# WAF: Wrong Approach Firewall

2026-06-07

https://fosstodon.org/@slink

# /me

- Born when UNIX time fit into 28 bits
- Linux since ~1992 (kernel ~0.9.8 IIRC)
- Learned “everything” from FOSS
- Since 2009, Independent Developer and Consultant
- Varn^WVinyl-Cache Maintainer (1 of 3)
- Runs a small company (UPLEX)
- Clients are several CDNs and websites
  - You know them all, but I will not tell you

# Disclaimer

- This is very much the perspective of
  - A FOSS Person
  - Working with Vinyl Cache
- First hand experience with „Cloud WAFs“ (Imperva, Akamai)
- … but no extensive first hand experience with other commercial products
- I want to talk about how things work, not what to buy
  - But you sh^Wcan hire me for your project

# Why this matters to me

- I want more people/organizations/states to become digitally sovereign again
  - Which, to me, means to be *in power* of the technology used and *independent* of the tech oligopoly (cloudflare has ~80% market share)
- I want to help provide *to everyone* the tools otherwise only available in commercial CDNs
  - Delivering *enterprise* features as FOSS
- WAF functionality is a key component
  - But I want to also deliver the *right* solution

# Show of hands

- Who has heard of VINYL Cache ?

# Show of hands

- Who has heard of Varnish Cache ?

# Why Vinyl Cache

https://vinyl-cache.org/

- Fast! Typical >10kreq/s per core
- Domain language VCL to manipulate HTTP & content
  - Gets compiled into machine code
- Real-time logging and stats, highly efficient
- Lots of modules to extend core functionality
- But of course the discussed topics are not specific

# Outline

- Firewall, WAF
- HTTP
- Signed URLs / signed requests
- Regular Expressions
- The relevance for HTTP Caching

# Why a firewall?

- If all our computers and applications were safe, we would not need any firewalls
- Firewalls are an independent instance
  - Different Software
  - Independently Configured
- Prevent f*ups
  - Don’t tell me that your server software is secure
- Sure, not perfect either
  - Just make it unlikely that two components have the same security issue.

# Network Firewall

- A (stateful) Layer 4 Router with Filters
- Basic Algorithm
  - Look at Packet
  - Belongs to known accepted connection ? Accept
  - Else check Rules
- Rules look at L3 / L4 Information
  - IP Addresses, Protocols
  - TCP/UDP Ports
- Rewrite Addresses/Ports (NAT), Sequence Numbers, Rate Limit / shape traffic etc. etc.

# Typical network firewall

- OUT:
  - Allow DNS (53/udp)
  - Allow HTTP(s) (443/tcp&udp)
  - Allow some ICMP (maybe ping, path MTU discovery)
  - (Maybe SMTP ...)
  - Deny ALL
- IN:
  - Allow HTTP(s) (80/tcp, 443/tcp&udp)
  - Deny ALL

# Network Firewall

- Accepted best practice:

  Only allow known good traffic

  → positive security model (allowlist)

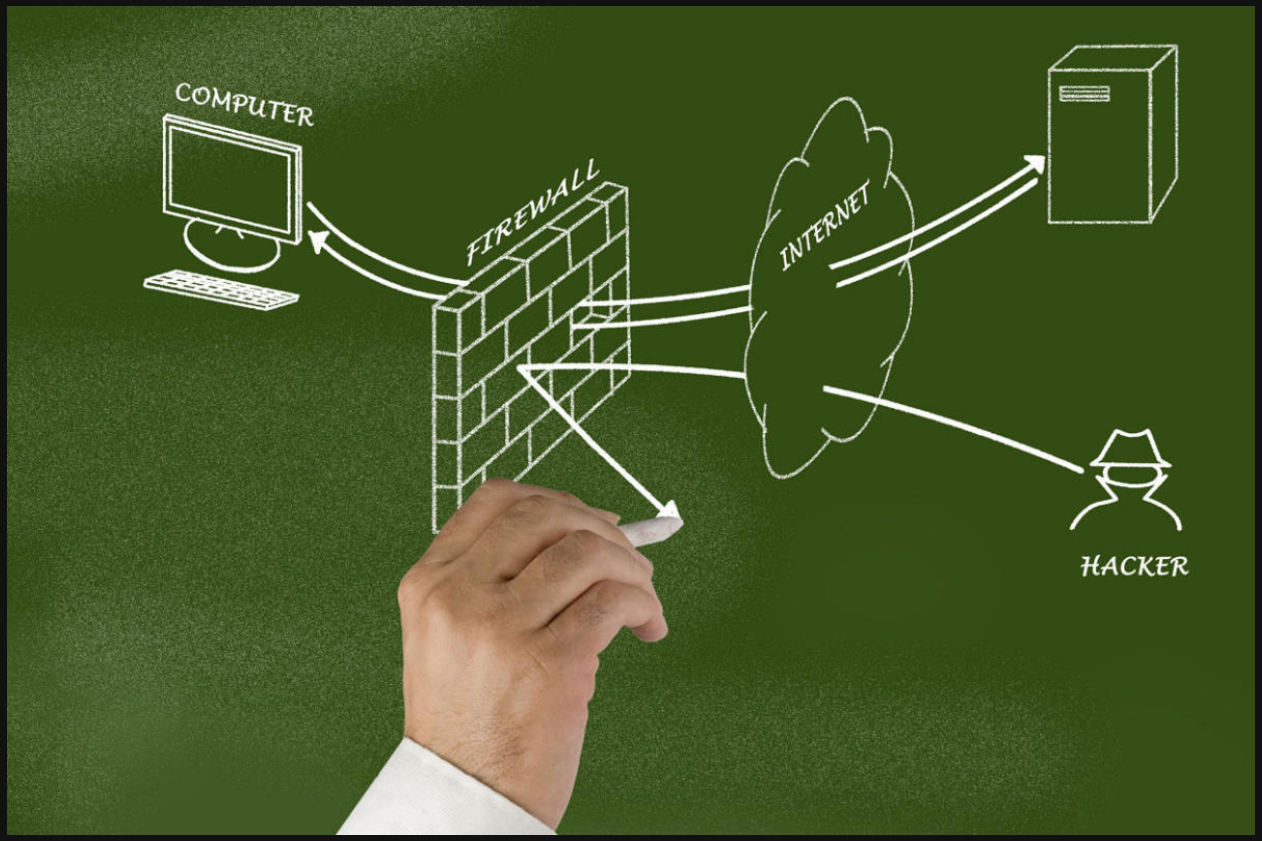
# Network Firewalls are not sufficient

- Common Ruleset is

  „deny everything but https“
- But „all“ traffic is *https* anyway.
- And everything is encrypted

  → ?!?

# Web Application Firewall




https://images.techhive.com/images/article/2015/05/firewall_thinkstock-100583207-large.jpg

# WAF

- Act on clear HTTP
  - Some exceptions, e.g. JA4-based inspecting TLS
- Inspect HTTP request and/or response
- At origin (e.g. mod_security)
- Proxy under your control (Vinyl Cache)
- CDN/Cloud
  - This implies re-encryption! (breaks TLS promise)
  - „Adversary in the Middle (AitM) as a service

# WAF models

- Negative model / denylist / blocklist
- „Self learning“
- Positive model / allowlist

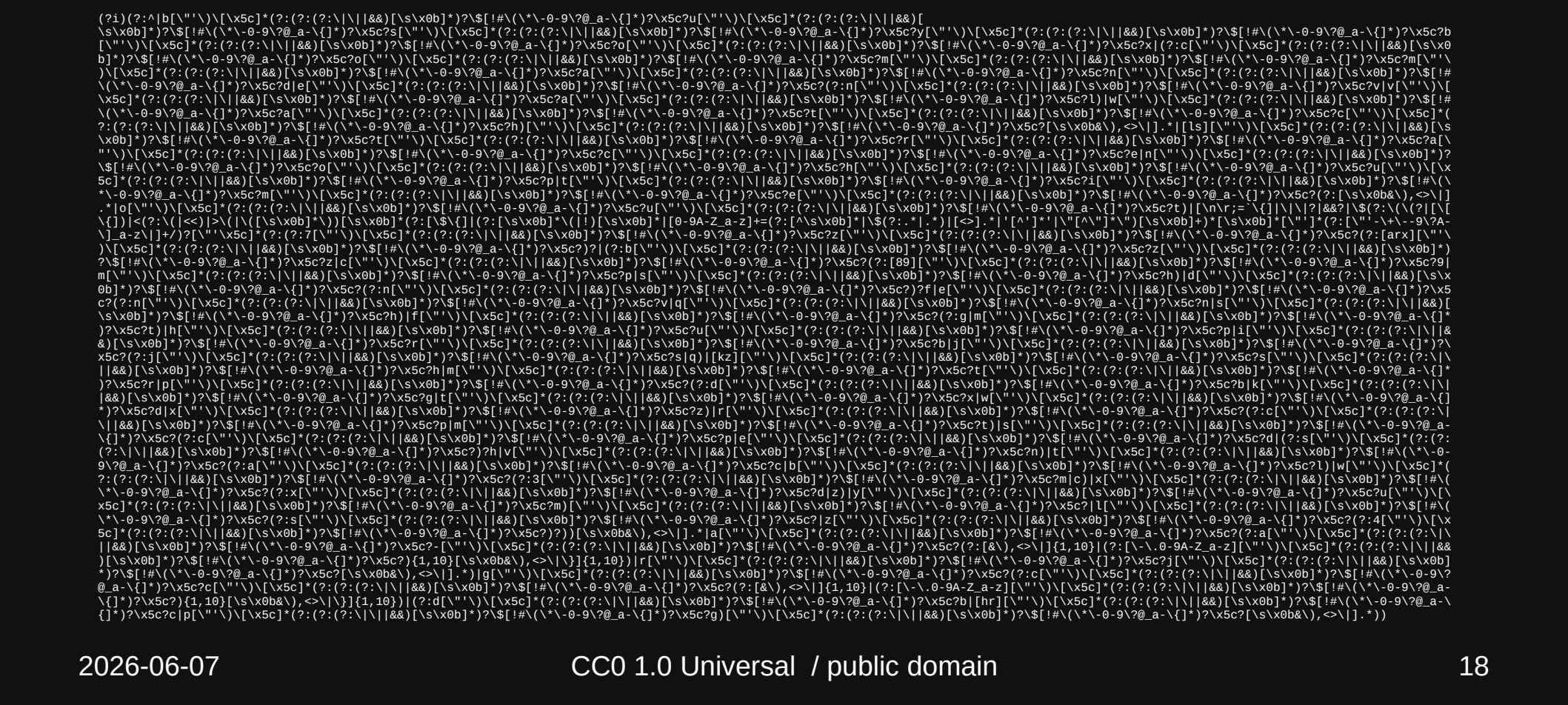
# Wrong Approach Firewall

- Most WAFs follow a Negative Security Model („denylist / blocklist“)
  - Identify „bad requests“ and block them
  - „if bin/bash is contained in the request, it must be evil“

    https://github.com/coreruleset/coreruleset/blob/v4.0/dev/rules/unix-shell.data

# Ragebait

- https://github.com/coreruleset/coreruleset/blob/main/rules/REQUEST-932-APPLICATION-ATTACK-RCE.conf



# Ragebait off

- Yes, it’s generated
- Case in point:
  - This stuff *is* fckn complex
  - Don’t talk to me about „I just want a simple solution“. This is not it.

# Wrong Approach

- PRO (illusion)
  - „just works“ on any application
- CON
  - it does not, you get to deal with false positives all the time
    - „just disable the offending rule“
  - Only works for existing attacks
  - Modifying the attack just slightly might circumvent
  - Slow
  - LOTS of hidden complexity

# Wrong Approach

- CON cnt'd: Negative security model (denylist)
  - Simplicity illusion
  - False sense of security
  - In practice, disabling rule sets because of errors opens attack vectors
  - Lots of rules which do not apply to software used
  - High internal complexity
  - Reactive, can only mitigate known attacks

# Self Learning

- Inspect traffic for some time
- Generate generalizing rules
  - If we see 123 and 456, allow all numbers

- Issues:
  - What if it learned adversarial traffic?
  - Sensitivity dilemma: Need to trade off between false positives and false negatives → no protection when it counts
  - „black box“

# Self Learning with AI

- Do I really need to comment on that?

- „Oh, if we hand it over to a stochastic predictor, it will do the right thing magically“

# WAF done right

- Positive security model:
  - Allow known good, deny all else (or modify request)
- PRO
  - Protect against 0days
  - Caching: faster, more resilient site as an aside
- CON
  - Needs to be *designed in* or retrofit with substantial effort

# Fundamentals

- So now we know the wrong approach and the right approach, how do we get there?

  → understand some basic concepts

# HTTP

> GET|POST /path HTTP/1.1
> Foo: Bar
> ...
> [body (for POST, PUT]

< HTTP/1.1 200 OK
< Bar: Buzz
< ...
< [body]

# HTTP

> GET|POST /path HTTP/1.1
> Foo: Bar
> ...
> [bo

< HTTP/1.1 200 OK
< Bar: Buzz
< ...
< [body]

Request Line: Method Target HTTP-Version

Target: origin-form | absolute-form | authority-form | „*“

Origin-form: absolute-path “?“ query

Status Line: HTTP-Version Code [Reason]

# HTTP

> GET|POST /path HTTP/1.1
> Foo: Bar
> ...
> [header]

Request Header-Field: Field-Name: Field-Value

< HTTP/1.1 200 OK
< Bar: Buzz
< ...
< [header]

Response Header-Field: Field-Name: Field-Value

# HTTP

> GET|POST /path HTTP/1.1
> Foo: Bar
> ...
> [body (for POST, PUT]

< HTTP/1.1 200 OK
< Bar: Buzz
< ...
< [body]

- Conceptually the SAME with HTTP/2 and /3!

# Characters

- Target:
  - Unreserved: `a-z A-Z 0-9 - . _ ~`
  - Pct-encoded: %xx (`%2f` == /)
  - Sub-delims: `! $ & ' ( ) * + , ; =`
  - And also: `: @`
- Fields: Basically anything
  - which is not a delimiter

# HTTP: Identify legit requests

- The client can send ANYTHING, for example
  - Modify request target (URL)
    - Append/Change Query
    - Use Percent-Encoding (in weird ways)
    - …
  - Add arbitrary header fields
    - With arbitrary values

→ First option: Signing

# Signing

- Using signatures, we can know (at the WAF) what is valid and what not
  - Because only we can create the signature
- Generic format:

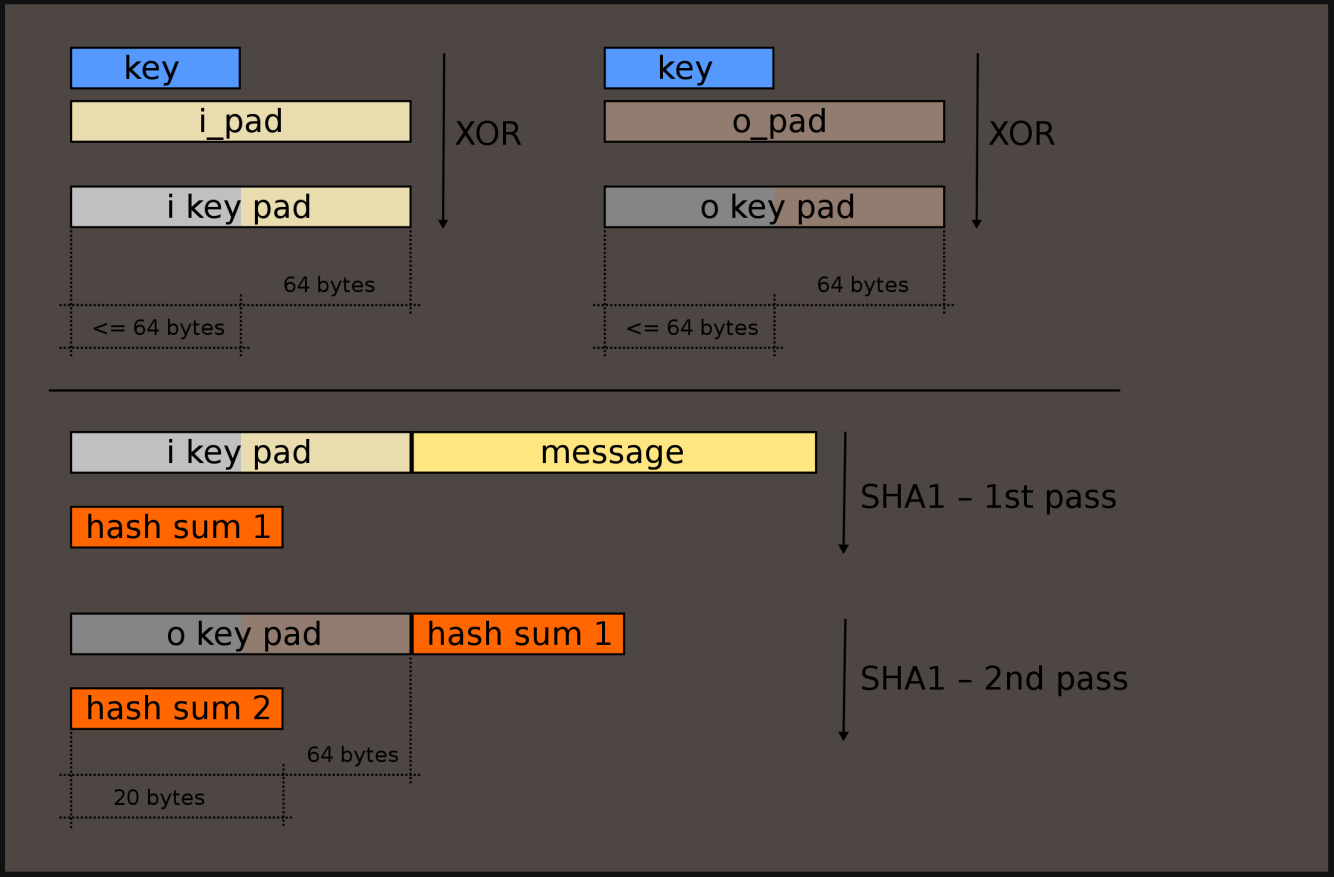
  <signed-part><delimiter><signature>[unsigned-part]

# Algorithms

- Hashes: Symmetric (e.g. SHA256 HMAC)
  - Keys for signing and validation are the same
  - → use if you control both
  - Short-ish signature, Very fast (Millions per second)
- Asymmetric (e.g. ECDSA256, RSA2048)
  - Signing key can be contained
  - „Validation everywhere“
  - RSA: Longer signature, slower, (more secure?)
- Post Quantum: No personal experience yet

# Signing: HMAC

- ALWAYS USE for Symmetric hashes
- „Adds a password to the hash function“




Source: https://en.wikipedia.org/wiki/HMAC

# Signatures examples

- Signed URLs
  - Application generates Links as signed URLs
  - Vinyl Cache / WAF can verify these

    `/path?foo=bar&sig=8iYnRwQFtFm7PUywch62I3uvU5WNPsceyFNK2_-Exks`
  - https://vinyl-cache.org/tips/signed_urls/index.html
- Signed Cookies
  - Tips & Tricks article: https://vinyl-cache.org/tips/cookiesign/index.html

# Tokens

- Add an expiry timestamp to your signed thing and you have a „Web Token“
  - Has the resource or identifier for what a used is allowed to do
  - Has a valid time stamp
  - Can not be forged because of the signature
- Examples
  - JSON Web Token (JWT) https://www.jwt.io/
  - Akamai Auth Token 2.0 https://techdocs.akamai.com/adaptive-media-delivery/docs/auth-token-20-verification-amd

# Token Use

- Tokens are usually used for „Content Protection“ (Access to Video streams, Paywalls)
- But you can also use them for security
- JWT:
  - JSON header + JSON payload + signature
  - All as base64, separated by .
  - validation in VCL:

https://gitlab.com/uplex/varnish/libvmod-frozen/-/blob/master/examples/jwt/jwt.vcl

# JWT example

- I built a CDN where the URLs *are* JWTs

  `https://c.d.n/eyJhbGciOiJSUzI1NiIsImtpZCI6InZ0YyIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3NzQ1MjY0NjcsIm9pZCI6InZ0YyIsIm…`

- Super convenient to have structured data (JSON)
  - Which object
  - On which storage
  - …

# Where are we now?

- Signed-everything gives us an ideal WAF:
  - Access only possible to objects / with cookies which were explicitly allowed by the application
  - Only need a very small set of fixed „entry URLs“ (ideally only one, the home page)
  - „Zero attack surface“ (= *very small)*
- We can implement the optimal case if we design for it from day one
- But the real world is messy…

# In the real world, we need pattern matching

- Pattern matching means regular expressions
- Master them!
- I wanted to include a primer, but it was just too much
- But just one thing…

# Regular expressions

- Quick quiz: Who of you thinks they know regexen reasonably well?

The following slides have been corrected after the talk.

A detail had been wrong, which was not relevant for the point to make, but still wrong. Thank you, anonymous reporter!

# What do you think

- … is the regular expression equivalent of glob (shell)

  /*/

  in a regular expression **if used on urls**?
- In the following /…/ is not the regex-delimiter, / are not special, so /a/ matches literally „/a/“

# Glob /*/ → This?

gpn24.de

```
/.*/
```

# Glob /*/ → Or this?

```
/[^/]+/
```

# Glob /*/ → Or this?

```
/[^./][^/]*/
```

# Glob /*/ → THIS

gpn24.de

```
/[^./?][^/?]*/
```

# Enough…

- Back to WAFs

# For the real world, we need to

- Understand HTTP:
  - For each path (pattern)
    - Which request methods are allowed?
    - Which query parameters are allowed?
    - Which request headers are needed with which values?
    - Optionally, match body data (hard in the general case)
  - Optionally, also check the response
  - Less is more!

# Large scale pattern matching: The problem

```
if (req.url ~ "^/foo/") {
        if (req.method == "GET") {
              …
        }
        elsif ...
}
elsif (req.url ~ "^/bar/") {
        ...
}
elsif (req.url ~ "^/baz/") {
        ...
}
elsif (req.url ~ "^/quux/") {
        ...
}
```

# Set matching (url, query)

- https://vinyl-cache.org/tutorials/vmod_sets.html
- VCL can call code for matches

```
ub vcl_init {
      # The sub parameter associates a subroutine with each element.
      new path_matcher = selector.set();
      path_matcher.add("/foo", sub=recv_foo);
      path_matcher.add("/bar", sub=recv_bar);
      path_matcher.add("/baz", sub=recv_baz);
      path_matcher.add("/quux", sub=recv_redirect, string="www.quux.com");
}
sub vcl_recv {
      if (path_matcher.hasprefix(req.url)) {
              call path_matcher.subroutine();
      } else {
              Return (synth(404));
}
```

# Why is this important

- Reduces matcher complexity from *O(n * l)* to *O(l)* for n rules and url length *l*
- In VCL, we can call *code* for each match, which allows for tremendous flexibility while keeping the code manageable

# Filter headers

- https://vinyl-cache.org/tutorials/hdr_filter.html

```
sub vcl_init {
    new req_minimal = re2.set(
        anchor=start,
        case_sensitive=false);
    req_minimal.add("Host:");
    # vinyld normalizes to just gzip
    req_minimal.add("Accept-Encoding:");
    req_minimal.add("If-(Modified-Since|None-
Match):");
}
sub vcl_recv {
    req_minimal.hdr_filter(req);
}
```

# Body matching example


```
import re;
sub vcl_init {
        new re_image = re.regex("(?i)^(?:" +
            "\xFF\xD8\xFF\xE0[\x00-\xFF]{2}JFIF|" + # JPEG/JFIF
            "\xFF\xD8\xFF\xE1[\x00-\xFF]{2}Exif|" + # JPEG/Exif
            "\x89PNG\x0D\x0A\x1A\x0A|" +            # PNG
            "GIF8[79]a|" +                          # GIF
            "BM.{4}[\x00-\xFF]{2}\x00\x00|" +       # BMP
            "RIFF[\x00-\xFF]{4}WEBP|" +             # WebP
            "\x00\x00\x00[\x14-\x28]ftypavif" +     # AVIF
            ")", forbody = true);
}
sub vcl_deliver {
        if (! re_image.match_body(resp_body)) {
                return (synth(500, "No image data seen"));
        }
}
```

# Body matching

- Tutorial coming really soon
  - https://code.vinyl-cache.org/vinyl-cache/homepage/pulls/152
- Get notified:
  - RSS/Atom: https://vinyl-cache.org/atom.xml
  - Mastodon: https://fosstodon.org/@vinyl_cache

# Now you have all the building blocks

- … in just ~3 slides

- The remaining hard part is putting it all together
- Usually with a code generator
  - I use python for json → vcl

# How do we make this accessible?

- We need generators
  - Swagger/OpenAPI to VCL?
- I'd imagine some kind of generalized „WAF swagger", which includes „web" behavior
- Are there any existing projects?

# But a simple WAF is easy

- Check host name
- Allow only GET and HEAD (should be all yout blog needs)
- No request body
- Strip query parameters
- Remove almost all headers

→ will break for anything more complex, but it is a starting point

# Simple WAF

```
# req_minimal from before

sub vcl_recv {
    if (req.http.host != "example.com") {
        return (synth(404));
    if (req.method != "GET" && req.method != "HEAD") {
        return (synth(405));
    }
    if (req.http.Content-Length || req.http.Transfer-
Encoding) {
        return (synth(413));
    }
    # strip query parameters
    if (req.url ~ "\?") {
        set req.url = regsub(req.url, "\?.*", "");
    }
    # note: removes Cookie, Authorization etc...
    req_minimal.hdr_filter(req);
}
```

# Caching

- By reducing variance of the request, you optimize caching (in Vinyl Cache)
- By implementing caching, you
  - make the site resiliant against DoS, load spikes etc

# Thank you!

- To all the people making GPN possible
- To https://www.sovereign.tech/ for investing into our work

**Sovereign Tech Agency**