Microsoft Azure

Secure DevOps: Application Security Principles and Practices

Manual Security Verification

Your name
Your Title
Microsoft

Module Overview

- Security testing overview
- Requirements and design verification
- Code review

Microsoft Azure

Security testing overview

Functional Testing vs. Security Testing

Functional testing and security testing are not the same

Functional Testing: Verifying that an application can be used by legitimate users

Security Testing: Verifying that an application cannot be misused by malicious users

Security testing tips

Single Rule of Security Testing: there are no rules!

"Malicious users wouldn't try to do that" - Yes they will!

Malicious users will try to circumvent application client components

Malicious users will try to compromise application dependencies

Microsoft Azure

Requirements and design verification

Security requirements

Published checklist of current requirements

Covers all aspects from the organization's security and privacy policies and any regulatory requirements

Microsoft's is called *Liquid*

Security Bug Bar

Manage vulnerabilities

How do we handle security vulnerabilities found during development, test and prod?

Define a security bug bar

Defines how we handle security vulnerabilities

Includes threats, such as (in order from most severe to least severe):

- Elevation of privilege
- Denial of service
- Targeted information disclosure
- Spoofing
- Tampering

Security Bug Bars

There needs to be a consistent way to determine the risk of a security bug

Not all security bugs are created equal

A common industry standard in the CVSS, the Common Vulnerability Scoring System

Microsoft also has the MSRC BugBar (and uses CVSS!)

CVSS

A way to describe the risk of a vulnerability

Managed by Forum of Incident Response and Security Teams (FIRST)

Current version is 3.1

There’re three measurements:

1. Base Metrics for qualities intrinsic to a vulnerability
2. Temporal Metrics for characteristics that evolve over the lifetime of vulnerability
3. Environmental Metrics for vulnerabilities that depend on an implementation or environment

It can be complex

$$Exploitability = 20 \times AccessVector \times AttackComplexity \times Authentication$$

$$Impact = 10.41 \times (1 - (1 - ConfImpact) \times (1 - IntegImpact) \times (1 - AvailImpact))$$

$$f(Impact) = \begin{cases} 0, & \text{if } Impact = 0 \\ 1.176, & \text{otherwise} \end{cases}$$

$$BaseScore = roundTo1Decimal(((0.6 \times Impact) + (0.4 \times Exploitability) - 1.5) \times f(Impact))$$

CVSS – an example

5. VMware Guest to Host Escape Vulnerability (CVE-2012-1516)

CVSS v3.1 Base Score: 9.9

Metric	Value	Comments
Attack Vector	Network	VMX process is bound to the network stack and the attacker can send RPC commands remotely.
Attack Complexity	Low	The only required condition for this attack is for virtual machines to have 4GB of memory. Virtual machines that have less than 4GB of memory are not affected.
Privileges Required	Low	The attacker must have access to the guest virtual machine. This is easy in a tenant environment.
User Interaction	None	The attacker requires no user interaction to successfully exploit the vulnerability. RPC commands can be sent anytime.
Scope	Changed	The vulnerable component is a VMX process that can only be accessed from the guest virtual machine. The impacted component is the host operating system which has separate authorization authority from the guest virtual machine.
Confidentiality	High	Full compromise of the host operating system via remote code execution.
Integrity	High	Full compromise of the host operating system via remote code execution.
Availability	High	Full compromise of the host operating system via remote code execution.

CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

The MSRC BugBar

Designed to help identify the risk of a security bug
A set of objective conditions helps keep the process consistent
Derive your own based on your policies

Critical	Fix now
Important	Fix before next update
Moderate	Fix in next update
Low	Might not need fix

Code review

Why Do Manual Security Code Review?

Many automated tools already exist on the market

So why do manual code review?

Automated tools are far from perfect

- They tend to find a subset of issues
- They suffer from false positives
- In the quest to reduce false positives, they miss issues

With this said, automated tools should be complemented with manual code reviews!

Downside to manual code review

Manual code review is slow

Based on measurements at Microsoft

- a good security code reviewer can do no more than a couple thousand lines of code a day per person

There is no replacement for a manual code review conducted by knowledgeable people

There are some patterns you can follow to speed things up

Prioritization of which code to review

Assuming there is a large body of code to review, how do you prioritize which code must be reviewed first?

The most important factor is the code's attack surface

This detail can be found in your threat model(s)

- Threat Models include interfaces and trust boundaries
- High attack surface code should be reviewed earliest and most in-depth
- Anonymously and remotely accessible code is the most vulnerable

Security code review golden rule

Most security vulnerabilities are caused by untrusted data

"All input is evil until proven otherwise"

"A Secure System is a system that does what it's supposed to do and nothing else"

In the face of untrusted input a system can start to do things not intended (eg; SQL injection)

Lab – Pull Request for Manual Security Review