Attack Patterns: Knowing Your Enemy in Order to Defeat Them

BlackHat DC
2007

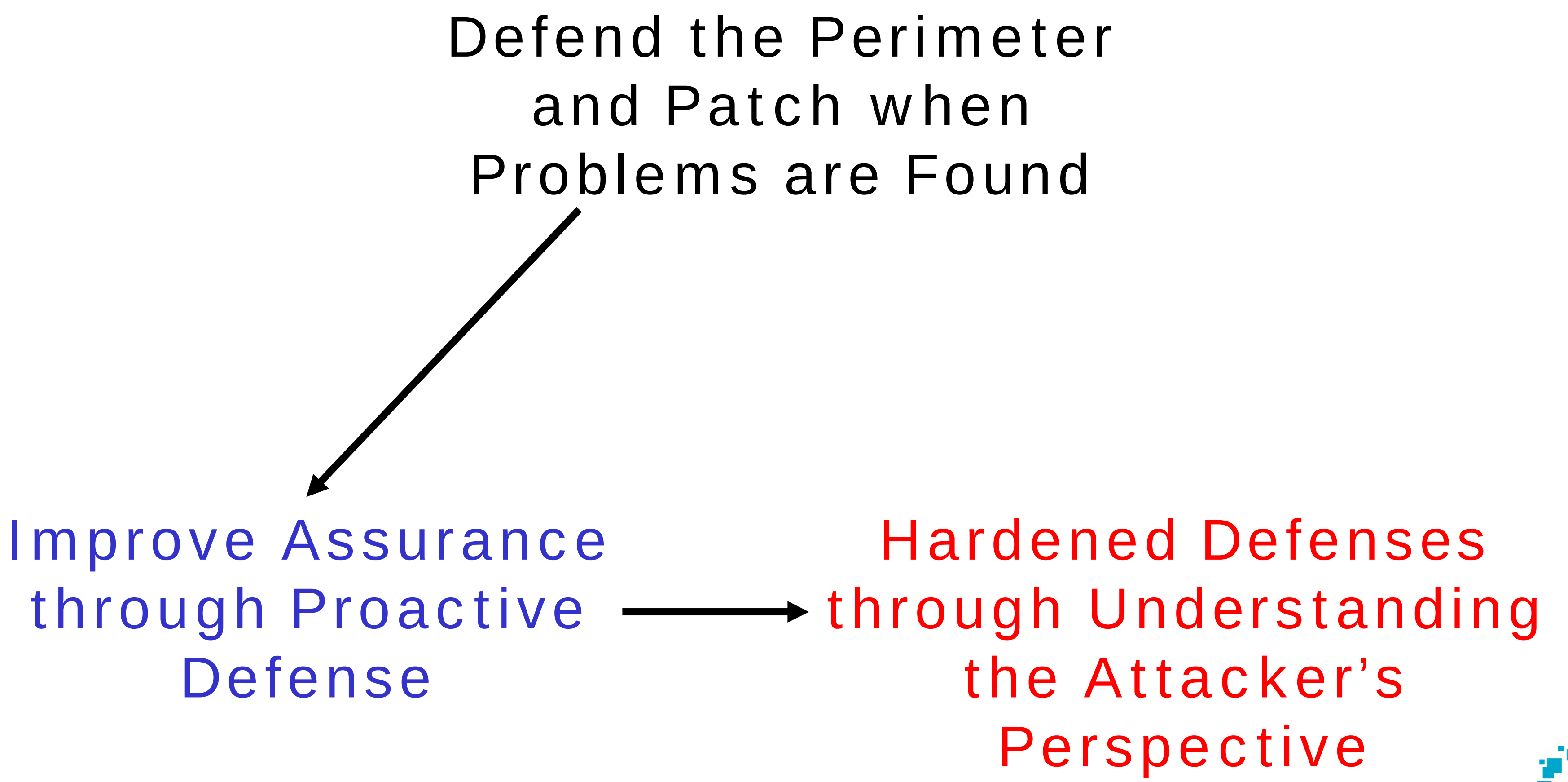
Sean Barnum
Managing Consultant
sbarnum@cigital.com

cigital

Software Confidence. Achieved.

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

Evolution of Software Assurance



Defend the Perimeter and Patch when Problems are Found

Improve Assurance through Proactive Defense

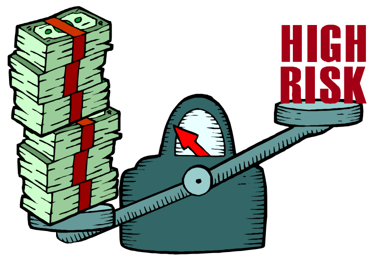
Hardened Defenses through Understanding the Attacker's Perspective

- 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

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

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

Attack Patterns Background

What are Attack Patterns?

- An attack pattern is a blueprint for an exploit. They are developed by reasoning over large sets of software exploits.

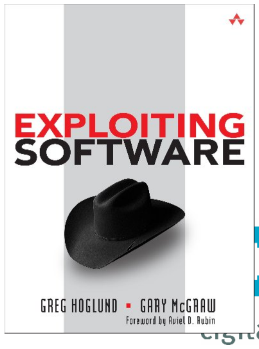
- Attack patterns help identify and qualify the risk that a given exploit will occur in a software system.

Background & Related Concepts

- Design Patterns
  - Christopher Alexander and then the Gang of Four (Gamma, et al)
- *Exploiting Software* [Hoglund & McGraw]
  - Applying pattern concept to methods of exploit
- 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)

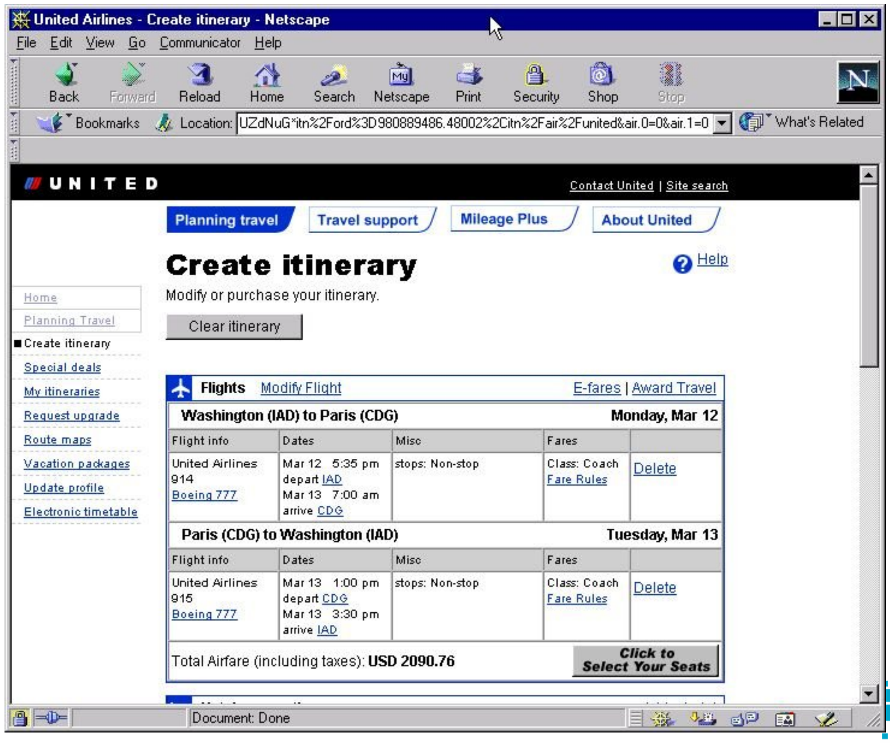
Knowledge: 48 Attack Patterns

- 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



Location: UZdNuG*itn%2Ford%3D980889486.48002%2Citn%2Fair%2Funited&air.0=0&air.1=0

Create itinerary

Modify or purchase your itinerary.

Washington (IAD) to Paris (CDG) — **Monday, Mar 12**

Flight info	Dates	Misc	Fares	
United Airlines 914 Boeing 777	Mar 12 5:35 pm depart IAD Mar 13 7:00 am arrive CDG	stops: Non-stop	Class: Coach Fare Rules	Delete

Paris (CDG) to Washington (IAD) — **Tuesday, Mar 13**

Flight info	Dates	Misc	Fares	
United Airlines 915 Boeing 777	Mar 13 1:00 pm depart CDG Mar 13 3:30 pm arrive IAD	stops: Non-stop	Class: Coach Fare Rules	Delete

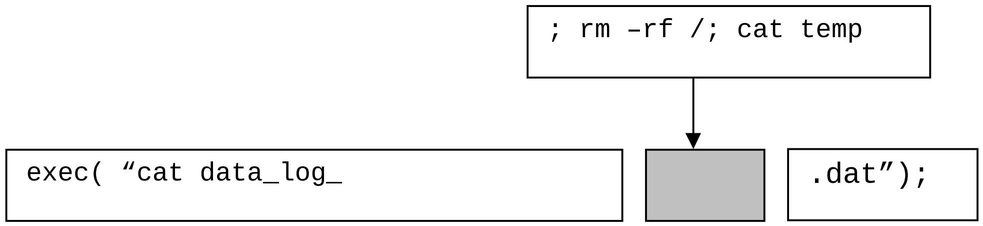
Total Airfare (including taxes): **USD 2090.76**

cigital

Attack Pattern 2: Command delimiters

- Use off-nominal characters to string together multiple commands
- Example: shell command injection with delimiters

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



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

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
        - Severity
        - Likelihood of Exploit
        - Attack Prerequisites
        - Attacker Skill or Knowledge Required
        - Resources Required
        - Attack Motivation-Consequences
        - Context Description

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: 1. Identifies the user-controllable input that causes arbitrary HTTP header injection. 2. Crafts a malicious input consisting of data to terminate the original response and start a second response with headers controlled by the attacker. 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 Ability of attacker to inject custom strings in HTTP header 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 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. 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 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 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 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. CWE - HTTP Response Splitting CWE - Injection
Submission Source	Chiradeep B Chhaya2007-01-09First Draft
Modification Source	

cigital

Attack Pattern Generation

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
- 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?

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?

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

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

Leveraging Attack Patterns

cigital

Where They Are Leveraged

- 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 across the software development lifecycle (SDLC)

- 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

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 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

Resource: Misuse/Abuse Cases

- Objective
  - Capture and personify attacking behaviors against the system as requirements for attack resistance
- Key Factors
  - Can be derived from Attack Patterns
  - Form basis for security testing of attack resistance
  - Consist of typical use case fields
  - Relationships with Use Cases
  - Mapping to relevant Attack Patterns
  - Efficacy Targets
    - Resistance
    - Recovery

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 identify appropriate recommended or non-recommended design patterns

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)

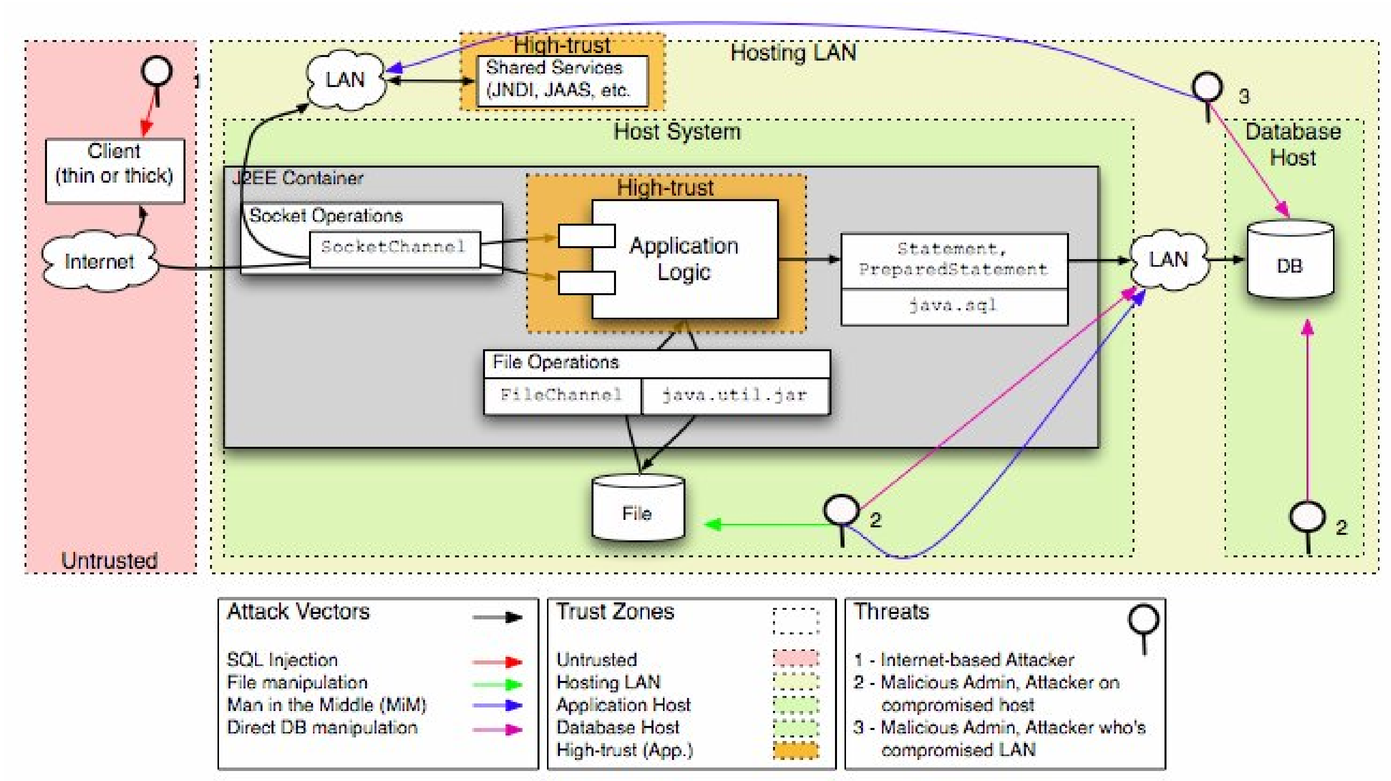
cigital

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 Analysis Diagrams



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

Where They Are Leveraged – Implementation

- Attack Patterns can be an invaluable resource in guiding secure code implementation practices through targeting 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 and confirm the absence of

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

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

Where They Are Leveraged – Operations

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

- Operations
  - Development perspective
    - Using relevant attack patterns to identify appropriate secure operations configurations
  - 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

Where They Are Leveraged – Security Policy

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

- Generating security policies
  - Development perspective
    - Using relevant attack patterns to identify appropriate security policies and guidelines
  - Security Assurance perspective
    - Using relevant attack patterns to identify appropriate guidelines and context for verifying compliance with appropriate security policies

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

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
- SDLC Stage

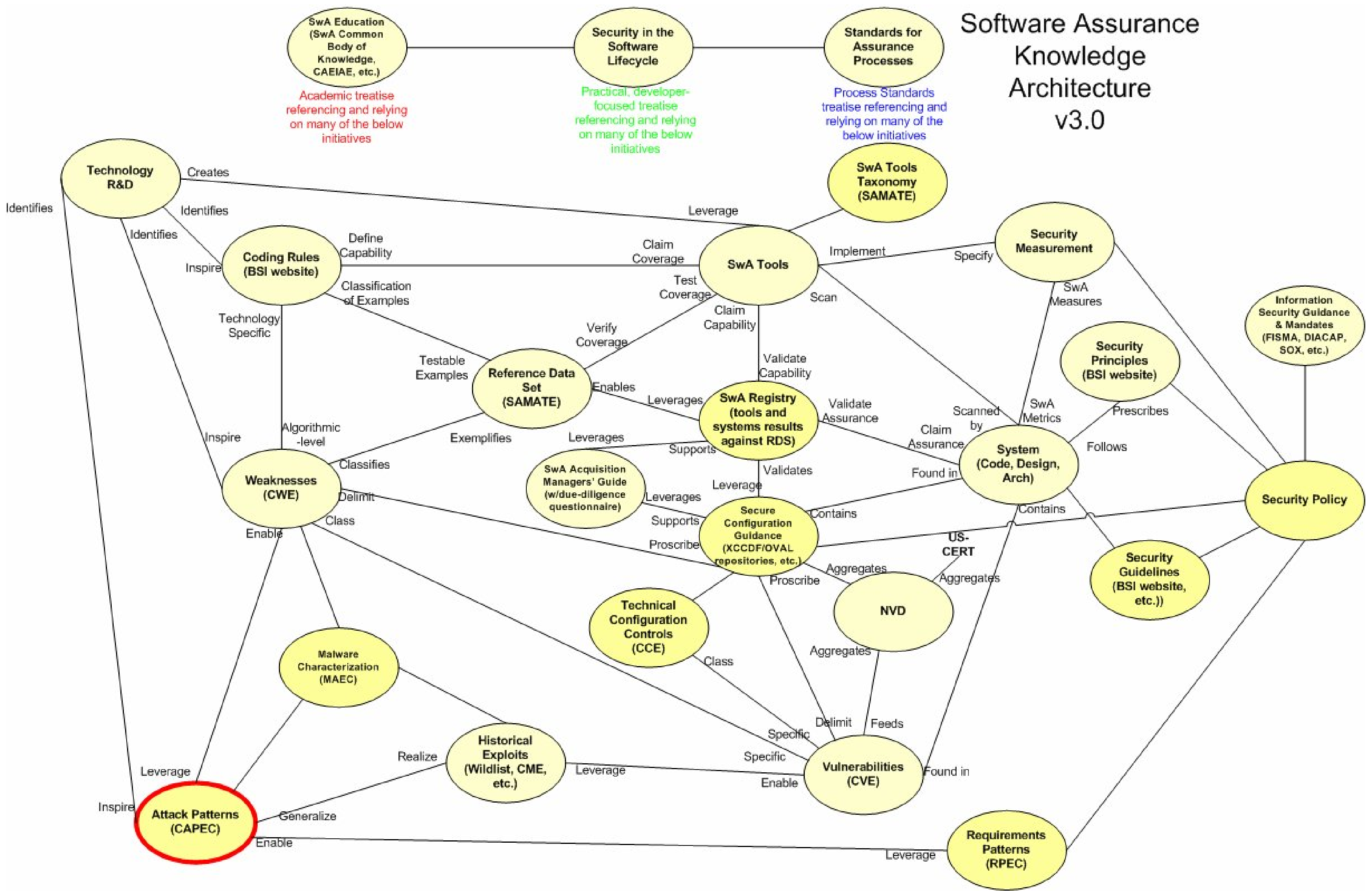
cigital

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)

cigital

The Big Picture



Software Assurance Knowledge Architecture v3.0
SwA Education (SwA Common Body of Knowledge, CAEIAE, etc.)
Academic treatise referencing and relying on many of the below initiatives
Security in the Software Lifecycle
Practical, developer-focused treatise referencing and relying on many of the below initiatives
Standards for Assurance Processes
Process Standards treatise referencing and relying on many of the below initiatives
SwA Tools Taxonomy (SAMATE)
Technology R&D
Creates
Identifies
Identifies
Identifies
Leverage
Coding Rules (BSI website)
Define Capability
Inspire
Classification of Examples
Technology Specific
Claim Coverage
SwA Tools
Test Coverage
Claim Capability
Implement
Specify
Security Measurement
Scan
SwA Measures
Information Security Guidance & Mandates (FISMA, DIACAP, SOX, etc.)
Security Principles (BSI website)
Verify Coverage
Testable Examples
Reference Data Set (SAMATE)
Enables
Validate Capability
SwA Registry (tools and systems results against RDS)
Leverages
Validate Assurance
Scanned by
SwA Metrics
Prescribes
Claim Assurance
System (Code, Design, Arch)
Follows
Inspire
Algorithmic -level
Exemplifies
Leverages
Supports
Classifies
Weaknesses (CWE)
SwA Acquisition Managers' Guide (w/due-diligence questionnaire)
Validates
Found in
Security Policy
Delimit
Leverages
Leverage
Class
Supports
Secure Configuration Guidance (XCCDF/OVAL repositories, etc.)
Contains
Contains
Enable
Proscribe
US-CERT
Security Guidelines (BSI website, etc.)
Aggregates
Proscribe
Aggregates
Technical Configuration Controls (CCE)
NVD
Malware Characterization (MAEC)
Class
Aggregates
Delimit
Feeds
Specific
Historical Exploits (Wildlist, CME, etc.)
Realize
Specific
Leverage
Vulnerabilities (CVE)
Enable
Found in
Leverage
Inspire
Attack Patterns (CAPEC)
Generalize
Enable
Leverage
Requirements Patterns (RPEC)

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

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

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