Serverless Security: What's Left to Protect?

Guy Podjarny, Snyk
@guypod

About Me

- Guy Podjarny, **@guypod** on Twitter
- CEO & Co-founder at Snyk
- History:
  - Cyber Security part of Israel Defense Forces
  - First Web App Firewall (AppShield), Dynamic/Static Tester (AppScan)
  - **Security**: Worked in Sanctum -> Watchfire -> IBM
  - **Performance**: Founded Blaze -> CTO @Akamai
- O'Reilly author, speaker

Serverless Changes Security

Serverless Is…

Better for some security concerns

Neutral for some security concerns

Worse for some security concerns

When is Serverless Better for Security?

Serverless has no Servers!

(that you manage and secure)

Less need to worry about…

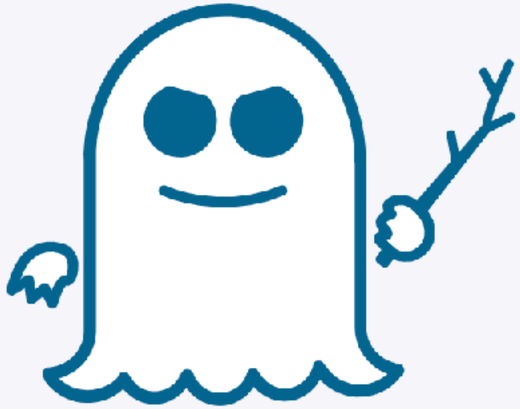
Vulnerable OS Dependencies

"patching" your servers

Heartbleed

Shellshock

Meltdown



Spectre

Verizon:

“Most attacks exploit known vulnerabilities that have never been patched despite patches being available for months, or even years”

Symantec:

"Through 2020, **99% of vulnerabilities exploited** will continue to be ones known by security and IT professionals for **at least one year**"

With Serverless, it’s the platform’s responsibility

And it’s the platform operators core competency

FaaS ~= PaaS

FaaS naturally relies less on OS dependencies

Less need to worry about…

Denial of Service

DoS means a stream of specific “heavy” requests take down server(s) (e.g. ReDoS).

In FaaS, there is no long standing server to take down

Auto Scale FTW!

Deal with bad demand just like dealing with good demand

Caveat:
Concurrent execution limit

AWS Lambda default limit 1000 concurrent function executions pre region

Caveat:
Resource exhaustion

Back-end resources are often not equally elastic

Caveat:
Limited in size

100Gbps attack will still take you down…
Consider a DDoS protection solution too.

Caveat:
Your bill

You still pay for extra traffic…

Less need to worry about…

Long-lived
Compromised Servers

Attackers penetrate in stages

Attackers like to lay in wait

FaaS forces Statelessness - including bad state

Attackers need to repeat attacks many times, risking detection & remediation

Caveat:
Containers are often reused

A compromised function may still compromise other users

FaaS > PaaS

PaaS Servers are managed, but still live longer

Security in Serverless

Better

- Vulnerable OS Dependencies
- Denial of Service
- Long-lived Compromised Servers

So attackers will go…

Right?

When is Serverless Neutral for Security?

Still need to worry about…

Permissions

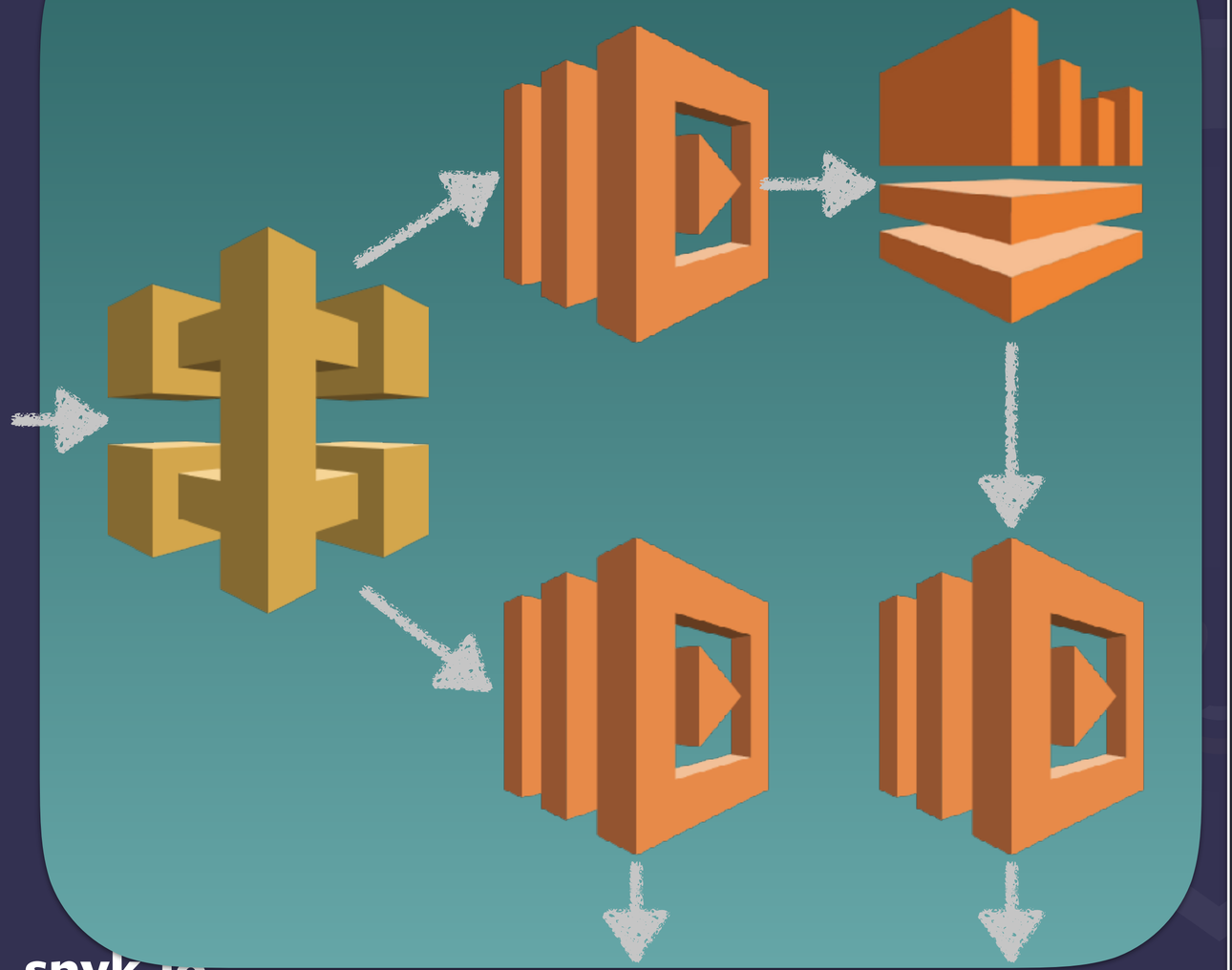
and authorization

Who can invoke your function?

Who can access code & env vars for your function?

If your function was compromised, what can it do?

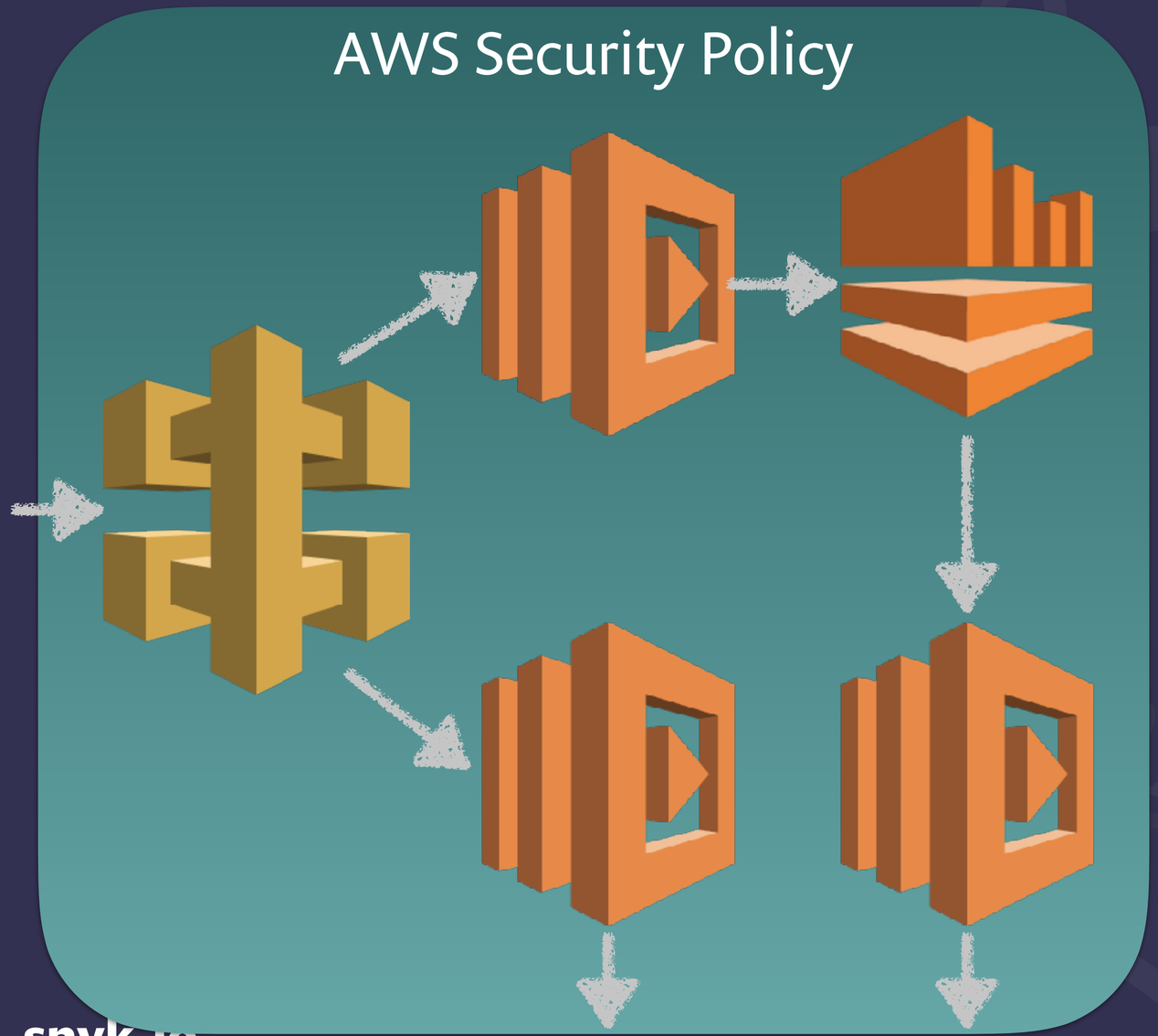
AWS Security Policy



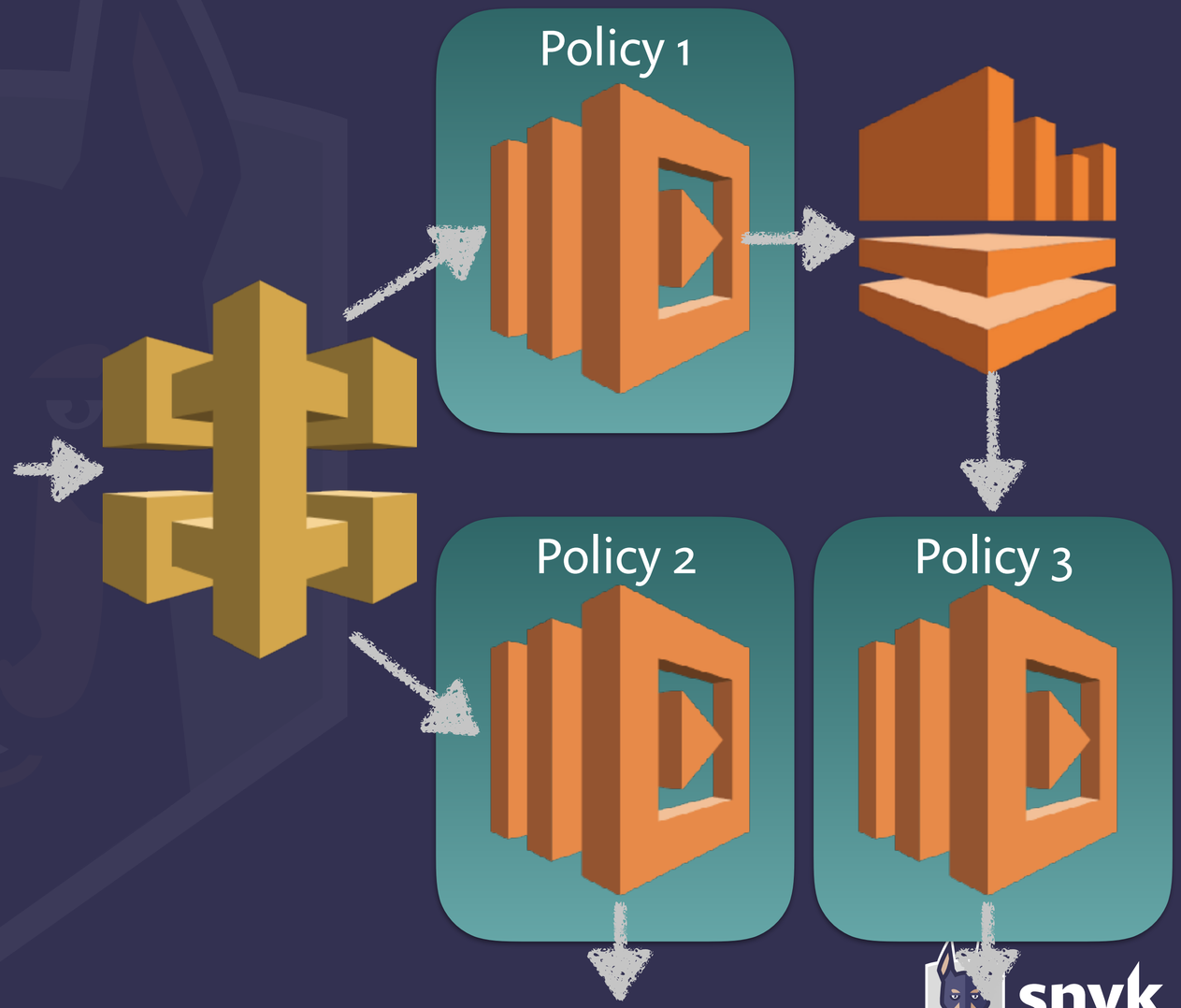
```
functions:
  create:
    handler: todos/create.create
    events:
      - http:
          path: todos
          method: post
          cors: true

  list:
    handler: todos/list.list
    events:
      - http:
          path: todos
          method: get
          cors: true

  get:
    handler: todos/get.get
    events:
      - http:
          path: todos/{id}
          method: get
          cors: true

  update:
    handler: todos/update.update
```

```
iamRoleStatements:
  - Effect: Allow
    Action:
      - dynamodb:Query
      - dynamodb:Scan
      - dynamodb:GetItem
      - dynamodb:PutItem
      - dynamodb:UpdateItem
      - dynamodb:DeleteItem
    Resource: "arn:aws:dyna
  - Effect: Allow
    Action:
      - lambda:InvokeFuncti
      - lambda:InvokeAsync
    Resource: "*"
```

Easier



Safer



Keep your policies granular

iRobot automates IAM roles

https://www.youtube.com/watch?v=BVVHzCd8APw

Use Least Privilege Principle

Policies continuously expand until somebody adds an asterix

Adding permissions is easy.
Removing permissions is hard

Caveat:
FaaS permissions are limited to
Platform actions

PureSec FunctionShield

Disable outbound internet connectivity (except for AWS resources) from the serverless runtime environment, if such connections are not required

Disable read/write on the /tmp/ directory, if such operations are not required

Disable child process execution, if such execution is not required by the function

Disable read access to the function's handler and prevent source code leakage

```
var AWS = require('aws-sdk');
const FunctionShield = require('@puresec/function-shield');
FunctionShield.configure({
    policy: {
        // 'block' mode ⇒ active blocking
        // 'alert' mode ⇒ log only
        // 'allow' mode ⇒ allowed, implicitly occurs if key does not exist
        outbound_connectivity: "block",
        read_write_tmp: "block",
        create_child_process: "block" },
    token: process.env.FUNCTION_SHIELD_TOKEN });

exports.hello = async (event) ⇒ {
    // ... // your code
};
```

Serverless is *a bit* better for authorization

if managed properly, which it often isn't

Still need to worry about…

Securing Data (at rest)

FaaS apps still store data

and that data can be stolen or tampered with

Bad:
FaaS means more state
outside the server

can no longer keep sensitive data “on the machine”

Good:
FaaS allows more granular data access controls

Control which functions can access which data

Tips & Tricks

- Encrypt all sensitive persistent data
- Encrypt all sensitive off-box state data
- Minimize functions that can access each data store
- Use separate DB credentials per function
  - And control what these credentials should do
- Monitor which functions are accessing which data

Still need to worry about…

Vulnerabilities in Your Code

Serverless doesn’t protect the Application Layer

SQL Injection
Cross Site Scripting
Remote Command Execution
Cross Site Request Forgery
Bad auth logic
...

OWASP Top 10 - 2013		OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

App Sec Tips

- Dynamic App Sec Testing
- Static App Sec Testing
- Standardize input processing to include sanitization
  - Use shared libraries across functions
- Make API Gateway models as strict as possible
- Secure each function independently
  - Secure unit tests FTW!

Still need to worry about…

Vulnerable Application Dependencies

Serverless functions use Application Dependencies

npm, Maven, PyPI, nuget…

For many functions, the majority of code is dependencies

3rd party code can hold vulnerabilities just like 1st party code

Example: Fetch file & store in s3 (Serverless Framework Example)

```
'use strict';

const fetch = require('node-fetch');
const AWS = require('aws-sdk'); // eslint-disable-line import/no-extraneous-dependencies

const s3 = new AWS.S3();

module.exports.save = (event, context, callback) => {
  fetch(event.image_url)
    .then((response) => {
      if (response.ok) {
        return response;
      }
      return Promise.reject(new Error(
            `Failed to fetch ${response.url}: ${response.status} ${response.statusText}`));
    })
    .then(response => response.buffer())
    .then(buffer => (
      s3.putObject({
        Bucket: process.env.BUCKET,
        Key: event.key,
        Body: buffer,
      }).promise()
    ))
    .then(v => callback(null, v), callback);
};
```

19 Lines of Code

```
"dependencies": {
  "aws-sdk": "^2.7.9",
  "node-fetch": "^1.6.3"
}
```

2 Direct dependencies

19 dependencies (incl. indirect)

191,155 Lines of Code

More code ~=
More Vulnerabilities

Over time, dependencies grow stale & vulnerable

EQUIFAX DATA BREACH

Equifax's Mega-Breach Was Made Possible by a Website Flaw It Could Have Fixed

Security

Equifax's disastrous Struts patching blunder: THOUSANDS of other orgs did it too

Those are just the ones known to have downloaded outdated versions

Failure to patch two-month-old bug led to massive Equifax breach

Critical Apache Struts bug was fixed in March. In May, it bit ~143 million US consumers.

DAN GOODIN - 9/13/2017, 11:12 PM

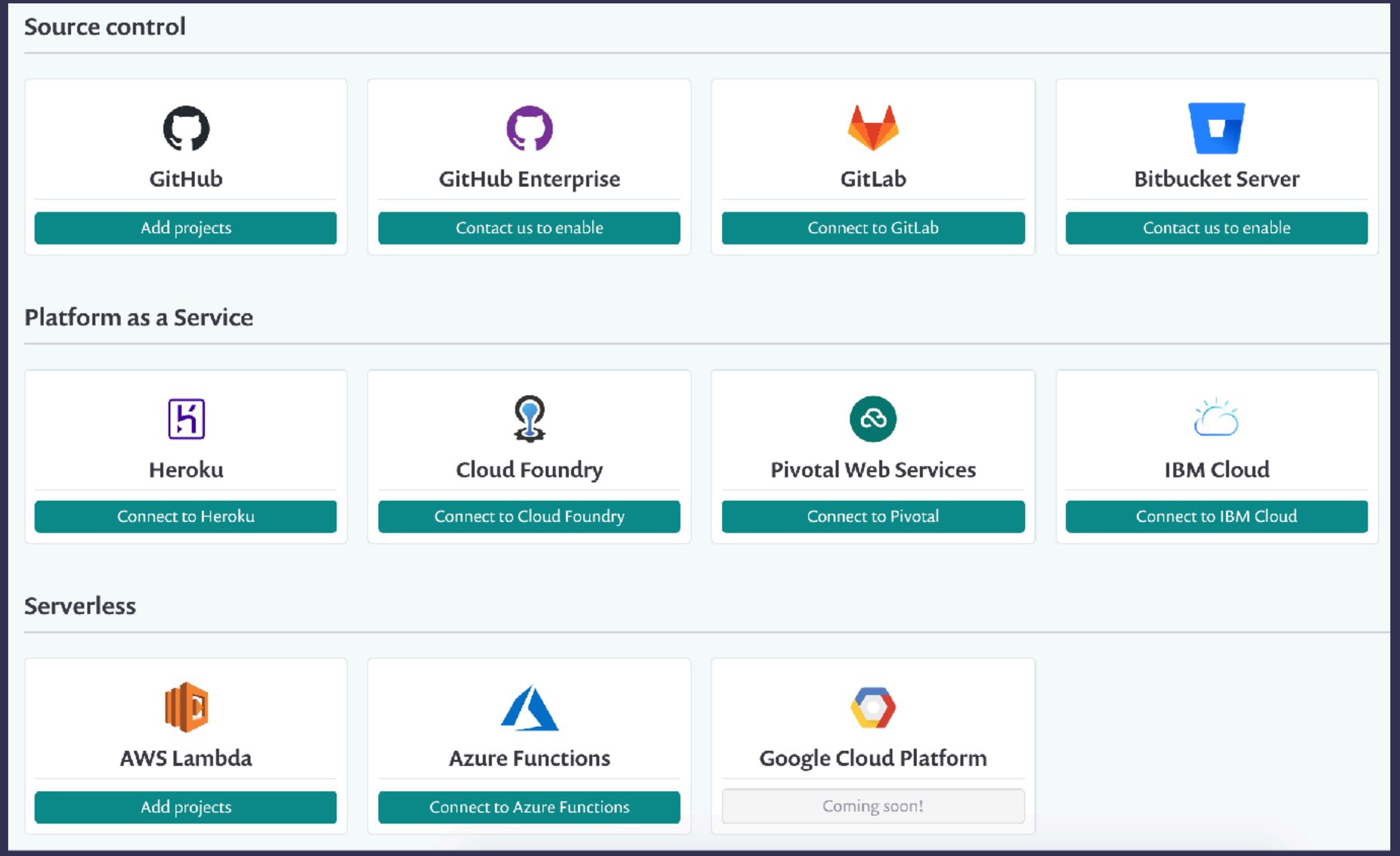
The platform manages
OS Dependencies.

You need to manage
App Dependencies.

<Shameless Plug>

Snyk for Serverless!

(and PaaS)

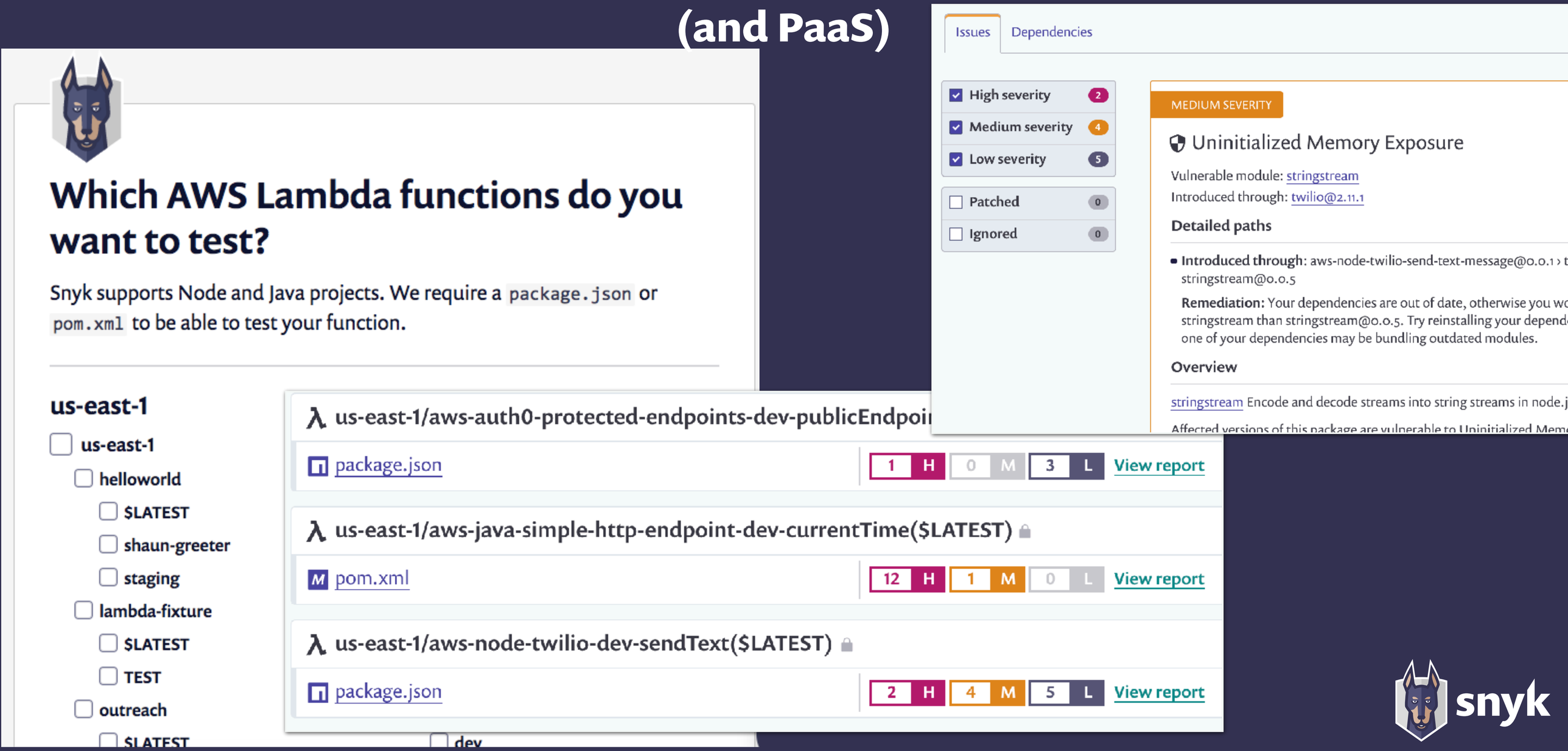


snyk.io

snyk

Snyk for Serverless!

(and PaaS)



snyk

Fix during Dev

Open a fix PR

github.com/guypod/goof

View test report

1. Choose which vulnerabilities you would like to fix.
2. Open a pull request with the upgrades and patches to address these vulnerabilities
3. The repository will be watched for new vulnerabilities and you will receive an a
   vulnerability affects your project.

Vulnerabilities with a fix

An upgrade or patch is available to fix the vulnerable dependencies.

- [x] L **Regular Expression Denial of Service (DoS)** in hawk
- [x] H **Content & Code Injection (XSS)** in marked
- [x] H **Cross-site Scripting (XSS) via Data URIs** in marked
- [x] M **Remote Memory Exposure** in mongoose

[Snyk Alert] Fix for 4 vulnerable dependency paths #9872

Merged **BerkeleyTrue** merged 1 commit into `staging` from `snyk-fix-9a4b3f6d` on Jul 25, 2016

Conversation 1 · Commits 1 · Files changed 1

snyk-bot commented on Jul 24, 2016 — Contributor

The following newly disclosed vulnerabilities impact one or more of the npm packages this project uses:

- npm:minimatch:20160620

As these vulnerabilities are now publicly known, attackers can try to use them against your application, making fixing them a matter of urgency.

To help expedite the fix, Snyk created this pull request with the necessary changes to address the vulnerabilities.

This pull request includes:

Reviewer
No review

Assignee
No one as

Labels
None yet

Projects

snyk.io

Snyk

</Shameless Plug>

Security in Serverless

Better

- Vulnerable OS Dependencies
- Denial of Service
- Long-lived Compromised Servers

Neutral

- Permissions
- Securing Data at rest
- Vulnerabilities in your code
- Vulnerable App Dependencies

Neutral ~= Worse

By eliminating some threats,
Serverless shifts attacker attention to what's left

When is Serverless Worse for Security?

Serverless doesn’t create new security problems.

Serverless means more…

Independent Services
Flexible Interfaces
Use of 3rd party services

including the risks that entails!

More need to worry about…

Third Party Services

and securing data in transit

For each service, worry about…

What data are you sharing?

and how well does the other service manage it?

For each service, worry about…

Is data in transit secured?

Is it using HTTPS? Is it within a VPC? Is it encrypted?

For each service, worry about…

Who are you talking to?

You use an API key, but how do you authenticate them?
Validate HTTPS cert, especially when exiting your network

For each service, worry about…

Do you trust its responses?

If the other service is compromised, can it be used to get to you?

For each service, worry about…

How to store API keys?

Be sure to use a KMS and rotate keys!

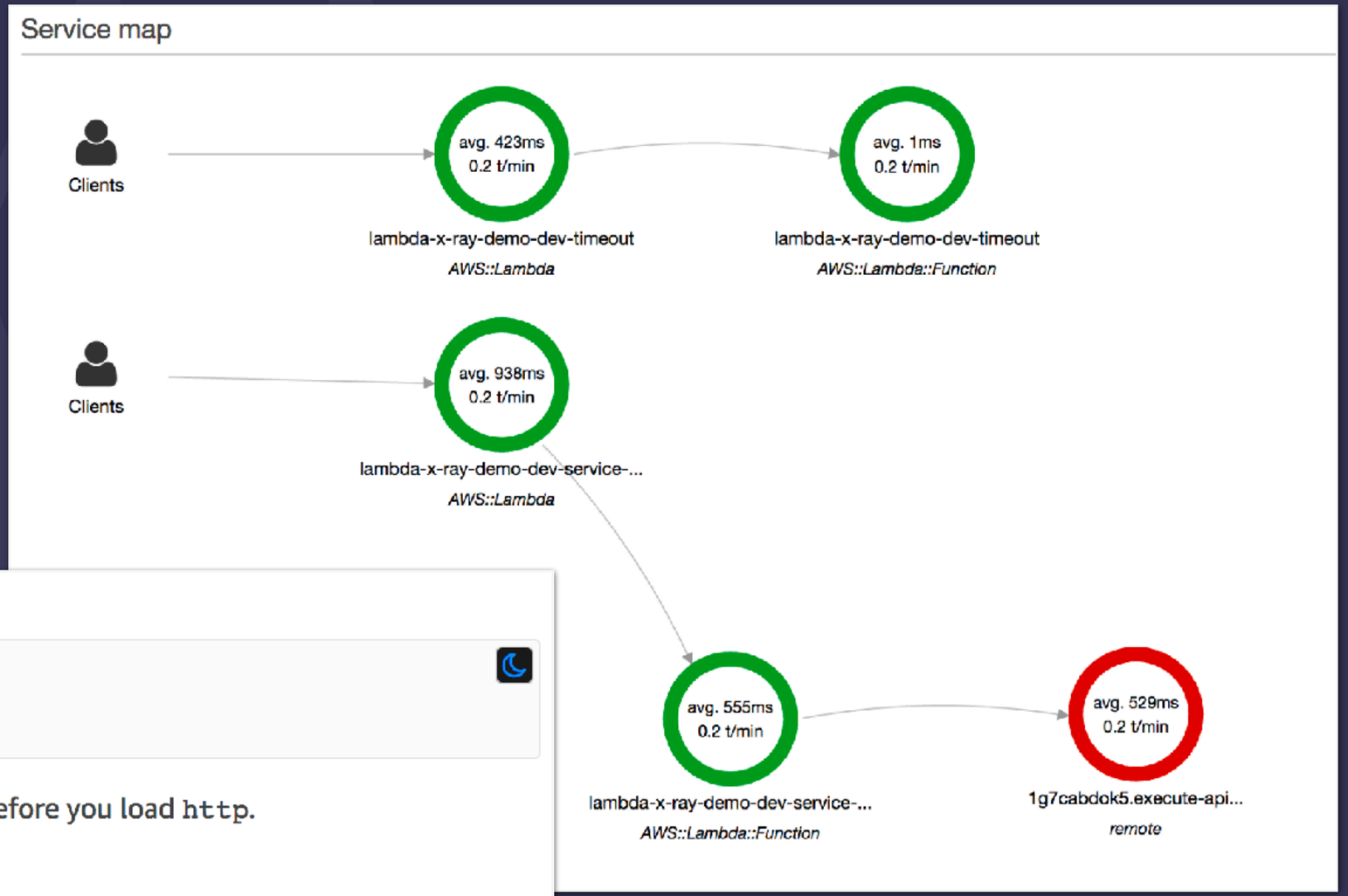
For each service, worry about…

- What data are you sharing?
- Is data in transit secured?
- Who are you talking to?
- Do you trust its responses?
- How to store API keys?

Worry about 1st party services too!

Don’t let your least secure function take down the system

Use AWS X-Ray for visibility



Example app.js - HTTP Client

```
var AWSXRay = require('aws-xray-sdk');
var http = AWSXRay.captureHTTPs(require('http'));
```

To enable tracing on all HTTP clients, call `captureHTTPsGlobal` before you load `http`.

Example app.js - HTTP Client (Global)

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.captureHTTPsGlobal(require('http'));
var http = require('http');
```

Image via Yan Cui's blog

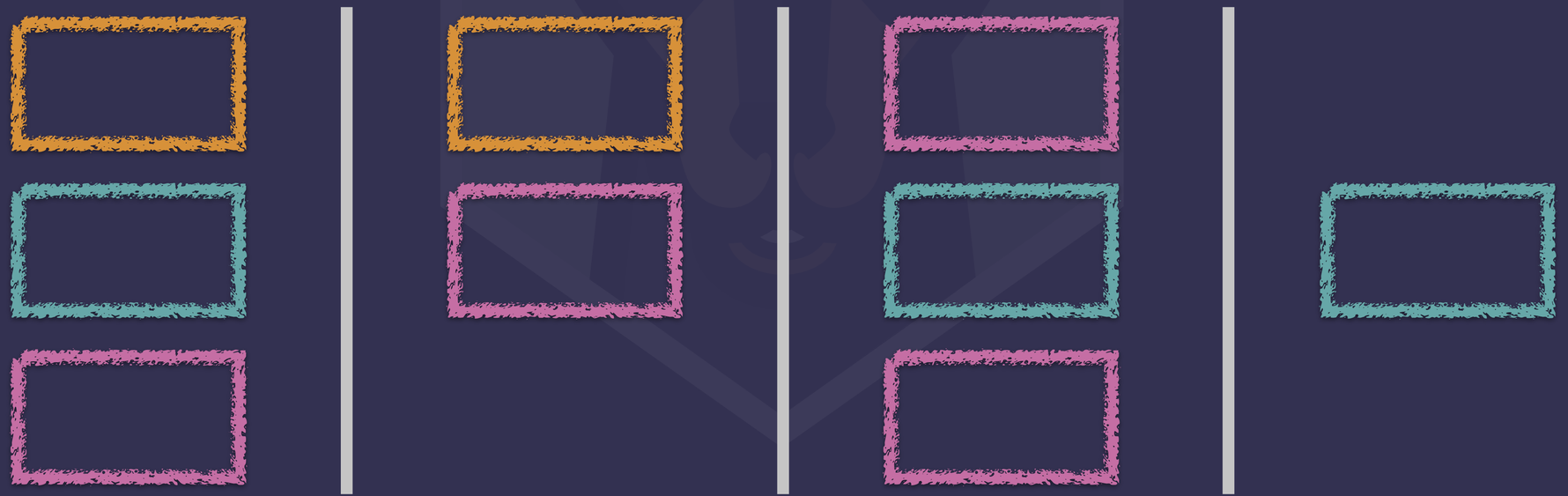
snyk

More need to worry about…

Attack Surface

mo' functions, mo' problems

Serverless -> Granularity -> Flexibility

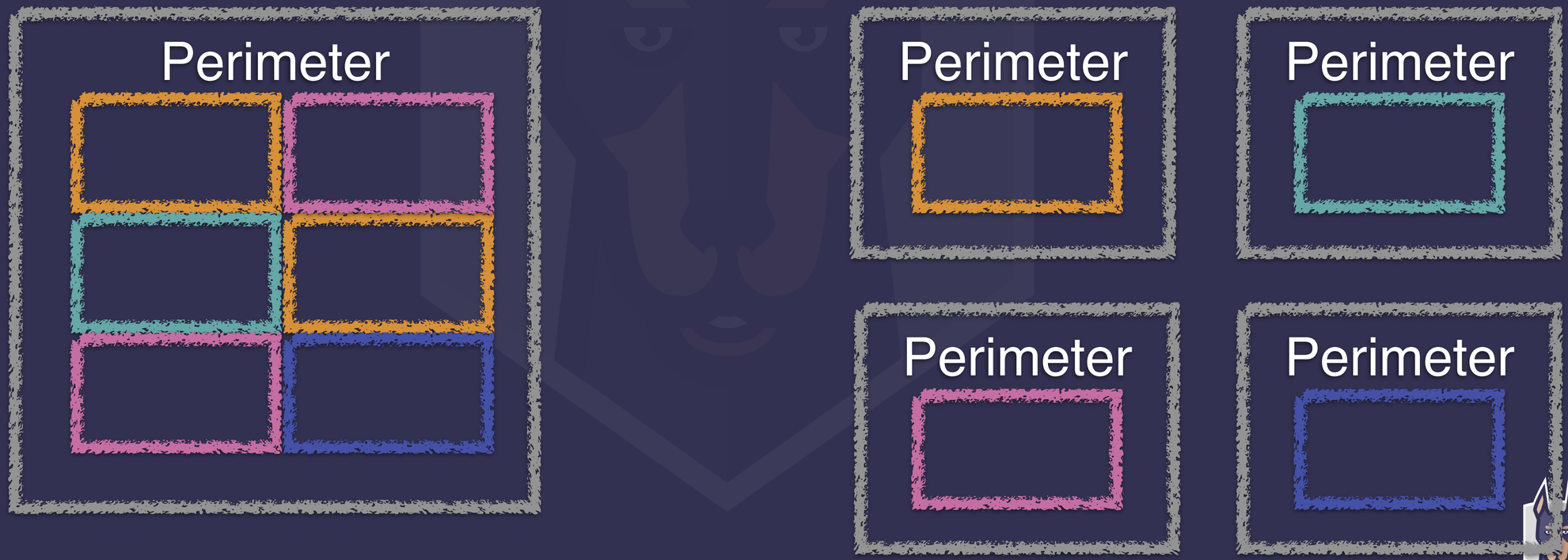


Flexibility -> Risk

Harder to define what's "right", more use-cases to abuse

A function is a perimeter

That needs to be secured



Tips & Tricks

- Test every function for security flaws, independently
- Don’t rely on limiting access to a function
  - Access controls will change over time, without code changes
- Use shared input/output processing libraries
  - Make it easier to process input securely than insecurely
- Limit functionality to what you actually need
  - Sometimes you need to work more to let functions do less
- Monitor both individual functions and full flows

More need to worry about…

Security Monitoring

mo' functions, mo' problems

Why NOT deploy a function?

Super easy, (effectively) zero cost

Easy deployment + min cost = LOTS of functions

Many of which are lightly used

Easy to add,
Hard to remove

Policies tend to expand...

Expanding is easy, contracting is hard

LOTS of functions,
many of which are lightly used,
with overly open policies…

our future?

Every Function introduces Risk

No ops cost != No cost of ownership

Risk & Management costs still exist

Tips & Tricks

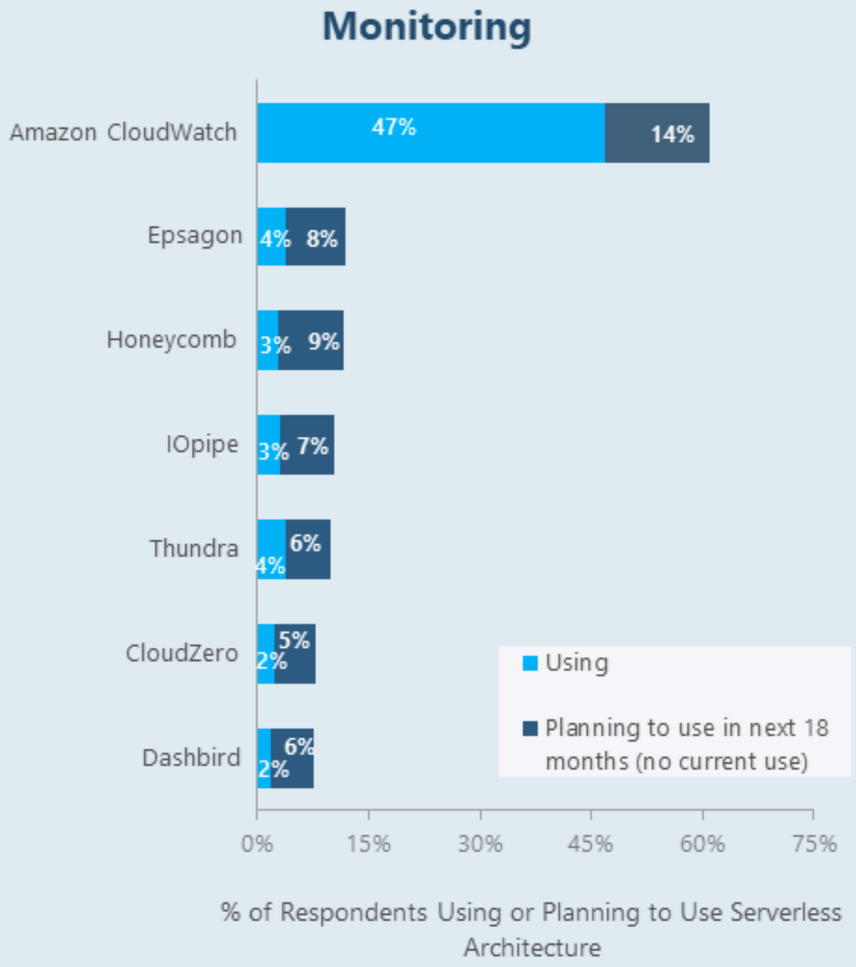
- Consider before you deploy. Do you need this?
- Separate networks/accounts for groups of functions
- Track what you have deployed, and how it's used
- Minimize permissions up front
- Chaos-style reduce permissions and see what breaks
- **Monitor for known vulnerabilities** in functions

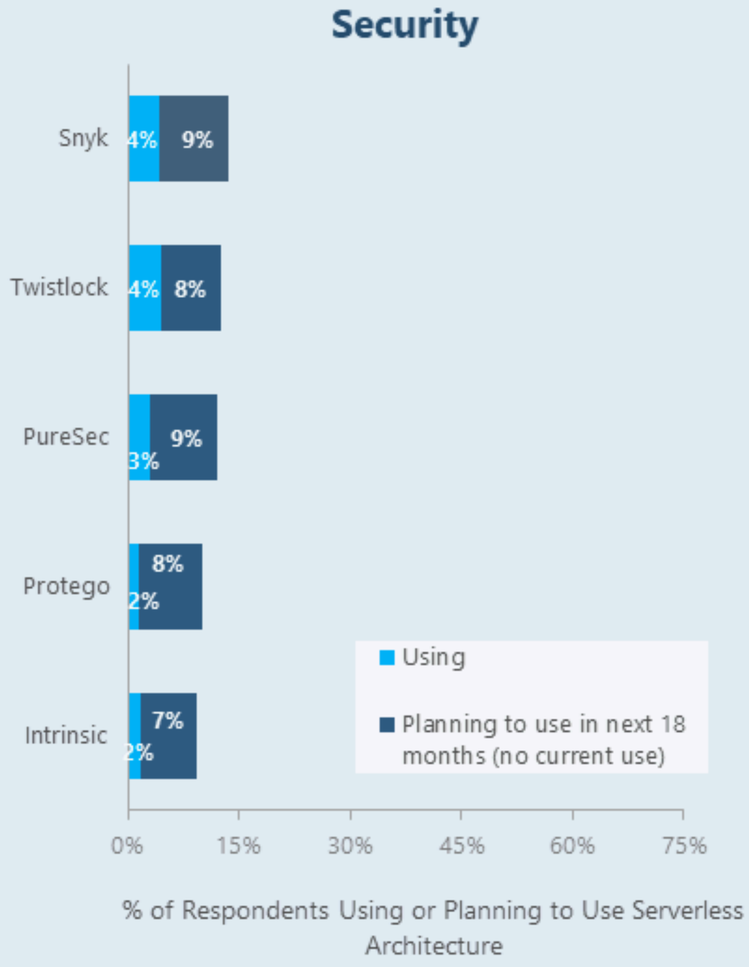
Security in Serverless

Better	Neutral	Worse
Vulnerable OS Dependencies	Permissions	Third Party Services
Denial of Service	Securing Data at rest	Attack Surface
Long-lived Compromised Servers	Vulnerabilities in your code	Security Monitoring
	Vulnerable App Dependencies	

Tools?

Serverless Monitoring and Security in Use or Planned for Use





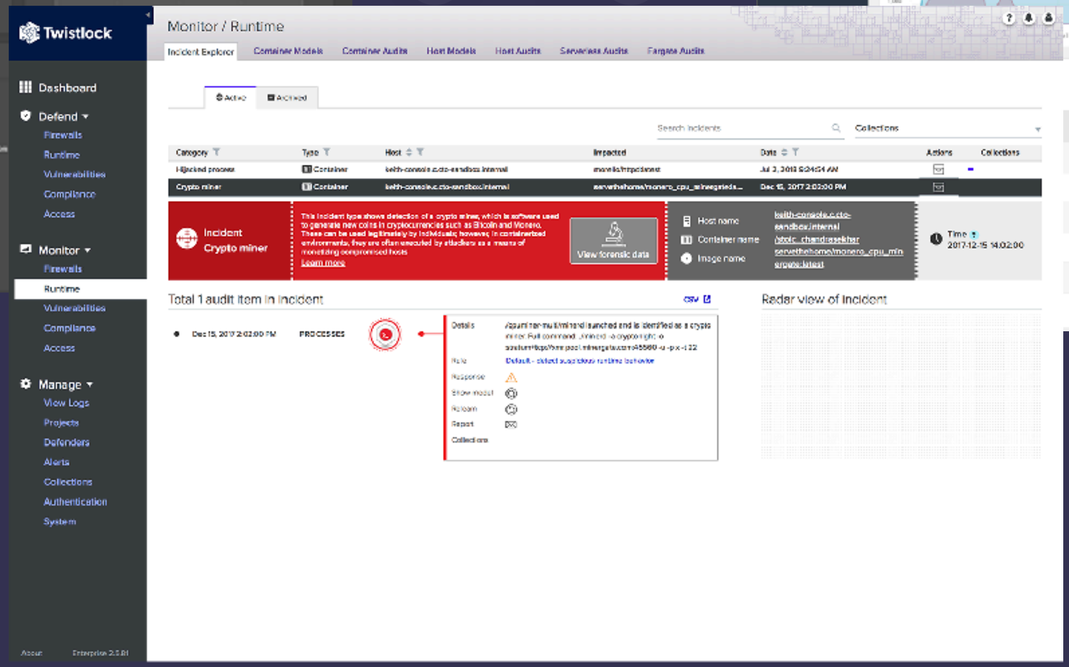
Source: The New Stack Serverless Survey 2018. Q. Please indicate which of the following your organization is using or planning to use within the next 18 months. n=382

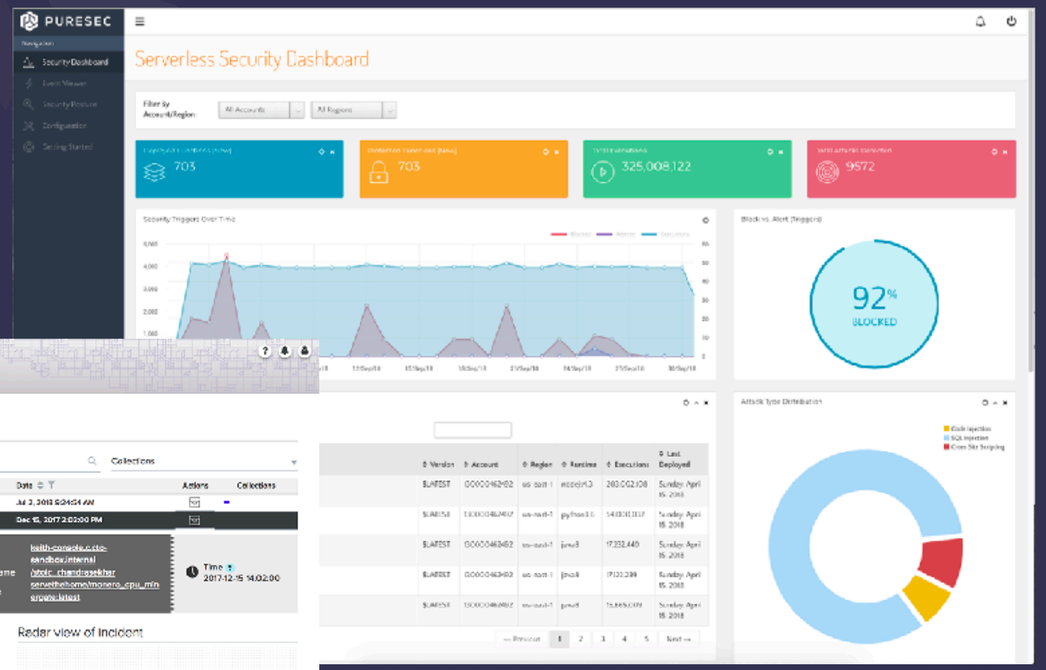
https://thenewstack.io/serverless-roadmaps-monitoring-security-frameworks-tools/

Monitor attacks in the SOCC

(Security Operation Control Center)

Protego



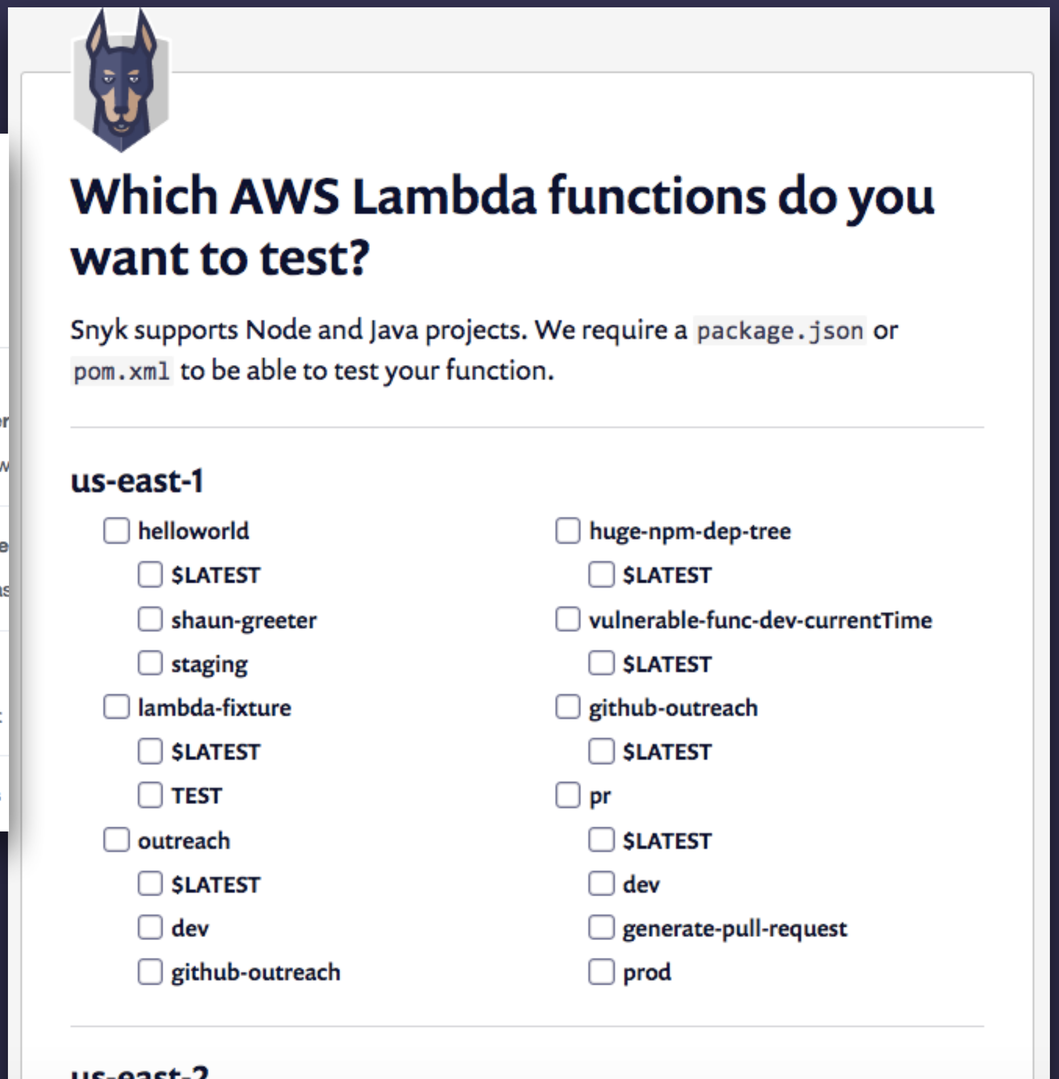


PureSec

snyk.io

snyk

Monitor & Fix vulnerable libs in Dev & CI/CD



snyk.io

Summary

Serverless dramatically reduces some top security threats

But...

Security is hard -
and attackers don’t give up

Serverless shuffles security priorities

Previously easy attacks are now hard.
Attackers will move on to the next item on the list

Security in Serverless

Better	Neutral	Worse
Vulnerable OS Dependencies	Permissions	Third Party Services
Denial of Service	Securing Data at rest	Attack Surface
Long-lived Compromised Servers	Vulnerabilities in your code	Security Monitoring
	Vulnerable App Dependencies	

Serverless is defined now.
Let's build Security in.

Thank You!

Guy Podjarny, Snyk
@guypod