Engineering Your Software For Attack

Robert A. Martin

Senior Principal Engineer

Cyber Security Center

Center for National Security

The MITRE Corporation

MITRE

World Series

Red Sox lead series 4-2

Game 6, Wednesday, October 30, 8:07 PM (ET)
Fenway Park, Boston, Massachusetts

St. Louis Cardinals												
	1	2	3	4	5	6	7	8	9	R	H	E
Cardinals	0	0	0	0	0	0	1	0	0	1	9	1
Red Sox	0	0	3	3	0	0	0	0	x	6	8	1

Final

Boston Red Sox

Game 7 - Thu, Oct 31 Cardinals @ Red Sox 8:07 PM (ET)

Red Sox find safety in numbers, more memorable Game 6 scenes

SOX ARE CHAMPS

In Series win, Sox go from worst to first

Create account Log in

Article Talk Read View source View history Search

2013 World Series

From Wikipedia, the free encyclopedia

The **2013 World Series** was the 109th 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.[1] 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 1999 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 1958 Series) to feature two teams with identical regular-season records.[2] 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

2013 World Series

Team (Wins)	Manager	Season
Boston Red Sox (4)	John Farrell	97–65, .599, 5.5 GA
St. Louis Cardinals (2)	Mike Matheny	97–65, .599, 3 GA

Dates: October 23–30

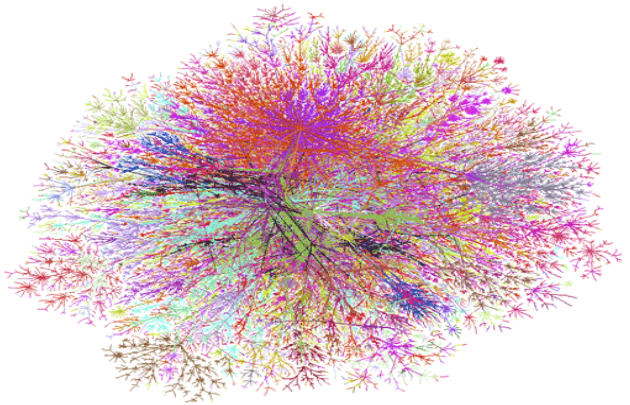
Navigation
- Main page
- Contents
- Featured content
- Current events
- Random article
- Donate to Wikipedia

Interaction
- Help
- About Wikipedia
- Community portal
- Recent changes
- Contact page

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

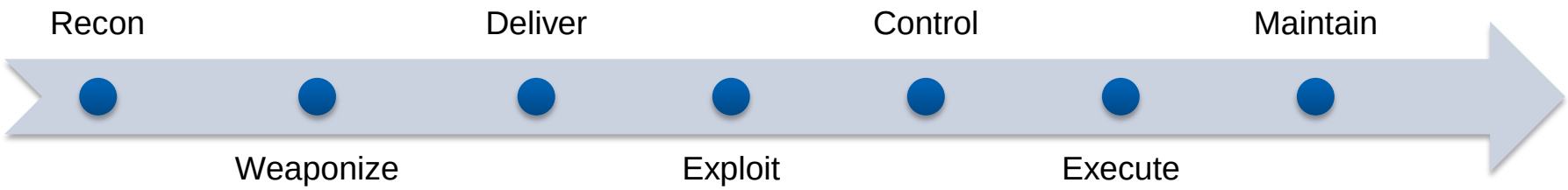


Cyber Attack Lifecycle

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



Proactive Detection Mitigation

Incident Response & Mission Assurance

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**

MITRE

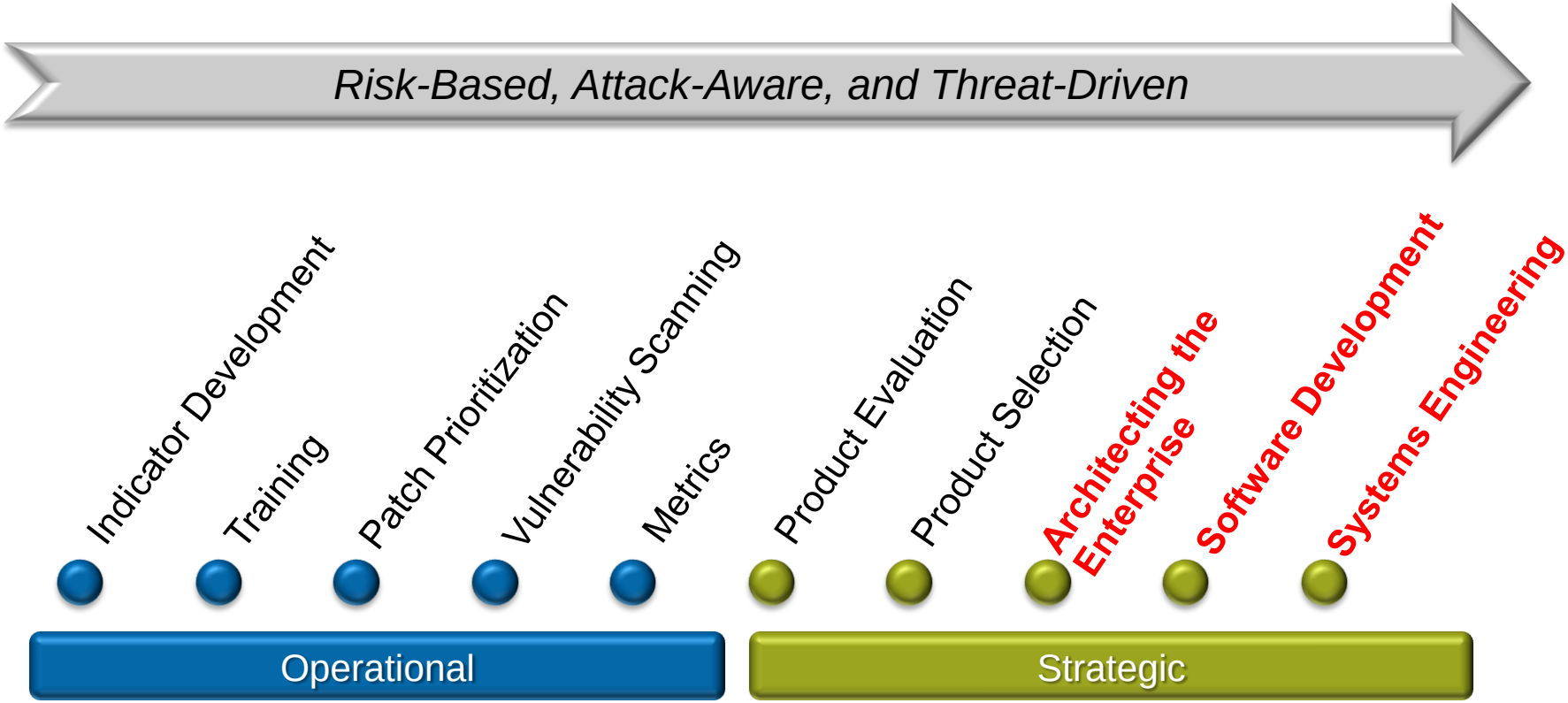

player
penalty
Home
Guests
penalty
player
player
penalty
period
penalty
player
shots on goal


HOCKEY
1

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

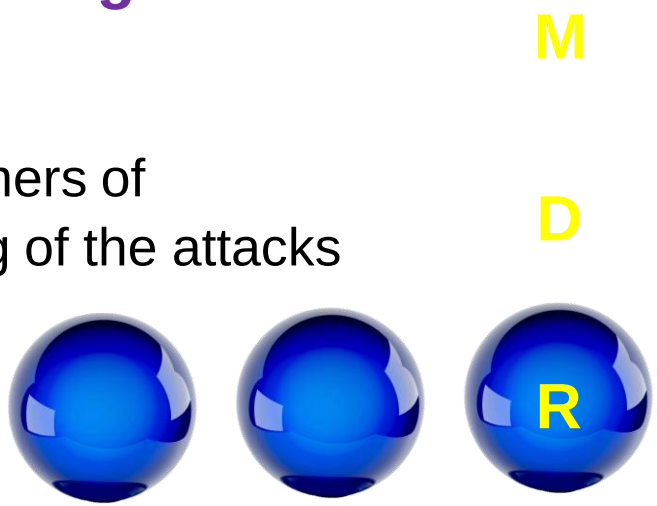


From Just a Mitigation Approach

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



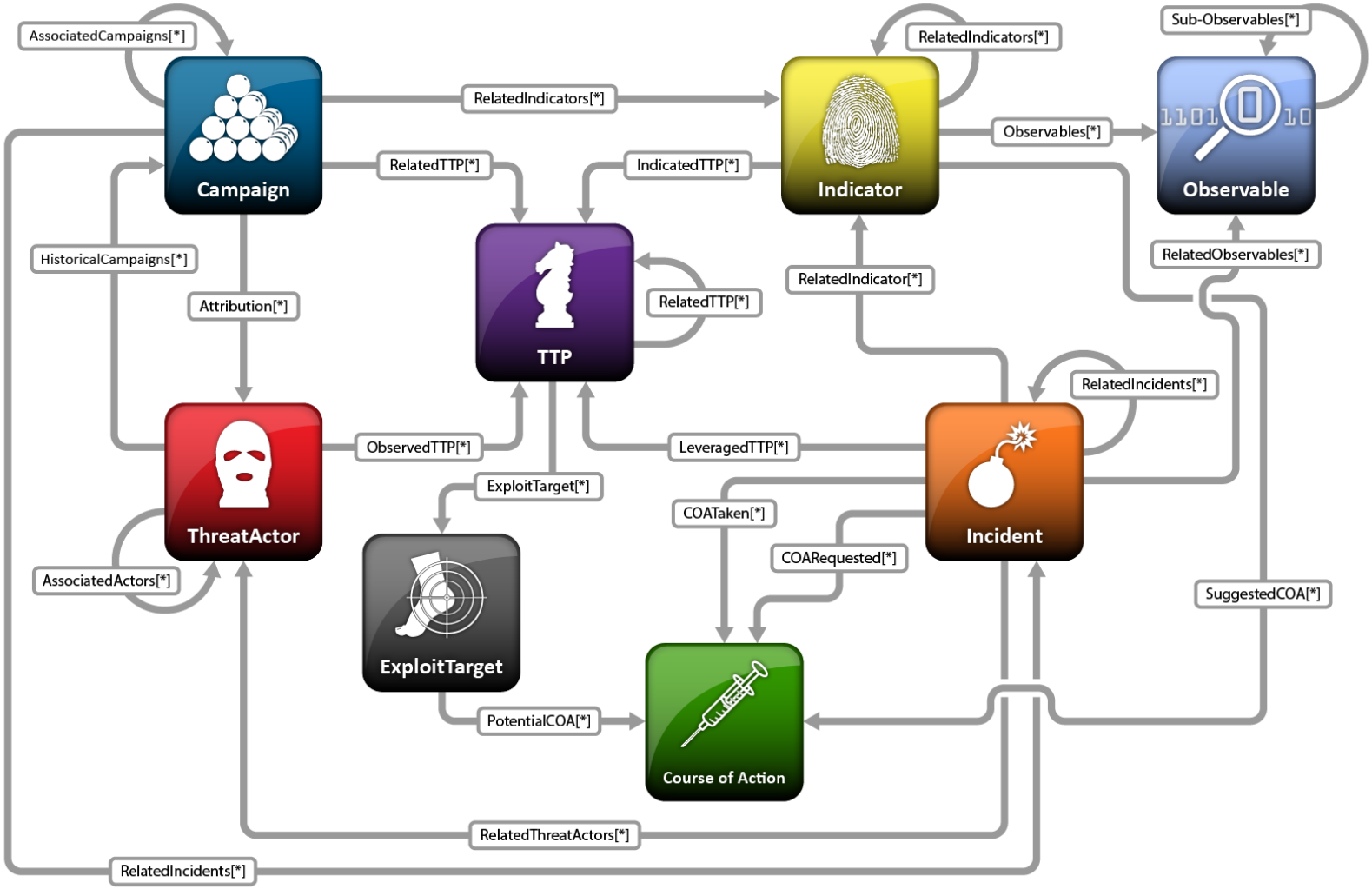
What is “Cyber Threat Intelligence?”

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?

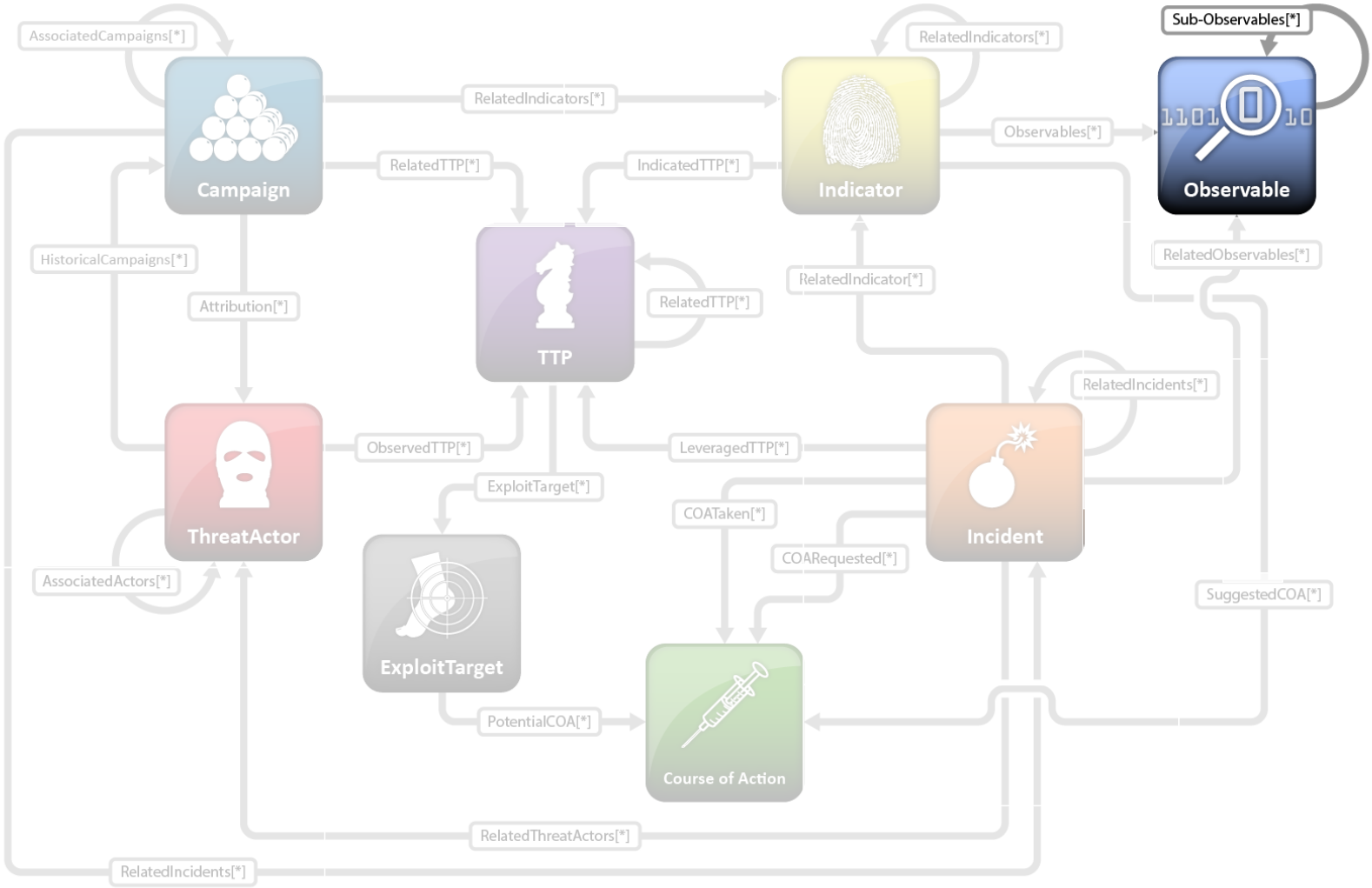
MITRE

Structured Threat Information eXpression (STIX) v1.0 Architecture



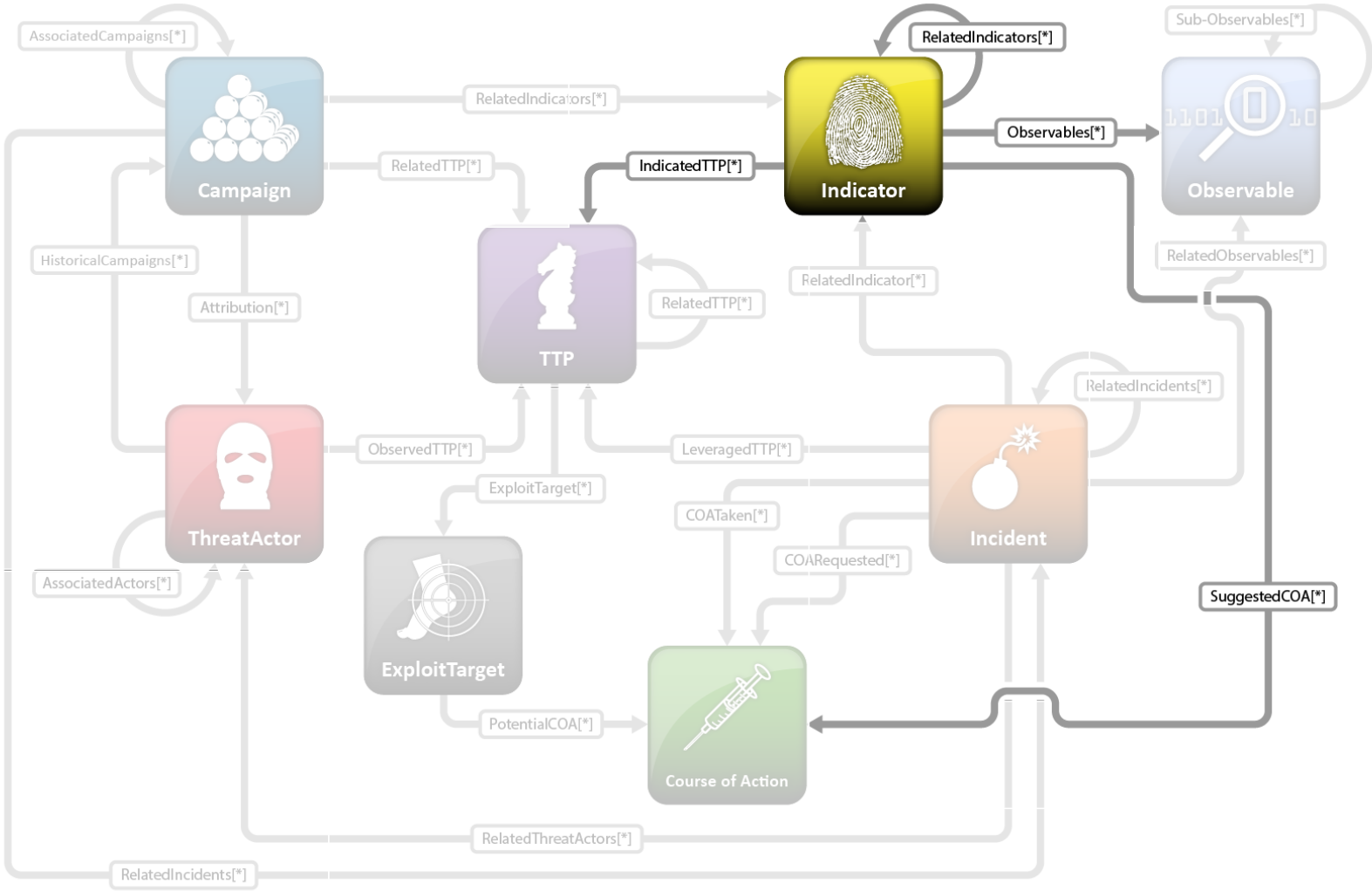
MITRE

Structured Threat Information eXpression (STIX) v1.0 Architecture



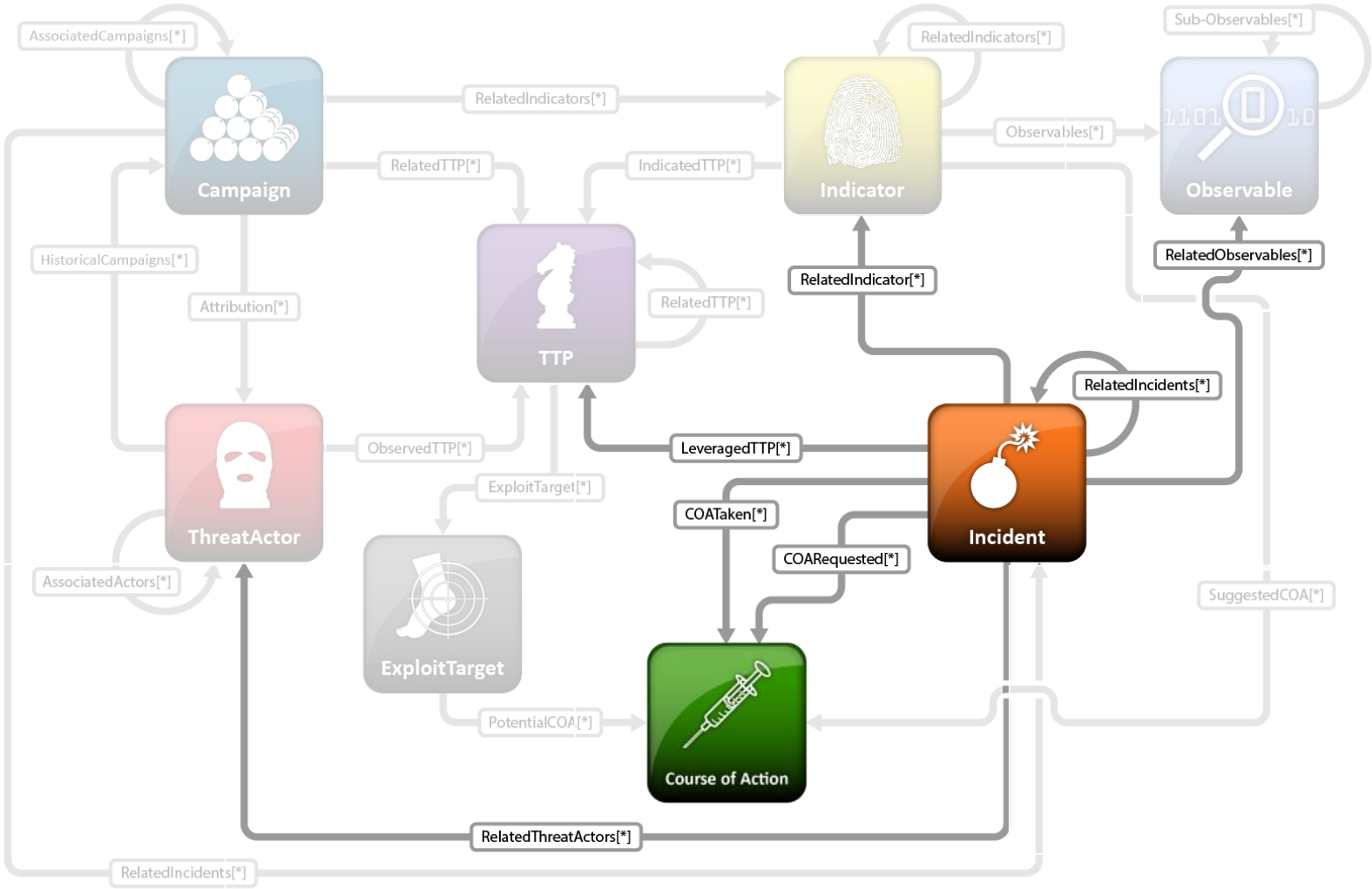
MITRE

Structured Threat Information eXpression (STIX) v1.0 Architecture



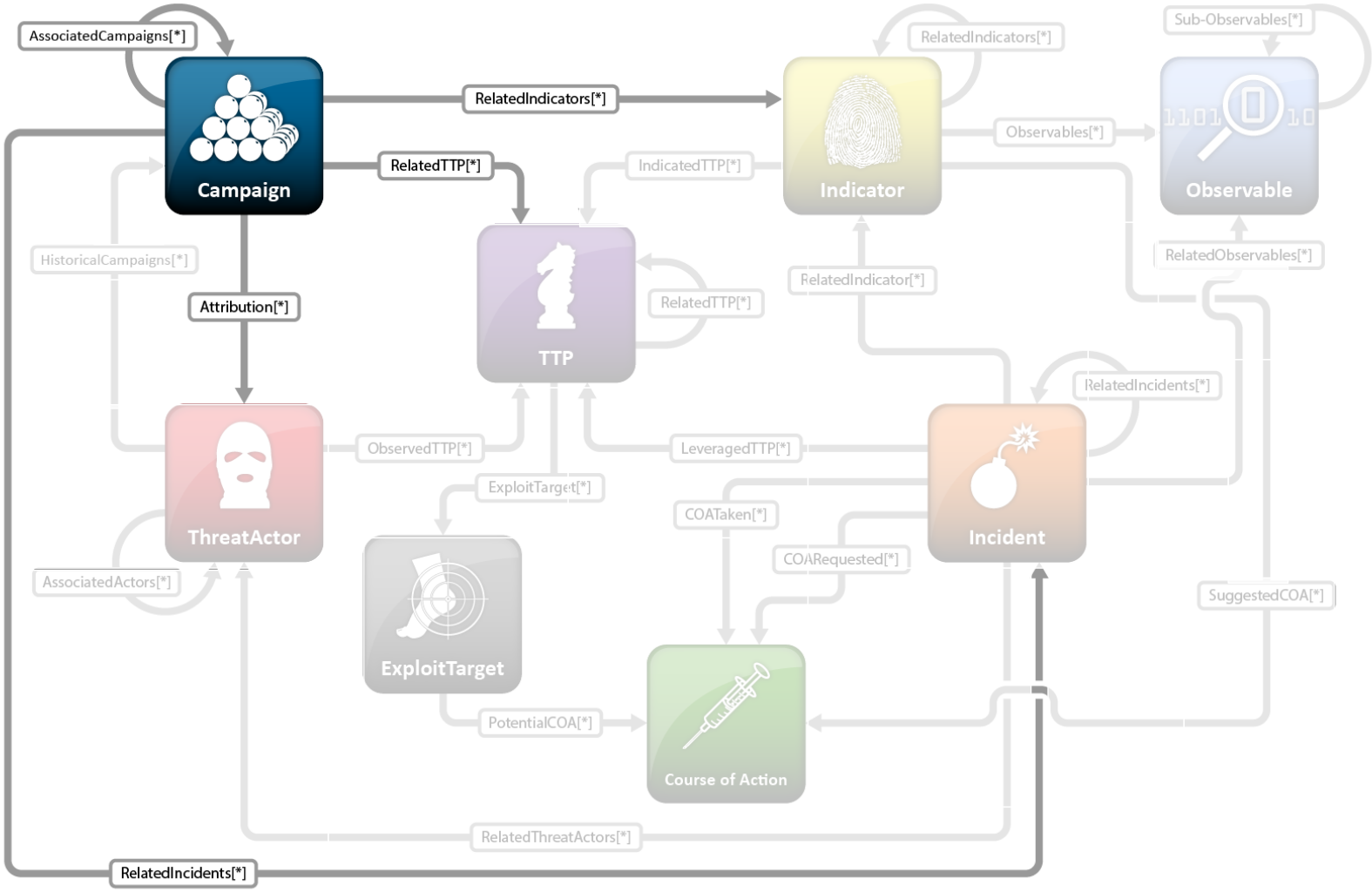
MITRE

Structured Threat Information eXpression (STIX) v1.0 Architecture



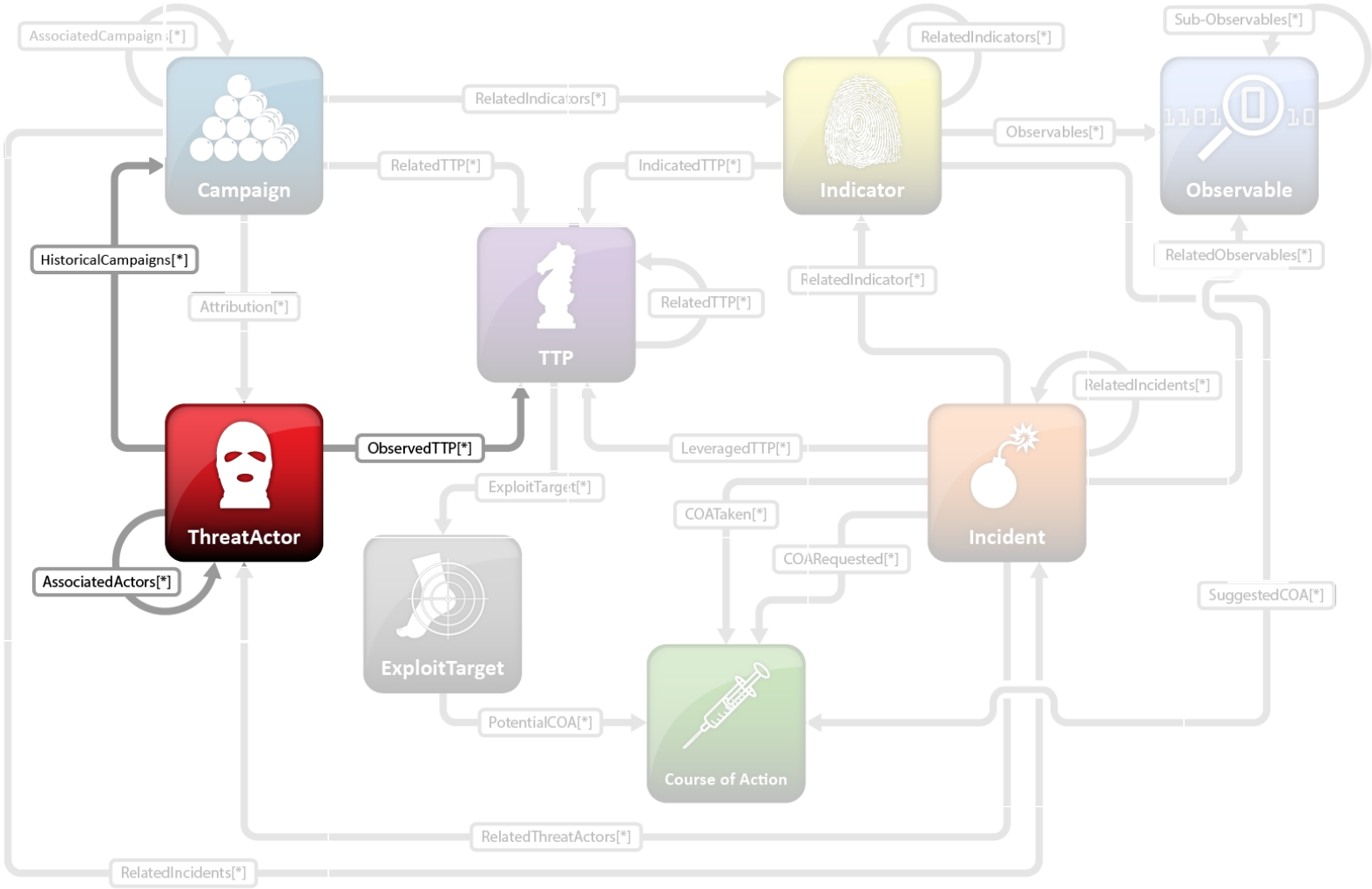
MITRE

Structured Threat Information eXpression (STIX) v1.0 Architecture



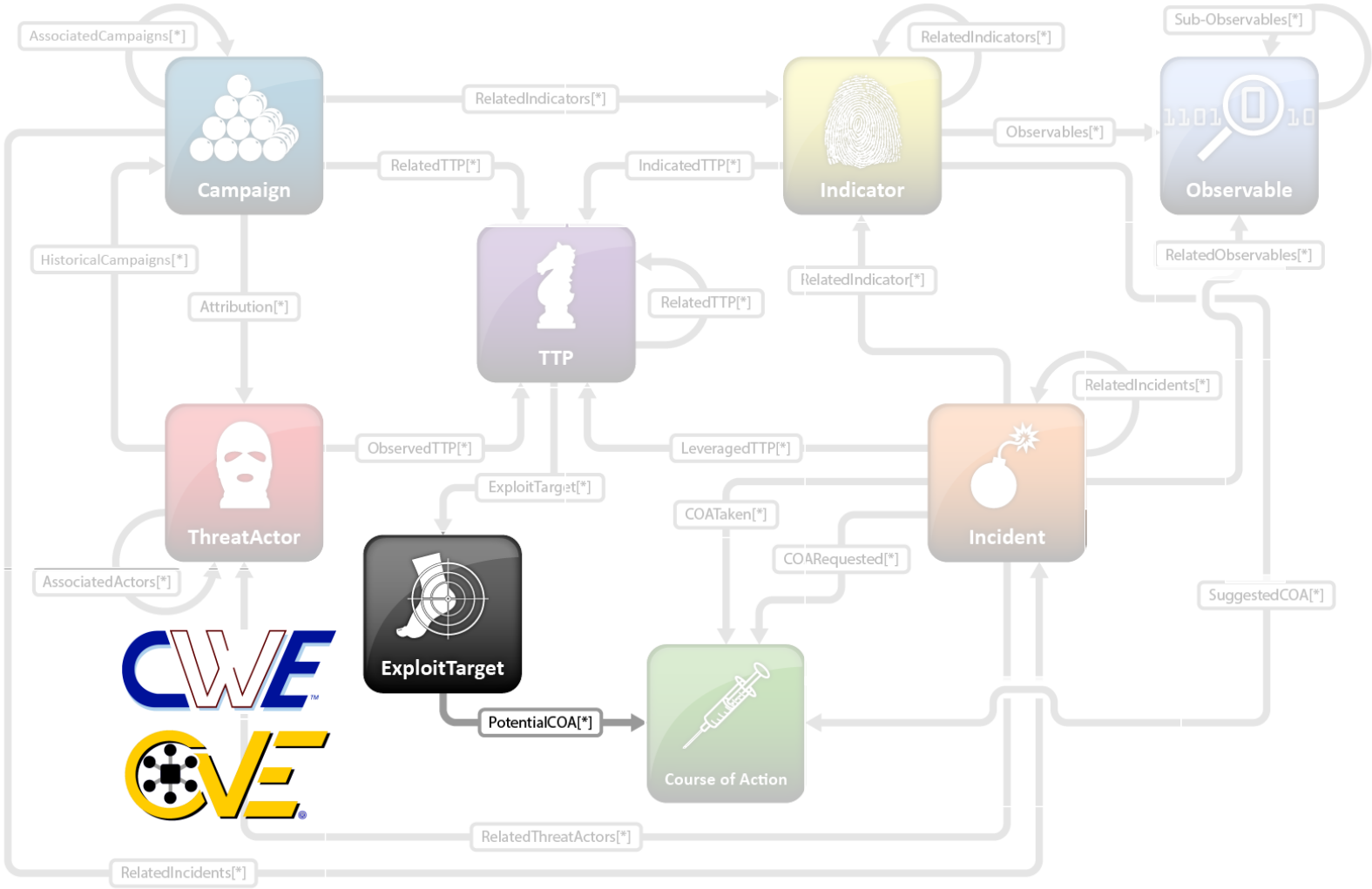
MITRE

Structured Threat Information eXpression (STIX) v1.0 Architecture



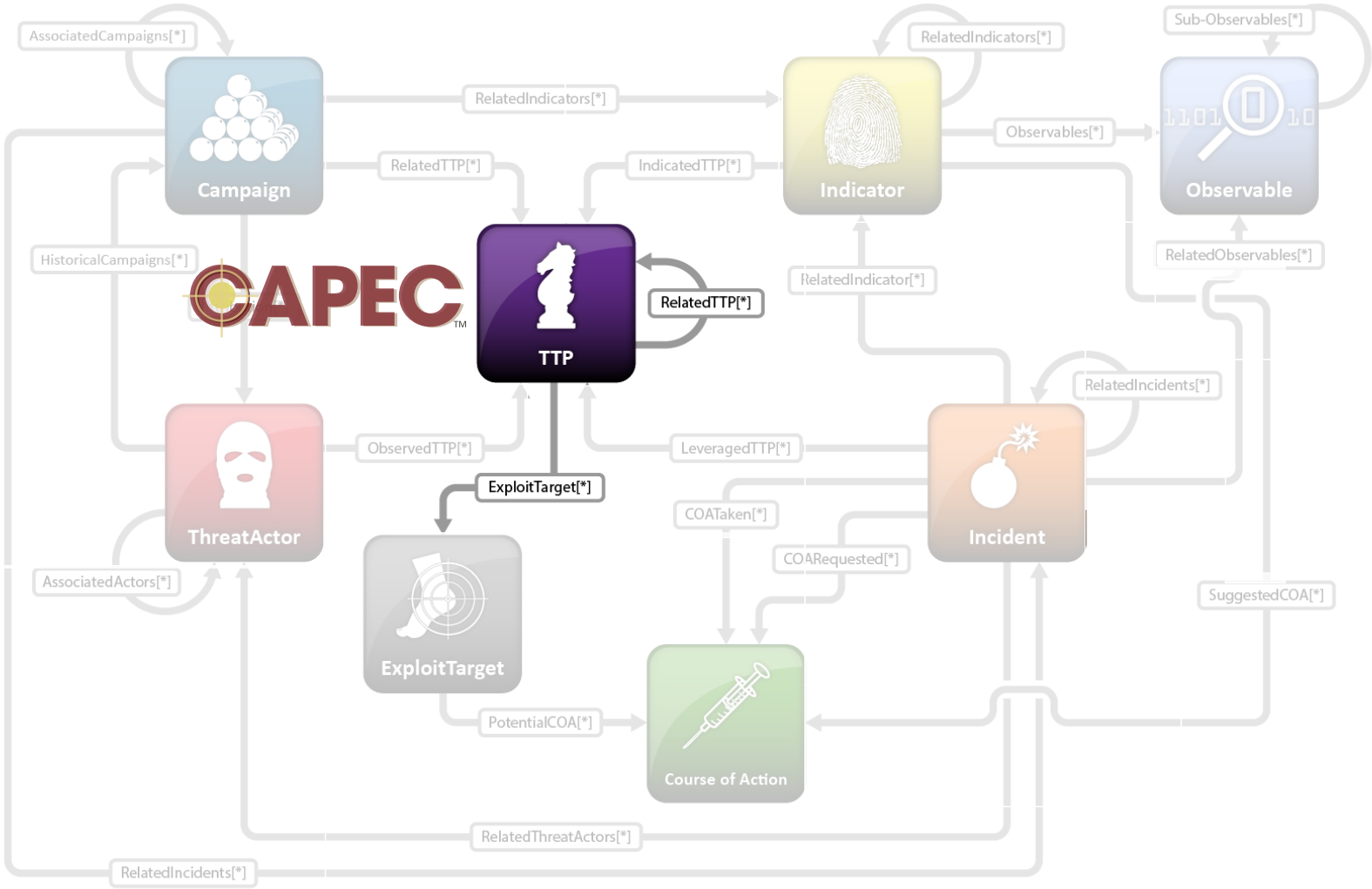
MITRE

Structured Threat Information eXpression (STIX) v1.0 Architecture



MITRE

Structured Threat Information eXpression (STIX) v1.0 Architecture



MITRE

Threat Classification Taxonomy Cross Reference View

last edited by Robert Auger 10 months, 3 weeks ago

Page history

Tags: Threat Classification

Check for plagiarism

Threat Classification 'Taxonomy Cross Reference View'

This view contains a mapping of the WASC Threat Classification's Attacks and Weaknesses with MITRE's Common Weakness Enumeration, MITRE's Common Attack Pattern Enumeration and Classification, OWASP Top Ten 2010 RC1 (original mapping with OWASP Top Ten from Jeremiah Grossman & Bill Corry) and SANS/CWE and OWASP Top Ten 2007 and 2004 (original mapping from Dan Cornell, Denim Group)

WASC ID	Name	CWE ID	CAPEC ID	SANS/CWE Top 25 2009	OWASP Top Ten 2010	OWASP Top Ten 2007	OWASP Top Ten 2004
WASC-01	Insufficient Authentication	287		642	A3 – Broken Authentication and Session Management, A4 – Insecure Direct Object References	A7 – Broken Authentication and Session Management, A4 – Insecure Direct Object Reference	A3 – Broken Authentication and Session management, A2 – Broken Access Control
WASC-02	Insufficient Authorization	284		285	A4 – Insecure Direct Object References, A7 – Failure to Restrict URL Access	A10 – Failure to Restrict URL Access, A4 – Insecure Direct Object Reference	A2 – Broken Access Control
WASC-03	Integer Overflows	190	128	682			
WASC-04	Insufficient Transport Layer Protection	311 523		319	A10 – Insufficient Transport Layer Protection	A9 – Insecure Communications	
WASC-05	Remote File Inclusion	98	193 253	426		A3 – Malicious File Execution	
WASC-06	Format String	134	67				
WASC-07	Buffer Overflow	119 120	10 100	119			A5 – Buffer Overflows
WASC-08	Cross-site Scripting	79	18 19 63	79	A2 – Cross-Site Scripting	A1 – Cross Site Scripting (XSS)	A4 – Cross Site Scripting (XSS)
WASC-09	Cross-site Request Forgery	352	62	352	A5 – Cross-Site Request Forgery	A5 – Cross Site Request Forgery (CSRF)	
WASC-10	Denial of Service	400	119	404	A7 – Failure to Restrict	A10 – Failure to	A9 – Denial of

SideBar

WASC Projects
- Distributed Open Proxy Honeypots
- Script Mapping
- The Web Security Glossary
- Web Application Firewall Evaluation Criteria
- Web Application Security Scanner Evaluation Criteria
- Web Application Security Statistics
- Web Hacking Incidents Database
- WASC Threat Classification

WASC Project Leaders
- Robert Auger
- Ryan Barnett
- Romain Gaucher
- Sergey Gordeychik
- Ofer Shezaf
- Brian Shura

WASC Main Website
- http://www.webappsec.org/

WASC Mailing Lists
- http://lists.webappsec.org/

WASC on Twitter
- http://twitter.com/wascupdates

Join us on Linkedin!
- http://www.linkedin.com/groups?gid=83336

Recent Activity

Insufficient Data Protection Working edited by Robert Auger

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. Sect933

confidence

functions as intended

free of vulnerabilities

DoD Software-based System

Program Office Milestone Reviews with OSD on SwA

Program Protection Plan's "Application of Software Assurance Countermeasures"

Development Process
- Static Analysis
- Design Inspection
- Code Inspections
- CVE
- CAPEC
- CWE
- Pen Test
- Test Coverage

Operational System
- Failover Multiple Supplier Redundancy
- Fault Isolation
- Least Privilege
- System Element Isolation
- Input checking/validation
- SW load key

Development Environment
- Source
- Release Testing
- Generated code inspection

MITRE

Software Assurance Methods

Countermeasure Selection

Development Process

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

Operational System

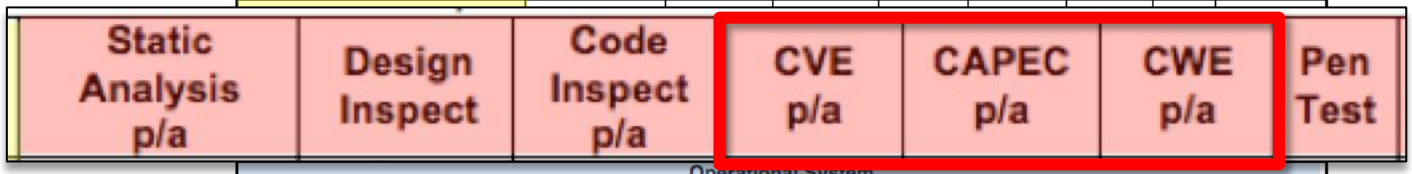
Implement countermeasures to the design and acquisition of end-item software products and their interfaces

Development Environment

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

Table 5.3-5-5: Application of Software Assurance Countermeasures (sample)

Development Process								
Software (CPI, critical function components, other software)	**Static Analysis p/a**	**Design Inspect**	**Code Inspect p/a**	**CVE p/a**	**CAPEC p/a**	**CWE p/a**	**Pen Test**	**Test Coverage p/a**
Developmental CPI SW	100/80%	Two Levels	100/80	100/60	100/60	100/60	Yes	75/50%
Developmental Critical Function SW	100/80%	Two Levels	100/80	100/70	100/70	100/70	Yes	75/50%



Operational System						
	Failover Multiple Supplier Redundancy	**Fault Isolation**	**Least Privilege**	**System Element Isolation**	**Input checking / validation**	**SW load key**
Developmental CPI SW	30%	All	all	yes	All	All
Developmental Critical Function SW	50%	All	All	yes	All	all
Other Developmental SW	none	Partial	none	None	all	all
COTS (CPI and CF) and NDI SW	none	Partial	All	None	Wrappers/ all	all

Development Environment								
SW Product	**Source**	**Release testing**	**Generated code inspection p/a**					
C Compiler	No	Yes	50/20					
Runtime libraries	Yes	Yes	70/none					
Automated test system	No	Yes	50/none					
Configuration management system	No	Yes	NA					
Database	No	Yes	50/none					
Development Environment Access	Controlled access; Cleared personnel only							

Additional Guidance in PPP Outline and Guidance

MITRE

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

FY 2013

Chief Information Officer

Federal Information Security Management Act

Reporting Metrics

Prepared by:

US Department of Homeland Security

Office of Cybersecurity and Communications

Federal Network Resilience

November 30, 2012

4. VULNERABILITY AND WEAKN

Purpose and Use

- Unpatched vulnerabilities are a major
- A key goal of vulnerability managemer

of vulnerabilities identifie
tion is that vulnerability
er asset management). Th
ility management capabil
—covering enough of the
for a successful attack
able to find and fix vulner
—has a low enough rate c
, to avoid unknown weak

ge of network boundary c
ue to be adequately free

ge of hardware assets ide
dentifies NIST National Vu
the organization's enter

ercentage of hardware as
he security of the system

Common Vulnerabilitie

Common Vulnerability

Open Vulnerability and

ntage of information syste

[44] Once all organizations are reporting monthly to C
[45] The presence of this question about identifying w
organization to use the tools described in section 4.
and remove common weaknesses like register over
from compromising software.

		For systems in development and/or maintenance:		For systems in production:	
		Use methods described in Table 9 to identify and fix instances of common weaknesses, prior to placing that version of the code into production.	Can the organization find SCAP compliant tools and good SCAP content?	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).	Can the organization find SCAP compliant tools and good SCAP content?
Impact Level	High				
	Moderate				
	Low				

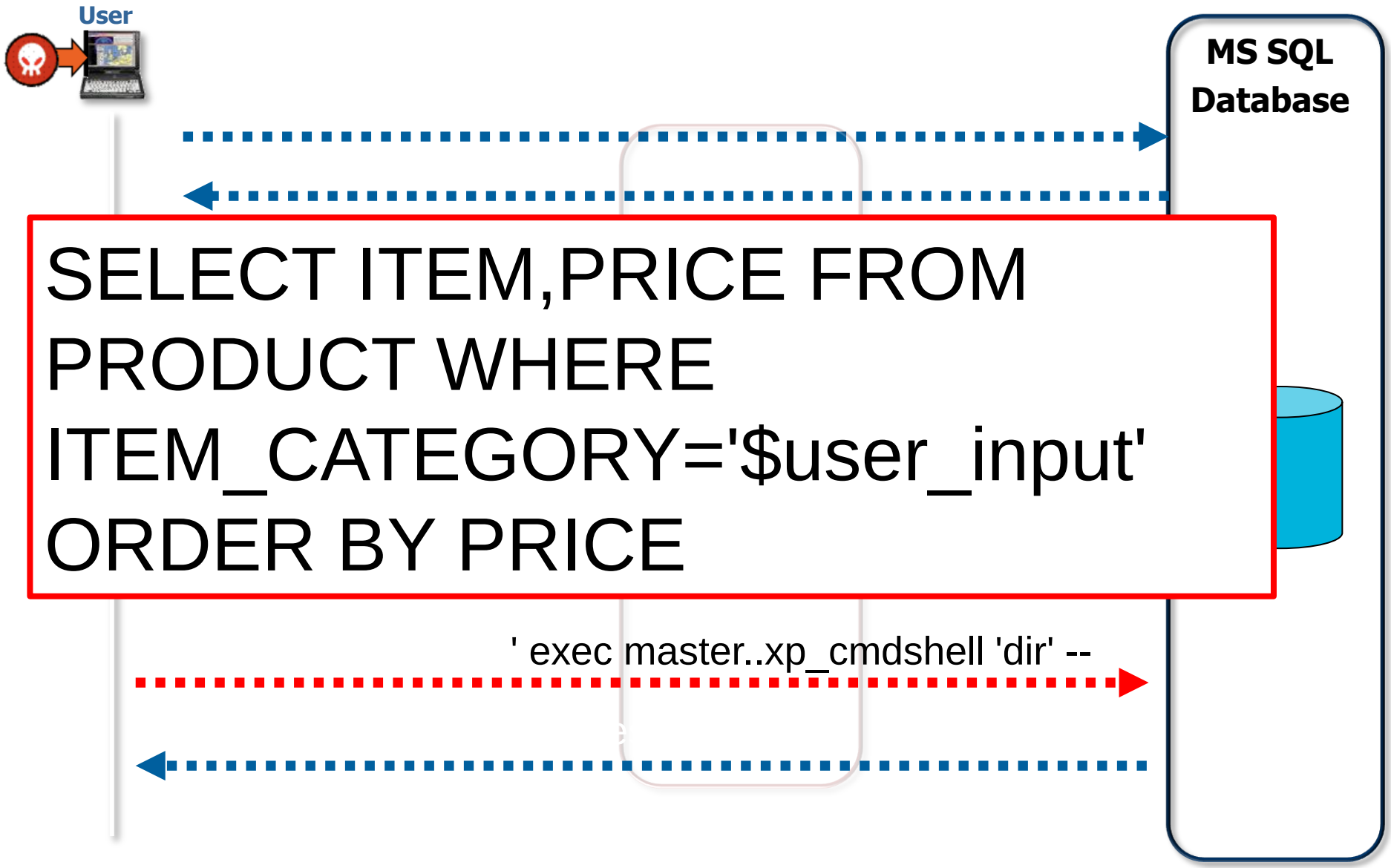
Table 8 – Responses to Question 4.3

Identify Universe Enumeration	Find Instances Tools and Languages	Assess Importance
• Common Weakness Enumeration (CWE) • Web scanners for web-based applications	• Static Code Analysis tools • Manual code reviews (especially for weaknesses not covered by the automated tools)	• Common Weakness Scoring System (CWSS)
• Common Attack Pattern Enumeration and Classification (CAPEC)	• Dynamic Code Analysis tools • Web scanners for web-based applications • PEN testing for attack types not covered by the automated tools.	—

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

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



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 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


Google Earth
ファイル(F) 編集(E) 表示(V) ツール(T) 追加(A) ヘルプ(H)
Severity
Emergency
Critical
Warning
JSOC
SQL Injection Probe Detected.
SQL Injection Detected
Image IBCAO
Image © 2010 TerraMetrics
Image USDA Farm Service Agency
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
©2010 Google

Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

CWE and SANS Institute TOP 25 MOST DANGEROUS SOFTWARE ERRORS

CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
Community
Related Activities
Discussion List
Research
CWE/SANS Top 25
CWSS
News
Calendar
Free Newsletter
Compatibility
Program
Requirements
Declarations
Make a Declaration
Contact Us
Search the Site

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)* **Status:** Draft

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

MITRE

CWE Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

CWE & SANS Institute TOP 25 MOST DANGEROUS SOFTWARE ERRORS

CWSS CWRAF

CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
FAQs
Community
Use & Citations
SwA On-Ramp
T-Shirt
Discussion List
Discussion Archives
Contact Us
Scoring
CWSS
CWRAF
CWE/SANS Top 25
Compatibility
Requirements
Coverage Claims Representation
Compatible Products
Make a Declaration
News
Calendar
Free Newsletter
Search the Site

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')

Weakness ID: 78 *(Weakness Base)* **Status:** Draft

Description

Description Summary

The software constructs
component, but it does
command when it is sen

Extended Description

This could allow attacker
weakness can lead to a
operating system, such
allow the attacker to spe
privileges that the attack
the principle of least priv
that increases the amou

There are at least two su

1. The application i
externally-supplie
system("nslooku
as an argument.
remove comman
arguments, whic
2. The application a
commands to use
example, the pro
user. If the COM
programs. If the
might not be able

From a weakness standp
programmer clearly inte
executed. In the second
untrusted party, but the
can provide input.

Alternate Terms

Common Consequences

Scope	Effect
Confidentiality Integrity Availability Non-Repudiation	**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* Attackers could execute unauthorized commands, which could then be used to disable the software, or read and modify data for which the attacker does not have permissions to access directly. Since the targeted application is directly executing the commands instead of the attacker, any malicious activities may appear to come from the application or the application's owner.

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 reduction of false positives if all potentially-vulnerable operations can be assessed within limited time constraints.

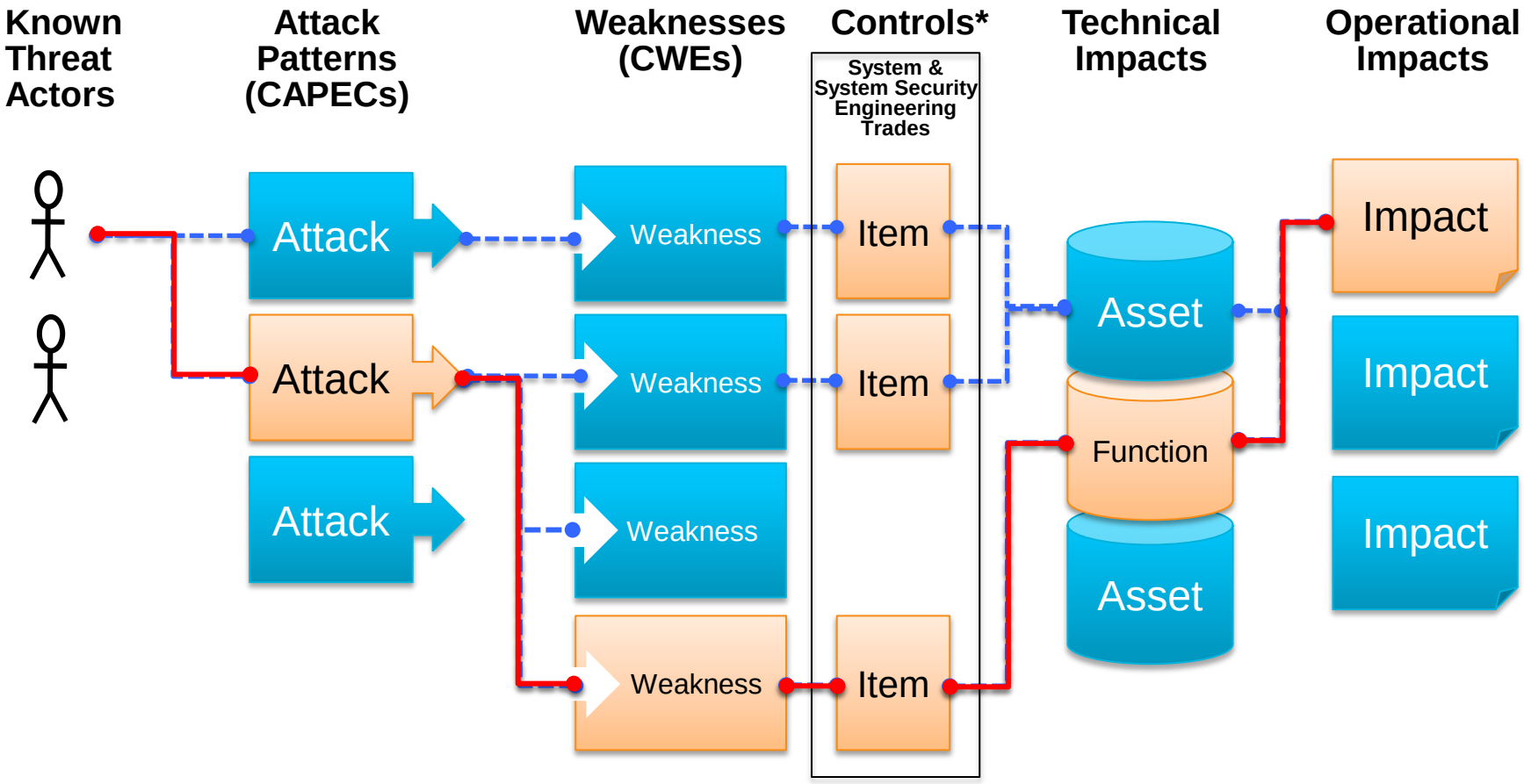
Effectiveness: High

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

Engineering For Attack – ISO/IEC Technical Report 20004:

Refining Software Vulnerability Analysis Under ISO/IEC 15049 and ISO/IEC 18045

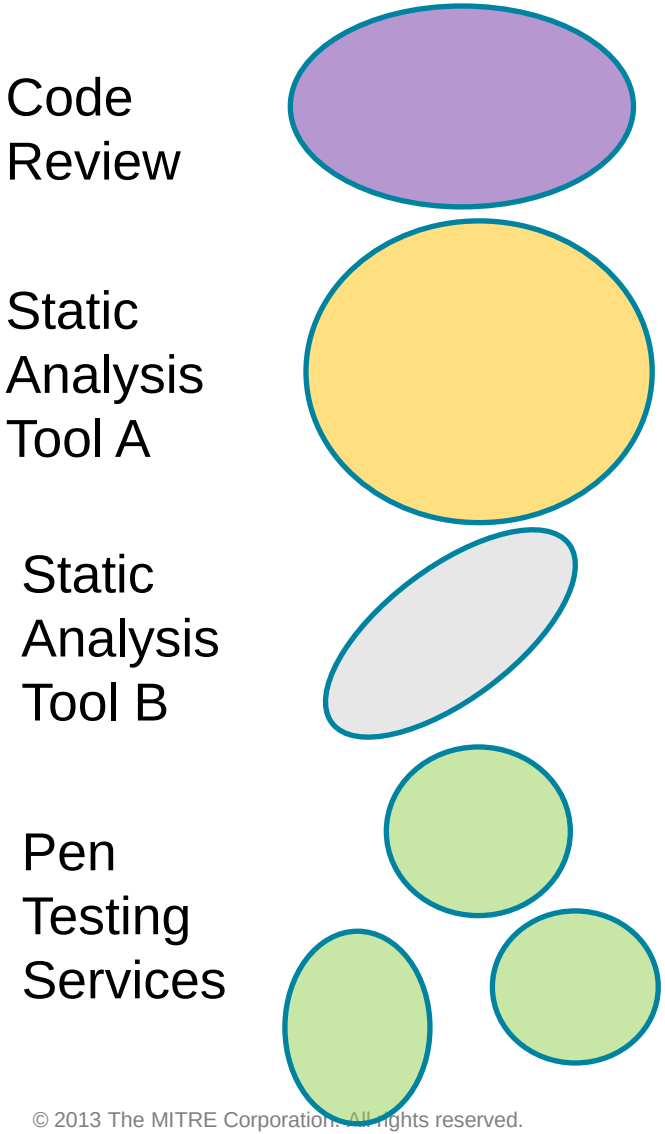


*** Controls include architecture choices, design choices, added security functions, activities & processes, physical decomposition choices, code assessments, design reviews, dynamic testing, and pen testing**

MITRE

Utilizing Coverage Claims

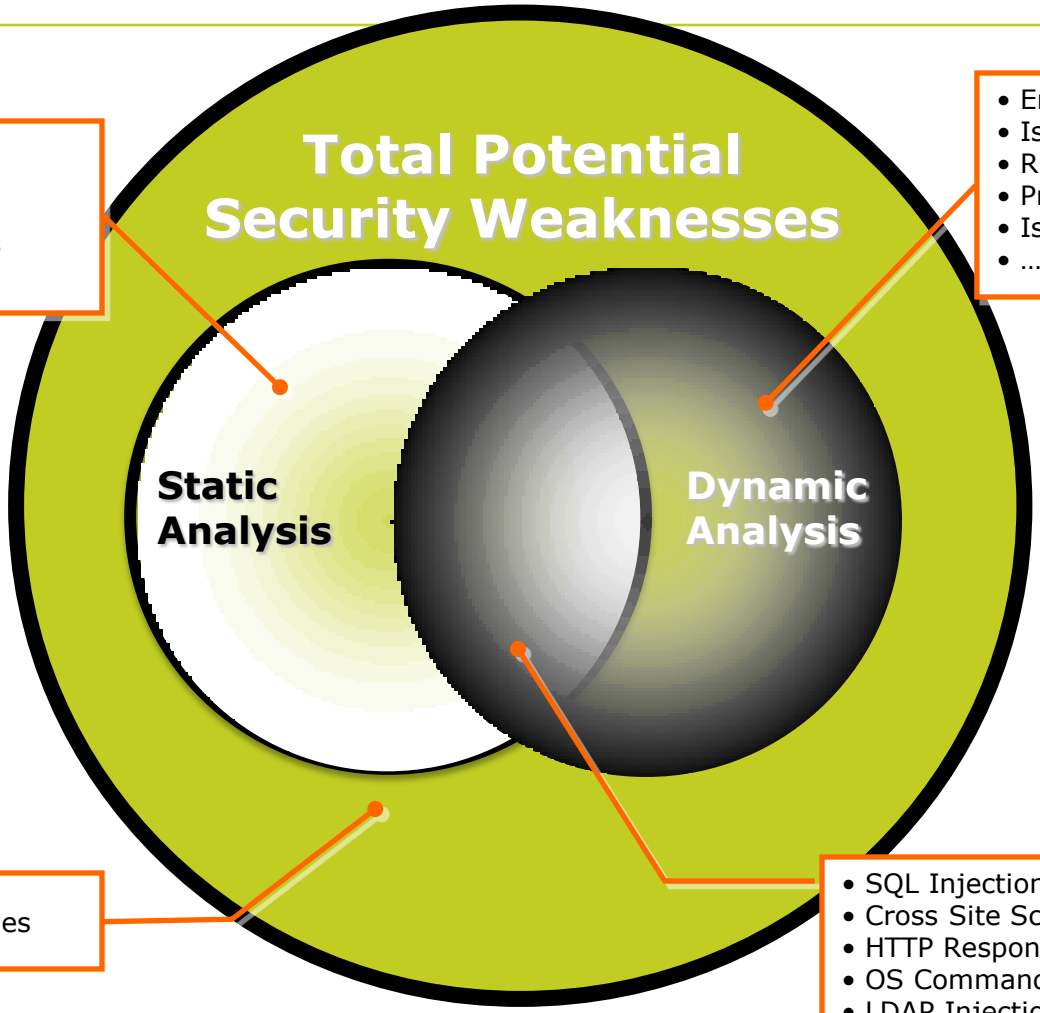
CWE's a capability *claims* to cover



Which static analysis tools and Pen Testing services find the CWE's I care about?

MITRE

Leveraging and Managing to take Advantage of the Multiple Perspectives of Analysis



- 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
- …

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

	Static Code Analysis	Penetration Test	Data Security Analysis	Code Review	*Architecture Risk Analysis*
Cross-Site Scripting (XSS)	X	X		X	
SQL Injection	X	X		X	
Insufficient Authorization Controls		X	X	X	*X*
Broken Authentication and Session Management		X	X	X	*X*
Information Leakage		X	X		*X*
Improper Error Handling	X				
Insecure Use of Cryptography		X		X	*X*
Cross Site Request Forgery (CSRF)		X		X	
Denial of Service	X	X	X		*X*
Poor Coding Practices	*X*			*X*	

Notional

	Architecture Analysis	Design Review	Source Code Static Analysis	Binary Static Analysis	Automated Dynamic Analysis	Penetration Testing	Red Team Assessment
(1) Modify data	Review of Architecture and Design		Review of Code			Review of Live System	
(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							

Notional

Vulnerability Analysis Focus By Phase and Impact

	Architecture Analysis	Design Review	Source Code Static Analysis	Binary Static Analysis	Automated Dynamic Analysis	Penetration Testing	Red Team Assessment
(1) Modify data	CWE-23	CWE-23	CWE-131	CWE-131	CWE-311	CWE-311	CWE-311
	Relative Path Traversal		**Incorrect Calculation of Buffer Size**		**Missing Encryption of Sensitive Data**		
(2) Read Data	CWE-14	CWE-14	CWE-129	CWE-129	CWE-209	CWE-209	CWE-209
	Compiler Removal of Buffer Clearing		**Improper Validation of Array Index**		**Information Exposure Through an Error Messages**		
(3) DoS: unreliable execution	CWE-36	CWE-36	CWE-476	CWE-476	CWE-406	CWE-406	CWE-406
	Absolute Path Traversal		**Null Pointer Dereference**		**Network Amplification**		
(4) DoS: resource consumption	CWE-395	CWE-395	CWE-190	CWE-190	CWE-412	CWE-412	CWE-412
	Use of NullPointerException		**Integer Overflow**		**Unrestricted Externally Accessible Lock**		
(5) Execute unauthorized code or commands	CWE-88	CWE-88	CWE-120	CWE-120	CWE-120	CWE-79	CWE-79
	Argument Injection		**Buffer Overflow**		**Cross-site Scripting**		
(6) Gain privileges / assume identity	CWE-96	CWE-96	CWE-489	CWE-489	CWE-309	CWE-309	CWE-309
	Static Code Injection		**Leftover Debug Code**		**Use of Password System for Primary Authentication**		
(7) Bypass protection mechanism	CWE-89	CWE-89	CWE-357	CWE-357	CWE-665	CWE-665	CWE-665
	SQL Injection		**Insufficient UI Warning of Dangerous**		**Improper Initialization**		
(8) Hide activities	CWE-78	CWE-78	CWE-168	CWE-168	CWE-444	CWE-444	CWE-444
	OS Command Injection		**Improper Handling of Inconsistent**		**HTTP Request Smuggling**		

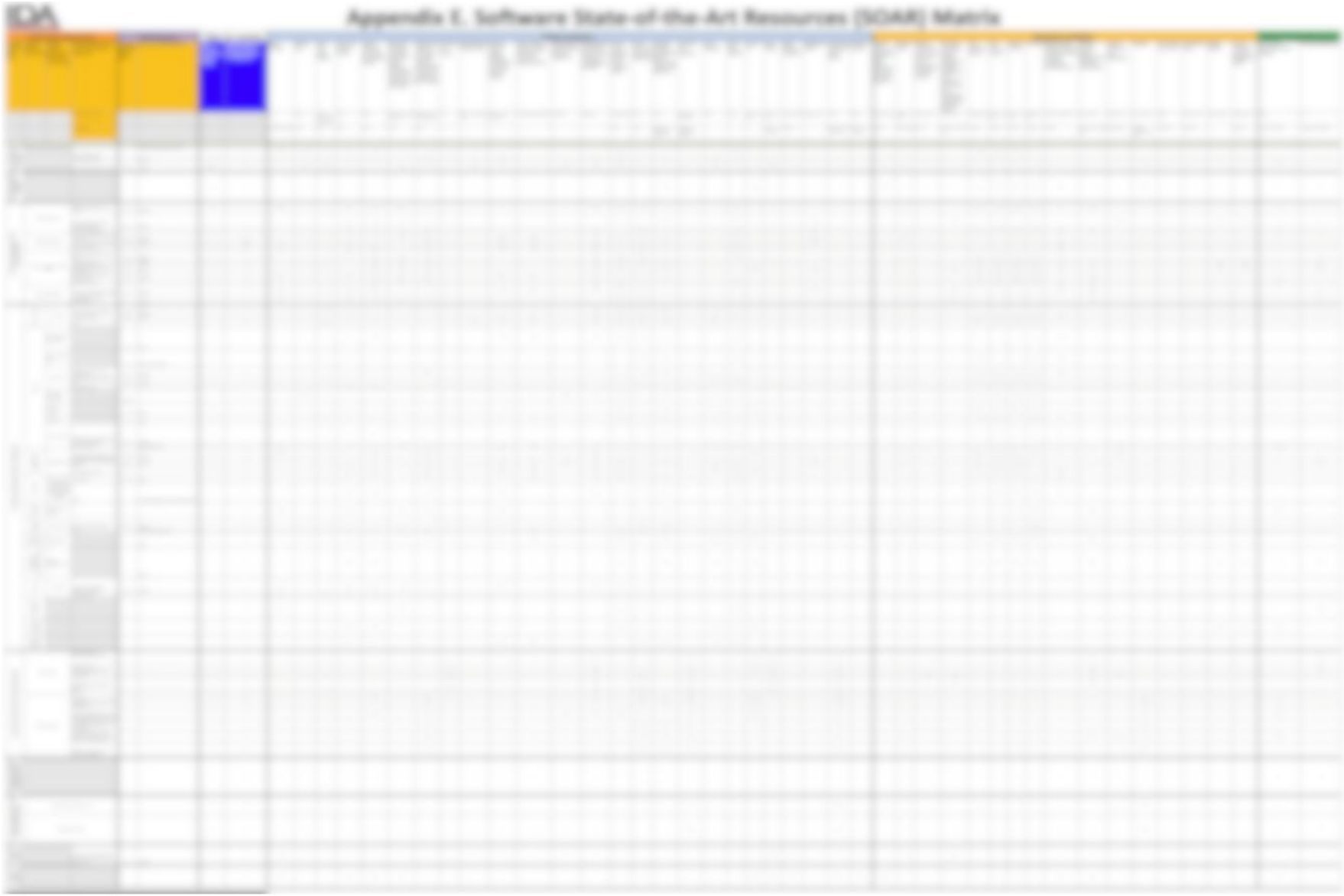
Impacts by Detection Method

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

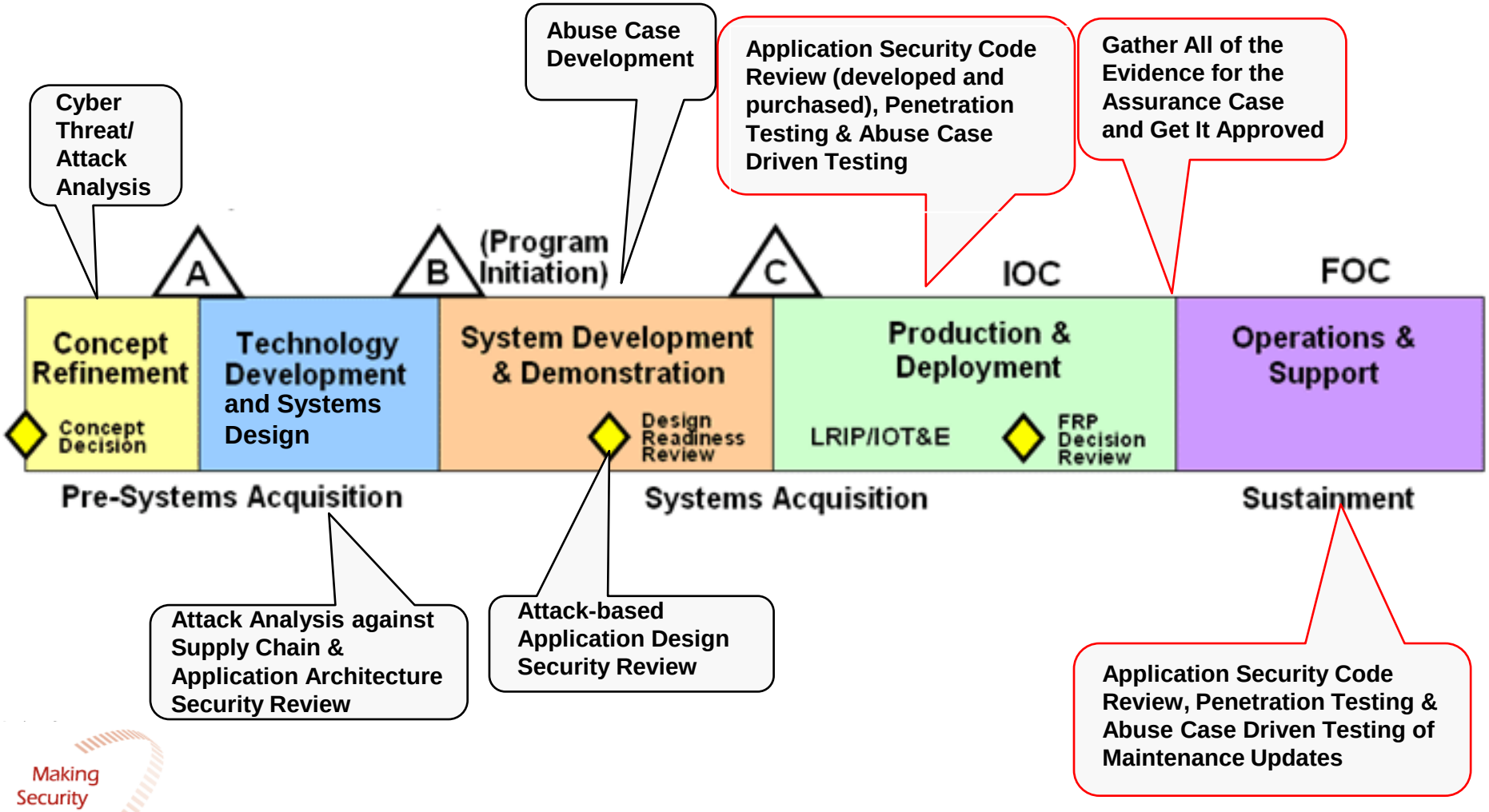
Notional

Technical Impact	Automated Analysis	Automated Dynamic Analysis	Automated Static Analysis	Black Box	Fuzzing	Manual Analysis	Manual Dynamic Analysis	Manual Static Analysis	Other	White Box
Execute unauthorized code or commands		78, 120, 129, 131, 476, 805	78, 79, 98, 120, 129, 131, 134, 190, 798, 805	79, 129, 134, 190, 494, 698, 798		98, 120, 131, 190, 494, 805	476, 798	78, 798		
Gain privileges / assume identity			798	259, 798		259	798	798, 807	628	
Read data	209, 311, 327	78, 89, 129, 131, 209, 404, 665	78, 79, 89, 129, 131, 134, 798	14, 79, 129, 134, 319, 798		89, 131, 209, 311, 327	209, 404, 665, 798	78, 798		14
Modify data	311, 327	78, 89, 129, 131	78, 89, 129, 131, 190	129, 190, 319		89, 131, 190, 311, 327		78		
DoS: unreliable execution		78, 120, 129, 131, 400, 476, 665, 805	78, 120, 129, 131, 190, 400, 805	129, 190	400	120, 131, 190, 805	476, 665	78		
DoS: resource consumption		120, 400, 404, 770, 805	120, 190, 400, 770, 805	190	400, 770	120, 190, 805	404	770		412
Bypass protection mechanism		89, 400, 665	79, 89, 190, 400, 798	14, 79, 184, 190, 733, 798	400	89, 190	665, 798	798, 807		14, 733
Hide activities	327	78	78			327		78		
Other		400, 404	400, 798	198, 484, 494, 698, 733, 798	400	494	404, 798	596, 798, 807	628	484, 733

Planning to Leverage "State of the Art Resource" (SOAR): Software Table of "Verification Methods"

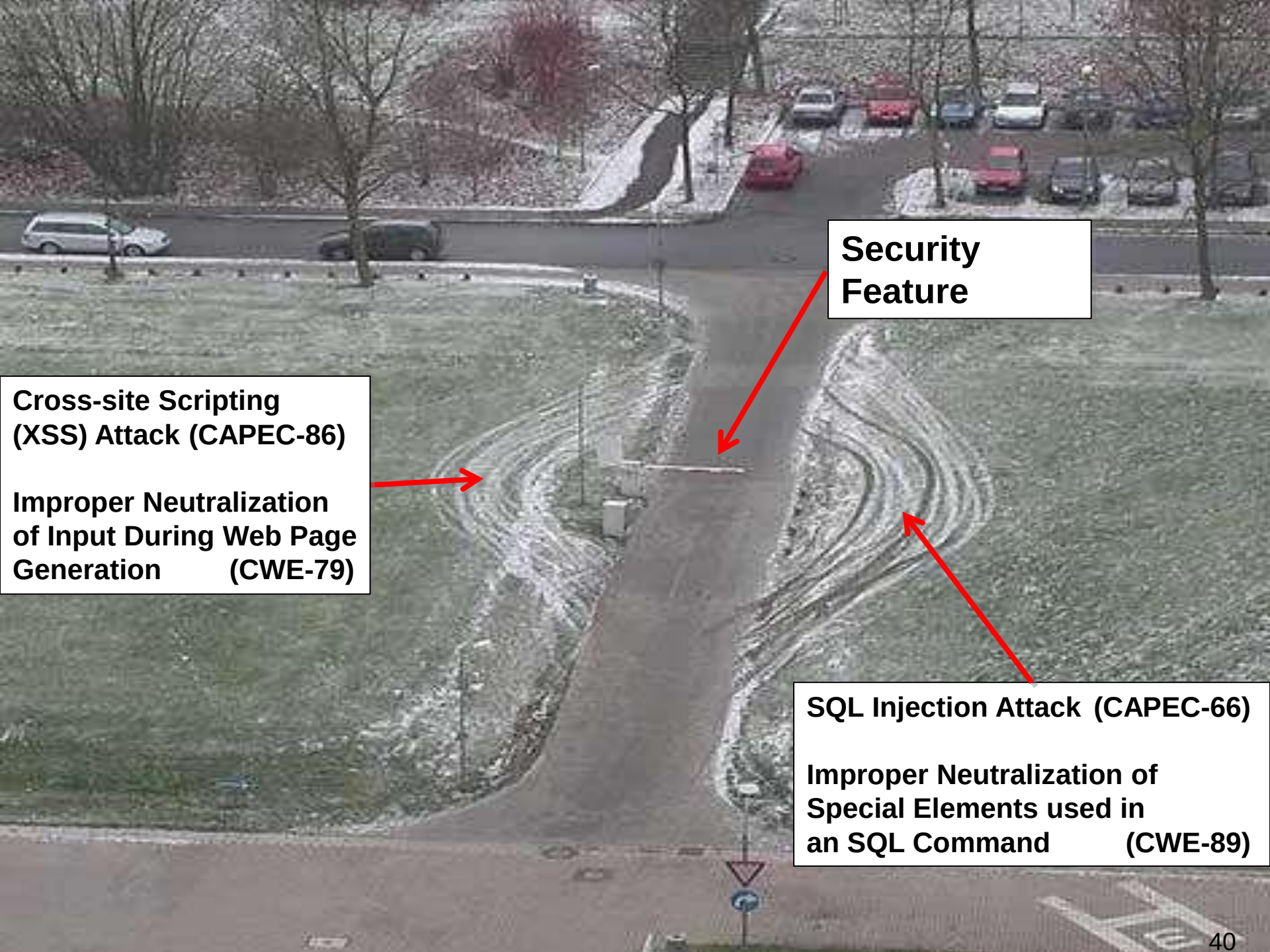


SwA and Systems Development (example)



Making Security Measurable™

* Ideally Insert SwA before RFP release in Analysis of Alternatives


Security Feature
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 Common Attack Pattern Enumeration and Classification
A Community Knowledge Resource for Building Secure Software

Home > CAPEC List > CAPEC-1000: Mechanism of Attack (Release 1.7.1) Search by ID: Go

CAPEC List
Full CAPEC Dictionary
Methods of Attack View
Reports
About CAPEC
Documents
Resources
Community
Related Activities
Collaboration List
T-Shirt
News & Events
Calendar
Free Newsletter
Compatibility
Program
Requirements
Make a Declaration
Contact Us
Search the Site

CAPEC-1000: Mechanism of Attack

Definition Graph List Slice XML.zip

Mechanism of Attack

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

View Data

View Structure: Graph

View Objective

Relationships

Nature	Type	ID	Name	Description	V
HasMember		118	Data Leakage Attacks		1000
HasMember		119	Resource Depletion		1000
HasMember		152	Injection (Injecting Control Plane content through the Data Plane)		1000
HasMember		156	Spoofing		1000
HasMember		172	Time and State Attacks		1000
HasMember		210	Abuse of Functionality		1000
HasMember		223	Probabilistic Techniques		1000
HasMember		225	Exploitation of Authentication		1000
HasMember		232	Exploitation of Privilege/Trust		1000
HasMember		255	Data Structure Attacks		1000
HasMember		262	Resource Manipulation		1000
HasMember	A	286	Network Reconnaissance		1000
HasMember	A	403	Social Engineering Attacks		1000
HasMember		436	Physical Security Attacks		1000
HasMember	A	437	Supply Chain Attacks		1000

	CAPECs in this view		Total CAPECs
Total	**412**	out of	474
Views	0	out of	6
Categories	19	out of	68
Attack Patterns	400	out of	400

BACK TO TOP

Page Last Updated: May 04, 2012

MITRE

Sharing knowledge of our opponents and watching the plays develop, we can make the saves that protect our **net**works and the software running on them.

CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
FAQs
Community
SwA On-Ramp
T-Shirt
Discussion List
Discussion Archives
Contact Us
Scoring
CWSS
CWRAF
CWE/SANS Top 25
Compatibility
Requirements
Coverage Claims Representation
Compatible Products
Make a Declaration
News
Calendar
Free Newsletter
Search the Site

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

BACK TO TOP

Section Contents
Software Assurance
Engineering for Attacks
Software Quality
Prioritizing Weaknesses
Manageable Steps
Pocket Guides
Staying Informed
Finding More Information
Other Items of Interest
Discussion List
CWE Newsletter
Terms of Use

Page Last Updated: May 13, 2012

Questions?

ramartin@mitre.org