Security in knowledge

The Top Ten Things I Wish Static Analysis Tools Commonly Did

Randall Brooks, CISSP, CSSLP

Raytheon

RSA CONFERENCE 2013

Session ID: ASEC-R35A

Session Classification: Intermediate

Agenda

- Introduction
- The List
- Key Takeaways
- How to Apply
- Q&A

Introduction

- This work was based on internal research performing Static Application Security Testing (SAST)
  - The term Static Analysis Tools is to cover those tools used for Static Code Analysis
- No specific tools will be referenced within this presentation and many tools support most of this list
- *Disclaimer:* The views of its author does not reflect the view of his employer

To achieve software assurance (SwA), one must employ static analysis to find Common Weakness Enumerations (CWEs), but the use of these tools have proven difficult. This session will cover the top ten features that I wish static analysis tools commonly would do.

The List

- 10. Scare me by default
  - Please don't hide the truth so that later I find out my code is full of weaknesses
- 9. Adjust to defined thresholds
  - My SANS/CWE Top 25
  - Defense Information Systems Agency (DISA) Application Security and Development (ASD) Security Technical Implementation Guide (STIG)
  - Others?

The List

- 8. Give me CWEs by default
  - It's the standard
- 7. Support DISA's ASD STIG or another customer-based requirement set
  - Have built-in settings for this (e.g., this weakness is a CAT I)

Brief Listing of the Top 25

This is a brief listing of the Top 25 items, using the general ranking.

NOTE: 16 other weaknesses were considered for inclusion in the Top 25, but their general scores were not high enough. They are listed in a separate "On the Cusp" page.

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
[17]	65.5	CWE-732	Incorrect Permission Assignment for Critical Resource
[18]	64.6	CWE-676	Use of Potentially Dangerous Function
[19]	64.1	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[20]	62.4	CWE-131	Incorrect Calculation of Buffer Size
[21]	61.5	CWE-307	Improper Restriction of Excessive Authentication Attempts
[22]	61.1	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
[23]	61.0	CWE-134	Uncontrolled Format String
[24]	60.3	CWE-190	Integer Overflow or Wraparound
[25]	59.9	CWE-759	Use of a One-Way Hash without a Salt

CWE-89 - SQL injection - delivers the knockout punch of security weaknesses in 2011. For data-rich software applications, SQL injection is the means to steal the keys to the kingdom. CWE-78, OS command injection, is where the application interacts with the operating system. The classic buffer overflow (CWE-120) comes in third, still pernicious after all these decades. Cross-site scripting (CWE-79) is the bane of web applications everywhere. Rounding out the top 5 is Missing Authentication (CWE-306) for critical functionality.

Source: http://cwe.mitre.org

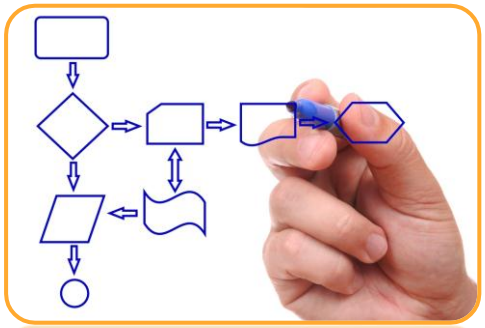
The List

- 6. Annotate if the issue is one of the SANS/CWE Top 25

- 5. Give me the same repeatable results in an IDE or standalone solution
  - Some are plugins
  - Some work directly with a compiler

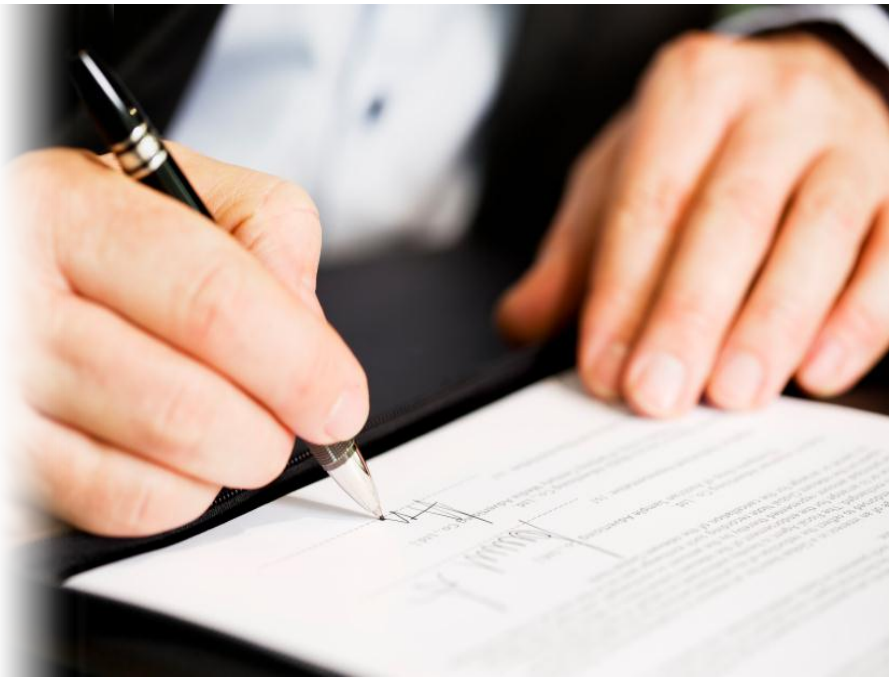
The List

- 4. Support architecture analysis
  - Unified Modeling Language
- 3. Support static binary analysis
  - Binary Disassembly linked to source code



The List

- 2. Have a license structure that supports third party assessment
  - Site License
  - Project License
  - Key Server



And my number 1 is…….

Security in knowledge

The List

- 1. Have a common XML-based reporting structure
  - A standard schema
  - Include CWEs
  - Fast import and export
  - It's ok to add your extensions


```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" ... attributeFormDefault="unqualified">
- <xs:element name="Compound_Element">
 - <xs:annotation>
    <xs:documentation>The Compound_Element structure represents a meaningful aggregation of several weaknesses, as in a chain like CWE-690: CWE-252 Unchecked Return Value can result in CWE-476 NULL Pointer Dereference or as in a composite like CWE-352 Cross-Site Request Forgery.</xs:documentation>
   </xs:annotation>
 - <xs:complexType>
    <xs:group ref="Common_Attributes" />
   - <xs:attribute name="ID" type="xs:integer" use="required">
     - <xs:annotation>
        <xs:documentation>This attribute provides a unique identifier for the entry. It will be static for the lifetime of the entry. In the event that this entry becomes deprecated, the ID will not be reused and a pointer will be left in this entry to the replacement. This is required for all Compound_Elements.</xs:documentation>
       </xs:annotation>
     </xs:attribute>
   - <xs:attribute name="Name" type="xs:string" use="required">
     - <xs:annotation>
        <xs:documentation>The Name is a descriptive name used to give the reader an idea of the meaning behind the compound weakness structure. All words in the name should be capitalized except for articles and prepositions unless they begin or end the name. Subsequent words in a hyphenated chain are also not capitalized. This is required for all Compound_Elements.</xs:documentation>
       </xs:annotation>
     </xs:attribute>
```

Key Takeaways

- Please perform SAST
  - Integrate with Dynamic Application Security Testing (DAST)
- Not all Static Analysis Tools are created equal
- Conduct both internal and external third party assessments
- Work with the vendor to get a license model that works for how you do business

How to Apply

- In **1** month you should
  - Consider your work flow
  - License needs
- In **3** months you should
  - Have an open dialog with vendor of choice
  - Attend a DHS SwA Forum
- In **6** months you should
  - Implement improvements
  - Try to use multiple tools for greater code coverage

Questions and Answers

Randall Brooks, CISSP, CSSLP
brooks@raytheon.com

http://rtncyberjobs.com

Personal Data

Randall Brooks, a thirteen year Raytheon employee, is an Engineering Fellow in the Trusted Networking Solutions (TNS) Cyber Defense Solutions (CDS) business area in Largo, FL. Mr. Brooks focuses on Software Assurance (SwA) and secure development life cycles (SDLC). He is a recipient of the Raytheon Excellence in Technology Meritorious and Distinguished Awards. He has been awarded three US patents on Intrusion Detection and Prevention, and two US and one UK patent(s) on Cross Domain solutions. He is also a CISSP, CSSLP, ISSEP, ISSAP and an ISSMP. He is a graduate of Purdue University with a Bachelors of Science from the School of Computer Science.

He can be reached at brooks@raytheon.com or www.linkedin.com/in/rbrooks