Engineering Your Software For Attack

Robert A. Martin
Senior Principal Engineer
Cyber Security Center
Center for National Security
The MITRE Corporation
Red Sox find safety in numbers, more memorable Game 6 scenes

2013 World Series

The 2013 World Series was the 108th edition of Major League Baseball's championship series. The best-of-seven playoff pitted the National League champion St. Louis Cardinals against the American League champion Boston Red Sox. The Red Sox had home field advantage for the series, based on the American League's win in the All-Star Game at Citi Field in Queens, New York, on July 16. The Series started on Wednesday, October 23, ending on Game 6 which occurred the following Wednesday, October 30, 2013.

This was the fourth meeting of the Cardinals and Red Sox in the World Series (previously meeting in 1946, 1967, and 2004). It is the first World Series since 1990 to pair the two teams with the best regular-season records in their respective leagues, and only the third in history (following the 1949 and 1988 Series) to feature two teams with identical regular-season records. Because both teams share the best overall regular-season records in baseball, this will be only the fourth time since the introduction of the Division Series (1995) in which the
What We’ve Learned

Making systems secure by just reducing attack surface really hard – maybe impossible

- Software Systems & Networks too large and complex

- Zero vulnerabilities for all assets on network?
  - Assumes you know all assets
  - Assumes you can know all vulnerabilities

Cyber Attack Lifecycle
Characteristics of the Advanced Persistent Threat

1. We won’t always see the initial attack
2. We can’t keep the adversary out
3. Advanced Persistent Threat is not a “hacker”
### Cyber Threat Intelligence Sharing Building Blocks – Phases of a Cyber Attack Lifecycle

<table>
<thead>
<tr>
<th>Recon</th>
<th>Deliver</th>
<th>Control</th>
<th>Maintain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weaponize</td>
<td>Exploit</td>
<td>Execute</td>
<td></td>
</tr>
</tbody>
</table>

#### Proactive Detection Mitigation
- **Recon**: Obtain information to conduct the attack
- **Weaponize**: Place payload into delivery vehicle
- **Deliver**: Send the attack to the potential victim
- **Exploit**: The point of no return
- **Control**: Direct the victim system to take actions
- **Execute**: Fulfill mission requirements
- **Maintain**: Insure future access

#### Incident Response & Mission Assurance
Hockey Goalie
Elements of an Attacker Aware Cyber Threat Intelligence Sharing-Based Approach

1. Understanding of the Attackers Building Blocks

2. Effective Cyber Threat Intelligence Sharing Model

3. Agile defensive posture aligned with threat from the attackers and attack techniques

4. Development team working side-by-side with operators (DevOps)
Extending the Threat-Driven Perspective Beyond Operational Defense

Risk-Based, Attack-Aware, and Threat-Driven

Operational

Strategic
From Just a Mitigation Approach

A traditional information assurance approach based solely on regulation, which resulted in an approach based on mitigation and compliance around static defenses.

To a threat/attacker based cyber defense that understands attacks and balances Mitigation with Detection and Response

- Defenders become demanding consumers of intelligence, informed by understanding of the attacks their software systems face.
- Producers of intelligence.
What is “Cyber Threat Intelligence?”

Consider these questions:

- What activity/attacks are we seeing?
- What attacks should I look for on my networks and systems and why?
- Where has this attack been seen?
- What does it do?
- What weaknesses does this attack exploit?
- Why does attacker do this?
- Who is responsible for this attack?
- What can I do about it?
Structured Threat Information eXpression (STIX) v1.0 Architecture
Structured Threat Information eXpression (STIX) v1.0 Architecture
Structured Threat Information eXpression (STIX) v1.0 Architecture
Structured Threat Information eXpression (STIX) v1.0 Architecture
Structured Threat Information eXpression (STIX) v1.0 Architecture
Structured Threat Information eXpression (STIX) v1.0 Architecture
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC-01</td>
<td>Insufficient Authentication</td>
<td>287</td>
<td></td>
<td>642</td>
<td>A3 – Broken Authentication and Session Management</td>
<td>A7 – Broken Authentication and Session Management</td>
<td>A3 – Broken Authentication and Session Management</td>
</tr>
<tr>
<td>WASC-03</td>
<td>Integer Overflows</td>
<td>190</td>
<td>128</td>
<td>682</td>
<td>A4 – Insecure Direct Object Reference</td>
<td>A7 – Insecure Direct Object Reference</td>
<td></td>
</tr>
<tr>
<td>WASC-04</td>
<td>Insufficient Transport Layer Protection</td>
<td>311 523</td>
<td>319</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-05</td>
<td>Remote File Inclusion</td>
<td>98</td>
<td>193 253</td>
<td>426</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-06</td>
<td>Format String</td>
<td>134</td>
<td>67</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-07</td>
<td>Buffer Overflow</td>
<td>119 120</td>
<td>10 100</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-08</td>
<td>Cross-site Scripting</td>
<td>79</td>
<td>18 19 63</td>
<td>79</td>
<td>A2 – Cross-Site Scripting</td>
<td>A4 – Cross-Site Request Forgery (CSRF)</td>
<td>A3 – Cross Site Scripting (XSS)</td>
</tr>
<tr>
<td>WASC-09</td>
<td>Cross-site Request Forgery</td>
<td>352</td>
<td>62</td>
<td>352</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-10</td>
<td>Denial of Service</td>
<td>400</td>
<td>119</td>
<td>404</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Software Assurance.—The term “software assurance” means the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software, throughout the life cycle. Section 933
Software Assurance Methods

Development Process
Apply assurance activities to the procedures and structure imposed on software development

Operational System
Implement countermeasures to the design and acquisition of end-item software products and their interfaces

Development Environment
Apply assurance activities to the environment and tools for developing, testing, and integrating software code and interfaces

Additional Guidance in PPP Outline and Guidance
13.7.3. Software Assurance

13.7.3.1. Development Process

13.7.3.1.1 Static Analysis

13.7.3.1.2 Design Inspection

13.7.3.1.3 Code Inspection

13.7.3.1.4. Common Vulnerabilities and Exposures (CVE)

13.7.3.1.5. Common Attack Pattern Enumeration and Classification (CAPEC)

13.7.3.1.6. Common Weakness Enumeration information (CWE)

13.7.3.1.7. Penetration Test

13.7.3.1.8 Test Coverage

13.7.3.2. Operational System

13.7.3.2.1. Failover Multiple Supplier Redundancy

13.7.3.2.2. Fault Isolation

13.7.3.2.3. Least Privilege

13.7.3.2.4. System Element Isolation

13.7.3.2.5. Input Checking/Validation

13.7.3.2.6. Software Encryption and Anti-Tamper Techniques (SW load key)

13.7.3.3. Development Environment

13.7.3.3.1 Source Code Availability

13.7.3.3.2 Release Testing

13.7.3.3.3 Generated Code Inspection

13.7.3.3.3. Additional Countermeasures
4. VULNERABILITY AND WEAKNESS MANAGEMENT

Purpose and Use

- Unpatched vulnerabilities are a major concern for all organizations.
- A key goal of vulnerability management is to prevent vulnerabilities from being exploited. This is accomplished by identifying and fixing vulnerabilities prior to placing them into production. The goal is to ensure that vulnerabilities are not present in critical software or hardware assets. Preventing vulnerabilities from being exploited is essential to maintaining the security of the organization's assets.

Table 8 – Responses to Question 4.3

<table>
<thead>
<tr>
<th>Impact Level</th>
<th>For systems in development and/or maintenance:</th>
<th>For systems in production:</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Use methods described in Table 9 to identify and fix instances of common weaknesses, prior to placing that version of the code into production.</td>
<td>Report on configuration and vulnerability levels for hardware assets supporting those systems, giving application owners an assessment of risk inherited from the general support system (network).</td>
</tr>
<tr>
<td>Moderate</td>
<td>Can the organization find SCAP compliant tools and good SCAP content?</td>
<td>Can the organization find SCAP compliant tools and good SCAP content?</td>
</tr>
<tr>
<td>Low</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9 – Methods to Identify and Fix Instances of Common Weaknesses

<table>
<thead>
<tr>
<th>Identify Universe Enumeration</th>
<th>Find Instances Tools and Languages</th>
<th>Assess Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Vulnerabilities</td>
<td>Static Code Analysis tools</td>
<td>Common Weakness Scoring System (CWSS)</td>
</tr>
<tr>
<td>Common Attack Pattern</td>
<td>Manual code reviews (especially for weaknesses not covered by the automated tools)</td>
<td></td>
</tr>
<tr>
<td>Enumeration and Classification (CAPEC)</td>
<td>Dynamic Code Analysis tools</td>
<td></td>
</tr>
<tr>
<td>Web scanners for web-based applications</td>
<td>Web scanners for web-based applications</td>
<td></td>
</tr>
<tr>
<td>PEN testing for attack types not covered by the automated tools.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See guidance that describes the purpose and use of these tools and how they can be used today in a practical way to improve security of software during development and maintenance.
SQL Injection Attack Execution Flow

User

Web Form with ' in all fields

1. Web Form with ' in ITEM_CATEGORY field

4. SQL error message

5. Web Form with: 'exec master..xp_cmdshell 'dir' --

6. a listing of all directories

SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY='$user_input'
ORDER BY PRICE
Simple test case for SQL Injection

Test Case 1: Single quote SQL injection of registration page web form fields

Test Case Goal: Ensure SQL syntax single quote character entered in registration page web form fields does not cause abnormal SQL behavior

Context:
• This test case is part of a broader SQL injection syntax exploration suite of tests to probe various potential injection points for susceptibility to SQL injection. If this test case fails, it should be followed-up with test cases from the SQL injection experimentation test suite.

Preconditions:
• Access to system registration page exists
• Registration page web form field content are used by system in SQL queries of the system database upon page submission
• User has the ability to enter free-form text into registration page web form fields

Test Data:
• ASCII single quote character

Action Steps:
• Enter single quote character into each web form field on the registration page
• Submit the contents of the registration page

Postconditions:
• Test case fails if SQL error is thrown
• Test case passes if page submission succeeds without any SQL errors
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Weakness ID: 89 (Weakness Base)

Description

Description Summary
The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.

Extended Description
Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands.

SQL injection has become a common issue with database-driven web sites. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages
All

Technology Classes
Database-Server
CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Description

The software constructs an OS command, but it doesn’t properly neutralize or escape special elements used in the command when it is sent to the OS.

Extended Description

This could allow attackers to execute an arbitrary OS command.

There are at least two subvulnerabilities:

1. The application permits the execution of an externally-supplied system command, such as 'nslookup', as an argument.

2. The application permits the execution of arbitrary commands to users, such as the command 'whoami', which could be used by the attacker to determine the user. If the command is permitted to user-level processes, it might then be possible to use the victim’s credentials to perform actions as the victim.

Common Consequences

- Technical Impact: Execute unauthorized code or commands; DoS: crash / exit / restart; Read files or directories; Modify files or directories; Read application data; Modify application data; Hide activities

Likelihood of Exploit

High

Detection Methods

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives.

Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes.

Automated static analysis might not be able to detect the usage of custom API functions or third-party libraries that indirectly invoke OS commands, leading to false negatives - especially if the API/library code is not available for analysis.

This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software’s operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Effectiveness: Moderate

Manual Static Analysis

Since this weakness does not typically appear frequently within a single software package, manual white box techniques may be able to provide sufficient code coverage and reduce the number of false positives if all potentially-vulnerable operations can be assessed within limited time constraints.

Effectiveness: High
Technical Impacts – Common Weakness Risk Analysis Framework (CWRAF)

1. Modify data
2. Read data
3. DoS: unreliable execution
4. DoS: resource consumption
5. Execute unauthorized code or commands
6. Gain privileges / assume identity
7. Bypass protection mechanism
8. Hide activities
Known Threat Actors

Attack

Weakness

Weakness

Weakness

Weakness

Attack

Attack

Attack

Weakness

Weakness

Weakness

Weakness

Controls*

Technical Impacts

Operational Impacts

* Controls include architecture choices, design choices, added security functions, activities & processes, physical decomposition choices, code assessments, design reviews, dynamic testing, and pen testing

© 2013 The MITRE Corporation. All rights reserved.
Which static analysis tools and Pen Testing services find the CWE’s I care about?
Leveraging and Managing to take Advantage of the Multiple Perspectives of Analysis

Total Potential Security Weaknesses

- Null Pointer Dereference
- Threading Issues
- Issues in Dead Code
- Insecure Crypto Functions
- ...

- Environment Configuration Issues
- Issues in integrations of modules
- Runtime Privileges Issues
- Protocol Parser/Serializer Issues
- Issues in 3rd party components
- ...

- Application Logic Issues

- SQL Injection
- Cross Site Scripting
- HTTP Response Splitting
- OS Commanding
- LDAP Injection
- ...

© 2013 The MITRE Corporation. All rights reserved.
Leveraging and Managing to take Advantage of the Multiple Perspectives of Analysis

- Different perspectives are effective at finding different types of weaknesses
- Some are good at finding the cause and some at finding the effect

<table>
<thead>
<tr>
<th></th>
<th>Static Code Analysis</th>
<th>Penetration Test</th>
<th>Data Security Analysis</th>
<th>Code Review</th>
<th>Architecture Risk Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-Site Scripting (XSS)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL Injection</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insufficient Authorization Controls</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Broken Authentication and Session Management</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Information Leakage</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Improper Error Handling</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insecure Use of Cryptography</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Cross Site Request Forgery (CSRF)</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Denial of Service</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Poor Coding Practices</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notional</td>
<td>Architecture Analysis</td>
<td>Design Review</td>
<td>Source Code Static Analysis</td>
<td>Binary Static Analysis</td>
<td>Automated Dynamic Analysis</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------</td>
<td>---------------</td>
<td>-----------------------------</td>
<td>------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>(2) Read Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3) DoS: unreliable execution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4) DoS: resource consumption</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5) Execute unauthorized code or commands</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(6) Gain privileges / assume identity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(7) Bypass protection mechanism</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(8) Hide activities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>Phase</td>
<td>CWEs</td>
<td>CWEs</td>
<td>CWEs</td>
<td>CWEs</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>Modify data</td>
<td>CWE-23</td>
<td>CWE-23</td>
<td>CWE-131</td>
<td>CWE-131</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Relative Path Traversal</td>
<td></td>
<td>Incorrect Calculation of Buffer Size</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Read Data</td>
<td>CWE-14</td>
<td>CWE-14</td>
<td>CWE-129</td>
<td>CWE-129</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compiler Removal of Buffer Clearing</td>
<td></td>
<td>Improper Validation of Array Index</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>DoS: unreliable execution</td>
<td>CWE-36</td>
<td>CWE-36</td>
<td>CWE-476</td>
<td>CWE-476</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Path Traversal</td>
<td></td>
<td>Null Pointer Dereference</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>DoS: resource consumption</td>
<td>CWE-395</td>
<td>CWE-395</td>
<td>CWE-190</td>
<td>CWE-190</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use of NullPointerException</td>
<td></td>
<td>Integer Overflow</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Execute unauthorized code or commands</td>
<td>CWE-88</td>
<td>CWE-88</td>
<td>CWE-120</td>
<td>CWE-120</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Argument Injection</td>
<td></td>
<td>Buffer Overflow</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Gain privileges / assume identity</td>
<td>CWE-96</td>
<td>CWE-96</td>
<td>CWE-489</td>
<td>CWE-489</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Static Code Injection</td>
<td></td>
<td>Leftover Debug Code</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Bypass protection mechanism</td>
<td>CWE-89</td>
<td>CWE-89</td>
<td>CWE-357</td>
<td>CWE-357</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL Injection</td>
<td></td>
<td>Insufficient UI Warning of Dangerous</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Hide activities</td>
<td>CWE-78</td>
<td>CWE-78</td>
<td>CWE-168</td>
<td>CWE-168</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS Command Injection</td>
<td></td>
<td>Improper Handling of Inconsistent</td>
<td></td>
</tr>
</tbody>
</table>
### Impacts by Detection Method

This table is incomplete, because many CWE entries do not have a detection method listed.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute unauthorized code or commands</td>
<td></td>
<td>78, 120, 129, 131, 476, 805</td>
<td>78, 79, 98, 120, 129, 131, 134, 190, 798, 805</td>
<td>79, 129, 134, 190, 494, 698, 798</td>
<td></td>
<td>98, 120, 131, 190, 494, 805</td>
<td>476, 798</td>
<td>78, 798</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gain privileges / assume identity</td>
<td></td>
<td>798</td>
<td></td>
<td>259, 798</td>
<td></td>
<td></td>
<td></td>
<td>798, 807</td>
<td>628</td>
<td></td>
</tr>
<tr>
<td>Modify data</td>
<td>311, 327</td>
<td>78, 89, 129, 131</td>
<td>78, 89, 129, 131, 190</td>
<td>129, 190, 319</td>
<td>89, 131, 190, 311, 327</td>
<td>209, 404, 665, 798</td>
<td>78, 798</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DoS: unreliable execution</td>
<td></td>
<td>78, 120, 129, 131, 400, 476, 665, 805</td>
<td>78, 120, 129, 131, 190, 400, 805</td>
<td>129, 190</td>
<td>400, 120, 131, 190, 805</td>
<td>476, 665</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DoS: resource consumption</td>
<td></td>
<td>120, 400, 404, 770, 805</td>
<td>120, 190, 400, 770, 805</td>
<td>190</td>
<td>400, 770</td>
<td>120, 190, 805</td>
<td>404</td>
<td>770</td>
<td>412</td>
<td></td>
</tr>
<tr>
<td>Bypass protection mechanism</td>
<td></td>
<td>89, 400, 665</td>
<td>79, 89, 190, 400, 798</td>
<td></td>
<td>14, 79, 184, 190, 733, 798</td>
<td>400</td>
<td>89, 190</td>
<td>665, 798</td>
<td>798, 807</td>
<td>14, 733</td>
</tr>
<tr>
<td>Hide activities</td>
<td></td>
<td></td>
<td></td>
<td>327</td>
<td></td>
<td></td>
<td></td>
<td>327</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td></td>
<td></td>
<td>400, 404</td>
<td>400, 798</td>
<td>198, 484, 494, 698, 733, 798</td>
<td>400</td>
<td>494</td>
<td>404, 798</td>
<td>596, 798, 807</td>
</tr>
</tbody>
</table>
Planning to Leverage “State of the Art Resource” (SOAR): Software Table of “Verification Methods”
SwA and Systems Development (example)

* Ideally Insert SwA before RFP release in Analysis of Alternatives
Cross-site Scripting (XSS) Attack (CAPEC-86)

Improper Neutralization of Input During Web Page Generation (CWE-79)

SQL Injection Attack (CAPEC-66)

Improper Neutralization of Special Elements used in an SQL Command (CWE-89)
# Software, Network Traffic, Physical, Social Engineering, and Supply Chain Attack Patterns

## CAPEC-1000: Mechanism of Attack

### Mechanism of Attack

**View ID:** 1000 (View: Graph)

**View Data**

**View Structure:** Graph

**View Objective**

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>118</td>
<td>Data Leakage Attacks</td>
<td><img src="data%E6%B3%84%E6%BC%8F%E6%94%BB%E5%87%BB" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>119</td>
<td>Resource Depletion</td>
<td><img src="%E8%B5%84%E6%BA%90%E8%80%97%E7%AB%AD" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>152</td>
<td>Injection (Injecting Control Plane content through the Data Plane)</td>
<td><img src="%E6%B3%A8%E5%85%A5%E6%8E%A7%E5%88%B6%E5%B9%B3%E9%9D%A2%E5%86%85%E5%AE%B9%E9%80%9A%E8%BF%87%E6%95%B0%E6%8D%AE%E5%B9%B3%E9%9D%A2" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>156</td>
<td>Spoofing</td>
<td><img src="%E6%AC%BA%E9%AA%97" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>172</td>
<td>Time and State Attacks</td>
<td><img src="%E6%97%B6%E9%97%B4%E5%92%8C%E7%8A%B6%E6%80%81%E6%94%BB%E5%87%BB" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>210</td>
<td>Abuse of Functionality</td>
<td><img src="%E6%BB%A5%E7%94%A8%E5%8A%9F%E8%83%BD" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>223</td>
<td>Probabilistic Techniques</td>
<td><img src="%E6%A6%82%E7%8E%87%E6%8A%80%E6%9C%AF" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>225</td>
<td>Exploitation of Authentication</td>
<td><img src="%E5%88%A9%E7%94%A8%E8%AE%A4%E8%AF%81" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>232</td>
<td>Exploitation of Privilege/Trust</td>
<td><img src="%E5%88%A9%E7%94%A8%E6%9D%83%E9%99%90%E6%88%96%E4%BF%A1%E4%BB%BB" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>255</td>
<td>Data Structure Attacks</td>
<td><img src="%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84%E6%94%BB%E5%87%BB" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>262</td>
<td>Resource Manipulation</td>
<td><img src="%E8%B5%84%E6%BA%90%E6%93%8D%E7%BA%B5" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>286</td>
<td>Network Reconnaissance</td>
<td><img src="%E7%BD%91%E7%BB%9C%E4%BE%A6%E5%AF%9F" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>403</td>
<td>Social Engineering Attacks</td>
<td><img src="%E7%A4%BE%E4%BC%9A%E5%B7%A5%E7%A8%8B%E5%AD%A6%E6%94%BB%E5%87%BB" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>436</td>
<td>Physical Security Attacks</td>
<td><img src="%E7%89%A9%E7%90%86%E5%AE%89%E5%85%A8%E6%94%BB%E5%87%BB" alt="Image" /></td>
</tr>
<tr>
<td>HasMember</td>
<td>🍀</td>
<td>437</td>
<td>Supply Chain Attacks</td>
<td><img src="%E4%BE%9B%E5%BA%94%E9%93%BE%E6%94%BB%E5%87%BB" alt="Image" /></td>
</tr>
</tbody>
</table>

### CAPECs in this view

- **Total CAPECs:** 474
- **Views:** 0
- **Categories:** 68
- **Attack Patterns:** 400

---

*Page Last Updated: May 04, 2012*
Sharing knowledge of our opponents and watching the plays develop, we can make the saves that protect our networks and the software running on them.
Getting Started in Software Assurance (SwA)

Recognizing that your software environment and program’s software supply chain has weaknesses that may be exploited by attackers as operational vulnerabilities is a major step in securing your software supply chain. However, this step pales in comparison to the enormity of securing the entire supply chain for your software. The key to improving your software assurance is to make incremental improvements in the security of the software in your supply chain. No single remedy will absolve or mitigate all of the weaknesses in your software, or the risk. Several methods, tools, and culture changes will be required in concert to build a secure supply chain to cover the known-unknown weaknesses. There is no crystal ball, or magic wand, you can use to ensure your software is absolutely secure against the unknown-unknown weaknesses. However, you can take steps to reduce the risk and exposure of your software and users to new, or existing, software vulnerabilities.

This section of the CWE Web site introduces specific steps you can take to assess your individual software assurance situation and compose a tailored plan to strengthen your assurance of the integrity, reliability, and robustness of your software supply chain. Learn more by following the links below:

- Engineering for Attacks
- Software Quality
- Prioritizing Common Weaknesses Based Upon Your Environment
- Manageable Steps
- Software Assurance Pocket Guide Series
- Staying Informed
- Finding More Information about Software Assurance
Questions?

ramartin@mitre.org