Dynamic Bot Blocking with Web-Server Access-Log Analytics

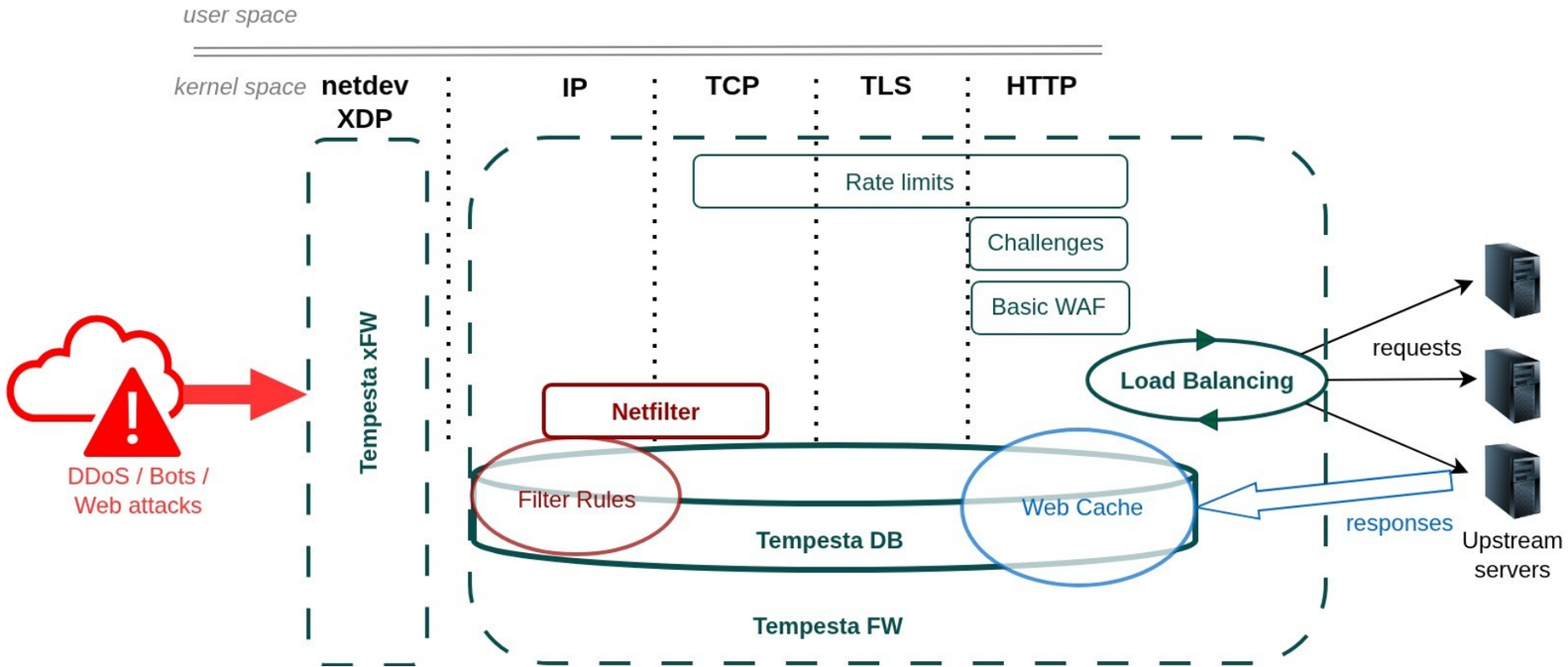
Alexander Krizhanovsky

Tempesta Technologies, Inc.

ak@tempesta-tech.com

Tempesta FW & xFW

- Linux kernel HTTP accelerator + XDP volumetric DDoS filter
  https://github.com/tempesta-tech/tempesta
  https://tempesta-tech.com/tempesta-escudo/knowledge-base/XFW/
- Embedded into the TCP/IP stack: outperforms HAProxy, Nginx etc.
- Full DDoS protection & Web security

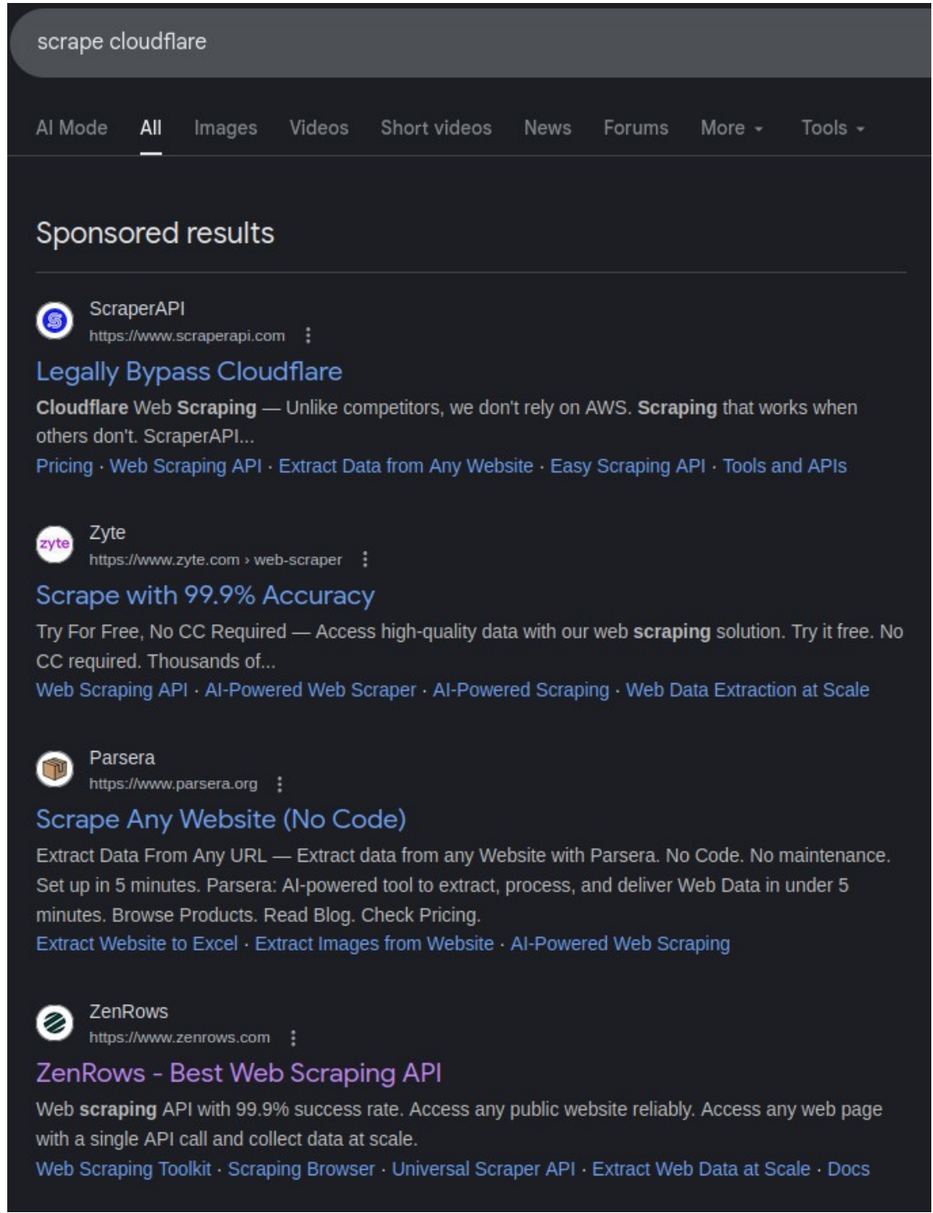


Bots

- L7 DDoS
  - HTTP flood, HTTP/2 control frame floods = high rates
  - slow HTTP = high RAM/CPU/socket usage
- Security scanners, password crackers = high rate of 4xx/5xx errors
- “Bots”
  - **content scrapers** (AI boosted)
  - carting/checkout abuse, e.g. inventory scalping
  - booking bots (time crucial, so high rates)
  - ….

Bot Protection

- Rate limits: floods, scanners etc
- Slow HTTP – hard to rate limit CPU usage
- Cloud proxies: ***focus on large vendors***
  - thousands IPs w/ only one request
    https://lwn.net/Articles/1008897/
  - CAPTCHA, JS & other challenges' solvers
  - full browser emulation (inc. resource reqs)
  - Impersonalization & consistency (L3-L7)
  - behavior: timing, mouse movements etc
- Custom bots + residential proxies *(pricy)*
  https://www.reddit.com/r/webscraping/



Data Analysis

- Identify clients
  - IP addresses
  - fingerprints (p0f, JA3, JA4, Tempesta Fingerprints)
    … or other *combined metrics*, e.g. unusual `<User-Agent, JA3>`
  - **not** `Referer, User-Agent` or other headers
- Classify
  - RPS, BPS (high or low), GeoIP, response time, HTTP errors
  - history (behavior)
- Events
  - **rapid** growth on classification, (semi-)manual

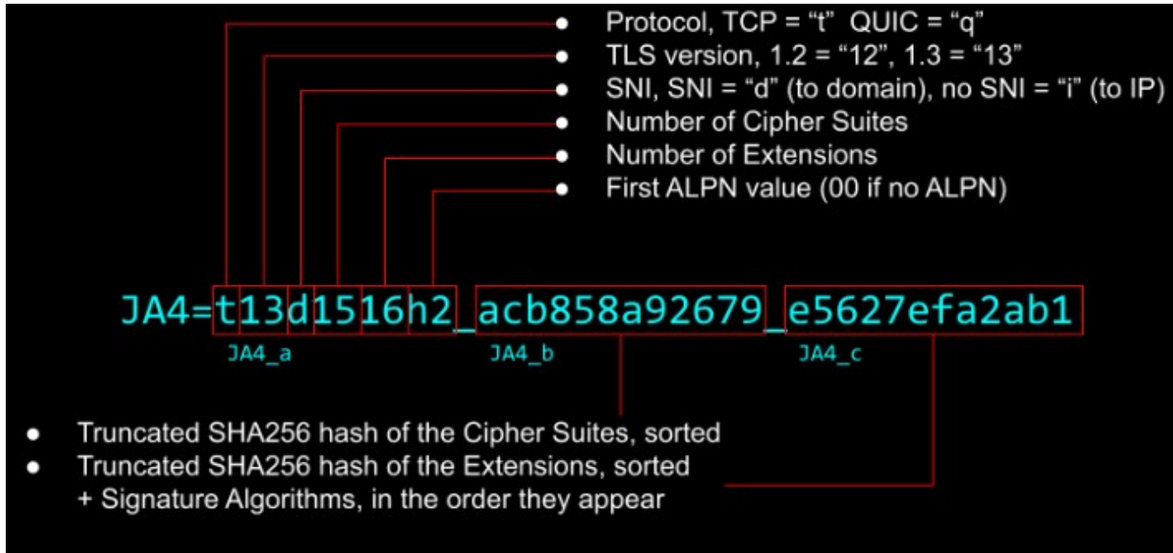
Tempesta Technologies

Traffic Fingerprints

- HTTP
  - Version (HTTP/1.1, HTTP/2, HTTP/3)
  - Method (GET, POST etc)
  - Which headers are presented and in which order
- TLS (`ClientHello`)
  - Version (TLS 1.2, TLS 1.3)
  - Announced ciphersuites
- Network
  - Versions (IPv4, IPv6)
  - TCP window size

John Althouse's Fingerprints: JA3 & JA4

- JA3 – TLS only
  https://github.com/salesforce/ja3
  - wide adoption (e.g. implemented in Envoy)
  - `MD5(TLS Version, Cipher, TLS Extension, Elliptic Curve, Elliptic CurvePoint Format)`
  - `"6734f37431670b3ab4292b8f60f29984"`
- JA4+ - TLS, HTTP, TCP, Latency
  https://blog.foxio.io/ja4%2B-network-fingerprinting
  - SHA256 is expensive in the kernel
  - No *similarity* for some parts
- ML classification: large **distance** between **similar** values



p0f Fingerprints

https://lcamtuf.coredump.cx/p0f3/

- Passive traffic fingerprinting
  - opposed to `nmap` sending specially crafted packets
  - also tries to guess software, e.g. Windows, Firefox etc
- IPv4/IPv6, TCP handshake, HTTP
  - MTU, TCP options, MSS & window size, timestamps, HTTP headers

Tempesta Fingerprints

https://tempesta-tech.com/knowledge-base/Traffic-Filtering-by-Fingerprints/

- TLS & HTTP
- Fast computation
- Similarity encoding (DA & ML friendly)

TLS (tft)

```
3 bits: ALPN: “h2”,  “http/1.1”, “http/1.1,h2”, “h2,http/1.1”
1 bit: set if handshake has unknown ALPN
1 bit: found virtual host for SNI
1 bit: abbreviated handshake
1 bit: TLS version
(alignment to 1 byte)
2 bytes: sum * 11 + cipher_suite (order of cipher suites)
2 bytes: sum * 11 + extension_type
2 bytes: sum * 11 + elliptic_curve
```

Tempesta Technologies

Access Logs

- Usually (except Varnish) **formatted string** writes to a **file**

```
[367562.584711] 192.168.100.1 "tempesta-tech.com" "GET / HTTP/1.1"
  200 25207 "-" "Wget/1.25.0" "tft=66cbda9cafc40009" "tfh=b1c0008c0280"
```

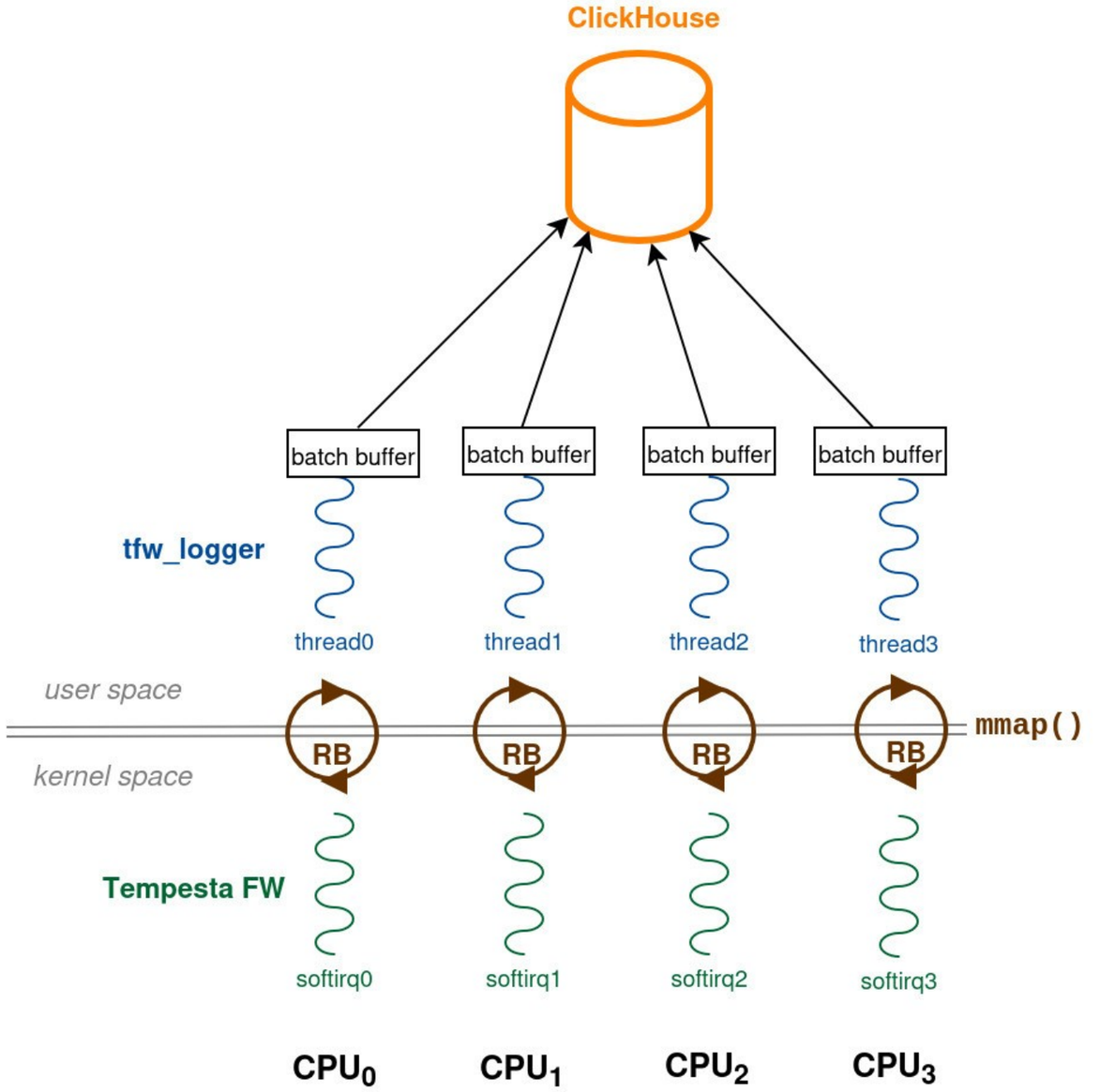
  - hard to store 100K+/s records
  - hard to analyze (`grep` & Co.)
- Log shipping: read, parse, send, (re-)encode, store
  - Filebeat + Logstash, Vector, Fluent Bit, Kafka etc
  - ClickHouse, MongoDB, TimescaleDB, InfluxDB etc
- ClickHouse
  - high ingestion performance
  - powerful and fast analytics

Tempesta Technologies

`tfw_logger`: Log Shipping

https://tempesta-tech.com/knowledge-base/Access-Log-Analytics/

- Per-CPU threads & ring buffers (like `io_uring`)
- Zero-copy
- Binary encoding
- Optimized batches for ClickHouse ingestion
- Similar ClickHouse tables for security events and xFW blocking events



Access Log in ClickHouse

- **34K records/s** in a 4 vCPU VM on i9-12900HK 8GB RAM (w/ ClickHouse)

```
SELECT DISTINCT timestamp,uri,user_agent,tfh,dropped_events
FROM access_log LIMIT 15;
```

timestamp	uri	user_agent	tfh	dropped_events
2025-11-05 16:10:30.250	/	baremetal	657681479538678246 4	**11**
2025-11-05 16:10:30.402	/	vm.web	657681479538678246 4	0
2025-11-05 16:10:29.145	/	tempesta-1	657681479538678246 4	0
2025-11-05 16:10:30.574	/	tempesta-2	657681479538678246 4	0
2025-11-02 16:00:03.285	/	tempesta-2	657681479538678246 4	3
2025-11-02 16:00:04.104	/	tempesta-1	657681479538678246 4	0
2025-11-02 16:00:04.234	/	baremetal	657681479538678246 4	0
2025-11-02 16:00:04.630	/	vm.web	657681479538678246 4	0
2025-11-02 16:00:05.286	/	tempesta-2	657681479538678246 4	**1380**
2025-11-02 16:00:06.104	/	tempesta-1	657681479538678246 4	0
2025-11-02 16:00:06.234	/	baremetal	657681479538678246 4	0

Tempesta Technologies

Dynamic Blocking by Tempesta Fingerprints

- WAF acceleration API
- LRU-managed `<storage_size>` only for the most aggressive clients
- `<hash> <connections/s> <messages/s>`, (`0` = full blocking)

tempesta_fw.conf :

```
tft storage_size=1073741824 {
    hash 66cbda9cafc40009 0 0;
    hash 66cbda9cafc40010 10 1000;
}

jfh storage_size=1073741824 {
    hash b1c0008c0280 0 0;
    hash b1c0008c0281 10 1000;
}
```

Tempesta Technologies

Identifying Shopping Bots

- Inventory scalping → many requests to the `cart/` URL
- Thousands of IPs, fake `User-Agent`

```
SELECT tft, tot_n, cart_n, round((100. * cart_n) / tot_n, 2) AS cart_pct
FROM (
  SELECT tft, count() AS tot_n,
         sum(if(startsWith(uri, '/cart/'),1,0)) AS cart_n
  FROM access_log
  WHERE timestamp >= now() - INTERVAL 10 MINUTE GROUP BY tfh, tft, address
) WHERE tot_n >= 20 AND (cart_n / tot_n) >= 0.3
  ORDER BY cart_pct DESC, cart_n DESC LIMIT 10;
```

tft	tot_n	cart_n	cart_pct
7407100078318223377	**32920**	**32081**	**97.45**
7407183517036511248	130	63	48.46
6567475626450092048	140	48	34.29
6567475626450092080	45	15	33.33
10415504629338275857	1262	406	32.17

Tempesta Technologies

Identifying Security Scanning Bots

- `xmlrpc.php` is still enabled by default in WordPress → L7 DDoS vector

```
SELECT uri, count(*) AS hits FROM access_log
GROUP BY uri ORDER BY hits DESC LIMIT 3;
```

uri	hits
/	894986
//xmlrpc.php	**56386**
/xmlrpc.php	**6905**

```
SELECT if(method=3,'GET',if(method=10,'POST','OTHER')) AS http_method,
       status, count() AS hits
FROM access_log WHERE uri LIKE '%xmlrpc.php'
GROUP BY http_method, status ORDER BY hits DESC;
```

http_method	status	hits
POST	**200**	**63350**
GET	405	18
POST	504	7
GET	404	1

- `POST` with response status code `200`

Tempesta Technologies

Validate TLS Tempesta Fingerprints

- **3** different TLS fingerprints with `wp_n / tot_n >= 0.7`

```
SELECT hex(tft) AS tft_hex, left(uri, 40) AS uri_short, count() AS hits
FROM access_log WHERE tft IN ('...', '...', '...')
GROUP BY tft_hex, uri_short ORDER BY hits DESC;
```

tft_hex	uri_short	hits
E51FBA42695A0010	//xmlrpc.php	60084
E51FBA42695A0010	//wp-login.php	2944
0D37190DF4E70015	/xmlrpc.php	1039
E51FBA42695A0010	//?author=1	61
E51FBA42695A0010	//wp-json/wp/v2/users/	61
E51FBA42695A0010	/	53
01C3F16C3D0D0010	/xmlrpc.php	45
E51FBA42695A0010	//wp-includes/wlwmanifest.xml	42
E51FBA42695A0010	//?author=2	42
E51FBA42695A0010	//?author=3	21
E51FBA42695A0010	//wp-includes/ID3/license.txt	19
E51FBA42695A0010	//feed/	19
E51FBA42695A0010	/blog/lean-video-co2000nferencing-billin	4
E51FBA42695A0010	/blog/fast-programming-languages-c-cpp-r	3
E51FBA42695A0010	/blog/tempesta-fw-0-7-release-wordpress-	1

^ "no ALPN"

Tempesta Technologies

L7 DDoS: Slow HTTP

- **Heaviest endpoint** accessed from **thousands** of **IPs**
  - highest cumulative `response_time` → most popular fingerprint
  - top `response_time` → just an outlier for normal clients
  - intersection of the top talkers and the top cumulative `response_time`
    - *who aims to spend server time as much as possible?*

L7 DDoS: Slow HTTP – Top Average Response Time

```
SELECT hex(tft) AS tft_hex, sum(response_time) AS tot_resp_time,
       count() AS tot_req, sum(response_time)/count() AS avg_resp_time
FROM access_log GROUP BY tft HAVING tft IN (
  SELECT tft FROM (SELECT tft, sum(response_time) AS s FROM access_log
                   GROUP BY tft ORDER BY s DESC LIMIT 20)
  ) AND tft IN (SELECT tft FROM (SELECT tft, count() AS c FROM access_log
                  GROUP BY tft ORDER BY c DESC LIMIT 20)
) ORDER BY avg_resp_time DESC;
```

tft_hex	tot_resp_time	tot_req	avg_resp_time
398A2452C0320030	21264088	7735	**2749**.0740788623143
398A9769C0320010	24137024	11469	**2104**.5447728659865
B94C7202A1480035	12519004	82918	150.98053498637208
E51FBA42695A0015	8315604	58670	141.73519686381456
66CB4E46EF170015	2869480	887197	3.2343211259731492

^
"no ALPN"

Tempesta Technologies

Tempesta WebShield

https://github.com/tempesta-tech/webshield
https://tempesta-tech.com/knowledge-base/Bot-Protection/

- Small **extensible** Python daemon
- Periodically executes queries for **trigger events**
  - Manual, configured limits or z-score thresholds on trained data
- **Detectors** (that large SQL queries) with **model validation**
  - HTTP requests and error responses per second
  - cumulative response time
  - unusual GeoIP
- **Automatically** issues **blocking** rules with **time-outs**
  - Tempesta Fingerprints, IPSet, nftables

Tempesta Technologies

Detector Validation Example

- Define detectors to use:

```
DETECTORS=["tft_rps","tft_errors", ...]
```

- Trigger event as 10 standard deviaons:

```
DETECTOR_TFT_RPS_DEFAULT_THRESHOLD=10
DETECTOR_TFT_ERRORS_DEFAULT_THRESHOLD=10
```

- Define the baseline time frame & maximum data sets intersection:

```
BLOCKING_WINDOW_DURATION_SEC=3600
DETECTOR_TFT_RPS_INTERSECTION_PERCENT = 10
DETECTOR_TFT_ERRORS_INTERSECTION_PERCENT = 10
```

- Algorithm

1) On trigger event get TLS fingerprints with **top RPS**
2) Check if the same fingerprints were also top RPS **1 hour ago**
3) If not, **goto (1)** for TLS fingerprints with **top error codes/s**

Future Work

- **Behavior analysis**
  - requested URLs ratios (e.g. `cart/` to other URLs)
  - classify delay-weighted transition graphs
- **Correlations**: TLS fingerprints, User-Agent, HTTP/2 Priority
- **Scoring**: many *probabilistic* parameters

Bots in the Wild

- Reddit at DEF CON 33 (Apr ‘25): *https://www.youtube.com/watch?v=yGYR-tE0ljw*
- Simple `User-Agent` filtration still works in ***many cases***
- TLS fingerprints still work in ***most cases***
  - GREASE (RFC 8701): random ALPN, Extensions, Cipher Suites etc.
  - Firefox Fingerprinting protection: random permutations
  - sort Extensions, Cipher Suites etc
  - replace all GREASE fields with a single flag
  - *good:* many bots **still** use unique fingerprints
  - *bad:* impersonalization libraries
    - `curl-impersonate`, `curl_cffi`, `ja3proxy` ...

Tempesta Technologies

So What About Scraping Clouds?

- **CDNs aren’t enough without site-specific bot protection**
  - cost vs profit
  - being scrapped just because there is a big target behind the CDN
  - custom protection frequently requires custom scraper
- **Use** your secret weapon – **your data**
  https://community.shopify.com/t/shopify-bot-exploit-add-to-cart-abuse-is-corrupting-analytics-shopify-refuses-to-act-at-platform/412062/13
  - *all the queries above will be different for **your resource***
  - your database knows your clients’ behavior
  - Python + ClickHouse = cheap and powerful protection

Thanks!

https://github.com/tempesta-tech/webshield

https://github.com/tempesta-tech/tempesta

https://tempesta-tech.com/tempesta-escudo/knowledge-base/XFW/

We are hiring! *https://tempesta-tech.com/careers/*

ak@tempesta-tech.com

https://www.linkedin.com/in/alexander-krizhanovsky/

https://x.com/a_krizhanovsky

Tempesta Technologies