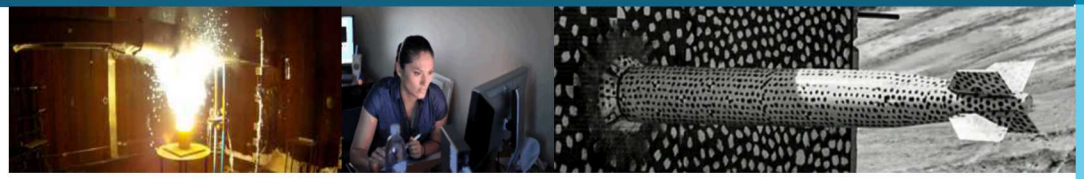
SAND2019-5102C

Secure Software Development Practices at Sandia National Laboratories



PRESENTED BY

Angela (Ang) Rivas and Michael Coram

acrivas@sandia.gov mcoram@sandia.gov

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

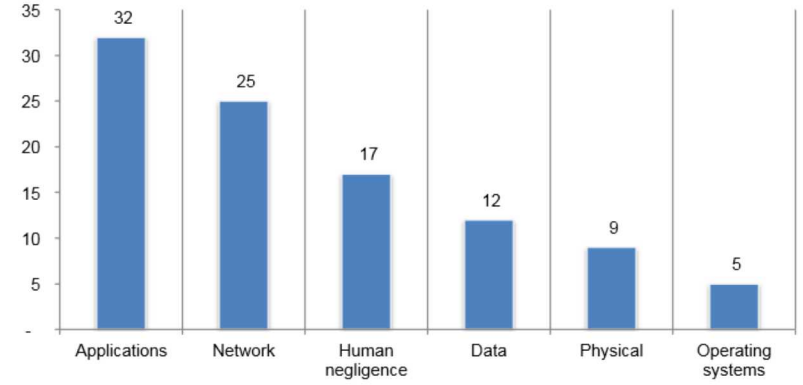
Background

Describing the Current State of Software Security

Ponemon Institute State of Application Security Risk Management (2016)

- Security pros view the application layer as extremely vulnerable
- Lack of application security testing
  - Only 14% do so throughout the development life cycle
  - **46% don't test at all**

Figure 7. Where do security compromises most likely occur?
100 points allocated based on the level of risk presented by each layer



Building Security In Maturity Model (BSIMM) (2017)

- **33% of firms use threat modeling** to identify potential attackers
- **62% of firms surveyed use internal penetration** testing tools on their projects
  - Only 6% of the firms customize these tools to accommodate changing environments or specific needs
- 4% of respondents state their projects use containers specifically to improve their application security posture
- 4% of the organizations use application behavior monitoring and diagnostics

Understanding Software Security at Sandia

Key Insights from Survey of the State of Software Security Practices at Sandia National Laboratories

- **Developers want** an official source for information and resources
- **Developers want** relevant training
- Some projects use security scanning tools, but do so independently and with little or no guidance to evaluate results

Formalizing Software Security at Sandia

Mission:

Secure software, from the start.

Vision:

We build highly adaptive, self-securing, cyber resilient software and data systems.

Establish a Software Security Group (SSG)

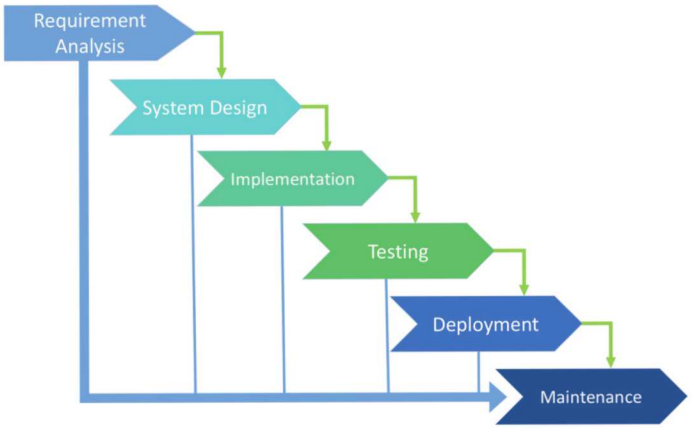
Enable a culture of secure software development

Provide enterprise services.

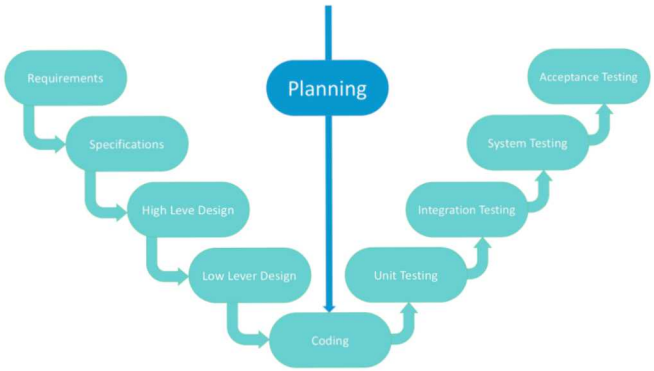
Build a program to develop technologies and methods to secure data and software from cyber-attacks

Establish and advance the Sandia strategy for data and software security

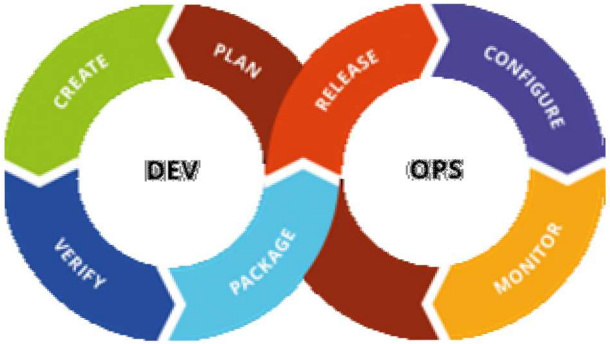
Reviewing Existing Software Development Lifecycles



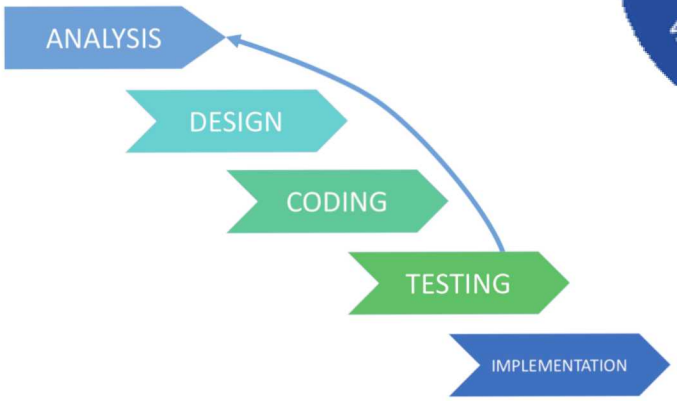
Waterfall



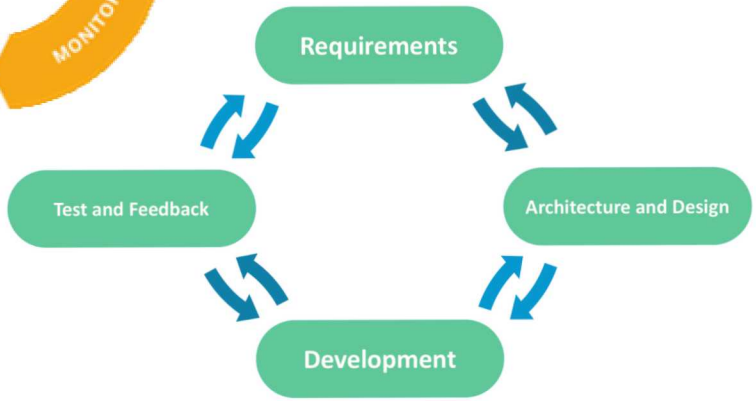
V-Shaped



DevOps



Iterative

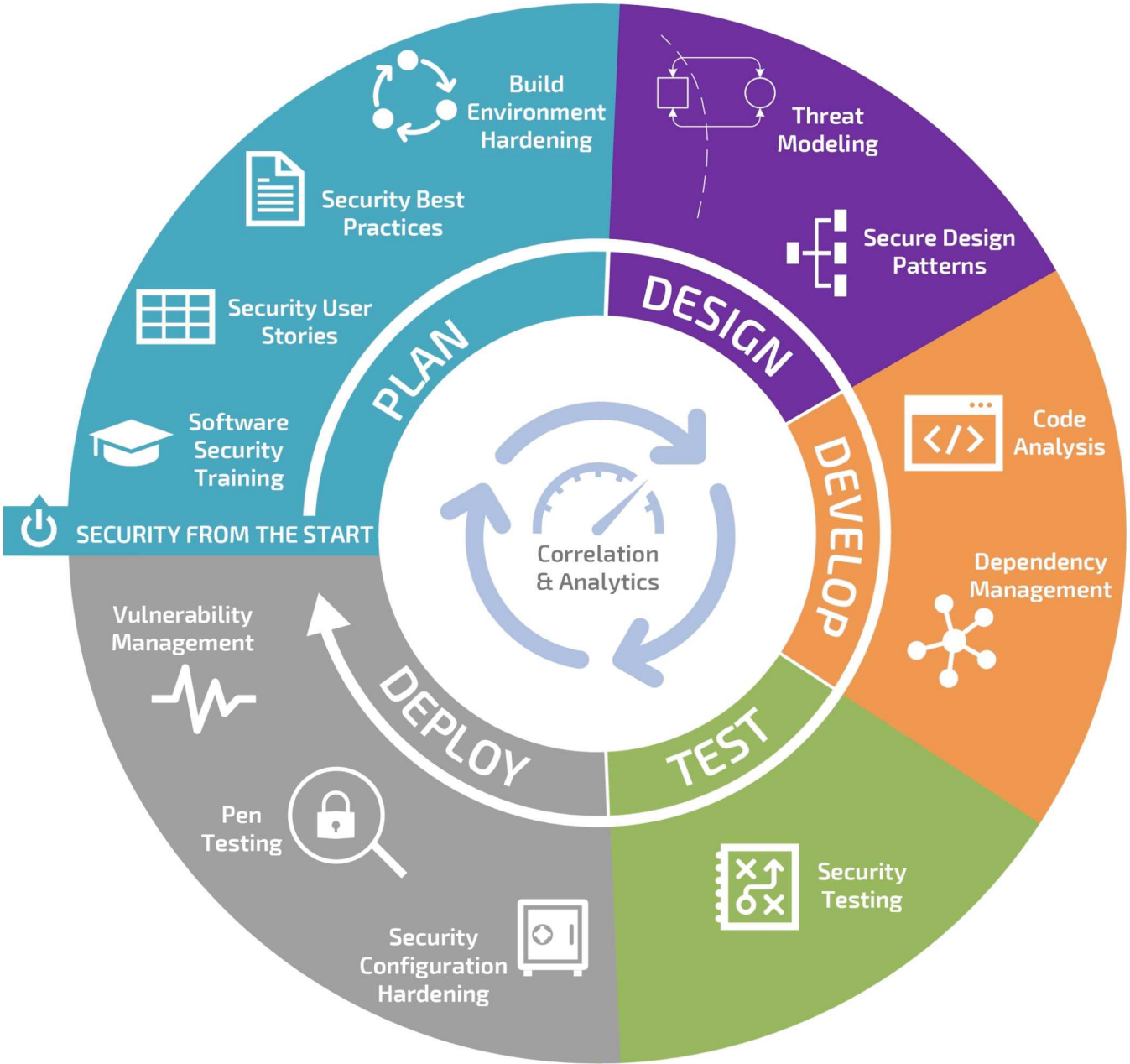


Agile

https://existek.com/blog/sdlc-models/

https://www.alliedcode.com/software-devops/devops-graphic/

Redefining the Secure Software Development Lifecycle (SDLC)



Empower development teams to be software security champions by offering awareness and technical training.

- Software Security Awareness Training
- Developer Technical Training
- Developer Secure Software Development Lifecycle (SDLC) Training and Application
- Developer Deep Dive Dialogues

> [W]e should be spending money on security training for developers. These are people who can be taught expertise in a fast-changing environment, and this is a situation where raising the average behavior increases the security of the overall system.
>
> Bruce Schneier

Identify and account for software security risks and activities through user stories and tasks.

Think about how you will address security topics (CIA triad, authentication, etc.).

Even if you aren't working in an agile environment, understand what the bad guys might want to do (internal or external) and document it.

Security user stories, evil user stories, and abuser stories are all ways of capturing malicious ways your system can be used (so you are aware and can fix it).

As a user, I should only be able to see my information and not other employees'

As a foreign adversary, I want to covertly and subtlety change how the system works

As a malicious insider, I want to steal classified information to which I am unauthorized

Ensure that best practices for secure coding are available and implemented in development projects.

Establish secure design best practices that can be applied to any project.

Establish language and framework specific guidance for security

- Memory management issues differ substantially between Java and C/C++
- Mitigating CSRF will differ between Struts and Spring

Provide checklist templates for secure code reviews

- Encourage teams to augment based on their security risks and level of experience

Develop template applications, examples, and reference architectures

- Make it easy for developers to add security from the start

Make training materials publicly available and vet resources for developers

Publish this information on a Secure Software Portal

Lead an awareness campaign (#SecureSoftware)

Harden development environments used to build and the test the application.

Hardening is something that you have to do to prevent malicious injection of code.

Restrict who can submit code and run builds.

Protect your test environments, especially if they include real data

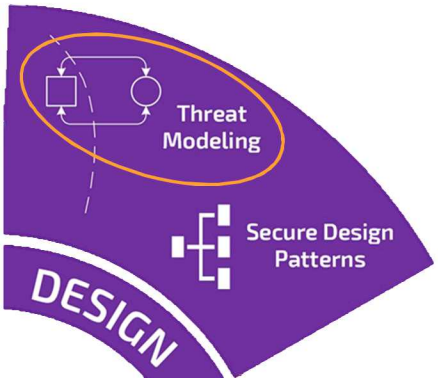
- Use containers and VMs.
- Disable unnecessary accounts, software, services, and ports.

Maintain and backup logs (preferably with monitoring and alerts).

Encrypt data at rest and in motion.

Collaborate with development teams to perform threat modeling on the system's architecture.

- What are you building?
- What can go wrong?
- What are you doing to defend against threats?
- Validate previous steps
- Report



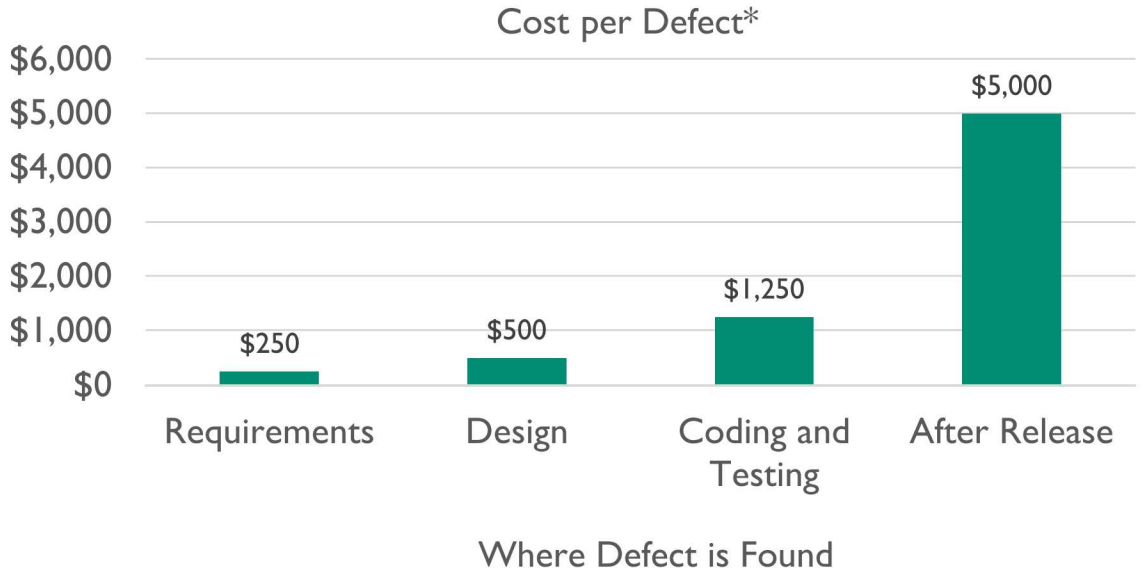
If we had our hands tied behind our backs (we don't) and could do only one thing to improve software security—threat modeling, better security code reviews, or better security testing—we would do threat modeling every day of the week.

Michael Howard and Steve Lipner

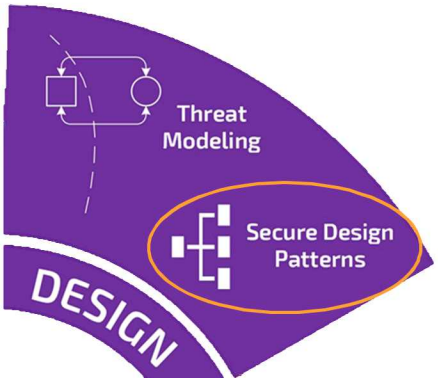
Provide a library of secure design patterns that can be utilized by development teams to address or mitigate security vulnerabilities and weaknesses.

Design patterns are a way of providing a general solution to a security problem, offering one more way to improve software security early on (in the design phase) where it's cheaper to fix, rather than later in the lifecycle.

Patterns may be applied to architecture, design, or development.



Capers Jones, A Short History of the Cost Per Defect Metric



Provide tools and assistance to development teams to help them analyze the code and application to find vulnerabilities as the software is developed.

Use Static Analysis Security Testing (SAST) tools

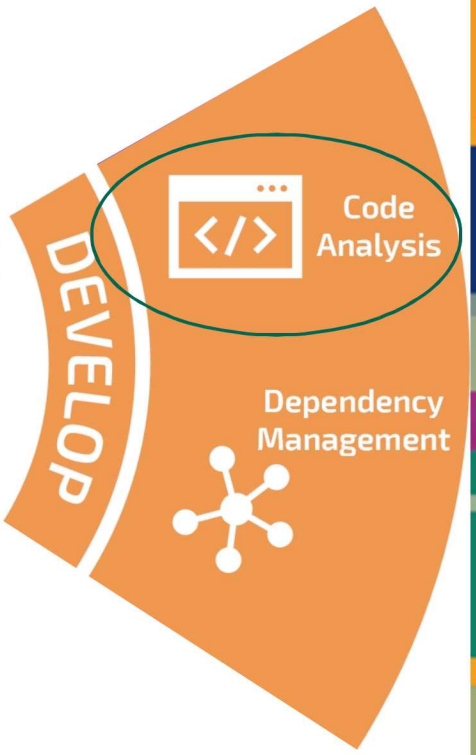
- Automatically finds potential defects, but resolution requires expertise
- Will not find all security vulnerabilities, so encourage additional analysis
- Likely to have many false positives, so help teams find the real issues
- Help developers integrate them into both developer tools (e.g. IDE) and the build environment (e.g. CI/CD)

Use Dynamic Analysis Security Testing (DAST) too

- Scans running software, potentially finding issues SAST mis
- Allow developers to run ad hoc scans to identify defects earl
- Integrate into the build environment

Perform manual security reviews

- Encourage teams to integrate into existing code review pro
- Provide checklists to help teams perform reviews
- Assist in focused reviews of security significant componer



Ensure teams are managing their dependencies and mitigating vulnerabilities

Open Source Software is a major risk

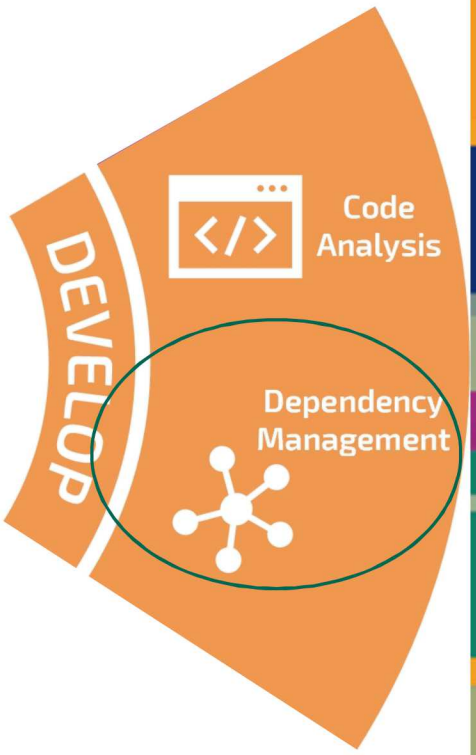
- 60% percent of enterprise codebases contain open-source vulnerabilities[1]
- 88% increase in application vulnerabilities between 2016 and 2018[2]
- 78% of vulnerabilities are found in indirect dependencies[2]
- In the average application, over a third of the code base is open source[3]

Utilize Software Component Analysis (SCA) software

- Integrate into the build environment (CI/CD) to detect issues prior to deployment
- Ensure proper mitigation of security issues
- Continually check software for new vulnerabilities

Create a curated repository of trusted software

- Do not let vulnerable software into your environment
- Developers and build tools download libraries from the trusted repository rather than the Internet
- Make sure to keep the repository updated!



[1] https://www.zdnet.com/article/60-percent-of-codebases-contain-open-source-vulnerabilities/
[2] https://snyk.io/opensourcesecurity-2019/
[3] https://www.csoonline.com/article/3157377/open-source-software-security-challenges-persist.html

Develop and execute security tests specific to the application, such as access control tests

Test early and often – even though it's a phase of the secure SDLC, testing should be integrated throughout the lifecycle

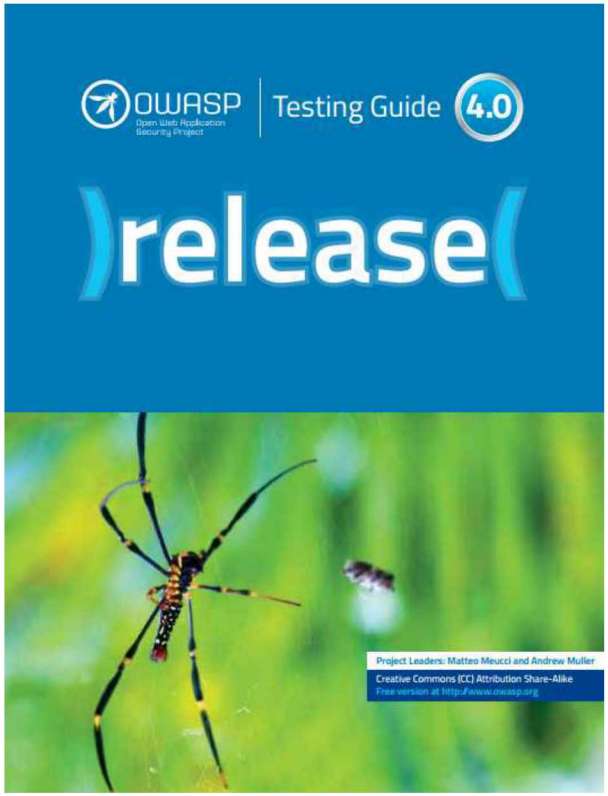
Includes functional, non-functional, and regression

Think with an adversarial or "let's break it" mindset

Use the right tools – especially to automate!

Remember those use and abuse cases? Test them!

Security testing can help you find areas for improvement – so that in addition to being more secure, you can also be more efficient!



Help teams configure their deployment environments to be hardened against attacks.

Use Defense in Depth

- Do not rely on a single defense

Keep up with security patches

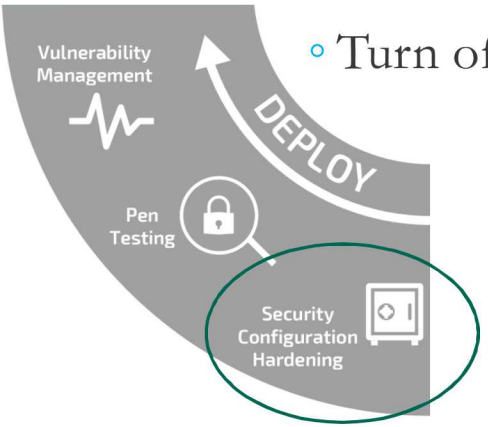
- Patch Tuesday can lead to Exploit Wednesday

Control who has administrative permissions

- Consider the entire stack – container, virtual machine, host machine
- Use Principle of Least Privilege

Reduce the attack surface

- Use firewalls, including Web Application Firewalls
- Turn off unnecessary services



Insecure deployment environments can make the most secure software vulnerable

Perform independent penetration testing as a final validation that security issues have been resolved.

For security significant and high risk applications, penetration testing by skilled individuals can help validate

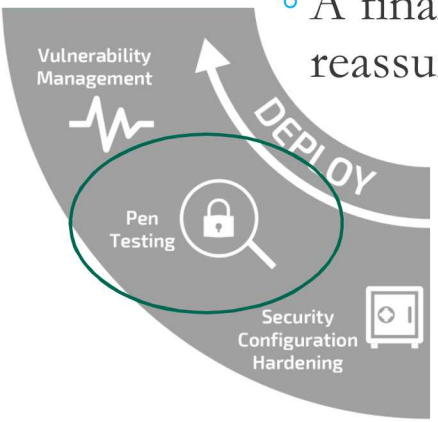
- Recommend that a specialized team, independent of the Secure Software Group, perform this activity

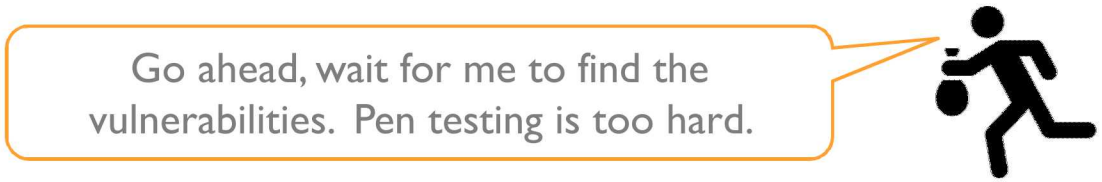
Consider "light red teaming" for applications with lower risk

- Utilize the Secure Software Group to grow their skills, encourage an adversarial mindset, and validate work performed in prior steps

Following this Secure Software Development lifecycle minimizes the chance the pen testing will find an issue

- A final check, performed as an attacker, can provide validation and reassurance





Encourage teams to manage vulnerabilities throughout the application's lifetime

Need to continue to monitor for vulnerabilities and address them

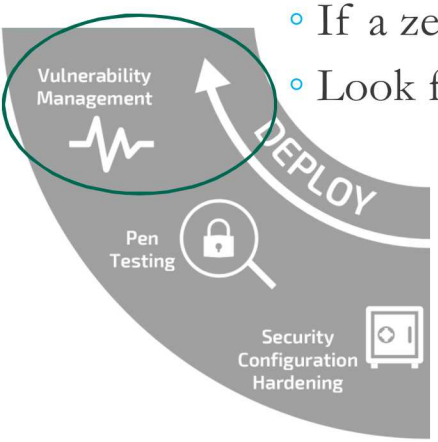
- New vulnerabilities in Open Source Software can be reported
- Users can find security significant bugs
- Attackers can find issues that were missed

Bug fixes and new features will be needed

- Consider risk of introducing security vulnerabilities, especially for emergency fixes
- Repeat the security software lifecycle where possible, abbreviating where appropriate based on risk

Monitor logs

- If a zero-day exists, logs may be the only evidence
- Look for abnormal behavior – intrusion detection



Just because you deployed, does not mean you are done with security

Collect data throughout the process, analyze it, and correlate with opportunities for improvement.

Continually improve your secure development practices

- Perform root cause analysis to determine how a vulnerability was introduced
  - Consider how the issue could have been found earlier or avoided completely
- Focus on making the process easier for developers to follow
- Automate and optimize where possible

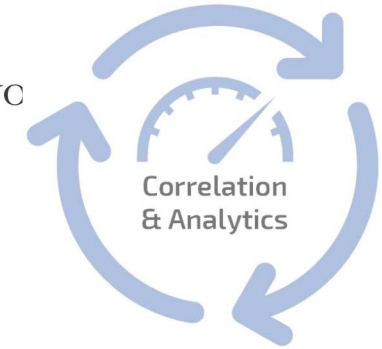
Share lessons learned across teams

- Capture new security best practices
- Identify common causes of vulnerabilities and methods to address
- Share tips and tricks for using security tools

Identify areas for future research

- Keep current on new tools, techniques, and threats
- Develop your own tools or extend existing ones to better meet yo
- Drive the state-of-the-art forward

Your adversaries will not rest, so neither should you



Summary

We are leading change by moving security earlier in the development process and providing a comprehensive approach that integrates and automates security through the development lifecycle.

By following these steps, regardless of the programming methodology or framework being used, SNL believes that the security of software applications can be improved, reducing both the risk and cost to the organization than if vulnerabilities are found after the product has deployed.

Learn more at NLIT!

Designing Security into Software Systems using Threat Modeling
10:00 AM - 10:40 AM : Room 120A
Gary Huang

Cross-Site Request Forgery Challenges and Solutions
10:45 AM - 11:25 AM : Room 110B
Michael Coram

Choosing Static Application Security Testing Tools
10:45 AM - 11:25 PM : Room 110B
Dr. Roger Hartley

Using Awareness and Training to Enable Secure Software Development at Sandia National Laboratories
10:00 AM - 10:40 AM : Room 120C
Angela Rivas

Benchmarking DevSecOps using Enterprise Search at Sandia National Laboratories
9:15 AM - 10:00 AM : Room 120A
Laritza Saenz