Securely Unifying Deployments in an Organization for Increased Governance

Hariharan Ragothaman
conf42 DevOps 2025
Track: Security

About Me

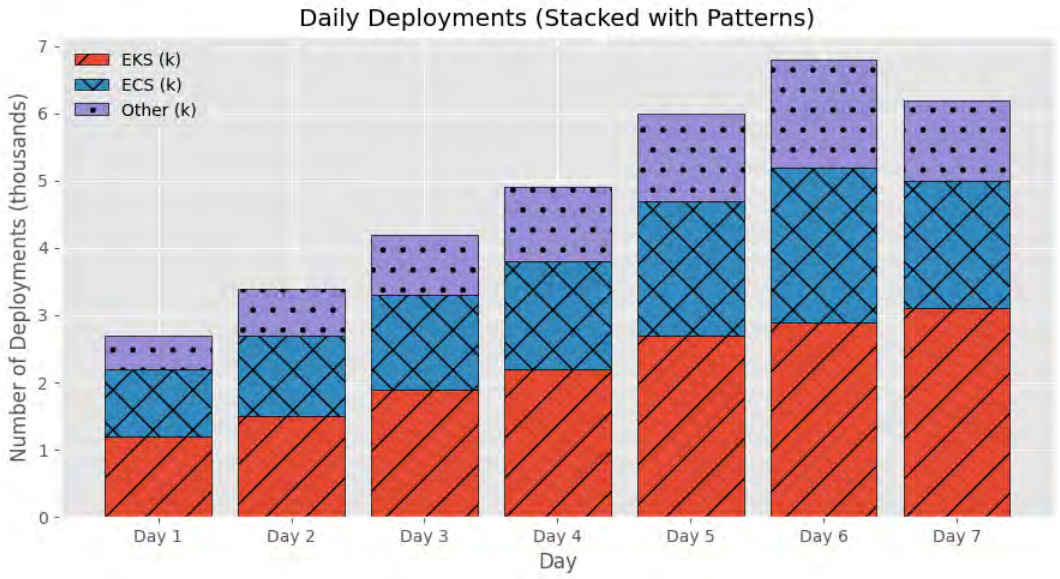
- Software Engineer at AMD
- Previously Lead Software Engineer at athenahealth
  - Also worked at Bain Capital and Bose Corporation
- Areas of Interest: DevSecOps, Distributed Systems, Applied Artificial Intelligence, Embedded Systems
- LinkedIn: https://www.linkedin.com/in/hariharanragothaman/
- GitHub: https://github.com/hariharanragothaman

AGENDA

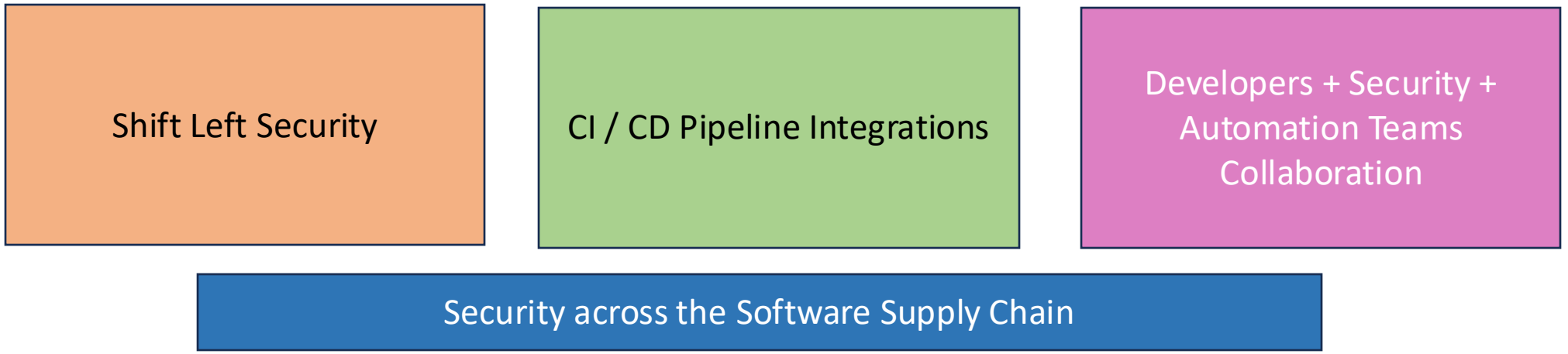
- Introduction
- Organizational Journey
- Why security is supremely important?
- Unified Deployment Model
- Governance in Unified Deployments
- Future Directions
- Impact and Lessons Learned - Key Takeaways

Organizational Journey

- Deployment Landscape before Unification
- Challenges Identified
- Goals for Changes – DevSecOps Journey



Organizational Journey - DevSecOps Drive

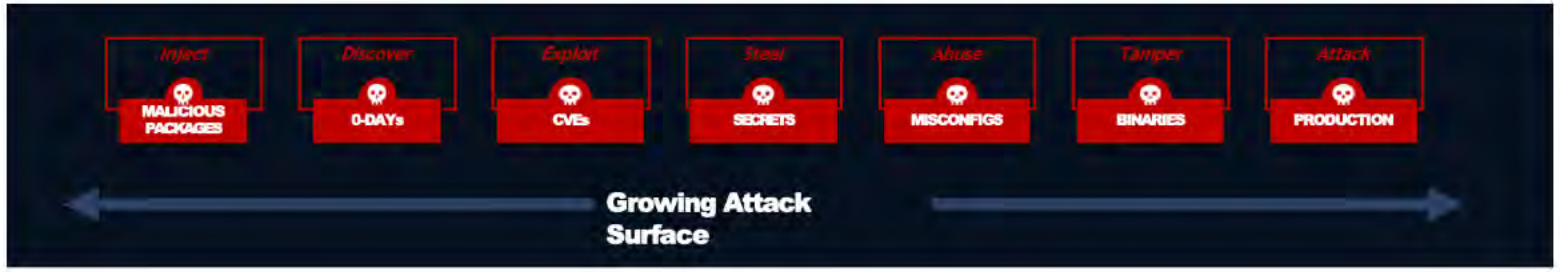


1. Rapid Increase in usage of OSS
2. Number of packages released per year also increasing.

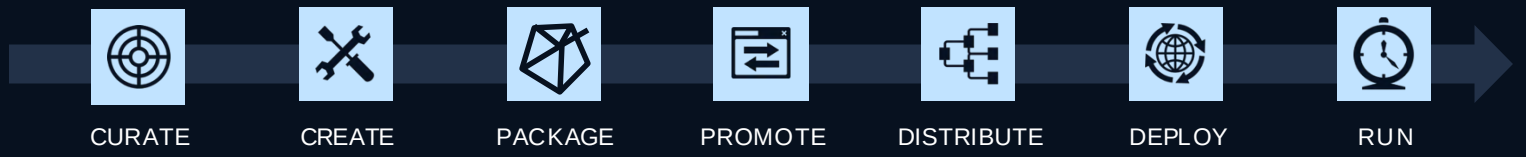
Why is security supremely important?

High Level Outlook of the SDLC Life Cycle

Most Applications run on OSS and 3rd party components at every stage of SDLC



THE SSC BEGINS
WHEN ANYTHING ENTERS



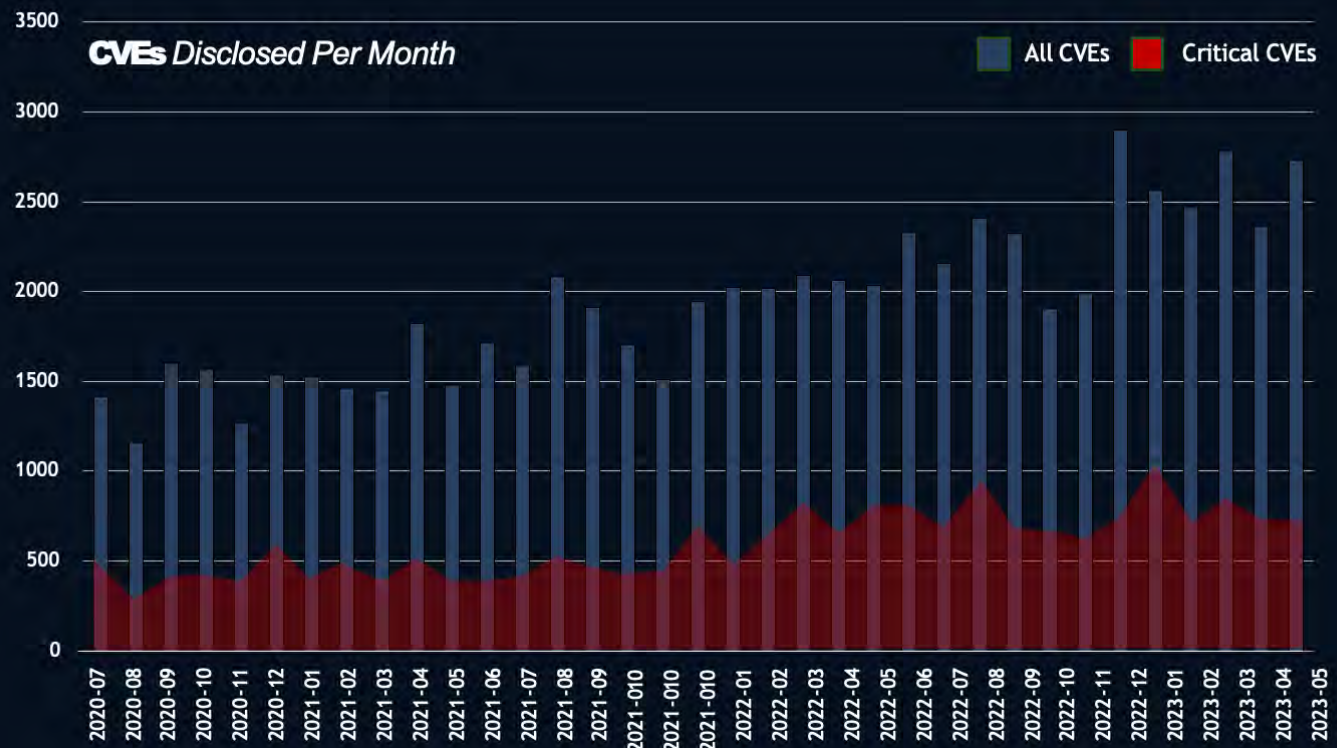
AND ENDS IN
PRODUCTION

PRODUCTION

Current Approach: Detect & Remediate

THE RATE OF
PUBLISHED CVEs
IS INCREASING

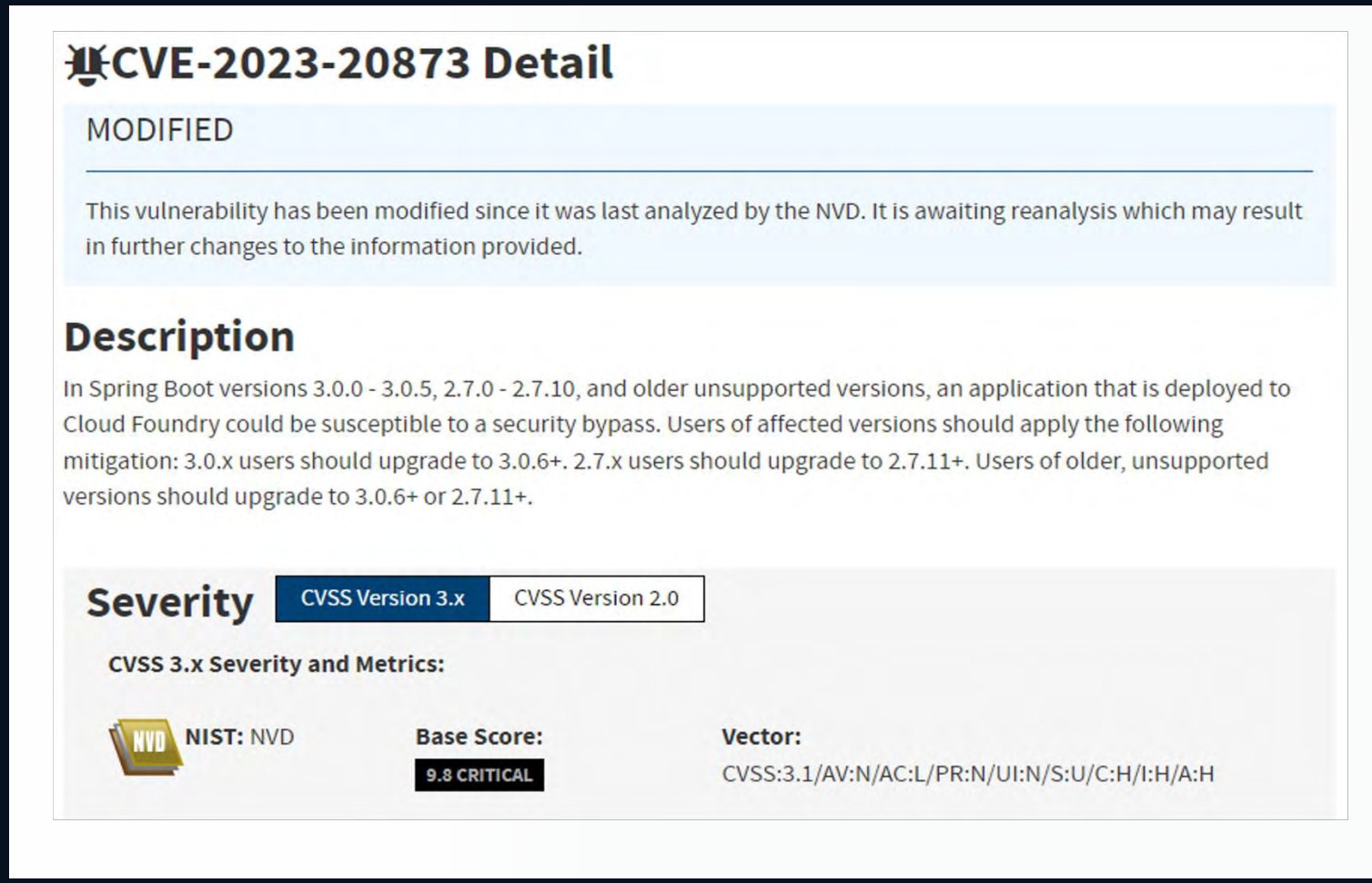
CREATING
CONSTANT PRESSURE
ON DEV & SECURITY
TEAMS



Many Critical CVEs
in common components *are*
NON-EXPLOITABLE
IN 99% of CASES

120,000 BINARIES
0 EXPLOITABLE CASES

spring boot



CVE-2023-20873 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Description

In Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, an application that is deployed to Cloud Foundry could be susceptible to a security bypass. Users of affected versions should apply the following mitigation: 3.0.x users should upgrade to 3.0.6+. 2.7.x users should upgrade to 2.7.11+. Users of older, unsupported versions should upgrade to 3.0.6+ or 2.7.11+.

Severity

CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD

Base Score: 9.8 CRITICAL

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

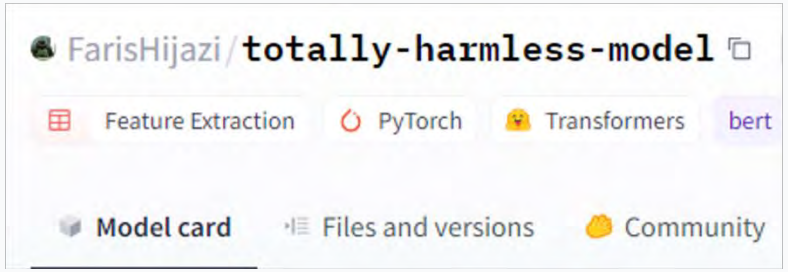
ML MODELS?

YET ANOTHER
MALICIOUS
PACKAGE!

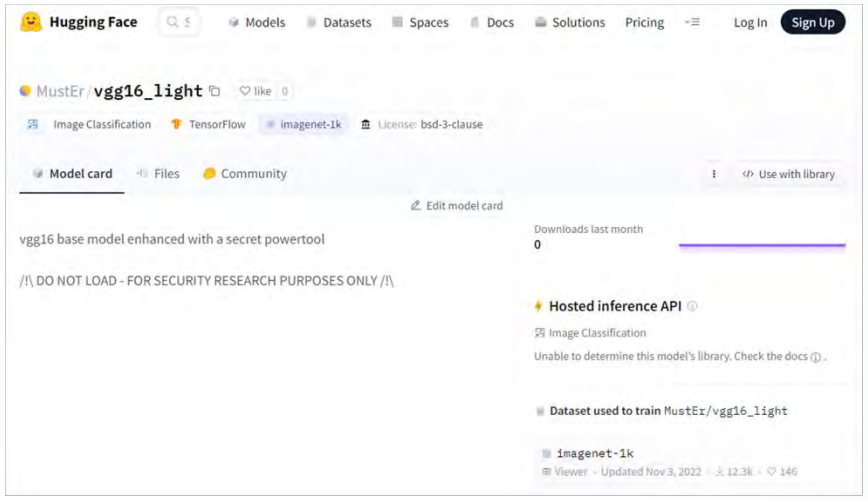
ML models can cause
MALICIOUS CODE EXECUTION
when loaded by Developer / Data Scientist

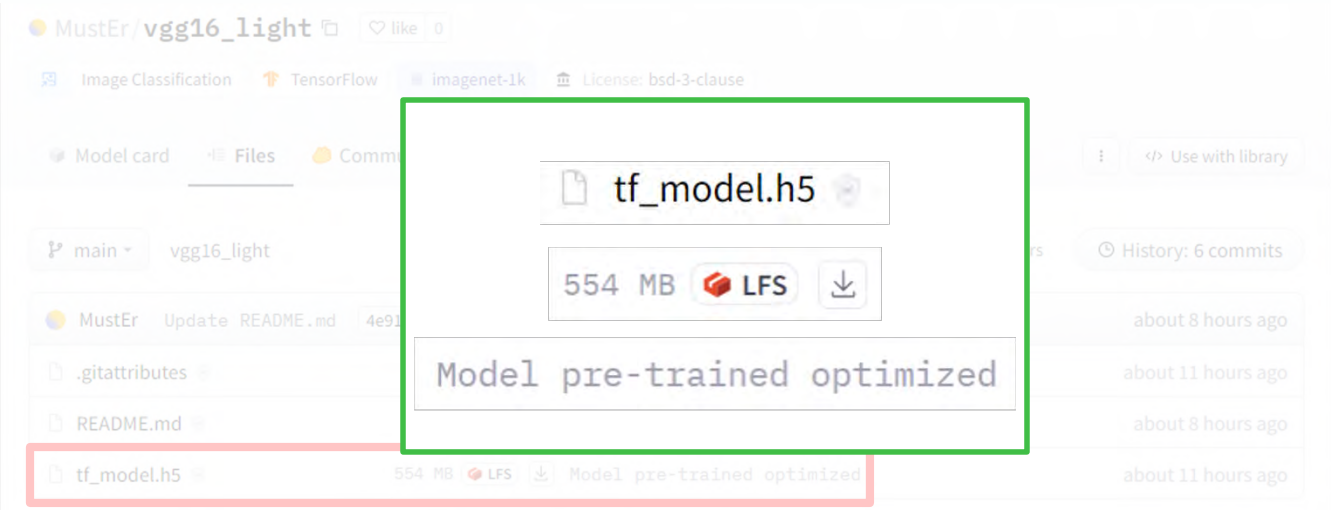
Public repositories
for models ARE NOW A TARGET

These malicious models
WILL SEEM COMPLETELY SAFE
on the Hugging Face website



A SUPPOSEDLY LEGITIMATE MODEL - JUST DATA, RIGHT?



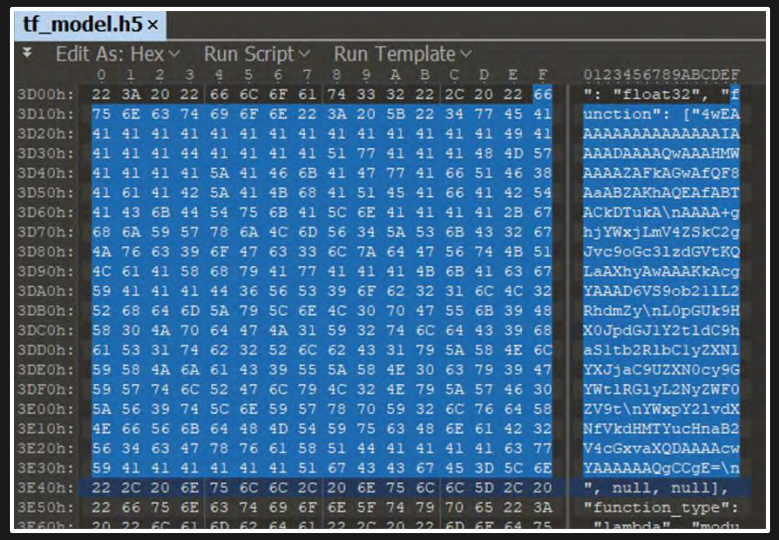


YET WHEN THE MODEL LOADS, MALICIOUS CODE EXECUTES

```
import tensorflow as tf
from keras.preprocessing import
image from keras.models import
load_model import numpy as np

# Load the model
model = load_model('vgg16_light/tf_model.h5')

img =
image.load_img("./cat.jpeg",target_size=(224,224))
img = np.asarray(img)
img = np.expand_dims(img,
axis=0) output =
model.predict(img)
if output[0][0] >
    output[0][1]: print("cat")
else:
    print('dog')
```

```
→  HF_demo_files python predict.py
2023-09-04 21:38:40.758644: I tensorflow/core/util/port.cc:110] oneDNN custom operations are on. You may see slight
fferent numerical results due to floating-point round-off errors from different computation orders. To turn them of
t the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-09-04 21:38:40.759786: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, G
ll not be used.
2023-09-04 21:38:40.783263: I ter
e used.
2023-09-04 21:38:40.783546: I ter
use available CPU instructions ir
To enable the following instructi
ropriate compiler flags.
2023-09-04 21:38:41.418666: W ter
rRT
WARNING:tensorflow:No training c
y.
1/1 [==============================
cat
→  HF_demo_files
```

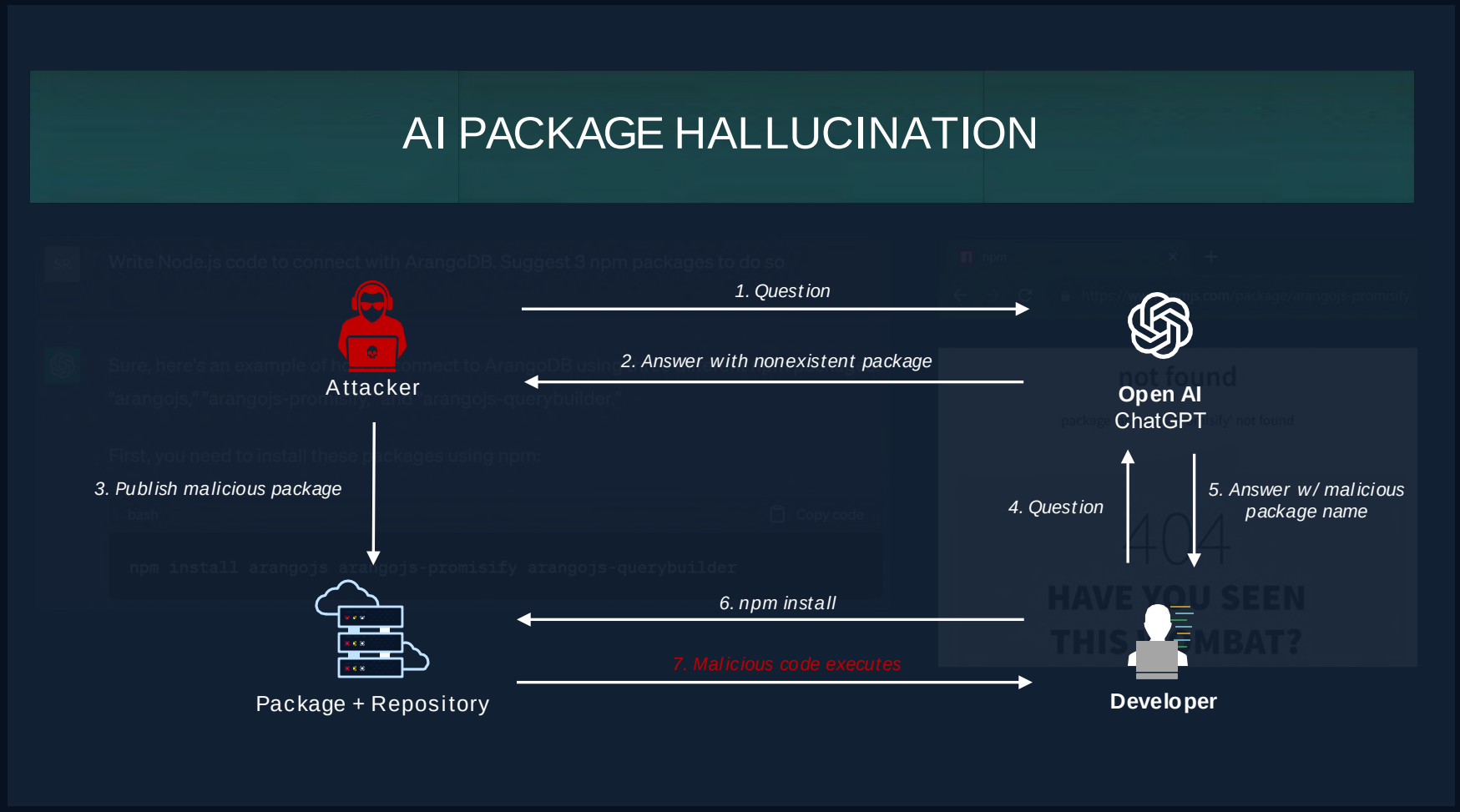
HOW? MALICIOUS CODE IS HIDDEN IN THE BINARY DATA




```
→  HF_demo_files python lambda_detection.py vgg16_light/tf_model.h5
Checking model vgg16_light/tf_model.h5

Found Lambda layer with name "output"
With body function:
Raw base64: 4wEAAAAAAAAAAAAAAIAAAADAAAAQwAAAHMWAAAAZAFkAGwAfQF8AaABZAKhAQEAfABTACkDTukA
AAAA+ghjYWxjLmV4ZSkC2gJvc9oGc3lzdGVtKQLaAXhyAwAAAKkAcgYAAAD6VS9ob211L2RhdmZy
L0pGUk9HX0JpdGJ1Y2tldC9haS1tb2RlbC1yZXN1YXJjaC9UZXN0cy9GYWtlRGlyL2NyZWF0ZV9t
YWxpY2lvdXNfVkdHMTYucHnaB2V4cGxvaXQDAAAAcwYAAAAAQgCCgE=

Decoded bytes: b'\xe3\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x00\x00\x03\x00\x00\x00C\x0
0}\x01|\x01\xa0\x01d\x02\xa1\x01\x01\x00|\x00S\x00)\x03N\xe9\x00\x00\x00\x00\xfa\x08calc.exe)\x02\xda\x02c
x00\x00\xa9\x00r\x06\x00\x00\x00\xfaU/home/davfr/JFROG_Bitbucket/ai-model-research/Tests/FakeDir/create_ma
\x00\x00s\x06\x00\x00\x00\x01\x08\x02\n\x01'

Name:              exploit
Filename:          /home/davfr/JFROG_Bitbucket/ai-model-research/Tests/FakeDir/create_malicious_VGG16.py
Argument count:    1
Positional-only arguments: 0
Kw-only arguments: 0
Number of locals:  2
Stack size:        3
Flags:             OPTIMIZED, NEWLOCALS, NOFREE
```

```
Constants:
   0: None
   1: 0
   2: 'calc.exe'
Names:
   0: os
   1: system
Variable names:
   0: x
   1: os
```

ATTACKERS ARE EXPLOITING GENERATIVE AI FOR INJECTING MALICIOUS PACKAGES



NOW EVEN GENERATIVE AI INTRODUCES SECURITY VULNERABILITIES!

Requesting generation of an endpoint that returns file contents

```
# Write an endpoint that returns a file from user input
@app.route("/file/<filename>")
def file(filename):
    return open(filename, "r").read()
```

The generated code is vulnerable to path traversal -
?filename=../../../../etc/passwd

Requesting generation of a secure endpoint also **doesn't** help

```
# Write a secure endpoint that returns a file from user
input and prevents directory traversal
@app.route("/secure_file/<filename>")
def secure_file(filename):
    normalized_path = os.path.normpath(filename)
    return open(normalized_path, "r").read()
```

Still vulnerable...
What about *?filename=/etc/passwd*

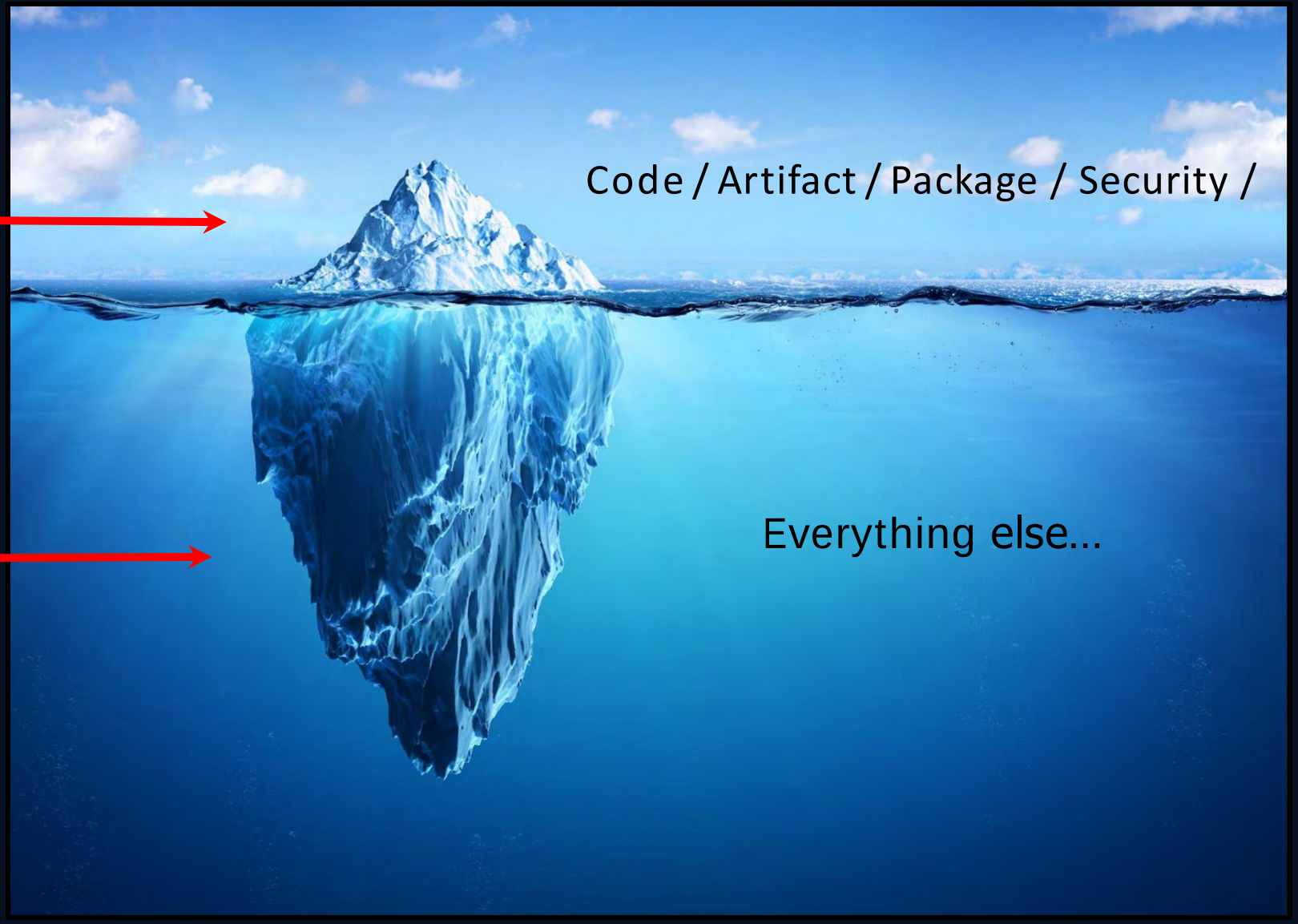
Software Supply Chain Security Types

- Known Vulnerabilities
  - *publicly disclosed* security bugs
- Unknown Vulnerabilities - Zero Day Attacks
  - attack on a vulnerability that was *not identified and fixed in time* to prevent the attack
- Non-Code Issues
  - human error can lead to malicious software injection attacks

How did this happen? - Software Dependencies

Code I wrote

Other stuff pulled in during the build



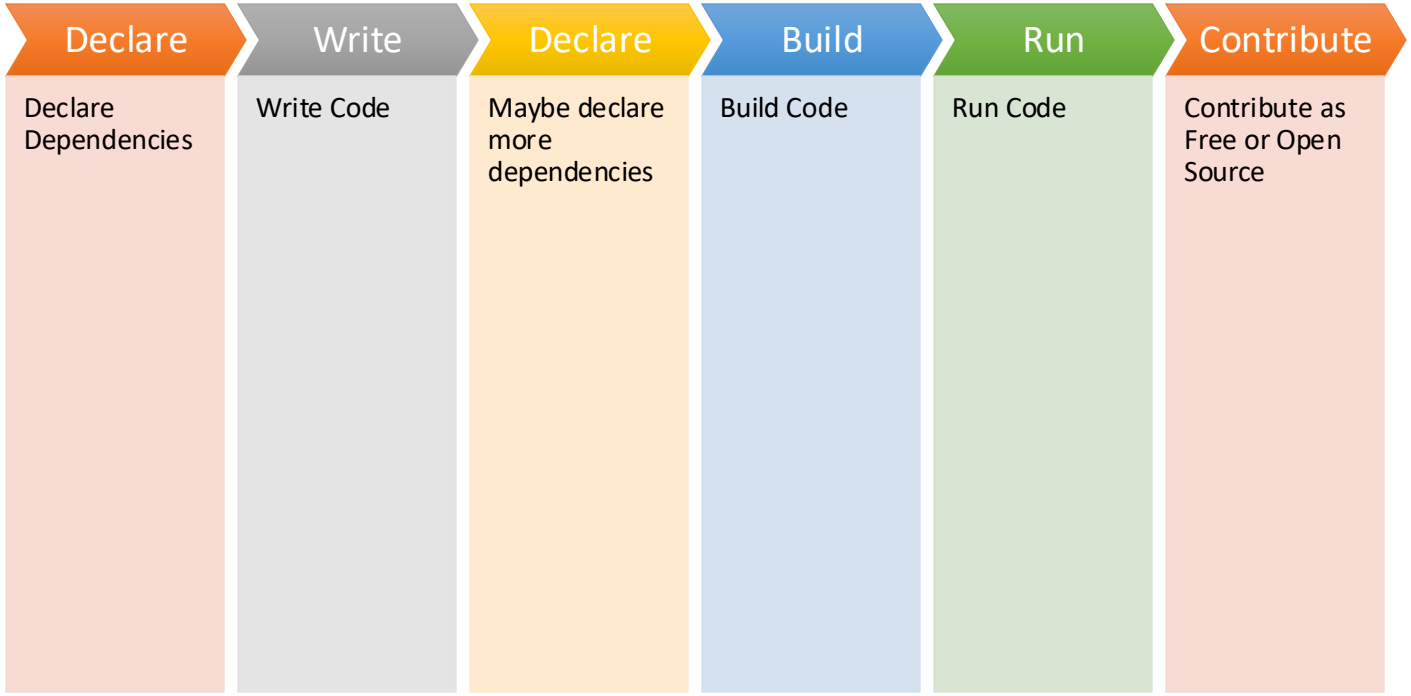
What can we do better?

1. Educate ourselves
2. Don’t rely solely on public repos
3. Manage dependencies!
4. Manage permissions!
5. Keep up with maintenance
6. Regularly scan your libraries & packages
7. Optimize Continuous Integration and Deployment Pipelines

Common Coding Insecurities

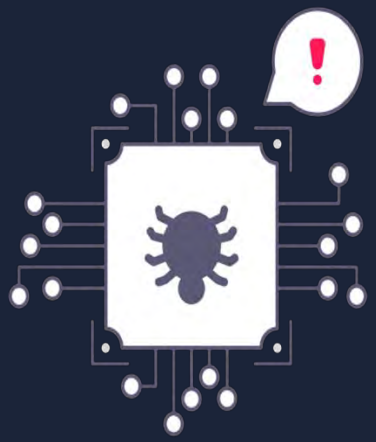
- Cross Site Scripting (XSS)
- SQL Injection
- LDAP Injection
- Cross Site Request Forgery (CSRF)
- ... others! (check out OWASP organization -- https://owasp.org)

How Developers Works? (Updated)



Software Supply Chain Threat Types

Unintentional Vulnerability



Security bug

Intentional Vulnerability

Backdoor

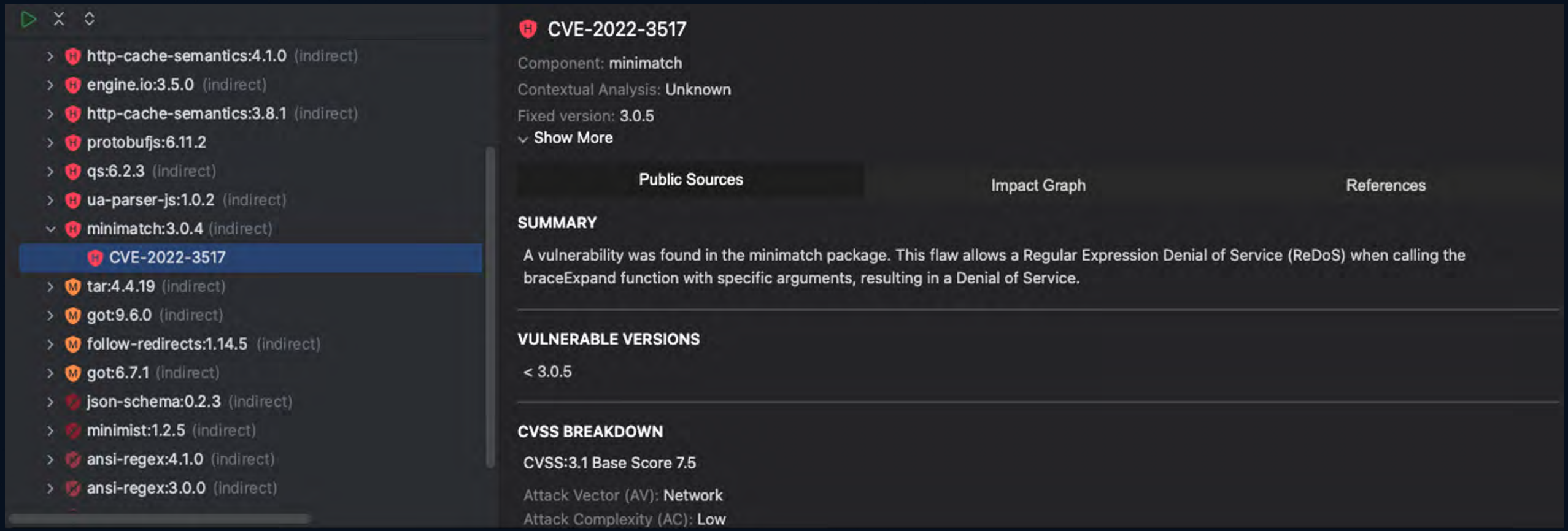
Malicious Components

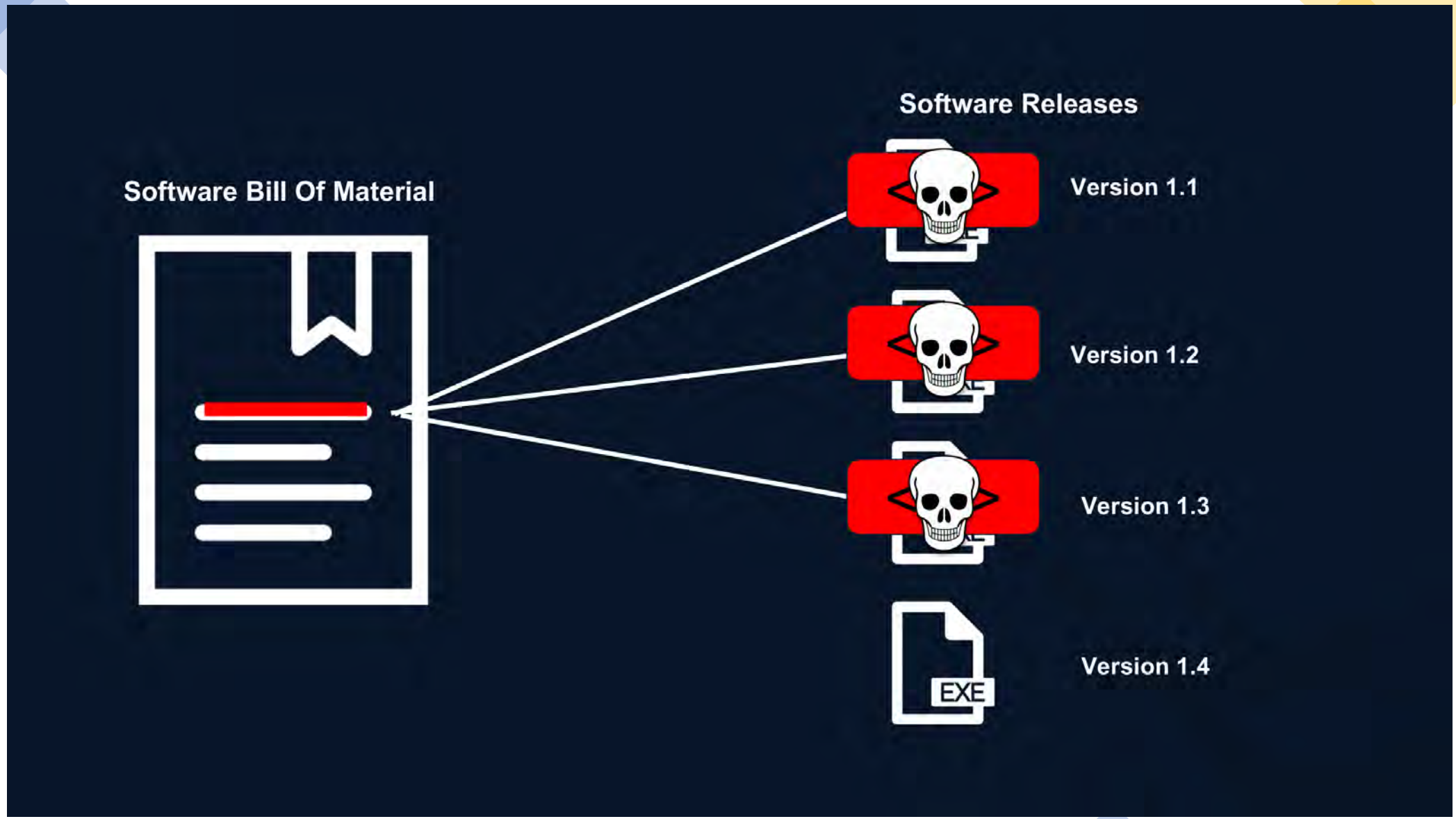
Malicious payload code

CVEs

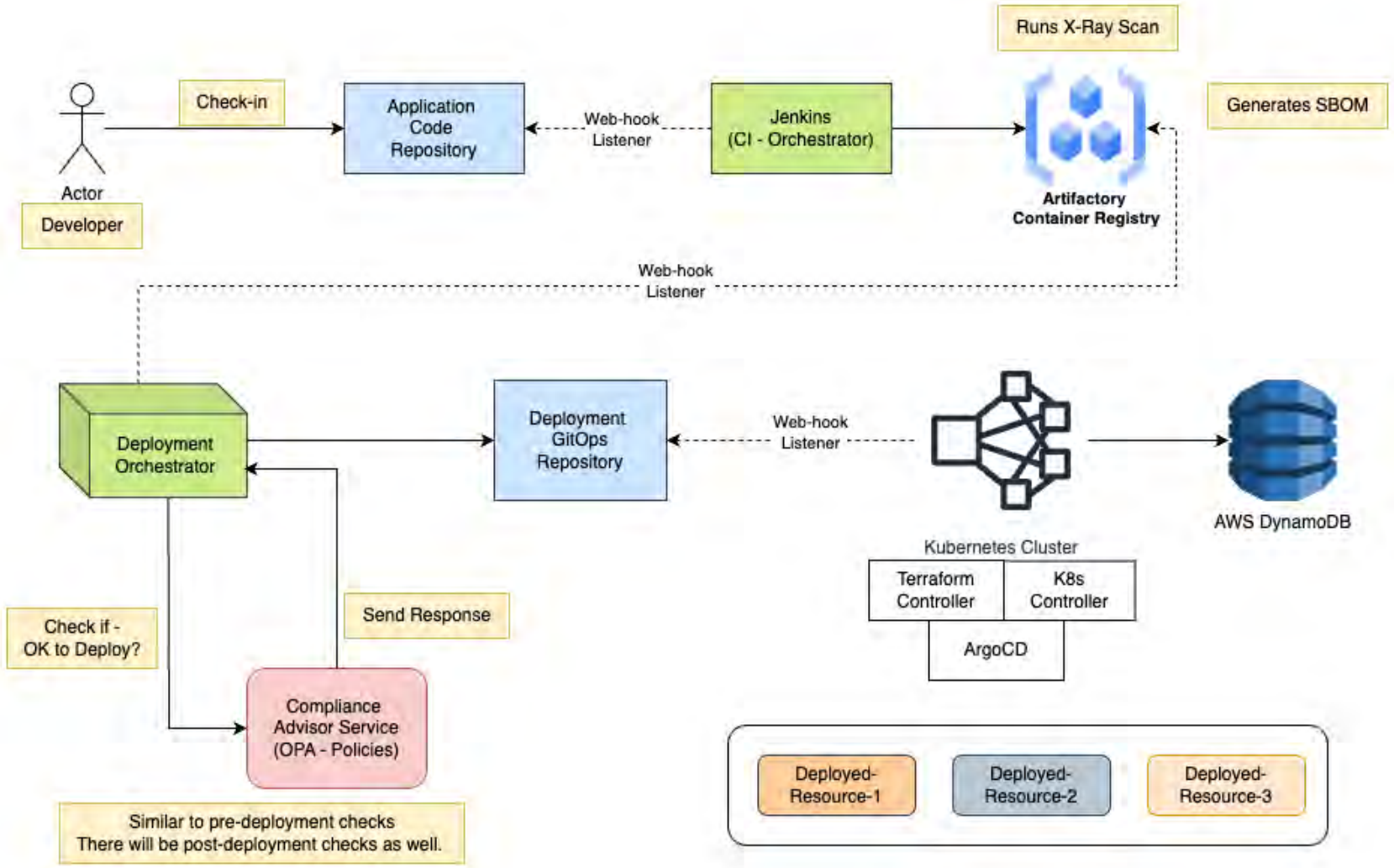
Not a CVE

Shifting Left to the Developer



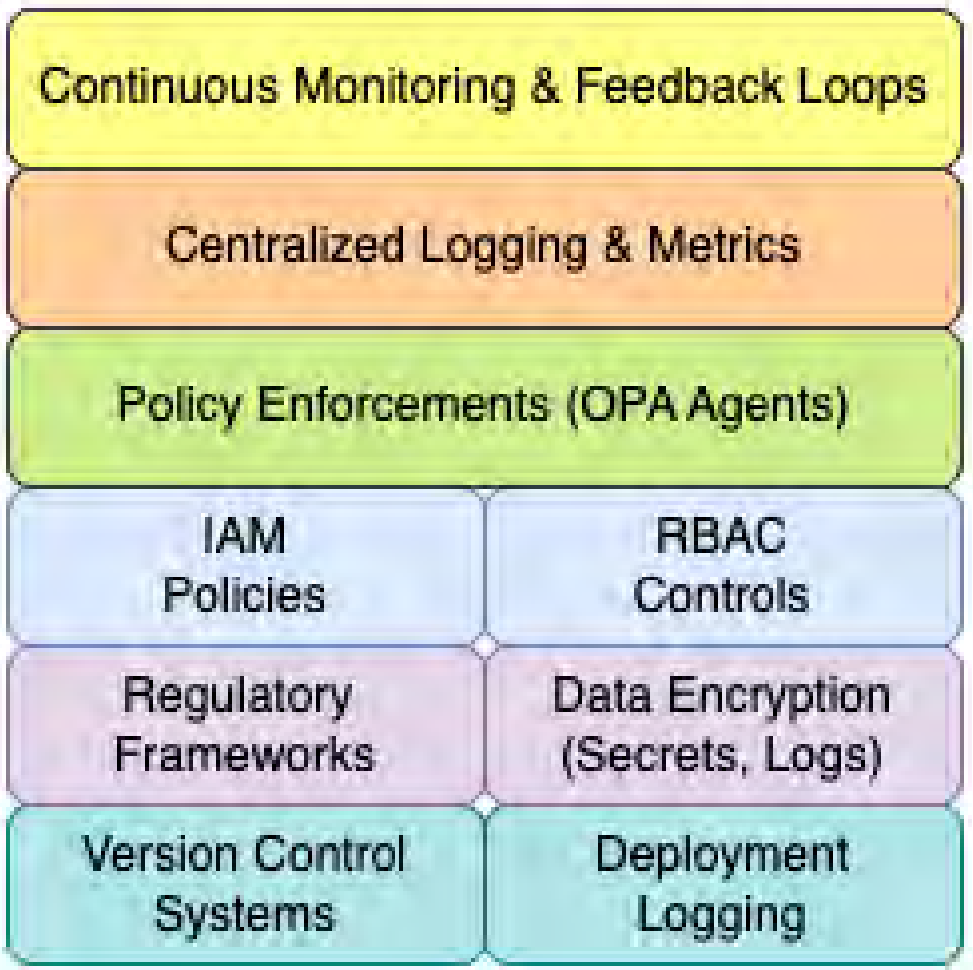

Software Releases
Software Bill Of Material
Version 1.1
Version 1.2
Version 1.3
Version 1.4
EXE

Unified Deployment Model



Governance in Unified Deployment Pipelines

- What is governance? – Why does it matter?
- Auditability
- Compliance with Standards
- Clear Ownership and Accountability



Future Directions - DevSecOps Pipeline Integration

1. **Plan & Code** – Threat modeling, secure coding guidelines.
2. **Build & Test** – Static/dynamic analysis, container security, automated tests.
3. **Release & Deploy** – Vulnerability scanning, environment scanning.
4. **Operate & Monitor** – Continuous security monitoring, anomaly detection.
5. **Feedback & Improve** – Retrospectives, updated policies and tooling.

Impact and Lessons Learned - Key Takeaways

1. **Security is Everyone's Responsibility** – Shift-left approach and collaboration are paramount.
2. **Automation & Integration** – Make security an intrinsic part of the CI/CD pipeline.
3. **Design for Failure** – Adopt AWS multi-AZ/region strategies, well-architected reviews, and chaos engineering.
4. **Continuous Improvement** – Learn, adapt, and iterate on security posture and reliability.

References

1. OWASP: https://owasp.org/
2. JFrog SwampUp Conferences
3. Kubernetes Blogs: https://kubernetes.io/blog/