What is AppSec?

… in organization

Jan Masarik

Whoami

- FI MUNI graduate (2019)
- (ex) AppSec Lead @ Kiwi.com
- OWASP Czech Chapter Lead
- Co-founder of TunaSec.cz
- Fan of CTF/bug bounty

Disclaimer

- We will focus on **web applications**, and we'll go **broad.**
- Most of the principles can be applied everywhere, but will be showcased on the domain of web security.
- Doing this presentation because I missed such overview in my studies, so had to learn it *the hard way*.

Role of an AppSec team?

Role of an AppSec team?

Keep the **App**lication code **Sec**ure *enough*

How to achieve *secure enough* code?

Technical measures

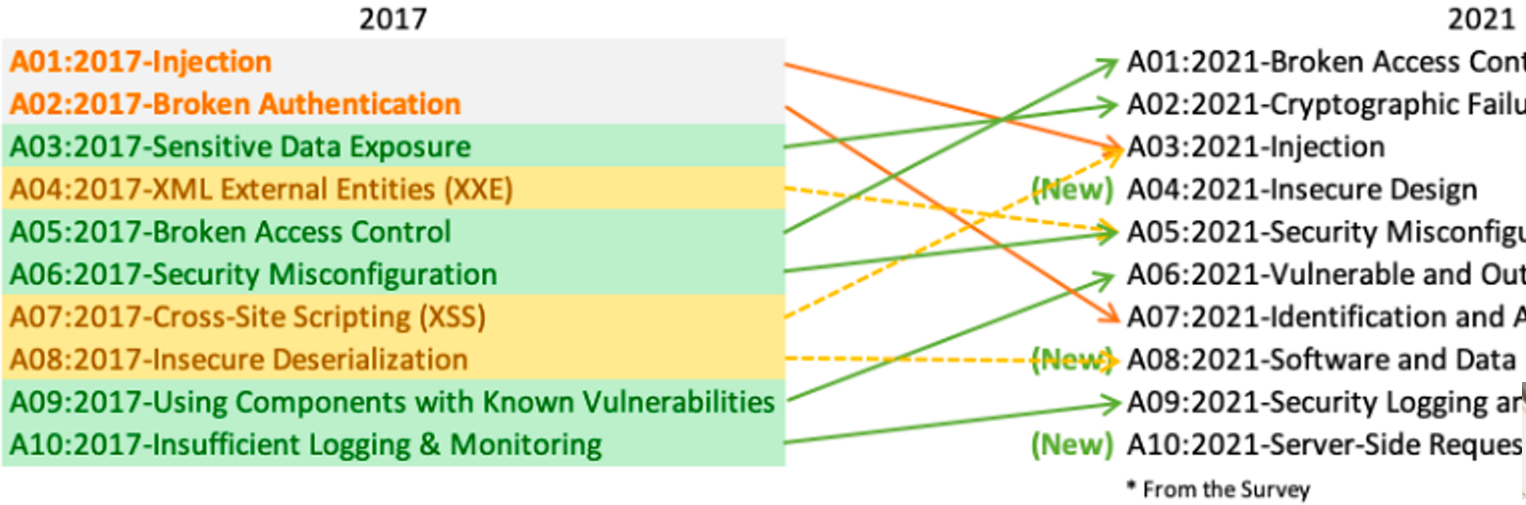
- Secure design/code review
- Dependency management
- Secrets detection
- Static analysis (SAST)
- Dynamic analysis (DAST)
- Penetration tests
- Bug bounty

***Soft* measures**

- Security champions
- Education (workshops, wikis)
- Security aware culture

OWASP Top 10

- 8 are based on vulnerability data
- 2 based on survey sent to community to *catch up with most recent trends*



2017		2021
A01:2017-Injection		A01:2021-Broken Access Control
A02:2017-Broken Authentication		A02:2021-Cryptographic Failures
A03:2017-Sensitive Data Exposure		A03:2021-Injection
A04:2017-XML External Entities (XXE)	(New)	A04:2021-Insecure Design
A05:2017-Broken Access Control		A05:2021-Security Misconfiguration
A06:2017-Security Misconfiguration		A06:2021-Vulnerable and Outdated Components
A07:2017-Cross-Site Scripting (XSS)		A07:2021-Identification and Authentication Failures
A08:2017-Insecure Deserialization	(New)	A08:2021-Software and Data Integrity Failures
A09:2017-Using Components with Known Vulnerabilities		A09:2021-Security Logging ar
A10:2017-Insufficient Logging & Monitoring	(New)	A10:2021-Server-Side Reques

* From the Survey

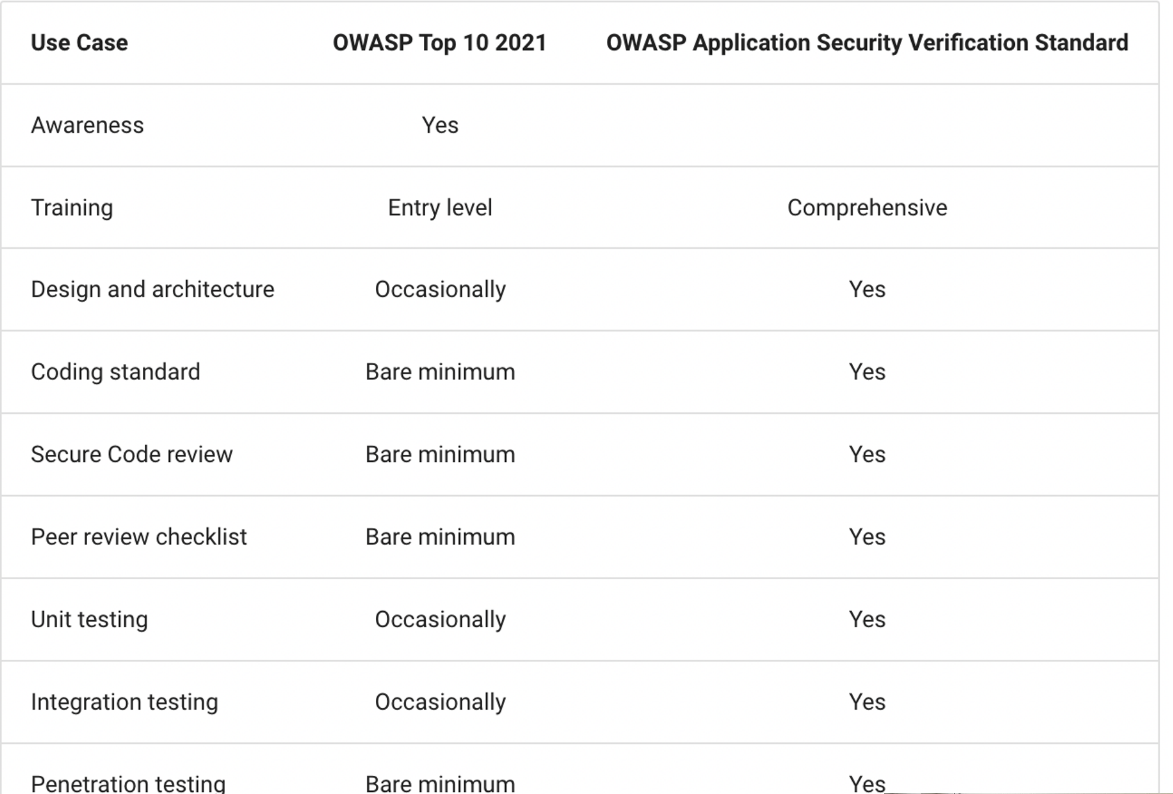
https://owasp.org/Top10/

OWASP Top 10

- **40**+ data submissions from AppSec companies (HackerOne, Veracode, …)
- Covering data from **500 000**+ real-world applications and APIs
- Primary goal is education of developers or managers
  - It’s just top list of 10 things with which you can avoid _80%*_ of problems
  - Not trying to be an exhaustive list, but it’s the best place to start!
- New version every 4 years (most recent in 2021)
- Originally only for web applications, now also versions for:
  - Serverless (2019)
  - Mobile (2016)
  - API (2019)

*80/20 rule, **n**

OWASP Top 10 - When to use?



Use Case	OWASP Top 10 2021	OWASP Application Security Verification Standard
Awareness	Yes	
Training	Entry level	Comprehensive
Design and architecture	Occasionally	Yes
Coding standard	Bare minimum	Yes
Secure Code review	Bare minimum	Yes
Peer review checklist	Bare minimum	Yes
Unit testing	Occasionally	Yes
Integration testing	Occasionally	Yes
Penetration testing	Bare minimum	Yes
Tool support	Bare minimum	Yes
Secure Supply Chain	Occasionally	Yes

https://owasp.org/Top10/A00_2021_How_to_use_the_OWASP_Top_10_as

Secure code review

Secure code review

- Essential skill for an AppSec engineer
  - You should be able to **write** code in order to effectively read it

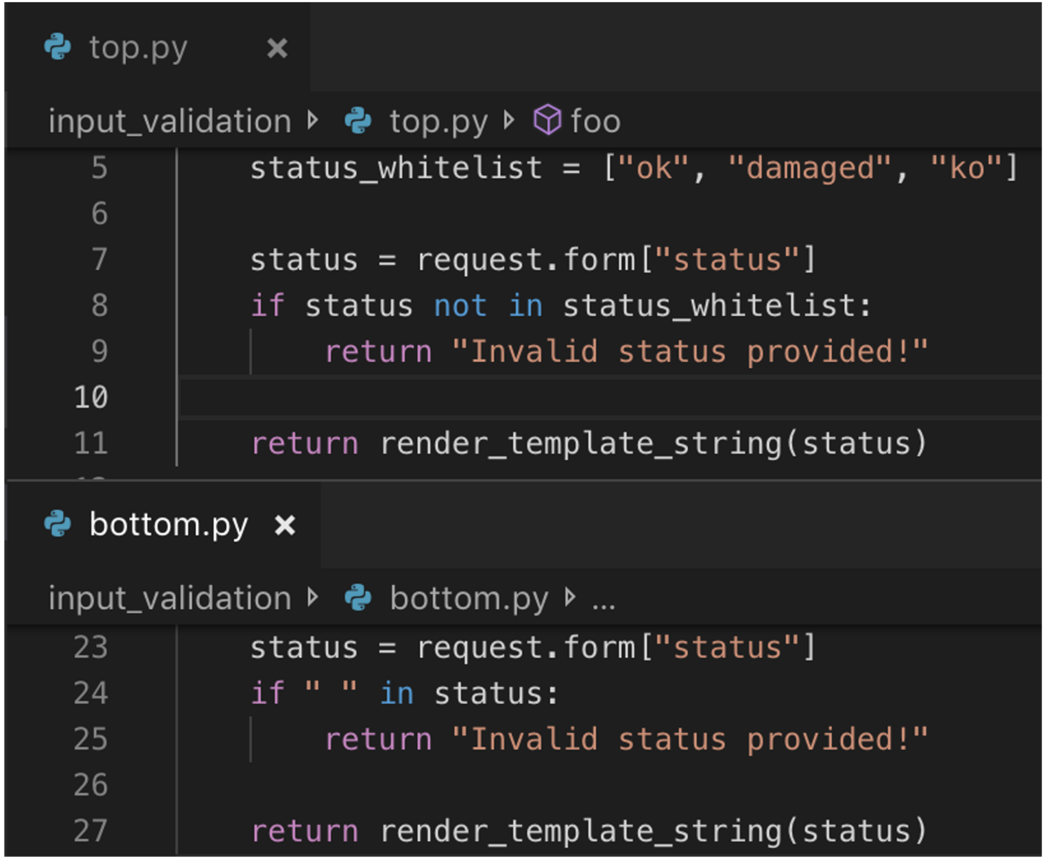
- Many different standards that can help with web coverage
  - OWASP Top 10 - enough for low hanging fruit
  - OWASP ASVS - comprehensive coverage with 3 levels based on org maturity

- **In depth** manual review of critical parts (e.g. authentication, payments)
  - Find a methodology that works for you, but keep some freedom
  - CTFs are a great way how to test and find one that suits you

- **Wide** review of the rest
  - Grep can get you further than you would expect, don‘t use SAST just for the sa

(Web) Secure code review *quiz*

[Input validation] **whitelist** *or* **blacklist**?


top.py ×
input_validation ▸ top.py ▸ foo

```python
    status_whitelist = ["ok", "damaged", "ko"]

    status = request.form["status"]
    if status not in status_whitelist:
        return "Invalid status provided!"

    return render_template_string(status)
```

bottom.py ×
input_validation ▸ bottom.py ▸ ...

```python
    status = request.form["status"]
    if " " in status:
        return "Invalid status provided!"

    return render_template_string(status)
```

Input validation

- Always prefer **whitelist** over **blacklist**
  - Would you keep a ***blacklist*** of people that *cannot* enter your house?... Probably not :-)

- Cast user input to desried type and keep the character set low
  - Use **enums** (no way to allow any unexpected input this way)
  - Limit possible characters only to the minimum required (Do you *really* need < or " in your phone number?)
  - Limit the maximum size of the input to something you won't hit (DoS by sending a very long password)
  - The more special characters you allow, the more problems you might have in the future

- Beware: input validation **is not** a replacement for parametrized st

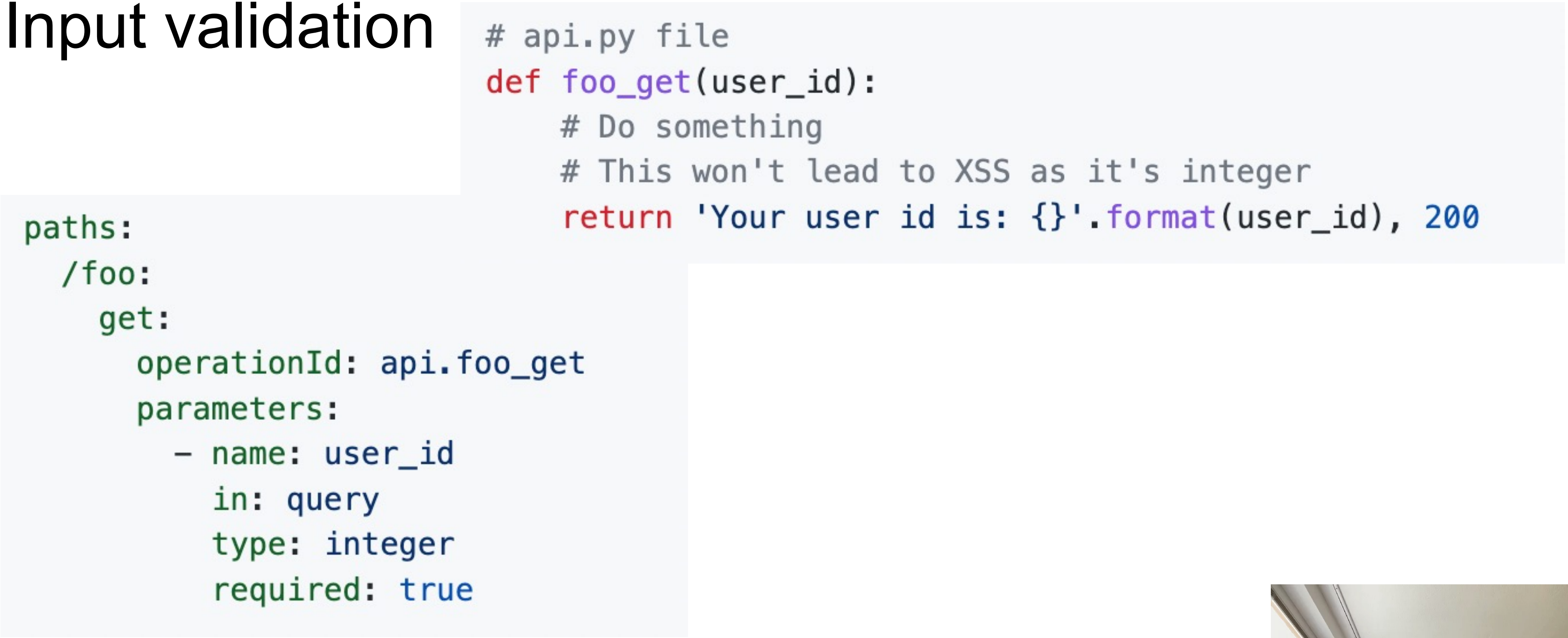
output escaping

Input validation

- Outsource input validation to frameworks
  - Some web frameworks (such as connexion) allows you to specify types/validation directly in the API schema. This is *the best* you can get.
  - Otherwise, use available framework-specific validation functions/modules:
    - Python Flask - WTForms or webargs
    - Python Django - Validators
    - Golang go-playground/validator.v9

- Typing is good, use it! (even in python)
  - Especially important for stability, but also security
  - Basically all companies use typed python for big projects

Input validation

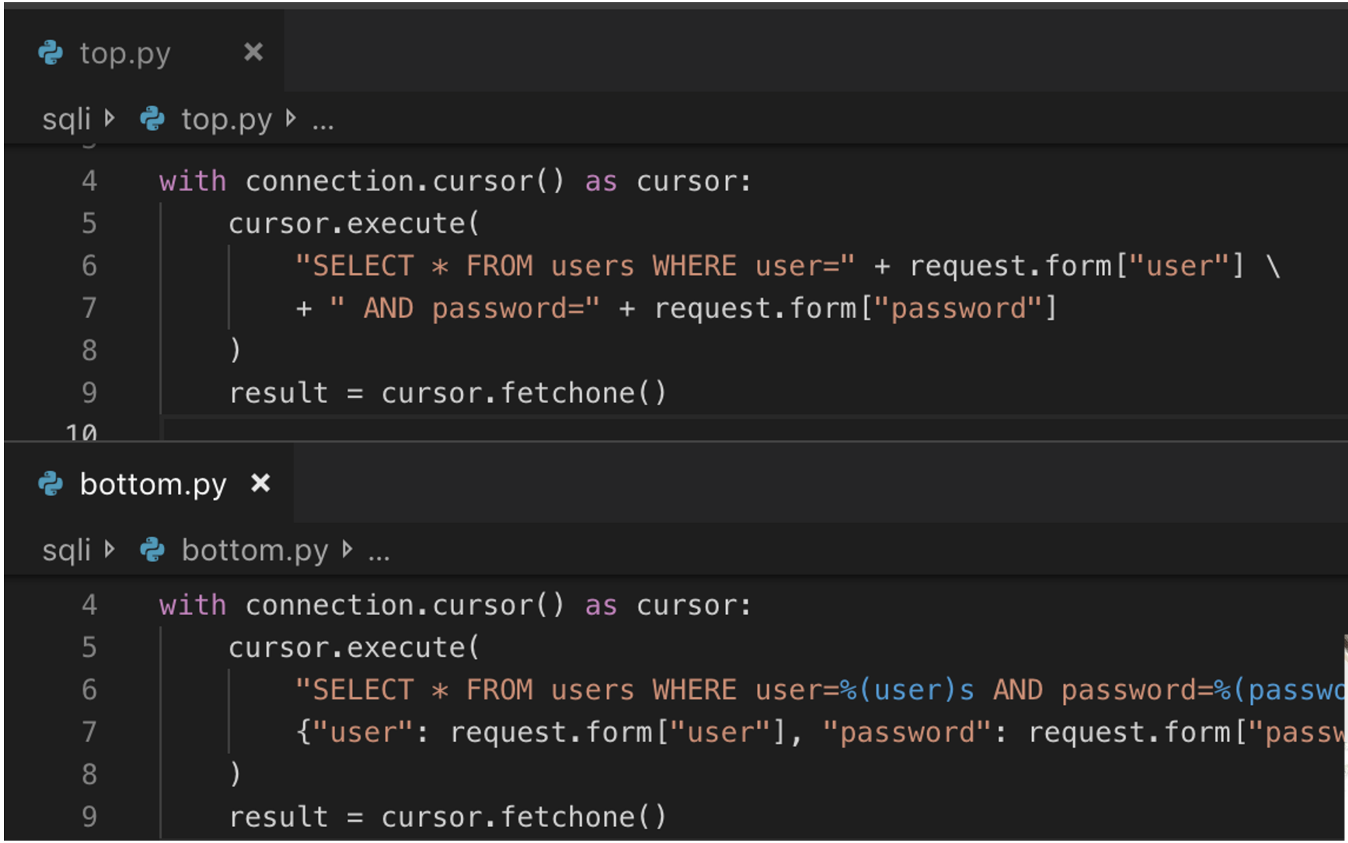


```python
# api.py file
def foo_get(user_id):
    # Do something
    # This won't lead to XSS as it's integer
    return 'Your user id is: {}'.format(user_id), 200
```

```yaml
paths:
  /foo:
    get:
      operationId: api.foo_get
      parameters:
        - name: user_id
          in: query
          type: integer
          required: true
```

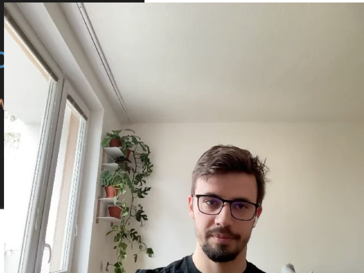
https://github.com/spec-first/connexion

[Injection] **parameterized** *or* **format**?



```python
with connection.cursor() as cursor:
    cursor.execute(
        "SELECT * FROM users WHERE user=" + request.form["user"] \
        + " AND password=" + request.form["password"]
    )
    result = cursor.fetchone()
```

```python
with connection.cursor() as cursor:
    cursor.execute(
        "SELECT * FROM users WHERE user=%(user)s AND password=%(passwo
        {"user": request.form["user"], "password": request.form["passw
    )
    result = cursor.fetchone()
```

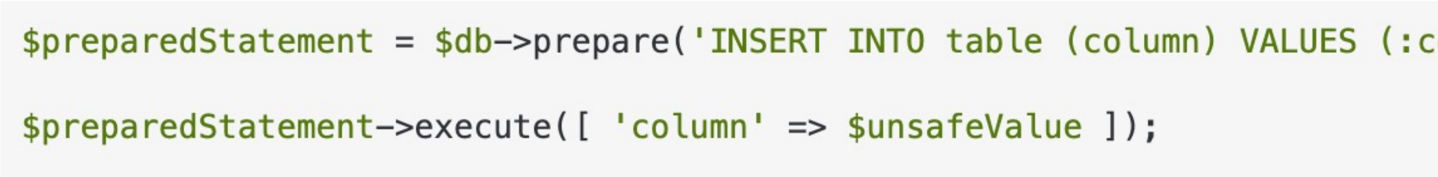


Injection

- #1 flaw in OWASP Top 10 for 9 years

- **Not** limited only to SQL (NoSQL, LDAP, command injection)

- **Force** people to use **prepared statements** or **ORMs**
  - First, hardcoded query gets prepared and compiled by DB server
  - Only afterwards, the user-defined values are inserted. This guarantees that user input isn‘t interpreted as SQL query -> no injection.

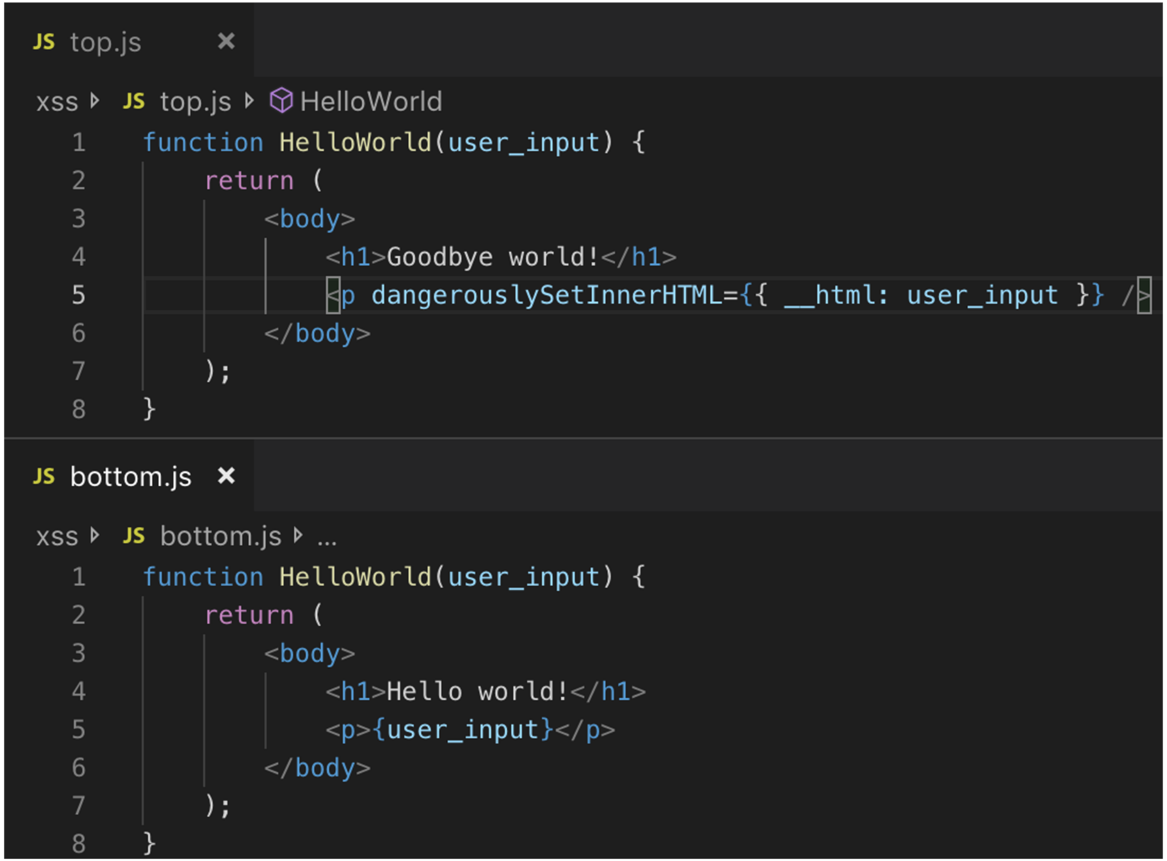


```
$preparedStatement = $db->prepare('INSERT INTO table (column) VALUES (:c

$preparedStatement->execute([ 'column' => $unsafeValue ]);
```

https://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php

[Framework gotchas - React] **dangerous** *or* **not**?

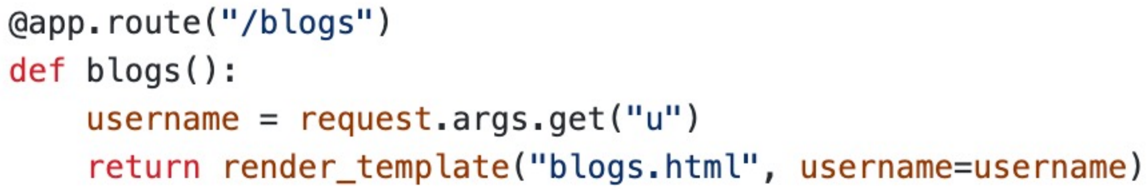


```
function HelloWorld(user_input) {
    return (
        <body>
            <h1>Goodbye world!</h1>
            <p dangerouslySetInnerHTML={{ __html: user_input }} />
        </body>
    );
}
```

```
function HelloWorld(user_input) {
    return (
        <body>
            <h1>Hello world!</h1>
            <p>{user_input}</p>
        </body>
    );
}
```

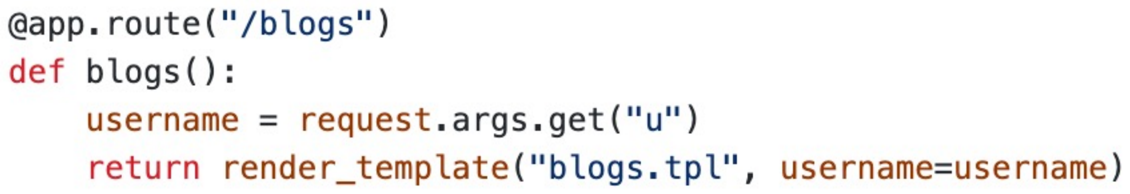
[Framework gotchas - Flask] auto-escaping

- Safe



```
@app.route("/blogs")
def blogs():
    username = request.args.get("u")
    return render_template("blogs.html", username=username)
```

- Unsafe (XSS if username is reflected back on page)



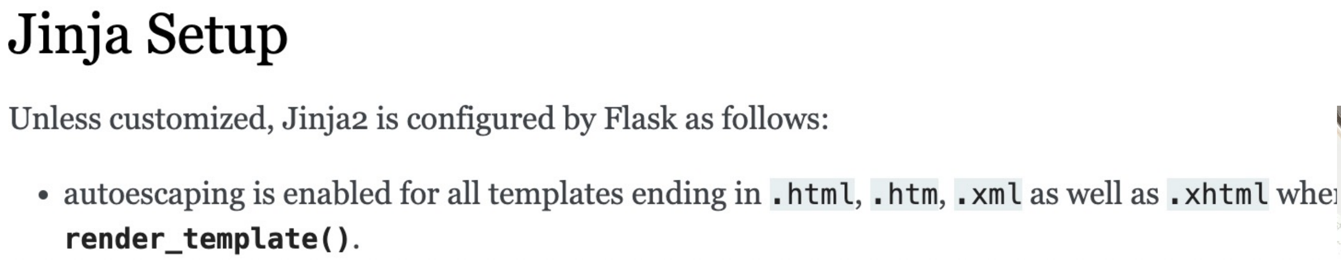
```
@app.route("/blogs")
def blogs():
    username = request.args.get("u")
    return render_template("blogs.tpl", username=username)
```

https://github.com/kiwicc

Framework gotchas

- Framework have evolved and lots of them are **secure by default**
    - All options on how to introduce vulnerability should be clearly marked as dangerous

- You should **read the docs** of your frameworks and look for any pitfalls
    - Obvious ones such as [React's dangerous functions]
    - Or less obvious ones, such as [Flask's auto-escaping enabled only for some extensions]
    - SAST rulesets or [lists of sinks] are a great place to start

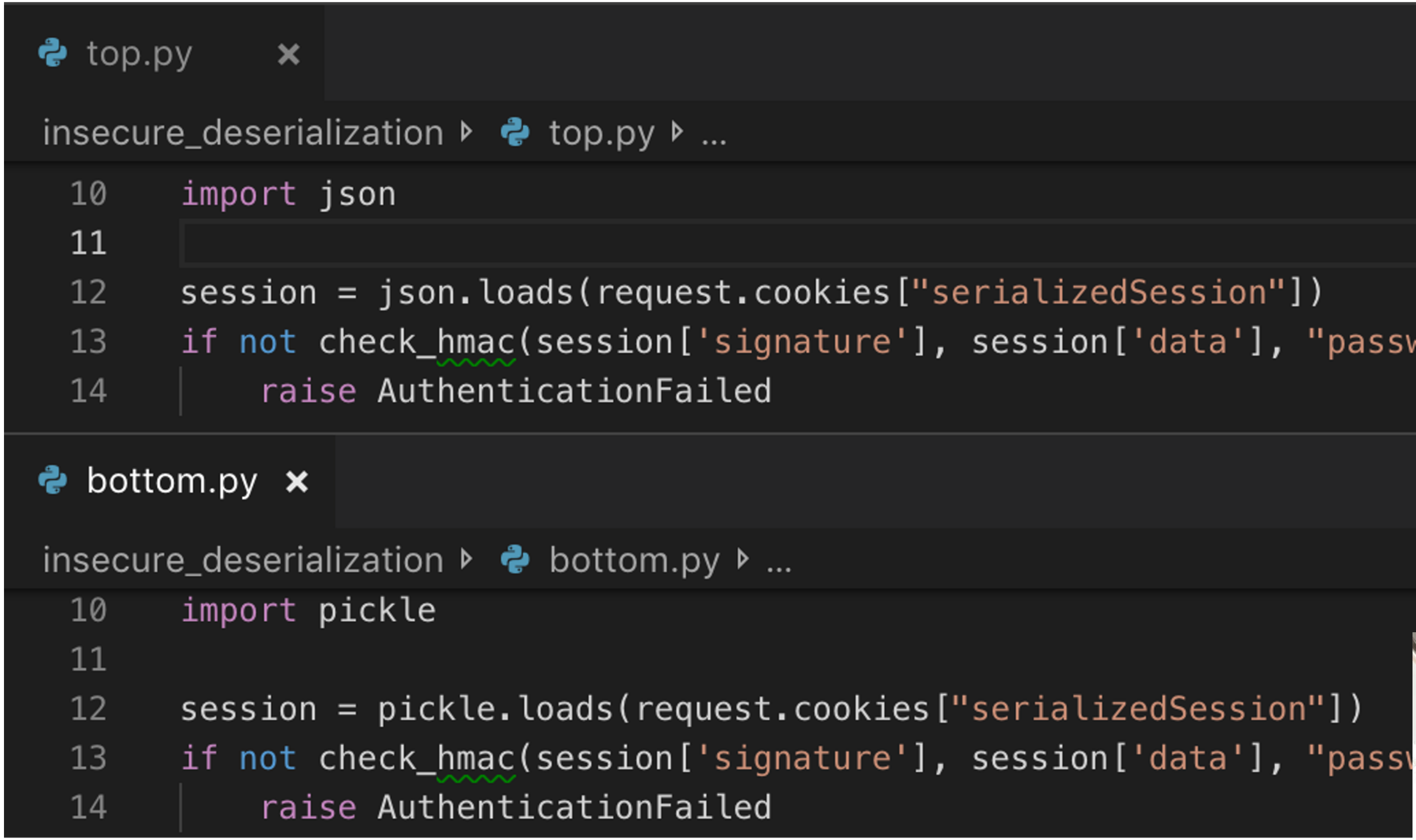


Jinja Setup

Unless customized, Jinja2 is configured by Flask as follows:

- autoescaping is enabled for all templates ending in `.html`, `.htm`, `.xml` as well as `.xhtml` whe
  `render_template()`.

[Deserialization] **pickle** *or* **json**?



top.py ×

insecure_deserialization ▸ top.py ▸ ...

```python
import json

session = json.loads(request.cookies["serializedSession"])
if not check_hmac(session['signature'], session['data'], "password123"):
    raise AuthenticationFailed
```

bottom.py ×

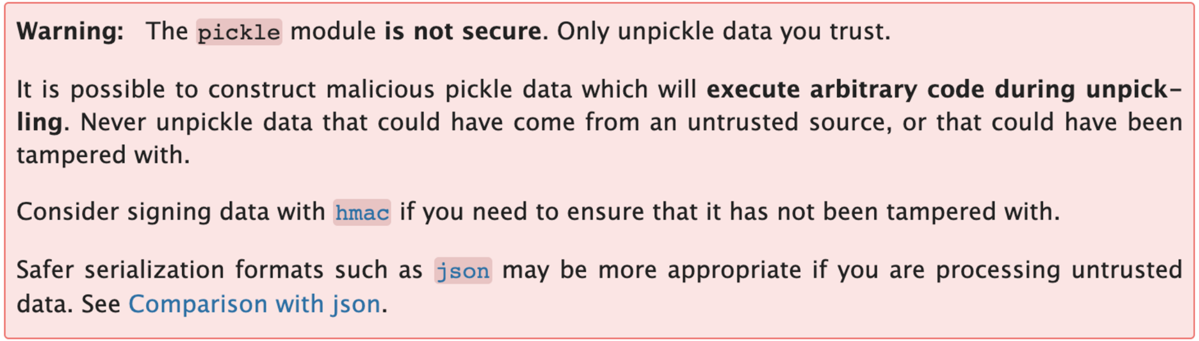
insecure_deserialization ▸ bottom.py ▸ ...

```python
import pickle

session = pickle.loads(request.cookies["serializedSession"])
if not check_hmac(session['signature'], session['data'], "pass
    raise AuthenticationFailed
```

Deserialization (language gotchas)

- **Read the docs**



Warning: The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See Comparison with json.

- Know the language you review code for and be aware of its specifics

- CTFs are great learning resource of similar language/framework
pitfalls

Static Application Security Testing (SAST)

SAST

- Principles discussed quite exhaustively in previous lecture

- Today, we'll focus on:
    - Web-specific tooling
    - Some best practices for a rollout of SAST in a big organization
    - Secrets detection in code

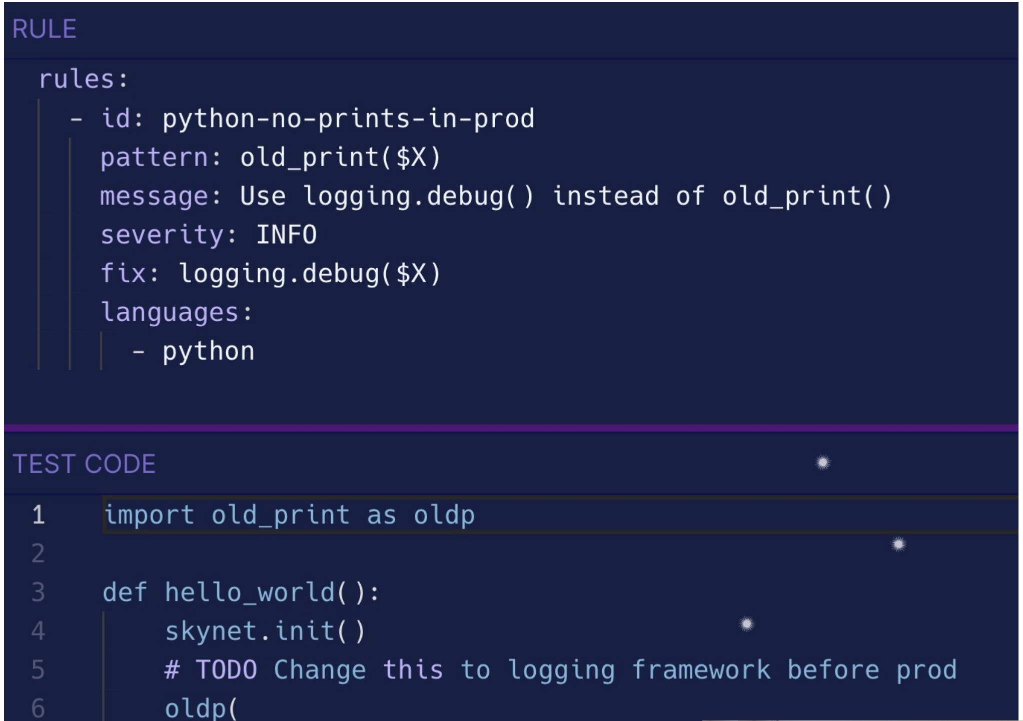
SAST - tooling

- GitHub/GitLab have both great SASTs
  - Github CodeQL - get bounties for writing SAST rules
  - GitLab's SAST (merged open-source tools into 1 image)
  - The closer it is to devs, the better.

- Language specific SAST tools (awesome-static-analysis)
  - Recommended multi-language SAST: **semgrep.dev**
  - Some language specific tools (e.g. Pysa for python) if you need to cover complex cases
  - Rulesets of this tools are **great** learning resource of vulnerable language-specific gotchas that can be independently used e.g. in code reviews.

- Build easily extensible alerting on regexes/keywords appearing ir
  - You might want to be aware that import cryptography newly appeared somewh talk to the developer trying to implement some potentially risky feature before I

SAST – semgrep.dev



RULE

```yaml
rules:
  - id: python-no-prints-in-prod
    pattern: old_print($X)
    message: Use logging.debug() instead of old_print()
    severity: INFO
    fix: logging.debug($X)
    languages:
      - python
```

TEST CODE

```python
import old_print as oldp

def hello_world():
    skynet.init()
    # TODO Change this to logging framework before prod
    oldp(
        '--> debug, skynet init vector
    )
    # oldp('don't detect this, it\'s co
```

SAST - implementation best practices

- Triage issues effectively
    - Prioritize issues based on the business risk.
    - Don’t bother devs with false positives / low severity findings

- Start slowly
    - Easy to get overwhelmed by the amount of findings
    - Choose few high-impact vulnerability classes and focus on them - repeat once done

- Define a clear process for the issue triage, e.g.
    - High signal, mid+severity – alert devs in CI/CD before commit lands
    - Low signal, high severity OR Mid signal, mid severity – alert SecEng
    - The rest – *backlog*

Secrets in code detection

- Technically still part of SAST, as you analyze the source code

- Easy detection and easy direct exploitation
  - API keys of cloud providers can be exploited for crypto mining
  - SaaS providers such as PayPal, GitHub or Twitter
  - Private RSA keys, database dumps, …

- How bad can it git?
  - Research scanning all GitHub commits for secrets over 6 months.
  - Thousands new, valid and unique secrets leaked every day
  - Still huge space for improvement in detection (they scanned secrets only for 1

- Low effort & High impact (rewards up to $15,000 for a single GitH

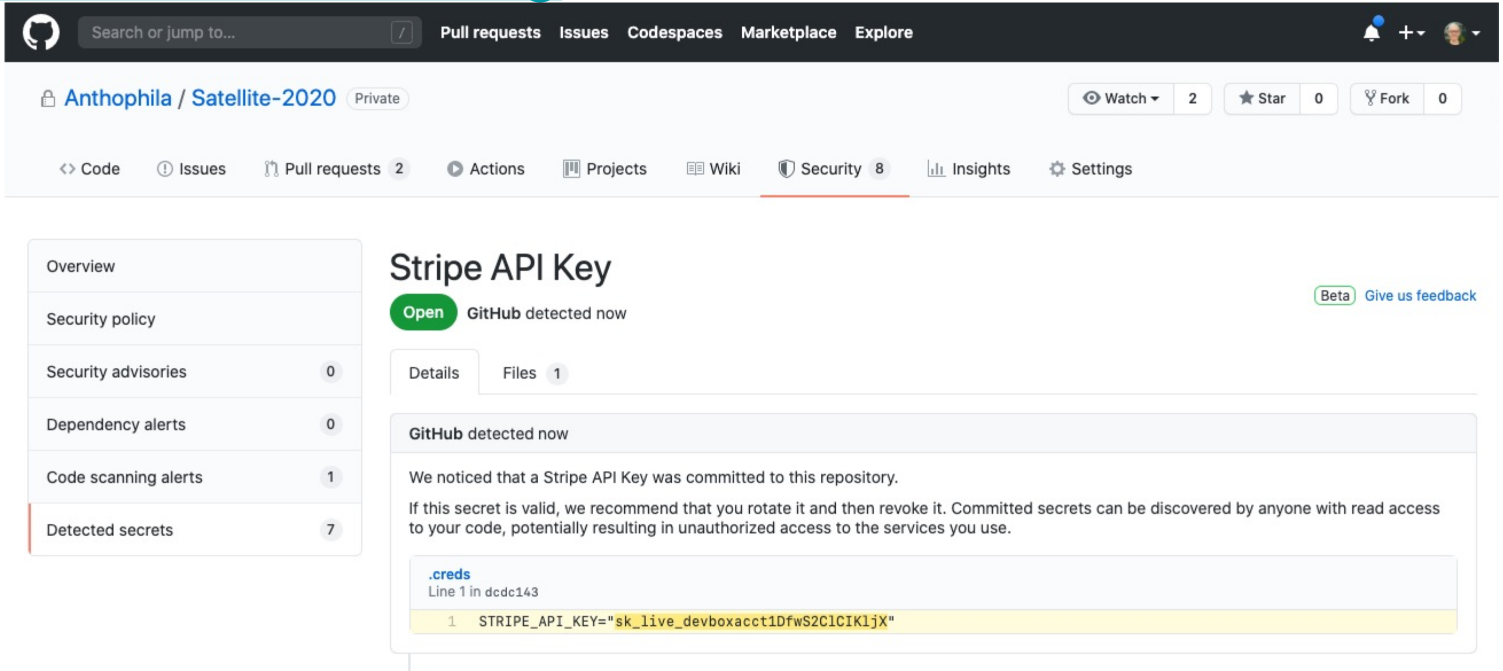
Secrets in code detection - tooling

- GitHub's token scanning - low false positives, auto-revocation (e.g. AWS), by default present on github.com
- GitLab’s SAST - gitleaks and TruffleHog with the default config

- gitleaks - can combine entropy and regexes
- TruffleHog - “the original” scanner, now inferior
- shhgit - real time monitoring of GitHub commits
- semgrep.dev – yup, they also can do this!

- Everything is about having a good config file to balance the signa
negatives / false positives)

GitHub's token scanning



Stripe API Key

Open **GitHub** detected now

GitHub detected now

We noticed that a Stripe API Key was committed to this repository.

If this secret is valid, we recommend that you rotate it and then revoke it. Committed secrets can be discovered by anyone with read access to your code, potentially resulting in unauthorized access to the services you use.

.creds
Line 1 in dcdc143

```
STRIPE_API_KEY="sk_live_devboxacct1DfwS2C1CIKljX"
```

First instance of secret detected

Update .creds

Resolve this alert

Once you've revoked this secret, resolve it as revoked here. You can also report it as a false positive o
low-risk/testing secret, or just ignore it.

Dynamic Application Security Testing (DAST)

DAST - tooling (web)

- Web security vulnerability scanner
  - Focused on web apps, spiders the website *deeply*
  - Great for automated discovery of several vulnerability classes or security headers checks
  - Burp Suite (paid, superior), OWASP ZAP (open-source, used in GitLab's DAST)

- Asset discovery
  - "Bug bounty" like monitoring tools, most of them originally made *for* bug bounty
  - Many companies don't have a list of their assets => cannot specify scope for the scanner
  - Searching for assets and monitoring all of them *lightly* (picking up the low-hanging fruit)
  - Might use Web security vulnerability scanners to scan some more appealing targets
  - E.g. Assetnote (great paid product), projectdiscovery.io (open-source), BugSh
    :-))

Web security vulnerability scanner

- Security scanner running on live web application
- Crawls the website as a human would, fuzzing different “malicious” inputs
  - Might be quite intrusive => should be used on staging or well known production environment
  - Sometimes problems with login to apps (especially *complicated* flows like SAML/OAuth2), which can be solved by using a “browser” like Selenium for login and then passing session to scanner.
- Similar to fuzzers/dynamic analysis for programs described in previous lecture, just specific for web

Asset Discovery - Bugshop

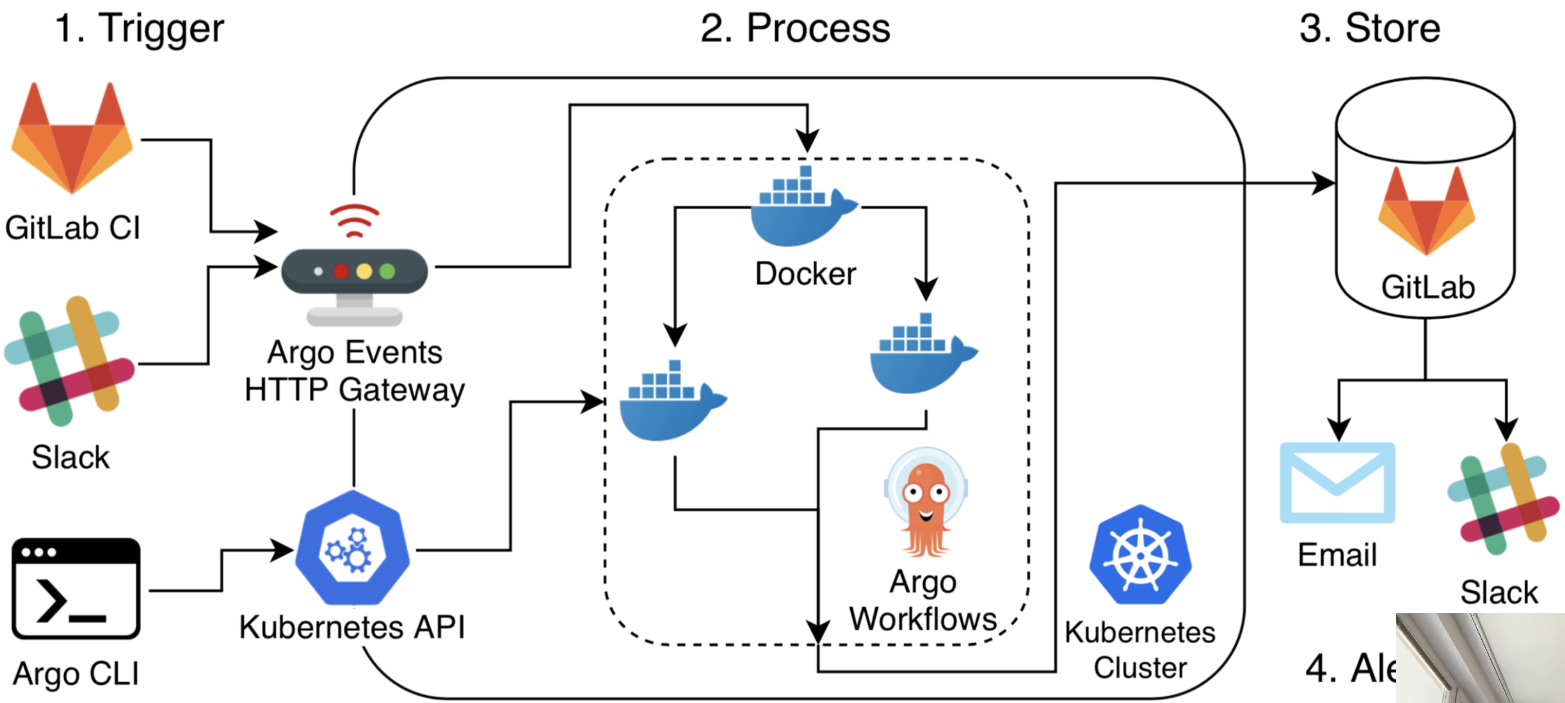


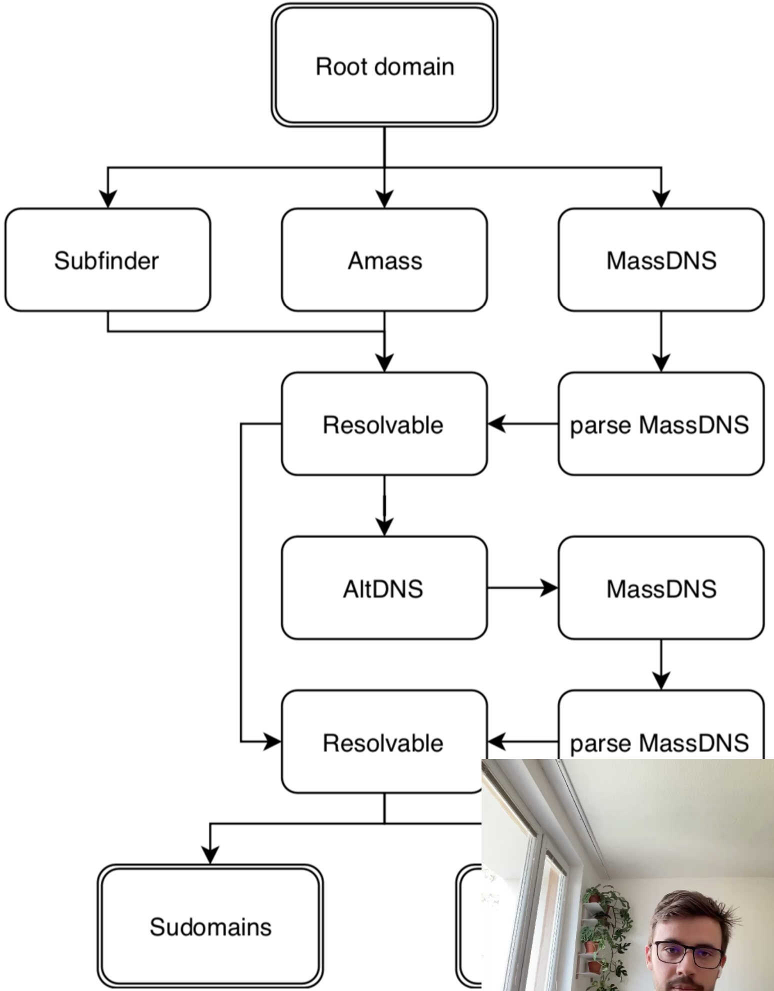
Figure 5.1: High level overview of the Bugshop.

Asset Discovery - Bugshop

- Start by **subdomain enumeration** workflow with a wildcard domain (e.g. *.muni.cz) and end with a list of hundreds subdomains (*assets*).

- Then run vulnerability and discovery checks on all newly found *assets*, find vulnerabilities (e.g. by Web security vulnerability scanner), and find more *assets* recursively.

- Further automation of workflows commonly used in bug bounty (git secret detection, bucket enumeration or subdomain takeovers)

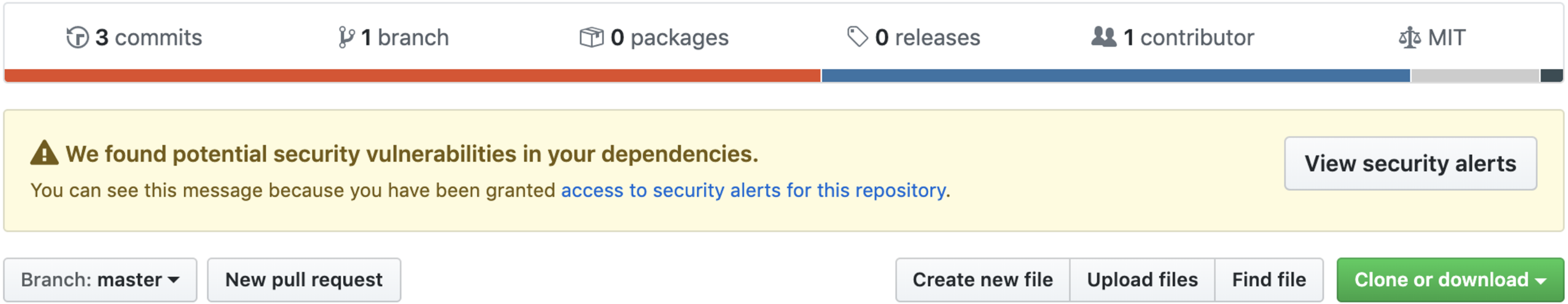


Dependency management

Software package registries

- Easy distribution of packaged code for use by other developers
  - Node.js - npm
  - Java - Maven Central
  - Python - PyPI
  - Ruby - RubyGems
  - Docker - DockerHub (distribution of docker images, not code)
- Heavy growth in the past years (especially Maven/npm)
- Convenient (1 command and code is ready to use)

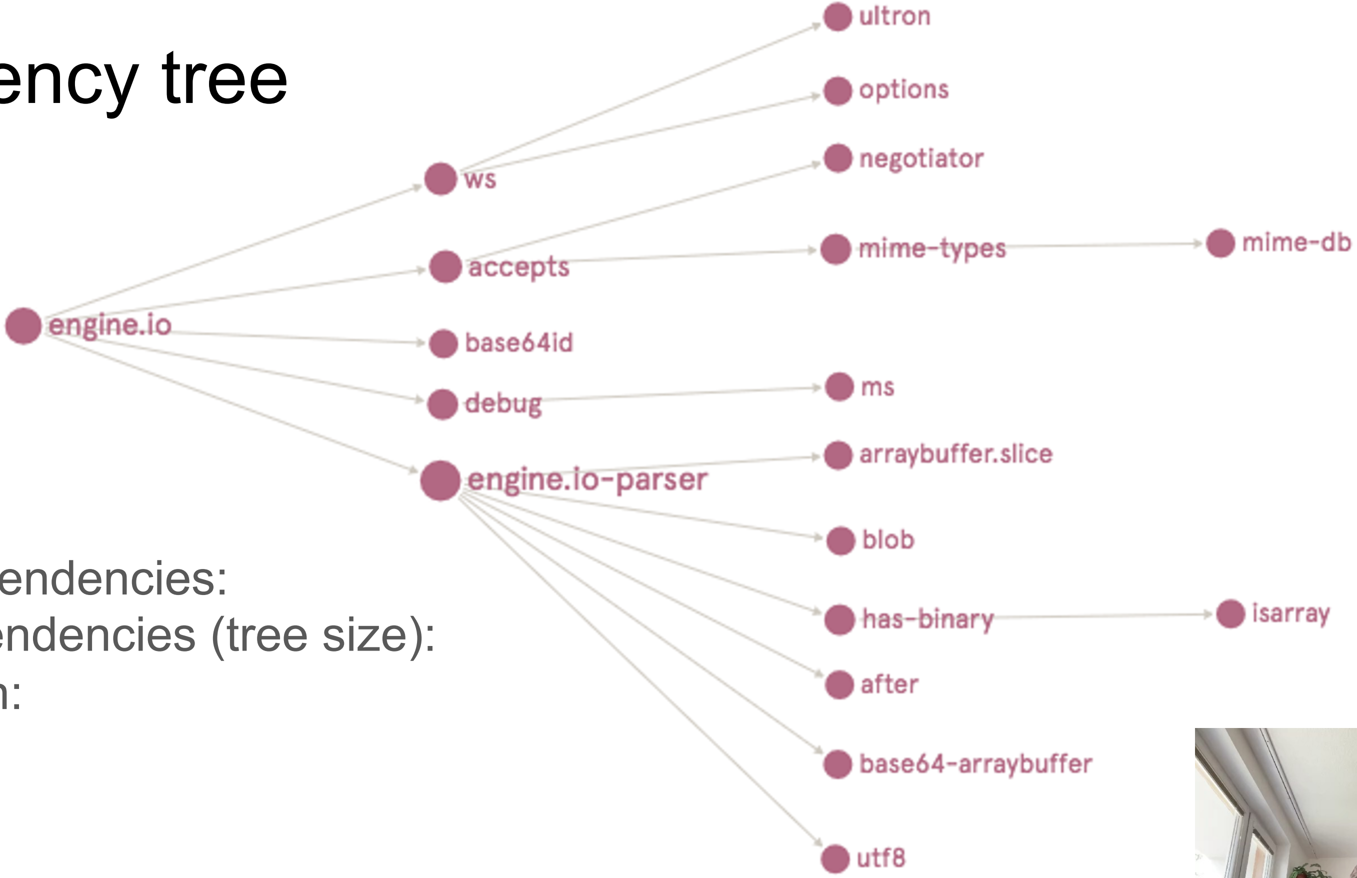
How many of you have seen this warning?



We found potential security vulnerabilities in your dependencies.

You can see this message because you have been granted access to security alerts for this repository.

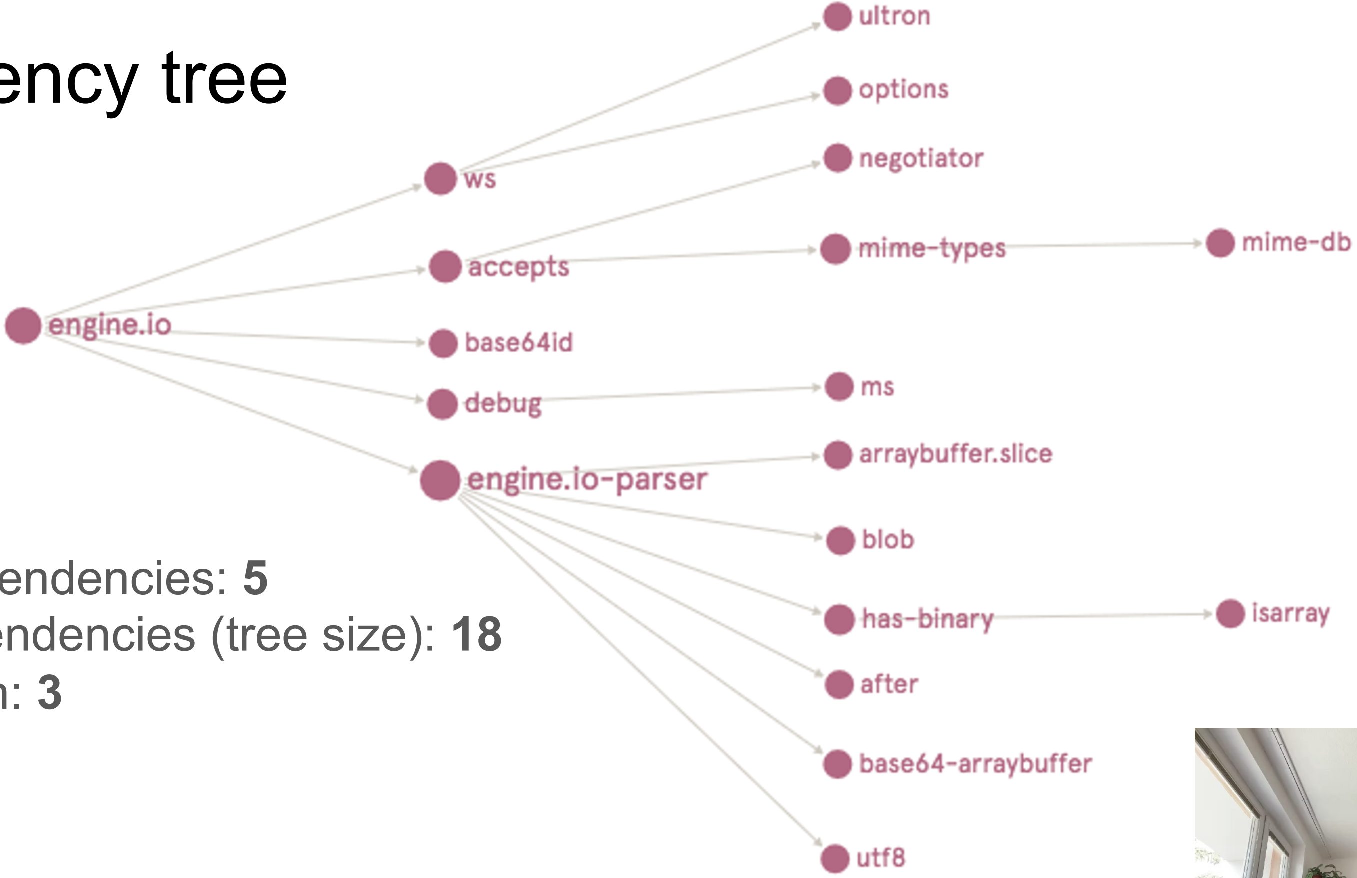
Dependency tree



- Direct dependencies:
- Total dependencies (tree size):
- Tree depth:

https://medium.com/graph-commons/analyzing-the-npm-dependency-network-e2cf318c

Dependency tree



- Direct dependencies: **5**
- Total dependencies (tree size): **18**
- Tree depth: **3**

https://medium.com/graph-commons/analyzing-the-npm-dependency-network-e2cf318c1

Quiz time

1 - Quiz

How many packages were on npm in April 2019 (Node.js package registry)?

▲ 959,567

◆ 42,517

● 370,426

■ 681,476

1 - Quiz

How many packages were on npm in April 2019 (Node.js package registry)?

▲	959,567	✓
◆	42,517	✗
●	370,426	✗
■	681,476	

2 - Quiz

How many packages on npm could be considered abandoned (no release in past 12 months)?

- ▲ 57,000 (7%)
- ◆ 122,000 (15%)
- ● 224,000 (27%)
- ■ 496,000 (61%)

2 - Quiz

How many packages on npm could be considered abandoned (no release in past 12 months)?

△ 57,000 (7%) ✗

◇ 122,000 (15%) ✗

○ 224,000 (27%) ✗

□ 496,000 (61%)

3 - Quiz

What is average depth of a package dependency chain on npm?

- ▲ 1 - 2
- ◆ 2 - 3
- ○ 3 - 4
- □ 4+

3 - Quiz

What is average depth of a package dependency chain on npm?

- ▲ 1 - 2 ✕
- ◆ 2 - 3 ✕
- ○ 3 - 4 ✕
- □ 4+

4 - Quiz

What is average depth of a package dependency chain on PyPI?

- ▲ 1 - 2
- ◆ 2 - 3
- ● 3 - 4
- ■ 4+

4 - Quiz

What is average depth of a package dependency chain on PyPI?

▲ 1 - 2 ✓

◆ 2 - 3 ✗

● 3 - 4 ✗

■ 4+

5 - Quiz

What is the avg number of dependencies for an npm package?

- ▲ 87
- ◆ 52
- ● 29
- ■ 6

5 - Quiz

What is the avg number of dependencies for an npm package?

▲	87	✓
◆	52	✗
●	29	✗
■	6	

6 - True or false

Should you update your dependencies automatically, right after the release comes out?

▲ False

◆ True

6 - True or false

Should you update your dependencies automatically, right after the release comes out?

▲ False ✓

◆ True ✗

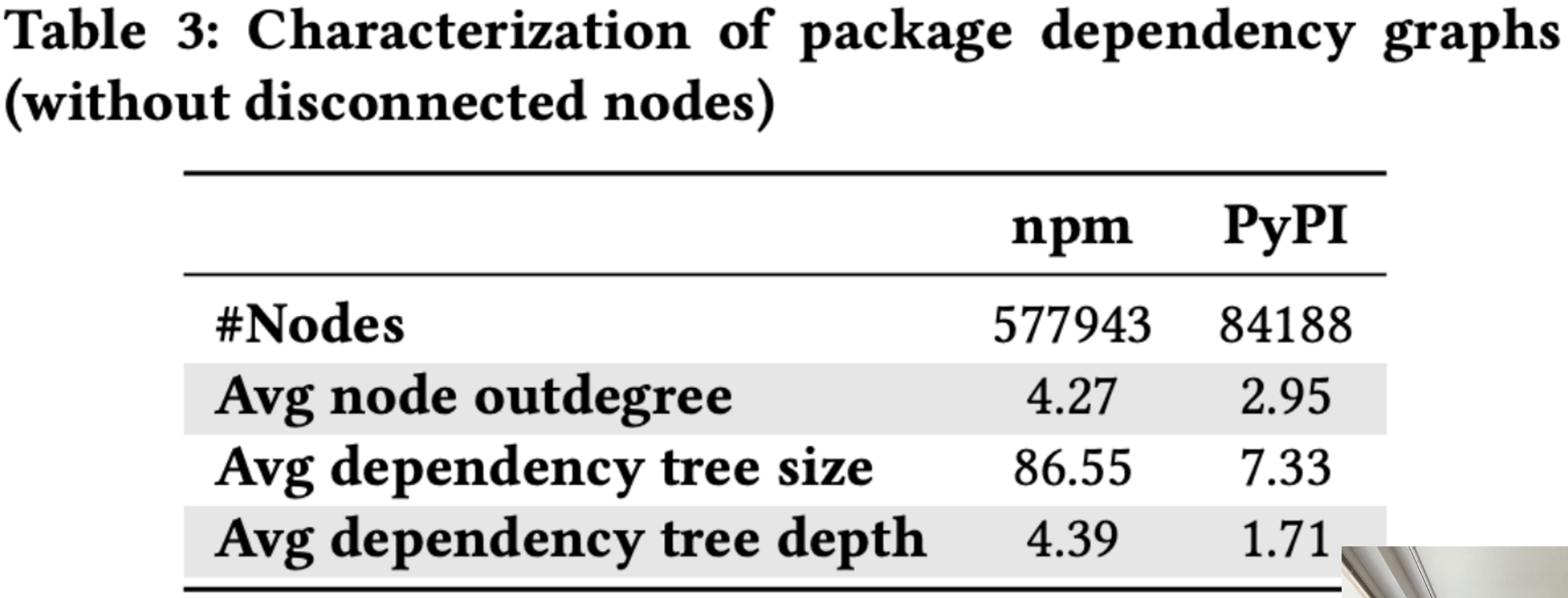
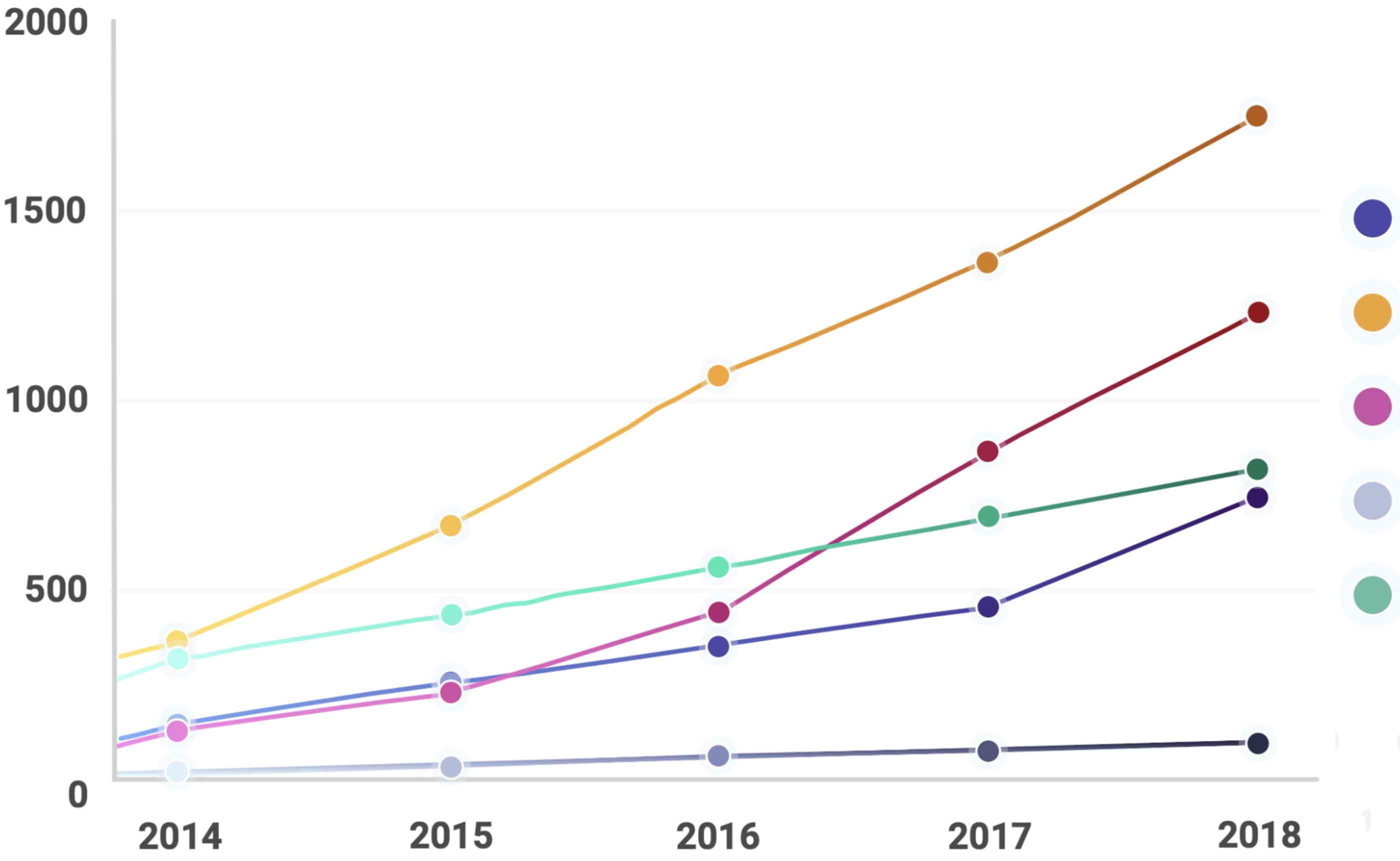


Table 3: Characterization of package dependency graphs (without disconnected nodes)

	npm	PyPI
#Nodes	577943	84188
Avg node outdegree	4.27	2.95
Avg dependency tree size	86.55	7.33
Avg dependency tree depth	4.39	1.71

https://www.researchgate.net/publication/331587729_Security_Issues_in_Language-based_Sofware_Ec

New vulnerabilities each year by ecosystem



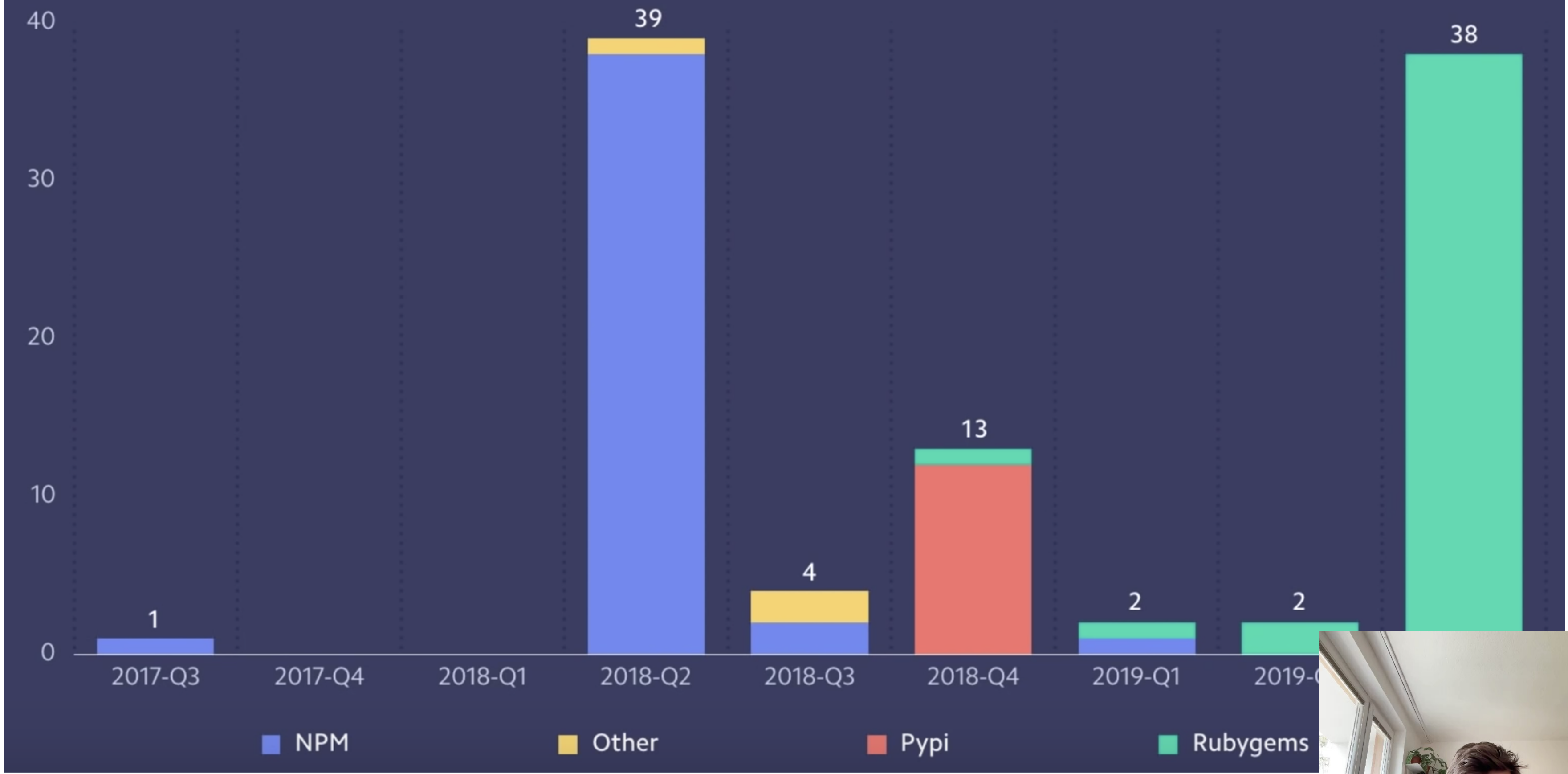
https://res.cloudinary.com/snyk/image/upload/v1551172581/The-State-Of-Open-Source-Security-Report-2019-

[non-malicious] Risks of package registries

- Packages with **known vulnerabilities** (outdated/abandoned dependencies)
- 88% growth in (reported) packages vulnerabilities over the past two years
- Growing dependency chains increase the chance of compromising your dependencies indirectly

WHAT IF I TOLD YOU
THAT YOU SHOULD NOT AUTO-UPDATE YOUR SOFTWARE

Intentionally Malicious Modules



[Stripe Sessions 2019 | When data contradicts security best practices]()

[malicious] Risks of package registries

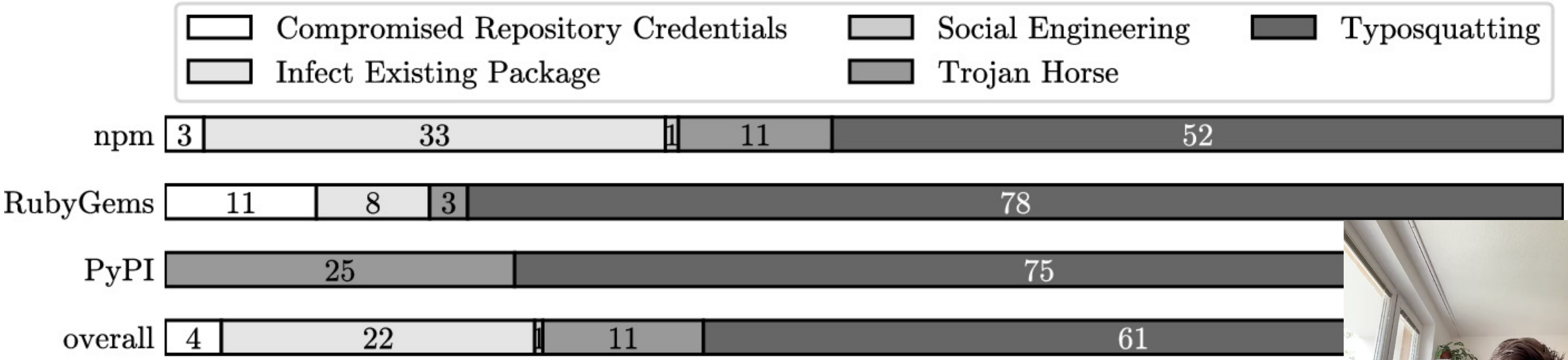
- **Malicious releases**
  - npm `event-stream` compromised via its dependency

- **Protestware**
  - npm node-ipc wiping Russian/Belarus machines with WITH-LOVE-FROM-AMERICA.txt message to show support of Ukraine
  - Not the greatest idea as it also wiped a ton of pro-Ukraine companies

- Version number **might not** be an immutable identifier in many registries
- **Private registries** can have unexpected default behaviour, which allowed one researcher to hack into Apple or Microsoft.

[malicious] Risks of package registries

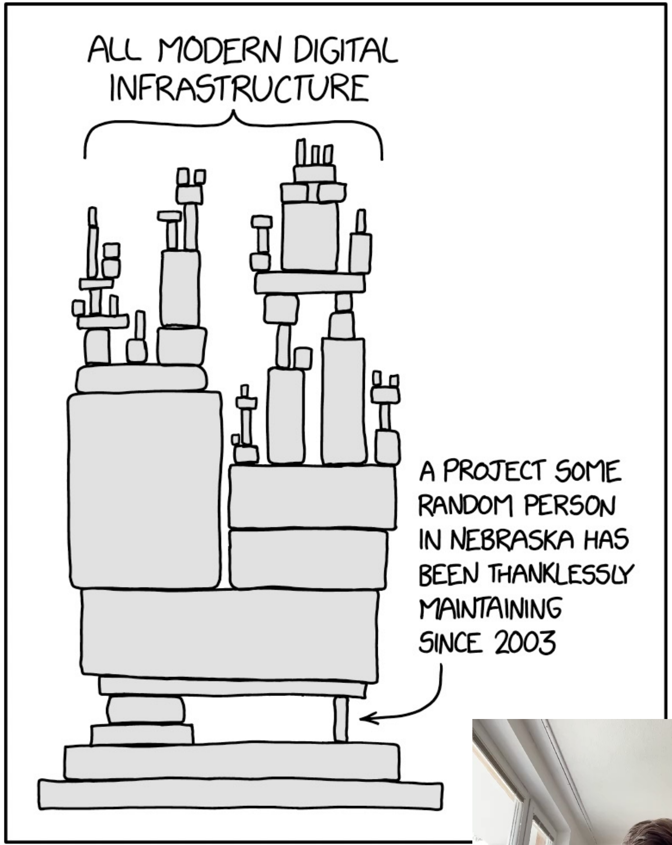
- Typosquatting of package names
  - pip install request (instead of requests)
  - as pip executes code during the **installation** => 1 typo == RCE
  - [SK-CSIRT identified malicious packages on PyPI]()
- Most (>50%) malicious packages mimic existing packages via typosquatting



Injection technique used to introduce the malicious package into a package
https://link.springer.com/chapter/10.1007/978-3-030-52683-2_2/figures/8

Risks of package registries

- ***Nobody* is reviewing the code** before installing on production servers
  - In ideal world, you would hold a list of approved *reviewed* packages with versions.
  - In reality, the whole package ecosystem is super fragile
  - E.g. Hacking 20 high-profile dev accounts could compromise half of the npm ecosystem


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

https://xkcd.com/2347/

Dependency management - tooling

- Dependency monitoring
  - GitHub
  - GitLab
  - Built-in in the package manager (npm audit / pipenv - safety)
  - Commercial (Snyk)
  - OWASP Dependency Check

- Automatically open pull request with dependency update
  - GitHub
  - Renovatebot
  - Commercial (Snyk). Sad part is that all commercial tools use their own *private licensed* database of vulnerabilities. (╯°□°)╯︵ ┻━┻

Dependency management - best practices

- **Automatically monitor dependencies** for known vulnerabilities
  - Both GitHub and GitLab have built-in **dependency scanning** available. Neither of them is perfect, but it's *something* and it's easy to start with.

- **Don't** auto-update right after the release (update != security patch)
  - Wait few days/weeks for community to spot bugs or hijacked/malicious packages.
  - Naturally, continue to apply **security patches** immediately.

- Use **immutable identifiers** for packages
  - Version number is a **mutable identifier** in Docker, Maven or PyPI.
  - Hash digests are preferable and protect you even from a compromise of the re
  - Auto-update tools such as renovatebot can help with this.

Penetration tests

Penetration tests - who does it?

- **Internal**
  - Done internally, e.g. by members of the AppSec team
  - Good for deep tests that require some internal knowledge of the application

- **External**
  - Outsource to an external company
  - Usually done this way so security team can focus on other issues
  - Compliance requirement in some cases (e.g. you cannot pentest yourself in PCI DSS)
  - Good way to earn public reputation (*pentested by Cure53 / XYZ*)

Penetration tests - types

- **Black box**
  - The same conditions as an attacker (no access to docs or code)
  - Not really effective in value/money ratio as pentester spends more time on app discovery
- **Grey box**
  - Access to app documentation or small chunks of code
  - Possible cooperation/chat between pentester and company
- **White box**
  - Source code available to the pentester (can run SAST tools on it)
  - Great for any deep pentest (business logic, auth)
  - Essentially required for pentesting iOS apps or similar

Penetration tests - methodology

- *Best effort* test
    - Give pentester a *free hand* on techniques used in testing
    - Usually lasts 3-10 days
- Detailed test following an official testing guide
    - **OWASP Testing Guide v4 (OTG4),** NIST 800-115 or OSSTMM
    - Test following an established methodology might be required by compliance (PCI DSS)
    - Usually lasts 2-4 weeks depending on size of the application

Penetration tests - best practices

- Rotate at least two pentesting companies
  - Each company uses different scanners or might check unique techniques
  - They can catch mistakes of each other => higher motivation
- **Pentest before release** & (ideally) regularly
  - Pentest can save you quite some money that you would spend on bug bounties later
- **Scope pentests** smartly
  - Let pentesters know which part of application is your priority and share all relevant docs/code
- Have a **healthy** bug bounty program :-)
  - Scoped pentests will never cover your whole external attack surface
  - Some companies keep pentests only for compliance

Bug bounty

Bug bounty

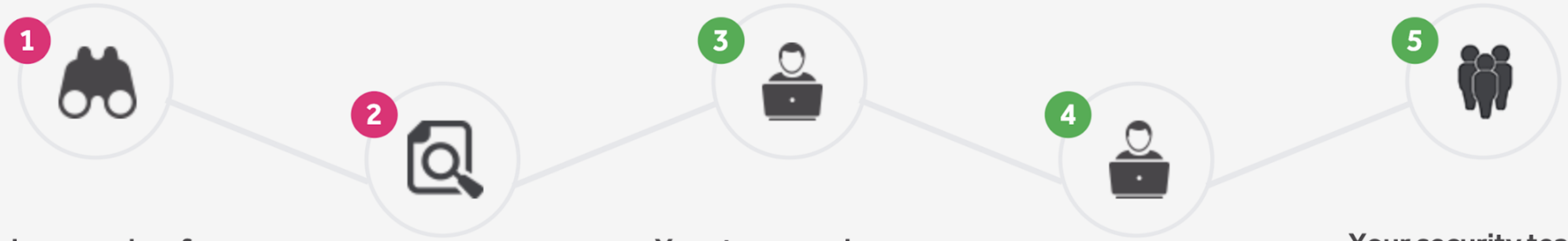
- “Please come, **hack our apps**, **report** it to us and **get paid.**”
  …and without lawsuits :-)

- Great part time income for students - you *can* learn a lot during it :)

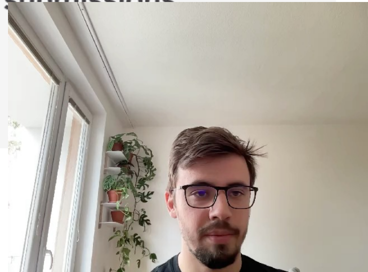
- Experiencing **huge** boom in the past few years

A Self-Managed HackerOne Bug Bounty Program

Use your abundant resources and past experience to run your own bug bounty program.



1. Hacker searches for vulnerabilities
2. Hacker submits it to your organization
3. Your team works closely with hackers to receive all relevant data
4. Your security staff validates all vulnerability reports
5. Your security team triages all submissions and fixes all valid submissions

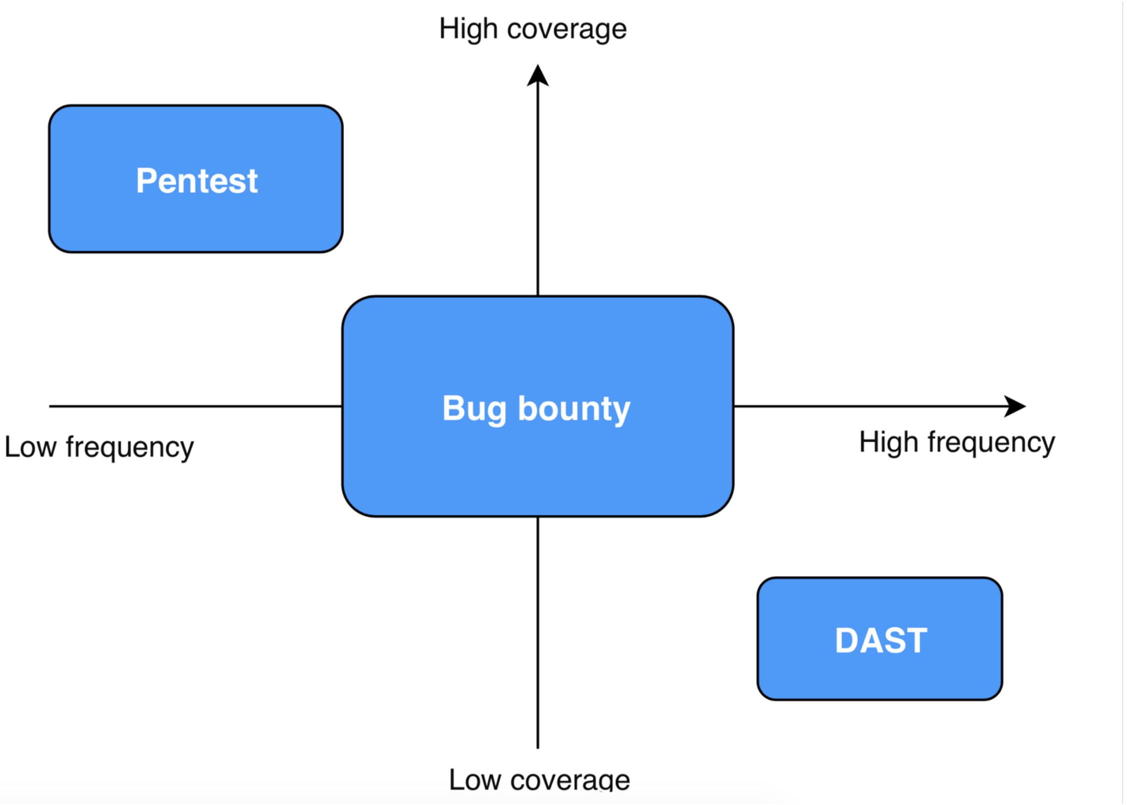


Bug bounty - types

- **Self-hosted program**
  - Company publishes website with the policy (scope, rewards, rules, contact)
  - [Pros] No 3rd party involved, hackers communicate directly with the company
  - [Cons] Company needs to handle payments, web platform and has to get interest of hackers
  - Examples include Medium, Google, Microsoft or Facebook

- **Bug bounty platforms**
  - Company makes contract with another company (bug bounty platform)
  - [Pros] Convenient web app for triage, payments handled by platform, hundreds of registered hackers ready to hack
  - [Cons] 3rd party has access to your bugs, cost ($XX XXX/year)
  - Example platforms are HackerOne, Bugcrowd, Synack, Intigriti or Hacktrophy

Pentest vs bug bounty vs DAST



OWASP AppSec USA, Zane Lackey - "Practical tips for web application security in the age
and DevOps", 2016. www.youtube.com/watch?v=Hmu21p9ybWs

Some general best practices

- *Make the right thing easy to do!*
- Show devs the cool side of security
  - Talk about the impact of bugs found, encourage and reward active people
  - Don’t underestimate *soft measures* mentioned in the beginning
- Outsource as much as possible to secure by default frameworks.
  - *Force* validation of input.
  - Stop reinventing the wheel with auth, sessions, CSRF protection or output escaping.
  - Great examples are React, Django or Connexion.
- Have secure, yet easy to use/manage secrets storage (e.g. Vault)
- Integrate most of the security checks to CI/CD pipelines
  - Continuous feedback to developers
  - Don’t forget to run checks also on schedule (unmaintained production code als
- Good example of AppSec at scale is Netflix’s concept of paved ro

… reality

- **Impossible** to do all of the above mentioned in a short amount of time
  - Resources (money/people) are usually very limited
- Prioritization is the key (decide based on risk)
  - Do you really need DAST in CI/CD if you don’t even have SAST or dependency scanning?
  - Go for quick wins - bottoms-up approach works better in *agile* companies
- Build a vision where are you heading
  - You can copy it from more mature companies, but don’t forget to adjust it based on company culture, maturity and your resources.
- Automation is the key, but tools alone **won’t** save you
  - Manual findings will be the impactful ones
  - You need to filter out the low priority issues

Thanks for your attention!

Prepare your questions ☺

Seminar

- Intro (10min)
- Dependency scanning (40min)
  - python safety (docker/pip required)
- SAST (40min)
  - python bandit hands-on (docker/pip required)
- HW setup (5min)

To prepare:

- Docker or pip
- Registered HackerOne account