# An Introduction to Attack Patterns as a Software Assurance Knowledge Resource

OMG Software Assurance Workshop
2007

Sean Barnum
Managing Consultant
sbarnum@cigital.com


Software Confidence. Achieved.

www.cigital.com
info@cigital.com
+1.703.404.9293

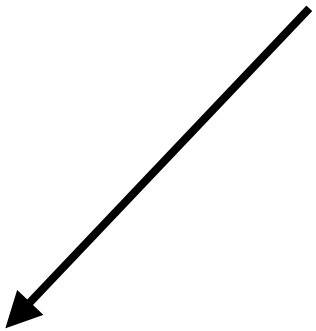
# About Cigital

- Software Quality Management consultants
- Founded in 1992 to address software security and software quality
- Recognized experts in software security and software quality
  - Widely published in books, white papers, and magazines
  - Home of Cigital Labs: cutting edge software quality research laboratory


cigital

# Evolution of Software Assurance

Defend the Perimeter and Patch when Problems are Found



Improve Assurance through Proactive Defense

Hardened Defenses through Understanding the Attacker's Perspective

cigital

# Attack Patterns

- Goal: Representing the *attacker's perspective* in a formalized and constructive way to provide *expert-level* understanding and guidance to software development personnel of all levels as to how their software is likely to be attacked, and thereby equip them to *build more secure software*

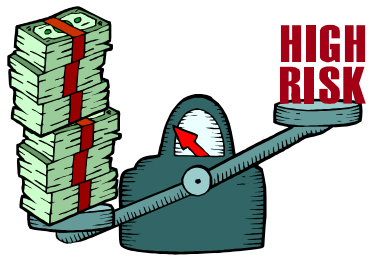
- Intended audience
  - Software development community
    - Provide knowledge to assist in building more secure software
  - Security researchers
    - Provide communication and knowledge capture mechanism for those researching exploits and other software security issues
  - Security professionals/practitioners
    - Provide knowledge to guide security assessment and auditing

cigital

# Why Should You Care About Attack Patterns?

# The Nature of Risk

- Software Assurance is an issue of **RISK**




- Defenses are constructed and strengthened to mitigate the risks of exploit of the system

- Exploring the Attacker's perspective helps to identify and qualify the nature of risk to the software

cigital

# The Long-established Principal of “Know Your Enemy”

- “One who knows the enemy and knows himself will not be endangered in a hundred engagements. One who does not know the enemy but knows himself will sometimes be victorious. Sometimes meet with defeat. One who knows neither the enemy nor himself will invariably be defeated in every engagement.”

- Chapter 3: “Planning the Attack”
  - The Art of War, Sun Tzu

cigital

# The Long-established Principal of "Know Your Enemy"

- Software Assurance Translation
  - "One who knows the enemy and knows himself will not be endangered in a hundred engagements.
  - Strong defensive preparedness combined with understanding the attacker's perspective yields high assurance
  - One who does not know the enemy but knows himself will sometimes be victorious. Sometimes meet with defeat.
  - A strong defense alone will protect you from known threats but will leave you vulnerable to others
  - One who knows neither the enemy nor himself will invariably be defeated in every engagement."
  - A lack of both a proactive defense and an understanding of the attacker's perspective leaves you completely vulnerable

- Chapter 3: "Planning the Attack"
  - The Art of War, Sun Tzu

cigital

# The Importance of Knowing Your Enemy

- An appropriate defense can only be established if you know how it will be attacked
- The challenge of the defender
  - The attacker's advantage (defender must stop all attacks; attacker need only succeed with one)
  - Prioritization of functionality over security
  - The knowledge gap between attacker's and those attempting to build secure software
- Remember!
  - Software Assurance must assume motivated attackers and not simply passive quality issues
  - Attackers are very creative, actively collaborate and have powerful tools at their disposal

cigital

# Resources for the Attacker's Perspective

- Practices and knowledge representing the attacker's perspective
  - Attack Surface Modeling
  - Threat Analysis
  - Misuse/Abuse Cases
  - Security Testing
    - Security Feature Testing
    - Risk-based Security Testing
    - Penetration Testing
    - Red Teaming
  - Attack Patterns

cigital

# Brief Introduction to the Common Weakness Enumeration (CWE)

# What Does Defense Mean?

- Minimizing vulnerabilities in software
  - Vulnerabilities are weaknesses in software that are exploitable to an attacker
  - Weaknesses typically result from coding errors, design flaws, misconfigurations or design decisions that are invalid for the given context
  - Once they reach the state of vulnerabilities, weaknesses are considerably riskier and more expensive to fix
  - Therefore, the goal of defense in software development is to minimize weaknesses in software as early in the lifecycle as possible

cigital

# How Do We Capture & Convey Weaknesses?

- There have been dozens of attempts to solve this problem in academia, government and commercial industry but they have all been disjoint
- Common Weakness Enumeration (CWE) offers a solution for today and the future
  - http://cwe.mitre.org

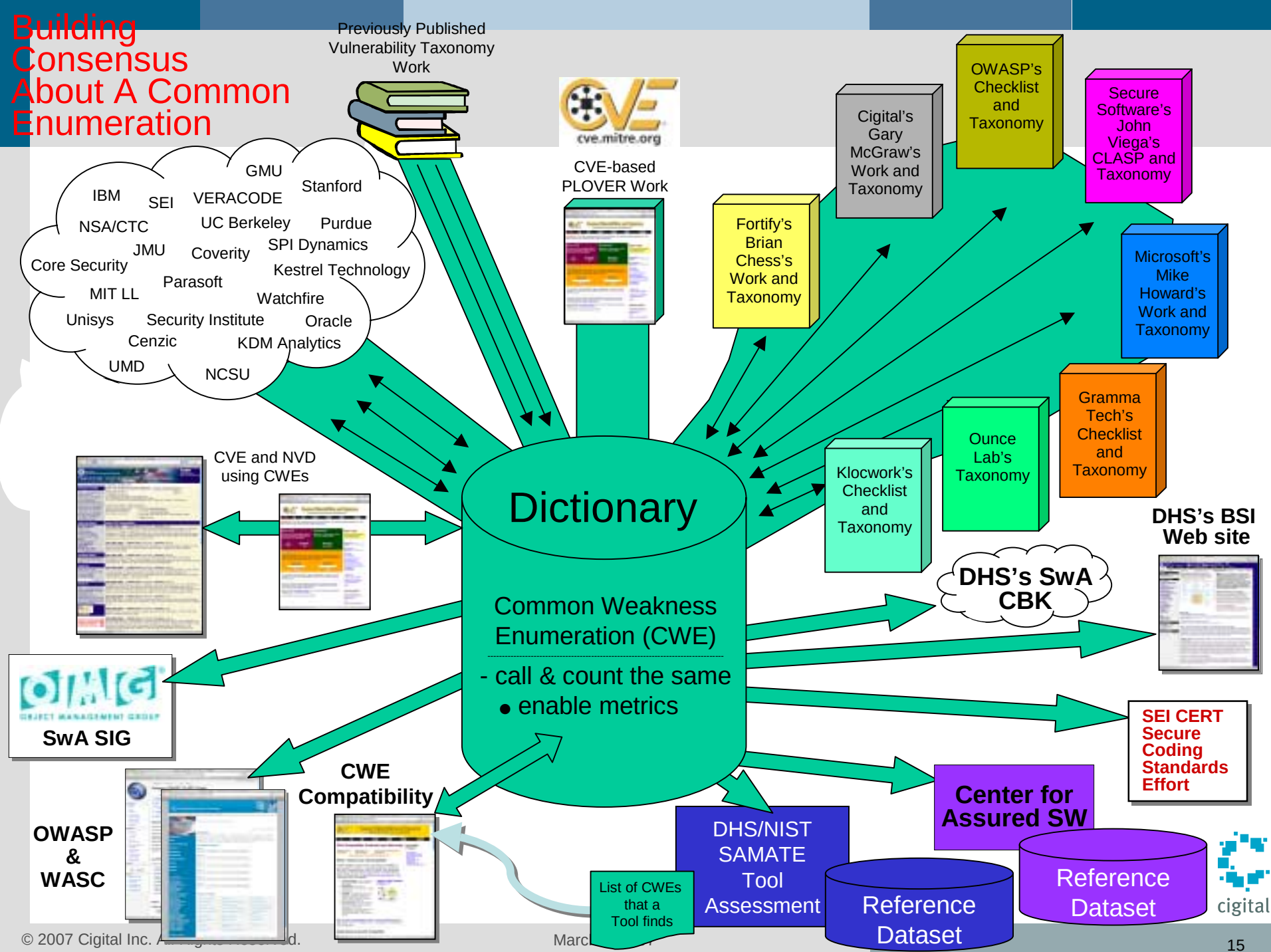
cigital

# Goal of the Common Weakness Enumeration Initiative

- To improve the quality of software with respect to known security issues within source code
  - define a unified measurable set of weaknesses
  - enable more effective discussion, description, selection and use of software security tools and services that can find these weaknesses

cigital

# Building Consensus About A Common Enumeration



# CWE Current Status

## Quality

- "Kitchen Sink" – In a good way
  - Many taxonomies, products, perspectives
  - Varying levels of abstraction
    - Directory traversal, XSS variants
- Mixes attack, behavior, feature, and flaw
  - Predominant in current research vocabulary, especially web application security
  - Complex behaviors don't have simple terms
  - New/rare weaknesses don't have terms

## Quantity

- Draft 5 - over 600 entries
- Currently integrating content from top 15 – 20 tool vendors and security weaknesses "knowledge holders" under NDA

## Accessibility

- Website is live with:
  - Historical materials, papers, alphabetical full enumeration, taxonomy HTML tree, CWE in XML, ability to URL reference individual CWEs, etc
  - http://cwe.mitre.org

cigital

# Attack Patterns Background

# What are Attack Patterns?

- An attack pattern is a blueprint for an exploit. It is a description of a common approach attackers take to attack software. They are developed by reasoning over large sets of software exploits and attacks.

- Attack patterns help identify and qualify the risk that a given exploit will occur in a software system.

cigital

# Related Concepts

- Attack/Threat trees
  - Attack patterns are paths through the tree from leaf to root
- Fault trees
  - Focused on reliability, safety and related characteristics
- Security Patterns
  - Consist of general solutions to recurring security problems (e.g. account lockout to prevent brute force attacks)

cigital

# Background

- Design Patterns
  - Christopher Alexander and then the Gang of Four (Gamma, et al)
- Attack Pattern concept emerges ~2001 among industry thought leaders
- Attack Patterns become “real” with *Exploiting Software* [Hoglund & McGraw]
  - Applying pattern concept to methods of exploit
- Attack Patterns become actionable with Common Attack Pattern Enumeration and Classification (CAPEC)

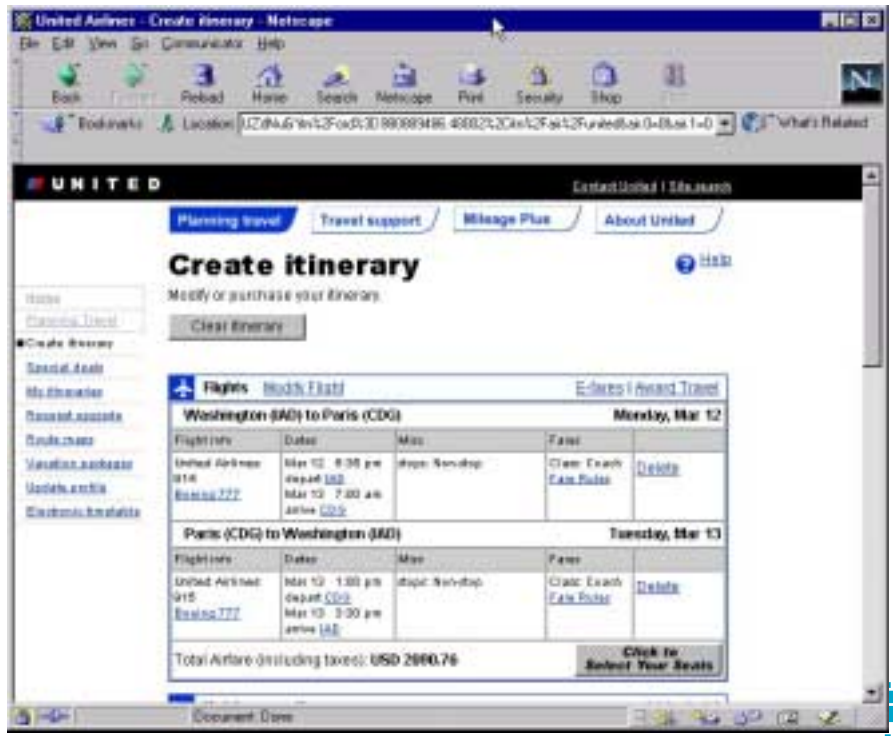
cigital

# Knowledge: 48 Attack Patterns

- Make the Client Invisible
- Target Programs That Write to Privileged OS Resources
- Use a User-Supplied Configuration File to Run Commands That Elevate Privilege
- Make Use of Configuration File Search Paths
- Direct Access to Executable Files
- Embedding Scripts within Scripts
- Leverage Executable Code in Nonexecutable Files
- Argument Injection
- Command Delimiters
- Multiple Parsers and Double Escapes
- User-Supplied Variable Passed to File System Calls
- Postfix NULL Terminator
- Postfix, Null Terminate, and Backslash
- Relative Path Traversal
- Client-Controlled Environment Variables
- User-Supplied Global Variables (DEBUG=1, PHP Globals, and So Forth)
- Session ID, Resource ID, and Blind Trust
- Analog In-Band Switching Signals (aka “Blue Boxing”)
- Attack Pattern Fragment: Manipulating Terminal Devices
- Simple Script Injection
- Embedding Script in Nonscript Elements
- XSS in HTTP Headers
- HTTP Query Strings
- User-Controlled Filename
- Passing Local Filenames to Functions That Expect a URL
- Meta-characters in E-mail Header
- File System Function Injection, Content Based
- Client-side Injection, Buffer Overflow
- Cause Web Server Misclassification
- Alternate Encoding the Leading Ghost Characters
- Using Slashes in Alternate Encoding
- Using Escaped Slashes in Alternate Encoding
- Unicode Encoding
- UTF-8 Encoding
- URL Encoding
- Alternative IP Addresses
- Slashes and URL Encoding Combined
- Web Logs
- Overflow Binary Resource File
- Overflow Variables and Tags
- Overflow Symbolic Links
- MIME Conversion
- HTTP Cookies
- Filter Failure through Buffer Overflow
- Buffer Overflow with Environment Variables
- Buffer Overflow in an API Call
- Buffer Overflow in Local Command-Line Utilities
- Parameter Expansion
- String Format Overflow in syslog()

# Attack Pattern 1: Make the client invisible

- Remove the client from the communications loop and talk directly to the server
- Leverage incorrect trust model (never trust the client)
- Example: hacking browsers that lie

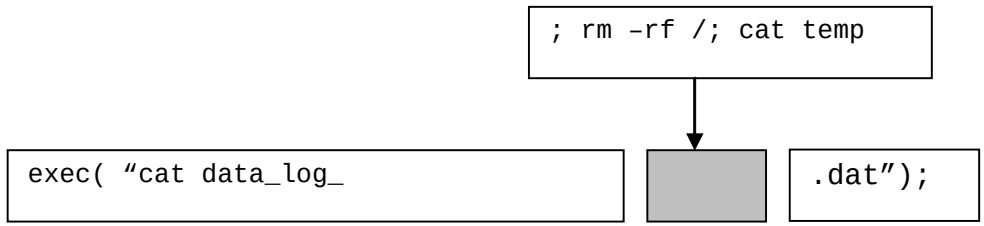



cigital

# Attack Pattern 2: Command delimiters

- Use off-nominal characters to string together multiple commands

<input type=hidden name=filebase value="bleh; [command]”>




- Example: shell command injection with delimiters

cat data_log_; rm -rf /; cat temp.dat

cigital

# Attack Pattern Generation

# Who Authors Attack Patterns?

- Most developers typically lack the experiential depth to perform attack abstraction analysis
- More suitable to a narrower membership of security analysts and researchers
- Conclusion:
  - They are created by a small group of very experienced people
  - They are used by a very large group of experienced and inexperienced software development personnel

cigital

# Where They Come From

- Input source – Exploits
  - Not many good official sources for Exploits – Lots of shady sources
  - POC exploits sometimes available with vulnerability reports
  - Results from malware analysis community are often for limited distribution
- Input source – Attacks
  - Primarily come from operations and incident response communities
  - Some come from researchers
- Analysis Approach
  - Batch vs Continual
  - Formal vs Informal

cigital

# Exploit Analysis Process

- Analyze the exploit
  - Reverse engineer it
  - Perform forensic analysis
  - Analyze any available patches by vendors of the target software
- Determine whether the exploit is an instantiation of any existing attack patterns
  - If so, add new exploit reference to existing attack pattern and stop there
  - If not, determine if this represents a new common attack approach
    - If so, continue with attack pattern generation
    - If not, archive exploit analysis performed and stop there
- Identify targeted vulnerability or weakness
  - If vulnerability, find related CVE, OVAL, weakness and context descriptions
- Define contextual prerequisites for attack
  - In what technical context (OS, platform, language, etc.) and under what conditions is this exploit possible?

cigital

# Exploit Analysis Process (continued)

- Determine the method of attack
  - Malicious data entry?
  - Maliciously crafted file?
  - Protocol corruption?
- Determine required attacker's skill
  - Script kiddie?
  - Experienced hacker?
- Determine required attacker's resources
  - Simple manual execution?
  - Distributed bot army?
  - Well-funded organization?
  - Tools?
- Determine motivation of attacker
  - Gain access to secure assets (information, CPU cycles, etc.)?
  - Denial of capability?
  - Vandalism or pure destructive intent?

cigital

# Adorning the Attack Pattern

- It is often useful to adorn the attack pattern with useful reference information
  - Source exploits
  - Targeted vulnerabilities including CVE & OVAL references
  - Targeted weaknesses including CWE references
  - Relevant security requirements
  - Relevant design patterns
  - Related attack patterns

cigital

# Evaluating and Verifying Attack Patterns

- Validate with a 3rd party review
- Verify that no existing attack pattern covers the exploits
  - If existing attack pattern found, determine if new one is needed or if existing one should be modified
- Validate that source exploits are actually instantiations of new attack pattern
  - If not, should attack pattern be modified
- Ensure attack pattern is not overly generic
- Ensure attack pattern is not overly specific
- Ensure attack pattern is accessible to target audiences

cigital

# Formally Representing Attack Patterns

# Drivers for Formal Representation

- Consistency between patterns & authors
- Ensure adequate completeness and quality
- Correlate and integrate with other relevant knowledge collections
- Ability for reader to focus on aspects they care about
- Ability for variations in content presentation
- Ability to search and subsect a set of patterns for given contexts

cigital

# A Proposed Attack Pattern Schema

- Primary Schema Elements
  - Identifying Information
    - Attack Pattern ID
    - Attack Pattern Name
  - Describing Information
    - Description
    - Related Weaknesses
    - Related Vulnerabilities
    - Method of Attack
    - Examples-Instances
    - References
  - Prescribing Information
    - Solutions and Mitigations
  - Scoping and Delimiting Information
    - Typical Severity
    - Typical Likelihood of Exploit
    - Attack Prerequisites
    - Attacker Skill or Knowledge Required
    - Resources Required
    - Attack Motivation-Consequences
    - Context Description

cigital

# A Proposed Attack Pattern Schema

- Supporting Schema Elements
  - Describing Information
    - Injection Vector
    - Payload
    - Activation Zone
    - Payload Activation Impact
  - Diagnosing Information
    - Probing Techniques
    - Indicators-Warnings of Attack
    - Obfuscation Techniques
  - Enhancing Information
    - Related Attack Patterns
    - Relevant Security Requirements
    - Relevant Design Patterns
    - Relevant Security Patterns
    - Related Security Principles
    - Related Guidelines

cigital

# Attack Patterns Example (part 1)

| | |
|---|---|
| **Name** | HTTP Response Splitting |
| **Attack_Pattern_ID** | |
| **Severity** | High |
| **Description** | HTTP Response Splitting causes a vulnerable web server to respond to a maliciously crafted request by sending an HTTP response stream such that it gets interpreted as two separate responses instead of a single one. This is possible when user-controlled input is used unvalidated as part of the response headers. An attacker can have the victim interpret the injected header as being a response to a second dummy request, thereby causing the crafted contents be displayed and possibly cached. To achieve HTTP Response Splitting on a vulnerable web server, the attacker:<br>1. Identifies the user-controllable input that causes arbitrary HTTP header injection.<br>2. Crafts a malicious input consisting of data to terminate the original response and start a second response with headers controlled by the attacker.<br>3. Causes the victim to send two requests to the server. The first request consists of maliciously crafted input to be used as part of HTTP response headers and the second is a dummy request so that the victim interprets the split response as belonging to the second request. |
| **Attack_Prerequisites** | User-controlled input used as part of HTTP header<br><br>Ability of attacker to inject custom strings in HTTP header<br><br>Insufficient input validation in application to check for input sanity before using it as part of response header |
| **Likelihood of Exploit** | Medium |
| **Methods of Attack** | Injection<br><br>Protocol Manipulation |
| **Examples-Instances** | In the PHP 5 session extension mechanism, a user-supplied session ID is sent back to the user within the Set-Cookie HTTP header. Since the contents of the user-supplied session ID are not validated, it is possible to inject arbitrary HTTP headers into the response body. This immediately enables HTTP Response Splitting by simply terminating the HTTP response header from within the session ID used in the Set-Cookie directive.<br><br>CVE-2006-0207 |
| **Attacker_Skill_or_Knowledge_Required** | High - The attacker needs to have a solid understanding of the HTTP protocol and HTTP headers and must be able to craft and inject requests to elicit the split responses. |
| **Resources_Required** | None |
| **Probing_Techniques** | With available source code, the attacker can see whether user input is validated or not before being used as part of output. This can also be achieved with static code analysis tools<br><br>If source code is not available, the attacker can try injecting a CR-LF sequence (usually encoded as %0d%0a in the input) and use a proxy such as Paros to observe the response. If the resulting injection causes an invalid request, the web server may also indicate the protocol error. |
| **Indicators-Warnings_of_Attack** | The only indicators are multiple responses to a single request in the web logs. However, this is difficult to notice in the absence of an application filter proxy or a log analyzer. There are no indicators for the client |
| **Solutions_and_Mitigations** | To avoid HTTP Response Splitting, the application must not rely on user-controllable input to form part of its |

# Attack Patterns Example (part 2)

| | |
|---|---|
| **Solutions_and_Mitigations** | To avoid HTTP Response Splitting, the application must not rely on user-controllable input to form part of its output response stream. Specifically, response splitting occurs due to injection of CR-LF sequences and additional headers. All data arriving from the user and being used as part of HTTP response headers must be subjected to strict validation that performs simple character-based as well as semantic filtering to strip it of malicious character sequences and headers. |
| **Attack Motivation-Consequences** | Run Arbitrary Code<br>Privilege Escalation |
| **Context Description** | HTTP Response Splitting attacks take place where the server script embeds user-controllable data in HTTP response headers. This typically happens when the script embeds such data in the redirection URL of a redirection response (HTTP status code 3xx), or when the script embeds usuch data in a cookie value or name when the response sets a cookie. In the first case, the redirection URL is part of the Location HTTP response header, and in the cookie setting, the cookie name/value pair is part of the Set-Cookie HTTP response header. |
| **Injection_Vector** | User-controllable input that forms part of output HTTP response headers |
| **Payload** | Encoded HTTP header and data separated by appropriate CR-LF sequences. The injected data must consist of legitimate and well-formed HTTP headers as well as required script to be included as HTML body. |
| **Activation_Zone** | API calls in the application that set output response headers. |
| **Payload_Activation_Impact** | The impact of payload activation is that two distinct HTTP responses are issued to the target, which interprets the first as response to a supposedly valid request and the second, which causes the actual attack, to be a response to a second dummy request issued by the attacker. |
| **Related Weaknesses** | CWE113 “HTTP Response Splitting” - Targeted<br>CWE74 “Injection” - Secondary |
| **Relevant_Security_Requirements** | All client-supplied input must be validated through filtering and all output must be properly escaped. |
| **Related Security Principles** | Reluctance to Trust |
| **Related_Guidelines** | Never trust user-supplied input. |
| **Related_Coding_Rules** | |
| **References** | G. Hoglund and G. McGraw. Exploiting Software: How to Break Code. Addison-Wesley, February 2004.<br>CWE - HTTP Response Splitting<br>CWE - Injection |
| **Submission Source** | Chiradeep B Chhaya2007-01-09First Draft |
| **Modification Source** | |

cigital

# Leveraging Attack Patterns

# Where They Are Leveraged

- By representing the attacker's perspective, attack patterns offer valuable knowledge, either *proscriptive by example* or *prescriptive by advice*, at **every stage of the software development lifecycle (SDLC)**

- Depending on the level of detail describing the attack pattern and the level of abstraction of the attack, any given attack pattern can have varying levels of usefulness at different stages of the SDLC

cigital

# Selecting Appropriate Attack Patterns for the Context

- The first step in leveraging attack patterns anywhere in the SDLC is identifying which patterns are appropriate for the business, technical and security context as well as the development activity being undertaken
- Identify the set of attack patterns that pose the most significant risk

cigital

# Leveraging Attack Patterns Across the SDLC

- Security Policy
- Requirements
- Architecture & Design
- Implementation
- Test
- Operations

cigital

# Where They Are Leveraged – Security Policy

- Attack Patterns can be an invaluable resource in guiding the **selection and definition of relevant security policies and standards**

- Generating security policies and standards
  - Development perspective
    - Using relevant attack patterns to identify appropriate security policies and standards to obviate or mitigate the attacks
  - Security Assurance perspective
    - Using relevant attack patterns to identify appropriate guidelines and context for verifying compliance with appropriate security policies

cigital

# Security Policy Example (simplistic)

- Relevant Attack Patterns
  - Password Brute Forcing
    - Try Common (default) Usernames and Passwords
    - Dictionary-based password attacks
- Resulting Security Policy
  - All systems must incorporate an account lockout mechanism to block account access for a system-specific period of time after a system-specific number of failed login attempts

cigital

# Leveraging Attack Patterns Across the SDLC

- Security Policy
- Requirements
- Architecture & Design
- Implementation
- Test
- Operations

cigital

# Where They Are Leveraged – Requirements

- Attack Patterns can be an invaluable resource in assisting to **define the system's behavior to prevent or react to a specific type of likely attack**

- Defining requirements
  - Development perspective
    - Using relevant attack patterns to identify appropriate ***positive security feature requirements*** to describe ***functionality*** that will be resistant and resilient to the specified attack
  - Security Assurance perspective
    - Using relevant attack patterns to identify appropriate ***negative security requirements (misuse/abuse cases)*** to specify the software's behavior when faced with the specified attack

cigital

# Resource: Security Feature Requirements

- Objective
  - Explicitly describe the presence and expected behavior of security-related functionality and features of the software
- Role of Attack Patterns
  - Content contained in each attack pattern, such as Attack Prerequisites and Related Weaknesses can help identify missing security functionality that could enable such an attack. This functionality can then be explicitly included
  - The Relevant Security Requirements element of some attack patterns can explicitly list recommended security requirements to mitigate that class of attack

cigital

# Resource: Security Requirements Example (simplistic)

- Relevant Attack Pattern
  - Session Fixation
- Identified Security Requirements
  - Regenerate session identifiers upon each new request. This ensures that fixated session identifiers are rendered obsolete.
  - Regenerate a session identifier every time a user enters an authenticated session and destroy the identifier when the user logs out of an authenticated session.
  - Set appropriate expiry times on cookies that contain session identifiers. This helps limit the window of opportunity for an attacker to use the identifier.
  - Do not use session identifiers as part of URLs or hidden form fields. It becomes easy for an attacker to trick a user into a fixated session when session identifiers are easily accessible.

cigital

# Resource: Use/Abuse/Misuse Cases

- Use Cases – “organized collections of scenarios based on the sequences of actions taken by normal users” – just stories about how people use the system
- Abuse Cases – a specialized form of Use Cases that focus on the exceptions and threats caused by hostile agents.
- Misuse Cases – a specialized form of a Use Case that focuses on the behavior of a system when it is used in an unexpected way by other than hostile agents.

*Simply* – Use cases look at the system from the normal users perspective; Abuse cases look at the system from the attackers’ perspective; misuse cases look at the system from the perspective of a naieve user.

**An abuse case or misuse case “threatens” a use case**

**A use case “mitigates” an abuse case or misuse case.**

cigital

# Resource: Misuse/Abuse Cases

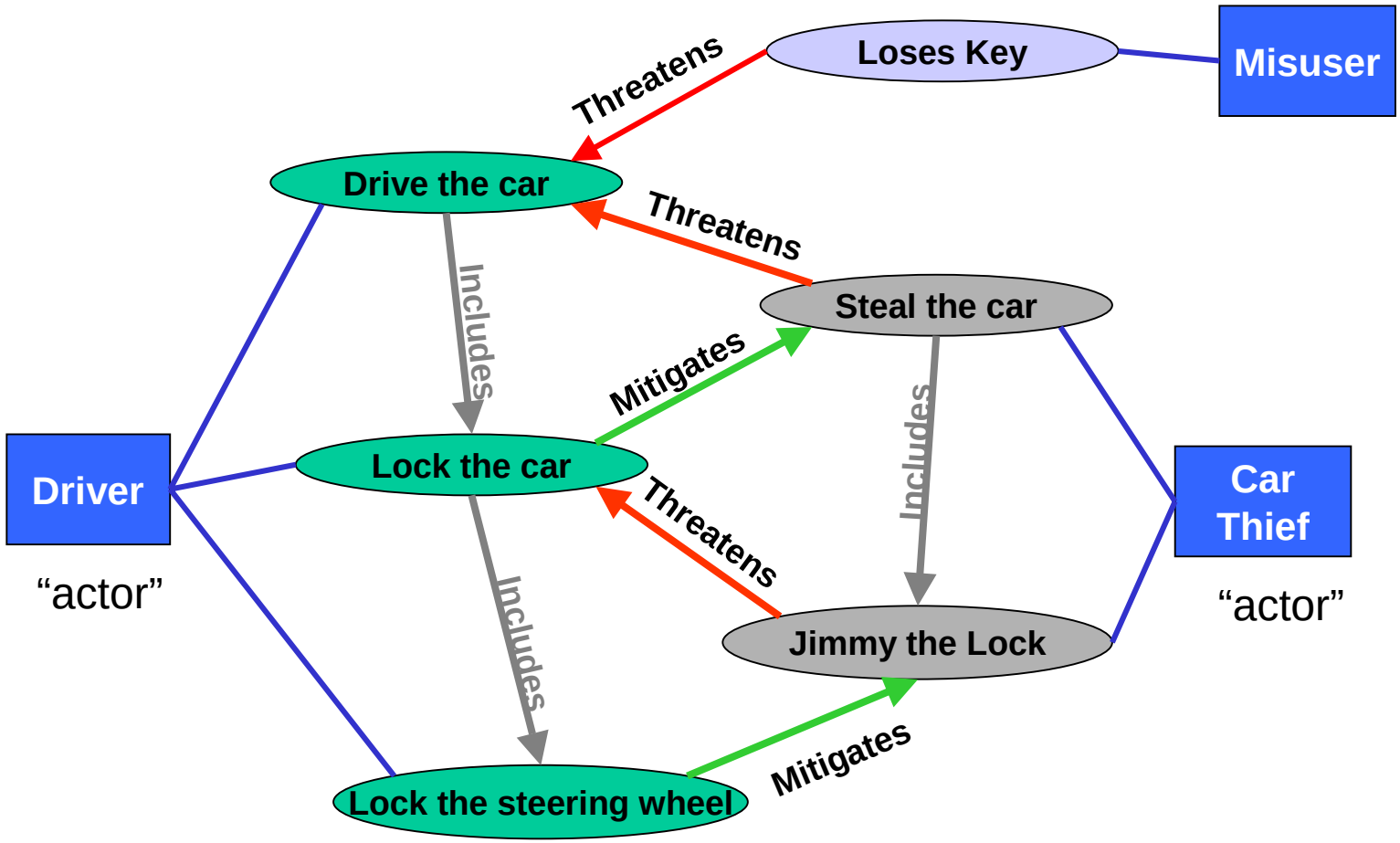
- Objective
  - Capture and personify attacking behaviors against the system as requirements for attack resistance
- Key Factors
  - Use cases formalize normative behavior (and assume correct usage)
  - Describing non-normative behavior is a good idea
    - Prepare for abnormal behavior (attack)
    - Misuse or abuse cases do this
    - Uncover exceptional cases
  - Leverage the fact that designers know more about their system than potential attackers do
  - Document explicitly what the software will do in the face of illegitimate use
  - Form basis for security testing of attack resistance
  - Consist of typical use case fields
  - Relationships with Use Cases
  - Efficacy Targets
    - Resistance
    - Recovery
- Role of Attack Patterns
  - Misuse and Abuse Cases can be directly derived from attack pattern descriptions

cigital

# Resource: Misuse/Abuse Cases Example (simplistic)

Use Case: left to right

Misuse and abuse case: right to left



# Leveraging Attack Patterns Across the SDLC

- Security Policy
- Requirements
- Architecture & Design
- Implementation
- Test
- Operations

cigital

# Where They Are Leveraged – Architecture and Design

- Attack Patterns can be an invaluable resource in assisting a **software architecture team to create secure designs**

- Architecture and design
  - Development perspective
    - Using relevant attack patterns as ***negative scenarios for a proposed architecture and design to deal with***
  - Security Assurance perspective
    - Using relevant attack patterns to ***put flesh to threat modeling as part of architectural risk analysis***
    - Using relevant attack patterns to ***identify appropriate recommended or non-recommended design patterns***

cigital

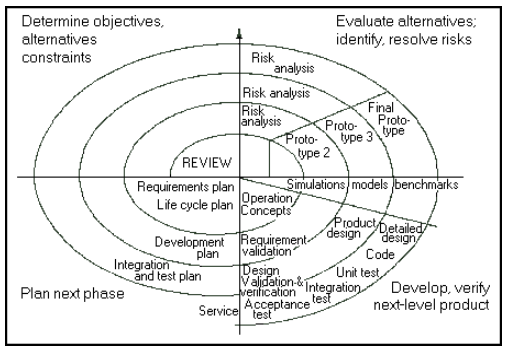
# Architecture and Design Development Example (simplistic)

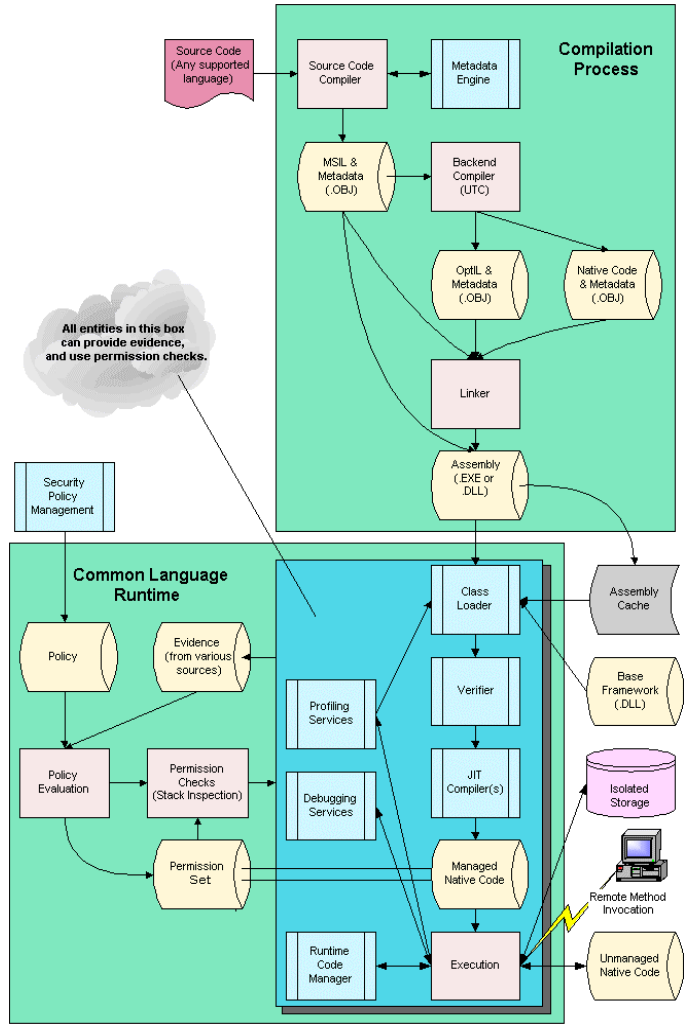
- Relevant Attack Patterns
  - Exploiting Trust in Client
    - Man-in-the-Middle
    - Create Malicious Client
    - Client-Server Protocol Manipulation
- Resulting Architecture & Design Decision
  - Place all user authentication and input validation on the server leaving a minimal user interface on the client

cigital

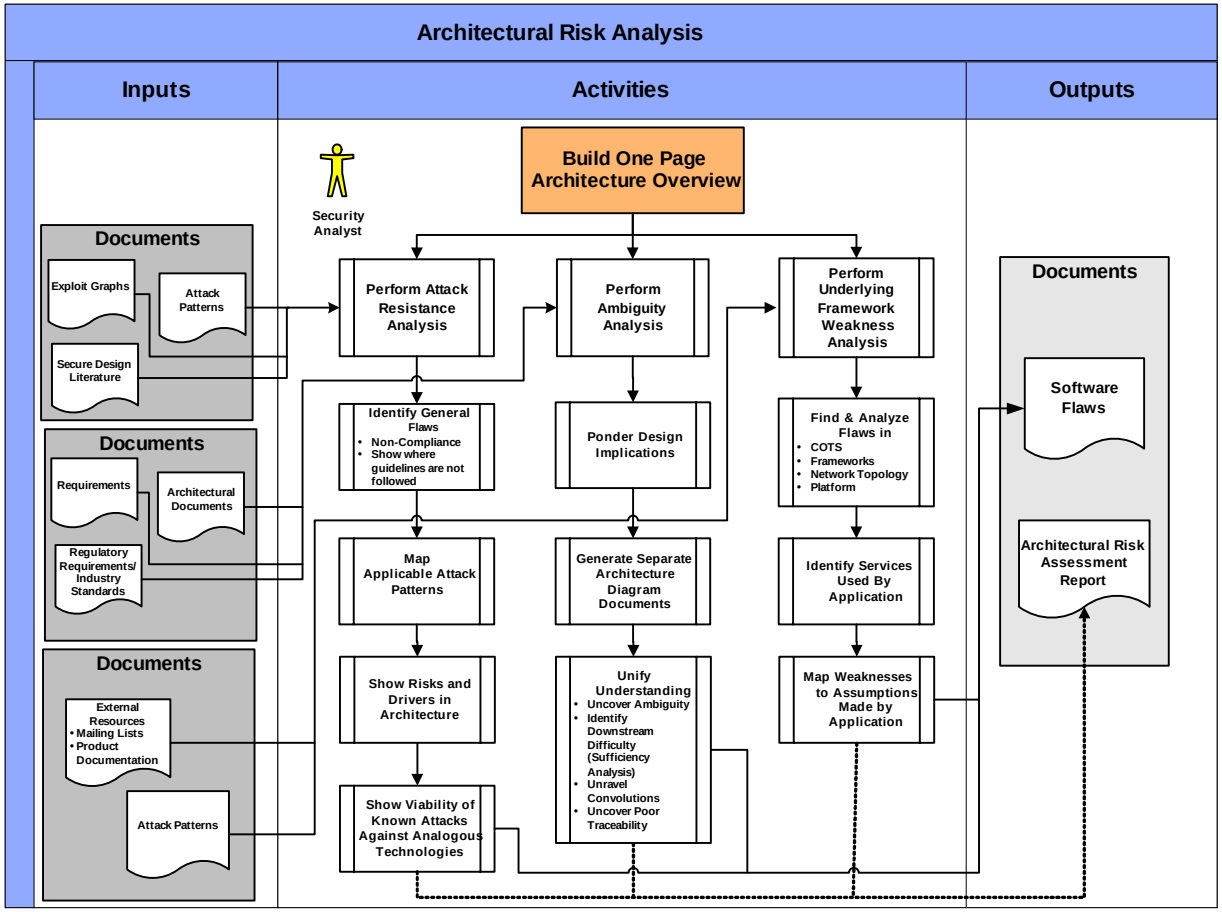
# A&D Practice: Architectural Risk Analysis

- Designers should not do this
- Build a one page white board design model (like that →)
- Use hypothesis testing to categorize risks
  - Threat modeling/Attack patterns
- Rank risks
- Tie to business context
- Suggest fixes
- Repeat



# A&D Practice: Architectural Risk Analysis




- Start by building a one page overview of your system
- Then apply the following three step process
  - Weakness analysis
  - Ambiguity analysis
  - Attack resistance analysis


cigital

# A&D Practice: Attack Surface Modeling

- Objective
  - Identify in somewhat objective terms how vulnerable a software system is to attack (characterize defensive posture)
- Key Factors
  - Entry/Exit Points
  - Amount of Code Running
  - Trust Boundaries
  - Assets
  - Vulnerabilities
  - Barriers/Challenges to Attack (difficulty to exploit)

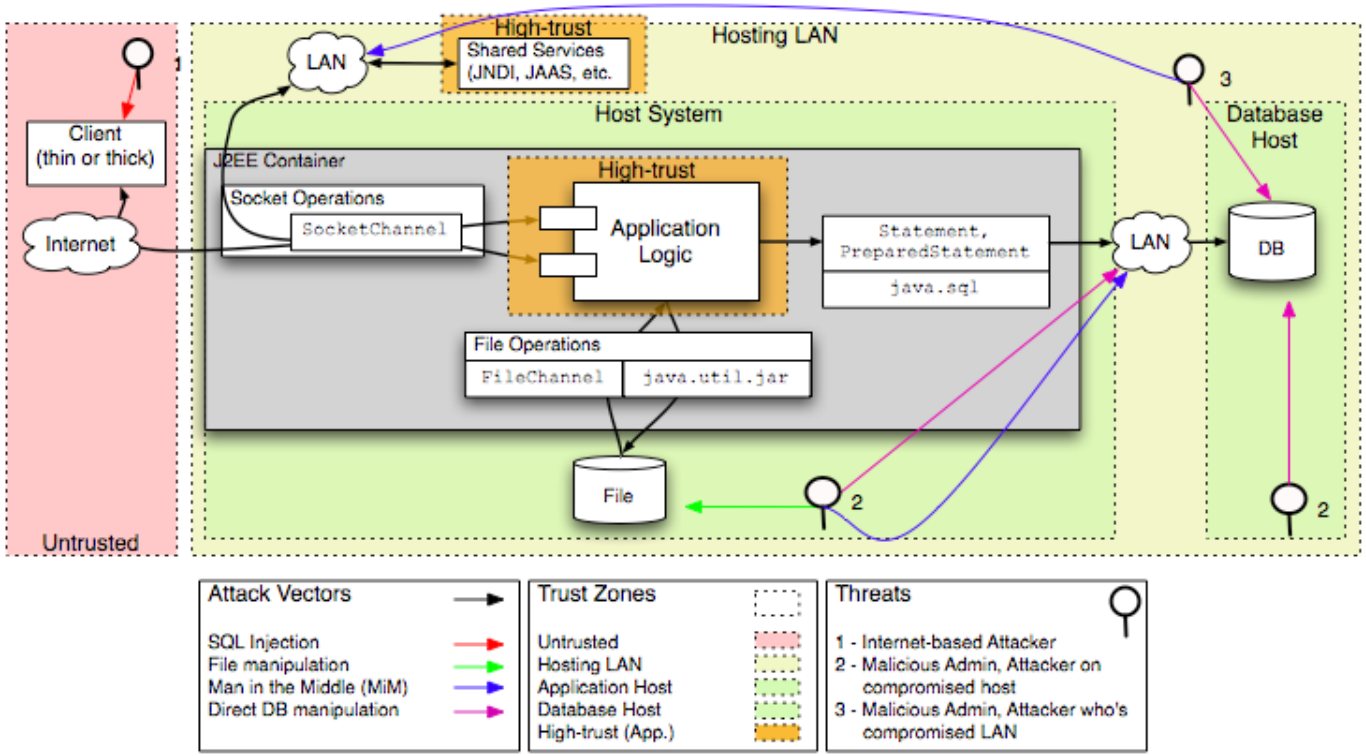
cigital

# A&D Practice: Threat Analysis

- Objective
  - To identify and understand the active threats that exist for a software system that induce assurance risk
- Key Factors
  - Actor Identification
  - Motivation
  - Capability
  - Access Vector against Attack Surface

cigital

# Threat/Attack Modeling Diagrams




- Diagram system
- List Threats (agents of maligned intent)
- Show attack vectors

# Leveraging Attack Patterns Across the SDLC

- Security Policy
- Requirements
- Architecture & Design
- Implementation
- Test
- Operations

cigital

# Where They Are Leveraged – Implementation

- Attack Patterns can be an invaluable resource in guiding **secure code implementation practices** through **prioritizing and avoiding specific weaknesses in the code**

- Implementation
  - Development perspective
    - Using relevant attack patterns as a mechanism to ***identify relevant weaknesses to avoid***
  - Security Assurance perspective
    - Using relevant attack patterns as a mechanism to ***identify relevant weaknesses to scan for (using software security tools where possible) and confirm their absence***

cigital

# Implementation Practice: Secure Coding

- Description
  - Writing software code in a manner that fulfills all expectations of behavior (what it should do and what it should not do) and minimizes the presence of common weaknesses which may lead to vulnerabilities
    - Understand common coding errors that lead to weaknesses
    - For a given implementation context, identify which weaknesses bring the highest risk
    - Provide training to developers in the understanding of common coding errors (especially high-risk errors) and the recommended secure coding practices to mitigate them
- Role of Attack Patterns
  - Relevant attack patterns help identify the high-risk weaknesses for a given implementation context

cigital

# Implementation Practice: Secure Coding Example (simplistic)

- Relevant Attack Pattern
  - HTTP Cookies
- Relevant High-Priority Weaknesses Identified through Attack Pattern
  - CWE-302 - Authentication Bypass by Assumed-Immutable Data
  - CWE-113 – HTTP Response Splitting
  - CWE-539 – Information Leakage Through Persistent Cookies
  - CWE-315 – Plaintext Storage in Cookies
  - CWE-384 – Session Fixation
  - CWE-565 – Use of Cookies
  - CWE-472 – Web Parameter Tampering
  - CWE-20 – Input Validation

cigital

# Implementation Practice: Secure Code Review

- Description
  - Performing analysis of software code to verify the absence of common weaknesses which may lead to vulnerabilities
    - Identify and prioritize weaknesses to be targeted
    - Review code to gain assurance that specific weaknesses do not exist
      - Most effective and efficient when done with tools
    - Mitigate and/or remediate identified issues
    - Provide demonstrable evidence of what activities were performed, what was found, what was fixed and what risk was accepted
- Role of Attack Patterns
  - Relevant attack patterns help identify the high-risk weaknesses they target

cigital

# Leveraging Attack Patterns Across the SDLC

- Security Policy
- Requirements
- Architecture & Design
- Implementation
- Test
- Operations

cigital

# Where They Are Leveraged – Test

- Attack Patterns can be an invaluable resource in guiding **software security testing in a practical and realistic context**

- Test
  - Development perspective
    - Using relevant attack patterns to ***identify necessary test cases for confirming the absence of relevant weaknesses*** as well as ***giving a practical context for testing security features***
  - Security Assurance perspective
    - Using relevant attack patterns to ***define appropriate roles and approaches for red team testing***

cigital

# Test Practice: Security Feature Testing

- Description
  - Performing traditional functional and non-functional testing of the security features of the software to assure their presence and correct behavior
    - E.g. testing an account lockout feature after multiple failed login attempts
- Role of Attack Patterns
  - Give a realistic bounding context for definition of test cases

cigital

# Test Practice: Risk-based Security Testing

- Description
  - Testing focused on reducing the risk profile of the software. In this case, testing to confirm the absence of targeted high-risk weaknesses and the correct behavior of the software in the face of non-normative user behavior
- Role of Attack Patterns
  - Identify high-priority test cases to confirm the absence of high-risk weaknesses targeted by relevant attack patterns
  - Form the templates for creation of Abuse Case and Misuse Case-driven test cases

cigital

# Test Practice: Penetration Testing (blackbox)

- Description
  - Testing the attack resistance of software by emulating an attacker executing a checklist of simple attack methods without any prior knowledge of the target infrastructure
    - Typically focuses on simply penetrating the outer barrier of the software and does not involve chaining of attacks
- Role of Attack Patterns
  - Specific attack pattern steps can assist in identification of penetration methods to add to checklist

cigital

# Test Practice: Red Teaming

- Description
  - Active testing of system attack resistance through emulation of a specific attacker profile
  - Team of testers creatively attack the system as an identified attacker/threat might
  - Red Teaming is a more involved and creative form of penetration testing
    - Penetration testing typically focuses on simply breaching the barrier security of the software where red teaming probes the full scope of the software as an attacker would
    - Red teaming emulates the creativity of the attacker where penetration testing is often a rote execution through a checklist of common attacks
- Role of Attack Patterns
  - Relevant attack patterns can help identify appropriate attack profiles for the Red Team to assume including typical methods

cigital

# Leveraging Attack Patterns Across the SDLC

- Security Policy
- Requirements
- Architecture & Design
- Implementation
- Test
- Operations

cigital

# Where They Are Leveraged – Operations

- Attack Patterns can be an invaluable resource in **securely operating a deployed system**

- Operations
  - Operating perspective
    - Using relevant attack patterns to ***identify appropriate secure operations configurations***
    - Using relevant attack patterns to ***classify and understand impact of observed attacks***
  - Security Assurance perspective
    - Operational knowledge of security issues can be leveraged to ***feed the attack pattern generation process*** and ***yield better attack pattern coverage and thereby better future software***

cigital

# Operations Practice: Improve Process with Real-world Lessons Learned

- Description
  - Pursuing continuous improvement by informing early lifecycle processes of lessons learned in late lifecycle processes in order to avoid such problems in the future
    - Capture real-world problems faced by operational software
    - Abstract this detailed information into knowledge that developers can understand
    - Leverage it to improve development processes and avoid such problems in the future
- Role of Attack Patterns
  - Provide the mechanism for capturing the abstracted knowledge and making it actionable in the SDLC

cigital

# Common Attack Pattern Enumeration and Classification (CAPEC)

# What is CAPEC?

- Effort targeted at:
  - Standardizing the capture and description of attack patterns
  - Collecting known attack patterns into an integrated enumeration that can be consistently and effectively leveraged by the community
  - Classifying attack patterns such that users can easily identify the subset of the entire enumeration that is appropriate for their context
- Funded by the DHS NCSD
- Led by Cigital

cigital

# Current CAPEC Status

- Extensive research performed and underway to identify and evaluate potential resources for creating attack patterns
- Schema definition completed (discussed earlier)
- In process of fleshing out and authoring ~100 patterns
- Draft attack taxonomy completed from analysis of existing taxonomies and identified patterns

cigital

# Draft Attack Taxonomy

- Organized by mechanism of attack
  - Abuse of Functionality
  - Spoofing
  - Probabilistic Techniques
  - Exploitation of Authentication
  - Resource Depletion
  - Exploitation of Privilege/Trust
  - Injection
  - Data Structure Attacks
  - Data Leakage Attacks
  - Resource Manipulation
  - Protocol Manipulation
  - Time & State Attacks

cigital

# Draft Attack Taxonomy: Spoofing subtree example

- Spoofing
  - Content Spoofing
    - Make Use of Configuration File Search Paths
    - Fake the Source of Data
    - Checksum Spoofing
    - Spoofing of UDDI/ebXML Messages
  - Identity Spoofing (Impersonation)
    - Principal Spoofing
    - Man-in-the-Middle
      - Utilize Rest's trust in the system resource to register man in the middle
    - Create Malicious Client
    - Client-Server Protocol Manipulation
      - Reflection Attack in an Authentication Protocol
    - XML Routing Detour Attacks
    - External Entity Attack
    - Phishing
      - Spear Phishing
      - Mobile Phishing (aka MobPhishing)
    - Pharming

cigital

# Draft Attack Taxonomy (snippet)

- Session Fixation
- Session Riding (aka Cross-site Request Forgery)

Resource Depletion

- Denial of Service through Resource Depletion
- Resource Depletion through Flooding
- Resource Depletion through Allocation
- Resource Depletion through Leak
- XML Parser Attack

Exploitation of Privilege/Trust

- Privilege Escalation
  - Direct Access to Executable Files
  - Use a User-Supplied Configuration File to Run Commands That Elevate Privilege
- Hijacking a privileged thread of execution
  - Implementing a callback to system routine (old AWT Queue)
  - Catching exception throw/signal from privileged block
- Subverting code-signing/identity facilities to gain their privilege
  - Calling signed code from another language within a sandbox that allows this
  - Lifting signing key and signing malicious code from a production environment
  - Using URL/codebase / G.A.C. (code source) to convince sandbox of privilege
- Target Programs That Write to Privileged OS Resources
- Exploiting Trust in Client
  - Man-in-the-Middle
  - Create Malicious Client
  - Client-Server Protocol Manipulation
    - Reflection Attack in an Authentication Protocol
  - Lifting Sensitive Data from the Client
    - Lifting data embedded in client distributions (thick or thin)
      - Lifting credential(s)/key material embedded in client distributions (thick or thin)
      - Lifting cached, sensitive data embedded in client distributions (thick or thin)
  - Removing Important Functionality from the Client
    - Removing/short-circuiting 'guard logic'
      - Removing/short-circuiting 'Purse' logic: removing/mutating 'cash' "decrements"
      - Removal of filters: Input filters, output filters, data masking
      - Subversion of authorization checks: cache filtering, programmatic security, etc.
- Exploitation of Authorization
  - Mapping a path to and accessing functionality not properly constrained by authorization framework/ACLs

Injecting Control Plane content through the Data Plane (AKA Injection)

- Analog In-Band Switching Signals (aka “Blue Boxing”)
- Parameter Injection
  - Argument Injection
  - User-Supplied Variable Passed to File System Calls
- Resource Injection

# Adorning Metadata

- Purpose
  - Reconnaissance
  - Penetration
  - Exploitation
- CIA Impact
  - Confidentiality Impact
  - Integrity Impact
  - Availability Impact
- Technical Context
  - Paradigm
  - Framework
  - Platform

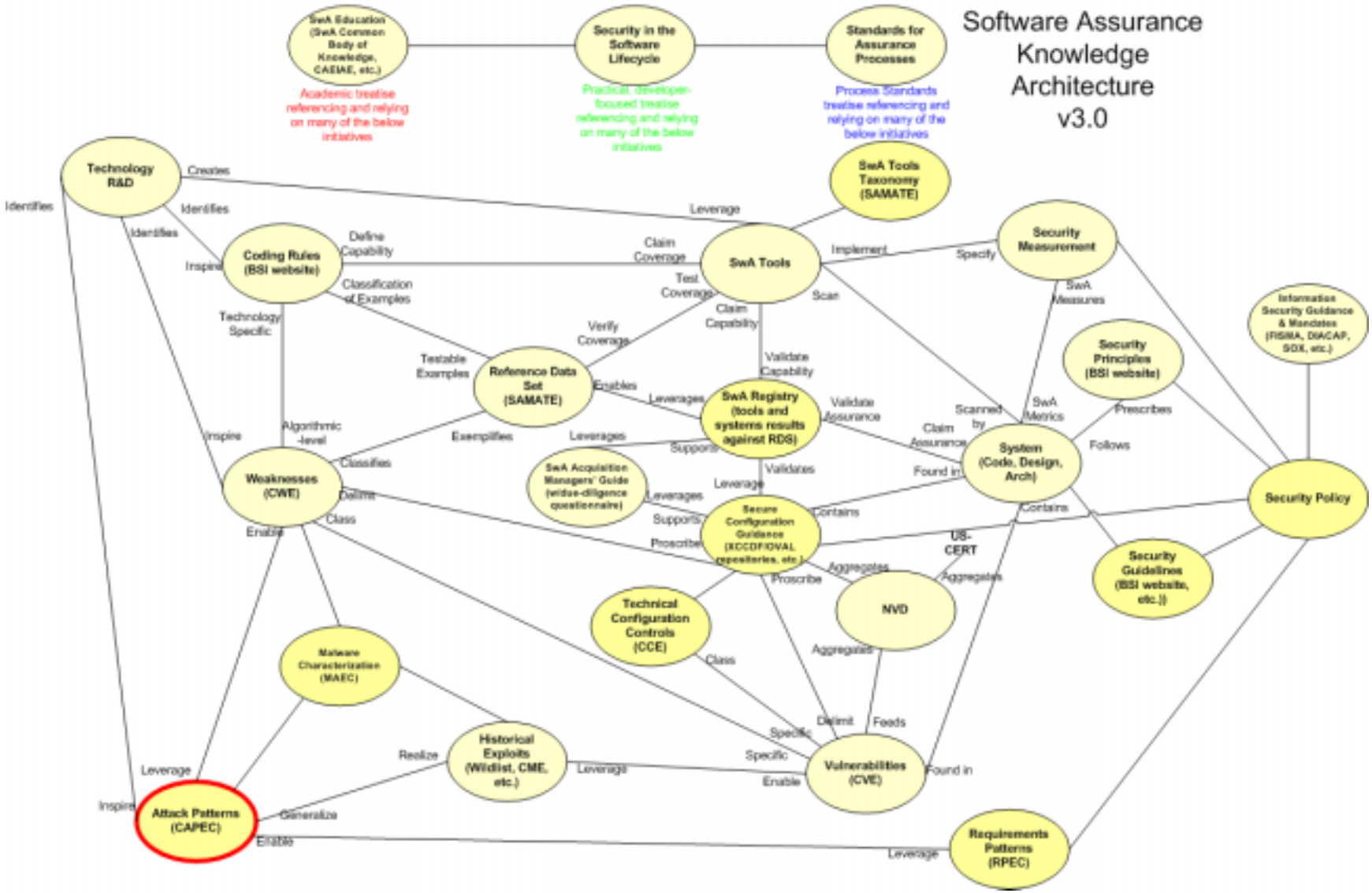
cigital

# Fitting CAPEC into the Bigger Picture

- CAPEC is most valuable when its content is aligned with related software assurance knowledge collections
  - Yields gestalt where the whole is greater than the sum of the parts
    - The DHS/DOD Software Assurance Knowledge Architecture
  - Common Weakness Enumeration (CWE)
  - Common Malware Enumeration (CME)
  - Security Principles
  - Security Guidelines
  - Etc.

cigital

# The Big Picture



# What to Expect Going Forward from CAPEC

- Draft attack pattern enumeration should be available for review in early to mid-March
- Initial release of CAPEC including deployment to publicly available website should late March to early April

cigital

# Community Involvement and Future Growth

- DHS/DOD Software Assurance programs
- OMG Software Assurance SIG
- Contribution/Involvement Opportunities
  - Community review & feedback
  - Contributing new APs

cigital

# Summary

- Understanding and representing the attacker's perspective is critical to building secure software
- Attack patterns are a powerful resource for capturing and communicating this perspective
- Attack patterns have direct value across the entire SDLC
- CAPEC is one ongoing effort to standardize, collect and share common attack patterns
- There are opportunities for you to get involved and contribute to realizing the value of attack patterns for the broader software community

cigital

# Never Underestimate Your Adversary

- "The individualist without strategy who takes opponents lightly will inevitably become the captive of others."
  - Chapter 9: "Maneuvering Armies"
    - The Art of War, Sun Tzu

cigital

# Questions?

**Further questions or want to get involved?**

sbarnum@cigital.com

cigital