



# Hands On with DevSecOps Tools Workshop

October 27, 2020

*Presented by ASMGi*



# Today's Presenter

**Alex Frankel**

*Software Developer, ASMGi*

*afrankel@asmgi.com*



**Timothy Thompson**

*Senior Security Specialist*

*tthompson@asmgi.com*



# Agenda

- ◆ Static Application Security Testing (SAST)
  - ◆ Software Composition Analysis (SCA)
  - ◆ Dynamic Application Security Testing (DAST)
- **What** (are these Tools)
  - **Why** (Reasons to Use)
  - **How** (to Use these Tools)
  - **Output** (Interpret the Results)
  - **Action** (Remediation Steps)
  - **Moving Forward** (Keys for Success)

# STATIC APPLICATION SECURITY TESTING (SAST)

# What is Static Application Security Testing (SAST)?

**Static Application Security Testing (SAST)**, or **static** analysis, is a testing methodology that analyzes source code to find security vulnerabilities that make your organization's applications susceptible to attack.

**SAST scans an application before the code is compiled.**



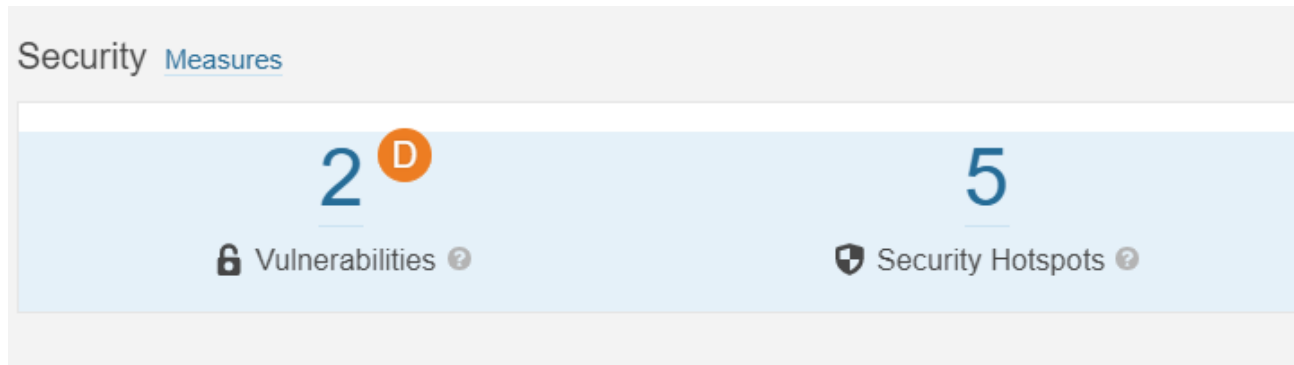
```
//Load values from database store with channel select
NotificationClient NotificationClient = null; // = NotificationClient.Get
if (NotificationClient == null)
{
    NotificationClient = new bl.desktop.NotificationClient() { Deny = false };
    //NotificationClient.Insert();
}
else
{
    NotificationClient.LastRequest = DateTime.Now;
    NotificationClient.RequestCount = NotificationClient.RequestCount + 1;
    //NotificationClient.Update();
}
if (NotificationClient.Deny == false)
{
    NotificationRequest NotificationRequest = new bl.desktop.NotificationRequest();
    NotificationRequest.ClientId = NotificationClient.ClientId;
    NotificationRequest.CurrentRequest.UserHostAddress = NotificationClient.CurrentRequest.UserHostAddress;
    NotificationRequest.LocationCount = NotificationClient.CurrentRequest.LocationCount;
    NotificationRequest.Timestamp = DateTime.Now;
    //Redirect message
    for (int i = 0; i <= NotificationClient.CurrentRequest.LocationCount; i++)
```

# Why use SAST?

- ◆ Scans all Code *even* Aspects that are not even used
- ◆ Static Code Scans do not require Fully Completed Code
- ◆ Create Coding Standards that are *enforced* across a Development Team
- ◆ Identification of Issues *early* in the Software Development Lifecycle
- ◆ Ultimately to save Time, Effort, and Money!



# Output – How to Interpret the SAST Results



**Filters** Clear All Filters

▼ **Type** **VULNERABILITY** Clear

- Bug 28
- Vulnerability 2
- Code Smell 787
- Security Hotspot 5

Ctrl + click to add to selection

▼ **Severity**

Blocker	0	Minor	0
Critical	2	Info	0
Major	0		

☐ Bulk Change

↑ ↓ to select issues ← → to navigate ↺ 1 / 2 issues 0 effort

Core/Application/Impl/CryptoHelper.cs

☐

**Use a strong cipher algorithm.** [Why is this an issue?](#)

9 months ago ▼ L19

Vulnerability ▼ Critical ▼ Open ▼ Not assigned ▼ [Comment](#)

No tags ▼

☐

**Use a strong cipher algorithm.** [Why is this an issue?](#)

9 months ago ▼ L32

Vulnerability ▼ Critical ▼ Open ▼ Not assigned ▼ [Comment](#)

No tags ▼

2 of 2 shown



# Output - How To Interpret SAST Results (continued)



**Vulnerabilities**

**Hot Spots**



- **BLOCKER**
  - Bug with a high probability to impact the behavior of the application in production: memory leak, unclosed JDBC connection, .... The code **MUST** be immediately fixed.
- **CRITICAL**
  - Either a bug with a low probability to impact the behavior of the application in production or an issue which represents a security flaw: empty catch block, SQL injection, ... The code **MUST** be immediately reviewed.
- **MAJOR**
  - Quality flaw which can highly impact the developer productivity: uncovered piece of code, duplicated blocks, unused parameters...
- **MINOR**
  - Quality flaw which can slightly impact the developer productivity: lines should not be too long, "switch" statements should have at least 3 cases, ...
- **Hot Spot**
  - A Security Hotspot highlights a security-sensitive piece of code that the developer needs to review. Upon review, you'll either find there is no threat, or you need to apply a fix to secure the code.




# Action - How To Remediate Vulnerabilities


- ◆ Many of the Tools available today provide a drill down to find exact problems and will outline items such as:
  - **What** is the issue
  - **Where** the issue can be found in the application
  - **Why** this item creates a security issue
  - **When** the issue was first detected
  - **How** - an estimation of effort to remediate the security issue

Web.UI/ClientApp/app/admin/header/header.component.html

☐ [issue?](#)

Add rel="noopener noreferrer" to this link to prevent the original page from being modified by the opened link. [Why is this an](#) 2 years ago ▼ L140  

 Vulnerability ▼  Blocker ▼  Open ▼ Not assigned ▼ 1min effort [Comment](#)

 No tags ▼

# Action - Dashboard Drilldown – Where To Find The Issue

```
12      {
13          _key = settings.DbColumnEncryptionSecret;
14      }
15
16      public static string Encrypt(string input)
17      {
18          byte[] inputArray = UTF8Encoding.UTF8.GetBytes(input);
19          TripleDESCryptoServiceProvider tripleDES = new TripleDESCryptoServiceProvider();
20
21          tripleDES.Key = UTF8Encoding.UTF8.GetBytes(_key);
22          tripleDES.Mode = CipherMode.ECB;
23          tripleDES.Padding = PaddingMode.PKCS7;
24          ICryptoTransform cTransform = tripleDES.CreateEncryptor();
25          byte[] resultArray = cTransform.TransformFinalBlock(inputArray, 0, inputArray.Length);
26          tripleDES.Clear();
27          return Convert.ToBase64String(resultArray, 0, resultArray.Length);
28      }
```

Use a strong cipher algorithm. [Why is this an issue?](#)

9 months ago ▾ L19 🔗

🔒 Vulnerability ▾ ⬆️ Critical ▾ ○ Open ▾ Not assigned ▾ Comment

🏷️ No tags ▾

# Action - Dashboard Drilldown – How To Correct the Issue

💡 Cipher algorithms should be robust

## Cipher algorithms should be robust

🔒 Vulnerability   🔴 Critical   © Main sources   🔍 cwe, owasp-a3, owasp-a6, privacy, sa...   Available Since Dec 17, 2019   SonarAnalyzer (C#)

[Strong cipher algorithms](#) are cryptographic systems resistant to cryptanalysis, they are not vulnerable to well-known attacks like brute force attacks for example.

It is recommended to use only cipher algorithms intensively tested and promoted by the cryptographic community.

### Noncompliant Code Example

```
var tripleDES1 = new TripleDESCryptoServiceProvider(); // Noncompliant: Triple DES is vulnerable to meet-in-the-middle attack

var simpleDES = new DESCryptoServiceProvider(); // Noncompliant: DES works with 56-bit keys allow attacks via exhaustive search

var RC2 = new RC2CryptoServiceProvider(); // Noncompliant: RC2 is vulnerable to a related-key attack
```

### Compliant Solution

```
var AES = new AesCryptoServiceProvider(); // Compliant
```

### See

- [OWASP Top 10 2017 Category A3](#) - Sensitive Data Exposure
- [MITRE, CWE-327](#) - Use of a Broken or Risky Cryptographic Algorithm
- [CERT, MSC61-J](#) - Do not use insecure or weak cryptographic algorithms
- [SANS Top 25](#) - Porous Defenses

# Moving Forward – Keys for Success

## ◆ Potential Issue: Language Support

### – Mitigating Step

- *Conduct appropriate research when selecting a Tool to confirm it supports the Languages used by your Solutions*

## ◆ Potential Issue: Time required to run a Scan

### – Mitigating Step

- *Scheduling Scans*
- *Conducting Scans on Code Check Ins*

## ◆ Potential Issue: Results can include many False Positives

### – Mitigating Steps

- *Rules can be tuned to help alleviate this Issue*



# **SOFTWARE COMPOSITION ANALYSIS**

## **(SCA)**

# What is Software Composition Analysis (SCA)?

- ◆ **Software Composition Analysis (SCA)** is an open source component management tool that:
  - Generates a report, listing all open source components in the application including direct and indirect dependencies
  - The tool can also detect software licenses, deprecated dependencies, as well as vulnerabilities and potential exploits

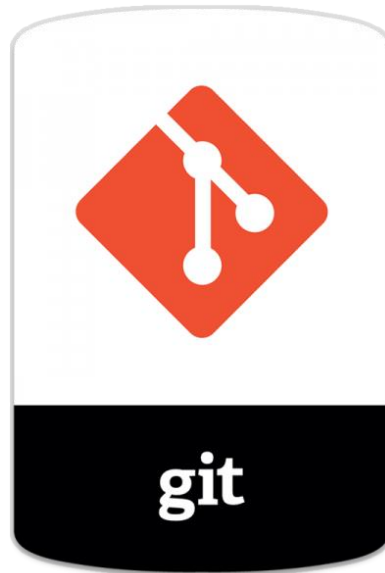


# Why use SCA?

- ◆ **SCA** was born out of a cross-industry rise in Open Source usage which made it increasingly hard for companies to track Open Source components manually using spreadsheets, emails and ticketing systems
- ◆ As Open Source usage grew in software creation, it became a necessity to automate the Open Source Management Process
- ◆ Protect applications by identifying:
  - Vulnerabilities in the Open Source components
  - Details on Current and Expired Licenses
  - Out-of-date Library Versions and Age



# How to Conduct a SCA Scan?



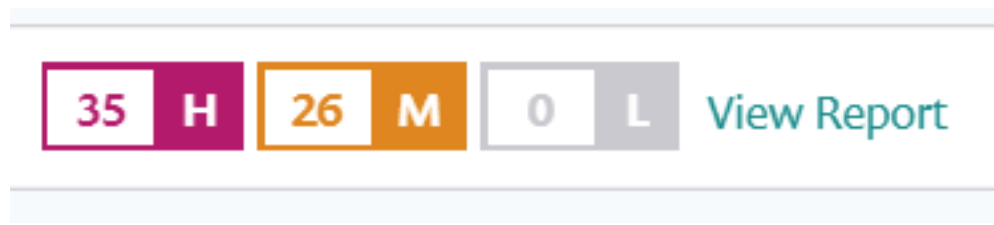
# Output - How to Interpret the SCA Results

## ◆ Results and Ranking

- High, Medium, and Low Severity Issues

## ◆ Information



- Most SCA Tools will link the issue to a detail page that includes:
  - **What** Version of the Open Source Code introduced the Security Issue
  - **Why** that Version is a Security Exploit and how the Attack is done
  - **How** - what Version is needed to correct the Issue



SCORE	ISSUE	IDENTIFIERS	PROJECT	EXPLOIT MATURITY	FIXABLE	INTRODUCED	STATUS	REACHABILITY
H 756	Cross-site Scripting (XSS) 7.2.1	CWE-79	Bitbucket Cloud	Proof Of Concept	Upgrade	14 Oct, 2020	Open	No Info

# Action - Drilling Into An Issue

## Cross-site Scripting (XSS)

Vulnerable module:   
Introduced through: @7.2.1  
Exploit maturity: Proof of concept  
Fixed in: 7.2.2, 8.1.1

 [Fix this vulnerability](#)

### Detailed paths and remediation

- Introduced through: ClientApp@0.0.0 > @7.2.1  
Remediation: Upgrade to @7.2.2

### Overview

 a JavaScript charting library based on SVG, with fallbacks to VML and canvas for old browsers.

Affected versions of this package are vulnerable to Cross-site Scripting (XSS). The `<a>` tag for text formats is translated into a `tspan` with `onclick`, allowing for script injection.

# Moving Forward – Keys for Success - SCA

## ◆ Potential Issue: Impact on Workflows and Routines

### – Mitigating Step

- *Run Scans on Check In*
- *Schedule Reports to Automatically Run*

## ◆ Potential Issue: Impact on Team Throughput

### – Mitigating Step

- *Establish Processes to Review Reports and Remediate Issues as “real time” as possible*

## ◆ Potential Issue: On Premise Versions Out of Date

### – Mitigating Steps

- *Ensure On Premise Versions of Solutions are kept up to date to ensure solution is working effectively*



# **DYNAMIC APPLICATION SECURITY TESTING (DAST)**

# What is Dynamic Application Security Testing?

- Dynamic Application Security Testing (**DAST**) is a black-box security testing methodology in which an application is tested from the outside
- Types of DAST
  - Authenticated
    - Uses known User Information
    - Able to Drill down in Application based on User Rights
  - Unauthenticated
    - Can Examine only Publicly Visible Information

# Why use DAST?

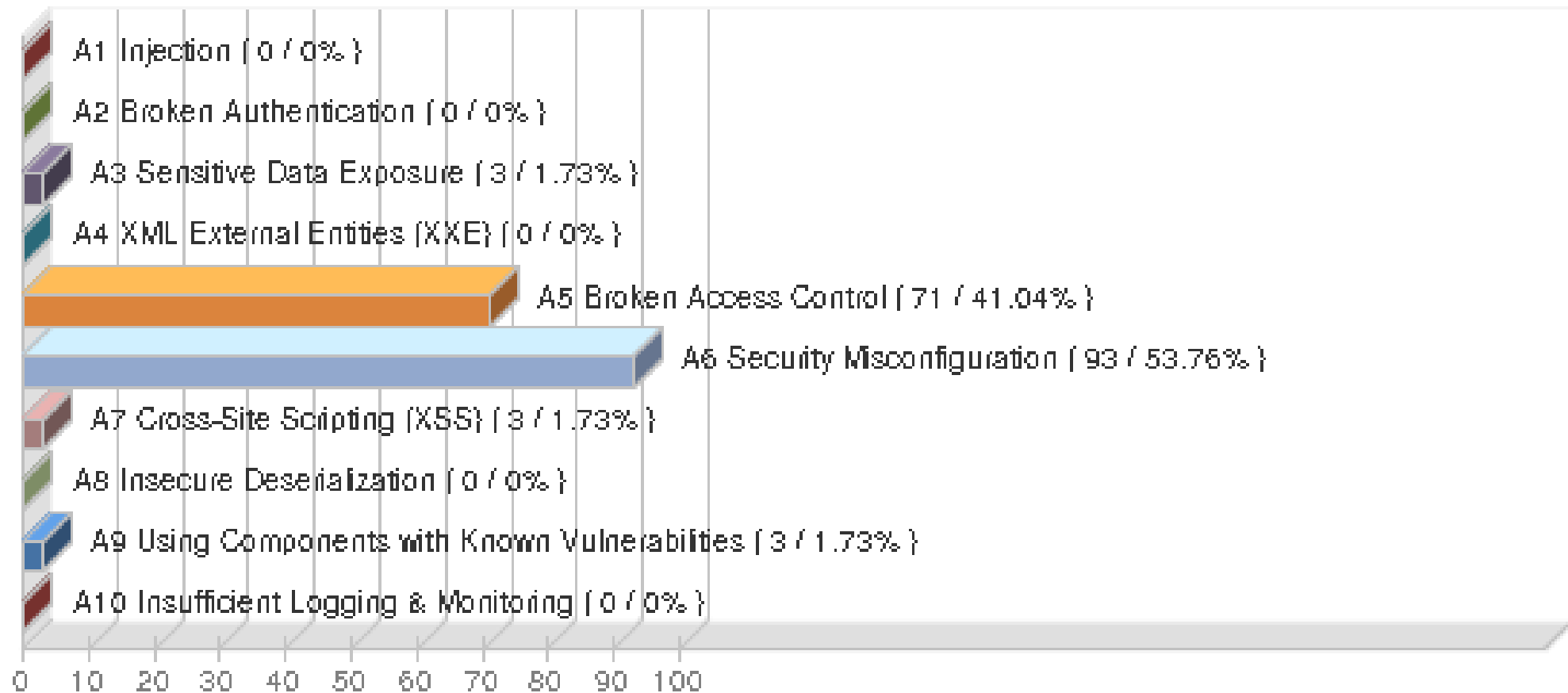
- ◆ Extremely good at finding Externally Visible Issues and Vulnerabilities
- ◆ Ability to identify Runtime Problems
- ◆ Excellent in finding Server Configuration and Authentication Issues
- ◆ Examines an Application when it is running and tries to Hack it just like an Attacker

# How to Conduct a DAST?

- ◆ Input Application URL (or IP Address)
- ◆ Configure Scanner for Solution Scan Requirements
  - Pre-Defined Security Profile
    - *OWASP Top 10*
    - *SANS Top 20*
  - Customer Security Profile
    - *Including Requirements such as Compliance Requirements (e.g. PCI, HIPAA) or Specific Vulnerabilities*
  - Authentication Requirements
    - *Authenticated or Non-Authenticated*
- ◆ Initiate Scan Process

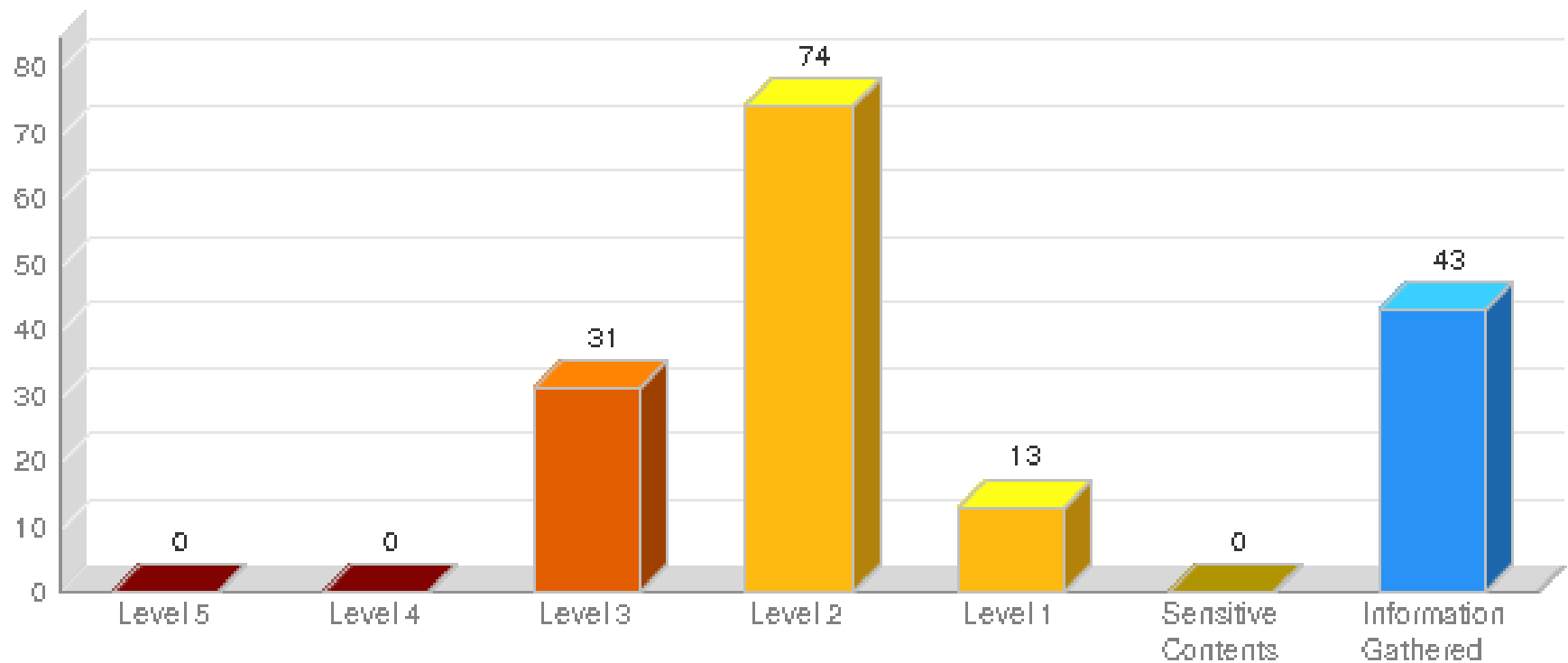


# Output - How to Interpret the DAST Results



**Example Output**  
(Based on OWASP Top 10)

# Output - How to Interpret the DAST Results (continued)



**Example Output**  
Based on Severity Level Requirements and Output

# How to Remediate Vulnerabilities

## ◆ Understanding the Vulnerabilities

- Many of the tools available today provide a drill down to find exact problems and will outline items such as:
  - *Threat*
  - *Impact*
  - *Solution*
  - *Detection Information*

## ◆ Remediation

- Once the Vulnerabilities have been “understood”, Remediation can be Completed

## ◆ Validation

- Iterative approach is required to ensure all Vulnerabilities have been Remediated

# Moving Forward – Keys for Success - DAST

## ◆ Potential Issue: No Code Visibility

### – Mitigating Step

- *Compliment DAST with a SAST and SCA Solution*

## ◆ Potential Issue: Time required to run a Scan

### – Mitigating Step

- *Scheduling and Incorporating DAST into Development Workflows and Processes*

## ◆ Potential Issue: Late in the Development Process

### – Mitigating Step

- *Incorporating DAST “early” in the Development Process (e.g. “Move to the Left”)*



# Moving Forward – Keys for Success – DAST (continued)

## ◆ Potential Issue: Security Profile Mis-Alignment

### – Mitigating Step

- *Understand Business Requirements to ensure all potential Vulnerabilities are Identified*

## ◆ Potential Issue: Missing Vulnerabilities by Un-Authenticated Scans

### – Mitigating Steps

- *Incorporate both Authenticated Scans when possible*



# Bonus - Example Tools / Solution Providers

## ◆ SAST

- SonarQube
- Sonar Cloud
- Veracode
- Coverity

## ◆ SCA

- White Source
- Snyk
- Veracode
- Black Duck
- Fossa

## ◆ DAST

- Qualys
- Rapid7
- Veracode
- Tenable
- Netsparker





# Thank You!

800 Superior Ave E, Ste 1050  
Cleveland, OH 44114

Phone: 216.255.3040  
Fax: 216.274.9647

Email: [info@asmgi.com](mailto:info@asmgi.com)

[www.asmgil.com](http://www.asmgil.com)