October 11th, 2018

Australian Cyber Conference

SANS

Continuous Security:

Exploring the DevOps Toolchain

SANS AppSec

APPLICATION & SOFTWARE SECURITY

CURRICULUM

Get the right training to build secure applications.

PLATFORM SECURITY

DEV531
Defending Mobile Applications Security Essentials

DEV541
Secure Coding in Java/JEE
GSSP-JAVA

DEV544
Secure Coding in .NET
GSSP-NET

@sansappsec

software-security.sans.org

CORE

STH.DEVELOPER
Application Security Awareness Modules

DEV522
Defending Web Applications Security Essentials
GWEB

DEV534
Secure DevOps:
A Practical Introduction

DEV540
Secure DevOps and Cloud Application Security

SPECIALIZATION

SEC542
Web App Penetration Testing and Ethical Hacking
GWAPT

SEC642
Advanced Web App Penetration Testing and Ethical Hacking

ASSESSMENT

AppSec CyberTalent Assessment
sans.org/appsec-assessment

$ WHOAMI

Eric Johnson

- Principal Security Engineer, Puma Security
  - Coder: static analysis engine, cloud automation, security tools
  - Security assessments: DevSecOps, cloud, source code, web apps, mobile apps
- Application Security Curriculum Manager, SANS Institute
  - SANS Certified Instructor
  - Contributing author of DEV540, DEV531, and DEV544
- Education & Training
  - Iowa State M.S. Information Assurance, B.S. Computer Engineering
  - AWS Certified Developer, CISSP, GSSP-Java, GSSP-.NET, GWAPT
- Contact information
  - Email: ejohnson@sans.org
  - Twitter: @emjohn20

Agenda

- ***Introduction***
- Pre-Commit
- Commit

Introduction

1. **State of DevOps**
2. **Security Challenges**
3. **DevSecOps Toolchain**

Current State of DevOps

High velocity and low cost of change enables DevOps organizations to run continuous experiments, respond to customers, pivot quickly

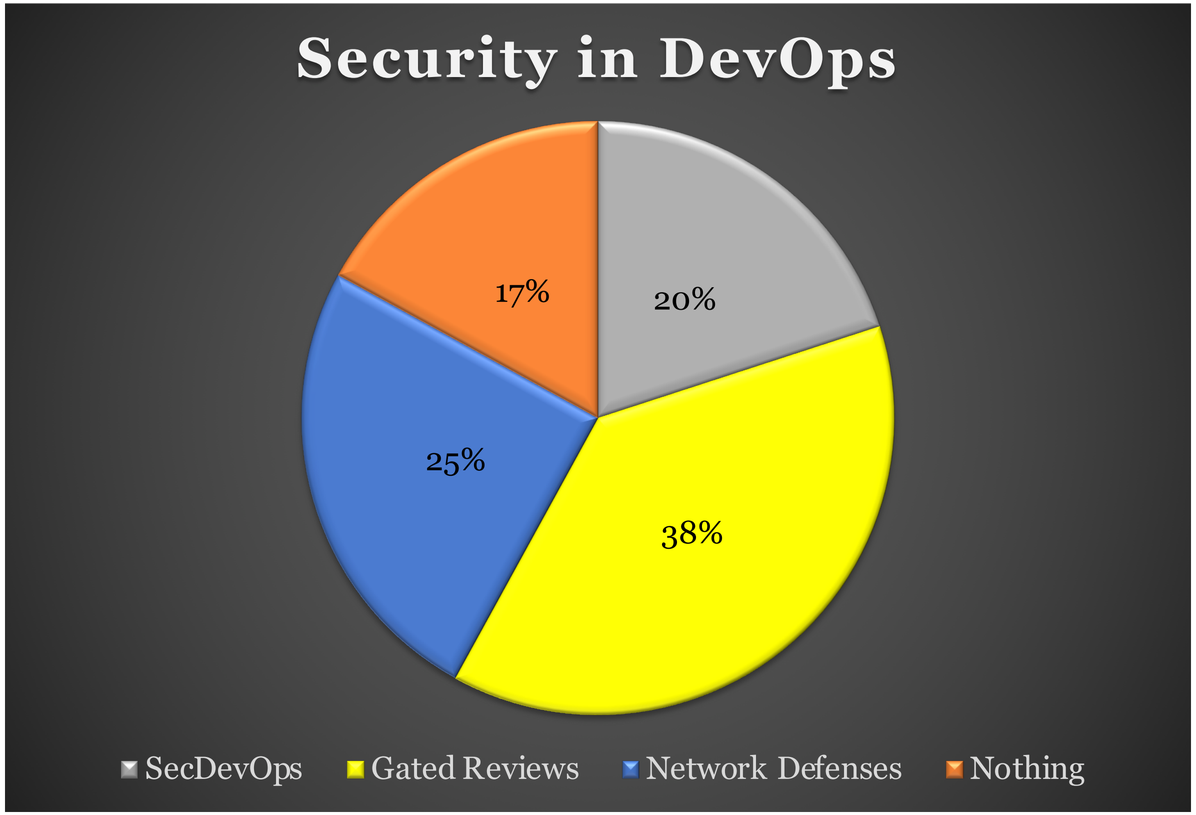
- Deploy 46x more frequently,
- 440x shorter lead times (<1 hour vs <1 month)
- Recover from failures 96x faster
- Spend 50% less time remediating security issues

puppet

Current State of DevOps Security

But… HPE study of DevOps teams in 2016 found that

- **Security is being short-changed**
  - Only 20% do security in development/delivery
  - 38% still depend on pen testing or other pre-production gate reviews
  - 25% rely on network defenses
  - 17% are doing nothing for security
- **Security is seen as somebody else's problem**



DevOps Culture and Security

DevOps culture **conflicts with traditional security culture**:

- Top down risk management instead of team-based decision making
- Need to know restrictions vs extended information sharing
- Zero failure vs fail fast and fail forward
- Limiting change – Security is always ready to say “No!”

Resources to help understand (and create) DevOps culture

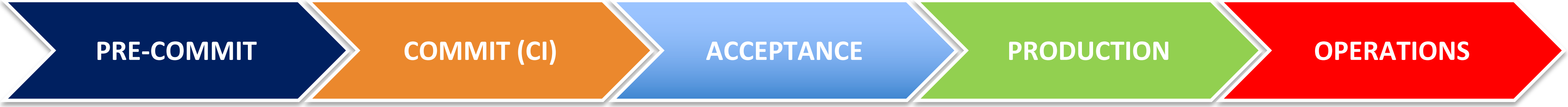
- The Phoenix Project
- Five Dysfunctions of a Team
- Lean Enterprise
- Building a DevOps Culture

SecDevOps / DevSecOps / DevOpsSec / Rugged DevOps

There are different, but compatible, memes around including security in DevOps. They all share common principles and goals:

- Make security a first-class problem and the security team a first-class participant in DevOps
- Increase trust and transparency between dev, ops, and sec
- Integrate security practices and ideas into DevOps culture, and DevOps into security culture
- Wire security into DevOps toolchains and workflows to incrementally improve security

DevSecOps Toolchain



PRE-COMMIT

COMMIT (CI)

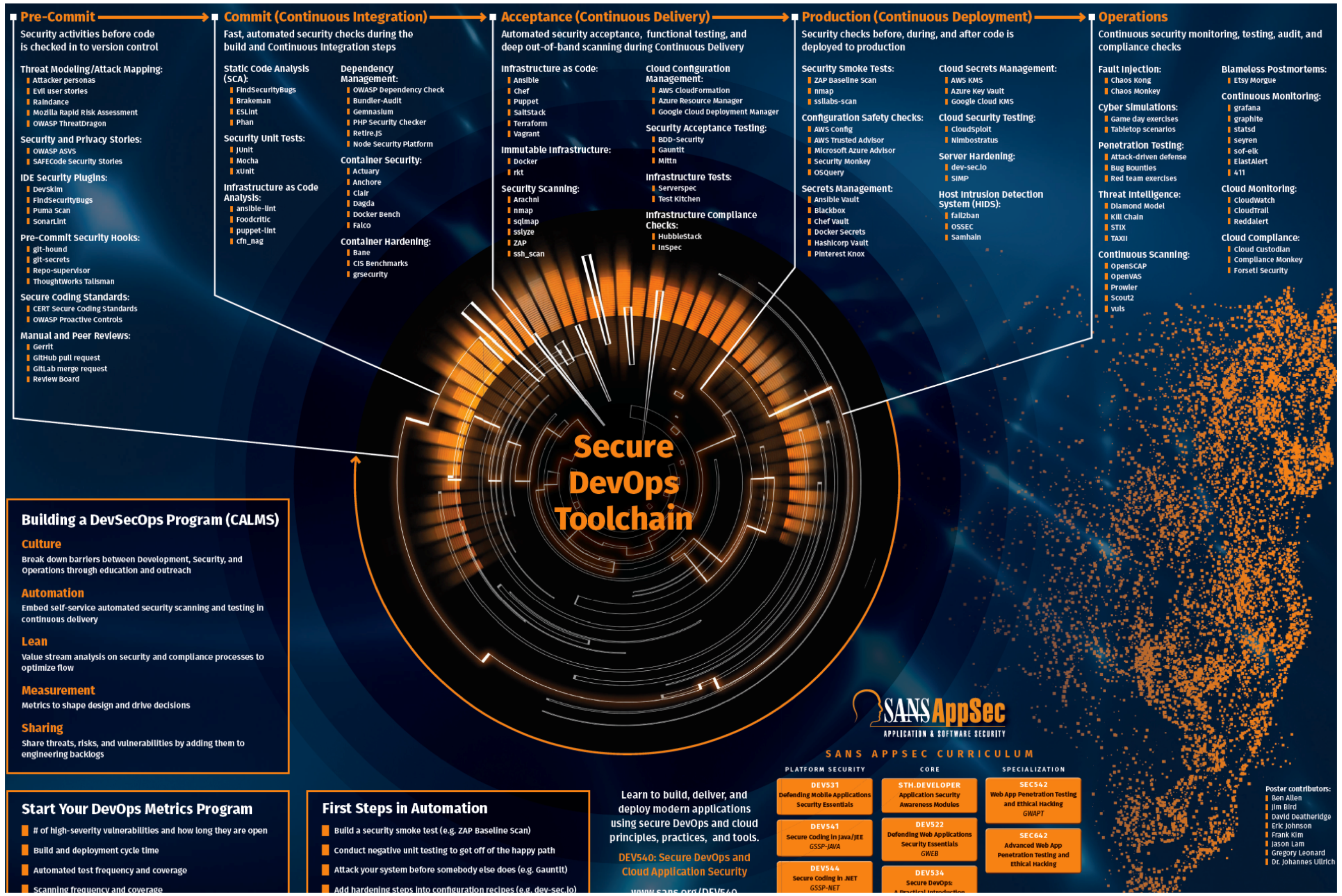
ACCEPTANCE

PRODUCTION

OPERATIONS

DevSecOps cycles through 5 key phases:

- SANS DevSecOps Toolchain poster lists several OSS tools for each phase
- Written by Ben Allen, Jim Bird, Eric Johnson, & Frank Kim
- https://sans.org/u/zAi



Pre-Commit

Security activities before code is checked in to version control

Threat Modeling/Attack Mapping:
- Attacker personas
- Evil user stories
- Raindance
- Mozilla Rapid Risk Assessment
- OWASP ThreatDragon

Security and Privacy Stories:
- OWASP ASVS
- SAFECode Security Stories

IDE Security Plugins:
- DevSkim
- FindSecurityBugs
- Puma Scan
- SonarLint

Pre-Commit Security Hooks:
- git-hound
- git-secrets
- Repo-supervisor
- ThoughtWorks Talisman

Secure Coding Standards:
- CERT Secure Coding Standards
- OWASP Proactive Controls

Manual and Peer Reviews:
- Gerrit
- GitHub pull request
- GitLab merge request
- Review Board

Commit (Continuous Integration)

Fast, automated security checks during the build and Continuous Integration steps

Static Code Analysis (SCA):
- FindSecurityBugs
- Brakeman
- ESLint
- Phan

Security Unit Tests:
- JUnit
- Mocha
- xUnit

Infrastructure as Code Analysis:
- ansible-lint
- Foodcritic
- puppet-lint
- cfn_nag

Dependency Management:
- OWASP Dependency Check
- Bundler-Audit
- Gemnasium
- PHP Security Checker
- Retire.JS
- Node Security Platform

Container Security:
- Actuary
- Anchore
- Clair
- Dagda
- Docker Bench
- Falco

Container Hardening:
- Bane
- CIS Benchmarks
- grsecurity

Acceptance (Continuous Delivery)

Automated security acceptance, functional testing, and deep out-of-band scanning during Continuous Delivery

Infrastructure as Code:
- Ansible
- Chef
- Puppet
- SaltStack
- Terraform
- Vagrant

Immutable Infrastructure:
- Docker
- rkt

Security Scanning:
- Arachni
- nmap
- sqlmap
- sslyze
- ZAP
- ssh_scan

Cloud Configuration Management:
- AWS CloudFormation
- Azure Resource Manager
- Google Cloud Deployment Manager

Security Acceptance Testing:
- BDD-Security
- Gauntlt
- Mittn

Infrastructure Tests:
- Serverspec
- Test Kitchen

Infrastructure Compliance Checks:
- HubbleStack
- InSpec

Production (Continuous Deployment)

Security checks before, during, and after code is deployed to production

Security Smoke Tests:
- ZAP Baseline Scan
- nmap
- ssllabs-scan

Configuration Safety Checks:
- AWS Config
- AWS Trusted Advisor
- Microsoft Azure Advisor
- Security Monkey
- OSQuery

Secrets Management:
- Ansible Vault
- Blackbox
- Chef Vault
- Docker Secrets
- Hashicorp Vault
- Pinterest Knox

Cloud Secrets Management:
- AWS KMS
- Azure Key Vault
- Google Cloud KMS

Cloud Security Testing:
- CloudSploit
- Nimbostratus

Server Hardening:
- dev-sec.io
- SIMP

Host Intrusion Detection System (HIDS):
- fail2ban
- OSSEC
- Samhain

Operations

Continuous security monitoring, testing, audit, and compliance checks

Fault Injection:
- Chaos Kong
- Chaos Monkey

Cyber Simulations:
- Game day exercises
- Tabletop scenarios

Penetration Testing:
- Attack-driven defense
- Bug Bounties
- Red team exercises

Threat Intelligence:
- Diamond Model
- Kill Chain
- STIX
- TAXII

Continuous Scanning:
- OpenSCAP
- OpenVAS
- Prowler
- Scout2
- vuls

Blameless Postmortems:
- Etsy Morgue

Continuous Monitoring:
- grafana
- graphite
- statsd
- seyren
- sof-elk
- ElastAlert
- 411

Cloud Monitoring:
- CloudWatch
- CloudTrail
- Reddalert

Cloud Compliance:
- Cloud Custodian
- Compliance Monkey
- Forseti Security

Secure DevOps Toolchain

Building a DevSecOps Program (CALMS)

Culture

Break down barriers between Development, Security, and Operations through education and outreach

Automation

Embed self-service automated security scanning and testing in continuous delivery

Lean

Value stream analysis on security and compliance processes to optimize flow

Measurement

Metrics to shape design and drive decisions

Sharing

Share threats, risks, and vulnerabilities by adding them to engineering backlogs

Start Your DevOps Metrics Program

- # of high-severity vulnerabilities and how long they are open
- Build and deployment cycle time
- Automated test frequency and coverage
- Scanning frequency and coverage

First Steps in Automation

- Build a security smoke test (e.g. ZAP Baseline Scan)
- Conduct negative unit testing to get off of the happy path
- Attack your system before somebody else does (e.g. Gauntlt)
- Add hardening steps into configuration recipes (e.g. dev-sec.io)

Learn to build, deliver, and deploy modern applications using secure DevOps and cloud principles, practices, and tools.

DEV540: Secure DevOps and Cloud Application Security

www.sans.org/DEV540

SANS AppSec

APPLICATION & SOFTWARE SECURITY

SANS APPSEC CURRICULUM

PLATFORM SECURITY

DEV531 Defending Mobile Applications Security Essentials

DEV541 Secure Coding in Java/JEE *GSSP-JAVA*

DEV544 Secure Coding in .NET *GSSP-NET*

CORE

STH.DEVELOPER Application Security Awareness Modules

DEV522 Defending Web Applications Security Essentials *GWEB*

DEV534 Secure DevOps: A Practical Introduction

SPECIALIZATION

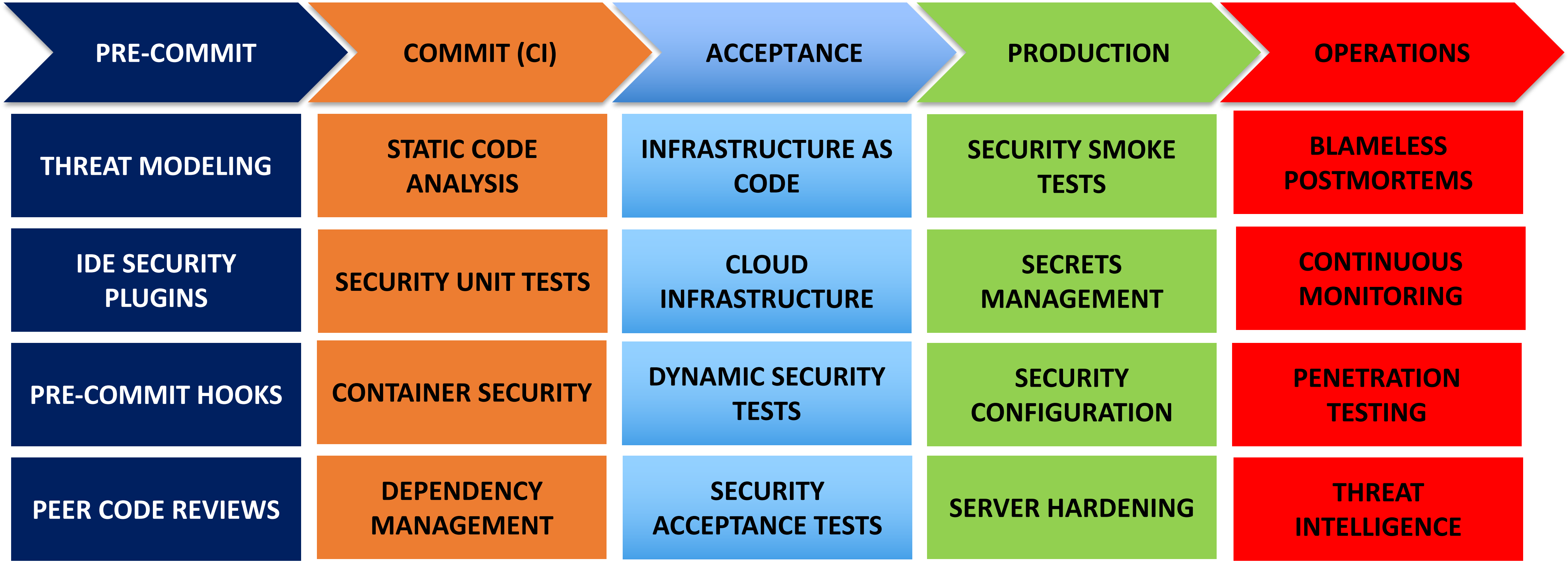
SEC542 Web App Penetration Testing and Ethical Hacking *GWAPT*

SEC642 Advanced Web App Penetration Testing and Ethical Hacking

Poster contributors:
- Ben Allen
- Jim Bird
- David Deatheridge
- Eric Johnson
- Frank Kim
- Jason Lam
- Gregory Leonard
- Dr. Johannes Ullrich

DevSecOps Security Controls

Breaking down the security controls in each DevSecOps phase:



PRE-COMMIT	COMMIT (CI)	ACCEPTANCE	PRODUCTION	OPERATIONS
THREAT MODELING	STATIC CODE ANALYSIS	INFRASTRUCTURE AS CODE	SECURITY SMOKE TESTS	BLAMELESS POSTMORTEMS
IDE SECURITY PLUGINS	SECURITY UNIT TESTS	CLOUD INFRASTRUCTURE	SECRETS MANAGEMENT	CONTINUOUS MONITORING
PRE-COMMIT HOOKS	CONTAINER SECURITY	DYNAMIC SECURITY TESTS	SECURITY CONFIGURATION	PENETRATION TESTING
PEER CODE REVIEWS	DEPENDENCY MANAGEMENT	SECURITY ACCEPTANCE TESTS	SERVER HARDENING	THREAT INTELLIGENCE

Agenda

- Introduction
- ***Pre-Commit***
- Commit

Pre-Commit Stage

1. **Threat Modeling**
2. **IDE Security Plugins**
3. **Pre-Commit Hooks**
4. **Peer Code Reviews**

DevSecOps Pre-Commit Phase

Applying security controls before code is written and committed:

PRE-COMMIT	COMMIT (CI)	ACCEPTANCE	PRODUCTION	OPERATIONS
THREAT MODELING	STATIC CODE ANALYSIS	INFRASTRUCTURE AS CODE	SECURITY SMOKE TESTS	BLAMELESS POSTMORTEMS
IDE SECURITY PLUGINS	SECURITY UNIT TESTS	CLOUD INFRASTRUCTURE	SECRETS MANAGEMENT	CONTINUOUS MONITORING
PRE-COMMIT HOOKS	CONTAINER SECURITY	DYNAMIC SECURITY TESTS	SECURITY CONFIGURATION	PENETRATION TESTING
PEER CODE REVIEWS	DEPENDENCY MANAGEMENT	SECURITY ACCEPTANCE TESTS	SERVER HARDENING	THREAT INTELLIGENCE

#1 Threat Modeling

Rapid Risk Assessments

Start with a high-level risk assessment for new systems/services

- Classify the data: legal and compliance requirements, sensitivity, etc.
- Focus on platform, language, and framework risks: is the team using well-understood tools, or something new, novel?
- Determine a risk rating and next steps: threat modeling, control gate requirements, security training …

Re-run risk assessment if/when team makes major change to design or data

PayPal risk questionnaire for new apps/services

Mozilla Rapid Risk Assessment (RRA) model – 30-minute review

Threat Modeling in DevOps

Iterative and lightweight threat modeling based on risk: early in design, or as major changes are made

Examine trust boundaries and assumptions in architecture

Ask these questions when you are making changes:

1. Are you changing the attack surface (new entry/exit points, new user role…)?
2. Are you changing the technology stack or application security controls?
3. Are you adding confidential/sensitive data?
4. Have threat agents changed – are we facing new risks?

Threat Modeling / RRA Tools

PRE-COMMIT

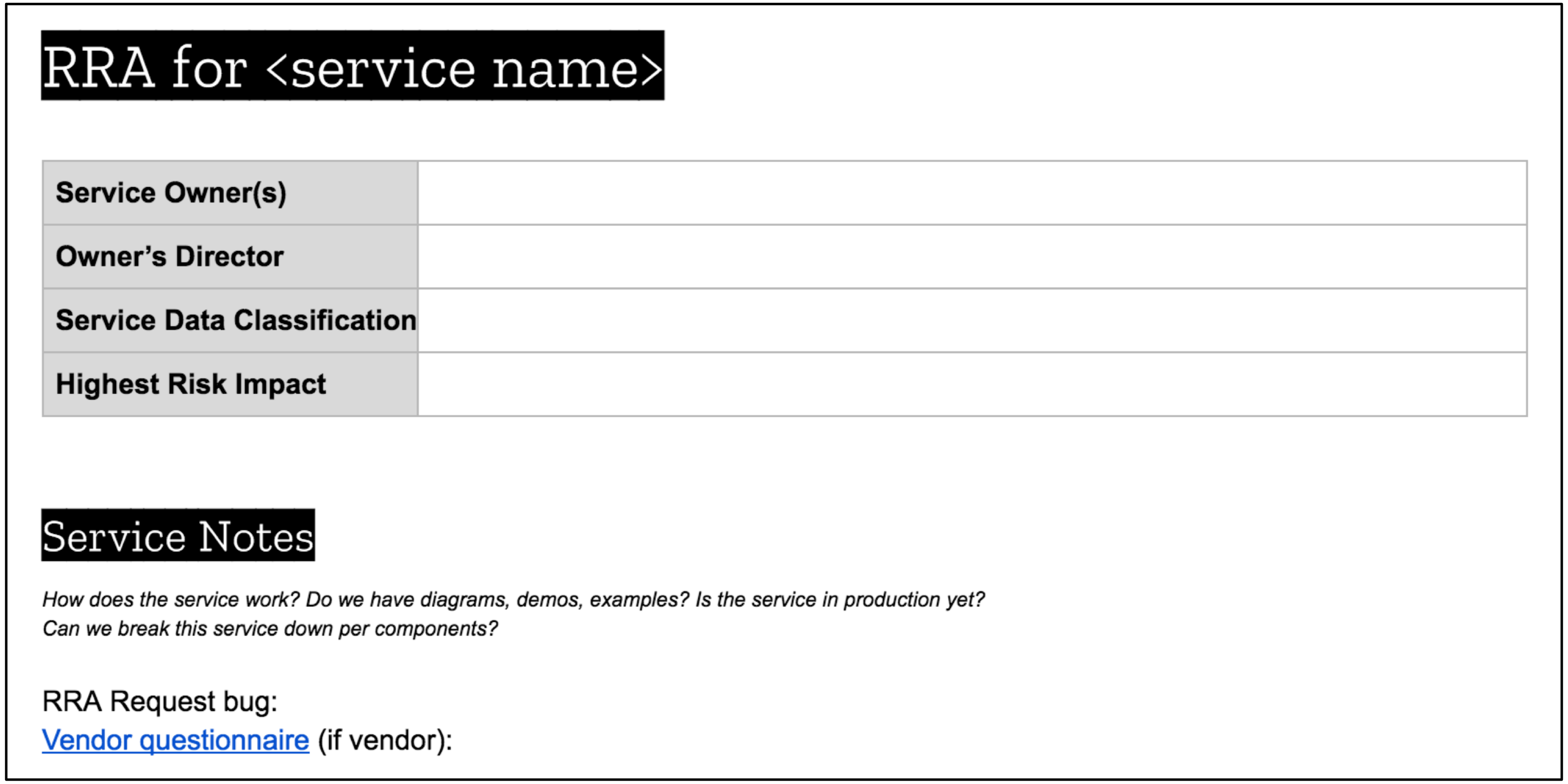
THREAT MODELING

Weaponizing the toolchain:

- OWASP User Security Stories
  - https://github.com/OWASP/user-security-stories
- OWASP Application Security Verification Standards
  - https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project
- Mozilla's Rapid Risk Assessment (RRA)
  - https://infosec.mozilla.org/guidelines/risk/rapid_risk_assessment.html
- OWASP Threat Dragon
  - https://www.owasp.org/index.php/OWASP_Threat_Dragon

Threat Modeling Example

Mozilla's rapid risk assessment guidance and Google Doc provide a blueprint for 30 minute RRAs:



RRA for <service name>

Service Owner(s)	
Owner’s Director	
Service Data Classification	
Highest Risk Impact	

Service Notes

How does the service work? Do we have diagrams, demos, examples? Is the service in production yet?
Can we break this service down per components?

RRA Request bug:
Vendor questionnaire (if vendor):

#2 IDE Security Plugins

IDE Security Plugins

Immediate, incremental scanning in each developer's IDE catches catch security mistakes as code is being changed/saved by the developer

- Security becomes part of the engineering workflow
- Shifting as far left as possible in the kill chain
- Must have low false positive rates (important)
- Run high value rules and disable noisy rules that distract engineers

IDE Security Plugin Tools

PRE-COMMIT

IDE SECURITY PLUGINS

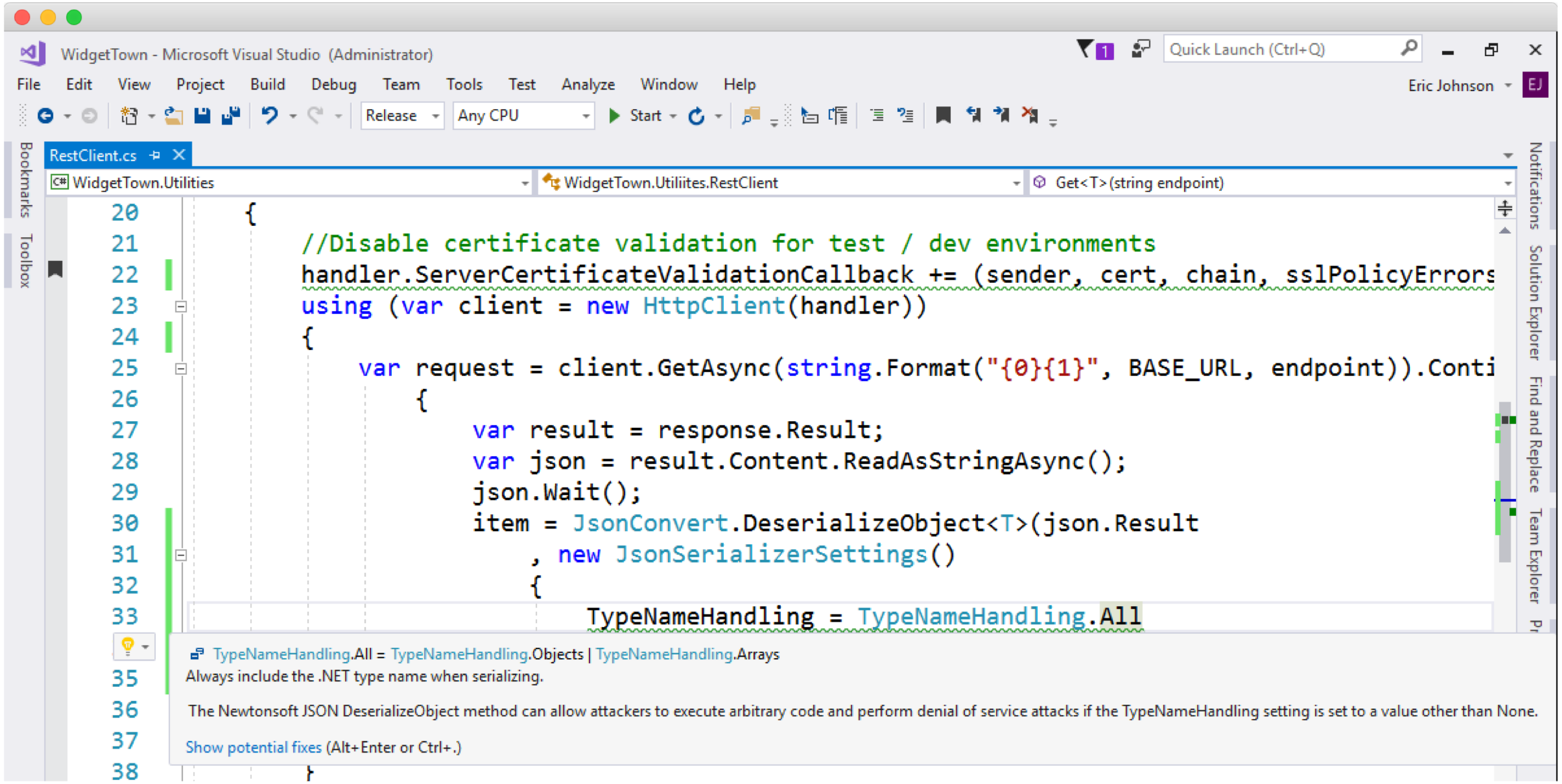
Weaponizing the toolchain:

- **FindSecurityBugs** plugin for Eclipse and IntelliJ
  - http://find-sec-bugs.github.io/
- **Puma Scan** plugin for Visual Studio
  - https://github.com/pumasecurity/puma-scan
- Microsoft's **DevSkim** for VSCode, Sublime, Visual Studio
  - https://github.com/Microsoft/DevSkim
- **SonarLint** plugins for Visual Studio, Intellij, and Eclipse
  - https://www.sonarlint.org/

Note: IDE plugins are also available for most commercial SAST products

IDE Security Plugin Example

Puma Scan identifying a JSON deserialization vulnerability:



```
    {
        //Disable certificate validation for test / dev environments
        handler.ServerCertificateValidationCallback += (sender, cert, chain, sslPolicyErrors
        using (var client = new HttpClient(handler))
        {
            var request = client.GetAsync(string.Format("{0}{1}", BASE_URL, endpoint)).Conti
            {
                var result = response.Result;
                var json = result.Content.ReadAsStringAsync();
                json.Wait();
                item = JsonConvert.DeserializeObject<T>(json.Result
                    , new JsonSerializerSettings()
                    {
                        TypeNameHandling = TypeNameHandling.All
```

TypeNameHandling.All = TypeNameHandling.Objects | TypeNameHandling.Arrays
Always include the .NET type name when serializing.

The Newtonsoft JSON DeserializeObject method can allow attackers to execute arbitrary code and perform denial of service attacks if the TypeNameHandling setting is set to a value other than None.

Show potential fixes (Alt+Enter or Ctrl+.)

```
        }
```

#3 Pre-Commit Hooks

Pre-Commit Hooks

- Git Hooks automatically run scripts at different points in workflows
  - Local: **pre-commit**, prepare-commit, commit, post-commit, post-checkout, pre-rebase
  - Server-side: **pre-receive**, update, **post-receive**
- Implement team-wide workflow policies, or check code for problems
- CAUTION: Repo owner can alter/uninstall hooks – so hooks cannot be enforced

Pre-Commit Hook Frameworks / Tools

PRE-COMMIT

PRE-COMMIT HOOKS

Weaponizing the toolchain:

- Open source frameworks to manage hooks for different languages + tools
  - Yelp pre-commit framework
  - Overcommit
- Pre-commit tools for scanning code:
  - AWS Labs git-secrets (https://github.com/awslabs/git-secrets)
  - Talisman (https://github.com/thoughtworks/talisman)
  - Auth0 repo-supervisor (https://github.com/auth0/repo-supervisor)

Pre-Commit Hook Example

AWS git-secrets blocking a commit that contains an access key and secret key id:

```
$ git commit -m "testing git-secrets"

Web/Licensing/appsettings.json:5:
    "AccessKey": "AKIAJNQ7C2FCRR6B4VWA",
Web/Licensing/appsettings.json:6:
    "SecretKey": "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"

[ERROR] Matched one or more prohibited patterns
```

#4 Peer Code Reviews

Peer Code Reviews

Disciplined peer code reviews are a fundamental engineering practice in DevOps: Google, Amazon, Facebook, Etsy, Twitter…

- Review for functional correctness (especially in high-risk code) and defensive coding
- Ensure that code takes advantage of secure framework capabilities and security libraries
- Watch out for hard-coded secrets, back doors, hand-rolled crypto!
- Leverage Static Analysis (SAST) to enforce good practices and catch common security/coding mistakes
- CAUTION: Developers need secure coding training, so they know what to look for

Peer Code Reviews | High Risk Code

Peer reviews should focus on high risk code, which may perform any of following functionality (not inclusive):

- Infrastructure Code
- Pipeline definitions
- Authentication
- Access control
- Output encoding
- Input validation

- Automated security / compliance tests
- High risk business logic
- Data entitlement checks
- Handling confidential data
- Cryptography

Peer Code Review Tools

PRE-COMMIT

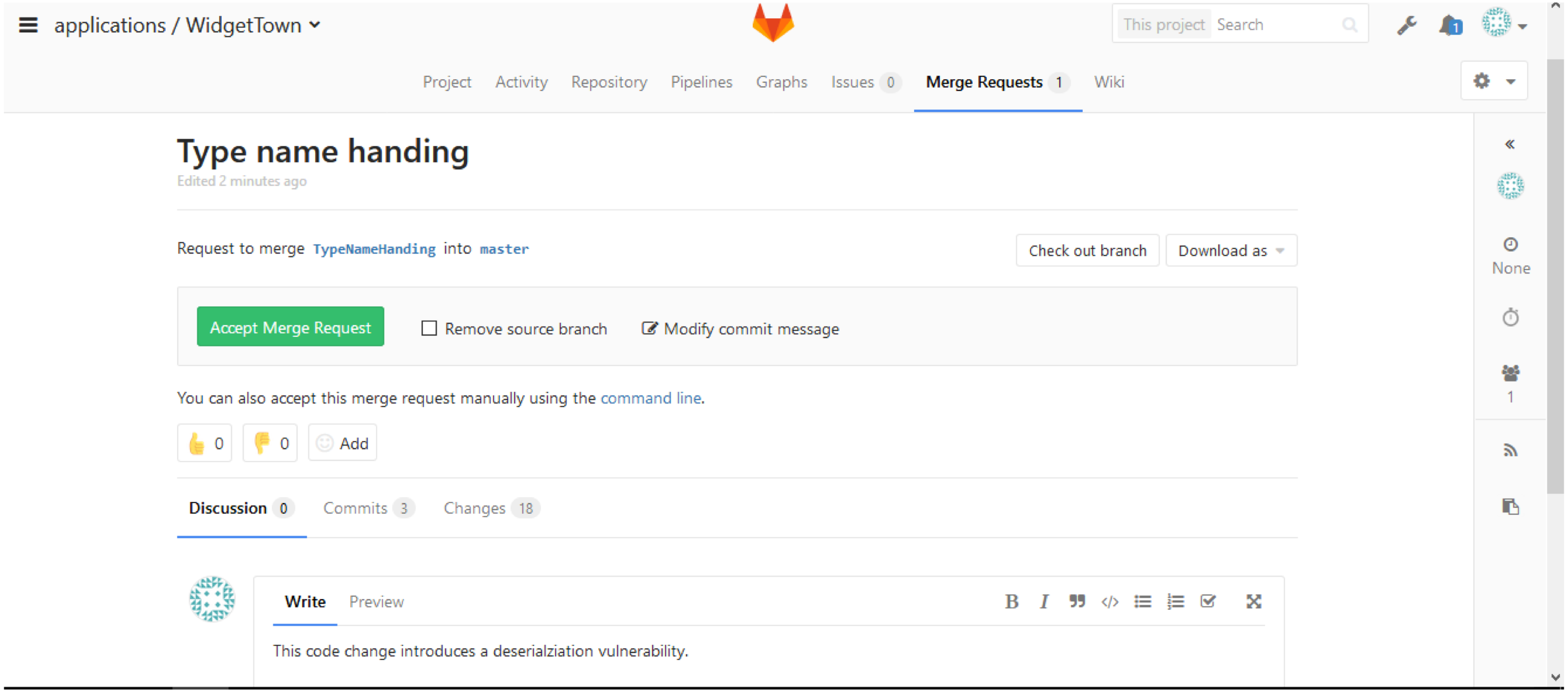
PEER CODE REVIEWS

Weaponizing the toolchain:

- Code review workflow tools enforce specific manual code review workflows and make it easy to involve multiple reviewers
  - Bitbucket/GitHub/GitLab pull request comments
  - Review Board or Gerrit (open source)
  - Atlassian Crucible
  - SmartBear Code Collaborator
  - Phabricator (from Facebook)

Peer Code Review Example

Gitlab pull request requiring peer review approval:



Type name handing

Edited 2 minutes ago

Request to merge `TypeNameHanding` into `master`

You can also accept this merge request manually using the command line.

This code change introduces a deserialziation vulnerability.

Agenda

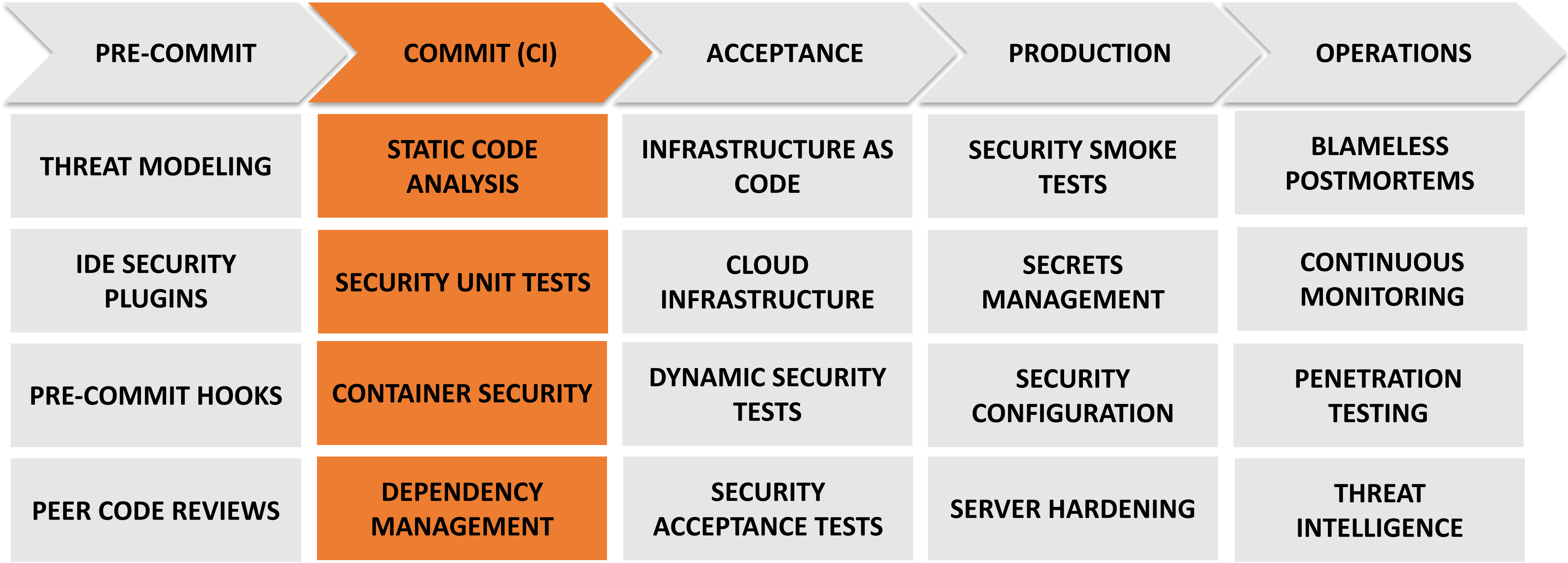
- Introduction
- Pre-Commit
- ***Commit***

Commit Stage

1. **Static Code Analysis**
2. **Security Unit Testing**
3. **Container Security**
4. **Dependency Management**

DevSecOps Commit Phase

Applying automated, fast, accurate security controls in the CI pipeline:



PRE-COMMIT	COMMIT (CI)	ACCEPTANCE	PRODUCTION	OPERATIONS
THREAT MODELING	STATIC CODE ANALYSIS	INFRASTRUCTURE AS CODE	SECURITY SMOKE TESTS	BLAMELESS POSTMORTEMS
IDE SECURITY PLUGINS	SECURITY UNIT TESTS	CLOUD INFRASTRUCTURE	SECRETS MANAGEMENT	CONTINUOUS MONITORING
PRE-COMMIT HOOKS	CONTAINER SECURITY	DYNAMIC SECURITY TESTS	SECURITY CONFIGURATION	PENETRATION TESTING
PEER CODE REVIEWS	DEPENDENCY MANAGEMENT	SECURITY ACCEPTANCE TESTS	SERVER HARDENING	THREAT INTELLIGENCE

#1 Static Code Analysis

Static Code Analysis in the Pipeline

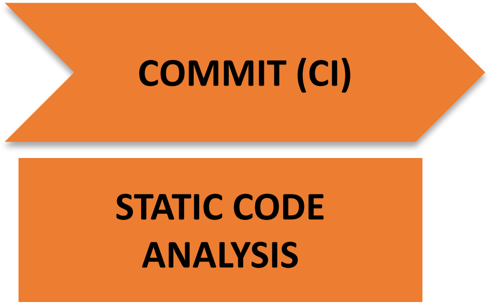
Limited opportunity to provide fast and clear feedback during commit and build:

- Automatically diff and scan changes, provide clear information on new findings to developers, feedback button to reject false positives
- Incremental scanning if possible – deep scanning takes too long for CI/CD, especially on large code bases.
- Run deep scans out of band
- Run scans in parallel with unit testing for speed
- Return results directly to engineers (IDE / backlog list)
- Minimize false positives by turning off rules / writing custom rules

Static Code Analysis Tools | Application Code

Weaponizing the toolchain:

- FindSecurityBugs (Java)
  - http://h3xstream.github.io/find-sec-bugs/
- Phan (PHP)
  - https://github.com/etsy/phan
- NodeJsScan (JavaScript)
  - https://github.com/ajinabraham/NodeJsScan
- Brakeman (Ruby)
  - http://brakemanscanner.org/
- Bandit (Python)
  - https://github.com/openstack/bandit



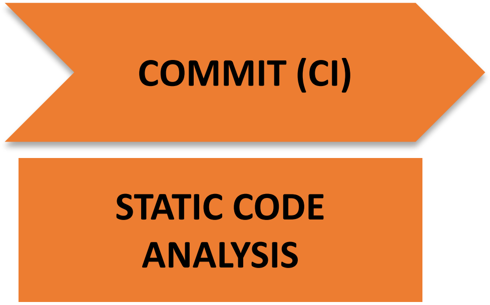
COMMIT (CI)

STATIC CODE ANALYSIS

Static Code Analysis Tools | Application Code

Weaponizing the toolchain (continued):

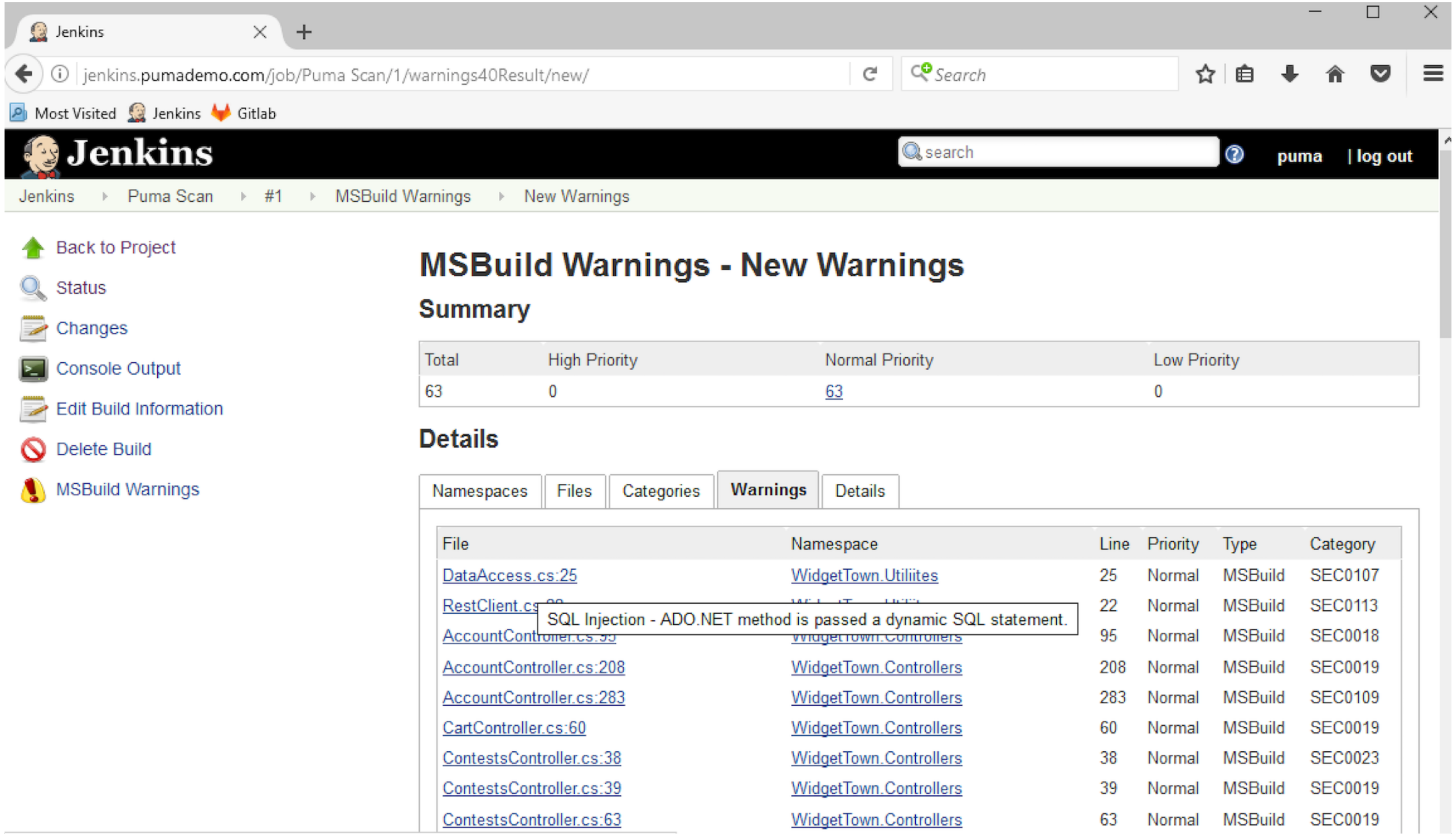
- Flawfinder (C)
  - http://www.dwheeler.com/flawfinder/
- Puma Scan (C#)
  - https://github.com/pumasecurity/puma-scan
- Gosec (Go)
  - https://github.com/GoASTScanner/gas


COMMIT (CI)

STATIC CODE ANALYSIS

Static Code Analysis Example in CI

Invoking a scan and capturing vulnerability data in a Jenkins CI pipeline:



jenkins.pumademo.com/job/Puma Scan/1/warnings40Result/new/

MSBuild Warnings - New Warnings

Summary

Total	High Priority	Normal Priority	Low Priority
63	0	63	0

Details

File	Namespace	Line	Priority	Type	Category
DataAccess.cs:25	WidgetTown.Utiliites	25	Normal	MSBuild	SEC0107
RestClient.cs:22	WidgetTown.Utiliites	22	Normal	MSBuild	SEC0113
AccountController.cs:95	WidgetTown.Controllers	95	Normal	MSBuild	SEC0018
AccountController.cs:208	WidgetTown.Controllers	208	Normal	MSBuild	SEC0019
AccountController.cs:283	WidgetTown.Controllers	283	Normal	MSBuild	SEC0109
CartController.cs:60	WidgetTown.Controllers	60	Normal	MSBuild	SEC0019
ContestsController.cs:38	WidgetTown.Controllers	38	Normal	MSBuild	SEC0023
ContestsController.cs:39	WidgetTown.Controllers	39	Normal	MSBuild	SEC0019
ContestsController.cs:63	WidgetTown.Controllers	63	Normal	MSBuild	SEC0019

SQL Injection - ADO.NET method is passed a dynamic SQL statement.

#2 Security Unit Testing

Security Unit Testing

Take advantage of engineering teams that are “test obsessed”:

- Get off the "happy path"!!
- Leverage “Evil User Stories”, “Abuse Cases”, and OWASP ASVS requirements to come up with test cases
- Ensure high levels of unit test coverage for high risk code
- **Red means STOP** – ensure team does not ignore/remove broken tests
- Write unit tests first when fixing vulnerabilities
- Use Unit tests to alert on changes to high risk code

Security Unit Testing Tools

Weaponizing the toolchain:

- JUnit (Java)
  - https://junit.org
- XUnit (C#, F#, VB)
  - https://xunit.github.io/
- Mocha (NodeJS)
  - https://mochajs.org/
- RSpec (Ruby)
  - http://rspec.info/
- PyUnit (Python)
  - https://wiki.python.org/moin/PyUnit

COMMIT (CI)

SECURITY UNIT TESTING

Security Unit Testing Example | Happy Path

The following code stays on the happy path by downloading Bob's license file:

```
[Theory]
[InlineData("bob@app.com", "L1ttleB0bbyTable$", "1", HttpStatusCode.Found)]
public async Task DownloadTest(string username, string password, string id,
                               HttpStatusCode responseCode)
{
   …
   var request = new HttpRequestMessage(HttpMethod.Get, $"/download/{id}");
   request.Headers.Add("Cookie", $"app-portal=${authCookie};");
   var response = await _client.SendAsync(request);
   Assert.Equal(responseCode, response.StatusCode);
}
```

Security Unit Testing Example | Evil Path

The following code performs an abuse case where Alice attempts to download Bob's license file:

```
[Theory]
[InlineData("bob@app.com", "L1ttleB0bbyTable$", "1", HttpStatusCode.Found)]
[InlineData("alice@app.com", "NotB0bbysPwd$", "1", HttpStatusCode.Forbidden)]
public async Task DownloadTest(string username, string password, string id,
                               HttpStatusCode responseCode)
{
   …
   var request = new HttpRequestMessage(HttpMethod.Get, $"/download/{id}");
   request.Headers.Add("Cookie", $"app-portal=${authCookie};");
   var response = await _client.SendAsync(request);
   Assert.Equal(responseCode, response.StatusCode);
}
```

#3 Container Security

Container Security Issues

- Lightweight isolation (do containers contain?)
- User namespacing is not enabled by default (added in Docker 1.10 Feb 2016)
- Untrusted content, compromised, and vulnerable images
- Docker Daemon presents its own attack surface
- Container sprawl and limited visibility, especially at scale
- Ephemeral run-time is difficult to track and manage

Container Security Resources

In-depth container security discussions could be a week-long discussion. Here are some resources to keep you busy:

- Docker Security Guidelines
- Docker Reference Architecture
- CIS Docker Benchmark
- NCC Group: Understanding and Hardening Linux Containers
- NIST SP 800-190 Application Container Security Guide
- CIS Kubernetes Benchmark

Container Security Tools

COMMIT (CI)

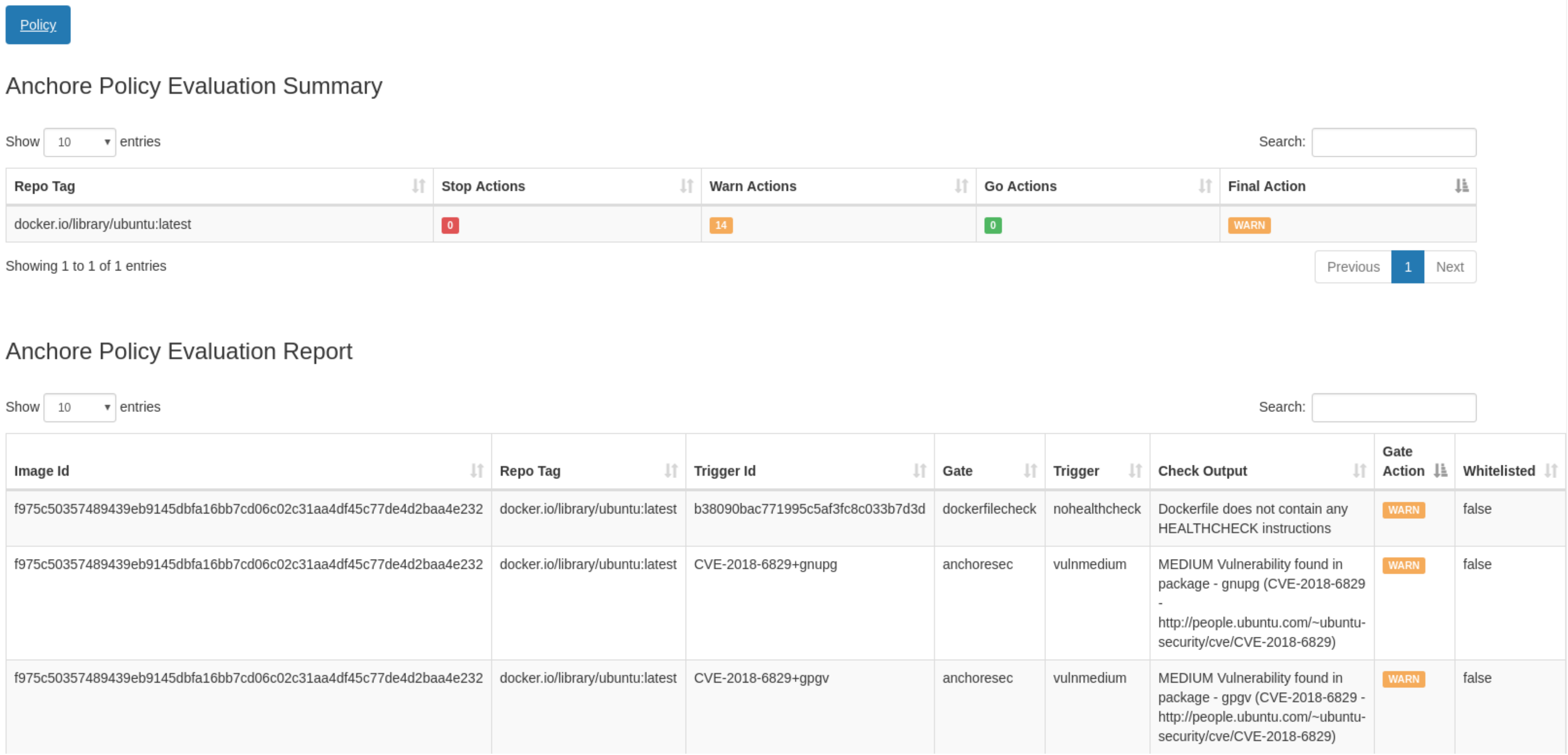
CONTAINER SECURITY

Weaponizing the toolchain:

- Docker Benchmark Inspec Profile
  - https://github.com/dev-sec/cis-docker-benchmark
- Anchore
  - https://anchore.com/opensource/
- Actuary
  - https://github.com/diogomonica/actuary
- Clair
  - https://github.com/coreos/clair
- Falco
  - https://github.com/draios/falco

Container Security Example

Invoking an Anchore image scan and capturing vulnerability data in a Jenkins CI pipeline:



Policy

Anchore Policy Evaluation Summary

Show 10 entries Search:

Repo Tag	Stop Actions	Warn Actions	Go Actions	Final Action
docker.io/library/ubuntu:latest	0	14	0	WARN

Showing 1 to 1 of 1 entries

Previous 1 Next

Anchore Policy Evaluation Report

Show 10 entries Search:

Image Id	Repo Tag	Trigger Id	Gate	Trigger	Check Output	Gate Action	Whitelisted
f975c50357489439eb9145dbfa16bb7cd06c02c31aa4df45c77de4d2baa4e232	docker.io/library/ubuntu:latest	b38090bac771995c5af3fc8c033b7d3d	dockerfilecheck	nohealthcheck	Dockerfile does not contain any HEALTHCHECK instructions	WARN	false
f975c50357489439eb9145dbfa16bb7cd06c02c31aa4df45c77de4d2baa4e232	docker.io/library/ubuntu:latest	CVE-2018-6829+gnupg	anchoresec	vulnmedium	MEDIUM Vulnerability found in package - gnupg (CVE-2018-6829 - http://people.ubuntu.com/~ubuntu-security/cve/CVE-2018-6829)	WARN	false
f975c50357489439eb9145dbfa16bb7cd06c02c31aa4df45c77de4d2baa4e232	docker.io/library/ubuntu:latest	CVE-2018-6829+gpgv	anchoresec	vulnmedium	MEDIUM Vulnerability found in package - gpgv (CVE-2018-6829 - http://people.ubuntu.com/~ubuntu-security/cve/CVE-2018-6829)	WARN	false

#4 Dependency Management

Dependency Management (Component Analysis)

Serious vulnerabilities can be inherited from open source libraries, docker images, and infrastructure templates:

- Use tools to automatically the scan code base or build artifacts and identify external dependencies (build a “bill of materials”)
- Identify out of date components
- Check against public vulnerability database(s) for known vulnerabilities in these components
- Many commercial tools also check for licensing risks or violations
- Caution that some tools may not check transitive dependencies within components
- Integrate into CI/CD—automatically fail build if serious problems are found

Dependency Management Tools

Weaponizing the toolchain:

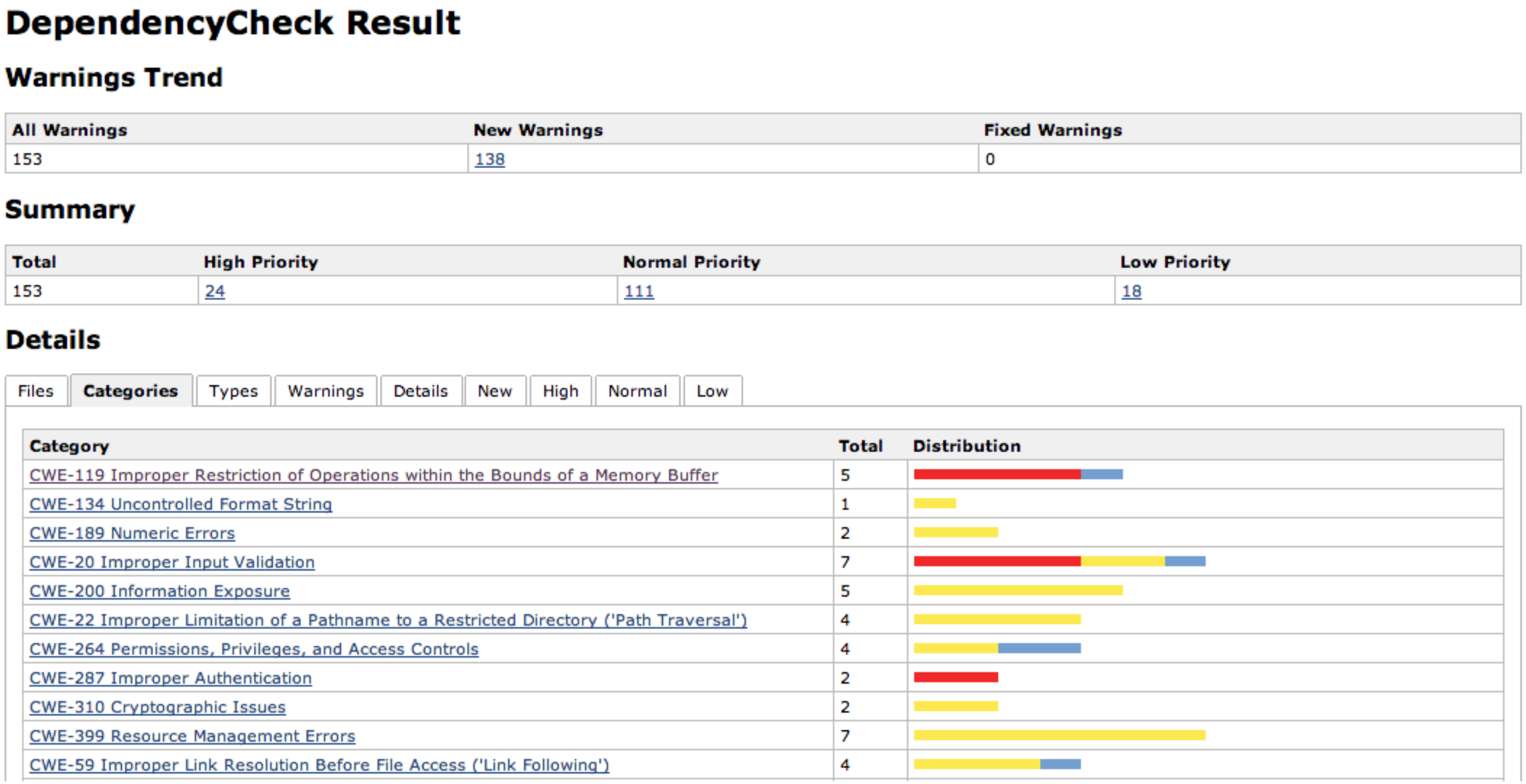
- OWASP Dependency Check (Java, .NET, Ruby, Python)
  - https://www.owasp.org/index.php/OWASP_Dependency_Check
- PHP Security Checker
  - https://security.sensiolabs.org/
- Bundler-Audit (Ruby)
  - https://github.com/rubysec/bundler-audit
- NPM Audit / Retire.JS (NodeJS)
  - https://retirejs.github.io/retire.js/
  - https://docs.npmjs.com/cli/audit

COMMIT (CI)

DEPENDENCY MANAGEMENT

Example of Dependency Analysis in CI

Invoking a dependency check scan and capturing vulnerability data in a Jenkins CI pipeline:



DependencyCheck Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
153	138	0

Summary

Total	High Priority	Normal Priority	Low Priority
153	24	111	18

Details

Files | Categories | Types | Warnings | Details | New | High | Normal | Low

Category	Total	Distribution
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer	5	🟥🟥🟥🟥🟦
CWE-134 Uncontrolled Format String	1	🟨
CWE-189 Numeric Errors	2	🟨🟨
CWE-20 Improper Input Validation	7	🟥🟥🟥🟥🟨🟨🟦
CWE-200 Information Exposure	5	🟨🟨🟨🟨🟨
CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	4	🟨🟨🟨🟨
CWE-264 Permissions, Privileges, and Access Controls	4	🟨🟨🟦🟦
CWE-287 Improper Authentication	2	🟥🟥
CWE-310 Cryptographic Issues	2	🟨🟨
CWE-399 Resource Management Errors	7	🟨🟨🟨🟨🟨🟨🟨
CWE-59 Improper Link Resolution Before File Access ('Link Following')	4	🟨🟨🟨🟦

DevSecOps Toolchain Summary

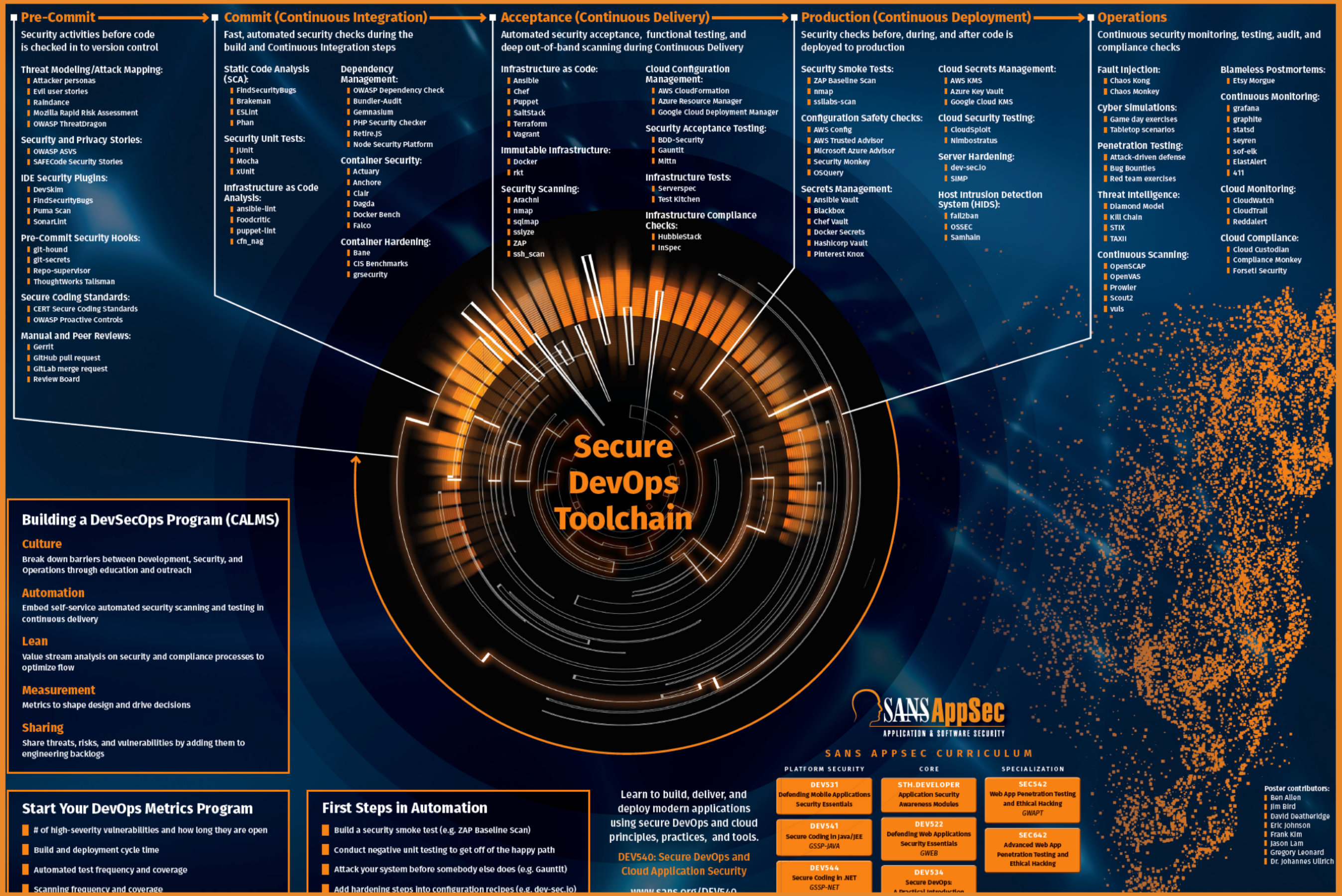
Exploring further…

PRE-COMMIT	COMMIT (CI)	ACCEPTANCE	PRODUCTION	OPERATIONS
THREAT MODELING	STATIC CODE ANALYSIS	INFRASTRUCTURE AS CODE	SECURITY SMOKE TESTS	BLAMELESS POSTMORTEMS
IDE SECURITY PLUGINS	SECURITY UNIT TESTS	CLOUD INFRASTRUCTURE	SECRETS MANAGEMENT	CONTINUOUS MONITORING
PRE-COMMIT HOOKS	CONTAINER SECURITY	DYNAMIC SECURITY TESTS	SECURITY CONFIGURATION	PENETRATION TESTING
PEER CODE REVIEWS	DEPENDENCY MANAGEMENT	SECURITY ACCEPTANCE TESTS	SERVER HARDENING	THREAT INTELLIGENCE

Thank you for attending!

ejohnson@sans.org

@emjohn20



Secure DevOps Toolchain

Pre-Commit
Security activities before code is checked in to version control

Threat Modeling/Attack Mapping:
- Attacker personas
- Evil user stories
- Raindance
- Mozilla Rapid Risk Assessment
- OWASP ThreatDragon

Security and Privacy Stories:
- OWASP ASVS
- SAFECode Security Stories

IDE Security Plugins:
- DevSkim
- FindSecurityBugs
- Puma Scan
- SonarLint

Pre-Commit Security Hooks:
- git-hound
- git-secrets
- Repo-supervisor
- ThoughtWorks Talisman

Secure Coding Standards:
- CERT Secure Coding Standards
- OWASP Proactive Controls

Manual and Peer Reviews:
- Gerrit
- GitHub pull request
- GitLab merge request
- Review Board

Commit (Continuous Integration)
Fast, automated security checks during the build and Continuous Integration steps

Static Code Analysis (SCA):
- FindSecurityBugs
- Brakeman
- ESLint
- Phan

Security Unit Tests:
- JUnit
- Mocha
- xUnit

Infrastructure as Code Analysis:
- ansible-lint
- Foodcritic
- puppet-lint
- cfn_nag

Dependency Management:
- OWASP Dependency Check
- Bundler-Audit
- Gemnasium
- PHP Security Checker
- Retire.JS
- Node Security Platform

Container Security:
- Actuary
- Anchore
- Clair
- Dagda
- Docker Bench
- Falco

Container Hardening:
- Bane
- CIS Benchmarks
- grsecurity

Acceptance (Continuous Delivery)
Automated security acceptance, functional testing, and deep out-of-band scanning during Continuous Delivery

Infrastructure as Code:
- Ansible
- Chef
- Puppet
- SaltStack
- Terraform
- Vagrant

Immutable Infrastructure:
- Docker
- rkt

Security Scanning:
- Arachni
- nmap
- sqlmap
- sslyze
- ZAP
- ssh_scan

Cloud Configuration Management:
- AWS CloudFormation
- Azure Resource Manager
- Google Cloud Deployment Manager

Security Acceptance Testing:
- BDD-Security
- Gauntlt
- Mittn

Infrastructure Tests:
- Serverspec
- Test Kitchen

Infrastructure Compliance Checks:
- HubbleStack
- InSpec

Production (Continuous Deployment)
Security checks before, during, and after code is deployed to production

Security Smoke Tests:
- ZAP Baseline Scan
- nmap
- ssllabs-scan

Configuration Safety Checks:
- AWS Config
- AWS Trusted Advisor
- Microsoft Azure Advisor
- Security Monkey
- OSQuery

Secrets Management:
- Ansible Vault
- Blackbox
- Chef Vault
- Docker Secrets
- Hashicorp Vault
- Pinterest Knox

Cloud Secrets Management:
- AWS KMS
- Azure Key Vault
- Google Cloud KMS

Cloud Security Testing:
- CloudSploit
- Nimbostratus

Server Hardening:
- dev-sec.io
- SIMP

Host Intrusion Detection System (HIDS):
- fail2ban
- OSSEC
- Samhain

Operations
Continuous security monitoring, testing, audit, and compliance checks

Fault Injection:
- Chaos Kong
- Chaos Monkey

Cyber Simulations:
- Game day exercises
- Tabletop scenarios

Penetration Testing:
- Attack-driven defense
- Bug Bounties
- Red team exercises

Threat Intelligence:
- Diamond Model
- Kill Chain
- STIX
- TAXII

Continuous Scanning:
- OpenSCAP
- OpenVAS
- Prowler
- Scout2
- vuls

Blameless Postmortems:
- Etsy Morgue

Continuous Monitoring:
- grafana
- graphite
- statsd
- seyren
- sof-elk
- ElastAlert
- 411

Cloud Monitoring:
- CloudWatch
- CloudTrail
- Reddalert

Cloud Compliance:
- Cloud Custodian
- Compliance Monkey
- Forseti Security

Building a DevSecOps Program (CALMS)

Culture
Break down barriers between Development, Security, and Operations through education and outreach

Automation
Embed self-service automated security scanning and testing in continuous delivery

Lean
Value stream analysis on security and compliance processes to optimize flow

Measurement
Metrics to shape design and drive decisions

Sharing
Share threats, risks, and vulnerabilities by adding them to engineering backlogs

Start Your DevOps Metrics Program
- # of high-severity vulnerabilities and how long they are open
- Build and deployment cycle time
- Automated test frequency and coverage
- Scanning frequency and coverage

First Steps in Automation
- Build a security smoke test (e.g. ZAP Baseline Scan)
- Conduct negative unit testing to get off of the happy path
- Attack your system before somebody else does (e.g. Gauntlt)
- Add hardening steps into configuration recipes (e.g. dev-sec.io)

Learn to build, deliver, and deploy modern applications using secure DevOps and cloud principles, practices, and tools.

DEV540: Secure DevOps and Cloud Application Security

www.sans.org/DEV540

SANS AppSec
APPLICATION & SOFTWARE SECURITY

SANS APPSEC CURRICULUM

PLATFORM SECURITY	CORE	SPECIALIZATION
DEV531 Defending Mobile Applications Security Essentials	STH.DEVELOPER Application Security Awareness Modules	SEC542 Web App Penetration Testing and Ethical Hacking *GWAPT*
DEV541 Secure Coding in Java/JEE *GSSP-JAVA*	DEV522 Defending Web Applications Security Essentials *GWEB*	SEC642 Advanced Web App Penetration Testing and Ethical Hacking
DEV544 Secure Coding in .NET *GSSP-NET*	DEV534 Secure DevOps: A Practical Introduction	

Poster contributors:
- Ben Allen
- Jim Bird
- David Deatheridge
- Eric Johnson
- Frank Kim
- Jason Lam
- Gregory Leonard
- Dr. Johannes Ullrich