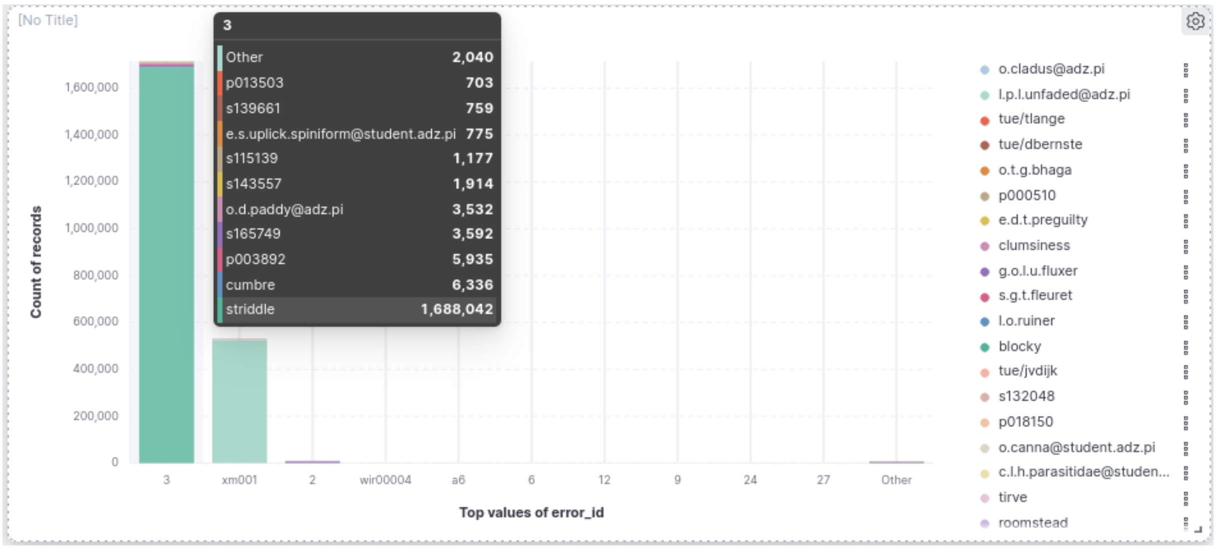


Figure 2.8: "c.brolly" behavior

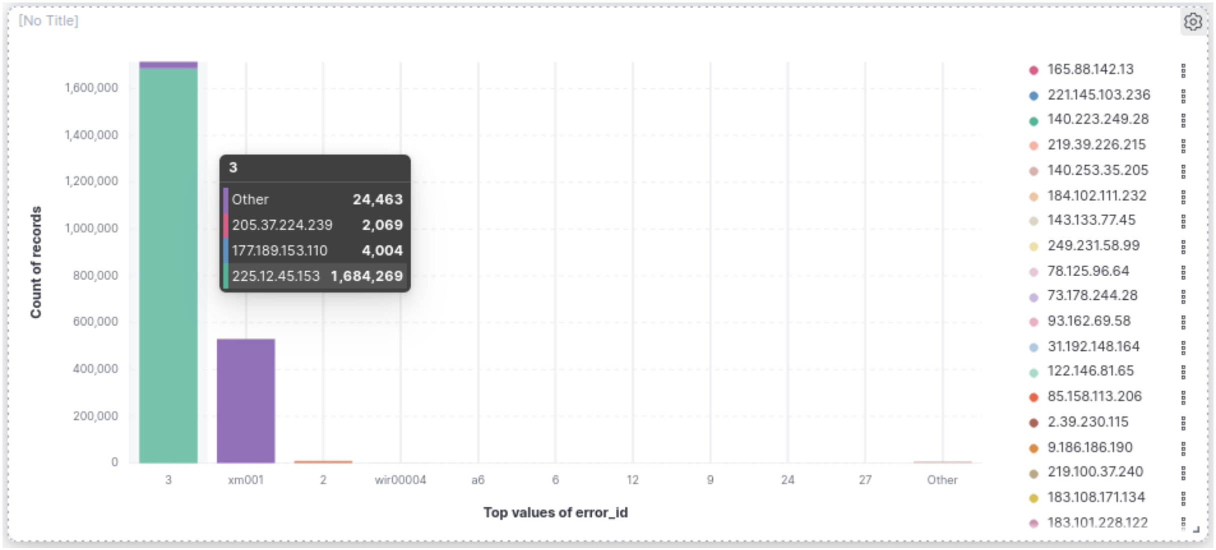
Figure 2.8 shows the pattern of issued commands by `c.brolly`, specifically the rate at which the user issues fetch and login commands. The black line represents the rate of login commands while the green line depicts the rate of fetch commands. After one successful login, the next IMAP commands issued by `c.brolly` hit a total of approximately 5000 fetch commands in the span of two hours. This observation is also demonstrated in Figure 2.8 where the rate of logins per hour is in the range of 3-14 attempts per hour followed by a sudden drop after the first appearance of the fetch commands in the logs. Afterward, the user seems to show no signs of activity. This type of behavior seems to resemble a data exfiltration attack where the attacker tries to gather as much information as possible. We make notice of this behavior for a possible anomaly detection later.

Finally, we take a look into the error tags of the login IMAP commands mentioned in Subsection 1.1.4. In Figure 2.9 and 2.10 we visualize these error codes by user and IP respectively.



3	
Other	2,040
p013503	703
s139661	759
e.s.uplick.spiniform@student.adz.pi	775
s115139	1,177
s143557	1,914
o.d.paddy@adz.pi	3,532
s165749	3,592
p003892	5,935
cumbre	6,336
striddle	1,688,042

Figure 2.9: Unsuccessful login error codes by user



3	
Other	24,463
205.37.224.239	2,069
177.189.153.110	4,004
225.12.45.153	1,684,269

Figure 2.10: Unsuccessful login error codes by IP

The user `striddle` with the IP address `225.12.45.153`, as previously discussed, seems to be the origin of the most reported response error code: `3`. Although there are other IPs and users related to this error code, their counts are relatively low compared to `striddle`. Nonetheless, it does not exclude the possibility that this entity `striddle` attempted to access other accounts. An interesting observation comes also from the fact that this error is also generated by the user `cumbre`.

Furthermore, the error code `xm001` displays a different pattern than `3`. This error is also generated by the user `c.brolly` and includes access from more IPs than `3`. Instead of displaying the overall count of these error codes, we present an alternative visualization in Figure 2.11 that sorts these error codes by the unique count of IPs and users, which supports our observation.

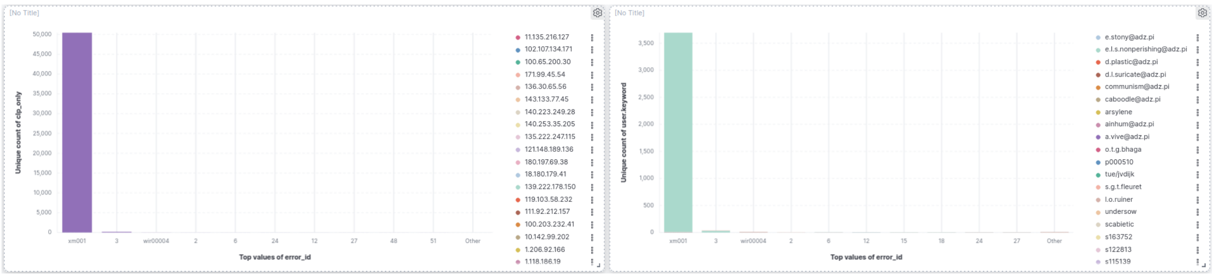


Figure 2.11: Unsuccessful login error codes by a unique user and IP

In contrast to the previous error code, `xm001` appears to be more widely spread across different users and IPs. It is the most frequently occurring error code in this type of visualization. Based on our analysis, we believe that this error code is likely associated with a single attacker. This is because, according to the IMAP specification, the tag is generated by the client, not the server. This characteristic can be leveraged to create a new dataset that can be used to evaluate an anomaly-based detection method. By using this error code, one could potentially differentiate between true and false attacks. Finally, we also make note that the error tag 2is generated by the users `origan` and `sciabietic`.

2.2 | Attack rationale

From the analysis performed, we reached several conclusions that helped us form our attack hypothesis. Firstly, the large volume of unsuccessful login attempts serves as a primary indicator of malicious behavior, respectively a brute force attack. Our theory is reinforced by the evidence of a persistent and rapid succession of login tries from various users, throughout several days. Next, the behavior of some "users" of changing their IP address and their ports with each command raised further suspicion. Such activity is not seen in the commands issued by the users deemed legitimate.

Looking at the context of the login error messages 4, some alphanumerical tags repeat instead of being unique for each command. Given that the tags are generated by clients to identify the issued command, we can correlate this to a malicious actor reusing the same login command numerous times for multiple users. This behavior is similar to a **password spraying attack**, using a trial-and-error approach to gain unauthorized access to accounts.

In a **password spraying attack**, the threat actor usually has a list of emails or usernames for which they attempt to log in using the same password. The process is repeated with a new password until a successful login.

After successfully gaining access to an account, the attacker uses the fetch IMAP commands to retrieve data from a selected mailbox. The unusual number of fetch commands is a clear indication of data being exfiltrated.

2.3 | Impact Analysis

As stated above, the attacker becomes successful at logging in with the credentials of the user `c.brolly@adzz.pi`. After the login, we can see that there is a *select* command request logged in the server, which is followed by 7636 logs of *fetch* commands. We assume that the attacker downloaded sent and received emails of the user `c.brolly`, so it is possible that they got access to the user's sensitive information and correspondences.

The immediate impact until detection is that the attackers can observe the victim's private conversations and craft a sophisticated spear-phishing attack. The phishing is highly likely to succeed as they have access to the content and the recipients of emails so they can construct the attack message using `c.brolly`'s language and send embedded malware to a party who was expecting an email from that user, as a simple example.

Another immediate impact of the attack for the user `c.brolly` is credential stuffing. As they have a successful login, now they have a match with that password and email combination. Thus, they can try different sites assuming that `c.brolly` uses the same password for those as well. Until the user is notified of the attack, all of their accounts are at risk of being compromised unless they have 2-factor authentication.

Furthermore, it is out of the scope of our dataset, but the attackers could access other TU/e account services with the same credentials. So, they could launch a ransomware attack or inject malware into other servers if the compromise of the account was not detected by the admin group.

Finally, there could be some compliance violations for GDPR regulations which could lead to some legal and financial penalties concerning TU/e, along with the fact that the reputation of TU/e as an institution could be damaged in the long term, if the incident response was not decisive and immediate.

3 | Design & Implementation

3.1 | Design

Having identified the password spraying attack, we decided to design a detection system with a "blacklisting" approach on malicious IPs. The threshold of blacklisting relies on the number of consecutive failed login attempts in a certain time frame. In our most optimally tuned case, the parameters were adjusted as three consecutive login attempts in an hour for a single IP address. When the threshold is met, the IP address and the users who had failed attempts from that address are added to a "Detected IP Addresses" set and a "Suspicious Users" set, respectively. Thus, when an attacker is attempting a spraying attack and uses the same IP for single or multiple users in a time frame, our detection system will raise an alert for the IP. However, if a successful login attempt is performed before the threshold is reached for an IP address, the corresponding counter will reset assuming that the previous attempts were from honest users.

3.2 | Implementation

3.2.1 | Pre-processing

To make our dataset smaller and to prevent any memory problems, we created a sub-dataset with logs that only have the login commands. In addition, to be able to detect the login attempts from the same IP addresses, we split the *cIp* column which had data in the format "IP:port" into 2 separate columns. We saved it as another CSV file since these operations are computationally intensive and take a significant amount of time before each run.

3.2.2 | IpInfo Class & Objects

To keep track of login attempt counts for each IP address, we constructed a class called *IpInfo*. This class has a *counter* field, representing consecutive failed login attempts, along with the functions to increase it by 1 and reset it when a successful login attempt is detected. Moreover, to determine if the number of attempts is reached in the established timeframe parameter, the class has *start_timestamp* and *end_timestamp* fields, and a *get_timestamp_diff* function to calculate the difference between them. Finally, it has a *users* set so that the suspicious users who tried a failed login attempt from that IP address are added to the global *Suspicious Users* set when the threshold is met, along with the functions to get, add and reset users.

3.2.3 | Flagging Script

After the preparation steps, we finally run our script to flag and "blacklist" the IP addresses. Before iterating the log file row by row, we define our sets for the malicious IP and affected users, as well as our hyperparameters of login attempts and timeframe tolerance. Then, for each row, we initialize the IP, user, and timestamp variables. If the IP address is not previously seen in the logs yet, we create an *IpInfo* object with its timestamp and add that object to our dictionary, to be able to fetch it later when the same IP is detected in another log entry. Afterward, we check if there is *logondenied* keyword in the context field of the log. If so, we treat that entry as a failed login attempt and increase the counter of that IP address by 1.

Subsequently, we perform our threshold check; which initially only tested if the number of attempt criteria is met in the determined timeframe. Then, we added the check if the corresponding user of the entry has already been added to the *Suspicious Users* set. As a final improvement, we also included executing a subnet check, which utilizes a set of subnets already added for the suspicious IP addresses. For this, we suppose the IPs reside in a /24 subnet. Under those circumstances, the IP address (along with its subnet) and the users who had a failed login attempt using that IP is added to the *Detected IPs*, *Detected Subnets* and *Suspicious Users* set, respectively. Contrarily, if an *ok* message is detected in the context field, the counter and the user list of that IpInfo object are reset. Therefore, we detect and flag the malicious IP addresses which probably participated in the password spraying attack.

The script also includes a check to see if one of the affected users has a successful login attempt, in this case, the alarm can be sent immediately for a quick security response. Our solution identifies only 2 such occurrences, catching the suspect `c.brolly`. The script wrongfully detects the user `jincan` as a victim of the attack, due to the IP which it used. This can represent a point of further improvement, nonetheless as of now, it will not overload the security analysts.

4 | Evaluation

From the results of running the last version of the python detection script, we can construct the following confusion matrix:

4.1 | Confusion Matrix

CONFUSION MATRIX

TARGET / OUTPUT	Malicious	Non Malicious	SUM
Malicious	49829 97.59%	965 1.89%	50794 98.10% 1.90%
Non Malicious	163 0.32%	100 0.20%	263 38.02% 61.98%
SUM	49992 99.67% 0.33%	1065 9.39% 90.61%	49929 / 51057 97.79% 2.21%

Figure 4.1: Caption

4.2 | Metrics

To go more into detail, we can interpret the results as follows:

- **True positives (TP)**: The algorithm correctly identified 97.59% of the IPs as malicious. This is a good result as it indicates that the algorithm is able to accurately detect malicious IPs. If we look at the total, we can see that it identified correctly 99.67% of the malicious IPs.
- **False positives (FP)**: The algorithm incorrectly identified 0.32% of the non-malicious IPs as malicious. This amounts to 163 IPs across the whole dataset. While this is a relatively small number, it could still lead to unnecessary actions being taken against innocent IPs. Therefore, it is worth looking into ways to reduce the false positive rate further.
- True negatives (TN): The algorithm correctly identified 0.20% of the non-malicious IPs as non-malicious. While this may seem like a low number, it is important to note that the majority of the IPs in the list are actually malicious, so the number of true negatives is expected to be low.

- False negatives (FN): The algorithm incorrectly identified 1.89% of the malicious IPs as non-malicious. This is not a big concern because those IPs would probably not be enough to successfully conduct the password spraying attack. Still, it may be worth investigating ways to improve the algorithm's detection rate for malicious IPs.

4.3 | Improvements

Overall, the results suggest that the algorithm is performing well in terms of detecting malicious IPs, with a high true positive rate and a low false positive rate. However, there is still room for improvement in terms of detecting all the malicious IPs and reducing the false positive rate. It may be worth considering incorporating additional features or algorithms to improve the accuracy of the results.

When talking about improvements it is worth mentioning that there are 13135 malicious IPs with only 1 failed login attempt. Those are the hardest to catch, but they also have less impact on the attack.

Some ideas that come to mind are taking into consideration port numbers, since many malicious IP also change their port at every request, or trying out new combinations of parameters.

4.4 | Considerations and Alternatives

Our method proves to be pretty simple to implement and understand. There are only 3 conditions that flag an IP as malicious, and only 2 parameters (number of consecutive failed login attempts in a certain timeframe) to change. Therefore it is a highly explainable heuristic-based model.

Alternatives for anomaly-based techniques could be statistical analysis or machine learning algorithms. For the latter, the events should be labeled, and at that point, it would be easier to use a signature-based approach. In our case, this could be the requests which include the tag `xm001` in the error message. Another disadvantage of the other anomaly-based methods is that they are less explainable, hence we believe that our approach is the most suitable for this use case.

Finally, the signature technique is not very robust in this attack scenario. If the attacker would have changed the code `xm001` to a random new string at each request, it would bypass the detection.

5 | Reverse Classroom

Conforming to the structure presented, our laboratory is mainly structured in 3 sections: the dataset description, the practical activity itself, and the showcasing of our solution.

5.1 | Dataset description

Within these first 10 minutes, we will first present the meaning of IMAP and the structure of the data.

Secondly, we will present the password spraying attack model and point the students towards the successful breach (`c.brolly@adz.pi`) by discussing the different users that successfully logged in as well as the types of unsuccessful login attempts.

Following this, we will inform everyone about the user **`striddle`** and their role in the data. Finally, we will give the students a small live explanation of how to effectively use Kibana to explore the anomalies within this dataset.

Lastly, we will inform them that we also prepared a Python skeleton notebook for them to play with and try to develop an anomaly detection technique.

5.2 | Practical activity

The main section of the lab will start with us letting everyone explore the dataset in Kibana. Shortly after, we intend to offer the first hint to set them in the right initial direction. Namely, we will direct their focus toward the login attempts and hint them towards filtering for `xm001` in the context field to find the attack logs that they need to focus on.

Following this, we will present them with a progress check-in, after 10 minutes of letting them explore, in the hopes that they have discovered in Kibana what we previously presented in the descriptive section, namely the 4 types of context errors for failed login command, the attack pattern and users that have successfully logged in and the lead anomaly (`c.brolly@adz.pi`).

Following this, we will present them with the option to formalize their findings within the notebook we provided. The notebook mainly consists of 3 parts: importing the data and understanding working with it, forming an anomaly detection, and finally checking their results against the data we provided. This data is generated by filtering the main dataset for the `xm001` signature. We expect this part to last approximately 20 minutes, in which they, using the notebook or perhaps both the notebook and Kibana will hopefully, in a creative manner, formalize the attack pattern and pay closer attention to the types of context error messages. During this time we will continuously check everyone for progress and make sure that everyone at least has an idea of the attack pattern, as well as help the rest with the formalization process.

Ten minutes into their attempt we will provide them with the last hints. Namely, trying to only filter for consecutive logs that show **`logondenied`** as they are correlated with the bad password attempts; as well as filtering the failed logins based on time, and the number of IPs trying to access a user's access by the same IP. As well as checking in that they should slowly move towards measuring their results in the last section of the notebook.

5.3 | Solution showcasing

During these last 5 minutes, we will showcase our solution by showing them the completed python script, which was the basis for the script they worked on. The confusion matrix containing the evaluation of our detection will also be briefly explained. As we present our solution, we will make sure to mention that it might not be the best and that there might be other possible options, although our solution does have the desired results.

6 | References

[1] Atmail. Imap 101: Manual imap sessions, 2015.

[2] atmail. Imap commands - the complete list for atmail webmail. https://www.atmail.com/blog/imap-commands/, 2022.

[3] CrowdStrike. Password spraying: What it is and how to protect yourself. https://www.crowdstrike.com/cybersecurity-101/password-spraying/.

[4] M. Rose J. Myers. Internet message access protocol - version 4rev1. RFC 3501, March 2003.

[5] Microsoft. Incident response playbook for password spray attacks. https://learn.microsoft.com/en-us/security/operations/incident-response-playbook-password-spray, 2021.

[6] Nylas. Nylas imap, therefore i am. https://www.nylas.com/blog/nylas-imap-therefore-i-am/, 2018.