Automated Threat Analysis and Management in a Continuous Integration Pipeline

Laurens Sion, Dimitri Van Landuyt, Koen Yskout, Stef Verreydt, Wouter Joosen
imec-DistriNet, KU Leuven, Belgium

https://secdev.ieee.org/2021
DevOps Lifecycle

Source: https://about.gitlab.com/stages-devops-lifecycle/
OWASP Top 10 2021

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
- A04:2021-Insecure Design
- A05:2021-Security Misconfiguration
- A06:2021-Vulnerable and Outdated Components
- A07:2021-Identification and Authentication Failures
- A08:2021-Software and Data Integrity Failures
- A09:2021-Security Logging and Monitoring Failures*
- A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey
OWASP Top 10 2021

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

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A06: 2021-Vulnerable and Outdated Components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

* From the Survey
Insecure Design

**A04:2021-Insecure Design** is a new category for 2021, with a focus on risks related to design flaws. If we genuinely want to "move left" as an industry, we need more threat modeling, secure design patterns and principles, and reference architectures. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks.
Insecure Design

**A04:2021-Insecure Design** is a new category for 2021, with a focus on **risks related to design flaws**. If we genuinely want to "move left" as an industry, we need more threat modeling, secure design patterns and principles, and reference architectures. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks.
Insecure Design

**A04:2021-Insecure Design** is a new category for 2021, with a focus on **risks related to design flaws**. If we genuinely want to "move left" as an industry, we need more threat modeling, secure design patterns and principles, and reference architectures. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks.
DevOps lifecycle

Progress threat mitigation
Which threats addressed?

Evolution security risk
Moving in the right direction?

Impact proposed changes
Do they introduce new threats?

Source: https://about.gitlab.com/stages-devops-lifecycle/
Frequent threat modeling

To find security design flaws
Threat modeling for systematic analysis design

Often manual exercise
Slow and expensive, frequently single-shot effort

Security expertise
Reliance on limited resource
Existing threat modeling tools support limited embedding in development lifecycle

Manual elicitation
ThreatDragon, ThreatSpec, ...

Automated elicitation
Pytm, SPARTA, IriusRisk, ThreatAgile, ...

Code analysis implementation-level vulnerabilities
Static and dynamic application security testing (SAST/DAST)
Can we leverage threat modeling tool support in a continuous integration context?

Reduce manual effort
Leverage existing analysis tools

Requires threat analysis engine
Elicit all applicable threats, mitigation status, risk, ...

Run analysis in CI/CD pipelines
Enable automated and frequent re-assessment
Automate threat management

Versioning design model
Together with code

Keeping track of threat analysis results
Linked to source code commits

Threat mitigation progress
Track evolution of threats during development
Elicitation engines

Not all tools elicit threats
Manual creation (e.g., ThreatSpec, ThreatDragon)

Elicit mitigated threats for progress monitoring
Existing tools remove mitigated threats (e.g., Pytm)

Richer elicitation enables more analyses
Risk, % mitigated, etc.

Leveraged existing engine
SPARTA threat analysis engine
Threat Analysis

List of threats
+ inherent risk (fully vulnerable)
+ residual risk (considering solutions)

+ security solutions
Continuous Threat Analysis & Management
Version model and code

Track evolution system design
Version model and code
Track evolution system design

Every push triggers CI job
Perform automated assessment
Version model and code
Track evolution system design

Every push triggers CI job
Perform automated assessment

Collect results
Submitted to server, linked to commit
Version model and code
Track evolution system design

Every push triggers CI job
Perform automated assessment

Collect results
Submitted to server, linked to commit

Results dashboard
Present analysis results to user
Risk evolution

Inherent risk
Residual risk

+ Insecure functionality
+ Secure functionality
- Secure functionality
- Insecure functionality
+ security solutions
- security solutions
Addresses threat management concerns

Evolution of security risk
Are measures effective in reducing risk?

Threat mitigation progress
What is the progress in mitigating threats?

Remaining threats to address
What are the most important threats to mitigate?
Discussion

1 Input model accuracy
Correspondence between model and code

2 Analysis activities
Types of analysis

3 Security metrics
What to measure for assessing security
1 Input model accuracy

Require model representation
Need model to analyze

Source code annotations
Embed model in code (e.g., threatspec)

Text-based model
Python, YAML, ... (e.g., pytm, threagile)
1 Input model accuracy

Require model representation
Need model to analyze

Conformance checking
Verify model corresponds to code

Automated reconstruction
Automatically extract model
2 Analysis activities

Threat management progress
Progress in threat mitigation?

Impact proposed changes
Security impact of feature branches?

Effectiveness of specific solutions
Do security solutions have the intended effect?
3 Security metrics

Current metrics
Threat count, inherent risk, residual risk, ...

Assess new metrics
Leverage historical analysis results
Conclusion

Step towards tighter integration threat modeling and code
Model together with code

Model from code
Automatic extraction

Threat modeling as a continuous concern
Continuous quality monitoring
Automated Threat Analysis and Management in a Continuous Integration Pipeline

Laurens Sion, Dimitri Van Landuyt, Koen Yskout, Stef Verreydt, Wouter Joosen
laurens.sion@cs.kuleuven.be

DistriNet  #IEEESecDev  https://secdev.ieee.org/2021