Continuous Security: Exploring the DevOps Toolchain
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

**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

@sansappsec software-security.sans.org
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
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 **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
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

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
DevSecOps Security Controls

Breaking down the security controls in each DevSecOps phase:

- **PRE-COMMIT**
  - THREAT MODELING
  - IDE SECURITY PLUGINS
  - PRE-COMMIT HOOKS
  - PEER CODE REVIEWS

- **COMMIT (CI)**
  - STATIC CODE ANALYSIS
  - SECURITY UNIT TESTS
  - CONTAINER SECURITY
  - DEPENDENCY MANAGEMENT

- **ACCEPTANCE**
  - INFRASTRUCTURE AS CODE
  - CLOUD INFRASTRUCTURE
  - DYNAMIC SECURITY TESTS
  - SECURITY ACCEPTANCE TESTS

- **PRODUCTION**
  - SECURITY SMOKE TESTS
  - SECURITY CONFIGURATION
  - SECURITY ACCEPTANCE TESTS
  - SERVER HARDENING

- **OPERATIONS**
  - BLAMELESS POSTMORTEMS
  - CONTINUOUS MONITORING
  - PENETRATION TESTING
  - THREAT INTELLIGENCE
Agenda

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

<table>
<thead>
<tr>
<th>Pre-Commit Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Threat Modeling</td>
</tr>
<tr>
<td>2. IDE Security Plugins</td>
</tr>
<tr>
<td>3. Pre-Commit Hooks</td>
</tr>
<tr>
<td>4. Peer Code Reviews</td>
</tr>
</tbody>
</table>
DevSecOps Pre-Commit Phase

Applying security controls before code is written and committed:

<table>
<thead>
<tr>
<th>Pre-Commit</th>
<th>Commit (CI)</th>
<th>Acceptance</th>
<th>Production</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threat Modeling</td>
<td>Static Code Analysis</td>
<td>Infrastructure As Code</td>
<td>Security Smoke Tests</td>
<td>Blameless Postmortems</td>
</tr>
<tr>
<td>IDE Security Plugins</td>
<td>Security Unit Tests</td>
<td>Cloud Infrastructure</td>
<td>Secrets Management</td>
<td>Continuous Monitoring</td>
</tr>
<tr>
<td>Peer Code Reviews</td>
<td>Dependency Management</td>
<td>Security Acceptance Tests</td>
<td>Server Hardening</td>
<td>Threat Intelligence</td>
</tr>
</tbody>
</table>
#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

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>

<table>
<thead>
<tr>
<th>Service Owner(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner's Director</td>
</tr>
<tr>
<td>Service Data Classification</td>
</tr>
<tr>
<td>Highest Risk Impact</td>
</tr>
</tbody>
</table>

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
Immediate, incremental scanning in each developer’s IDE catches 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

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:
#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

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

```
1 $ git commit -m "testing git-secrets"
2
3 Web/Licensing/appsettings.json:5:
4   "AccessKey": "AKIAJNQ7C2FCRR6B4VWA",
5 Web/Licensing/appsettings.json:6:
6   "SecretKey": "ry8F6PlPTBP4bFGqZ0IzvZ71Oht2gkgZvFK/CZecw"
7
8 [ERROR] Matched one or more prohibited patterns
```
#4 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 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
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:
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:

<table>
<thead>
<tr>
<th>Pre-commit</th>
<th>Commit (CI)</th>
<th>Acceptance</th>
<th>Production</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threat Modeling</td>
<td>Static Code Analysis</td>
<td>Infrastructure as Code</td>
<td>Security Smoke Tests</td>
<td>Blameless Postmortems</td>
</tr>
<tr>
<td>Ide Security Plugins</td>
<td>Security Unit Tests</td>
<td>Cloud Infrastructure</td>
<td>Secrets Management</td>
<td>Continuous Monitoring</td>
</tr>
<tr>
<td>Peer Code Reviews</td>
<td>Dependency Management</td>
<td>Security Acceptance Tests</td>
<td>Server Hardening</td>
<td>Threat Intelligence</td>
</tr>
</tbody>
</table>
#1 Static Code Analysis
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
Weaponizing the toolchain:

- **FindSecurityBugs (Java)**
- **Phan (PHP)**
  - [https://github.com/etsy/phan](https://github.com/etsy/phan)
- **NodeJsScan (JavaScript)**
  - [https://github.com/ajinabraham/NodeJsScan](https://github.com/ajinabraham/NodeJsScan)
- **Brakeman (Ruby)**
  - [http://brakemanscanner.org/](http://brakemanscanner.org/)
- **Bandit (Python)**
  - [https://github.com/openstack/bandit](https://github.com/openstack/bandit)
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
Static Code Analysis Example in CI

Invoking a scan and capturing vulnerability data in a Jenkins CI pipeline:
#2 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
The following code stays on the happy path by downloading Bob's license file:

```
[Theory]
[InlineData("bob@app.com", "LittleBobbyTable$", "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);
}
```
The following code performs an abuse case where Alice attempts to download Bob's license file:

```csharp
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
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
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:

Anchore Policy Evaluation Summary

Anchore Policy Evaluation Report

<table>
<thead>
<tr>
<th>Image Id</th>
<th>Repo Tag</th>
<th>Gate</th>
<th>Trigger</th>
<th>Check Output</th>
<th>Gate Action</th>
<th>Whitelisted</th>
</tr>
</thead>
<tbody>
<tr>
<td>f75c5035748439eb9d14d5df8d1b747b7cb6c02c33a4df5c77e6d41baa4e232</td>
<td>docker.io/library/ubuntu latest</td>
<td>dockerfilecheck</td>
<td></td>
<td>Dockerfile does not contain any HEALTHCHECK instructions</td>
<td></td>
<td>false</td>
</tr>
</tbody>
</table>
#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
Example of Dependency Analysis in CI

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

### Dependency Check Result

#### Warnings Trend

<table>
<thead>
<tr>
<th>All Warnings</th>
<th>New Warnings</th>
<th>Fixed Warnings</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>138</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Summary

<table>
<thead>
<tr>
<th>Total</th>
<th>High Priority</th>
<th>Normal Priority</th>
<th>Low Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>24</td>
<td>111</td>
<td>18</td>
</tr>
</tbody>
</table>

#### Details

<table>
<thead>
<tr>
<th>Category</th>
<th>Total</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CWE-134 Uncontrolled Format String</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CWE-189 Numeric Errors</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>CWE-20 Improper Input Validation</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>CWE-200 Information Exposure</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>CWE-22 Improper Linkation to a Restricted Directory ('Path Traversal')</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CWE-264 Permissions, Privileges, and Access Controls</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>CWE-267 Improper Authentication</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>CWE-310 Cryptographic Issues</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>CWE-399 Resource Management Errors</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>CWE-59 Improper Link Resolution Before File Access ('Link Following')</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
## DevSecOps Toolchain Summary

### Exploring further...

<table>
<thead>
<tr>
<th>Pre-commit</th>
<th>Commit (CI)</th>
<th>Acceptance</th>
<th>Production</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threat Modeling</td>
<td>Static Code</td>
<td>Infrastructure</td>
<td>Security Smoke Tests</td>
<td>Blameless Postmortems</td>
</tr>
<tr>
<td>IDE Security Plugins</td>
<td>Security Unit Tests</td>
<td>Cloud Infrastructure</td>
<td>Secrets Management</td>
<td>Continuous Monitoring</td>
</tr>
<tr>
<td>Peer Code Reviews</td>
<td>Dependency Management</td>
<td>Security Acceptance Tests</td>
<td>Server Hardening</td>
<td>Threat Intelligence</td>
</tr>
</tbody>
</table>
Thank you for attending!

ejohnson@sans.org
@emjohn20