ENPM 809W Introduction to Secure Software Engineering

Gananand Kini

Lecture 14

Code Analysis

Outline

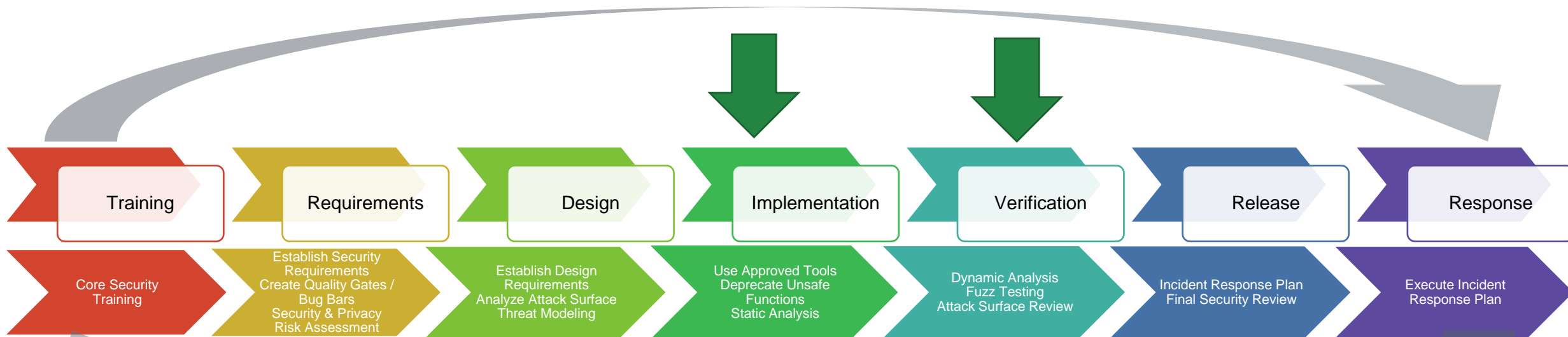- **Code Analysis**
- **Static Analysis**
- **Dynamic Analysis**
- **.NET C# Analyzers**

MITRE

Code Analysis

MITRE

Code Analysis

- **This is usually performed during both the Implementation as well as the Verification stages of the Secure Development Lifecycle.**
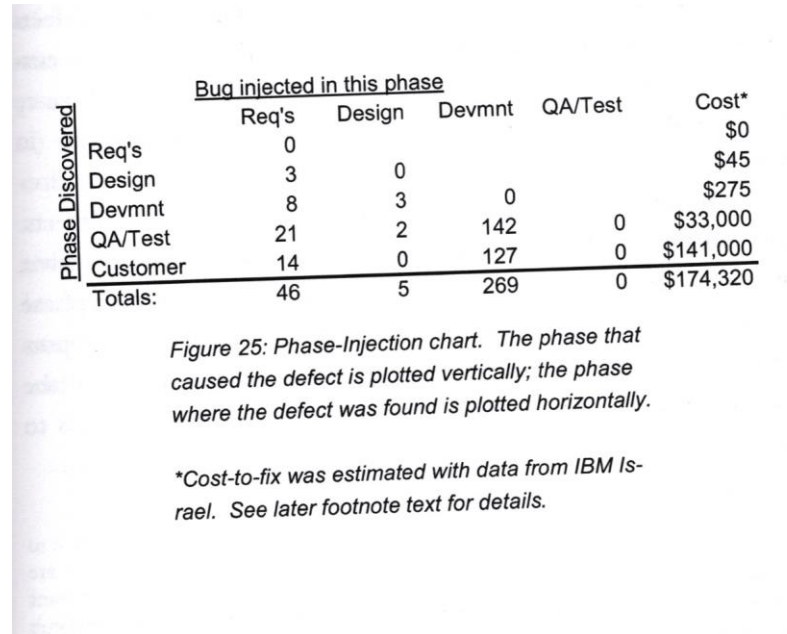

Training	Requirements	Design	Implementation	Verification	Release	Response
Core Security Training	Establish Security Requirements Create Quality Gates / Bug Bars Security & Privacy Risk Assessment	Establish Design Requirements Analyze Attack Surface Threat Modeling	Use Approved Tools Deprecate Unsafe Functions Static Analysis	Dynamic Analysis Fuzz Testing Attack Surface Review	Incident Response Plan Final Security Review	Execute Incident Response Plan

Sources:
1. https://en.wikipedia.org/wiki/Npm_(software)#Notable_breakages

MITRE

Phases Bugs injected versus Bugs detected

- **Typically, you don't want bugs to be found when the product reaches the customer.**

- **That is when it becomes the most expensive to fix.**

- **Usually lose trust in the software and the brand.**

- **Want to detect issues early!**

Phase Discovered	Bug injected in this phase				Cost*
	Req's	Design	Devmnt	QA/Test	
Req's	0				$0
Design	3	0			$45
Devmnt	8	3	0		$275
QA/Test	21	2	142	0	$33,000
Customer	14	0	127	0	$141,000
					$174,320
Totals:	46	5	269	0	

Figure 25: Phase-Injection chart. The phase that caused the defect is plotted vertically; the phase where the defect was found is plotted horizontally.

*Cost-to-fix was estimated with data from IBM Israel. See later footnote text for details.

Sources:
1. Jason A. Cohen. Best Kept Secrets of Peer Code Review: Modern Approach. Practical Advice. 2006. pp9-129.

Implementation Phase Checks

- **Usually it's cheapest to build checks during development processes that involve code check-in or integration of the code.**
    - Some may argue this slows down development, but really code that does not work or is shoddy should not exist in the source code repository in the first place.
    - A lot of code reuse can occur leading to propagation of such erroneous code or poor quality code.
- **Today this form of checking is referred to as "Secure DevOps" or "Security DevOps"**
- **Really nothing but automated mechanisms to check code before it is "committed" to a source code repository or version control system.**
- **Typically involves running automated tools that analyze the source code "statically" to ensure code is in working and high quality state before it is even submitted to the code version control system.**

MITRE

Verification Phase Checks

- **By now a lot of effort has been spent on development of a prototype or product and now the design has been "SOLIDified" somewhat …**

- **So now it technically becomes expensive to re-design or re-engineer parts or the whole of the software system if issues are discovered.**

- **Typically you run automated tools that run the compiled code and run tests etc. You typically run unit tests in the previous implementation phase.**

- **However, in this phase integration and runtime tests occur. The product is tested to ensure all requirements are being met.**

- **This includes performing any security testing and secure code review.**

- **What kinds of automated tools are used to aid with the secure code review?**

MITRE

Static Analysis Tools

- **Scan the code without running the resulting binary to find security issues.**

- **Can range from lint checking (similar to a grep search of the code base) to checking data flows based on a graph of the code (control flow, data flow etc.)**

- **Typically is instrumented as part of compilation. This allows low level access to the code.**

- **A survey paper by Emanuelsson and Nilsson from Ericsson: https://www.sciencedirect.com/science/article/pii/S1571066108003824/pdf?md5=c5f321979e147ac2d83b160921a1f2aa&pid=1-s2.0-S1571066108003824-main.pdf**

MITRE

Dynamic Analysis Tools

- **Scan the code and resources including memory for security issues as the code is run (dynamic).**

- **Typically involves memory and data flow analysis and keeping track of code as it runs.**

- **These tools can incorporate several different techniques.**

- **As an example, taint analysis could be used to check whether data provided from input is leaked to the output. It can also be used to check for memory leaks (data is propagated to a memory location, but that memory location is never freed) etc.**

- **Fuzzing is another dynamic analysis technique which runs the code using random inputs to detect crashes, exceptions etc.**

MITRE

Tool vendors

What about writing your own analyzer?

- See **https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/tutorials/how-to-write-csharp-analyzer-code-fix** for using Roslyn compiler framework to write your own C# analyzer.

- This will be the focus of the last lab.

- Roslyn is Microsoft's compiler for C# as an API.

- You are able to analyze code nodes and check for "rules" that when met indicate an issue (could be code style, security or other general programming).

- You are also able to show "potential fixes that fix the issue" to the developer.

- Is this form of analysis considered static or dynamic analysis?

MITRE

Analyzers in other languages…

- **Other languages also have similar things usually created/written by the Open-Source community**

- **Python has PyLint, Bandit**

- **Ruby has Rubocop**

- **These allow you to write custom rules to perform automated static code analysis**

MITRE

Security Code Scan (.NET)

- **https://security-code-scan.github.io/**

- **Static code analysis tool that is also available as a Visual Studio plugin or a standalone runner (integration with Git hooks etc).**

- **Can configure with custom data sources and sinks, as well as custom sanitizers and validators.**

MITRE

Microsoft.CodeAnalysis.NetAnalyzers

- PM> Install-Package Microsoft.CodeAnalysis.NetAnalyzers -Version 5.0.3
- https://github.com/dotnet/roslyn-analyzers#microsoftcodeanalysisnetanalyzers
- **This supercedes FxCop .NET analyzer since all the rules have been ported to NetAnalyzers.**
- **Enabled automatically by default in .NET5.**
- **For older .NET versions:**

<PropertyGroup>

  <EnableNETAnalyzers>true</EnableNETAnalyzers>

  <AnalysisLevel>latest</AnalysisLevel>

</PropertyGroup>

- **The documentation on rules can be found: https://docs.microsoft.com/en-us/dotnet/fundamentals/code-analysis/quality-rules/?view=vs-2019**
- **Includes design, style and security rules.**

Resources

- **Tutorial on how to write a Roslyn analyzer (includes setup instructions): https://devblogs.microsoft.com/dotnet/how-to-write-a-roslyn-analyzer/**

- **Tutorial on writing your own analyzer and fix in C#: https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/tutorials/how-to-write-csharp-analyzer-code-fix**

MITRE

Next Week

- Review.
- This week's final lab on Code Analysis due.
- Project Presentations.
- All Quizzes Due December 10th.
- Final Exam released.

MITRE