black hat ASIA 2023

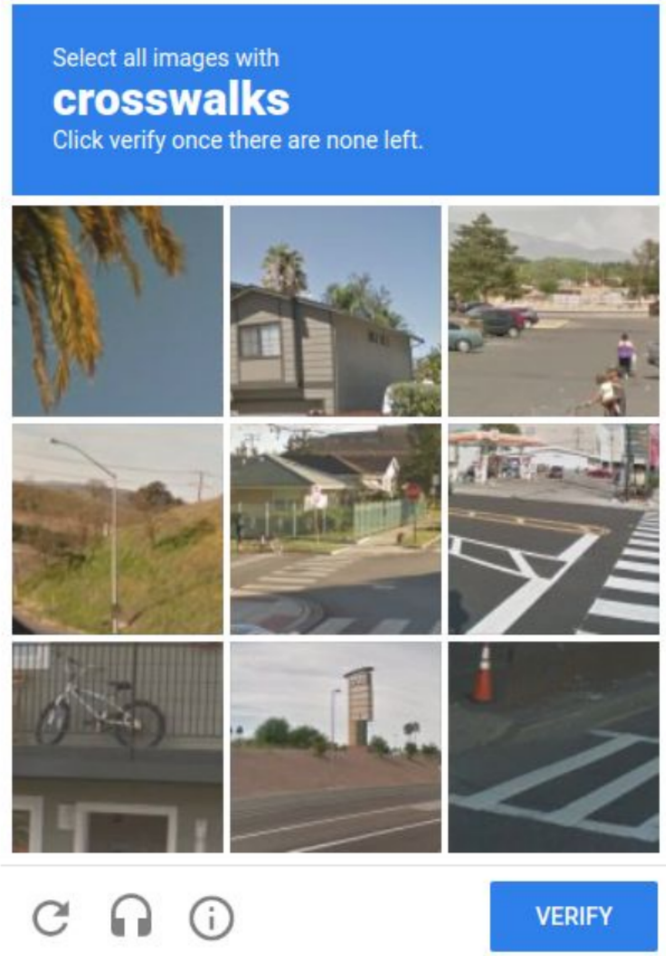
MAY 11-12

BRIEFINGS

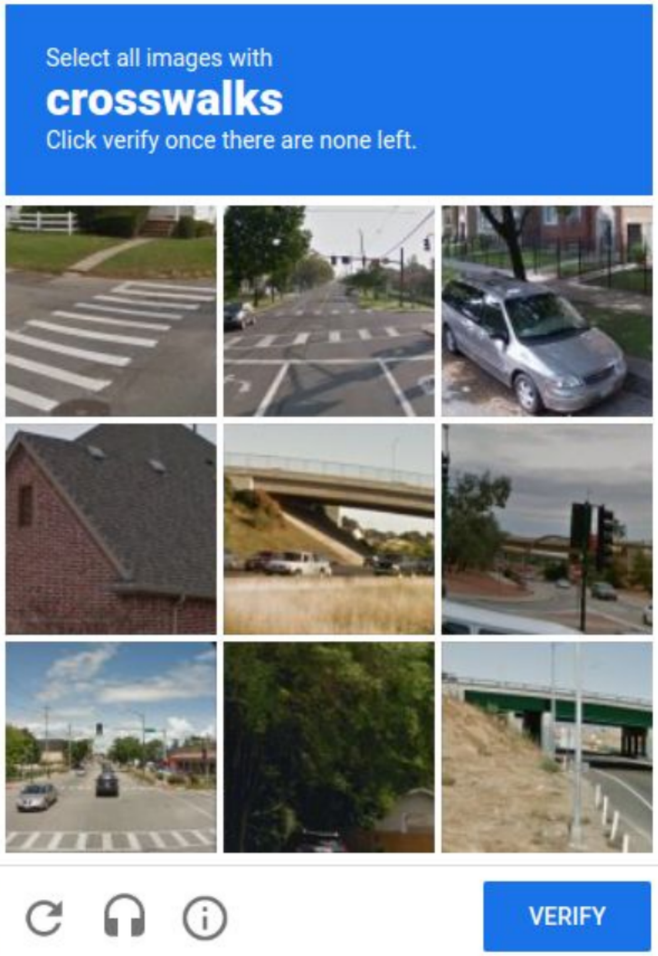
DATADOME

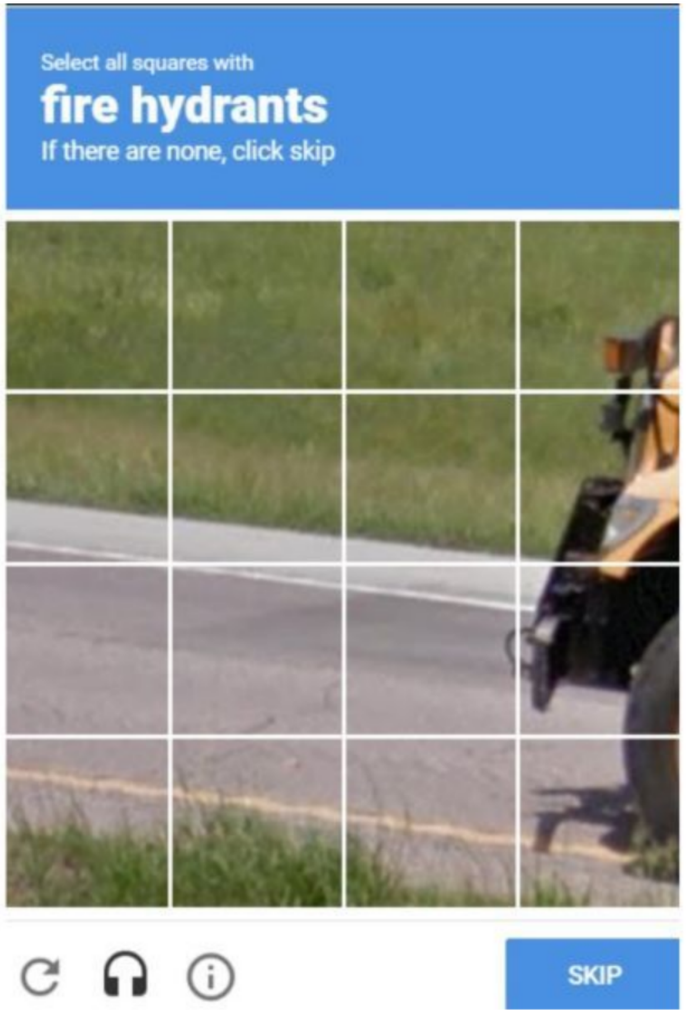
Leveraging Streaming-Based Outlier Detection and SliceLine to Stop Heavily Distributed Bot Attacks

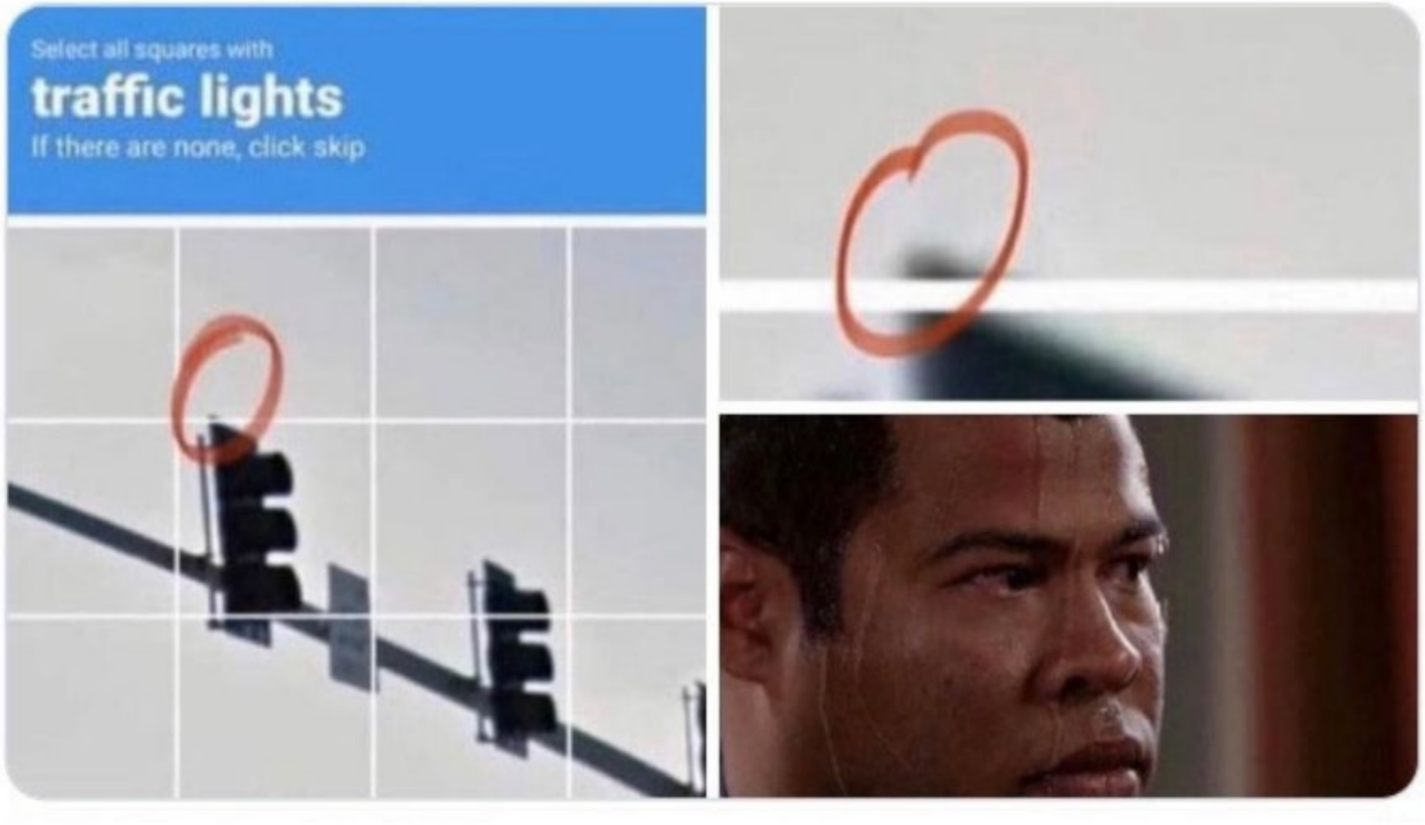
Antoine Vastel, PhD | Head of Research, DataDome
Konstantina Kontoudi, PhD | Lead Data Scientist, DataDome


Select all images with
crosswalks
Click verify once there are none left.
VERIFY

Select all squares with
bicycles
If there are none, click skip
SKIP


Select all images with
crosswalks
Click verify once there are none left.
VERIFY


Select all squares with
fire hydrants
If there are none, click skip
SKIP


Select all squares with
traffic lights
If there are none, click skip

All these CAPTCHAs because of bad bots.

But can we detect bad bots without CAPTCHAs?

Introduction to (Bad) Bots

What's a bot?

Program to **automate actions**.

Can be used for positive purposes:

- Automated **website testing**.

Nefarious purposes:

- Test batch of **stolen credentials**.
- Generate **fake video views/retweets**.

Attacks conducted by bots

Credential stuffing/account takeover → Steal user accounts.

DDoS → Make website/mobile app unavailable.

Carding → Test stolen credit cards.

Vote manipulation → Generate fake views, increase number of likes, retweets, etc.

Bot Technologies: HTTP Clients

node	python	GO
Axios	Aiohttp	net/http
Got	Requests	

Low CPU/RAM resources needed.

No JavaScript execution.

(Potentially) inconsistent:

- HTTP Headers
- TLS Fingerprints

Bot Technologies: Automated Browsers

Puppeteer +
(headless) Chrome

Selenium +
(headless) Firefox

Playwright +
(headless) Webkit

More CPU/RAM resources needed.

Execute JavaScript (JS) natively.

Consistent **HTTP headers** and **TLS fingerprints**.

Potentially inconsistent browser fingerprint (JS).

How to detect bots?

How to detect bots?

Signatures/(browser/TLS/ HTTP) fingerprints.

Behavioral analysis:

- Volume of requests.
- Browsing patterns.

Reputation: IP/session, proxy detection.

Context: country, time of the day, website targeted.

Detection Example: Selenium

Detection using browser fingerprinting (JS).

Selenium introduces attribute:

document.$cdc_asdjflasutopfhvcZLmcfl_

chromium / chrome / test / chromedriver / js / call_function.js

459 lines (426 loc) · 14 KB Blame Raw

```
/**
 * Returns the global object cache for the page.
 * @param {Document=} opt_doc The document whose cache to retrieve. Defaults to
 *     the current document.
 * @return {!Cache} The page's object cache.
 */
function getPageCache(opt_doc, opt_w3c) {
  var doc = opt_doc || document;
  var w3c = opt_w3c || false;
  // |key| is a long random string, unlikely to conflict with anything else.
  var key = '$cdc_asdjflasutopfhvcZLmcfl_';
  if (w3c) {
    if (!(key in doc))
      doc[key] = new CacheWithUUID();
    return doc[key];
  } else {
    if (!(key in doc))
      doc[key] = new Cache();
    return doc[key];
  }
}
```

Bypass Techniques Used by Bots

Forge TLS fingerprints.

CYCLETLS

Forge browser fingerprint and HTTP headers.

https://github.com/intoli/user-agents


```
{
  "appName": "Netscape",
  "connection": {
    "downlink": 10,
    "effectiveType": "4g",
    "rtt": 0
  },
  "platform": "Win32",
  "pluginsLength": 3,
  "vendor": "Google Inc.",
  "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (
  "viewportHeight": 660,
  "viewportWidth": 1260,
  "deviceCategory": "desktop",
  "screenHeight": 800,
  "screenWidth": 1280
}
```


Fake/simulate JavaScript execution:

- Forge JS proof of work payload (reverse engineer).

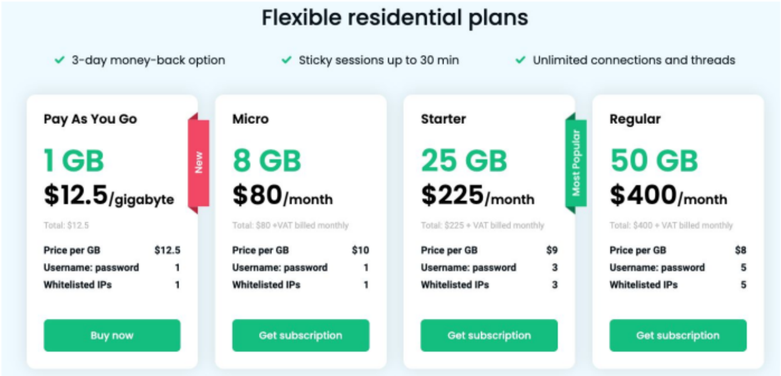
Bypass Techniques Used by Bots

Distribute attack using **proxies**:

→ Avoid IP-based rate limiting.

Distribute attack using **residential proxies**:

→ Avoid reputation-based blocking.

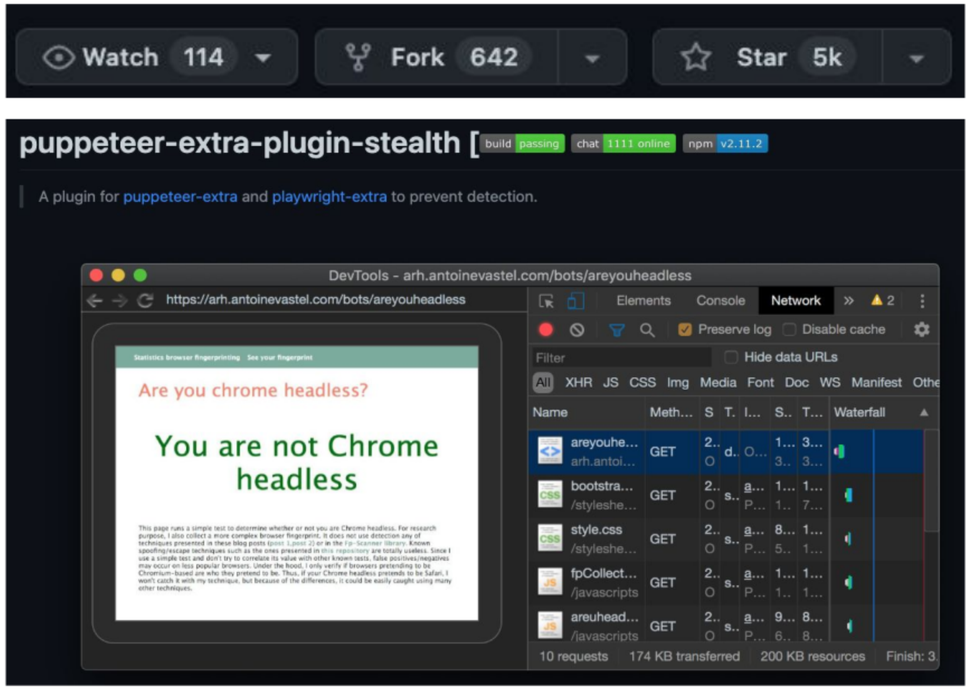


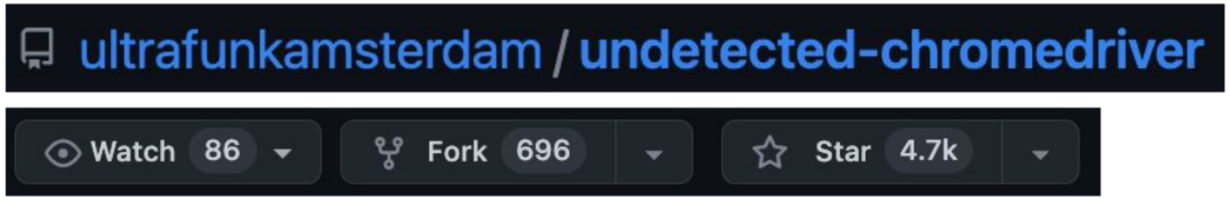
Distribute attack using **residential proxies located in same country as website targeted**:

→ Avoid geo-blocking.

Community Driven Anti-Detection Frameworks

Generic, for (headless) automated browsers:





To Conduct Credential Stuffing Attacks

README.md

OPEN BULLET 2

OpenBullet 2 is a cross platform automation suite powered by .NET core. It allows to perform requests towards a target webapp and offers a lot of tools to work with the results. This software can be used for **scraping** and **parsing data**, automated **pentesting** and much more.

Link to the Official Forum where you can find guides and become part of the community behind OpenBullet.

Found a bug? Create an issue!

IMPORTANT! Performing (D)DoS attacks or credential stuffing on sites you do not own (or you do not have permission to test) is **illegal!** The developer will not be held responsible for improper use of this software.

Bad Bots in 2023: Summary

Distribute their attacks.

Leverage thousands of residential proxies.

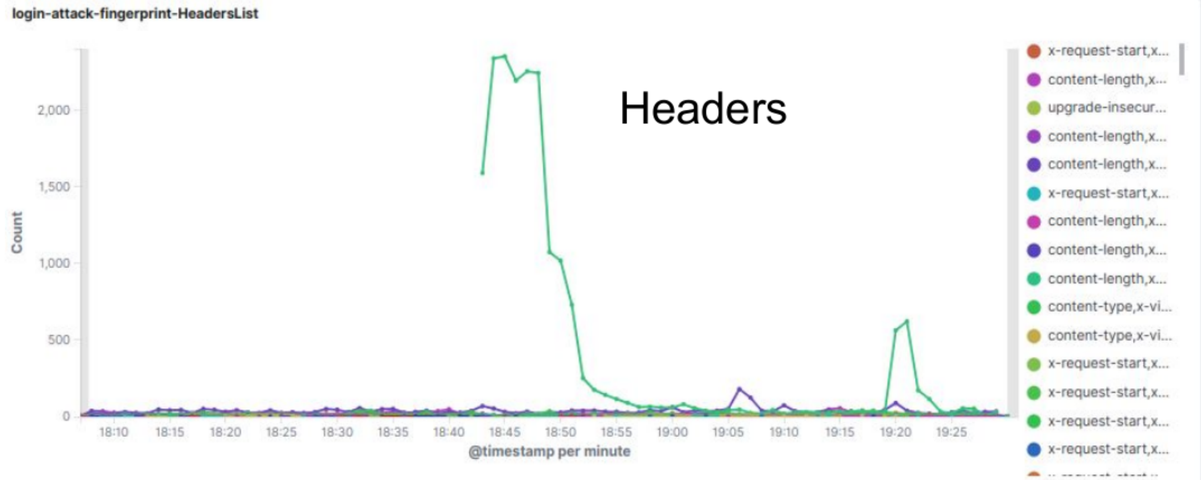
Constantly change and forge their signatures/fingerprints.

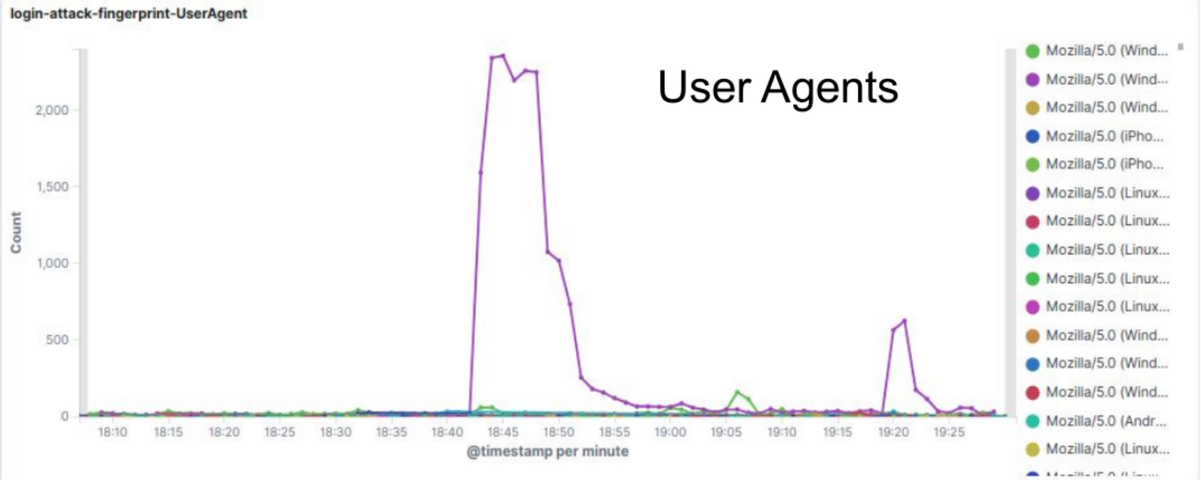
Q: How to block these ever-evolving and distributed bots?

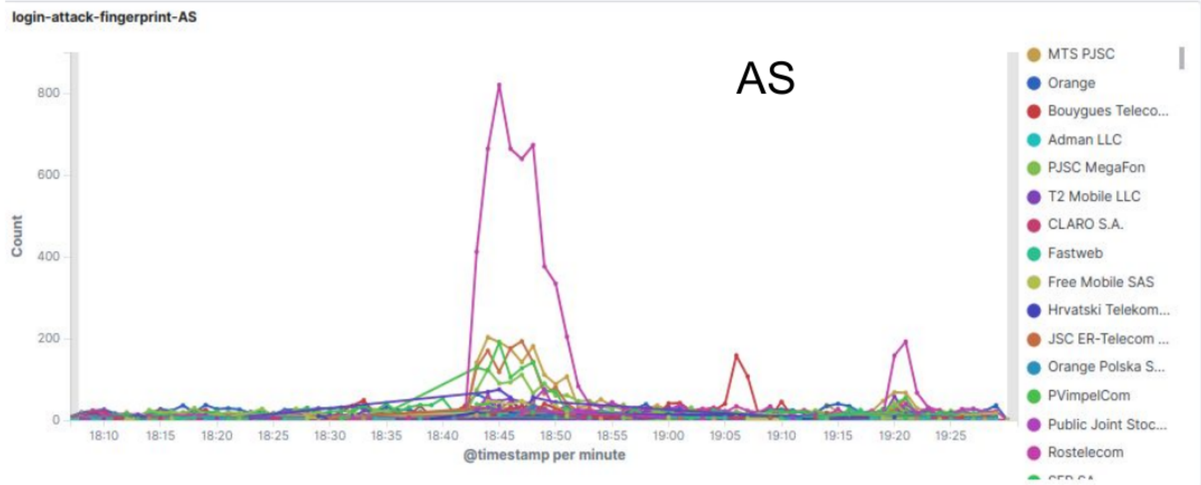
Detecting and Blocking Distributed Attacks Manually

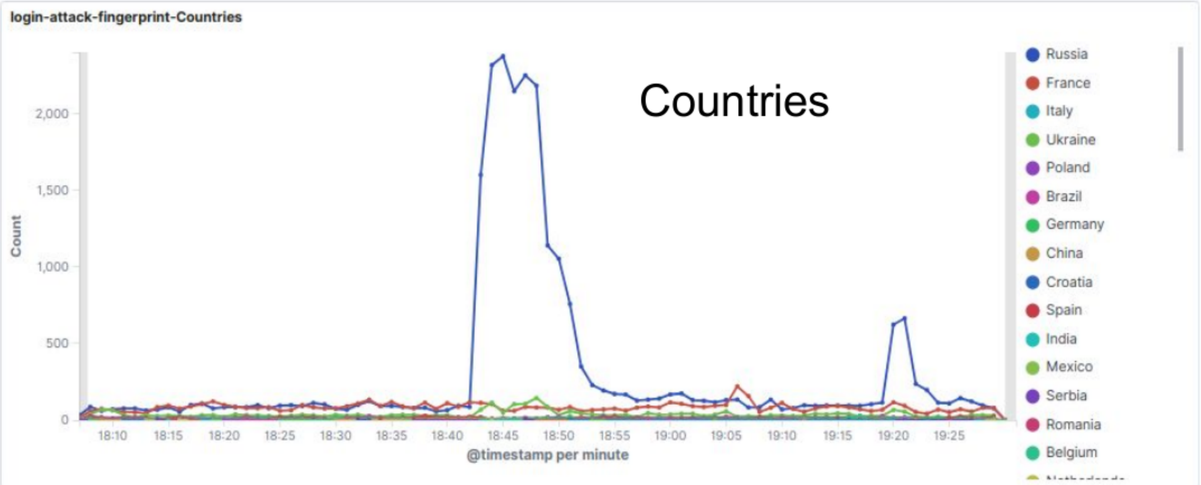
Detect a Traffic Spike

Drill-Down on Different Features









Derive Rules

Find a rule.

```
Country=Russia && User Agent=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.84 Safari/537.36
```

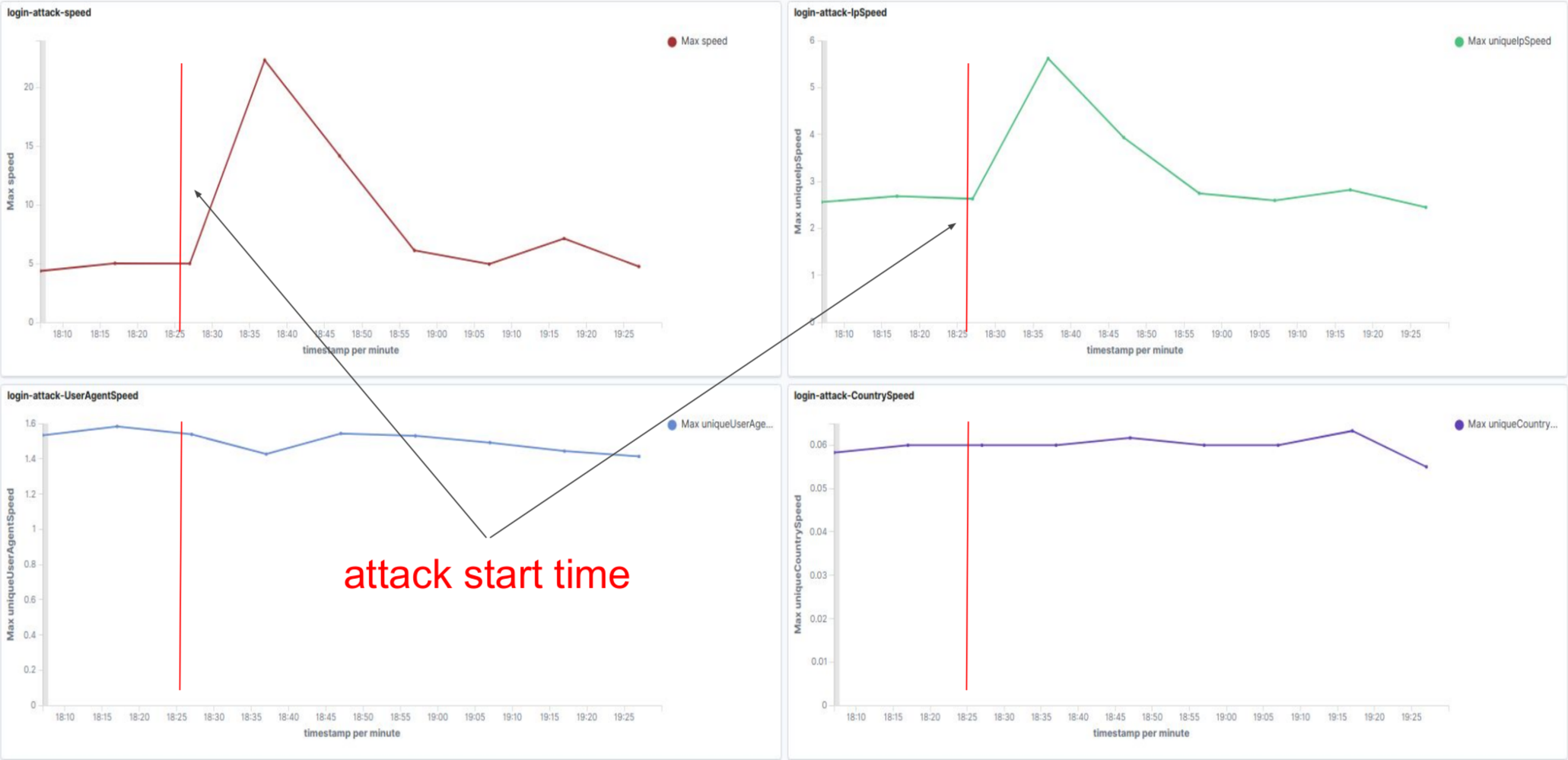
How to automate the analysis?

Detecting Distributed Attacks

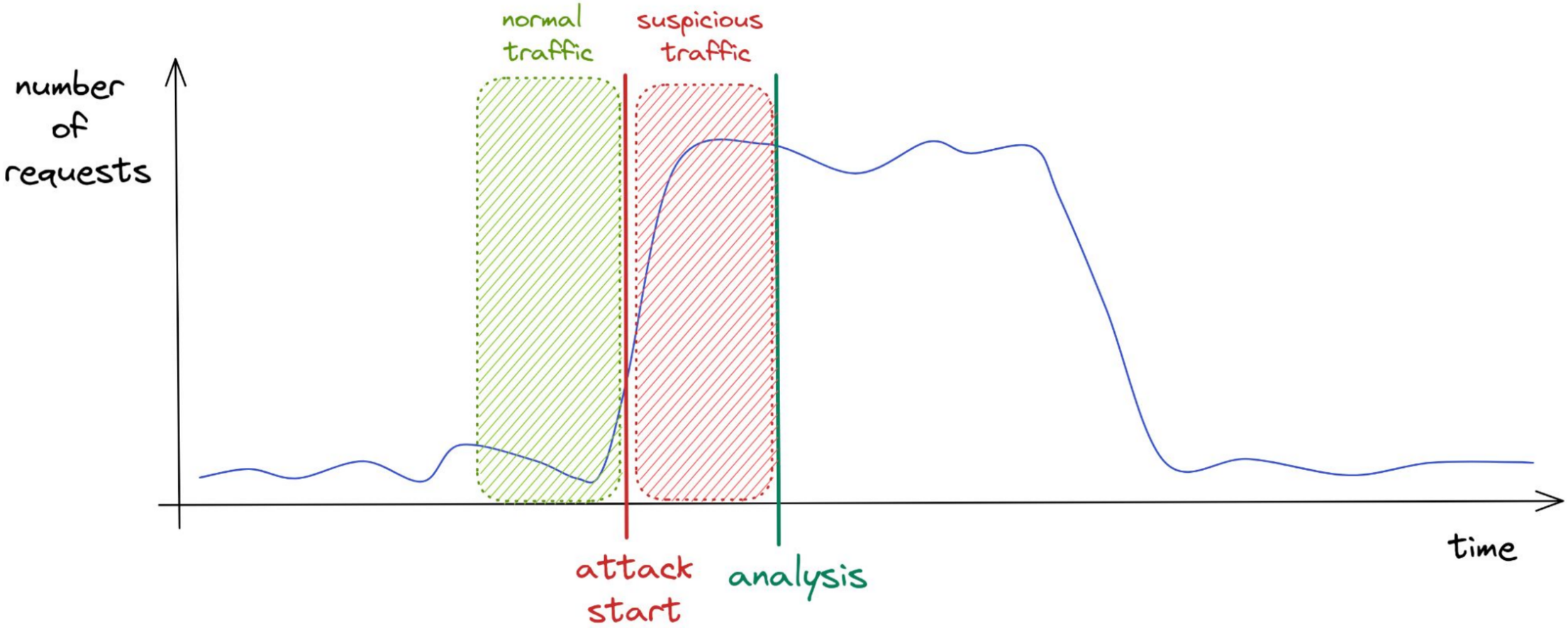
Detecting Distributed Attacks

- Compute **aggregate statistics on login:**
  - Number of unique User Agents.
  - Number of unique IPs.
  - Number of sessions.
- Detect **anomalies** on the resulting **time-series using a z-score based anomaly detection algorithm.**
- **Push an event** describing the attack (customer, start time).
- Implemented in streaming (Apache Flink).

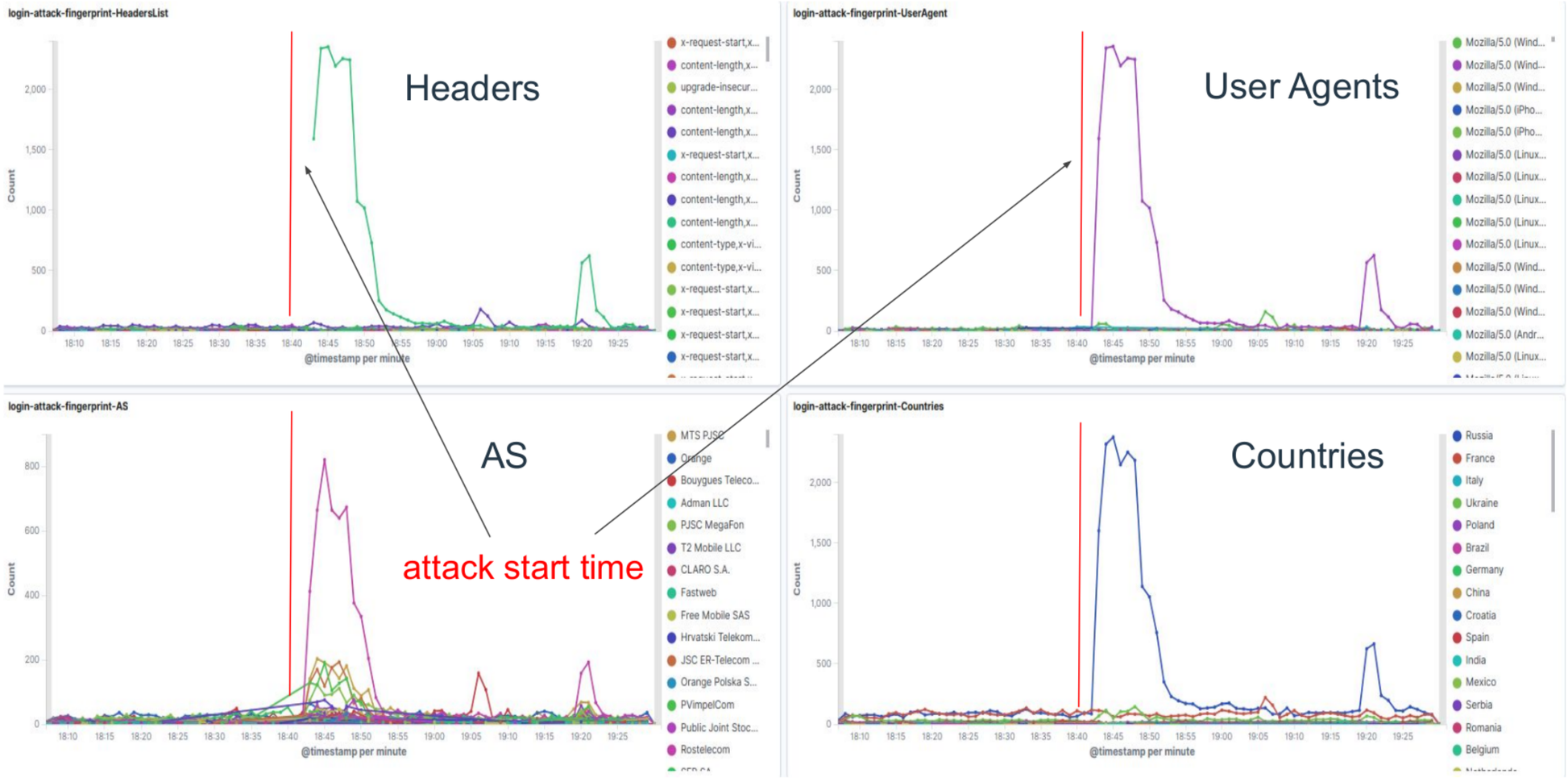
Detecting Distributed Attacks



General Idea



Next Step: Automate Rule Generation



Introducing Sliceline

What is Sliceline?

Algorithm to find **slices** of data where an ML model performs badly.

country	UA	IP type	error
FR	Chrome	residential	0.1
FR	Firefox	data_center	0.2
DE	Chrome	data_center	0.9
FR	Chrome	data_center	0.8
CN	Firefox	residential	0.2
CN	Chrome	residential	0.1

Slice: a conjunction of conditions.

country = DE

UA=Firefox && IP type=data_center

https://dl.acm.org/doi/10.1145/3448016.3457323

Generating Rules Using Sliceline

Use Sliceline to target groups of data instead of bad-performing slices.

country	UA	IP type	group
FR	Chrome	residential	0
FR	Firefox	data_center	0
DE	Chrome	data_center	1
FR	Chrome	data_center	1
CN	Firefox	residential	0
CN	Chrome	residential	0

Group definitions:

- **Human traffic** => group 0
- **Suspicious traffic** => group 1

How does it really work?

- Searches for slices with large errors.
  → **Need a way to quickly compute slice errors.**

- Uses matrix algebra to evaluate slices.
  → **Can profit from optimized implementations of matrix multiplication.**

- Uses pruning to reduce the search space without accessing the data.
  → **Reduces memory access for large datasets.**

Sliceline Internals: Encoding

country	UA	IP type	error
FR	Chrome	residential	0.1
FR	Firefox	data_center	0.2
DE	Chrome	data_center	0.9
FR	Chrome	data_center	0.8
CN	Firefox	residential	0.2
CN	Chrome	residential	0.1

1. One-hot encoding.

2. Express rules as binary vectors.

country = FR	country = DE	country = CN	UA = Chrome	UA = Firefox	IP type = residential	IP type = data_center	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	1	0	0	0	0	0

0	0	0	1	0	0	1

Sliceline Internals: Matching Rules With Matrix Multiplication

country = FR	country = DE	country = CN	UA = Chrome	UA = Firefox	IP type = residential	IP type = data_center	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

F: feature matrix

E

X

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

R

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

F x R

country = DE UA=Firefox && IP type=data_center

Sliceline Internals: Slice Errors

error
0.1
0.2
0.9
0.8
0.2
0.1

E

L = FxR

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

country = DE UA=Firefox &&
IP type=data_center

E^T x L

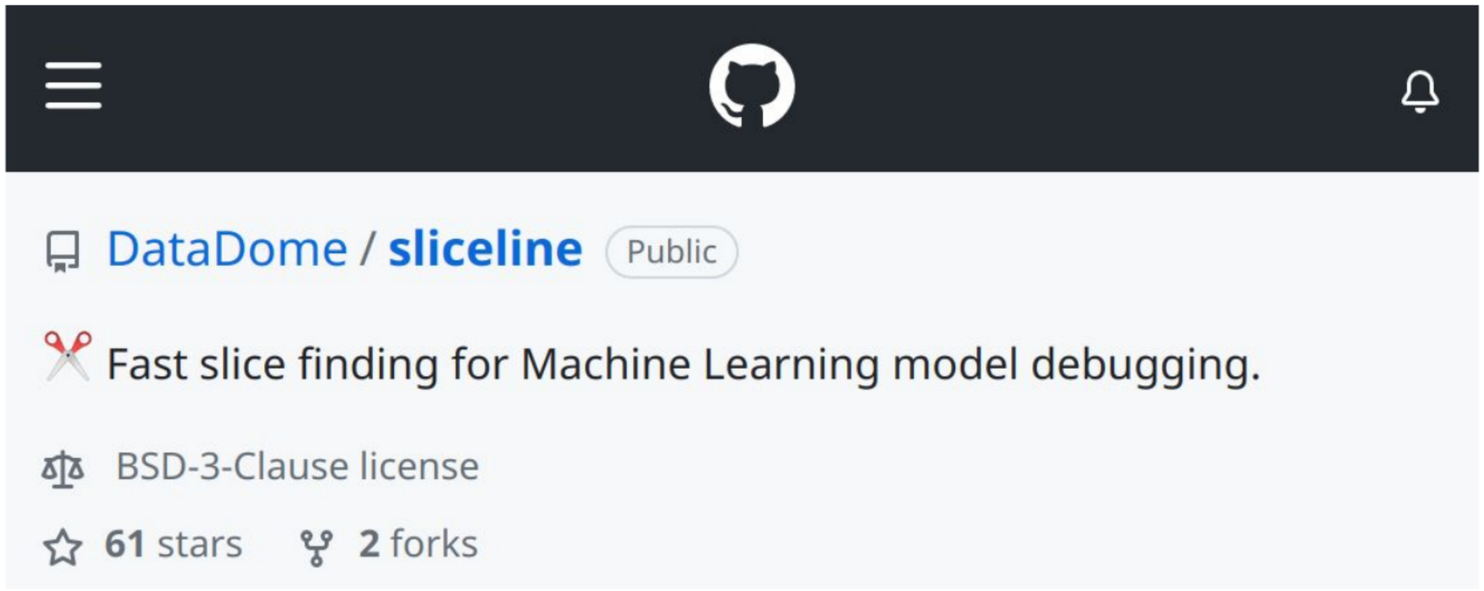
total slice error	0.9	1.7

I^T x L

slice size	1	2

mean slice error	0.9	0.8.5

Open Source Package



Original algorithm implemented in R.

Implemented in Python:

- Rewrote some part of R implementation using **matrix multiplications**.
- Leverage **numpy optimizations**.
- Compatible with pandas.

Speed up > x 1000

Syntax-agnostic: can generate **rules for any rule-engine**.

Code Example

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

Example Dataset:

- Gathered from a French e-commerce website.
- Human traffic (group 0): requests with old session.
- Suspicious traffic (group 1): requests from non french speaking countries and datacenter IPs.

Code Example

```
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"])
```

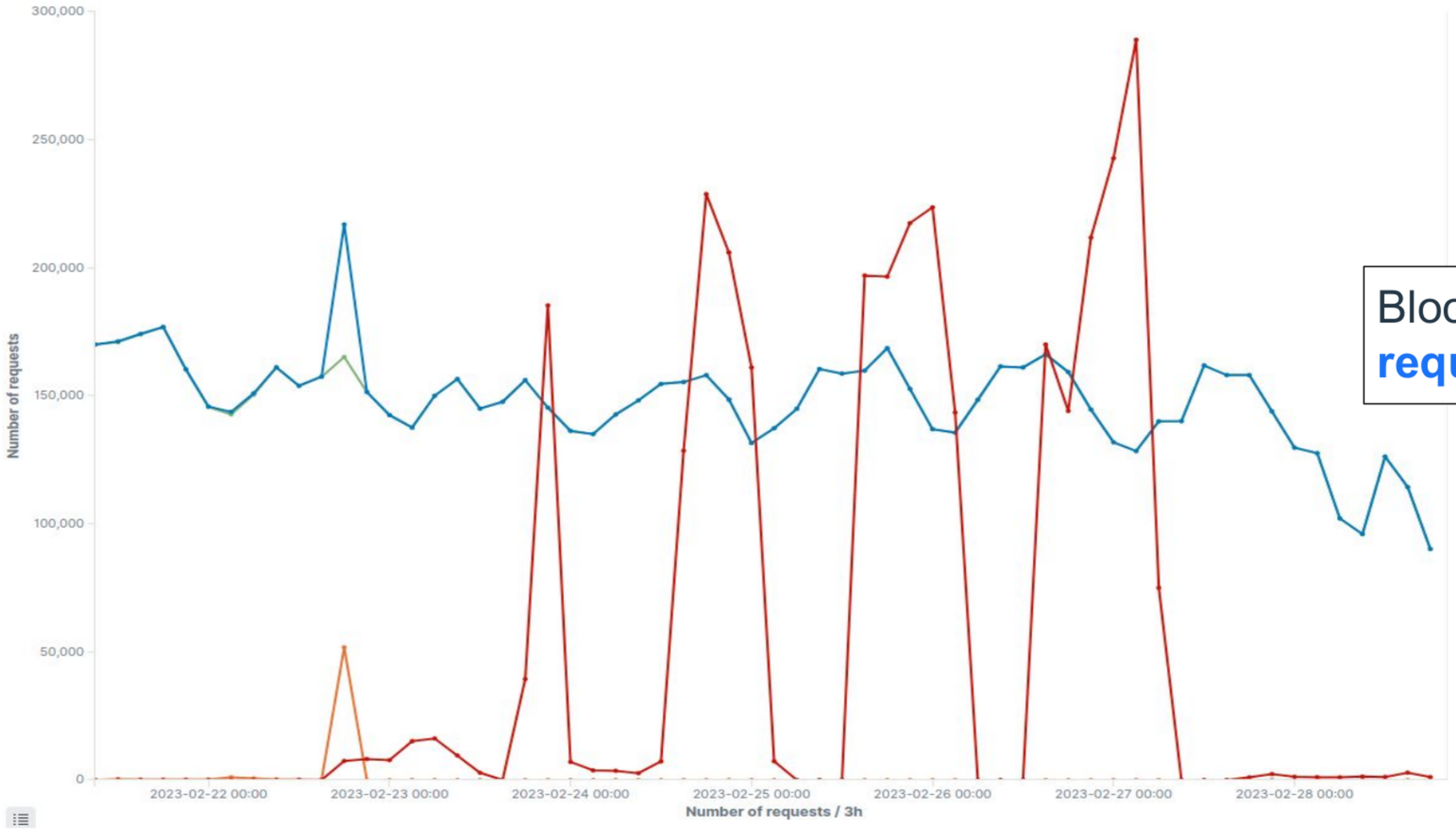
Apply the algorithm with 2 lines of code!

Get rules in the format you want!

```
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']}")
```

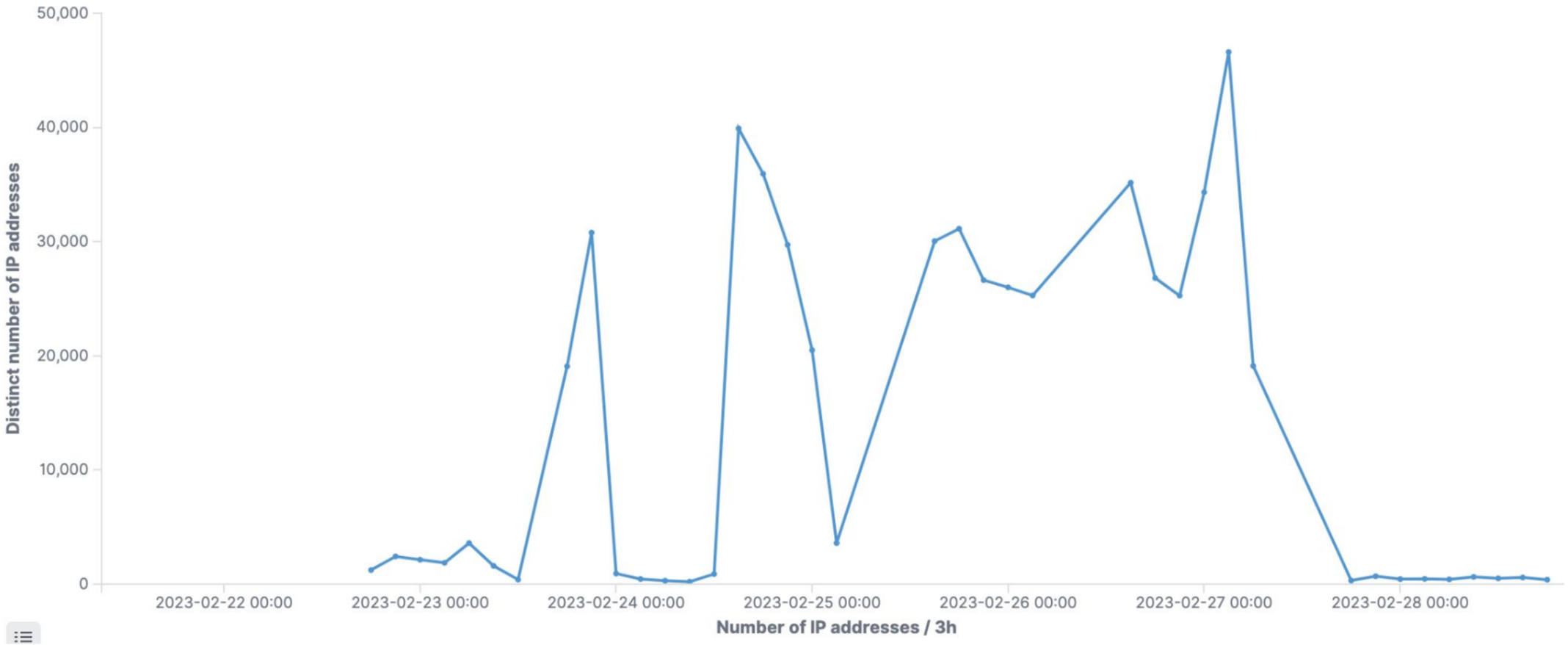
```
`Country`=`Germany` | slice size: 4149.0
`User Agent`=`Chrome` && `Country`=`Germany` | slice size: 4133.0
`Country`=`Germany` && `Accept Language`=`en-US,en;q=0.9` | slice size: 4097.0
`User Agent`=`Chrome` && `Country`=`Germany` && `Accept Language`=`en-US,en;q=0.9` | slice size: 4097.0
```

Credential Stuffing Attack on Gaming Platform



Blocked more than **3M requests** in a week.

Heavily Distributed Attack: > 187k IP Addresses



Takeaways

Approach to **detect distributed attacks** using traffic aggregations and anomaly detection.

We leverage Sliceline to **infer malicious signatures and to generate rules**.

Efficient against **bots that frequently adapt** and modify/forge their fingerprints.

Sliceline **can be applied to other security use cases** that rely on a rule engine.

Contact

DATADOME

Antoine Vastel PhD,
Head of Research @DataDome,
antoine.vastel@datadome.co.

Konstantina Kontoudi PhD,
Lead Data Scientist @DataDome,
konstantina.kontoudi@datadome.co.