Leveraging Streaming-Based Outlier Detection and Sliceline to Stop Heavily Distributed Bot Attacks

Antoine Vastel
Head of Research
antoine.vastel@datadome.co

Konstantina Kontoudi
Lead Data Scientist
konstantina.kontoudi@datadome.co

In this paper, we discuss how to leverage streaming-based outlier detection and Sliceline to quickly and safely generate large volumes of rules/signatures that can be used to block malicious bot traffic.

While machine learning (ML) use has become more and more widespread, rules are still relevant in a security context. Indeed, companies have invested a lot in efficient rule engines capable of quickly evaluating a significant volume of rules. Moreover, rules are often more convenient to create, manipulate, and interpret, making them still valuable in addition to ML approaches.

We showcase that while Sliceline was designed initially to identify subsets of data where ML models perform badly, its use can be adapted to generate a large number of rules linked to an attack in an unsupervised way, i.e. without using labeled data. We also present our optimized Python open-source implementation of Sliceline and show how it can be used in a particular, but difficult, subset of bot detection: distributed credential stuffing attacks, in which attackers leverage thousands of infected IP addresses to conduct their attack and bypass traditional security mechanisms such as IP-based rate limiting.

Through a real-world example, we first explain how streaming-based detection can be used to detect such attacks, and how we use data modeling to apply Sliceline on server-side signals (HTTP headers, TLS fingerprints, IP address, etc.) to identify and generate blocking signatures linked to a distributed attack. This approach enables us to block around 2.2 million malicious login attempts per month across dozens of customers.

Finally, we explain how this approach generalizes to other security use cases besides bot detection, and how it can be used in different rule engines.

Introduction

Bot detection refers to the process of identifying and distinguishing between human users and automated software programs, known as bots. The term "bot" can refer to a wide variety of programs, including search engine crawlers and malicious bots designed to carry out scalping or denial-of-service attacks.

Bot detection is important because bots can be used for a variety of purposes, some of which may be harmful or unethical. For example, bots can be used to carry out fraudulent activities such as account takeover or credential stuffing, to scrape websites for sensitive information, or to automate the posting of spam messages on social media platforms.

Several techniques and signals can be used for bot detection, including analyzing user behavior patterns with ML [2], browser fingerprinting [3][4], and other forms of fingerprinting such as TLS fingerprinting [5]. Some common indicators of bot activity include high-frequency requests, inconsistent patterns of user behavior, and the use of non-standard user agents or other identifying information.

Effective bot detection is an important part of maintaining the security and integrity of online systems and platforms, and is used by a wide variety of organizations, including e-commerce sites, social media platforms, and financial institutions.

Despite the fact that ML techniques are becoming increasingly used for bot detection, most companies have efficient rule engines available that operate hand in hand with ML algorithms. The advantage of rules is they can be instantaneously deployed to mitigate attacks and are highly interpretable, making them a great addition to ML based bot detection.

In this paper, we focus on bot detection using signals that can be collected at the application layer, more specifically, HTTP headers and client-side signals. In particular, we will explain how we can analyze large amounts of traffic data using streaming algorithms, identify anomalous events or outliers, and analyze the underlying traffic signatures to infer rules. Those techniques can be transposed to other bot detection signals, and to other cybersecurity domains where one can perform a behavioral analysis, to identify a portion of data that is suspect, and then use this information to extract rules.

Using Sliceline for Rule Generation

Sliceline is an algorithm proposed by Sagadeeva et al.[1] that can be used for ML model debugging. When training a ML model, we usually compute the average performance of the algorithm on some holdout dataset. Sliceline allows us to find which regions in the feature space are responsible for very high error rates. In other words, it finds slices, i.e. subspaces of the feature space where the model performs worse than average. Those slices are in fact expressed as a conjunction of conditions on the features which can be easily converted to rules.

The algorithm takes as input the dataset and the error for every sample in the dataset. It requires the features in the dataset to be categorical. This is adapted to the bot detection problem, as data collected are mostly categorial, e.g. user-agent, country, or information about the type of device (GPU model etc). Note that it does not mean that it cannot be used with continuous features. In case of continuous features, we need to apply pre-processing to define distinct intervals and assign every value to an interval. This process, called binarization, turns continuous features into categorical features.

We use a simple example to illustrate the intended use case of Sliceline: to debug ML models. In our example, we consider a dataset with only 3 features f_1, f_2, and f_3. Furthermore, we suppose that we trained a classification model and computed the log loss associated with every sample in the holdout dataset.

f_1	f_2	f_3	error
a	d	h	0.1
a	e	f	0.2
b	d	f	0.9
a	d	f	0.8
c	e	h	0.2
c	d	h	0.1

Figure 1: example dataset with 3 categorical features and associated per-sample error.

Examples of slices that Sliceline can find with this input are:

- f_1 = b with an average error of 0.9
- f_2= d ∧ f_3= f with an average error of 0.85

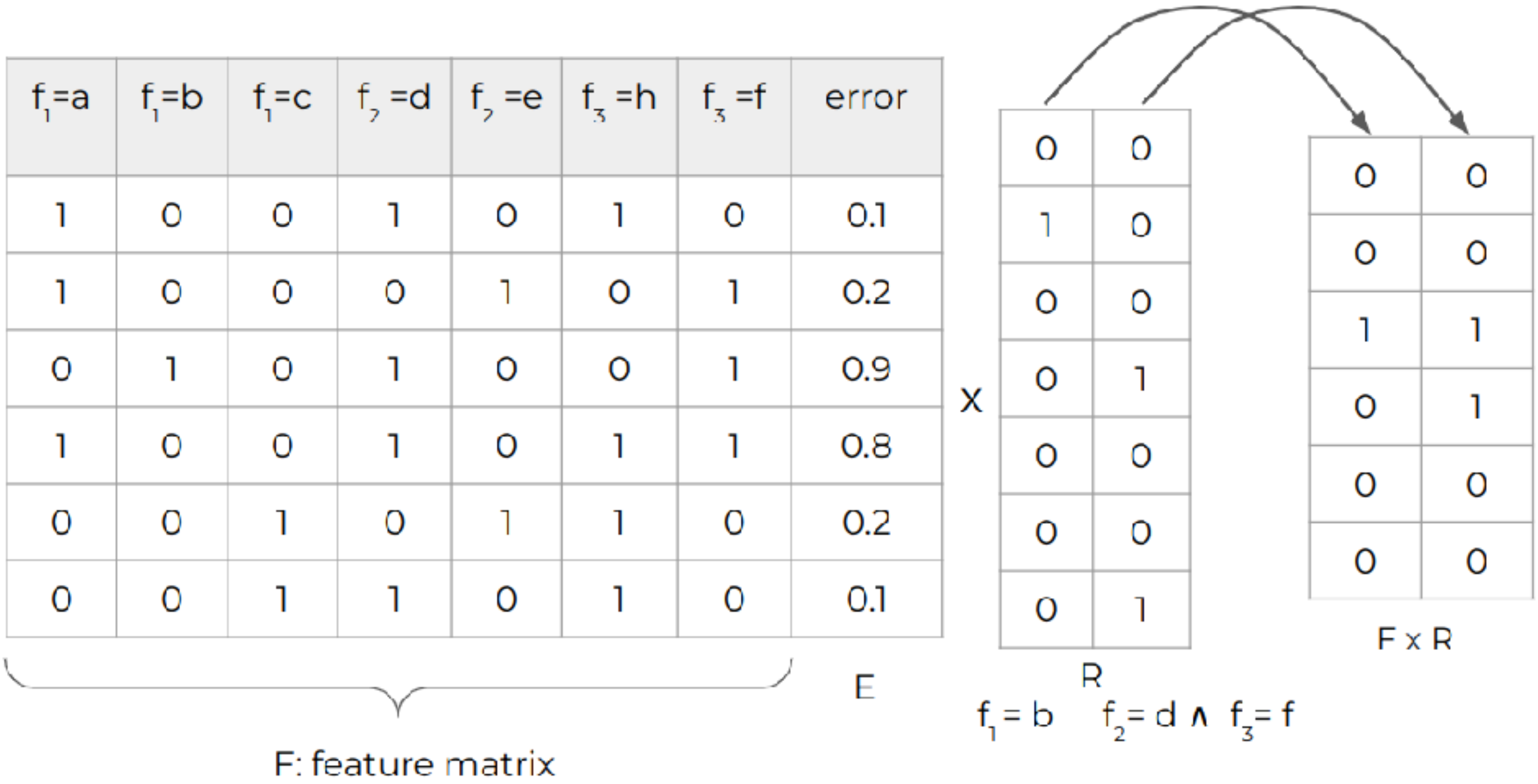
Both of those slices have a higher error than the average 0.38 of the dataset.

In principle, the slices we are looking for can be a combination of a lot of predicates, and the space of possible slices grows exponentially with the number and the cardinality of features. Sliceline manages to explore this space by using two techniques:

- Enumerating and evaluating slices using sparse linear algebra that can profit from existing efficient implementations.
- Pruning of slice candidates without accessing the data based on a score that balances between the slice size and the average slice error.

Those techniques make the algorithm efficient and scalable, with the exact complexity depending highly on the details of the dataset.

While, in this paper, we don't go through the implementation details of the Sliceline algorithm, we want to give readers some intuition on the usage of linear algebra to evaluate and generate rules. First, we start with our toy dataset and perform a first transformation: one-hot encoding. One-hot encoding is a method that allows encoding categorical data as numbers by creating one column per feature value. In the example below, the feature f_1 takes on 3 values a, b, and c, so to encode it, we create 3 binary columns as shown below.



f_1=a	f_1=b	f_1=c	f_2 =d	f_2 =e	f_3 =h	f_3 =f	error
1	0	0	1	0	1	0	0.1
1	0	0	0	1	0	1	0.2
0	1	0	1	0	0	1	0.9
1	0	0	1	0	1	1	0.8
0	0	1	0	1	1	0	0.2
0	0	1	1	0	1	0	0.1

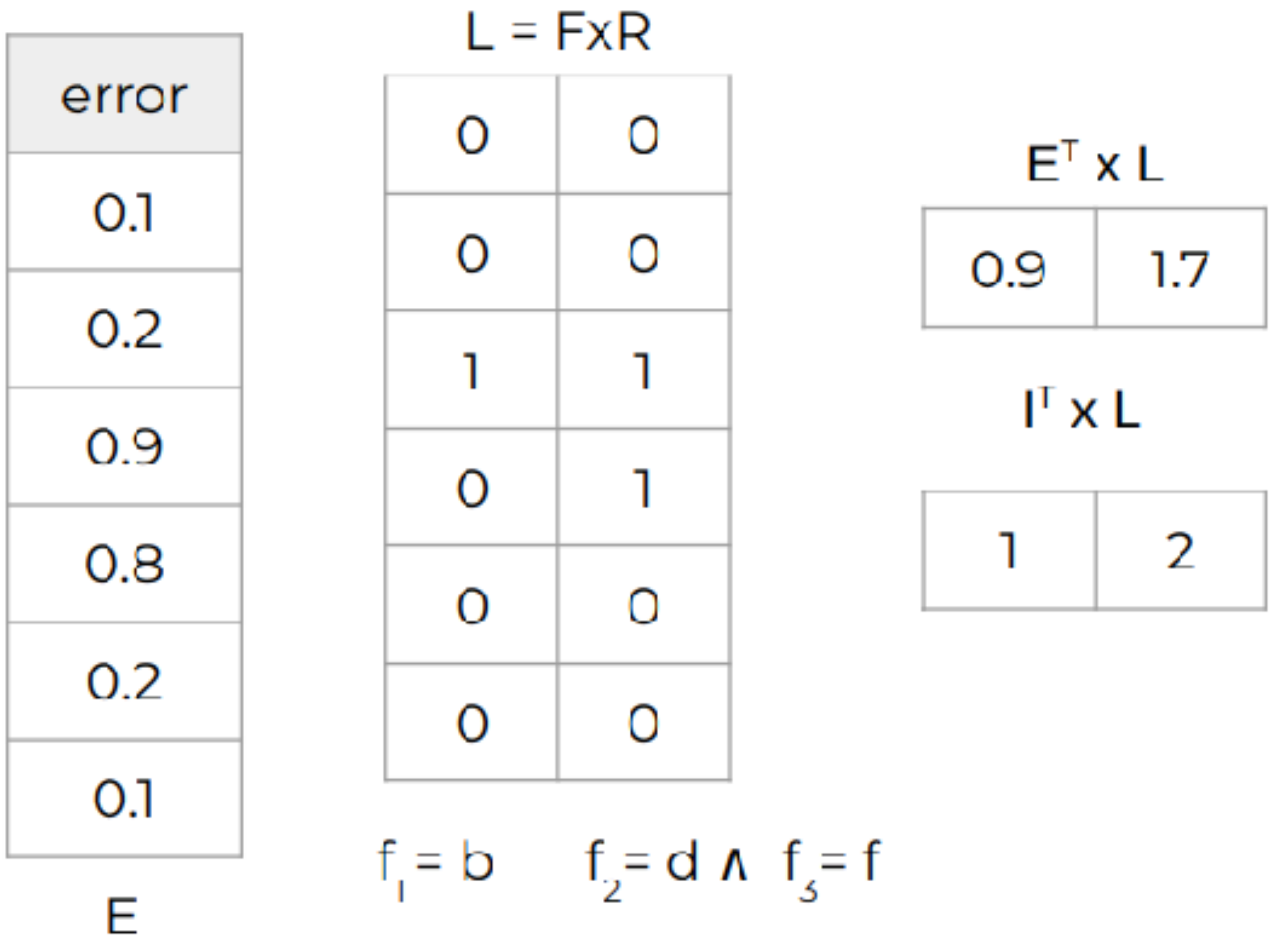
0	0
1	0
0	0
0	1
0	0
0	0
0	1

0	0
0	0
1	1
0	1
0	0
0	0

Figure 2: Example that shows how to use matrix multiplication to find matching samples.

The same transformation allows us to express rules as binary vectors as well: the vector will contain the value 1 at the position of the conditions it enforces. For instance, for the rule f_1 = b, only the second element of the binary vector will be one as this condition is represented by the second column in the one-hot encoded space. We can represent in this way all the candidate rules and easily compute a mask indicating which sample is matching every rule. If we denote the one-hot encoded feature matrix F and the matrix of rules R, then the result of the matrix multiplication FxR gives us exactly this: a mask where the rows correspond to samples and the columns correspond to rules. An element i,j of this matrix is equal to 1, which means that sample i matches with the rule j.

Having this mask, denoted L at hand, we can easily compute the average error for each slice, i.e. for the ensemble of samples matching a certain rule. This computation is central in the Sliceline algorithm as it allows computing the score, based on which the space of possible slices is explored. Thus, to compute the average slice error, we start by computing the matrix product of the error vector with the mask L. This gives us the total error in the slice. We can compute the product of L with the unit vector as well, effectively counting the number of samples that matched each rule. Finally, we can divide element-wise the total error and the size of the slice to obtain the average error.
This is the main idea behind the usage of linear algebra operations to evaluate rules.



error
0.1
0.2
0.9
0.8
0.2
0.1

E

L = FxR

f_1= b	f_2= d ∧ f_3= f
0	0
0	0
1	1
0	1
0	0
0	0

E^T x L

0.9	1.7

I^T x L

1	2

Figure 3: Example that shows how to compute mean slice error using dot product.

Even though Sliceline was proposed as a model debugging technique, we can repurpose it and use it to generate rules that target a specific group of data. In the cybersecurity context, we place ourselves in a situation where a defender gathered data from two groups:

- Group 0: A group considered normal, constituted mostly of non-malicious samples.
- Group 1: Another group of data that contains both malicious and normal traffic samples.

Then, we can assign group 0 to normal data and group 1 to suspicious data. Then, if we treat the group as the error that we give as input to the Sliceline algorithm, it will find slices targeting the suspicious group, as it is the one with the higher error. We show below the same toy dataset with the error replaced by the group.

f_1	f_2	f_3	group
a	d	h	0
a	e	f	0
b	d	f	1
a	d	f	1
c	e	h	0
c	d	h	0

Figure 4: Example dataset with 3 categorical features and associated group.

This simple technique allows us to generate rules to target bot traffic, provided that we can define two distinct groups of data.

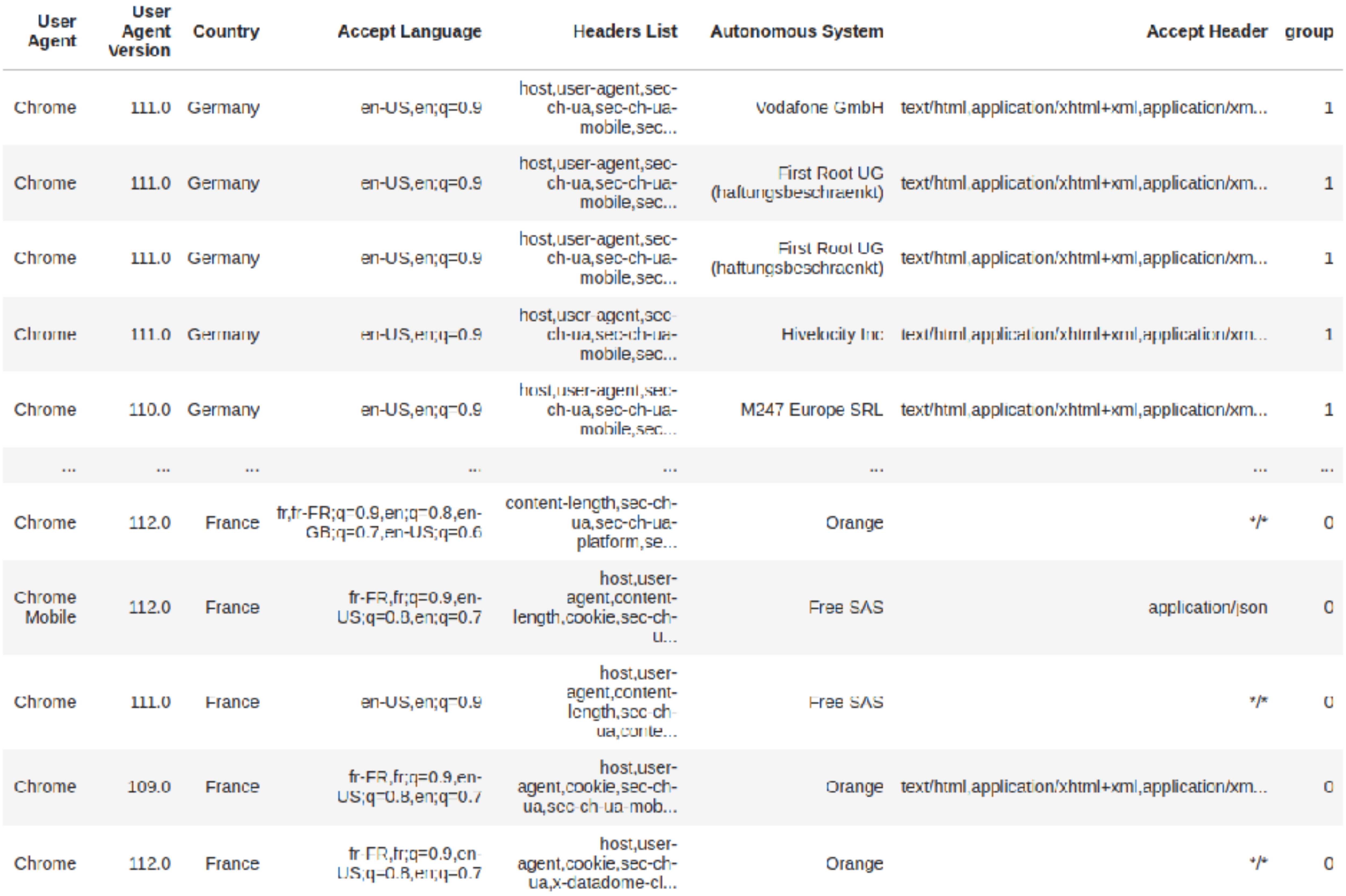
Optimized Python Open-Source Implementation of Sliceline

The authors of the original paper [1] provided an implementation of the algorithm in R and Apache SystemDS. We decided to implement it in Python and open-source it as a package for the community. We started by re-implementing the logic using Python and then implemented some performance improvements, which lead to a 1000-time speed-up compared to the R implementation.

This was possible because we used available Python libraries, such as Numpy, which leverage underlying C++ code but also sparse matrices. On top of that, we replaced some for-loop constructs with matrix multiplications which lead to further speed improvements. The source code of the package is available on GitHub [6], it is compatible with the pandas python library and follows the scikit-learn API conventions.

Applying Sliceline to a Toy Bot Detection Dataset

In this section, we apply Sliceline to the bot detection problem on a simple toy dataset constituted of HTTP headers and contextual information. For simplicity, we have created a dataset of traffic data, having both human and bot characteristics. We used data from a French e-commerce website. We collected data with old sessions coming from French-speaking countries that we consider as human and assigned them group 0. We also collected data from the same website coming from non-french speaking countries, with new sessions that use datacenter IP addresses, or IPs that were recently flagged as proxies, and assigned group 1 to the suspicious traffic.



User Agent	User Agent Version	Country	Accept Language	Headers List	Autonomous System	Accept Header	group
Chrome	111.0	Germany	en-US,en;q=0.9	host,user-agent,sec-ch-ua,sec-ch-ua-mobile,sec...	Vodafone GmbH	text/html,application/xhtml+xml,application/xm...	1
Chrome	111.0	Germany	en-US,en;q=0.9	host,user-agent,sec-ch-ua,sec-ch-ua-mobile,sec...	First Root UG (haftungsbeschraenkt)	text/html,application/xhtml+xml,application/xm...	1
Chrome	111.0	Germany	en-US,en;q=0.9	host,user-agent,sec-ch-ua,sec-ch-ua-mobile,sec...	First Root UG (haftungsbeschraenkt)	text/html,application/xhtml+xml,application/xm...	1
Chrome	111.0	Germany	en-US,en;q=0.9	host,user-agent,sec-ch-ua,sec-ch-ua-mobile,sec...	Hivelocity Inc	text/html,application/xhtml+xml,application/xm...	1
Chrome	110.0	Germany	en-US,en;q=0.9	host,user-agent,sec-ch-ua,sec-ch-ua-mobile,sec...	M247 Europe SRL	text/html,application/xhtml+xml,application/xm...	1
...	...	...	...	...	...	...	...
Chrome	112.0	France	fr,fr-FR;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6	content-length,sec-ch-ua,sec-ch-ua-platform,se...	Orange	*/*	0
Chrome Mobile	112.0	France	fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7	host,user-agent,content-length,cookie,sec-ch-u...	Free SAS	application/json	0
Chrome	111.0	France	en-US,en;q=0.9	host,user-agent,content-length,sec-ch-ua,conte...	Free SAS	*/*	0
Chrome	109.0	France	fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7	host,user-agent,cookie,sec-ch-ua,sec-ch-ua-mob...	Orange	text/html,application/xhtml+xml,application/xm...	0
Chrome	112.0	France	fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7	host,user-agent,cookie,sec-ch-ua,x-datadome-cl...	Orange	*/*	0

Figure 5: Sample of a real-world e-commerce traffic dataset constituted of human (group 0) and potential bot (group 1) traffic.

The following Python code samples showcase how we can apply Sliceline to our dataset. We store the dataset using a pandas data frame `df`:

Python

```python
from sliceline.slicefinder import Slicefinder

sf = Slicefinder(
    alpha = 0.80,
    k = 4,
   max_l = df.shape[1],
    min_sup = 1,
    verbose = True
)

sf.fit(df.drop("group", axis=1), df["group"])
```

Once the algorithm is fitted on the data, we can use the slices found to produce rules in the syntax that best suit our requirements. We can, for instance, do something like:

Python

```python
for slice, stats in zip(sf.top_slices_, sf.top_slices_statistics_):
    rule = None
    for feat, value in zip(df.columns, slice):
        if value is not None:
            if rule is None:
                rule = f"`{feat}`=`{value}`"
            else:
                rule += f" && `{feat}`=`{value}`"
    print(f"{rule} | slice size: {stats['slice_size']}")
```

This simple piece of code gives the following rules in our case:

Unset

```
`Country`=`Germany` | slice size: 4149.0
`User Agent`=`Chrome` && `Country`=`Germany` | slice size: 4133.0
`Country`=`Germany` && `Accept Language`=`en-US,en;q=0.9` | slice size:
4097.0
`User Agent`=`Chrome` && `Country`=`Germany` && `Accept
Language`=`en-US,en;q=0.9` | slice size: 4097.0
```

It means that, for this particular website, attacks were coming from Germany, using Chrome and a specific accept-language header.

Application to Distributed Attacks

Sliceline can be applied to the distributed bot attack detection problem by defining two distinct groups of traffic:

- Traffic before the attack: this part of the traffic contains only human data (group 0).
- Traffic during the attack: this part of the traffic contains human and bot traffic (group 1).

Note that in a real-world context, group 0 may contain some residual bot traffic. How it impacts the quality of the rules generated by Sliceline depends on the proportion of bot traffic in group 0, and how similar it is to the bot traffic also present in group 1.

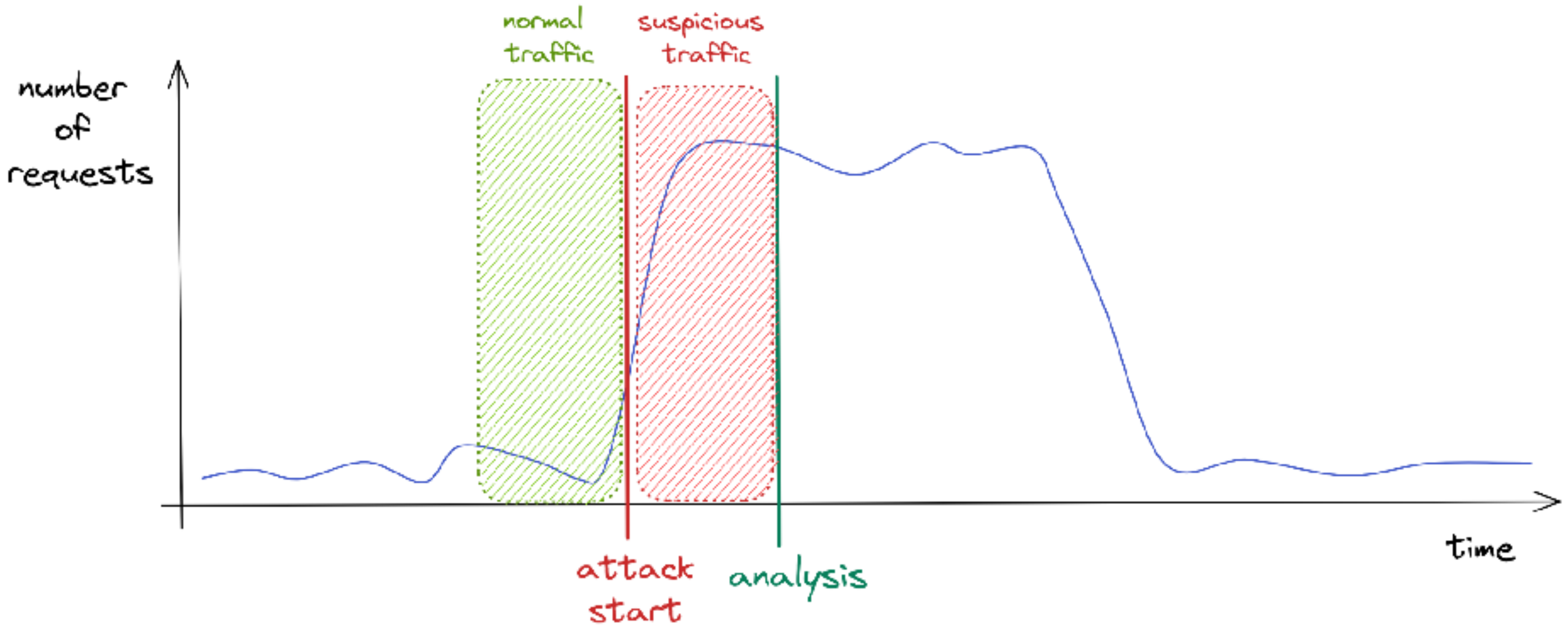


Figure 6: Graph of a time series illustrating the 2 groups used by Sliceline: human traffic before the attack (green, group 0) and mixed bot and human traffic (red, group 1).

The idea is presented in the graph above where we suppose that an anomaly detection algorithm provides the attack start time, and then a traffic analysis is triggered a bit later. If we identify the traffic before the attack as group 0 and traffic during the attack as group 1, then by applying Sliceline, we will find rules targeting the bot traffic.

The approach we propose to detect distributed attacks does not rely on traffic analysis by IP or session. Instead, we analyze the global traffic of each customer. To this end, we compute several aggregations and analyze their evolution over time. Examples of aggregations we compute are:

- the number of requests;
- the number of unique User Agents;
- the number of unique countries;
- the number of unique IP addresses;

All aggregations are computed over 10-minute windows in a streaming manner using Apache Flink [7]. Note that the size of the time window can be parametrized depending on the use cases:

- A small time window allows for faster detection but may increase false positives.
- A bigger time window may introduce lag in the outlier detection but decreases the risk of false positives.

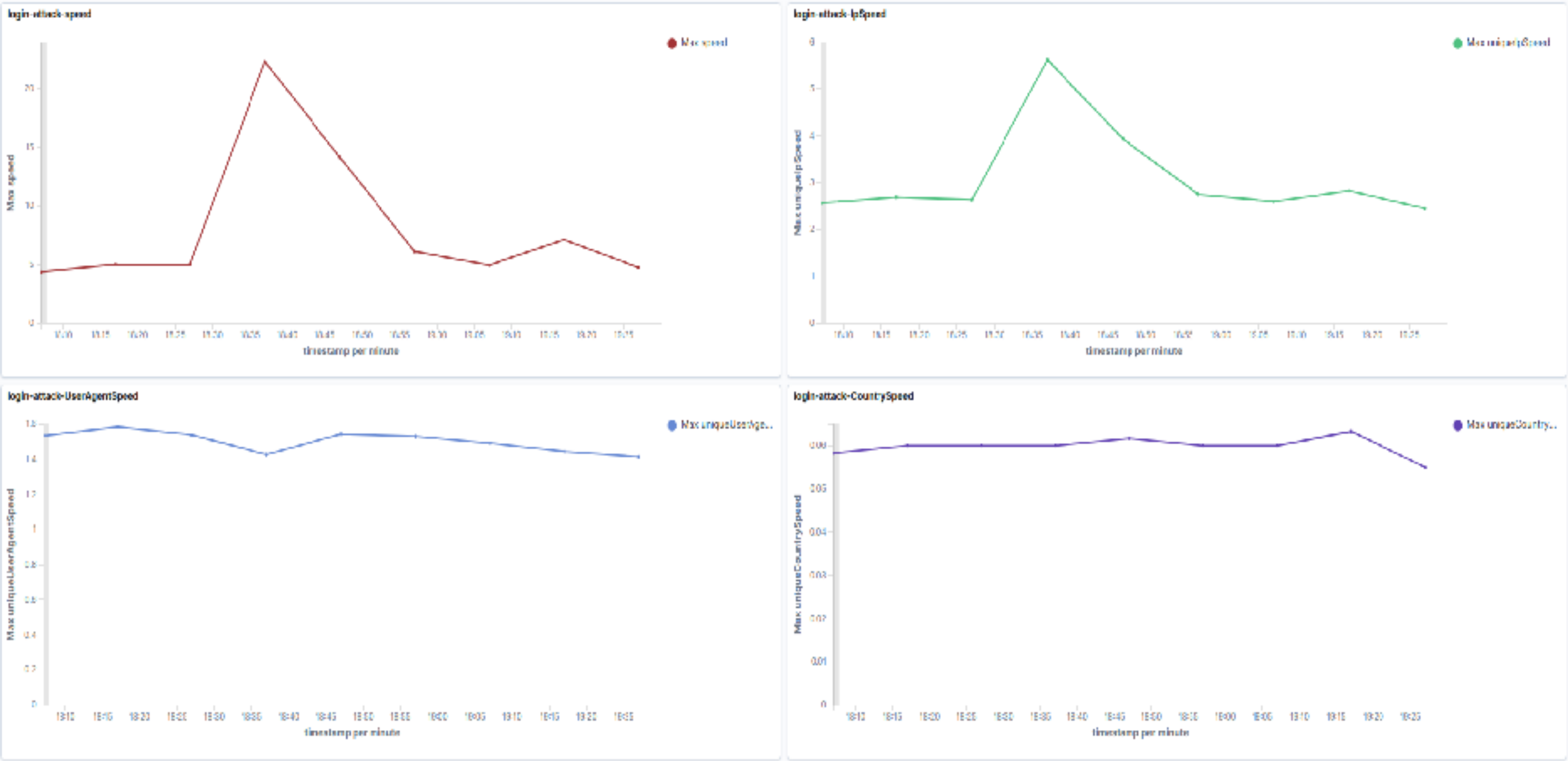


Figure 7: Example of four different aggregation times series (request count, number of IPs, number of user-agents, number of countries).

We analyze each time series independently and detect extremely high values using a z-score-based algorithm. A rolling window allows us to estimate the distribution of normal data and, based on this distribution, compute the z-score of new values that arrive. If the z-score exceeds a certain threshold, then we declare that an attack started and the traffic analysis is triggered.

This algorithm is lightweight and can be easily implemented in a streaming manner. The algorithm may suffer from 2 distinct issues:

- False positives: detecting an attack that is not due to bots either because the z-score threshold is too sensitive or because there was an increase in traffic due to some special event (breaking news, limited-edition sales, etc).
- False negatives: missing attacks that should have been detected.

To tackle the false negatives issue, one can gather example time series and label the attacks that need to be detected. This labeled dataset can further be used to better tune the thresholds. Regarding false positives, the rule-generation step that follows prevents us from generating rules that target humans. If the traffic spike is due to human traffic, then Sliceline will not find any difference between the characteristics of the traffic before and after the attack, and will therefore generate no rules.

Results: Real-World Credential Stuffing Attack Blocked by Sliceline

To exemplify our approach, we show a concrete example of an attack blocked using our streaming based outlier detection combined with Sliceline on a gaming platform. Figure 8 shows 4 different time series:

1. Blue: HTTP login traffic classified as human by detection engine (may contain bot traffic that has not been detected yet).
2. Orange: Traffic matching the rules generated by Sliceline before it gets deployed in the detection engine (undetected subset of the attack).
3. Green: Blue - Orange time series. Traffic considered as human after enforcement of the Sliceline generated rules.
4. Red: HTTP login traffic matching and blocked by the rules generated by Sliceline.

The login traffic of the gaming platform exhibited a small spike (blue line) that was captured by our anomaly detection algorithm. Sliceline was applied and generated a rule that subsequently blocked all the traffic denoted by the red line and totaled more than 3 million requests in a week.

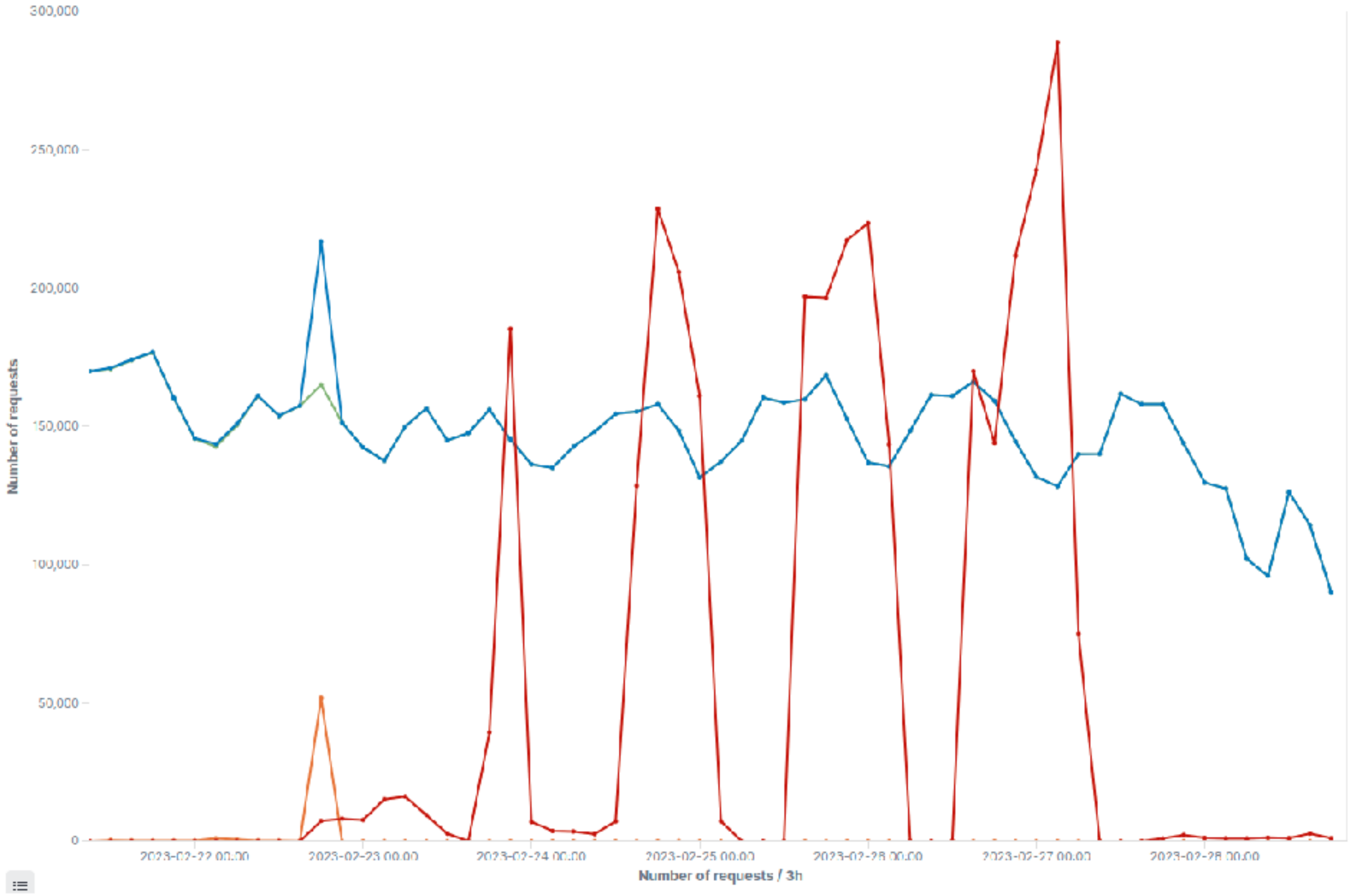


Figure 8: Time series related to the application of Sliceline on the login endpoint of a gaming platform.

The same blocked traffic is shown below but instead of the request number, we show the number of different IPs. More than 187k distinct IPs were used to distribute this attack.

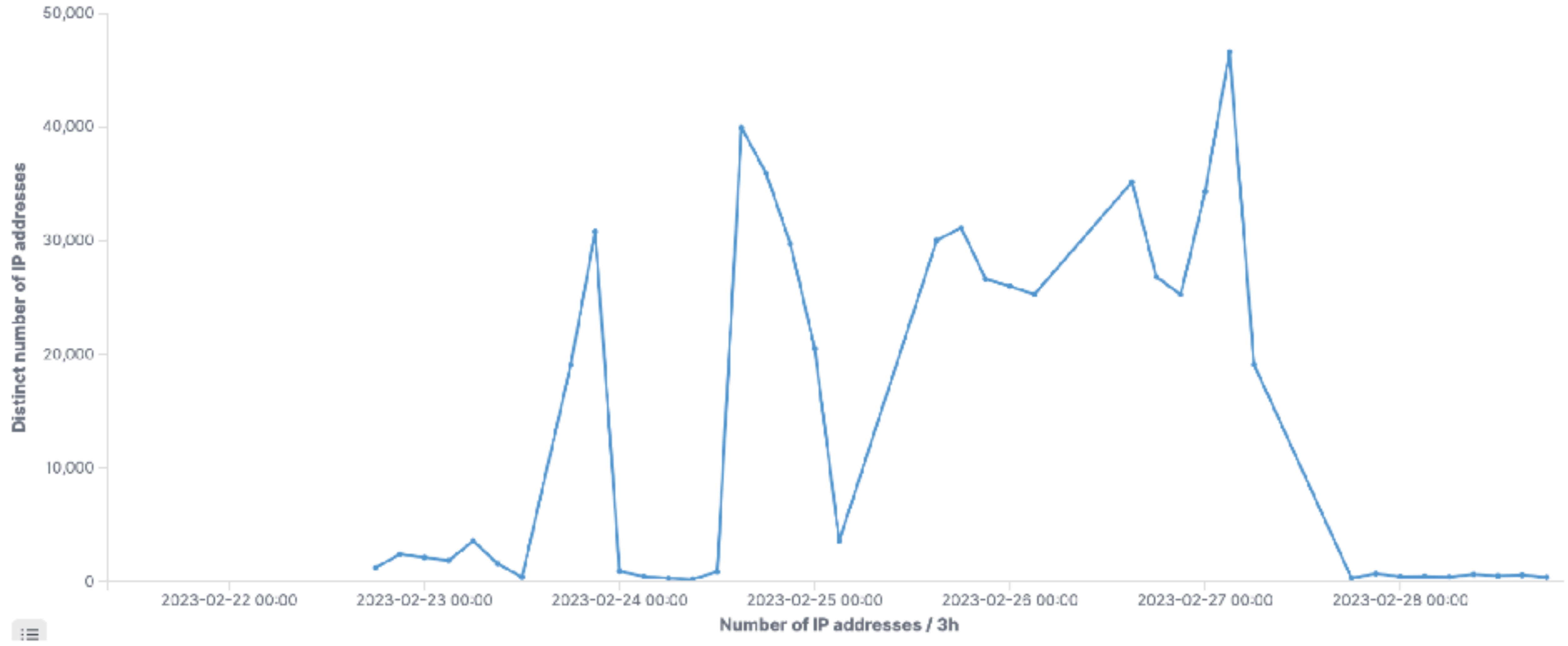


Figure 9: Distinct number of IP addresses/3h during the credential stuffing attack on a gaming platform.

While we leverage Sliceline in different ML pipelines to generate rules, our approach that combines streaming based outlier detection and Sliceline to protect login endpoints enables us to block around 2.2 million malicious login attempts per month across dozens of customers.

Conclusion

In order to bypass traditional bot detection techniques, such as IP based rate limiting and signature-based blocking, bots tend to distribute their attacks across thousands of IP addresses and frequently change their fingerprints. We proposed an approach that combines streaming-based anomaly detection with Sliceline to detect distributed bot attacks.

While Sliceline was originally proposed as a ML model debugging algorithm, we showcased how it can be used to generate rules that match malicious bot traffic activity. This approach enables us to block around 2.2M malicious login attempts per month across dozens of customers.

We implemented and open-sourced an optimized Python version of Sliceline. We rewrote part of the code to leverage matrix multiplication and sparse matrices, which lead to a 1000-time speed-up compared to the R implementation.

In this paper, we use the distributed bot attack detection problem to showcase our approach. However, our approach is entirely agnostic to the rules engine and can be applied to other cybersecurity settings provided that one can define the groups of normal and abnormal behavior.

References

[1] Sagadeeva, Svetlana, and Matthias Boehm. "Sliceline: Fast, linear-algebra-based slice finding for ml model debugging." Proceedings of the 2021 International Conference on Management of Data. 2021.

[2] Jacob, Gregoire, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. "PUBCRAWL: Protecting Users and Businesses from CRAWLers." In USENIX Security Symposium, pp. 507-522. 2012.

[3] Bursztein, Elie, et al. "Picasso: Lightweight device class fingerprinting for web clients." Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices. 2016.

[4] Vastel, Antoine, et al. "FP-Crawlers: studying the resilience of browser fingerprinting to block crawlers." MADWeb'20-NDSS Workshop on Measurements, Attacks, and Defenses for the Web. 2020.

[5] Li, Xigao, et al. "Good bot, bad bot: Characterizing automated browsing activity." 2021 IEEE symposium on security and privacy (sp). IEEE, 2021.

[6] https://github.com/DataDome/sliceline

[7] https://flink.apache.org/