PyRASP

Python *Runtime Application Self Protection*

Defending your Python Web Application
From the Inside

Renaud Bidou

ParaCyberBellum

PyRASP 101

INSTALL & CODE

```
C:\Tests> pip install pyrasp
```

```
from flask import Flask, request, Response
from werkzeug.exceptions import HTTPException
from waitress import serve

app = Flask(__name__)

from pyrasp import FlaskRASP
rasp = FlaskRASP(app)

@app.route('/', methods = [ 'GET' ])
def root():
    return 'Hello', 200
```

RUN & TEST

```
C:\Tests> python testflask.py

### PyRASP v0.8.3 ##########
[+] Starting PyRASP
[+] Loading default configuration
[+] XSS model loaded
[+] SQLI model loaded
[+] PyRASP succesfully started
##############################


[!] XSS: qs_values ->
(()=>{})["constructor"](...["alert(window.origin
)"].map(s=>String.fromCharCode(...s.split("").ma
p(c=>c.charCodeAt(0))))).call()
[!] Blacklisted IP: source_ip -> 194.98.65.65
[!] Blacklisted IP: source_ip -> 194.98.65.65
```

DESIGN CRITERIA

1. Secure & Signature-Free
2. Lightweight
3. Oneliner
4. Distributed & Multi-Platform
5. Useful Logging & Telemetry

WHY RASP ?

Natively Resistent

Request Smuggling

Encoding Tricks

HTTP Parameter Pollution

Environment Aware

Targeted System Protection

Framework Specificities Handling

Access to Application Internals

DevOps Friendly

Embedded in Application Code

Native CI/CD Pipeline Integration

MAIN SECURITY CHECKS & ENGINES

Flood & Brute Force	Threshold
Request Validation	Application Internals
Spoofing	Header Validation
Decoy	Path
SQL Injection	~~Grammatical Analysis~~ + Machine Learning
XSS	Machine Learning
Command Injection	System Internals
HPP	Grouping
DLP	RegExp

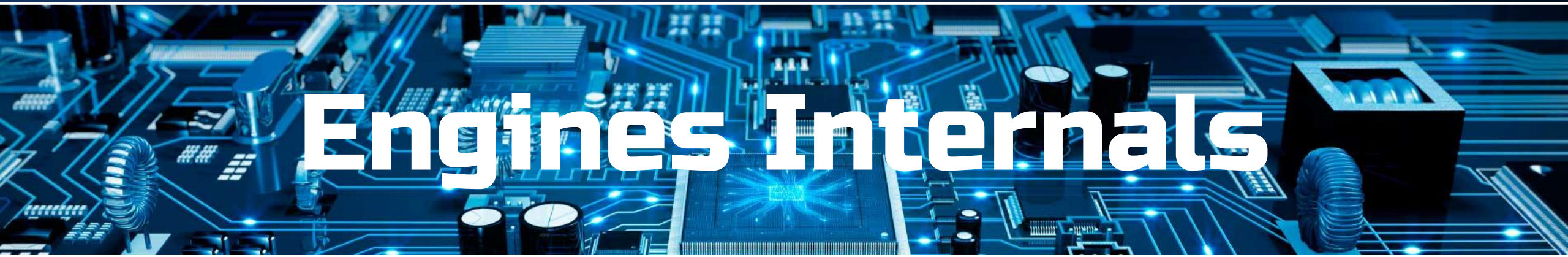
SUPPORTED PLATFORMS

FRAMEWORKS

- FLASK
- DJANGO
- FASTAPI

SERVERLESS

- AWS LAMBDA
- AZURE FUNCTIONS
- GCP FUNCTIONS



Engines Internals

INTERCEPTING REQUESTS & RESPONSES

```
def register_security_checks(self, app)
```

FLASK

```
@app.before_request
@app.after_request
```

FASTAPI

```
@app.middleware('http')
```


DJANGO

```
MIDDLEWARE = [ 'pyrasp.DjangoRASP', … ]
def __call__(self, request)
```

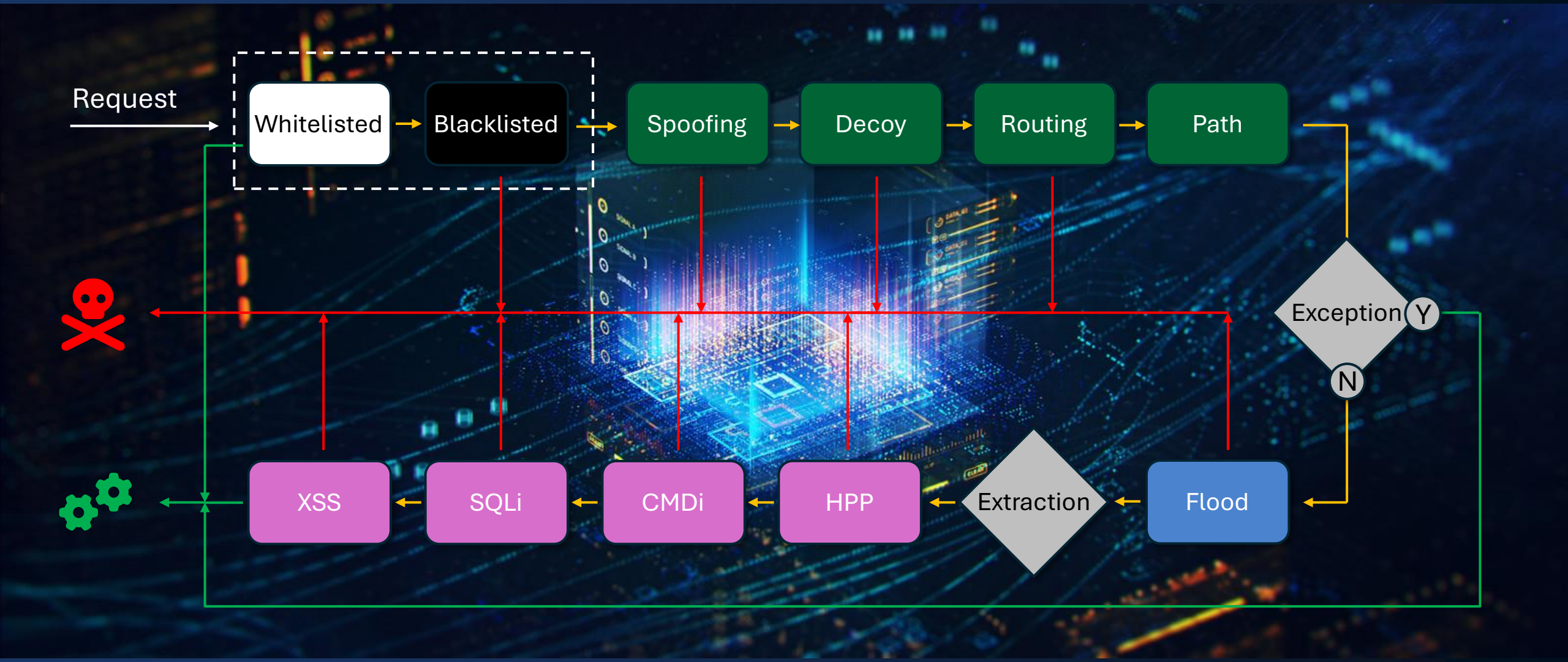
AWS LAMBDA

AZURE FUNCTIONS

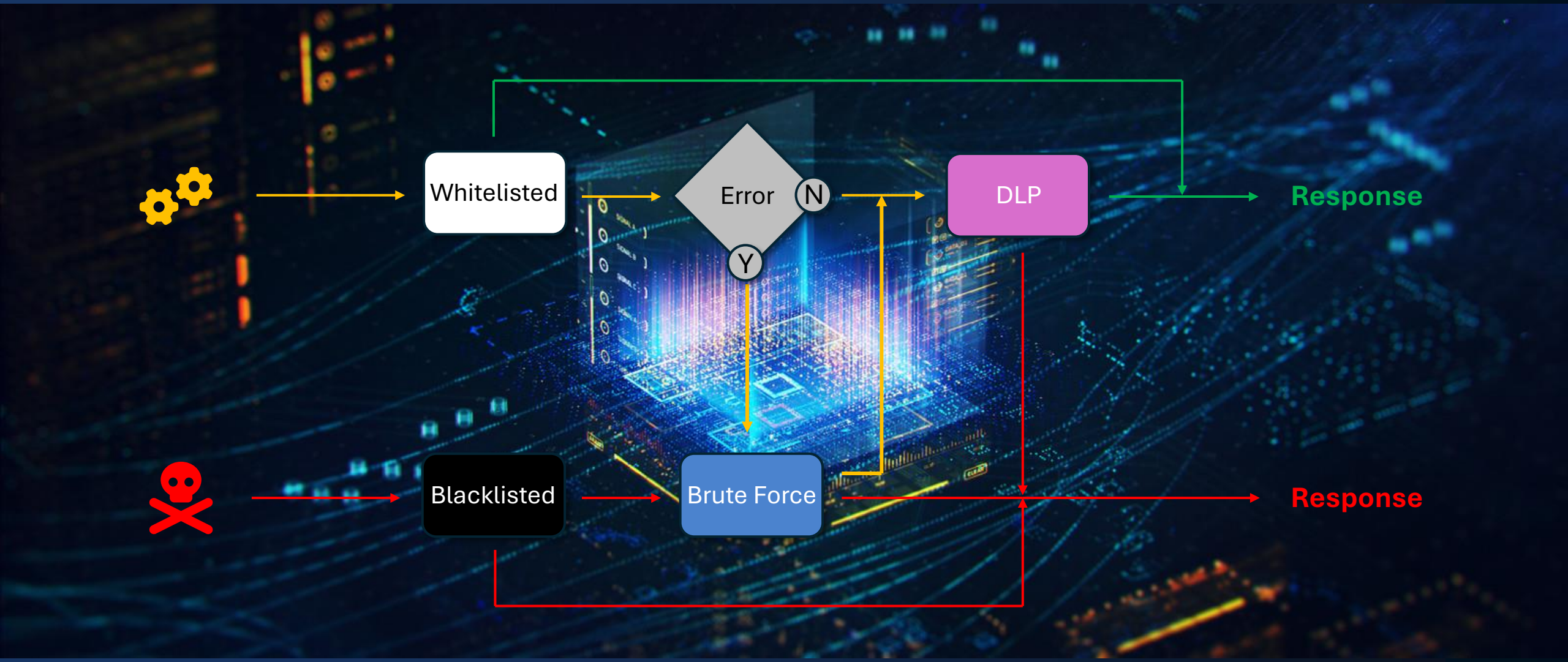
GCP FUNCTIONS

```
def decorator(request, context)
```

REQUEST PROCESSING OVERVIEW



RESPONSE PROCESSING OVERVIEW



SIMPLER (and most efficient) ENGINES

DECOYS

1. Set of commonly targeted paths

   `^/.env` `^/.git` `^/.aws` `/wp-` …

2. Attempt to connect ⇨ Blacklisted

* 0% False Positive

SPOOFING

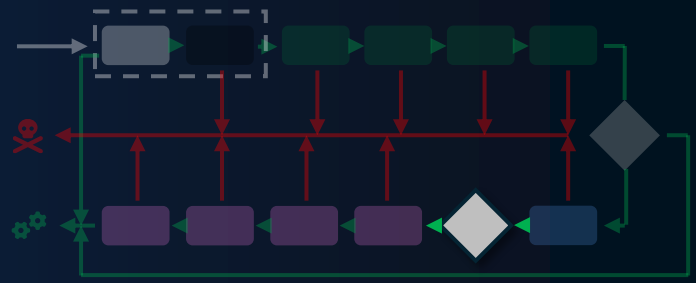
1. Check `Host` header

2. Doesn't match configuration
   ⇨ Blacklisted

* Scanners & Direct access prevention

99.99% Early Attack Detection

EXTRACTION



BASICS

Headers Names & Values

- Cookies Names & Values
- Referer
- User Agent

Query String Variables & Values

Posted Data Variables & Values

JSON Data Variables & Values

TRICKS

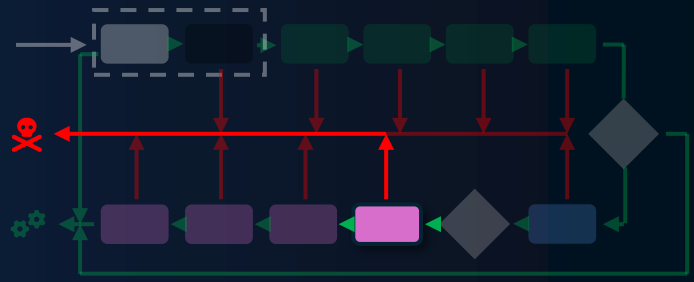
Base64 encoded values

- Decode
- Parse JSON
- Extract Variables & Values
- Base64 Decode
- Recurse (rare cases… so mandatory)

Example : JWT

HPP: TRIVIAL BUT...

Microsoft Azure Functions join duplicated parameters with comma



The Code

```
def testazure(req: func.HttpRequest) -> func.HttpResponse:
    params = dict(req.params)
```

The Query String

```
?a=select%20login&a=password/*&a=*/%20from%20/*&a=*/%20users#
```

The Outcome

```
params = {
    "a": "select login,password/*,*/ from /*,*/ users#"
}
```

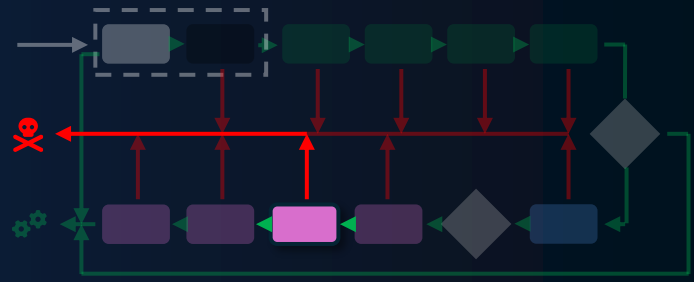
MSRC Case 87582

We examined your report and found that this is not a relevant security threat. The finding describes an assumption present in azure functions.

These are customer owned apps and at the http layer, we don't modify the format of any customer defined parameters. It's the responsibility of the customer to ensure that the parameters they're taking from the internet are not passed onto downstream components in an unsafe manner.

This is not a vulnerability. This case is now closed.

COMMAND INJECTION



1. Split stacked commands

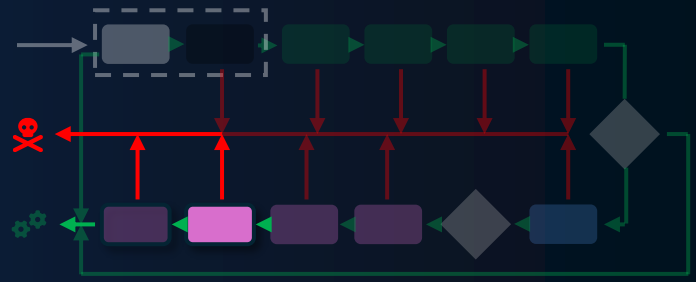
```
command_pattern = '(?:[&;|]|\$IFS)+\s*(\w+)'
commands = re.findall(command_pattern, str(injection)) or []
```

2. Call `shutil.which`

```
for command in commands:
    if shutil.which(command):
        command_injection = True
```

* Stick to the OS

GRAMMATICAL ANALYSIS (REMOVED)



1. Define injection points

```
'select * from test where id={{vector}}'
```

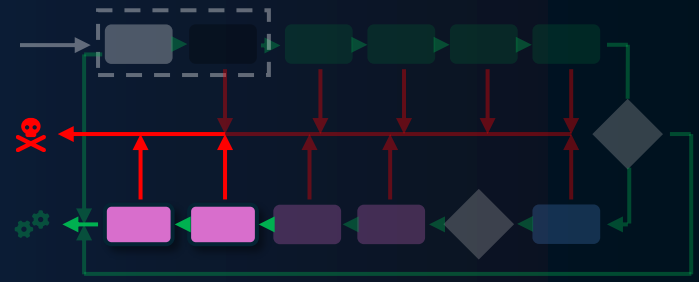
2. Replace {{vector}} with potential injection

3. Test statement against in-memory sqlite DB

```
temp_db = sqlite3.connect(":memory:")
try:
  temp_db.execute(statement)
except Exception as e:
  if 'no such table' in str(e):
    sql_injection = True
```

Was grammatically correct

XSS & SQLI ML ENGINES



Input Data

	XSS	SQLi
Valid	11146	2002
Attacks	10131	1615

Vectorizer

	XSS	SQLi
Features	3374	8580

Classification

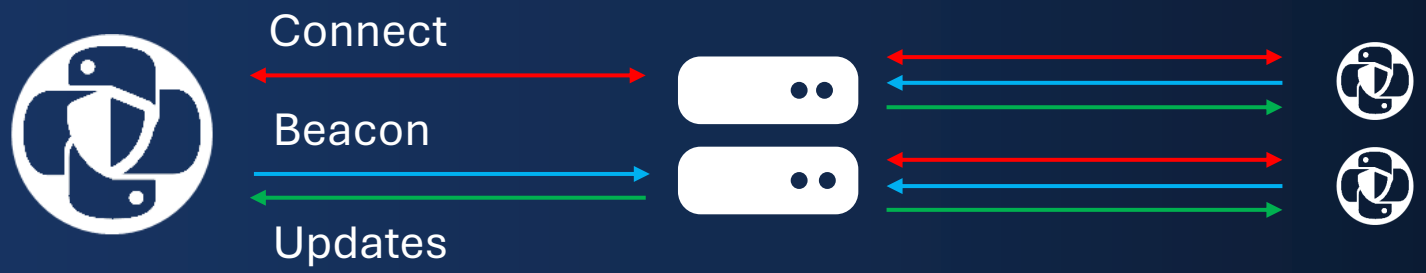
	XSS	SQLi
Model	RFC	LSV
Accuracy	0.99891	0.99351
Precision	0.99915	0.99474
Recall	0.99857	0.99059
F1 Score	0.99886	0.99265

False-Negative: Recall

False-Positive: Precision

S'More...

DISTRIBUTED INFRASTRUCTURE



Connect

Routes upload
Configuration download

Beacons

New blacklist entries
Telemetry

Updates

Blacklist updates (new – delete)
Configuration changes

Blacklisted once ⇨ **Blacklisted everywhere**

Zero-Trust Application Access

https://github.com/rbidou/pyrasp-agent

LOGS

Syslog

```
[<event_time>] "<application_name>" - "<event_type>" - "<source_ip>" - "<country>" - "<location>:<payload>",
"<mitre_code> - <pcb_code>", "<action>"
```

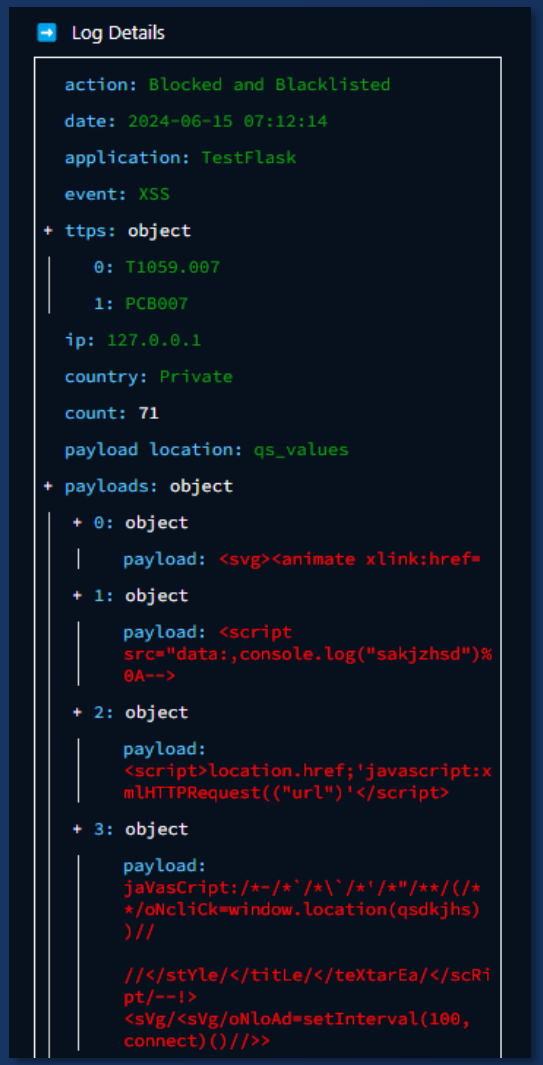
JSON / Webhook

```
{
   "time": "<event_time>",
   "application": "<application_name>",
   "log_data": [
      "<event_type>",
      "<source_ip>",
      "<country>",
      {
         "path": "<path>",
         "location": "<location>",
         "payload": "<payload>",
         "codes": "<codes>",
         "action": "<action>",
         "engine": "<engine>",
         "score": "<machine_learning_score>"
      }
   ]
}
```

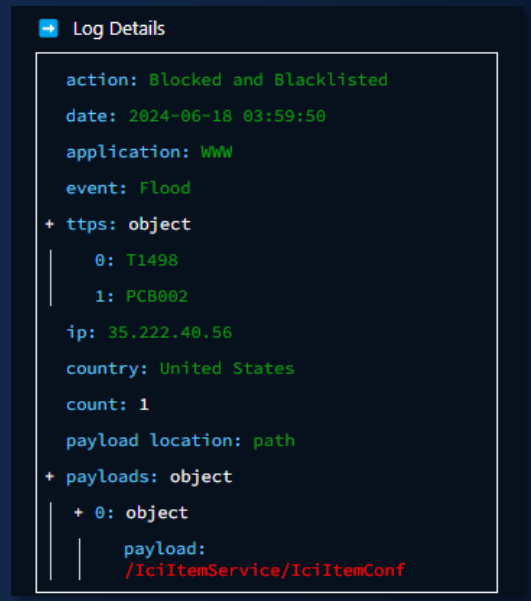
Beacon Telemetry

```
{
   "key": "<agent-key>",
   "version": "<agent-version>",
   "telemetry": {
      "cpu": <cpu_usage_percent>,
      "memory": <memory_usage_percent>,
      "requests": {
         "success": <valid_count>,
         "error": <errors_count>,
         "attacks": <attacks_count>
      }
   }
}
```

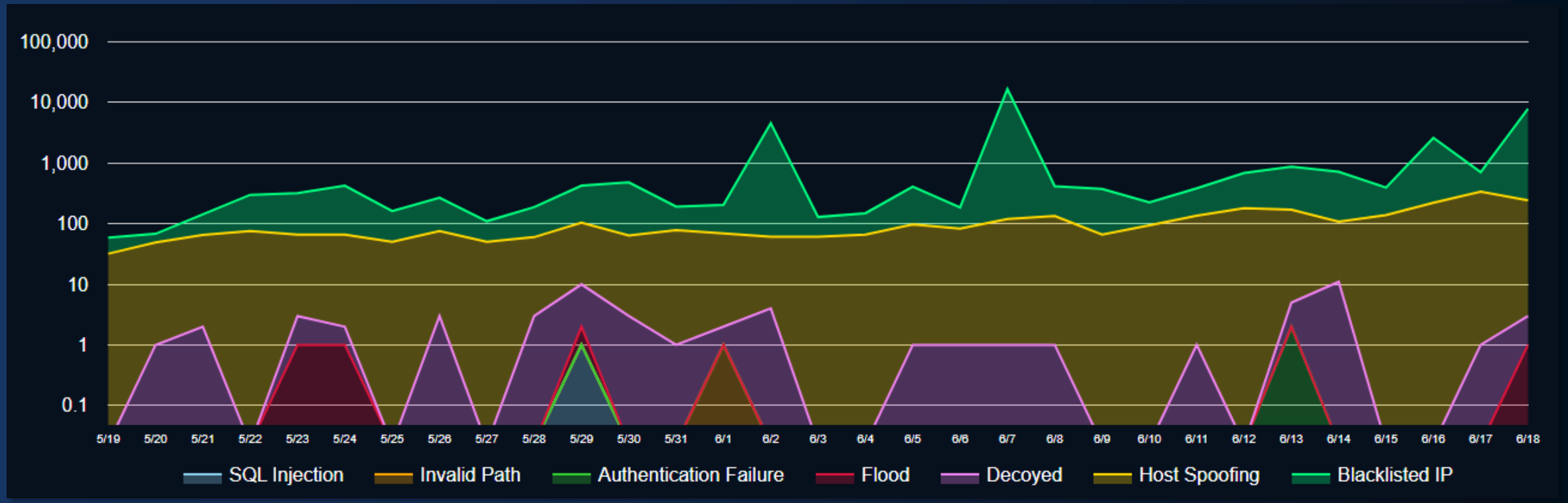
LOGS USAGE


Log Details
action: Blocked and Blacklisted
date: 2024-06-15 07:12:14
application: TestFlask
event: XSS
+ ttps: object
0: T1059.007
1: PCB007
ip: 127.0.0.1
country: Private
count: 71
payload location: qs_values
+ payloads: object
+ 0: object
payload: <svg><animate xlink:href=
+ 1: object
payload: <script src="data:,console.log("sakjzhsd")%0A-->
+ 2: object
payload: <script>location.href;'javascript:xmlHTTPRequest(("url")'</script>
+ 3: object
payload: jaVasCript:/*-/*`/*\`/*'/*"/**/(/**/oNcliCk=window.location(qsdkjhs))//
//</stYle/</titLe/</teXtarEa/</scRipt/--!>
<sVg/<sVg/oNloAd=setInterval(100,connect)()//>>

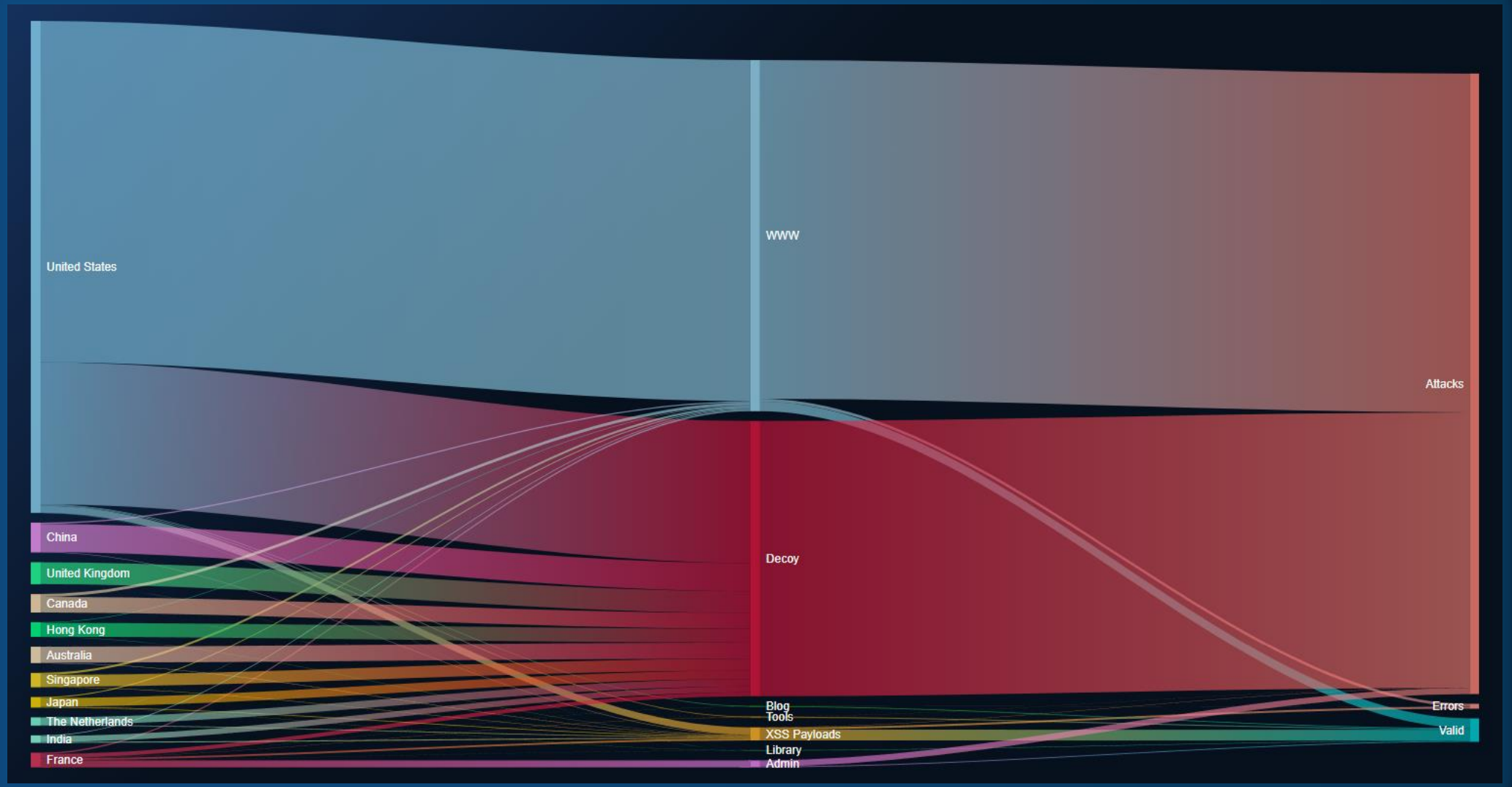

Log Details
action: Blocked and Blacklisted
date: 2024-06-15 13:20:16
application: TestFlask
event: SQL Injection
+ ttps: object
0: T1111
1: PCB006
ip: 127.0.0.1
country: Spain
count: 144
payload location: qs_values
+ payloads: object
+ 0: object
payload: ');confirm(1);//
+ 1: object
payload: 'test'
+ 2: object
payload: 1 or 1 = 1
+ 3: object
payload: 1' or 1 = 1 #
+ 4: object
payload: foo' or 'john dooe' not like 'mr. x
+ 5: object
payload: 1 AND sleep(ascii(SUBSTRING(@@DATABASE,1,1)))
+ 6: object
payload: 1 AND 1=CAST(@@DATABASE AS INT)--


Log Details
action: Blocked and Blacklisted
date: 2024-06-18 03:59:50
application: WWW
event: Flood
+ ttps: object
0: T1498
1: PCB002
ip: 35.222.40.56
country: United States
count: 1
payload location: path
+ payloads: object
+ 0: object
payload: /IciItemService/IciItemConf

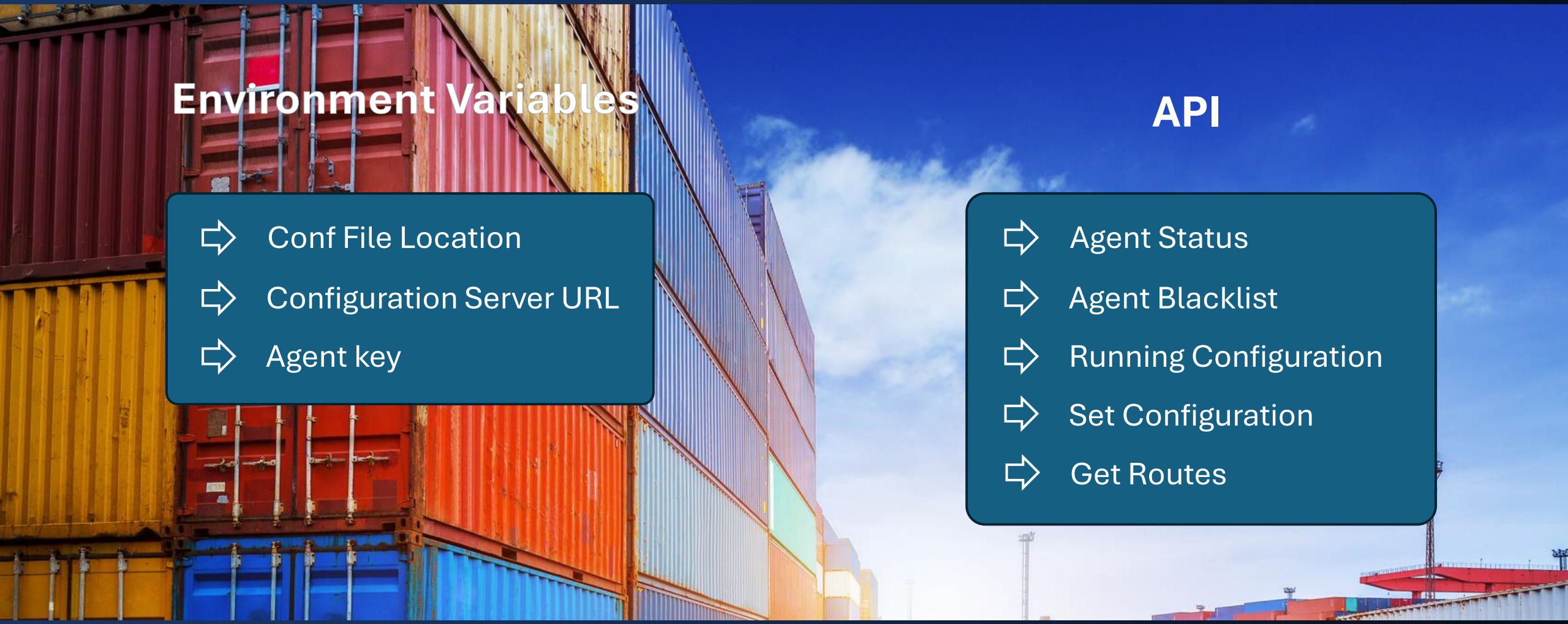
EARLY DETECTION EFFICIENCY



TELEMETRY: OUTPUTS



DEVOPS FRIENDLY



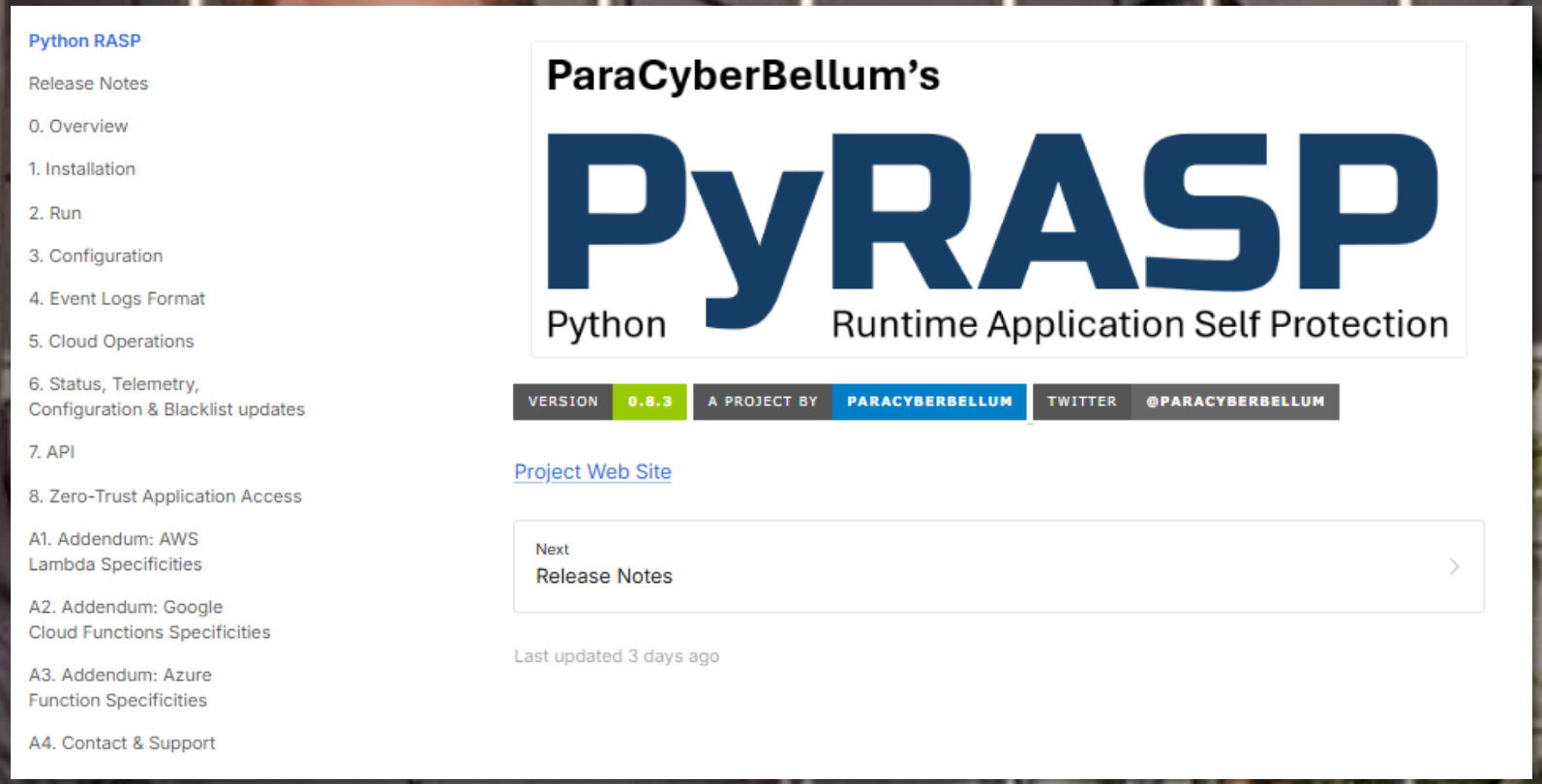
Environment Variables

- Conf File Location
- Configuration Server URL
- Agent key

API

- Agent Status
- Agent Blacklist
- Running Configuration
- Set Configuration
- Get Routes

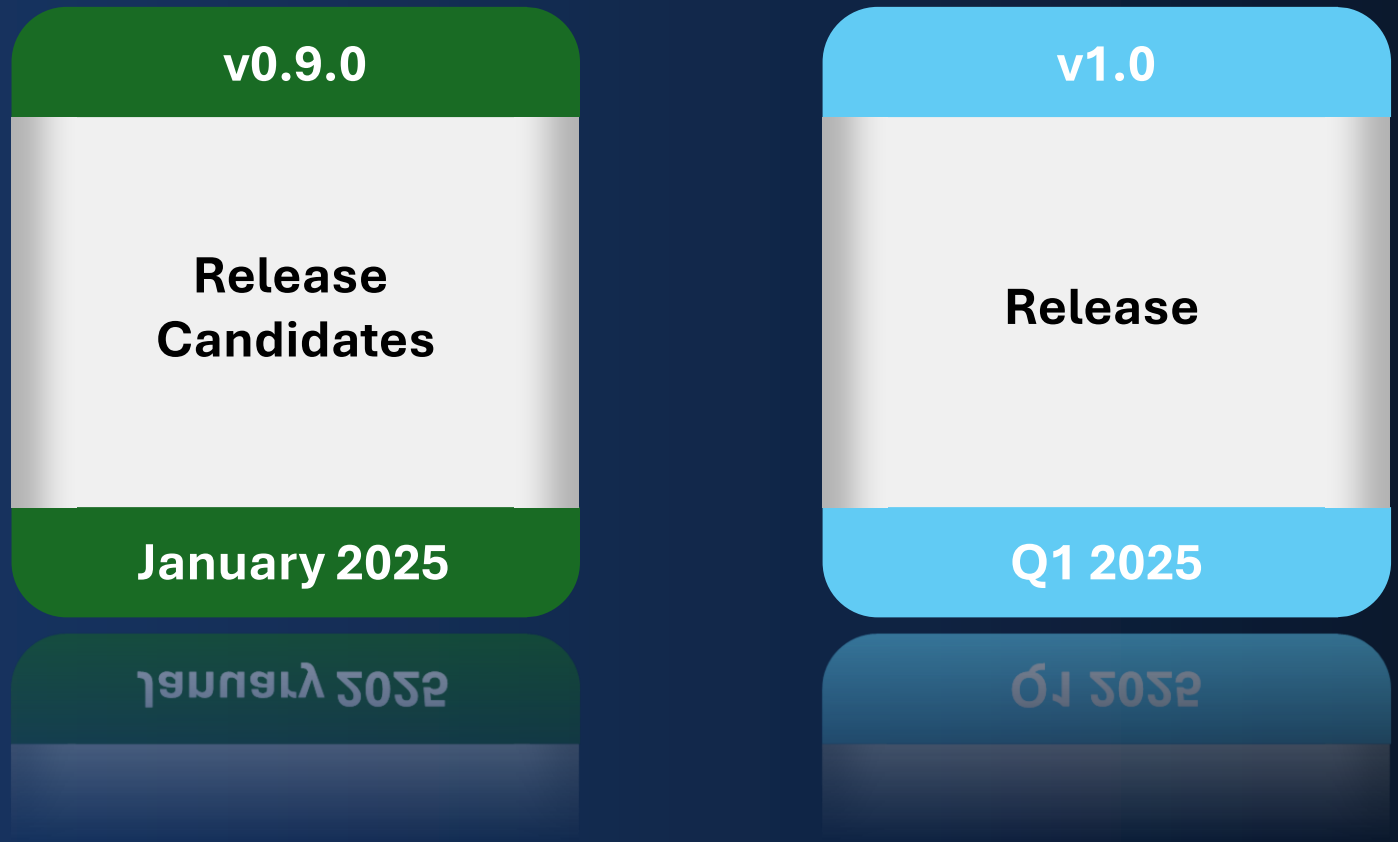
https://rbidou.gitbook.io/pyrasp



WOULD YOU LIKE TO KNOW MORE?


Coming Soon

Roadmap




Wrap-Up

PyRASP


1 Designed for Real Needs
2 Security First
3 Minimal Management
4 Runs in Production

Resources


https://pyrasp.paracyberbellum.io


https://rbidou.gitbook.io/pyrasp


https://pypi.org/project/pyrasp/


@ParaCyberBellum


https://github.com/rbidou/pyrasp


renaud@paracyberbellum.io


https://paracyberbellum.io

OSSIR

Si vis cyber pacem

ParaCyberBellum

PyRASP Project