Software Security Course

Lecture #3: Security on SDLC

C. Voliotis

Department of Informatics- University of Piraeus

The software development ecosystem

Part 1

How do we understand “security” on software development?

security-/sɪˈkjʊərɪti,sɪˈkjɔːrɪti/-Is the state of being free from danger or threat.

ασφάλεια η [asfália] **:**κατάσταση που χαρακτηρίζεται από την απουσία κινδύνου

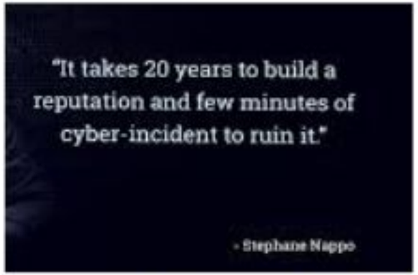
Safety is a property of the present: anything that was safe in the past or is designed to be safe in the future does not guarantee safety in the present.

Common Security Threats: Downtime, lack of privacy, sensitive info protection, data protection, malware, fraud, extortion etc. Most security threats originate from (or are tightly linked with) unsafe software “products”

There’s no such thing as 100% secure software

Safety is **a temporary result** of a set of processes, not an absolute achievement

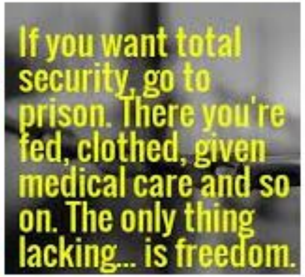
“Security is not a product, but a process.”



"It takes 20 years to build a reputation and few minutes of cyber-incident to ruin it."

- Stephane Nappo

CYBER SECURITY ISN'T EASY, BUT IT COMES DOWN TO THREE BASIC PRINCIPLES - PROTECT, DETECT, AND RESPOND.



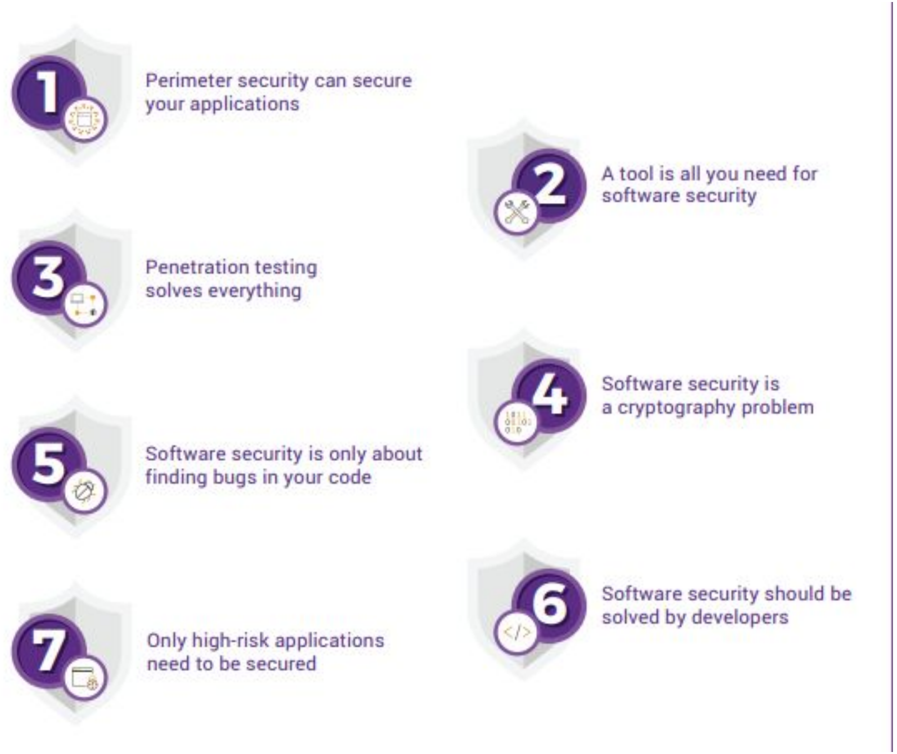
If you want total security, go to prison. There you're fed, clothed, given medical care and so on. The only thing lacking... is freedom.

House with No Door-VDGG(1970)

7 myths about security (link)

“Instead of securing broken software against attack, why don’t we just build software that’s not broken? That’s what software security is all about; building security into your software as it is being developed ”

“Building secure software means arming developers with tools and training, reviewing software architecture for flaws, checking code for bugs, and performing real security testing before release.”

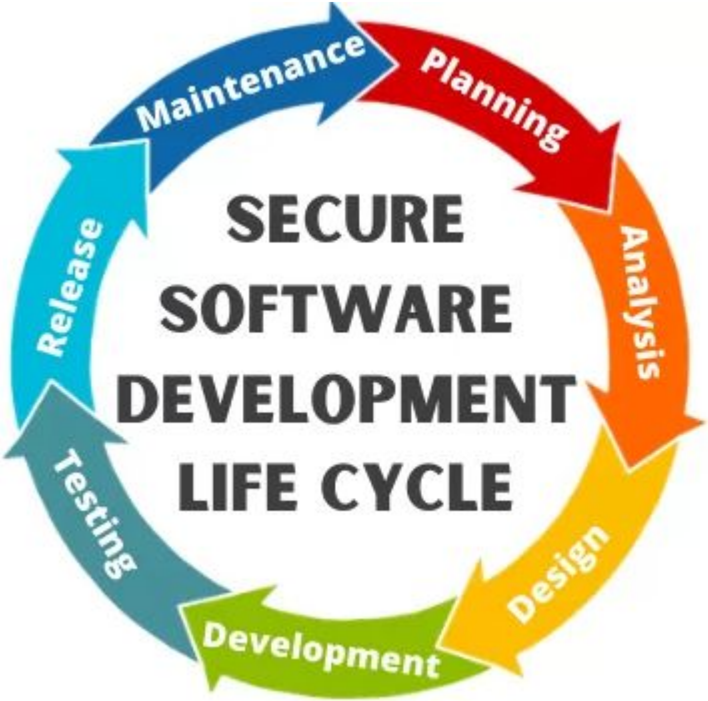


1. Perimeter security can secure your applications
2. A tool is all you need for software security
3. Penetration testing solves everything
4. Software security is a cryptography problem
5. Software security is only about finding bugs in your code
6. Software security should be solved by developers
7. Only high-risk applications need to be secured

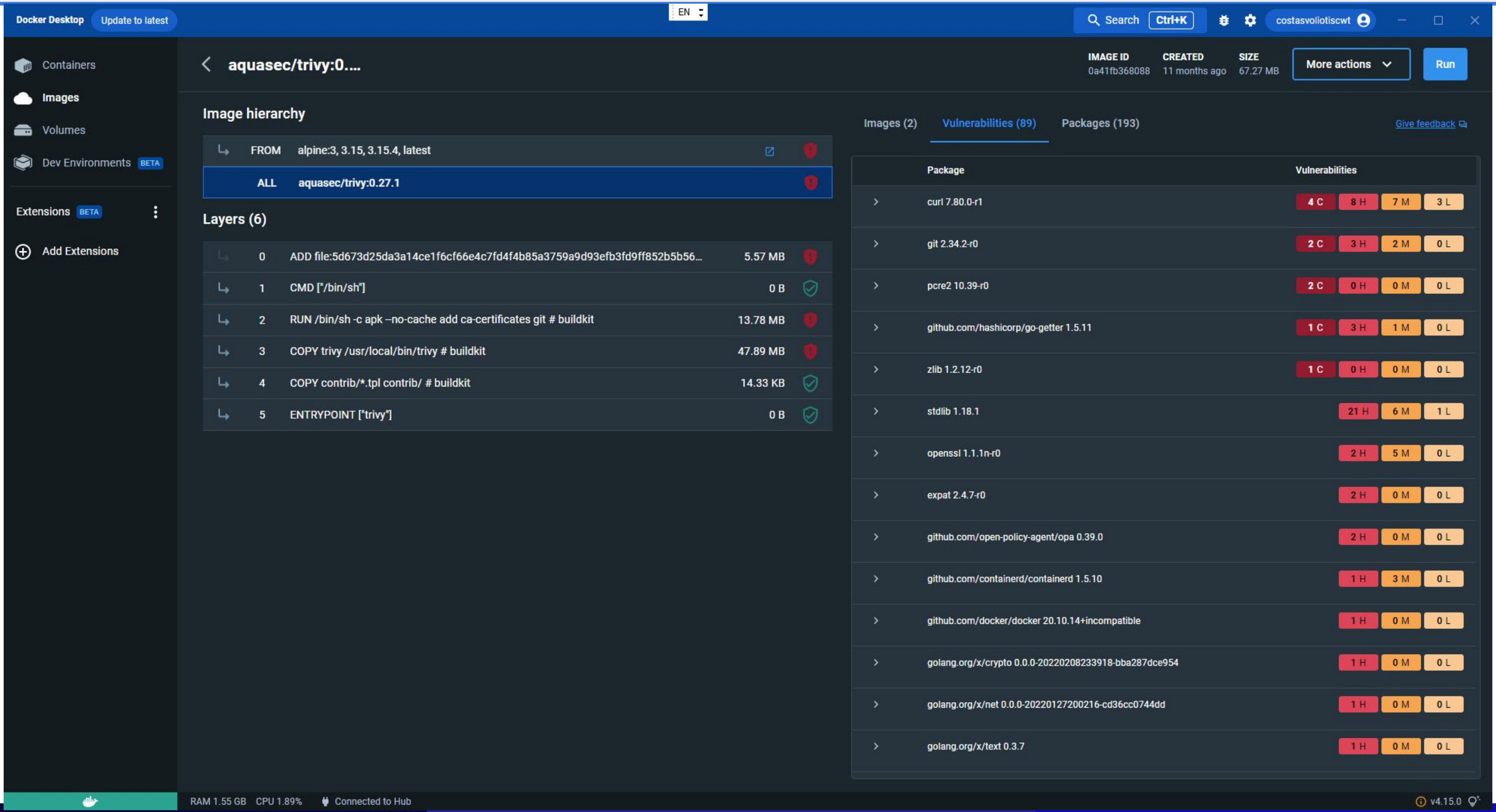
How software development is related to security?

1. The software implements products.
2. A software security audit always targets a single product, as opposed to an organization’s audit.
3. We “feel secure”(from software threats) if and only if all the software products we use are classified as“safe”.
4. **Simple definition**: When it comes to software, *security is a property of quality*. Quality refers to source code that supports a product’s value proposition without compromising consumer satisfaction, and without endangering the development unit’s business model

This course examines how we assess the security provided by a software product by analyzing the ***code base*** and the ***development process***



Review docker image security report



aquasec/trivy:0....

IMAGE ID 0a41fb368088 · CREATED 11 months ago · SIZE 67.27 MB

Image hierarchy

FROM	alpine:3, 3.15, 3.15.4, latest
ALL	aquasec/trivy:0.27.1

Layers (6)

0	ADD file:5d673d25da3a14ce1f6cf66e4c7fd4f4b85a3759a9d93efb3fd9ff852b5b56...	5.57 MB
1	CMD ["/bin/sh"]	0 B
2	RUN /bin/sh -c apk --no-cache add ca-certificates git # buildkit	13.78 MB
3	COPY trivy /usr/local/bin/trivy # buildkit	47.89 MB
4	COPY contrib/*.tpl contrib/ # buildkit	14.33 KB
5	ENTRYPOINT ["trivy"]	0 B

Package	Vulnerabilities			
curl 7.80.0-r1	4 C	8 H	7 M	3 L
git 2.34.2-r0	2 C	3 H	2 M	0 L
pcre2 10.39-r0	2 C	0 H	0 M	0 L
github.com/hashicorp/go-getter 1.5.11	1 C	3 H	1 M	0 L
zlib 1.2.12-r0	1 C	0 H	0 M	0 L
stdlib 1.18.1		21 H	6 M	1 L
openssl 1.1.1n-r0		2 H	5 M	0 L
expat 2.4.7-r0		2 H	0 M	0 L
github.com/open-policy-agent/opa 0.39.0		2 H	0 M	0 L
github.com/containerd/containerd 1.5.10		1 H	3 M	0 L
github.com/docker/docker 20.10.14+incompatible		1 H	0 M	0 L
golang.org/x/crypto 0.0.0-20220208233918-bba287dce954		1 H	0 M	0 L
golang.org/x/net 0.0.0-20220127200216-cd36cc0744dd		1 H	0 M	0 L
golang.org/x/text 0.3.7		1 H	0 M	0 L

The ecosystem

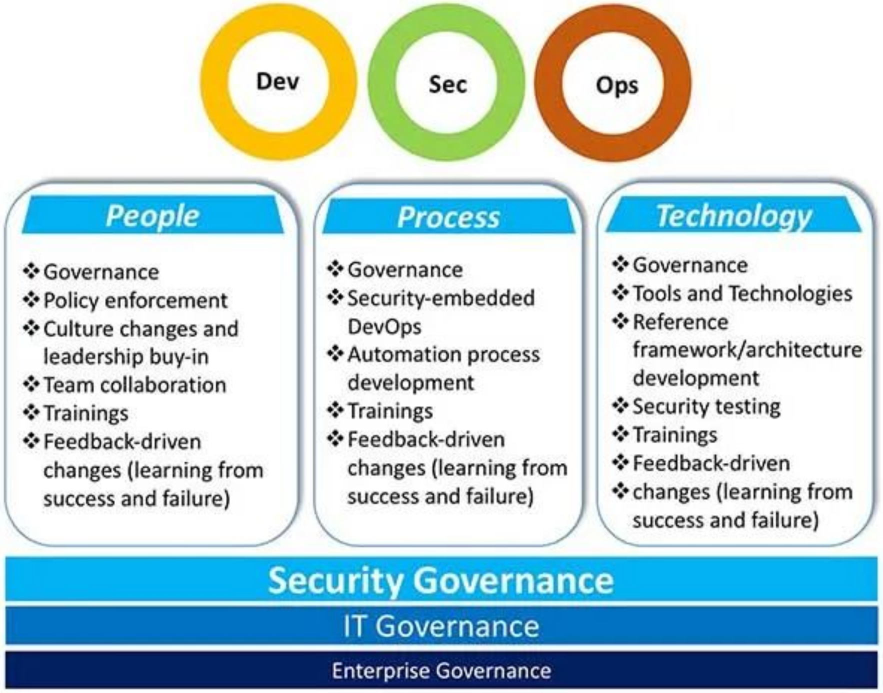
PPT Model

- **People:** The people involved are the stakeholders that want to implement the solution. They help in identifying the existing security threats and the relevant tools and procedures to be integrated to mitigate these threats.

- **Process:** The various processes that may be involved in this approach are Change management, Standard Operating Procedures (SOP), Segregation of Duties (SOD), Business continuity planning (BCP), etc.

- **Technology:** The technology required to automate various stages of security testing such as SAST, IAST, DAST, VAPT*, Deployment, etc. and to integrate them in the existing pipeline

link



People
- ❖Governance
- ❖Policy enforcement
- ❖Culture changes and leadership buy-in
- ❖Team collaboration
- ❖Trainings
- ❖Feedback-driven changes (learning from success and failure)

Process
- ❖Governance
- ❖Security-embedded DevOps
- ❖Automation process development
- ❖Trainings
- ❖Feedback-driven changes (learning from success and failure)

Technology
- ❖Governance
- ❖Tools and Technologies
- ❖Reference framework/architecture development
- ❖Security testing
- ❖Trainings
- ❖Feedback-driven
- ❖changes (learning from success and failure)

Security Governance

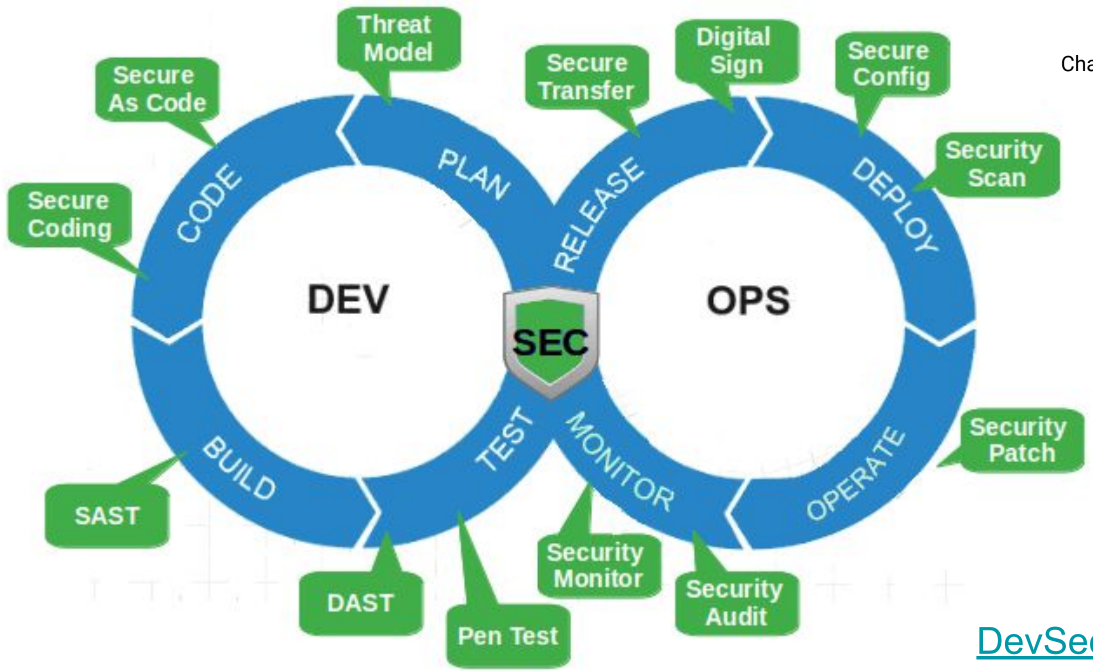
IT Governance

Enterprise Governance

* VAPT: Vulnerability Assessment and Penetration Test

When: Phases of security-quality review of a software product

INTERNAL AUDIT



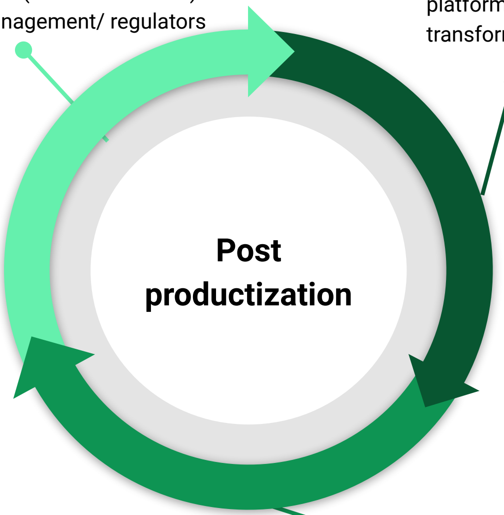
EXTERNAL AUDIT

CODE REFACTORING

Modifications, integration on a platform, digital transformation

DUE DILIGENCE

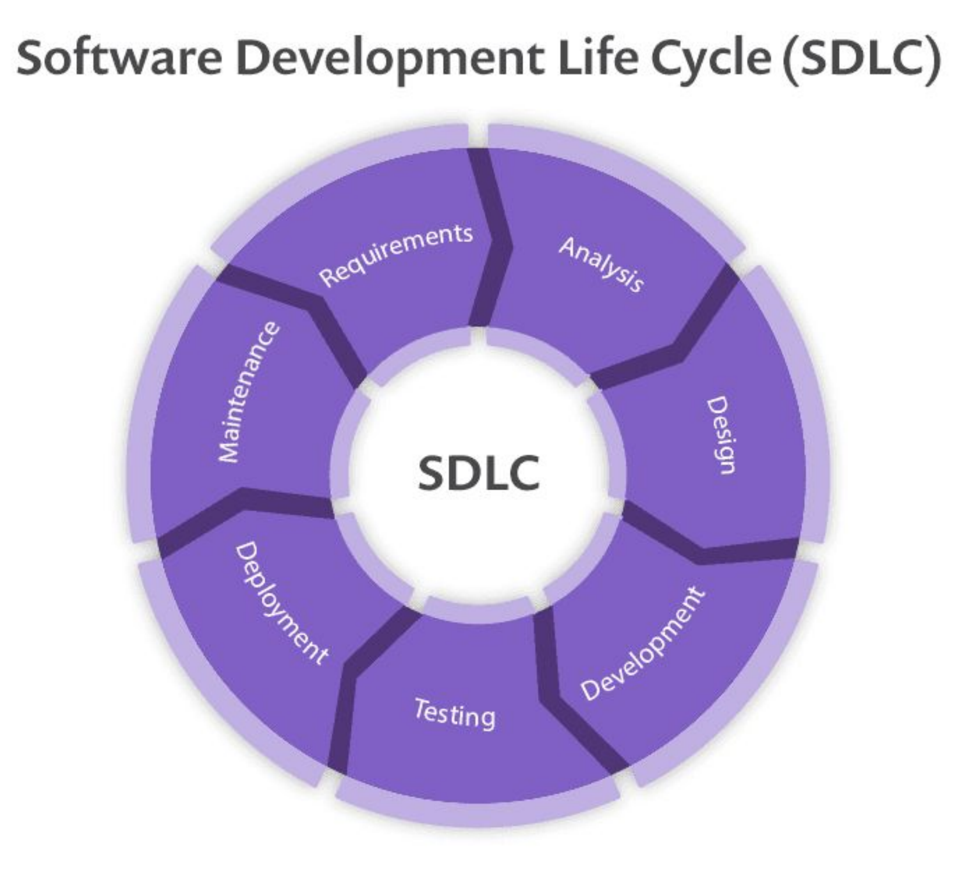
Acquisition(M&A/Investment)/ Change of management/ regulators

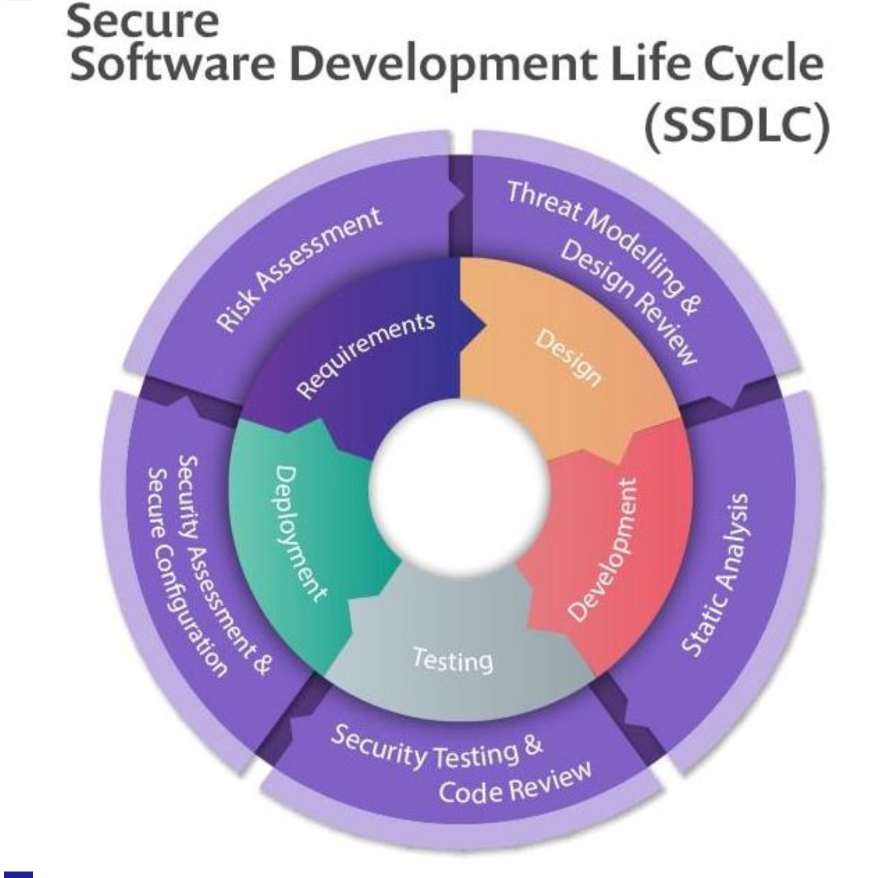


REGULATORS AUDIT
Deployment on a new market
Evaluation

DevSecOps Steps

SDLC vs SSDLC (more…)




Secure Software Development Life Cycle (SSDLC)
Risk Assessment
Threat Modelling & Design Review
Static Analysis
Security Testing & Code Review
Security Assessment & Secure Configuration
Requirements
Design
Development
Testing
Deployment

DevSecOps- Who integrates security controls?

Software developers : Accountable for the security and the quality of the commuted code

Software assets managers (Project Manager, Product Managers): Accountable for the reliability, the security and the cost of the development

Executives (CIO, CTO,CEO): Accountable for the business growth, product's hype and company's reputation

Regulatory firms: Accountable for the compliance of the deployed products and services with security quality regulations

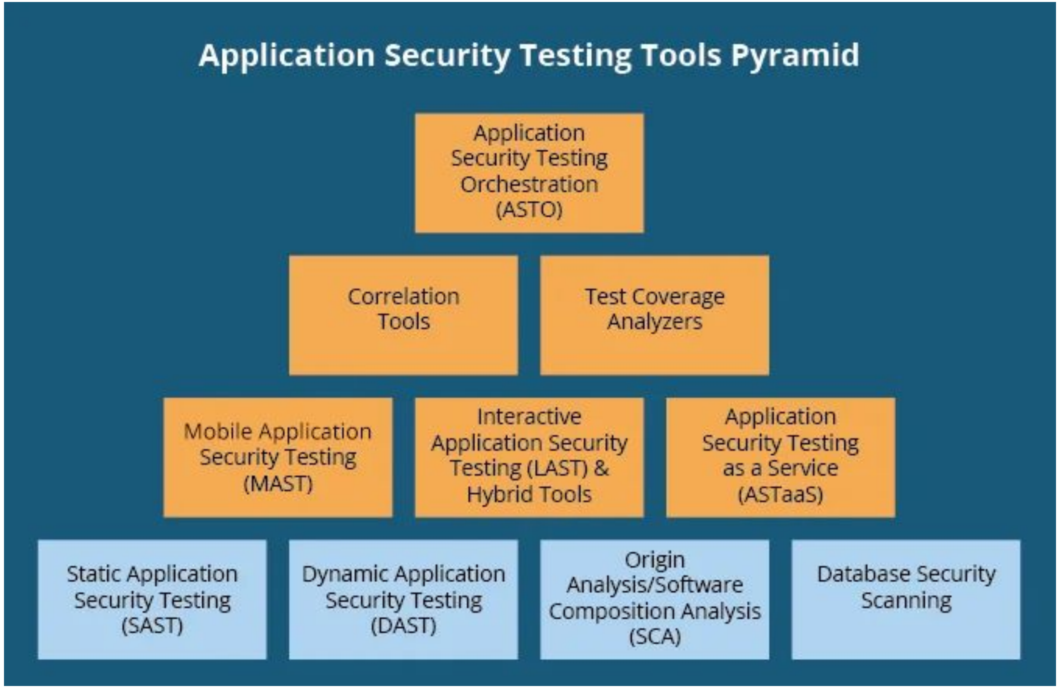
Buyers/Clients: Accountable for customer satisfaction and privacy

M&A advisors: Accountable for the mitigation of risk exposure post acquisition. Also accountable for the calculation of technical debt and the alignment of acquisitions prices and maintenance cost.

Role	Responsibilities
Security Officer	Architect of security review procedures Documents security requirements of project Makes sure all proj. deliverables meet the sec. req. Prepares the necessary docs Signs-off the deliverables of each phase
Code Auditor	Code auditing Functional testing Fills tracker with findings
Release Tester	Packaging tests Penetration testing Fills tracker with findings
Auditor	Checks that procedures are followed Checks that regulations are not violated Contributes legal / regulatory requirements to docs

Processes & Technologies Stack

- The goal is to explore all possible quality risks vulnerabilities in an application
- This is a test in breadth
- Methods of testing (white box / black box) depend on the details provided
- Tester produces report:
    - With a description of each vulnerability
    - With a rating of each vulnerability severity
    - With suggestions for risk mitigation
- The software may be setup in a testing environment
- Access to different user roles may be required



Application Security Testing Tools Pyramid

Application Security Testing Orchestration (ASTO)

Correlation Tools

Test Coverage Analyzers

Mobile Application Security Testing (MAST)

Interactive Application Security Testing (IAST) & Hybrid Tools

Application Security Testing as a Service (ASTaaS)

Static Application Security Testing (SAST)

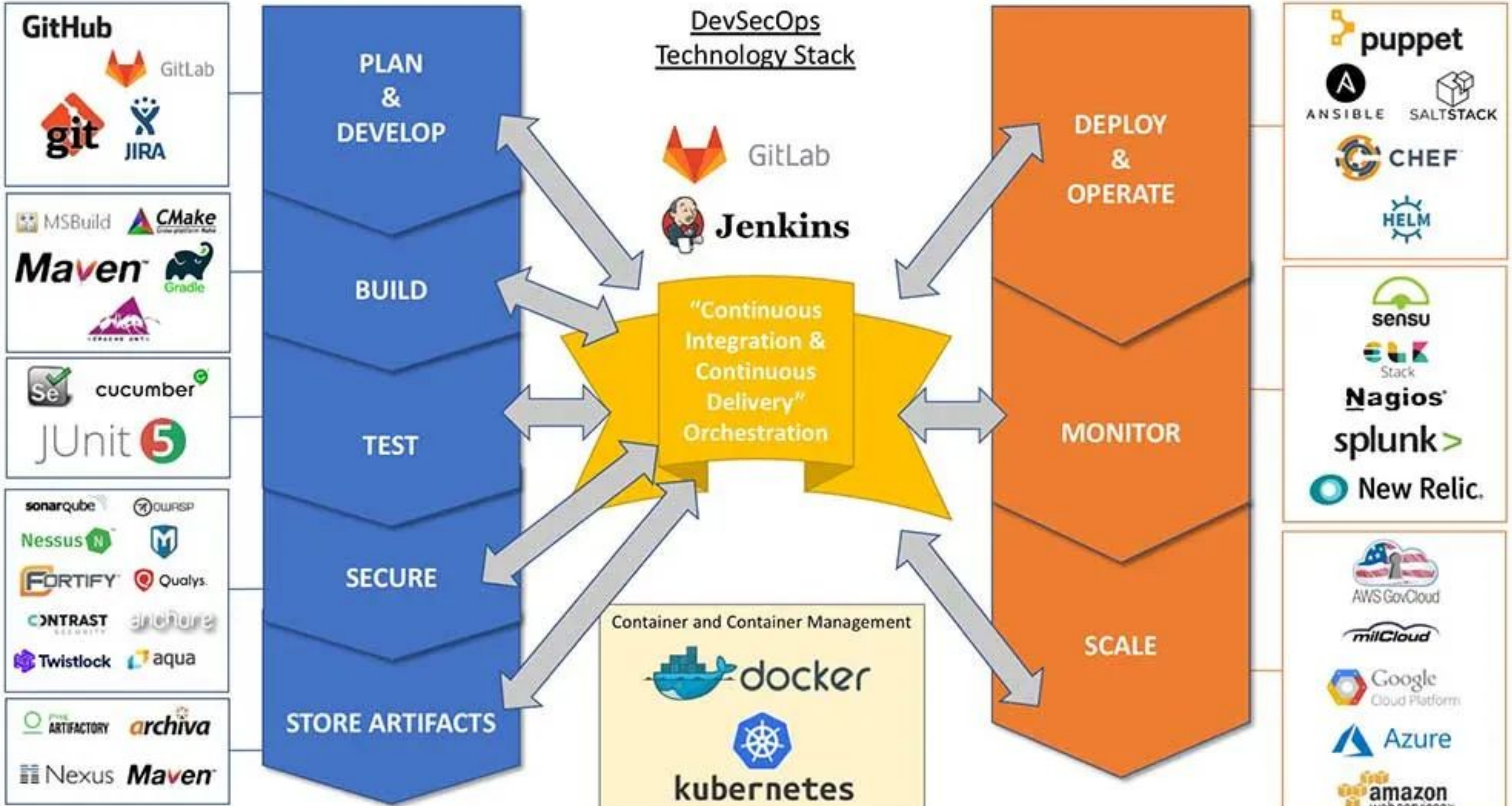
Dynamic Application Security Testing (DAST)

Origin Analysis/Software Composition Analysis (SCA)

Database Security Scanning

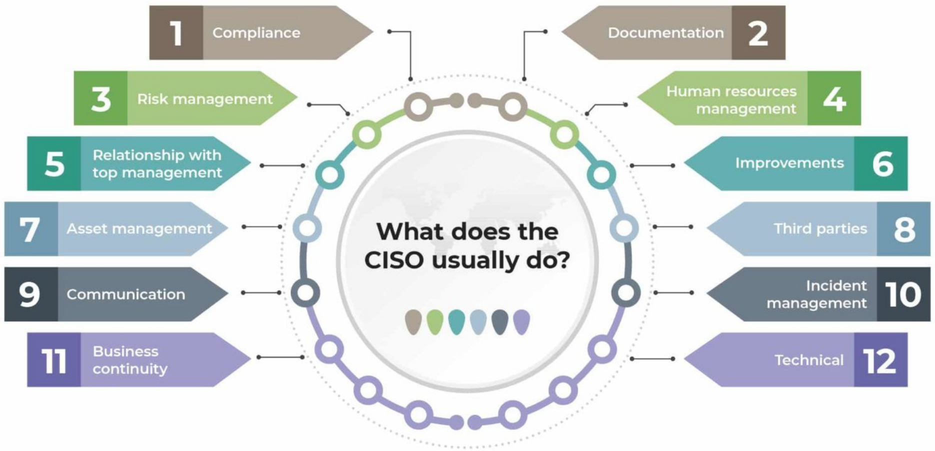
Processes & Technologies

- Bug tracking system
- Documentation Management System (Wiki etc.)
- Version control
- Development tools (compilers, IDEs, build tools etc.)
- Testing frameworks
- Continuous
- Integration tools



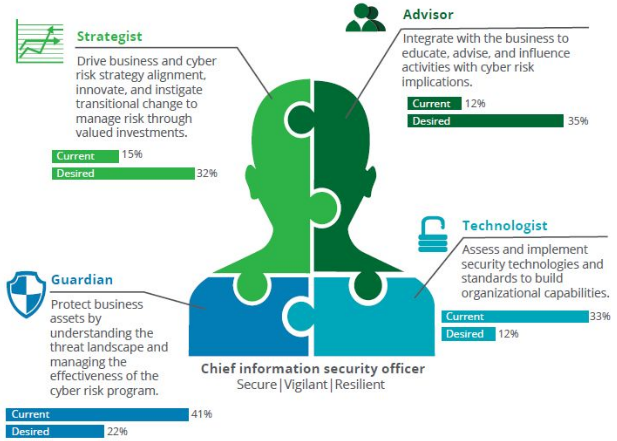
Decision making process:Challenges

1. **Cost:** Investing in security and quality can be expensive, especially when it involves hiring additional staff, purchasing new tools and technologies, and implementing new processes. Companies may be hesitant to invest in these areas unless they are absolutely necessary.
2. **Lack of Awareness:** Some companies may not fully understand the importance of investing in security and quality, or they may not be aware of the potential risks and consequences of not doing so. They may assume that their systems are secure and that any issues can be quickly addressed if they arise.
3. **Short-Term Thinking:** Some companies may prioritize short-term goals and immediate financial gains over long-term investments in security and quality. They may believe that they can save money by cutting corners in these areas, even if it means taking on more risk.
4. **Complexity:** Implementing effective security and quality measures can be complex and time-consuming. Companies may be hesitant to invest in these areas because they are unsure where to start or how to effectively address the issues.
5. **Lack of Accountability:** In some cases, companies may not have a clear understanding of who is responsible for security and quality. This can make it difficult to prioritize these areas and ensure that the necessary investments are being made.



More challenges..

1. **Rapidly evolving threats:** Cybersecurity threats are constantly evolving, and attackers are always finding new ways to breach security measures. This means that investing in cybersecurity requires ongoing monitoring and adaptation to keep up with the latest threats.
2. **Lack of skilled cybersecurity professionals:** The demand for skilled cybersecurity professionals far outstrips the supply, making it difficult for organizations to find and hire qualified experts. This can lead to a lack of effective cybersecurity measures and leave organizations vulnerable to attack.
3. **Balancing security with usability:** Security measures can be cumbersome and difficult to use, which can lead to user frustration and pushback. Finding the right balance between security and usability is a major challenge for cybersecurity investments.
4. **Limited budget:** Many organizations have limited budgets for cybersecurity investments, which can make it difficult to implement the necessary security measures to adequately protect against threats.
5. **Regulatory compliance:** Organizations are often subject to regulatory requirements that dictate certain cybersecurity measures, which can be complex and expensive to implement.
6. **Vendor management:** Many organizations rely on third-party vendors for various aspects of their cybersecurity, such as cloud services or software. Managing these vendors and ensuring they meet security standards can be a significant challenge.



Strategist

Drive business and cyber risk strategy alignment, innovate, and instigate transitional change to manage risk through valued investments.

Current	15%
Desired	32%

Advisor

Integrate with the business to educate, advise, and influence activities with cyber risk implications.

Current	12%
Desired	35%

Technologist

Assess and implement security technologies and standards to build organizational capabilities.

Current	33%
Desired	12%

Guardian

Protect business assets by understanding the threat landscape and managing the effectiveness of the cyber risk program.

Current	41%
Desired	22%

Chief information security officer

Secure | Vigilant | Resilient

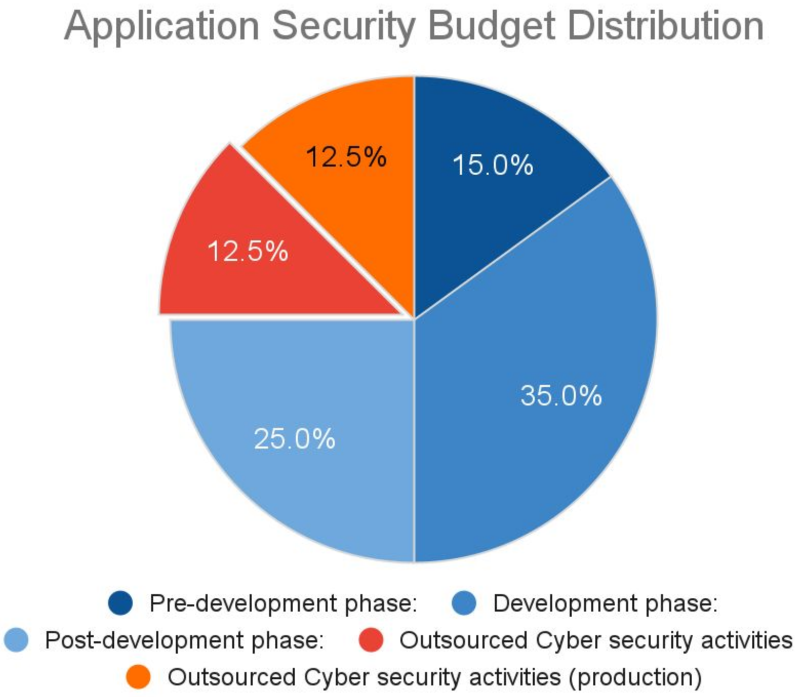
Source: Research from Deloitte's CISO Transition Labs.

Graphic: Deloitte University Press | DUPress.com

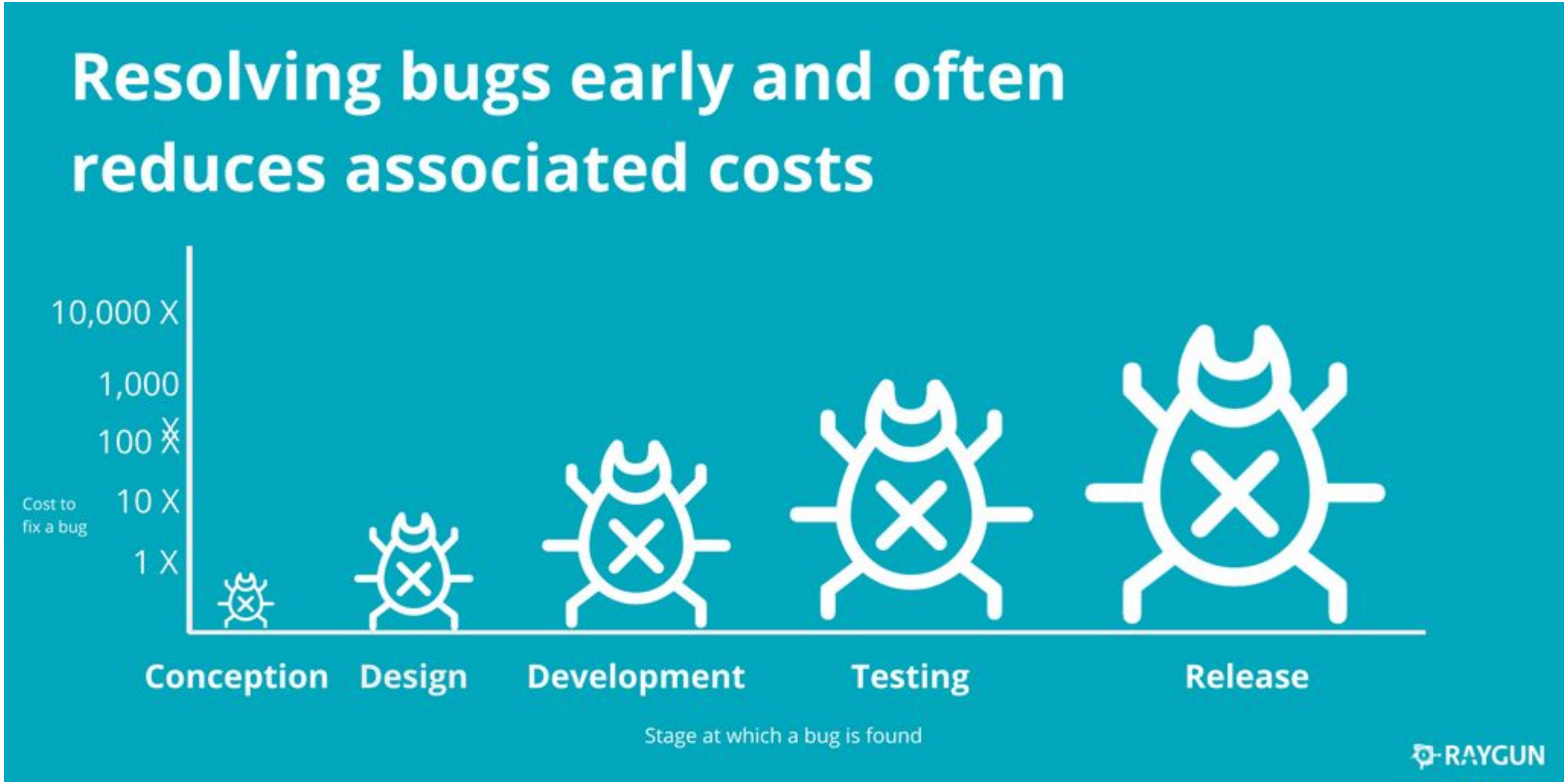
Security Risk Mitigation Budget distribution.

The distribution of budget spending for cybersecurity depends on the organization and the specific project. However, cybersecurity spending can be divided into three development phases: *pre-development, development,* and *post-development*. Deloitte reports that, organizations on average allocate about 20-30% of their cybersecurity budget towards *outsourcing activities*.

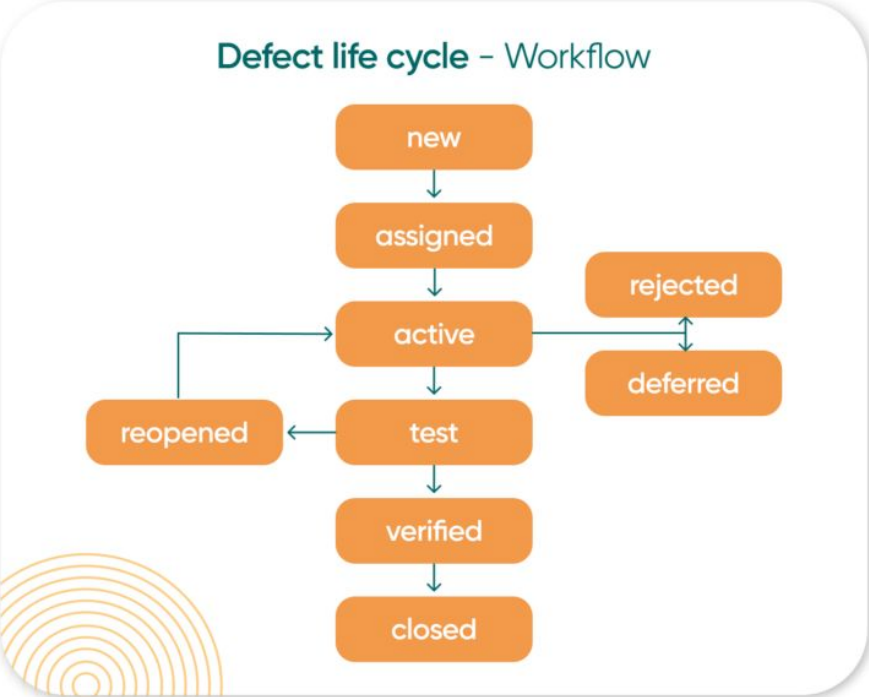
1. **Pre-development phase:** This phase includes activities such as risk assessment, threat modeling, and security design.
2. **Development phase:** This phase involves implementing security controls and integrating security measures into the development process. This phase includes activities such as secure coding, testing, and security reviews.
3. **Post-development phase:** This phase includes ongoing monitoring, incident response, and vulnerability management. The budget for this phase should be focused on continuous monitoring and improvement of security measures.
4. **Outsourced:** Split in two parts:testing+production

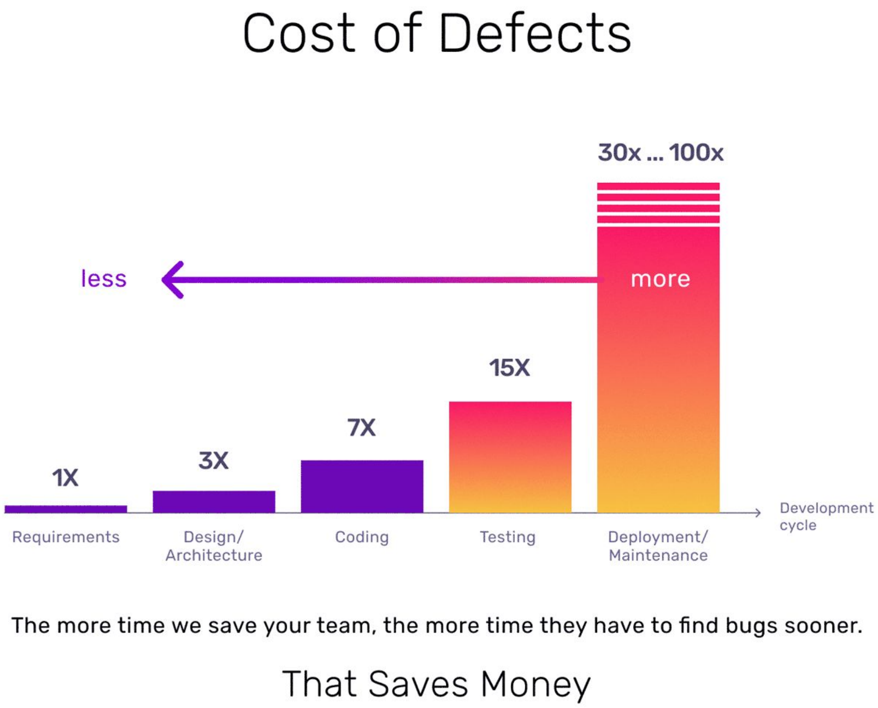


Cost of fixing a bug-the classic view



Cost of fixing a bug (more..)





How is this impact a software product reliability

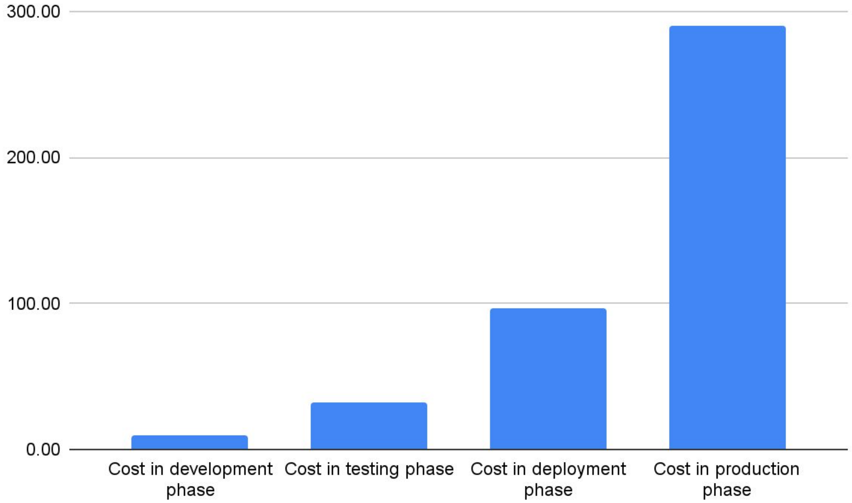
Cost of a bug in hours (dev phase)	10
Cost in development phase	10.00
Cost in testing phase	32.28
Cost in deployment phase	96.84
Cost in production phase	290.52

Cost of bug resolution per SDLC phase

Budget for Code Quality (incl. application security)	$250,000		Resolved bugs
% spend in pre-dev/dev phases	50%	$125,000	250
% spend in testing/deployment phases	25%	$62,500	39
% spend in deployment phase (outsourcing)	13%	$31,250	6
% spend in production phase (outsourcing)	13%	$31,250	2

Resolved bugs per SDLC phase for a fixed budget

This is a very conservative scenario. The cost is increased 1-30



Detected bugs per development stage

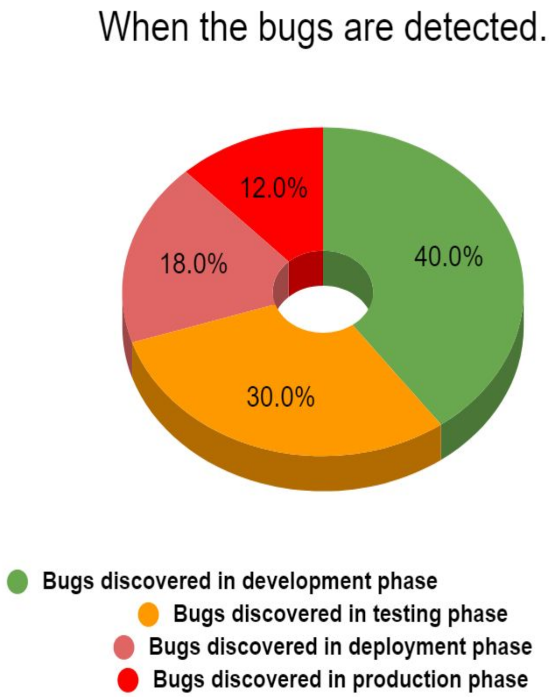
The percentage of bugs and security issues discovered during different phases of software development can vary depending on several factors such as the complexity of the software, the quality of the development process, and the testing methodologies employed.

<u>During development phases:</u> Bugs and security issues are often caught early in the development process when developers are writing code and testing their work. This is the phase where the majority of the issues are usually caught,. This phase typically accounts for around 50% of the total issues discovered.

During testing: Testing is a crucial phase of software development where software is tested for functionality, performance, and security. During this phase, additional bugs and security issues can be discovered and addressed. Depending on the quality of the testing process, this phase can account for up to 30% of the total issues discovered.

During deployment: Once the software is deployed, it is exposed to a wider range of environments and use cases. This can uncover issues that were not detected during development or testing. This phase typically accounts for around 10-20% of the total issues discovered.

Post-production: Even after software is deployed, issues can still be discovered by end-users or through ongoing monitoring and maintenance. This phase typically accounts for the remaining 10% of the total issues discovered.

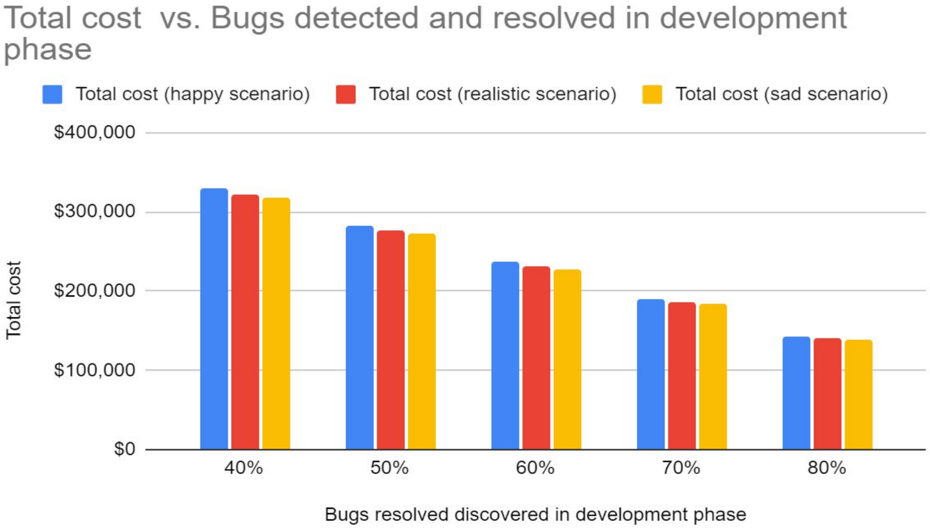


The goal of “shift-left” trend for code quality and security?

Cost for 100 bugs

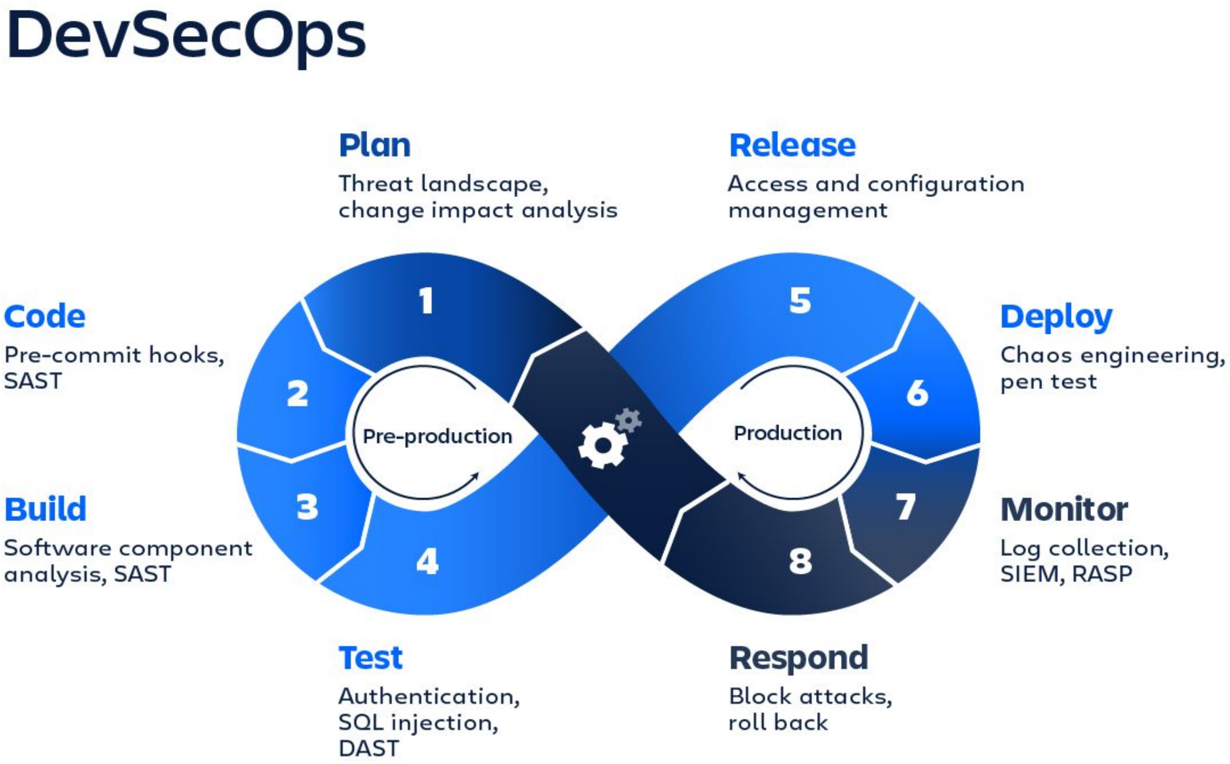
Bugs discovered and resolved in development phase	Cost increase	40%	50%	60%	70%	80%	40% vs 80%
Total cost (Sad scenario)	10-32-96-290	$329,885	$283,238	$236,590	$189,943	$143,295	**43.44%**
Total cost (Realistic scenario)	10-31-94-282	$320,938	$275,782	$230,625	$185,469	$140,313	**43.72%**
Total cost (Happy scenario)	10-31-93-278	$317,183	$272,653	$228,122	$183,592	$139,061	**43.84%**

that is all about!


Total cost vs. Bugs detected and resolved in development phase

It must be noticed that all scenarios converge to almost the same balance between early and late bugs resolution

Integrating security into the SDLC (more..)



DevSecOps

Plan
Threat landscape, change impact analysis

Code
Pre-commit hooks, SAST

Build
Software component analysis, SAST

Test
Authentication, SQL injection, DAST

Release
Access and configuration management

Deploy
Chaos engineering, pen test

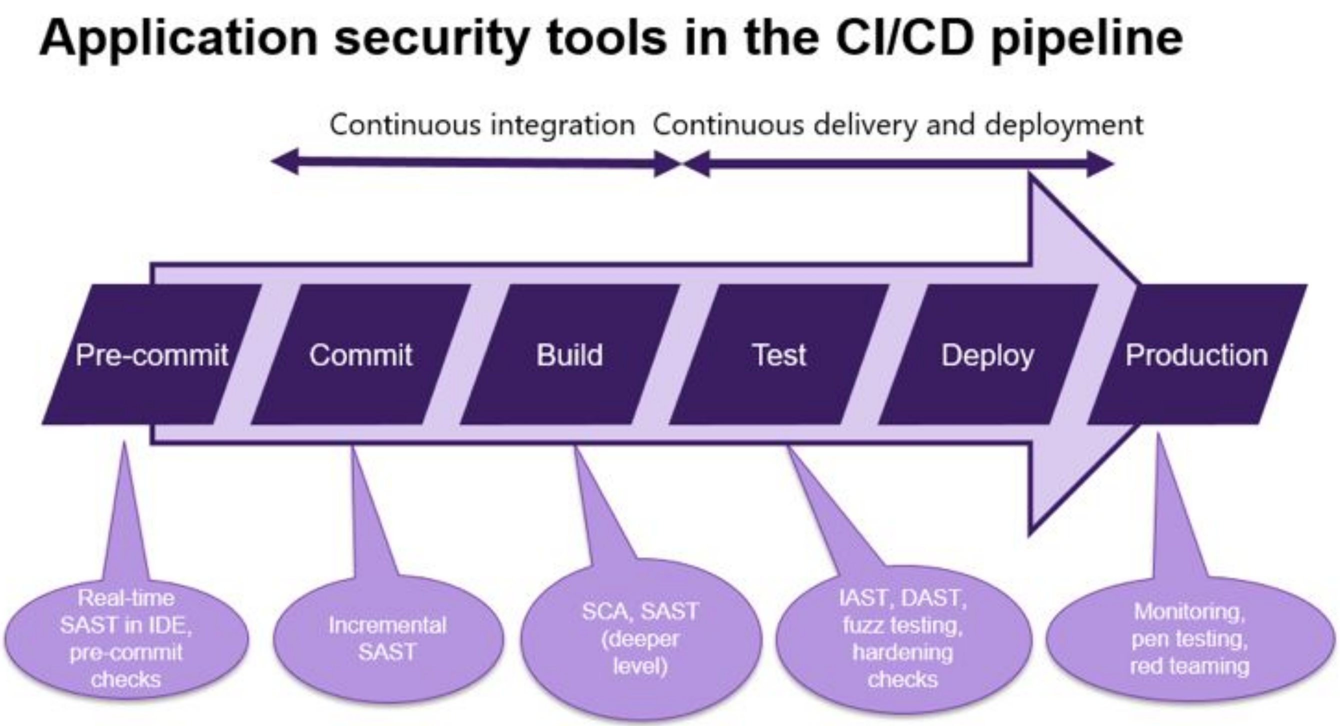
Monitor
Log collection, SIEM, RASP

Respond
Block attacks, roll back

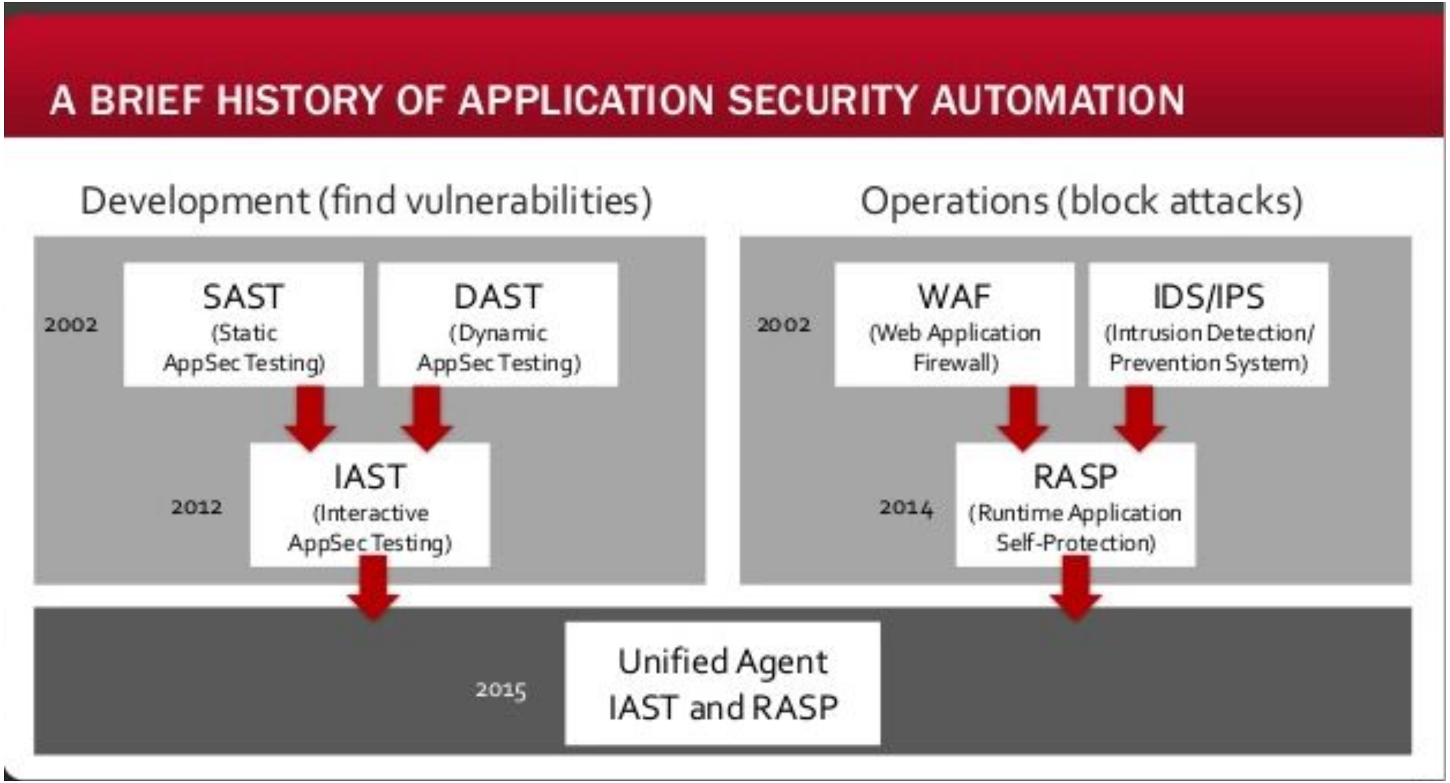
Building security in SDLC

Part 2

How do we fit security procedures into the SDLC?


Application security tools in the CI/CD pipeline
Continuous integration
Continuous delivery and deployment
Pre-commit
Commit
Build
Test
Deploy
Production
Real-time SAST in IDE, pre-commit checks
Incremental SAST
SCA, SAST (deeper level)
IAST, DAST, fuzz testing, hardening checks
Monitoring, pen testing, red teaming

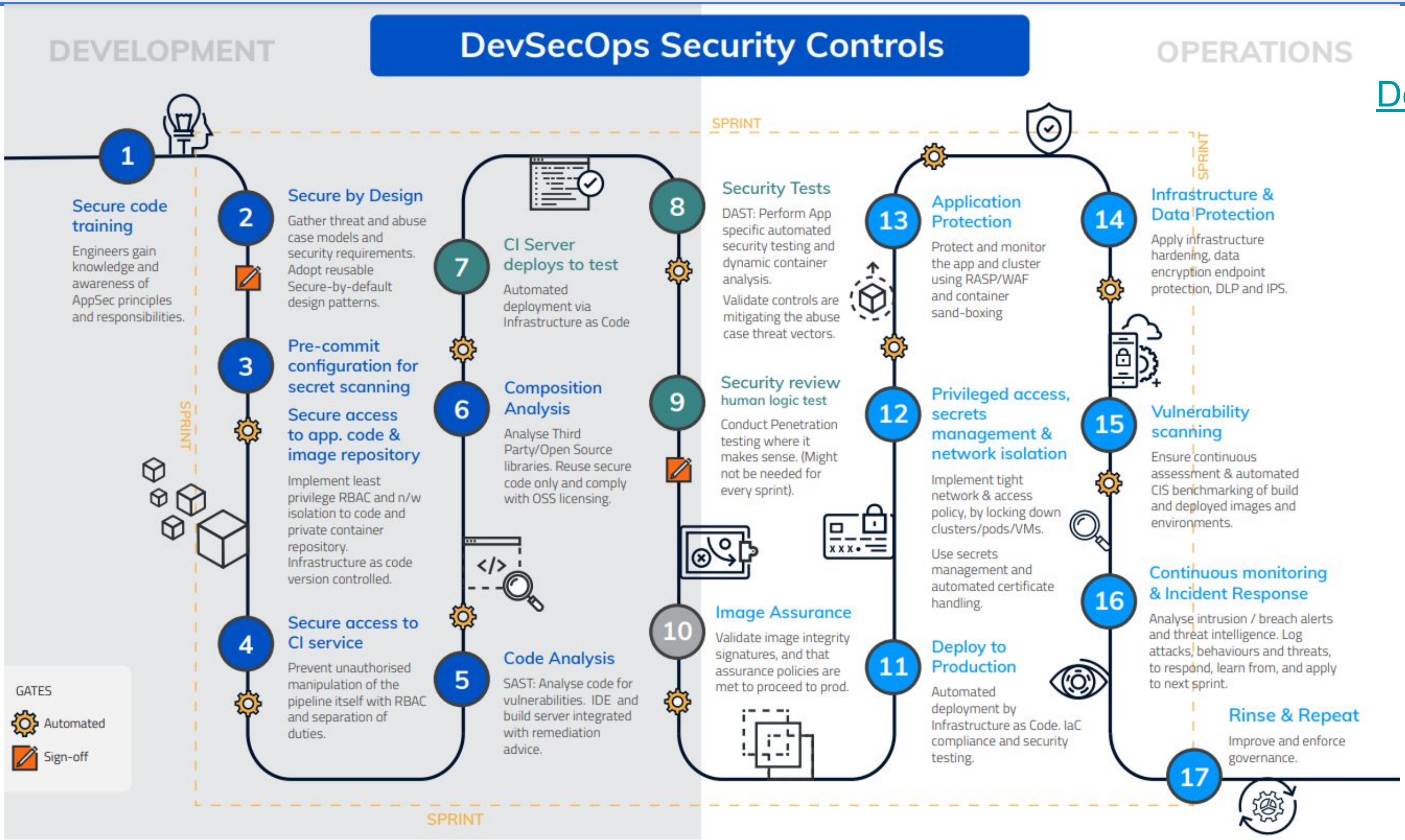
SAST, DAST, IAST, RASP


A BRIEF HISTORY OF APPLICATION SECURITY AUTOMATION
Development (find vulnerabilities)
Operations (block attacks)
2002
SAST
(Static AppSec Testing)
DAST
(Dynamic AppSec Testing)
2012
IAST
(Interactive AppSec Testing)
2002
WAF
(Web Application Firewall)
IDS/IPS
(Intrusion Detection/ Prevention System)
2014
RASP
(Runtime Application Self-Protection)
2015
Unified Agent IAST and RASP

Link

Security CI/CD detailed flow (more…)

DevSecOps Flow.pdf



DevSecOps Security Controls

1 Secure code training
Engineers gain knowledge and awareness of AppSec principles and responsibilities.

2 Secure by Design
Gather threat and abuse case models and security requirements. Adopt reusable Secure-by-default design patterns.

3 Pre-commit configuration for secret scanning

Secure access to app. code & image repository
Implement least privilege RBAC and n/w isolation to code and private container repository. Infrastructure as code version controlled.

4 Secure access to CI service
Prevent unauthorised manipulation of the pipeline itself with RBAC and separation of duties.

5 Code Analysis
SAST: Analyse code for vulnerabilities. IDE and build server integrated with remediation advice.

6 Composition Analysis
Analyse Third Party/Open Source libraries. Reuse secure code only and comply with OSS licensing.

7 CI Server deploys to test
Automated deployment via Infrastructure as Code

8 Security Tests
DAST: Perform App specific automated security testing and dynamic container analysis.
Validate controls are mitigating the abuse case threat vectors.

9 Security review human logic test
Conduct Penetration testing where it makes sense. (Might not be needed for every sprint).

10 Image Assurance
Validate image integrity signatures, and that assurance policies are met to proceed to prod.

11 Deploy to Production
Automated deployment by Infrastructure as Code. IaC compliance and security testing.

12 Privileged access, secrets management & network isolation
Implement tight network & access policy, by locking down clusters/pods/VMs.
Use secrets management and automated certificate handling.

13 Application Protection
Protect and monitor the app and cluster using RASP/WAF and container sand-boxing

14 Infrastructure & Data Protection
Apply infrastructure hardening, data encryption endpoint protection, DLP and IPS.

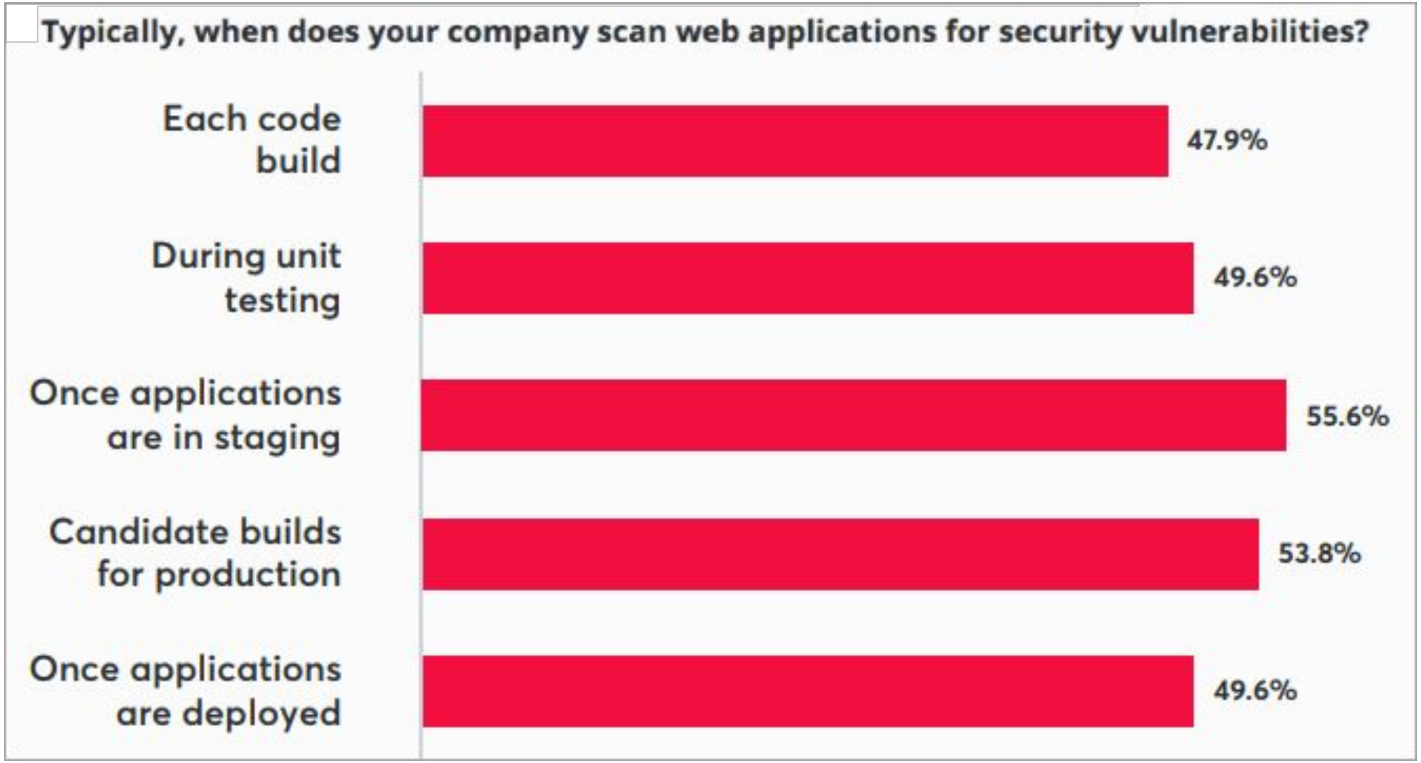
15 Vulnerability scanning
Ensure continuous assessment & automated CIS benchmarking of build and deployed images and environments.

16 Continuous monitoring & Incident Response
Analyse intrusion / breach alerts and threat intelligence. Log attacks, behaviours and threats, to respond, learn from, and apply to next sprint.

17 Rinse & Repeat
Improve and enforce governance.

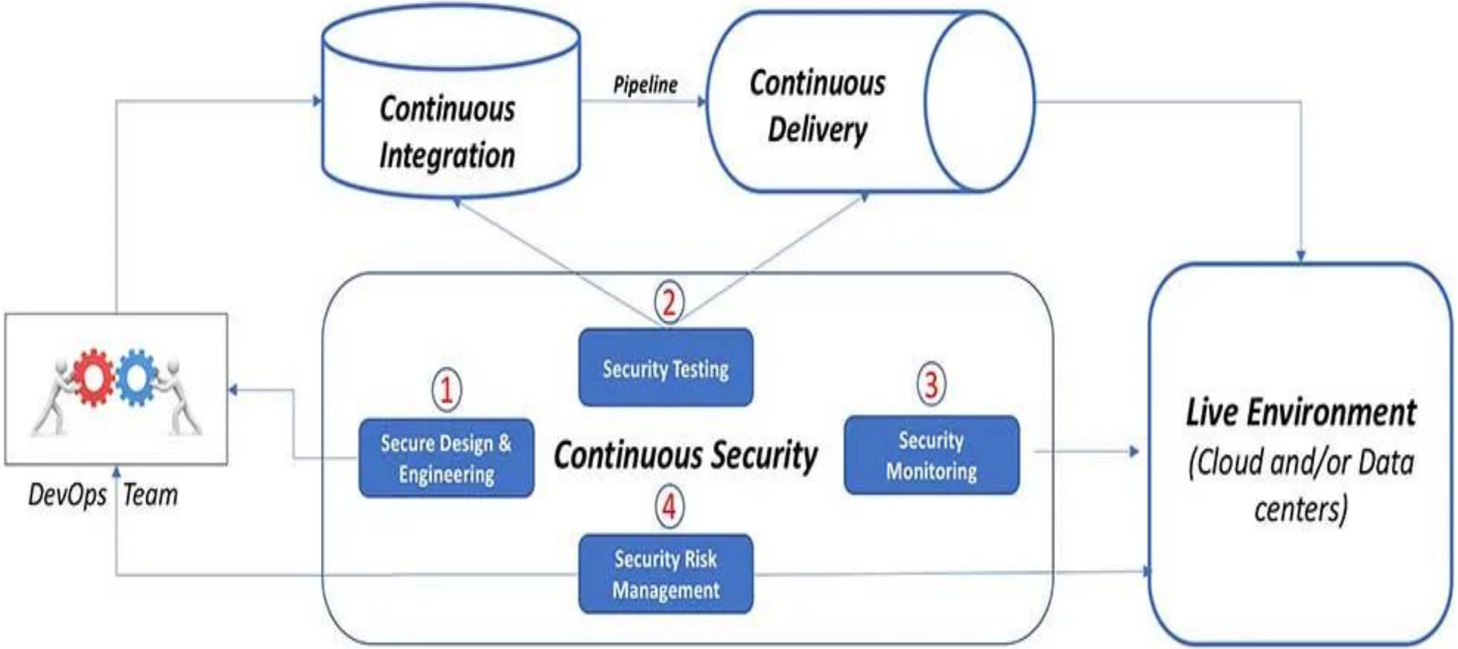
GATES
⚙ Automated
✎ Sign-off

When do they scan for vulnerabilities

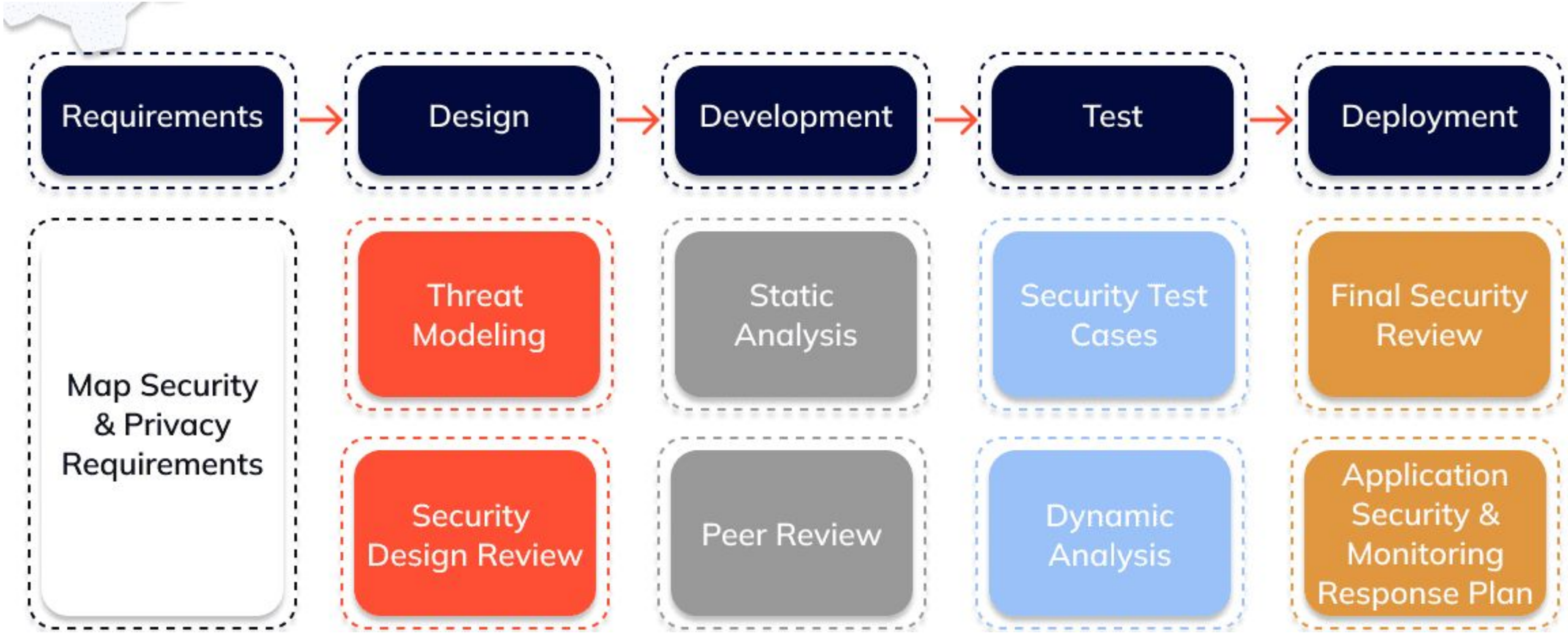


Link

Security processes in every SDLC phase



Security in SDLC



Security Phases (a)

Planning

Allocate resources

Assign security roles

Training

Acquisition of necessary tools (e.g. static code analysis tools)

Definition of project security requirements

Draft security plan

Design Phase

Security review of functional design specifications

Security review of technical design specifications

Compatibility checks with legal requirements

Investigation of approaches for security controls

Draft threat model

Security architecture document

Revision of security plan

Draft disposal plan

Implementation phase

- Code auditing-Scan the code-Eliminate the use of vulnerable components from the beginning.
  - Apply secure coding practices, integrate SAST tools. Enforce industry-followed secure-coding practices (e.g., OWASP and CERT) at this stage
  - Train developers to adopt security principles such as confidentiality, integrity, availability, and accountability while coding software modules
- Functional Security Testing
- Management of defects via security bug tracking project (JIRA)
- Revision of security architecture document
- Revision of threat model
- Revision of security plan
- Revision of disposal plan

Security Phases (a)

Test phase

Extensive system and integration testing occurs at this stage to prevent various security flaws in the software modules.

Web applications:

- Security scanning- This is commonly referred as DAST and IAST testing techniques.
- Fuzzing tools that follow fuzzing techniques for negative testing and validating the behavior of software modules
- Penetration testing - this is typically done by an external party with legal understanding with the organization to penetrate their systems and infrastructure to expose vulnerabilities and further help to fix the problems.

NoN web-based applications,

APIs, data access layer, integration layer, and middleware components all must be scanned with appropriate vulnerability scanning tools and techniques

Release phase

Packaging tests-Software composition analysis (SCA)

Scan for privileged credentials such as password and keys to avoid security mishaps. Penetration testing

Update to security bug tracker

Finalization of security architecture document

Revision of threat model

Revision of security plan

Finalization of disposal plan

Production phase

Review of defect discovered after release

Update to security bug tracker

Update to security plan

Update to security architecture document (if relevant)

Update to threat model (if relevant)

Update to disposal plan (if relevant)

Review of security patch

Re-run design, implementation and release tasks for patch

The tracker

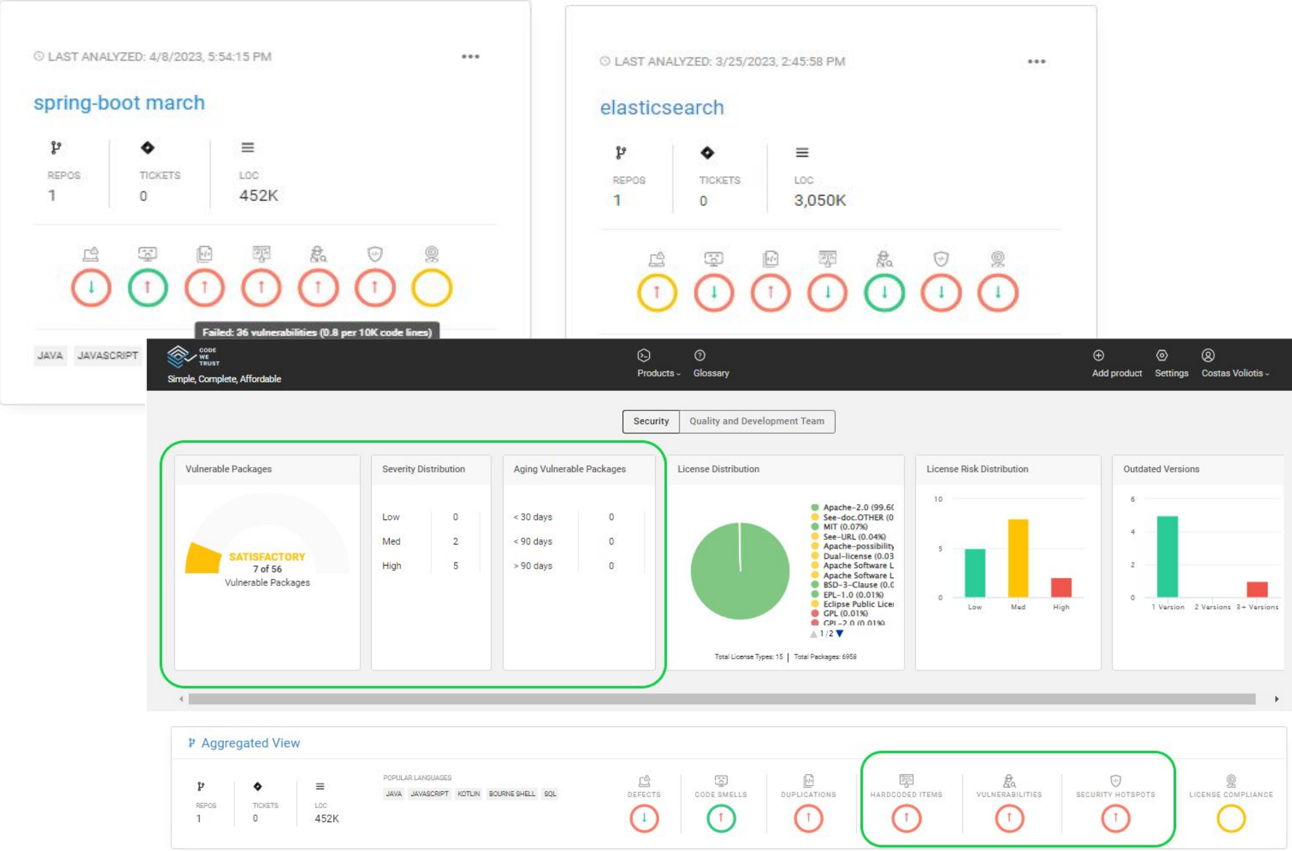
Part 3

Monitor security Issues with CodeWeTrust Scanner

Modern systems are tracking security bugs leveraging dedicated issue tracking tools (Jira, Azure devops)

And compiling digital signature of a code base

SBOM Software bill of material



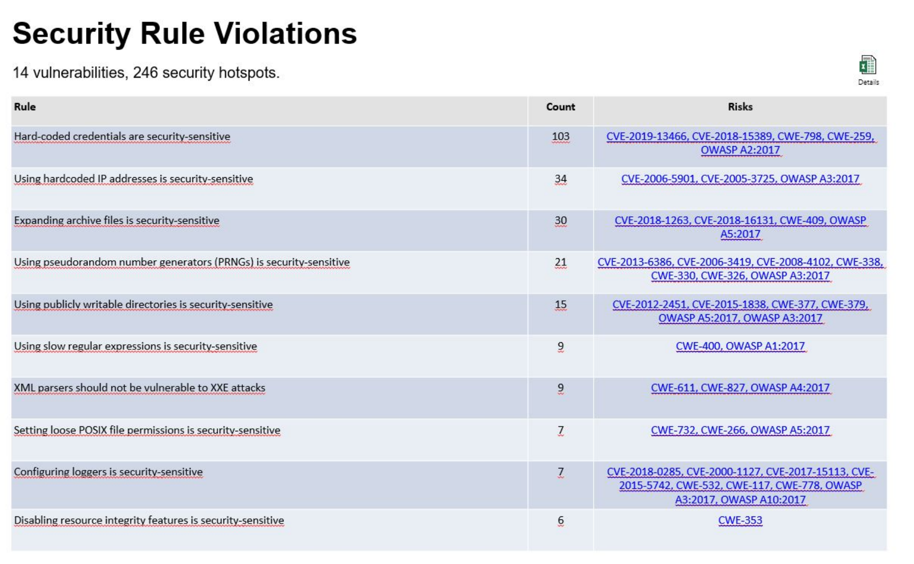
Severity Distribution

Low	0
Med	2
High	5

Aging Vulnerable Packages

< 30 days	0
< 90 days	0
> 90 days	0

Tracking Security issue with CodeWeTrust scanner

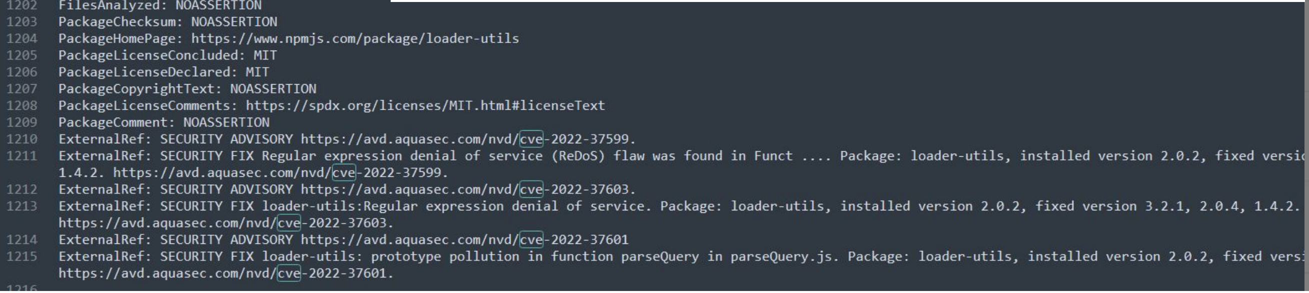


Security Rule Violations

14 vulnerabilities, 246 security hotspots.

Rule	Count	Risks
Hard-coded credentials are security-sensitive	103	CVE-2019-13466, CVE-2018-15389, CWE-798, CWE-259, OWASP A2:2017
Using hardcoded IP addresses is security-sensitive	34	CVE-2006-5901, CVE-2005-3725, OWASP A3:2017
Expanding archive files is security-sensitive	30	CVE-2018-1263, CVE-2018-16131, CWE-409, OWASP A5:2017
Using pseudorandom number generators (PRNGs) is security-sensitive	21	CVE-2013-6386, CVE-2006-3419, CVE-2008-4102, CWE-338, CWE-330, CWE-326, OWASP A3:2017
Using publicly writable directories is security-sensitive	15	CVE-2012-2451, CVE-2015-1838, CWE-377, CWE-379, OWASP A5:2017, OWASP A3:2017
Using slow regular expressions is security-sensitive	9	CWE-400, OWASP A1:2017
XML parsers should not be vulnerable to XXE attacks	9	CWE-611, CWE-827, OWASP A4:2017
Setting loose POSIX file permissions is security-sensitive	7	CWE-732, CWE-266, OWASP A5:2017
Configuring loggers is security-sensitive	7	CVE-2018-0285, CVE-2000-1127, CVE-2017-15113, CVE-2015-5742, CWE-532, CWE-117, CWE-778, OWASP A3:2017, OWASP A10:2017
Disabling resource integrity features is security-sensitive	6	CWE-353

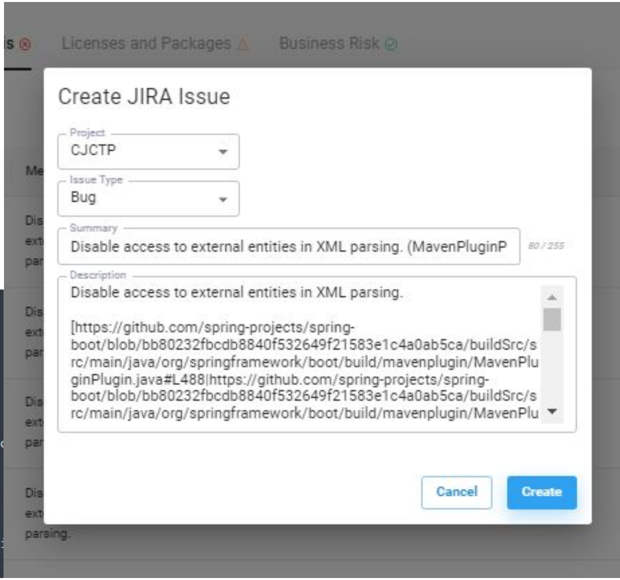
SBOM example



```

##### loader-utils

PackageName: loader-utils
SPDXID: SPDXRef-Package-loader-utils-2.0.2
PackageVersion: 2.0.2
PackageSupplier: NOASSERTION
PackageDownloadLocation: NOASSERTION
FilesAnalyzed: NOASSERTION
PackageChecksum: NOASSERTION
PackageHomePage: https://www.npmjs.com/package/loader-utils
PackageLicenseConcluded: MIT
PackageLicenseDeclared: MIT
PackageCopyrightText: NOASSERTION
PackageLicenseComments: https://spdx.org/licenses/MIT.html#licenseText
PackageComment: NOASSERTION
ExternalRef: SECURITY ADVISORY https://avd.aquasec.com/nvd/cve-2022-37599.
ExternalRef: SECURITY FIX Regular expression denial of service (ReDoS) flaw was found in Funct .... Package: loader-utils, installed version 2.0.2, fixed versi
1.4.2. https://avd.aquasec.com/nvd/cve-2022-37599.
ExternalRef: SECURITY ADVISORY https://avd.aquasec.com/nvd/cve-2022-37603.
ExternalRef: SECURITY FIX loader-utils:Regular expression denial of service. Package: loader-utils, installed version 2.0.2, fixed version 3.2.1, 2.0.4, 1.4.2.
https://avd.aquasec.com/nvd/cve-2022-37603.
ExternalRef: SECURITY ADVISORY https://avd.aquasec.com/nvd/cve-2022-37601
ExternalRef: SECURITY FIX loader-utils: prototype pollution in function parseQuery in parseQuery.js. Package: loader-utils, installed version 2.0.2, fixed versi
https://avd.aquasec.com/nvd/cve-2022-37601.
```



Create JIRA Issue

Project: CJCTP

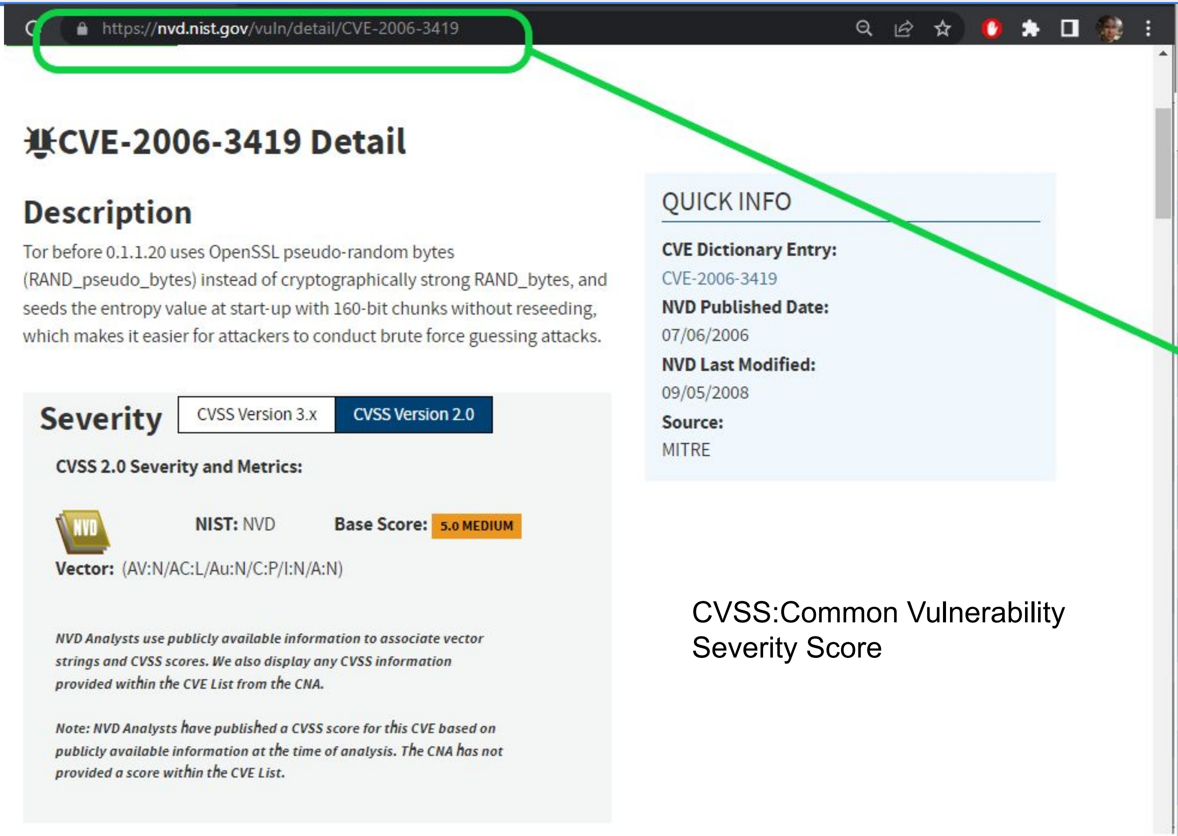
Issue Type: Bug

Summary: Disable access to external entities in XML parsing. (MavenPluginP 80 / 255

Description: Disable access to external entities in XML parsing.

[https://github.com/spring-projects/spring-boot/blob/bb80232fbcdb8840f532649f21583e1c4a0ab5ca/buildSrc/src/main/java/org/springframework/boot/build/mavenplugin/MavenPluginPlugin.java#L488|https://github.com/spring-projects/spring-boot/blob/bb80232fbcdb8840f532649f21583e1c4a0ab5ca/buildSrc/src/main/java/org/springframework/boot/build/mavenplugin/MavenPlu

CVE inventory-CVSS SCORE



https://nvd.nist.gov/vuln/detail/CVE-2006-3419

CVE-2006-3419 Detail

Description

Tor before 0.1.1.20 uses OpenSSL pseudo-random bytes (RAND_pseudo_bytes) instead of cryptographically strong RAND_bytes, and seeds the entropy value at start-up with 160-bit chunks without reseeding, which makes it easier for attackers to conduct brute force guessing attacks.

Severity

CVSS Version 3.x | CVSS Version 2.0

CVSS 2.0 Severity and Metrics:

NIST: NVD **Base Score:** 5.0 MEDIUM

Vector: (AV:N/AC:L/Au:N/C:P/I:N/A:N)

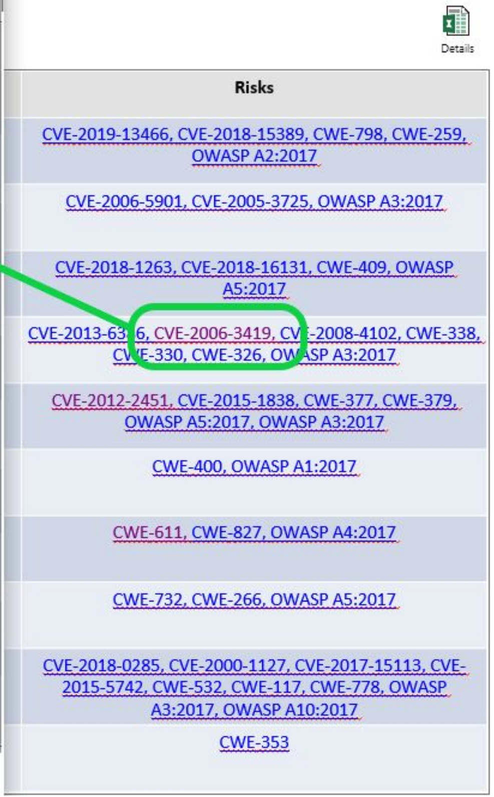
NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

QUICK INFO

CVE Dictionary Entry:
CVE-2006-3419
NVD Published Date:
07/06/2006
NVD Last Modified:
09/05/2008
Source:
MITRE

CVSS:Common Vulnerability Severity Score



Details

Risks
CVE-2019-13466, CVE-2018-15389, CWE-798, CWE-259, OWASP A2:2017
CVE-2006-5901, CVE-2005-3725, OWASP A3:2017
CVE-2018-1263, CVE-2018-16131, CWE-409, OWASP A5:2017
CVE-2013-6386, CVE-2006-3419, CVE-2008-4102, CWE-338, CWE-330, CWE-326, OWASP A3:2017
CVE-2012-2451, CVE-2015-1838, CWE-377, CWE-379, OWASP A5:2017, OWASP A3:2017
CWE-400, OWASP A1:2017
CWE-611, CWE-827, OWASP A4:2017
CWE-732, CWE-266, OWASP A5:2017
CVE-2018-0285, CVE-2000-1127, CVE-2017-15113, CVE-2015-5742, CWE-532, CWE-117, CWE-778, OWASP A3:2017, OWASP A10:2017
CWE-353

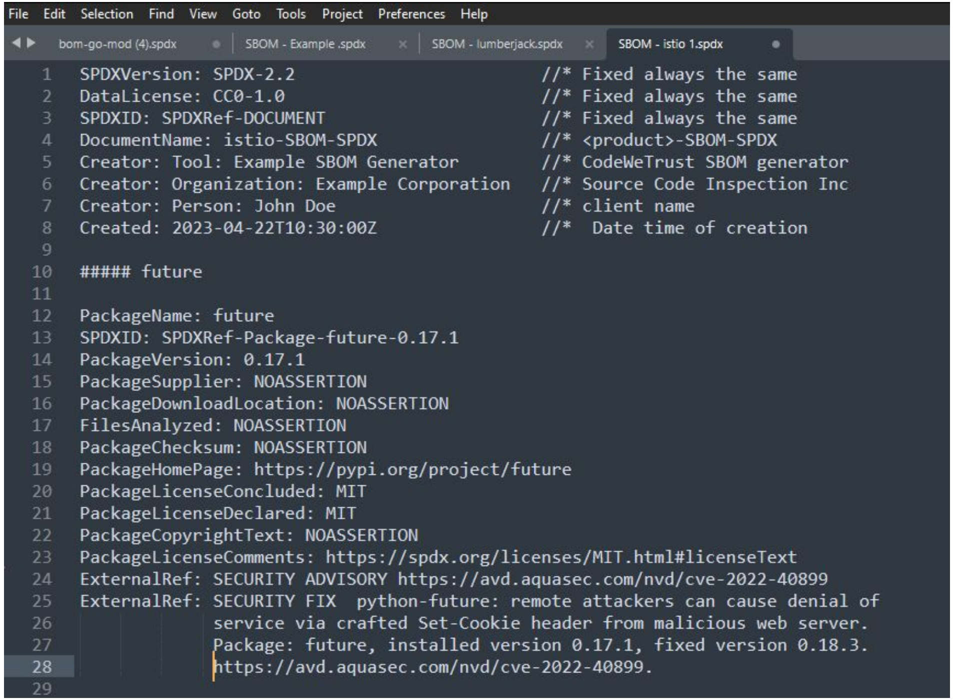
SBOM - Software Bill of material

Approve Standards:

SBOM-SPDX (SPDX) ,

SBOM-CycloneDX (OWASP)

Thi is the digital signature of a software component



```
SPDXVersion: SPDX-2.2                        //* Fixed always the same
DataLicense: CC0-1.0                         //* Fixed always the same
SPDXID: SPDXRef-DOCUMENT                     //* Fixed always the same
DocumentName: istio-SBOM-SPDX                //* <product>-SBOM-SPDX
Creator: Tool: Example SBOM Generator        //* CodeWeTrust SBOM generator
Creator: Organization: Example Corporation   //* Source Code Inspection Inc
Creator: Person: John Doe                    //* client name
Created: 2023-04-22T10:30:00Z                //*  Date time of creation

##### future

PackageName: future
SPDXID: SPDXRef-Package-future-0.17.1
PackageVersion: 0.17.1
PackageSupplier: NOASSERTION
PackageDownloadLocation: NOASSERTION
FilesAnalyzed: NOASSERTION
PackageChecksum: NOASSERTION
PackageHomePage: https://pypi.org/project/future
PackageLicenseConcluded: MIT
PackageLicenseDeclared: MIT
PackageCopyrightText: NOASSERTION
PackageLicenseComments: https://spdx.org/licenses/MIT.html#licenseText
ExternalRef: SECURITY ADVISORY https://avd.aquasec.com/nvd/cve-2022-40899
ExternalRef: SECURITY FIX  python-future: remote attackers can cause denial of
             service via crafted Set-Cookie header from malicious web server.
             Package: future, installed version 0.17.1, fixed version 0.18.3.
             https://avd.aquasec.com/nvd/cve-2022-40899.

```

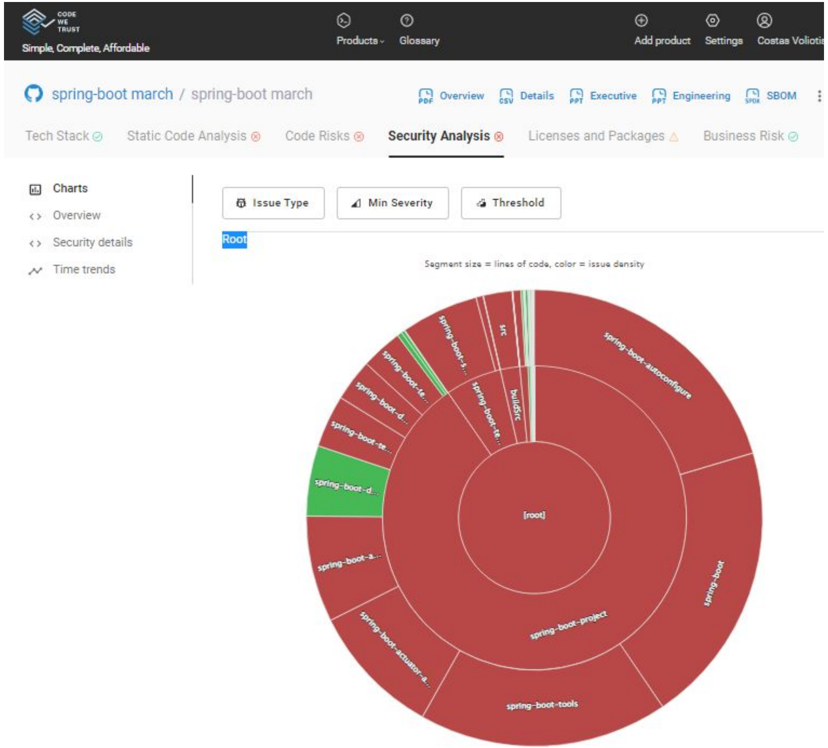
Benefits of tracking security bugs

Documentation of a bug’s life

- What is it and how was it discovered?
- Where was it introduced?
- Which versions of the software were affected by the bug?
- Which version(s) of the software addressed the bug?

Implementation of better software!

- A database of prior bugs for developers to consult
- Decision support
- Rating bugs according to criticality helps in prioritizing fixes
- Threat / risk documentation
- What were the reasons that a risk was accepted?
- Statistics
- How much time / effort did it take for this bug to be resolved?
- How many bugs were spotted in the first month after some refactoring effort?

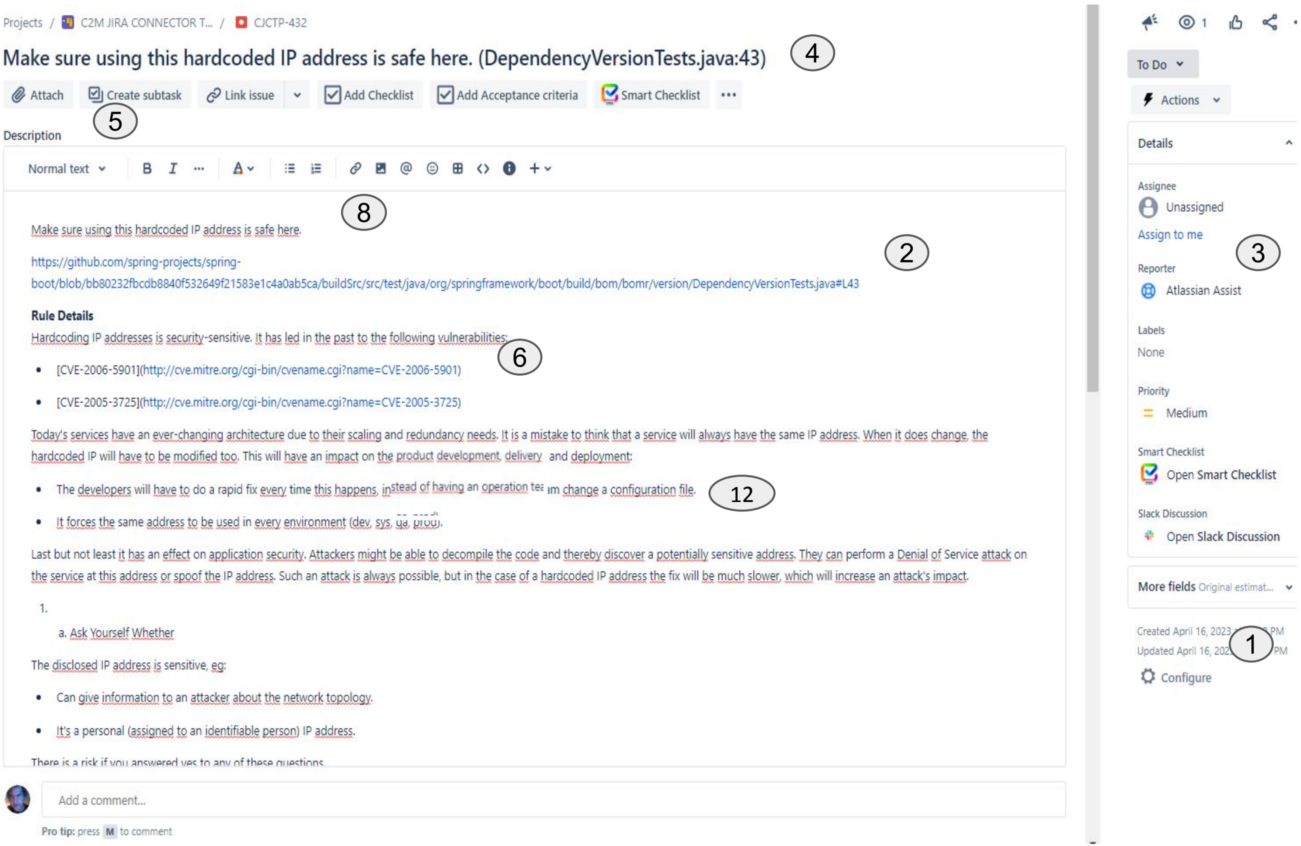


Bug tracker entries

Each defect has a single entry in the tracker

Each entry includes (at least) the following information

1. Date of discovery
2. Software component (where defect was found)
3. Author
4. Defect name
5. Defect description
6. CVE (if allocated)
7. Defect type (follow the CVE link)
8. Recommendation
9. Vulnerability score (follow the CVE link)
10. Affected versions (follow the CVE link)
11. Versions containing (follow the CVE link)
12. the fix



Make sure using this hardcoded IP address is safe here. (DependencyVersionTests.java:43)

Make sure using this hardcoded IP address is safe here.

https://github.com/spring-projects/spring-boot/blob/bb80232fbcdb8840f532649f21583e1c4a0ab5ca/buildSrc/src/test/java/org/springframework/boot/build/bom/bomr/version/DependencyVersionTests.java#L43

Rule Details

Hardcoding IP addresses is security-sensitive. It has led in the past to the following vulnerabilities:

- [CVE-2006-5901](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5901)
- [CVE-2005-3725](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3725)

Today's services have an ever-changing architecture due to their scaling and redundancy needs. It is a mistake to think that a service will always have the same IP address. When it does change, the hardcoded IP will have to be modified too. This will have an impact on the product development, delivery and deployment:

- The developers will have to do a rapid fix every time this happens, instead of having an operation tea m change a configuration file.
- It forces the same address to be used in every environment (dev, sys, qa, prod).

Last but not least it has an effect on application security. Attackers might be able to decompile the code and thereby discover a potentially sensitive address. They can perform a Denial of Service attack on the service at this address or spoof the IP address. Such an attack is always possible, but in the case of a hardcoded IP address the fix will be much slower, which will increase an attack's impact.

1.
   a. Ask Yourself Whether

The disclosed IP address is sensitive, eg:

- Can give information to an attacker about the network topology.
- It's a personal (assigned to an identifiable person) IP address.

There is a risk if you answered yes to any of these questions.

Add a comment...

Pro tip: press M to comment

Assignee: Unassigned

Assign to me

Reporter: Atlassian Assist

Labels: None

Priority: Medium

Smart Checklist: Open Smart Checklist

Slack Discussion: Open Slack Discussion

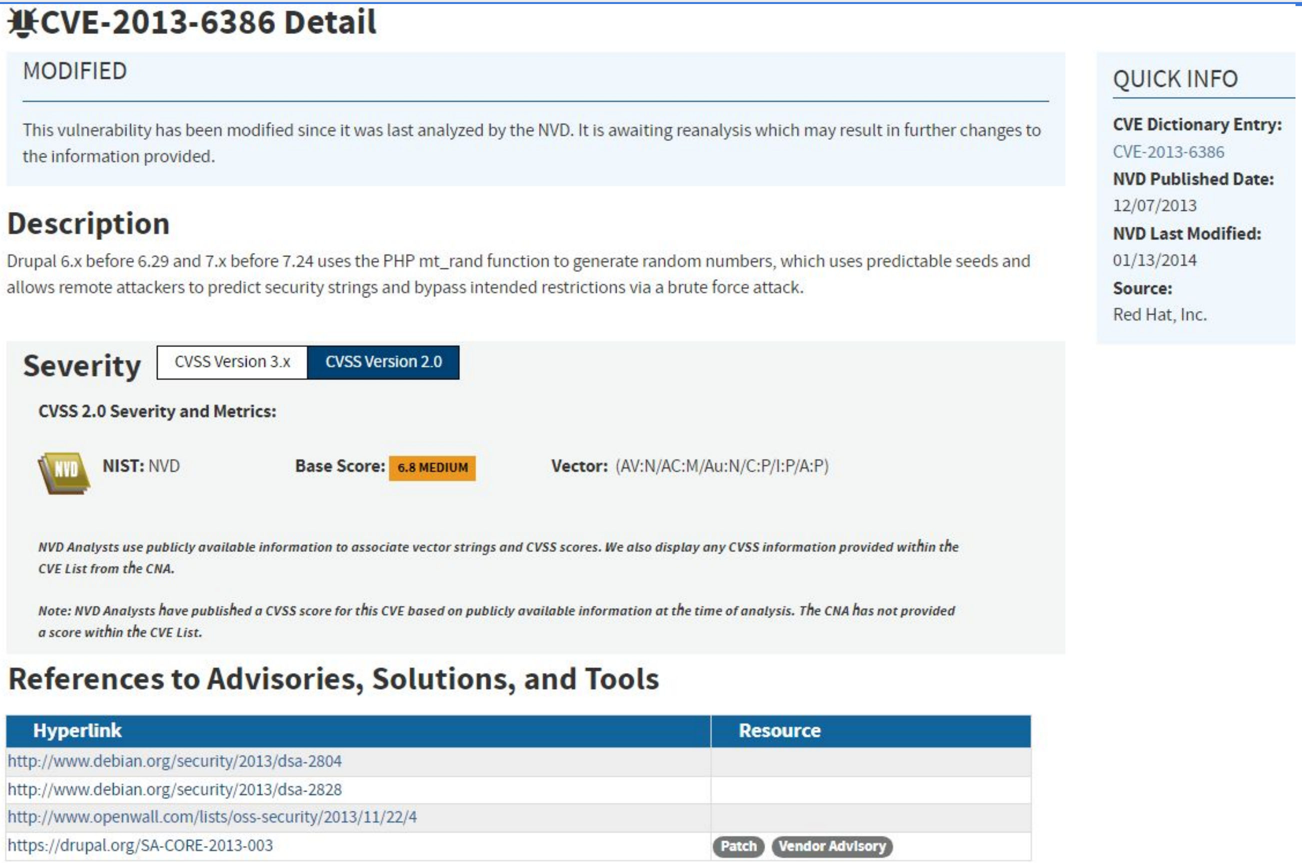
More fields Original estimat...

Created April 16, 2023 PM

Updated April 16, 202 PM

Example of CVE/CVE numbering/Severity (CVSS)

Reference



CVE-2013-6386 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.

QUICK INFO

CVE Dictionary Entry:
CVE-2013-6386
NVD Published Date:
12/07/2013
NVD Last Modified:
01/13/2014
Source:
Red Hat, Inc.

Description

Drupal 6.x before 6.29 and 7.x before 7.24 uses the PHP mt_rand function to generate random numbers, which uses predictable seeds and allows remote attackers to predict security strings and bypass intended restrictions via a brute force attack.

Severity

CVSS 2.0 Severity and Metrics:

NIST: NVD **Base Score:** 6.8 MEDIUM **Vector:** (AV:N/AC:M/Au:N/C:P/I:P/A:P)

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

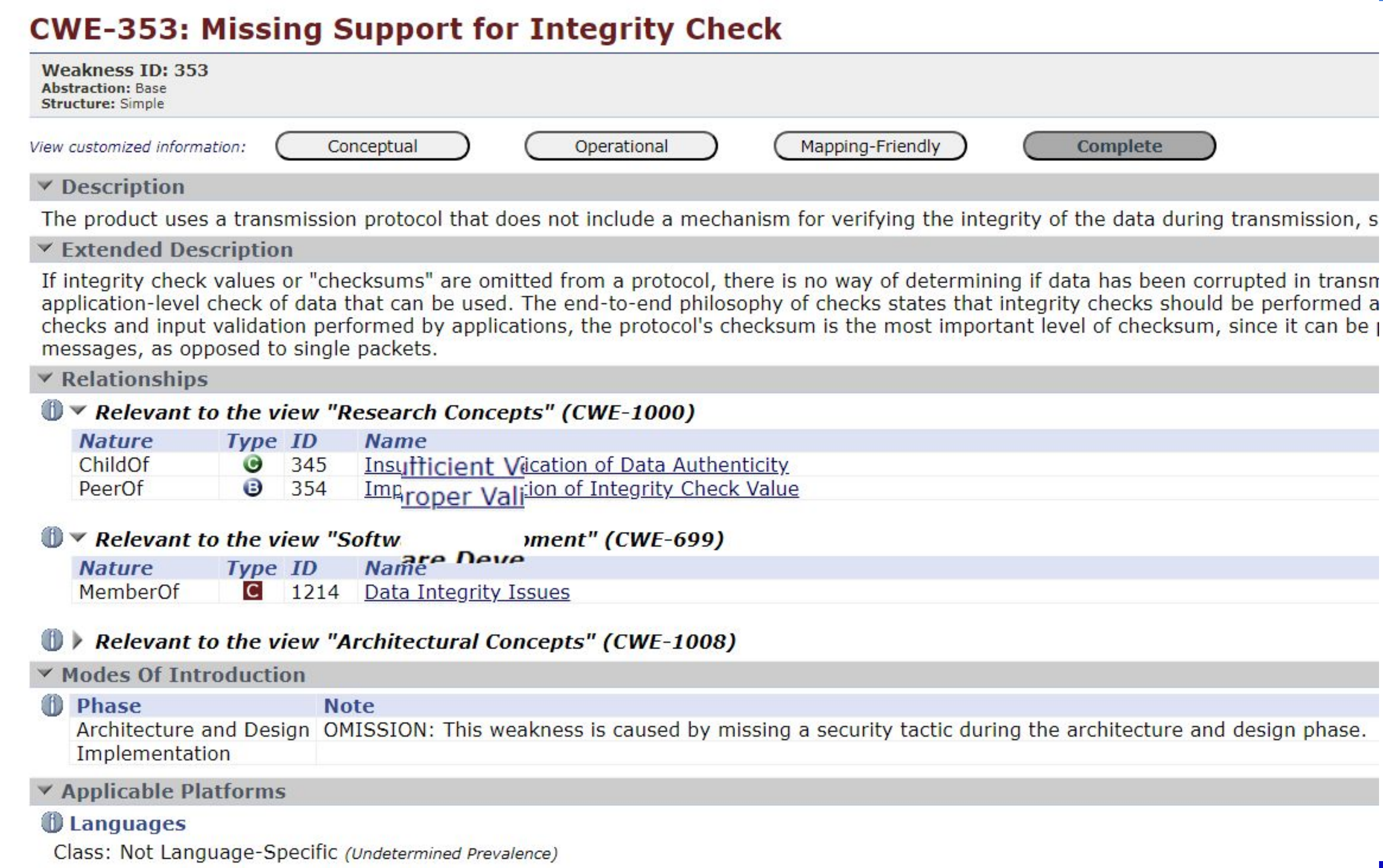
Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

References to Advisories, Solutions, and Tools

Hyperlink	Resource
http://www.debian.org/security/2013/dsa-2804	
http://www.debian.org/security/2013/dsa-2828	
http://www.openwall.com/lists/oss-security/2013/11/22/4	
https://drupal.org/SA-CORE-2013-003	Patch Vendor Advisory

Example of CWE

[Reference]



CWE-353: Missing Support for Integrity Check

Weakness ID: 353
Abstraction: Base
Structure: Simple

View customized information: Conceptual | Operational | Mapping-Friendly | Complete

Description

The product uses a transmission protocol that does not include a mechanism for verifying the integrity of the data during transmission, s

Extended Description

If integrity check values or "checksums" are omitted from a protocol, there is no way of determining if data has been corrupted in transm
application-level check of data that can be used. The end-to-end philosophy of checks states that integrity checks should be performed a
checks and input validation performed by applications, the protocol's checksum is the most important level of checksum, since it can be
messages, as opposed to single packets.

Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	*Type*	*ID*	*Name*
ChildOf	C	345	Insufficient Verification of Data Authenticity
PeerOf	B	354	Improper Validation of Integrity Check Value

Relevant to the view "Software Development" (CWE-699)

Nature	*Type*	*ID*	*Name*
MemberOf	C	1214	Data Integrity Issues

Relevant to the view "Architectural Concepts" (CWE-1008)

Modes Of Introduction

Phase	Note
Architecture and Design	OMISSION: This weakness is caused by missing a security tactic during the architecture and design phase.
Implementation	

Applicable Platforms

Languages

Class: Not Language-Specific *(Undetermined Prevalence)*

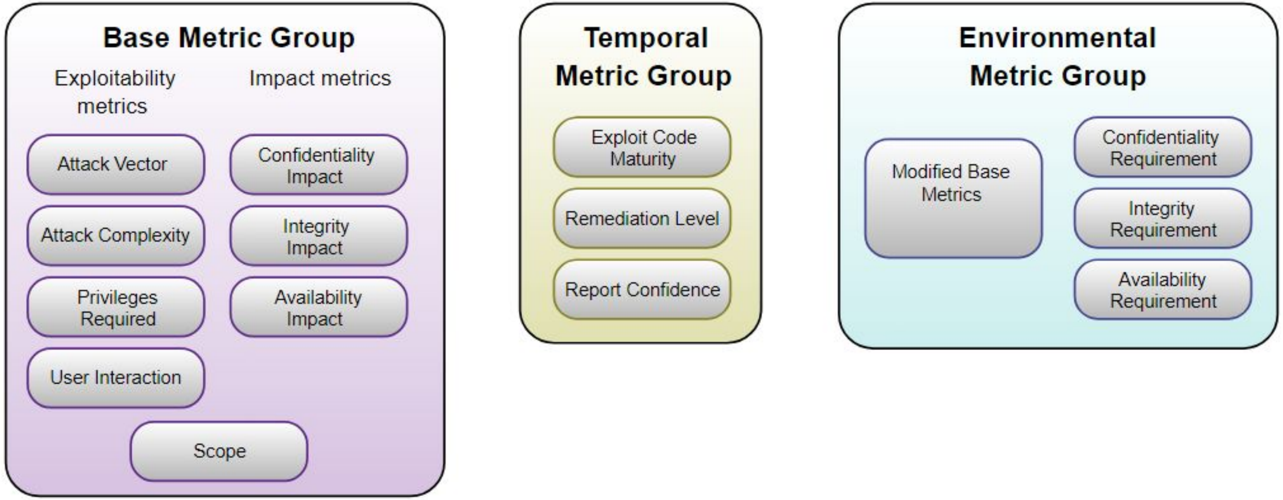
Common Vulnerability Scoring System (more…)

The Common Vulnerability Scoring System (CVSS) : open framework for communicating the characteristics and severity of software vulnerabilities.

The Base Metrics represents the intrinsic qualities of a vulnerability that are constant over time and across user environments

The Temporal Metrics reflects the characteristics of a vulnerability that change over time, and the Environmental group represents the characteristics of a vulnerability that are unique to a user's environment.

The Base metrics produce a score ranging from 0 to 10, which can then be modified by scoring the Temporal and Environmental metrics.



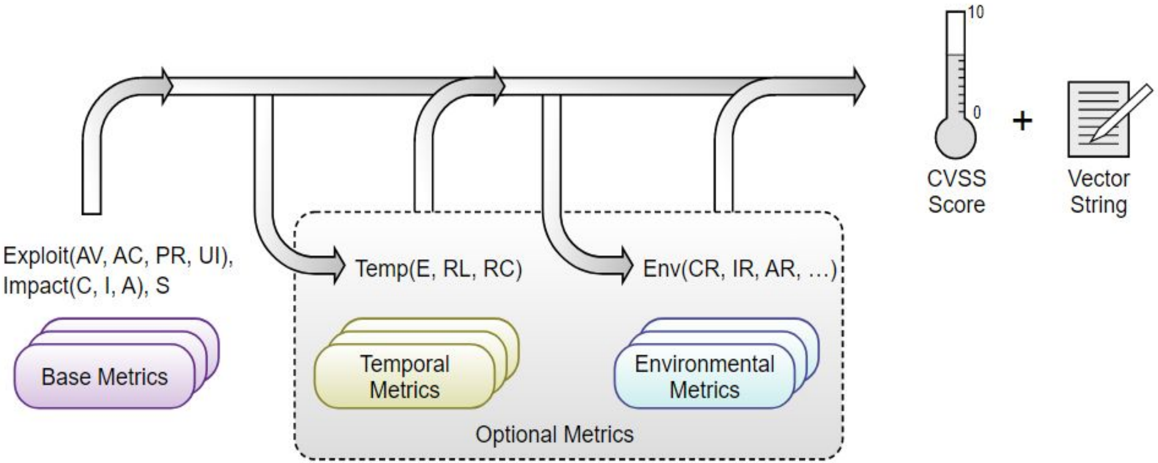
CVSS calculation scoring 3.1 (pdf)

When the Base metrics are assigned values by an analyst, the Base equation computes a score ranging from 0.0 to 10.0

The Base equation is derived from: the Exploitability sub-score equation, and the Impact sub-score equation. The Exploitability sub-score equation is derived from the Base Exploitability metrics, while the Impact sub-score equation is derived from the Base Impact metrics.

The Base Score can then be refined by scoring the Temporal and Environmental metrics in order to more accurately reflect the relative severity posed by a vulnerability to a user's environment at a specific point in time.

Generally, the Base and Temporal metrics are specified by vulnerability bulletin analysts, security product vendors, or application vendors. The Environmental metrics are specified by end-user organizations



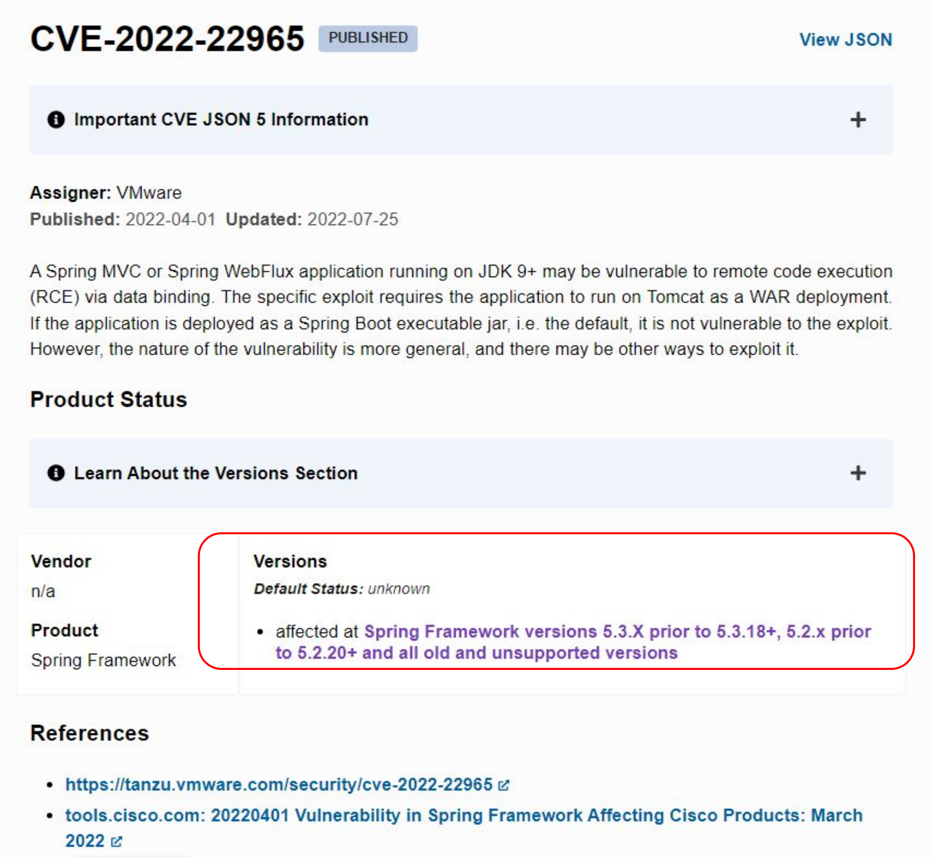
Post-release bug handling

Part 4

Security Updates/ Post-release bug fixing

Once software is released it is usually supported by security updates until it reaches the so called ’end-of-life’ (EOL) status.

Development groups must establish procedures to cater for post-release bug fixing



CVE-2022-22965 PUBLISHED

View JSON

Important CVE JSON 5 Information +

Assigner: VMware
Published: 2022-04-01 **Updated:** 2022-07-25

A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it.

Product Status

Learn About the Versions Section +

Vendor
n/a

Product
Spring Framework

Versions
Default Status: *unknown*

- affected at **Spring Framework versions 5.3.X prior to 5.3.18+, 5.2.x prior to 5.2.20+ and all old and unsupported versions**

References

- https://tanzu.vmware.com/security/cve-2022-22965
- tools.cisco.com: 20220401 Vulnerability in Spring Framework Affecting Cisco Products: March 2022

Internal process spots bug

A bug is found on a version of the software which has already been

deployed.

- Management must examine
- the releases / installations that are affected by this bug
- the cost of implementing a proper fix
- the threat that it imposes to the users and the organization
- whether the risk can be accepted
- whether users can be protected by means of a configuration change
- whether the fix needs to be pushed to the users
- whether the fix will only be available to a specific set of users (e.g.
- those using a newer version)

Once management decides that a fix needs to be deployed to customers

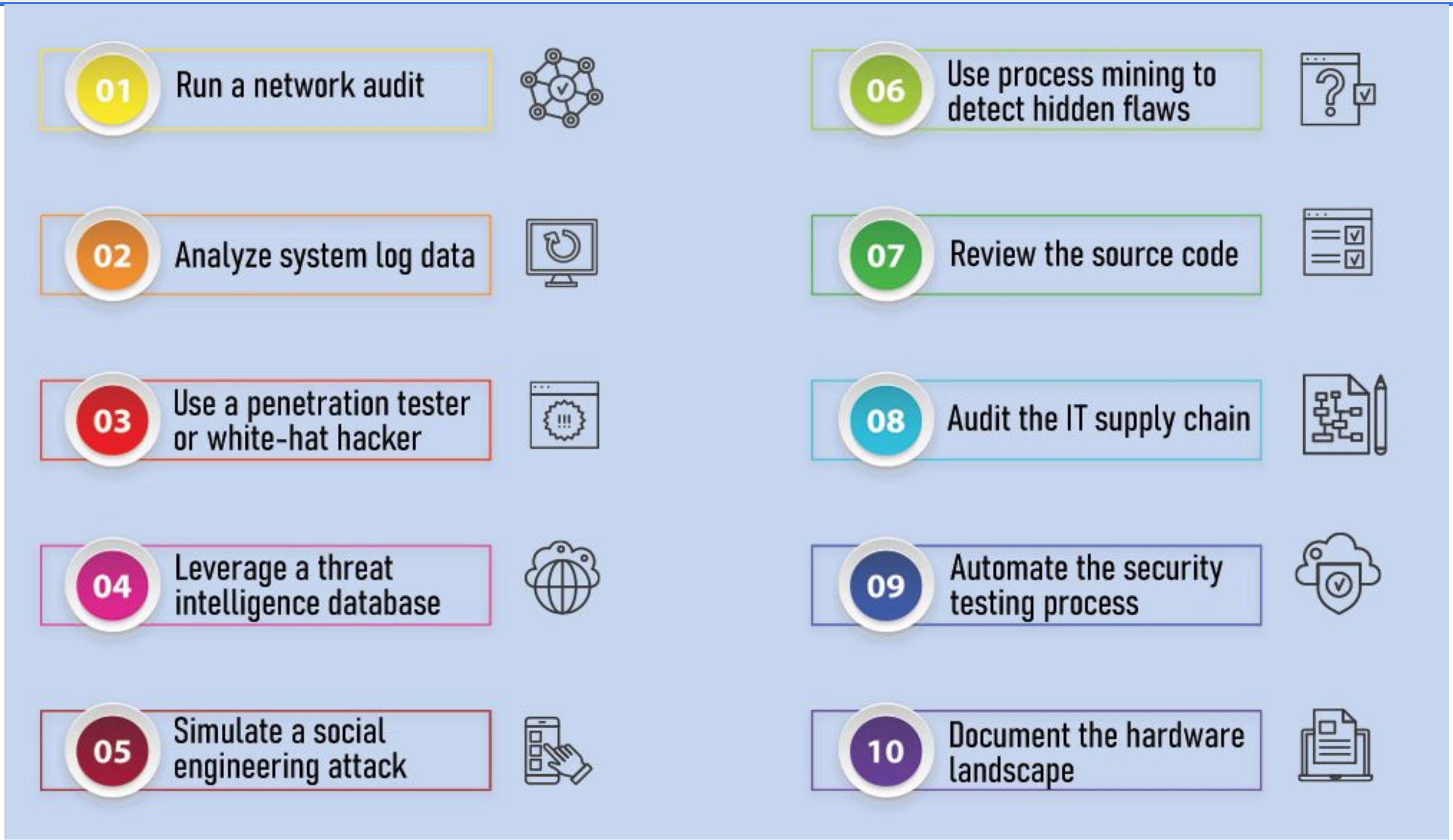
- The fix is implemented
- Full testing is performed to check for regression issues
- An advisory needs to be published letting the users know that
- unpatched versions suffer from the security bug
- Support / staff need to be informed about the bug and patching
- procedure

External process spots bug

An independent researcher / user reports a security bug

- Bug is evaluated
- Vulnerable releases / installations are identified
- Cost of fix is evaluated
- Coordinated advisory is published (referencing the researcher and CVE)
- Testing / training procedures remain the same as if the bug was discovered by an internal process
- Rewarding the researcher may have a positive impact to the company and the community as a whole

Summary: How to detect a security bug (link)



01 Run a network audit

02 Analyze system log data

03 Use a penetration tester or white-hat hacker

04 Leverage a threat intelligence database

05 Simulate a social engineering attack

06 Use process mining to detect hidden flaws

07 Review the source code

08 Audit the IT supply chain

09 Automate the security testing process

10 Document the hardware landscape

APPENDIX

Top security breaches (source)

Yahoo! Date: 2013-2016 **Impact:** Over 3 billion user accounts exposed→$35 millior

Microsoft: **Date:** January 2021,**Impact:**(60,000 companies worldwide) (4 zero-day vulnerabilities)

First American Financial Corp: May 2019:Impact, 885 million file records leaked, (**Insecure Direct Object Reference (IDOR))** → **$500.000 fine**

Facebook/Cambridge Analytica:Date: April 2018:**Impact:** 90 million users exposed- fine

LinkedIn: Date: April 2021,**Impact:** Over 700 million user records, ransomware,

JPMorgan Chase:Date: June 2014:**Impact:** 76 million households & 7 million small businesses, cost $250M

Marriott International: Date: September 2018:**Impact:** 500 million guests:On November 19, 2018→ $24M fine

Equifax:Date: September 2017:**Impact:** 148 million Americans (163 million worldwide) → $1.4B cost + $575 Fine

Common Types of Security Vulnerabilities

- Vulnerabilities in the source code
- Misconfigured system components
- Trust configurations
- Weak credentialing practices
- Lack of strong encryption
- Insider threat
- Psychological vulnerability
- Inadequate authentication
- Injection flaws
- Sensitive data exposure
- Insufficient monitoring and logs
- Shared tenancy vulnerabilities

Factors affect the cost of bugs and vulnerabilities resolution

1. Severity of the bug: Bugs can range from minor cosmetic issues (styling) to major functional problems. The severity of the bug can affect the amount of time and effort required to fix it.
2. Complexity of the bug: Some bugs are simple to fix, while others may require significant changes to the code. Logical bugs cost more.
3. Stage of the SDLC: Bugs that are discovered earlier in the development cycle are generally less expensive to fix than those found later in the cycle or after release.
4. Availability of resources: The availability of resources, including developers, testers, and tools, can affect the cost of bug fixing.
5. Code quality: High-quality code with good documentation and clear structure is generally easier and faster to fix than poorly written, messy code.
6. Communication and collaboration: Effective communication and collaboration between team members can help to identify and fix bugs more quickly and efficiently.
7. Testing environment: The testing environment can impact the ability to reproduce and diagnose bugs, which can affect the cost of fixing them.

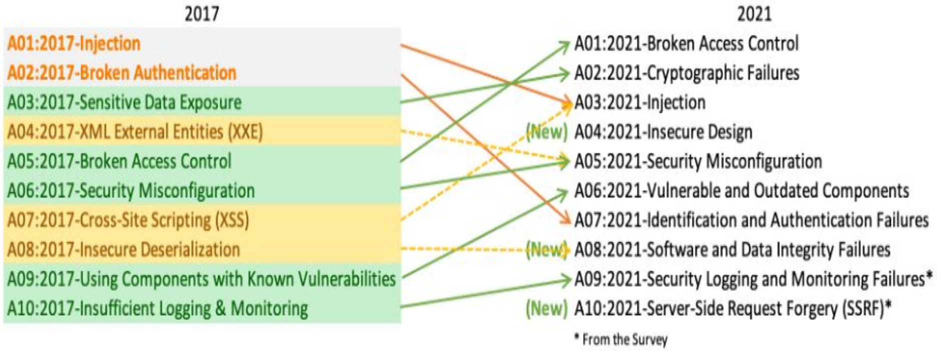
1. Severity of the vulnerability: Vulnerabilities can range from minor security weaknesses to critical security flaws.
2. Complexity of the vulnerability: Some vulnerabilities are straightforward to fix, while others may require significant changes to the code or architecture.
3. Stage of the SDLC: Vulnerabilities that are discovered earlier in the development cycle are generally less expensive to eliminate than those found later in the cycle or after release.
4. Availability of resources: The availability of resources, including developers, testers, and security experts, can affect the cost of vulnerability elimination.
5. Code quality: High-quality code with good documentation and clear structure is generally easier and faster to fix than poorly written, messy code.
6. Compliance requirements: If the software must comply with specific regulations or standards, such as PCI DSS or GDPR, the cost of vulnerability elimination may be higher due to additional compliance-related processes and requirements.
7. Impact on users and business: The potential impact of the vulnerability on users and the business can affect the urgency and resources allocated to eliminate the vulnerability.
8. Testing environment: The testing environment can impact the ability to identify and eliminate vulnerabilities, which can affect the cost of vulnerability elimination

How: Quality/Security scanner comparison matrix

FEATURE	Code WeTrust	MEND (White source)	Fossa	CAST	SNYK	Synopsis (Black Duck)	Synopsis (Coverity)	Sonar Source	Checkmarx	JetBrains Quodana
Standalone on-prem deployment Scanner+BI (Risk viewer)	✅							✅	✅	✅
Target audience	Executives Advisors Developers	Advisors Developers	Advisors Developers	Advisors	Developers Advisors	Advisors	Developers	Developers	Developers	Developers
Code reviews-Programming languages	25	0	0	15	0	0	22	30	18	7
Security (Software Composition Analysis) **Programming Languages**	ALL	ALL	ALL	ALL	ALL	ALL	22	0	18	7
Continuous Integration / Deployment	✅	✅			✅			✅	✅	✅
Source Code Quality Assessment	✅			✅				✅		
Full Vulnerabilities assessment (CVE, CWE)	✅	✅	✅	✅	✅					
“Blind” Audit - scanner	✅	limited		limited				limited	limited	✅
“Blind” Audit - BI	✅									
Licence Regulations Compliance Assessment	✅	✅	✅	✅	✅	✅				

OWASP Top 10 Security Risks & Vulnerabilities

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery



2017

A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

2021

A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

Last Update September 2021: Details

2022 CWE Top 25 Most Dangerous Software Weaknesses

OWASP top 10 PDF detailed

Vulnerabilities databases and lists

CWE : Common Weakness Enumeration

A Community-Developed List of Software & Hardware Weakness Types

CVE Common Vulnerabilities Enumeration

NVD NATIONAL VULNERABILITY DATABASE

OSWAP : The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

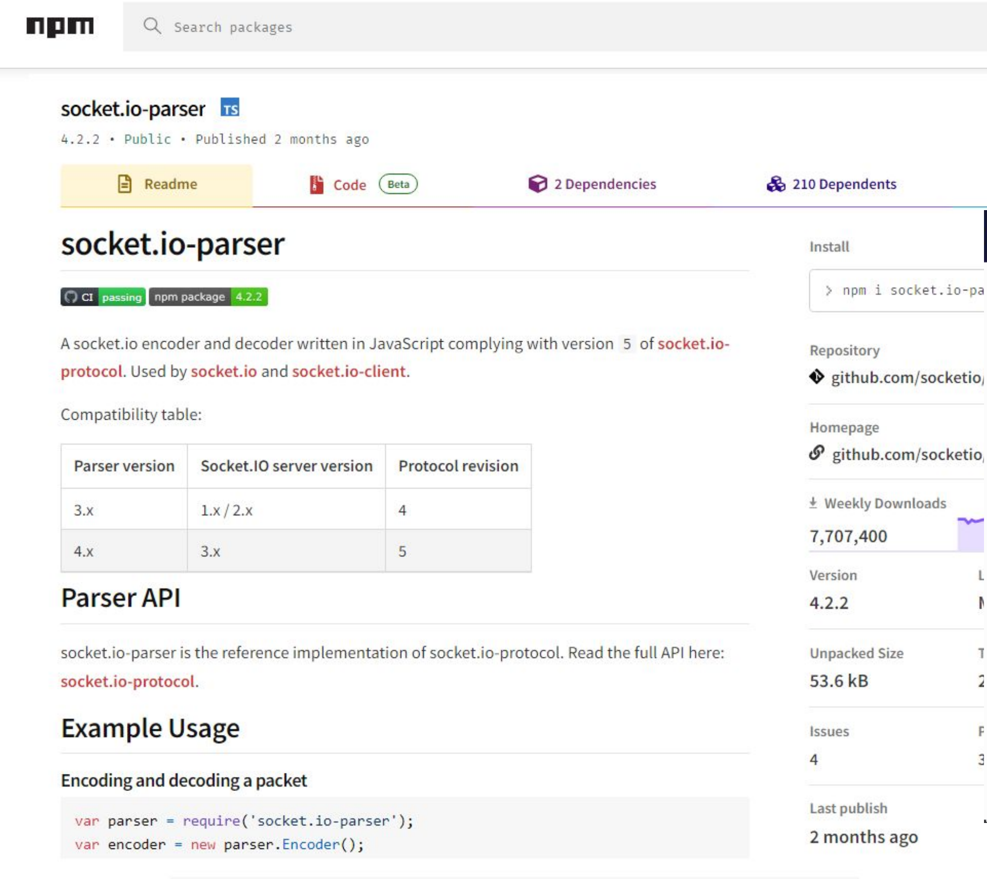
SecLists.Org Security Mailing List Archive

GitHub Security Lab

DEBIAN

RedHat RHSA

Example of advisory (for vendors)



socket.io-parser TS

4.2.2 • Public • Published 2 months ago

socket.io-parser

A socket.io encoder and decoder written in JavaScript complying with version 5 of socket.io-protocol. Used by socket.io and socket.io-client.

Compatibility table:

Parser version	Socket.IO server version	Protocol revision
3.x	1.x / 2.x	4
4.x	3.x	5

Parser API

socket.io-parser is the reference implementation of socket.io-protocol. Read the full API here: socket.io-protocol.

Example Usage

Encoding and decoding a packet

```
var parser = require('socket.io-parser');
var encoder = new parser.Encoder();
```

Install

```
> npm i socket.io-pa
```

Repository

github.com/socketio/

Homepage

github.com/socketio/

Weekly Downloads

7,707,400

Version

4.2.2

Unpacked Size

53.6 kB

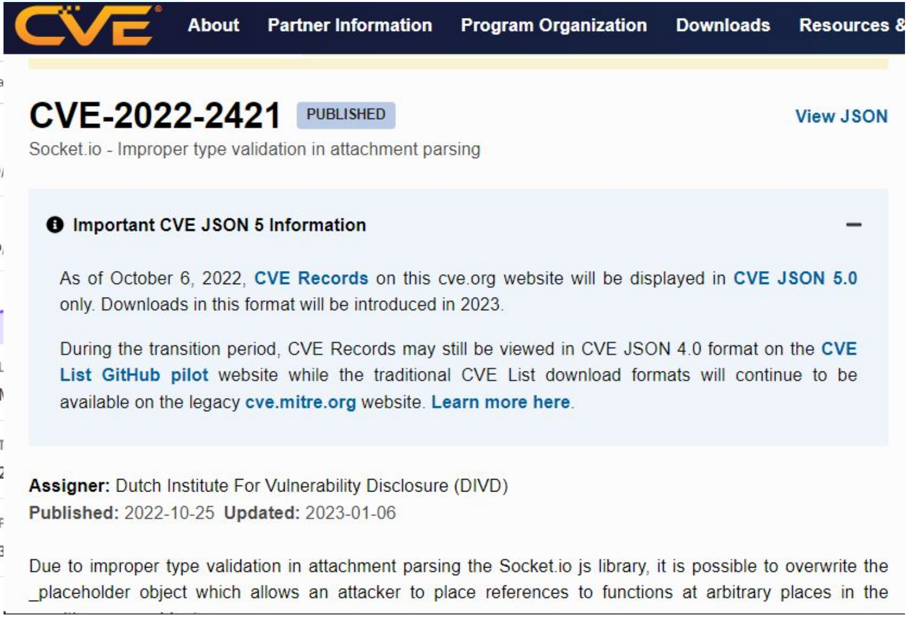
Issues

4

Last publish

2 months ago

Reference



CVE-2022-2421 PUBLISHED

View JSON

Socket.io - Improper type validation in attachment parsing

Important CVE JSON 5 Information

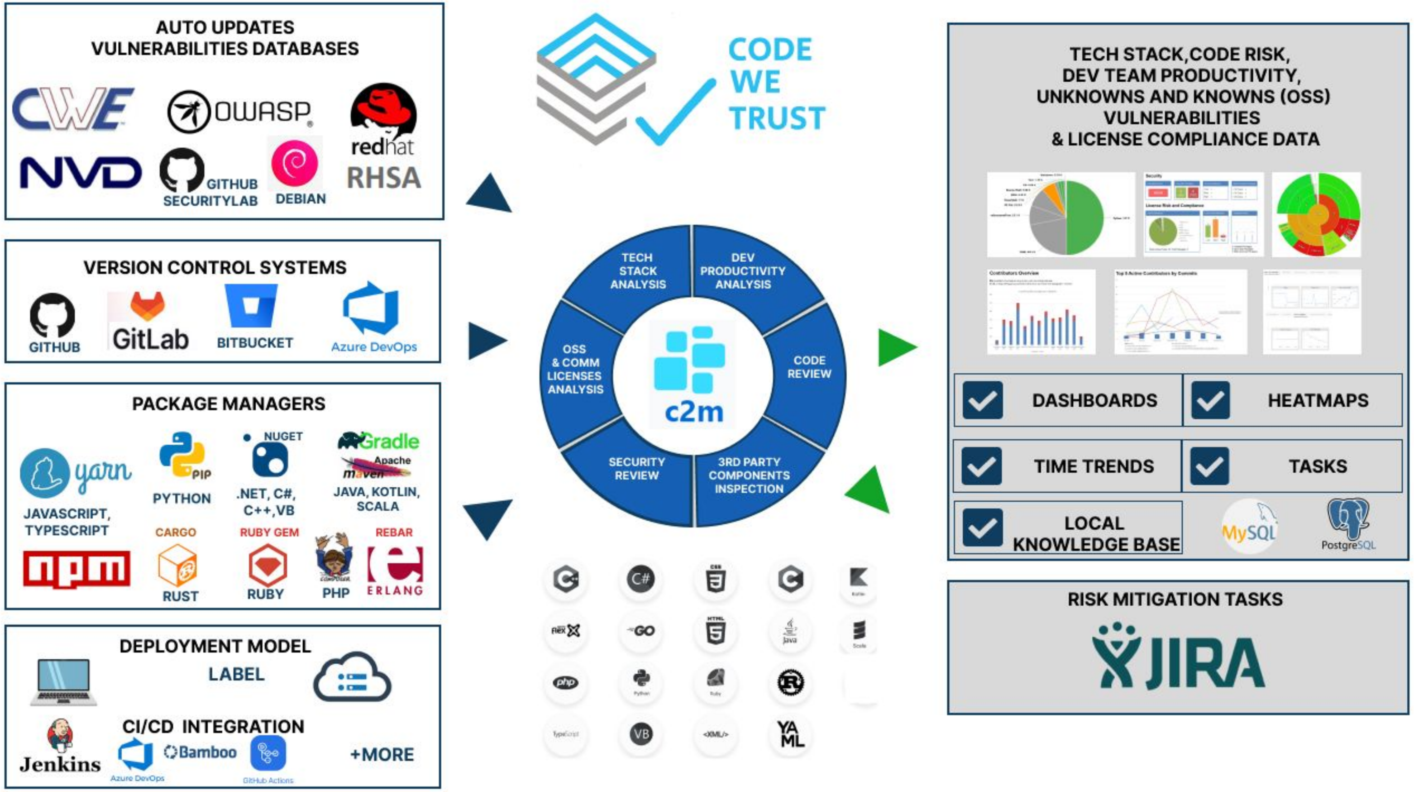
As of October 6, 2022, CVE Records on this cve.org website will be displayed in CVE JSON 5.0 only. Downloads in this format will be introduced in 2023.

During the transition period, CVE Records may still be viewed in CVE JSON 4.0 format on the CVE List GitHub pilot website while the traditional CVE List download formats will continue to be available on the legacy cve.mitre.org website. Learn more here.

Assigner: Dutch Institute For Vulnerability Disclosure (DIVD)
Published: 2022-10-25 **Updated:** 2023-01-06

Due to improper type validation in attachment parsing the Socket.io js library, it is possible to overwrite the _placeholder object which allows an attacker to place references to functions at arbitrary places in the

CodeWeTrust overview



AUTO UPDATES VULNERABILITIES DATABASES

CWE · OWASP · redhat RHSA · NVD · GITHUB SECURITYLAB · DEBIAN

VERSION CONTROL SYSTEMS

GITHUB · GitLab · BITBUCKET · Azure DevOps

PACKAGE MANAGERS

yarn — JAVASCRIPT, TYPESCRIPT · PIP — PYTHON · NUGET — .NET, C#, C++,VB · Gradle, Apache maven — JAVA, KOTLIN, SCALA

npm · CARGO — RUST · RUBY GEM — RUBY · PHP · REBAR — ERLANG

DEPLOYMENT MODEL

LABEL

CI/CD INTEGRATION

Jenkins · Azure DevOps · Bamboo · GitHub Actions · +MORE

CODE WE TRUST

TECH STACK ANALYSIS · DEV PRODUCTIVITY ANALYSIS · CODE REVIEW · 3RD PARTY COMPONENTS INSPECTION · SECURITY REVIEW · OSS & COMM LICENSES ANALYSIS

c2m

C# · GO · php · VB · <XML/> · YAML

TECH STACK, CODE RISK, DEV TEAM PRODUCTIVITY, UNKNOWNS AND KNOWNS (OSS) VULNERABILITIES & LICENSE COMPLIANCE DATA

✓ DASHBOARDS · ✓ HEATMAPS

✓ TIME TRENDS · ✓ TASKS

✓ LOCAL KNOWLEDGE BASE · MySQL · PostgreSQL

RISK MITIGATION TASKS

JIRA