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
How Novel is Application DoS?

Microservice Primer: High Level View

Architecture

Client Libraries and API Gateway

Circuit Breakers / Failover

Cache
Microservice Primer: Architecture

**GOOD**
- Scale
- Service independence
- Fault isolation
- Eliminates stack debt

**BAD**
- Distributed system complexity
- Deployment complexity
- Cascading service failures if things aren’t set up right
Simplified Microservice API Architecture

EDGE
- INTERNET
- ZUUL PROXY
- WEBSITE
- PROXIES

CORE API

Middle
- Middle Tier Service

Backend
- Backend Tier Service
  - Middle Tier Service
  - Middle Tier Service
  - Middle Tier Service
  - Middle Tier Service
  - Middle Tier Service
  - Middle Tier Service
  - Middle Tier Service
  - Middle Tier Service
  - Middle Tier Service
  - Middle Tier Service
  - Middle Tier Service
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

Often 1 to 1

New School Application DoS

Often 1 to Many
Simple Web Application Architecture

1. Web Browser
2. Web Server
3. Module Handler (based on file extension or framework configuration)
4. Web Application Platform
5. Data Store
Old School Application DoS Attack

```
> perl create_many_profiles.pl
POST /create_profile HTTP/1.1
...
profile_name=$counter + "hacker"
```

HTTP Timeouts

300 requests per second

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

IS6.0

ASP.NET
New School Microservice API DoS

EDGE

ZUUL PROXY

WEBSITE

PROXIES

CORE API

Middle

Middle Tier Service

Middle Tier Service

Middle Tier Service

Middle Tier Service

Backend Tier Service

Backend Tier Service

Backend Tier Service

Backend Tier Service

Backend Tier Service

Backend Tier Service

Backend Tier Service

Backend Tier Service

Backend Tier Service

Backend Tier Service

Middle

POST /recommendations HTTP/1.1
...
{"recommendations": {"range": [0,10000]}}
New School Microservice API DoS

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Method</th>
<th>Status</th>
<th>Type</th>
<th>Initiator</th>
<th>Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>browse</td>
<td>GET</td>
<td>200</td>
<td>document</td>
<td></td>
<td>50.4 KB</td>
<td>2.34s</td>
</tr>
<tr>
<td>pathEvaluator?withSize=true&amp;materialize=true&amp;model=harris</td>
<td>POST</td>
<td>200</td>
<td>xhr</td>
<td>common\bootstrap\i...</td>
<td>7.5 KB</td>
<td>1.85s</td>
</tr>
<tr>
<td>pathEvaluator?withSize=true&amp;materialize=true&amp;model=harris</td>
<td>POST</td>
<td>200</td>
<td>xhr</td>
<td>common\bootstrap\i...</td>
<td>9.7 KB</td>
<td>1.19s</td>
</tr>
<tr>
<td>pathEvaluator?withSize=true&amp;materialize=true&amp;model=harris</td>
<td>POST</td>
<td>200</td>
<td>xhr</td>
<td>common\bootstrap\i...</td>
<td>7.5 KB</td>
<td>1.19s</td>
</tr>
<tr>
<td>preflight?batchImages=true&amp;iclomoid=le6caco9-b2fa-4be4-9766-008cc537958_ROOT&amp;fromRow=4&amp;toRow=50...</td>
<td>GET</td>
<td>200</td>
<td>xhr</td>
<td>common\bootstrap\i...</td>
<td>29.8 KB</td>
<td>1.02s</td>
</tr>
<tr>
<td>pathEvaluator?withSize=true&amp;materialize=true&amp;model=harris</td>
<td>POST</td>
<td>200</td>
<td>xhr</td>
<td>common\bootstrap\i...</td>
<td>12.3 KB</td>
<td>977ms</td>
</tr>
<tr>
<td>pathEvaluator?withSize=true&amp;materialize=true&amp;model=harris</td>
<td>POST</td>
<td>200</td>
<td>xhr</td>
<td>common\bootstrap\i...</td>
<td>7.4 KB</td>
<td>769ms</td>
</tr>
<tr>
<td>login</td>
<td>POST</td>
<td>302</td>
<td>text/html</td>
<td>Other</td>
<td>2.6 KB</td>
<td>527ms</td>
</tr>
<tr>
<td>manifest</td>
<td>POST</td>
<td>200</td>
<td>xhr</td>
<td>cadmium\piever...</td>
<td>95.4 KB</td>
<td>512ms</td>
</tr>
<tr>
<td>manifest</td>
<td>POST</td>
<td>200</td>
<td>xhr</td>
<td>cadmium\piever...</td>
<td>93.5 KB</td>
<td>492ms</td>
</tr>
</tbody>
</table>
## Identifying Latent Service Calls

<table>
<thead>
<tr>
<th>Service</th>
<th>RPS</th>
<th>Circuit Breakers Open %</th>
<th>Error %</th>
<th>Success %</th>
<th>Failure %</th>
<th>Short Circuited %</th>
<th>Timeout %</th>
<th>Rejection %</th>
<th>Cache Responses</th>
<th>Thread Group</th>
<th>Isolation Strategy</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6.3</td>
<td>0.0</td>
<td>1.6</td>
<td>98.4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.6</td>
<td>0</td>
<td>THREAD</td>
<td></td>
<td>2634.2</td>
</tr>
<tr>
<td></td>
<td>130.7</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>THREAD</td>
<td></td>
<td>1963.5</td>
</tr>
<tr>
<td></td>
<td>2447.6</td>
<td>0.0</td>
<td>0.07</td>
<td>99.9</td>
<td>0.06</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.6</td>
<td>THREAD</td>
<td></td>
<td>218.0</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>THREAD</td>
<td></td>
<td>1135.1</td>
</tr>
<tr>
<td></td>
<td>1.7</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>THREAD</td>
<td></td>
<td>293.2</td>
</tr>
<tr>
<td></td>
<td>17.1</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>THREAD</td>
<td></td>
<td>1111.0</td>
</tr>
<tr>
<td></td>
<td>8.7</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>THREAD</td>
<td></td>
<td>218.5</td>
</tr>
<tr>
<td></td>
<td>11.4</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>THREAD</td>
<td></td>
<td>386.4</td>
</tr>
</tbody>
</table>

### Additional Tables

#### Cache Responses

<table>
<thead>
<tr>
<th>RPS</th>
<th>Cache Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>140.7</td>
<td>0</td>
</tr>
<tr>
<td>16.3</td>
<td>0</td>
</tr>
<tr>
<td>38.7</td>
<td>0</td>
</tr>
<tr>
<td>11.4</td>
<td>0</td>
</tr>
<tr>
<td>0.8</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Latency (ms)

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>2131.0</td>
</tr>
<tr>
<td>90%</td>
<td>1290.1</td>
</tr>
<tr>
<td>90%</td>
<td>957.9</td>
</tr>
<tr>
<td>90%</td>
<td>677.2</td>
</tr>
<tr>
<td>90%</td>
<td>396.9</td>
</tr>
</tbody>
</table>
Microservice Application DoS: Attack Patterns

Range
Object Out per Object in
Request Size
All of the Above
Application DoS Technique: Range

```json
{
    "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

```json
{
    "customizations": [
        "messages", 80017537,
        "contact", "synopsis", "brief",
        "logdata"
    ]
}
```
Application DoS Technique: Request Size

```json
{
  "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_posibly"
}
```
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?
Logical Work Per Request

Rate Limited

Service Impact

Service Impact

Service Auto-Scaling/Healthy

# Req

Logical Work Per Request
New School Application DoS Attack: Case Study

<table>
<thead>
<tr>
<th>Type</th>
<th>Status report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>Maximum Paths Per Request Exceeded</td>
</tr>
<tr>
<td>Description</td>
<td>The request entity is larger than the server is willing or able to process.</td>
</tr>
</tbody>
</table>

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

```json
{
  "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": "05745d1c11d19b49d7c0223fa050d59c0c2d3c5",
  "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
Repulsive Grizzly: Single Node
Cloudy Kraken Overview
Update Config
Push the latest configuration file and attack scripts to S3.
Reset the DynamoDB state.

Build Environment
Configure the VPCs in each region

Start up Attack Nodes
Launch instances

Collect data
Wait for data to come through SNS

Tear-Down
Tear down and reset the environment in each region
Cloudy Kraken Configuration

- Zip File and Configuration
- Reset State

S3 Bucket

DynamoDB Table
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
Results of Test

80% Error Rate
$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