Application Threat Modeling via the PASTA Methodology

Tony UcedaVelez

Managing Partner/ Founder, VerSprite

APAC 2013

OWASP

The Open Web Application Security Project

Speaker Background

OWASP
The Open Web Application Security Project

- Cornell University graduate
- Beginnings commercial finance consulting
- Transitioned to IT across multiple roles (System Administration, Development, Network Engineering, Support Operations, Implementation)
- Worked for top global companies across multiple sectors (Healthcare, Finance, Information Services, Government, Telecommunications, Banking, Consumer Electronics, Hospitality (F&B, Hotel, Tourism), BPO, Shared Service Models)
- Founder, Managing Partner at VerSprite

What is PASTA?

- <u>What is PASTA?</u>
- **Process for Attack Simulation & Threat Analysis**
  - **Integrated application threat analysis**
  - **Application threat modeling methodology that is risk based**
  - **Identify most viable threats and mitigate them.**
- **Provides a framework for efficiency and security integration**

- <u>Why PASTA is delicious?</u>
- **Current menu of application testing doesn't provide a full security meal**
  - **Pen Tests: Exploit driven**
  - **Risk Assessments: Subjective; lacks meat**
  - **Static Analysis: Weakness, flaw driven; disregards threats**
  - **Vuln Scans: (C'mon! As if this could provide a decent meal!)**
  - **Too much fighting at security dinner table: Security testing is adversarial**
  - **Integrated disciplines are needed via a unifying methodology**

PASTA™ Recipe

1. Define Objectives

- Identify Business Objectives
- Identify Security & Compliance Requirements
- Business Impact Analysis

2. Define Technical Scope

- Capture the boundaries of the technical environment
- Capture Infrastructure | Application | Software Dependencies

3. Application Decomposition

- Identify Use Cases | Define App Entry Points & Trust levels
- Identify Actors | Assets| Services | Roles| Data Sources
- Data Flow Diagramming (DFDs) | Trust Boundaries

4. Threat Analysis

- Probabilistic Attack Scenarios Analysis
- Regression Analysis on Security Events
- Threat Intelligence Correlation & Analytics

5. Vulnerability & Weakness Analysis

- Queries of Existing Vulnerability Reports & Issues Tracking
- Threat to Existing Vulnerability Mapping Using Threat Trees
- Design Flaw Analysis Using Use & Abuse Cases
- Scorings (CVSS/ CWSS) | Enumerations (CWE/CVE)

6. Attack Modeling

- Attack Surface Analysis
- Attack Tree Development | Attack Library Mgt
- Attack to Vulnerability & Exploit Analysis using Attack Trees

7. Risk & Impact Analysis

- Qualify & quantify business impact
- Countermeasure Identification & Residual Risk Analysis
- ID risk mitigation strategies

VerSprite
Navigate Beyond Risk

Taxonomy of Terms

OWASP
The Open Web Application Security Project

- **Asset.** An asset is a resource of value. It varies by perspective. To your business, an asset might be the availability of information, or the information itself, such as customer data. It might be intangible, such as your company's reputation.
- **Threat.** A threat is an undesired event. A potential occurrence, often best described as an effect that might damage or compromise an asset or objective.
- **Vulnerability.** A vulnerability is a weakness in some aspect or feature of a system that makes an exploit possible. Vulnerabilities can exist at the network, host, or application levels and include operational practices.
- **Attack (or exploit).** An attack is an action taken that utilizes one or more vulnerabilities to realize a threat.
- **Countermeasure.** Countermeasures address vulnerabilities to reduce the probability of attacks or the impacts of threats. They do not directly address threats; instead, they address the factors that define the threats.
- **Use Case.** Functional, as designed features of an application.
- **Abuse Case.** Deliberate abuse of functional use cases in order to yield unintended results
- **Attack Vector.** Point & channel for which attacks travel over (card reader, form fields, network proxy)
- **Attack Surface.** Logical area (browser stack) or physical area (hotel kiosk)
- **Actor.** Legit or adverse caller of use or abuse cases.
- **Impact.** Value of [financial] damage possibly sustained via attack.
- **Attack Tree.** Diagram of relationship amongst asset-actor-use case-abuse case-vuln-exploit-countermeasure

STAGE I – Define Business Objectives

Define the Business & Security Objectives:

"Capture requirements for the analysis and management of web based risks"

Baking in GRC

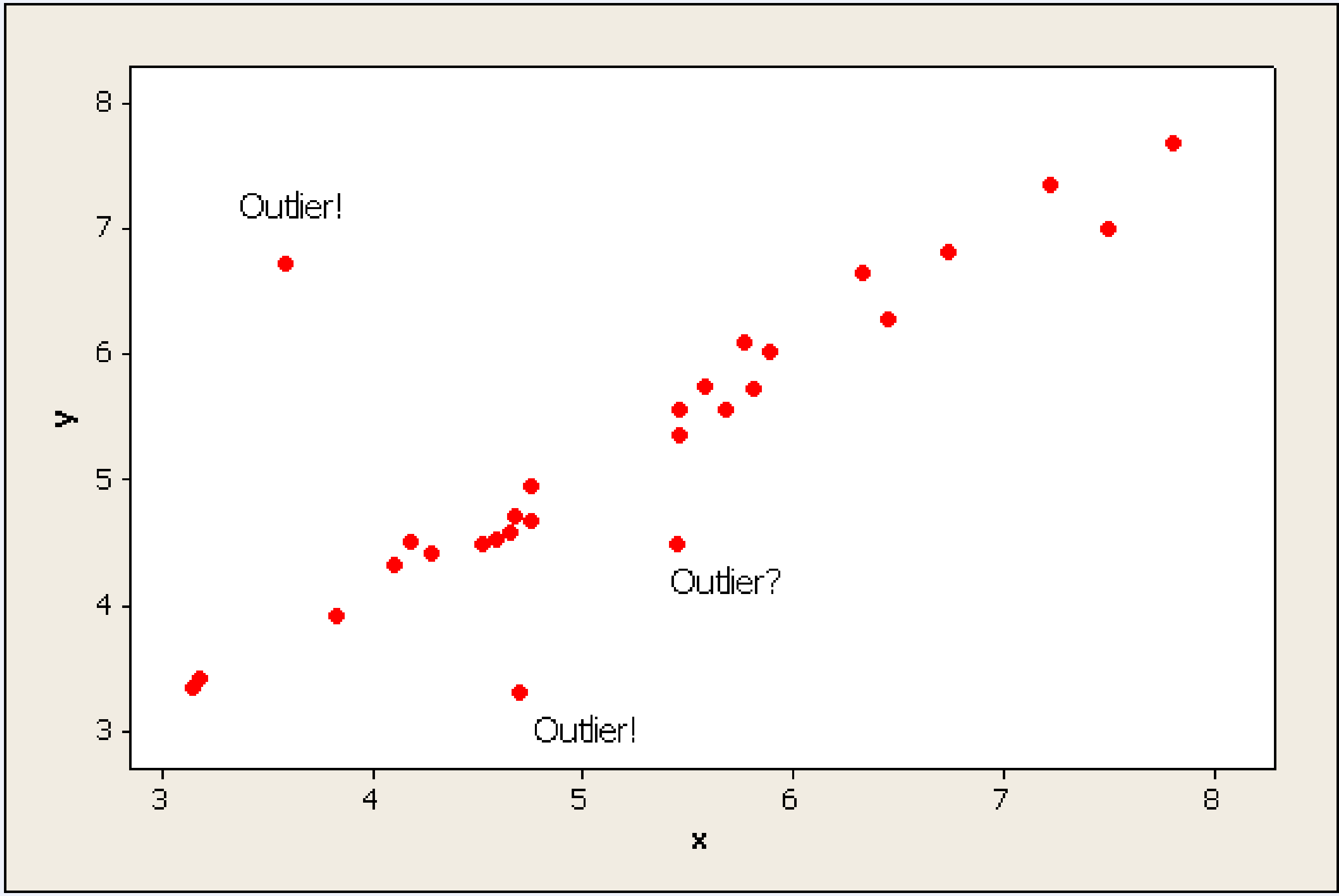
- **Using Unused Ingredients : Governance**
  - Policies (for people) – may factor in for apps whose attack vectors are heavily vulnerable to human resources
  - Standards (for technology) – factor in across network, server, client side technologies for pre-emptive risk mitigation.
- **Making Decent Food out of Leftovers : Risk Assessments 2nd Life**
  - Historical RAs provide prior risk profile of app
- **Regulatory landscape taken into consideration, but not the driver**
  - Key here is to not retrofit compliance; more costly
- **Where's the Beef: Business Objectives get Baked In**
  - How is an injection attack truly relevant to the business beyond trying to qualify a 9.4 CVSS score?

VerSprite Navigate Beyond Risk

How Outliers Affect Security

OWASP
The Open Web Application Security Project

- Objectives and Security both affect one another
- Over scoping of functional requirements
  - Orphaned features that lose maintenance
  - Insecure *Easter Eggs* in apps
  - 'I never knew that was there' scenario.



Threat Modeling Artifacts

OWASP
The Open Web Application Security Project

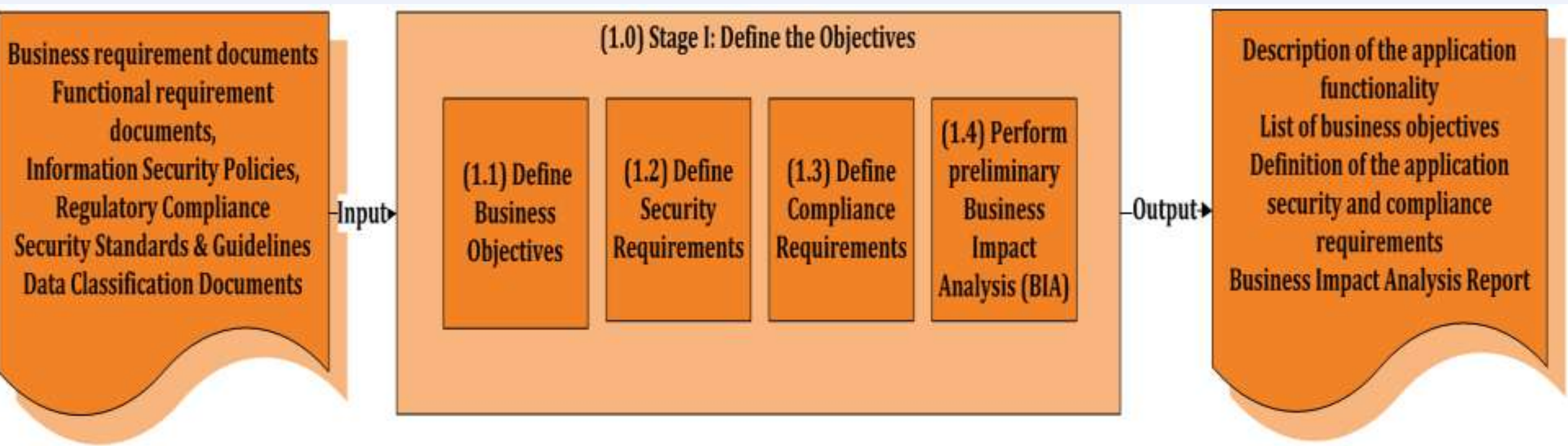
Application Profile: Online Banking Application	
General Description	The online banking application allows customers to perform banking activities such as financial transactions over the internet. The type of transactions supported by the application includes bill payments, wires, funds transfers between customer's own accounts and other bank institutions, account balance-inquires, transaction inquires, bank statements, new bank accounts loan and credit card applications. New online customers can register an online account using existing debit card, PIN and account information. Customers authenticate to the application using username and password and different types of Multi Factor Authentication (MFA) and Risk Based Authentication (RBA)
Application Type	Internet Facing
Data Classification	Public, Non Confidential, Sensitive and Confidential PII
Inherent Risk	HIGH (Infrastructure , Limited Trust Boundary, Platform Risks, Accessability)
High Risk Transactions	YES
User roles	Visitor, customer, administrator, customer support representative
Number of users	3 million registered customers

Compliance as a Business Objectives???

OWASP
The Open Web Application Security Project

Project Business Objective	Security and Compliance Requirement
Perform an application risk assessment to analyze malware banking attacks	Risk assessment need to assess risk from attacker perspective and identify on-line banking transactions targeted by the attacks
Identify application controls and processes in place to mitigate the threat	Conduct architecture risk analysis to identify the application security controls in place and the effectiveness of these controls. Review current scope for vulnerability and risk assessments.
Comply with FACT Act of 2003 and FFIEC guidelines for authentication in the banking environment	Develop a written program that identifies and detects the relevant warning signs – or "red flags" – of identity theft. Perform a risk assessment of online banking high risk transactions such as transfer of money and access of Sensitive Customer Information
Analyze attacks and the targets that include data and high risk transactions (Latest FFIEC)	Analyze attack vectors used for acquisition of customers' PII, logging credentials and other sensitive information. Analyze attacks against user account modifications, financial transactions (e.g. wires, bill-pay), new account linkages
Identify a Risk Mitigation Strategy That Includes Detective and Preventive Controls/Processes	Include stakeholders from Intelligence, IS, Fraud/Risk, Legal, Business, Engineering/Architecture. Identify application countermeasures that include preventive, detective (e.g. monitoring) and compensating controls against malware-based banking Trojan attacks

Stage 1 : Defines Business Objectives Mirrors DEFINE SDLC Phase



STAGE II

Define The Technical Scope: ”Defining the scope of technical assets/ components for which threat enumeration will ensue”

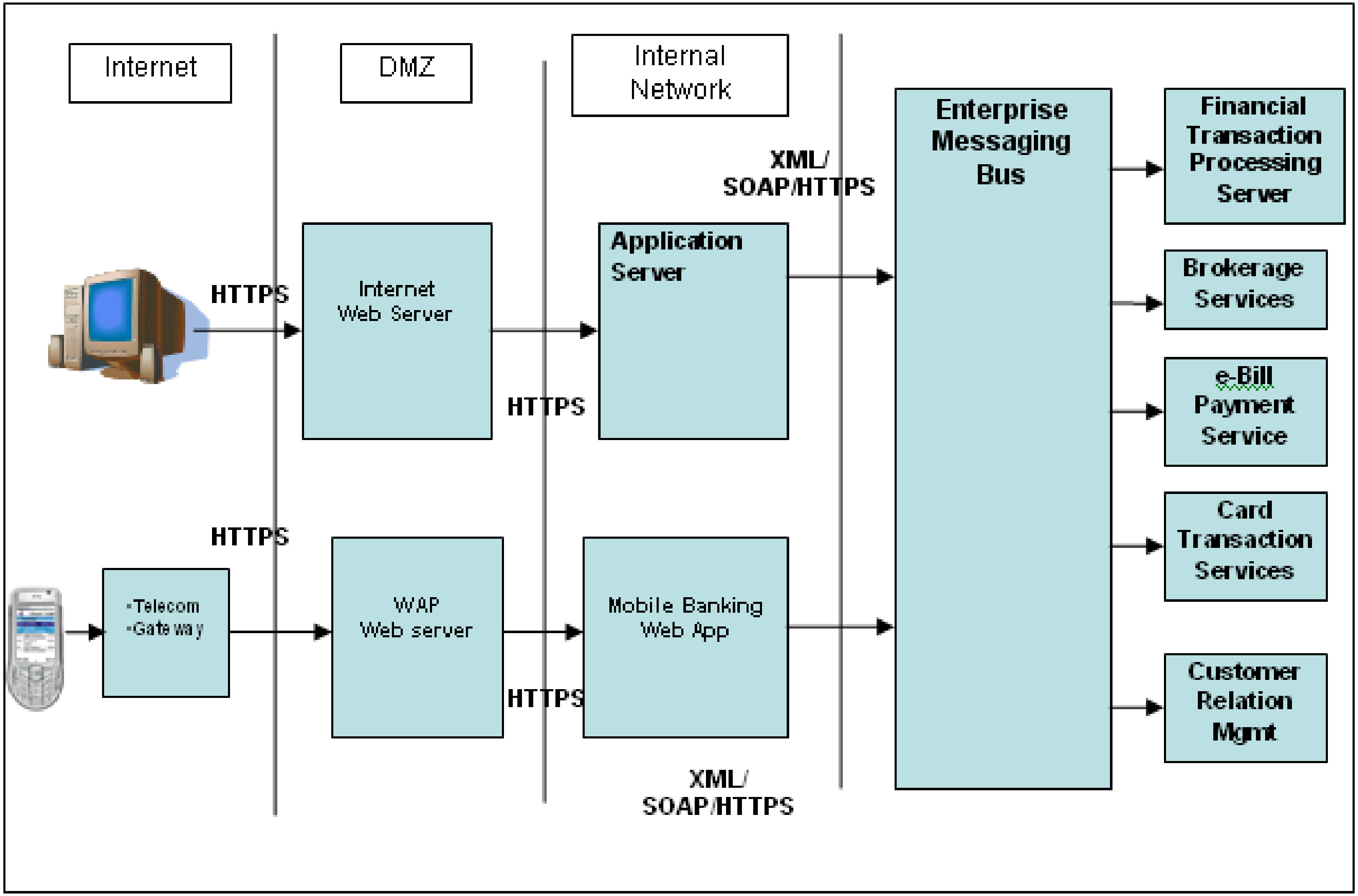
Technical Scope Definition

OWASP
The Open Web Application Security Project

- Define scope of technical landscape
  - **Application components**
  - **Network topology**
  - **Protocol/services**
  - **Use cases**
  - **Hardware/ COTS/ Middleware**

The Application Architecture Scope

OWASP
The Open Web Application Security Project

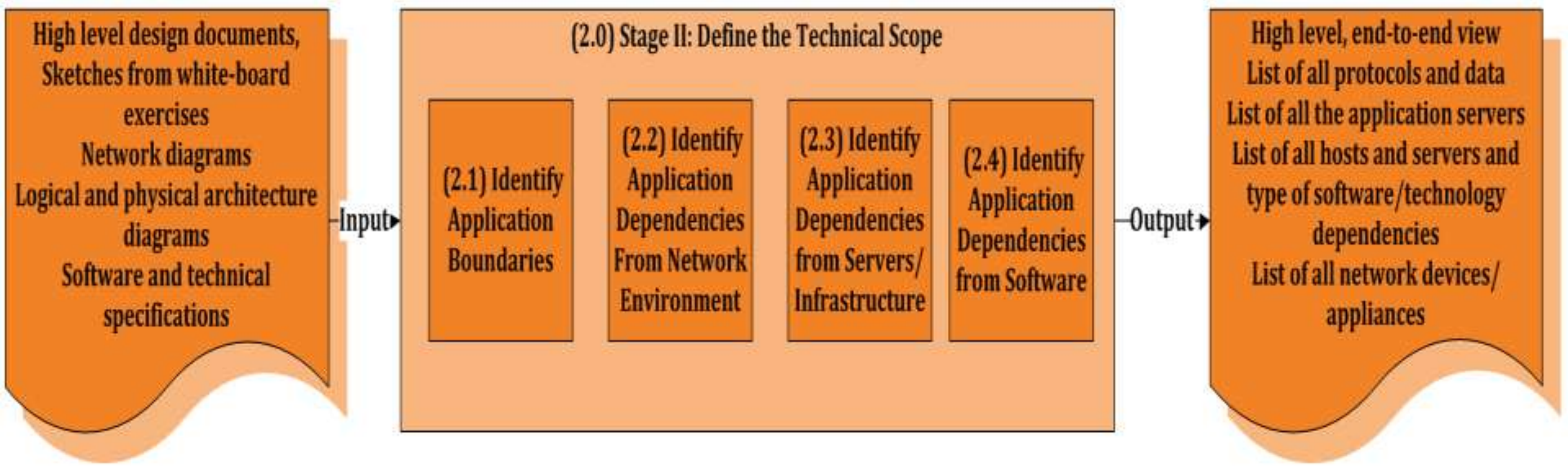


Baking in Technical Standards

OWASP
The Open Web Application Security Project

- Apply standard security architecture
- Apply internal security standards
- Apply client related security requirements
- Help develop security assurance against employed HW/ SW (COTS)
- End of this stage results in inherent countermeasures (people, process, technology)

Stage 2 : Technical Scoping Parallels DEFINE SLDC Phase



STAGE III

Decompose the Application :"Identify the application controls that protect high risk web transactions sought by adversaries"

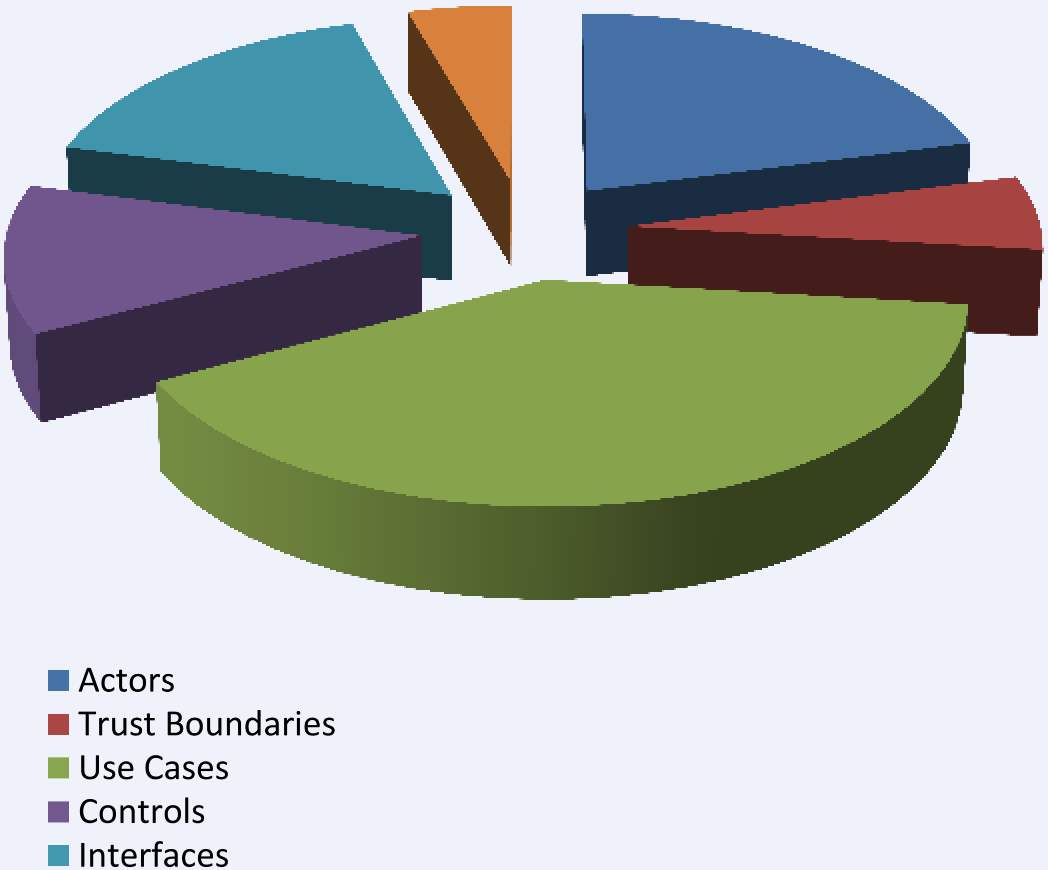
Application Slicing

OWASP
The Open Web Application Security Project

Application Dissection

OWASP
The Open Web Application Security Project

- Enumerate actors/ callers
- What calls do the actors make?
  - Key aspect of this phase
- Enumerate all use cases (transactions)
- Define trust boundaries (implicit vs explicit trust)
  - Domains, networks, hosts, services, etc
- ID data sources
- Can also enumerate target sub-set of use case



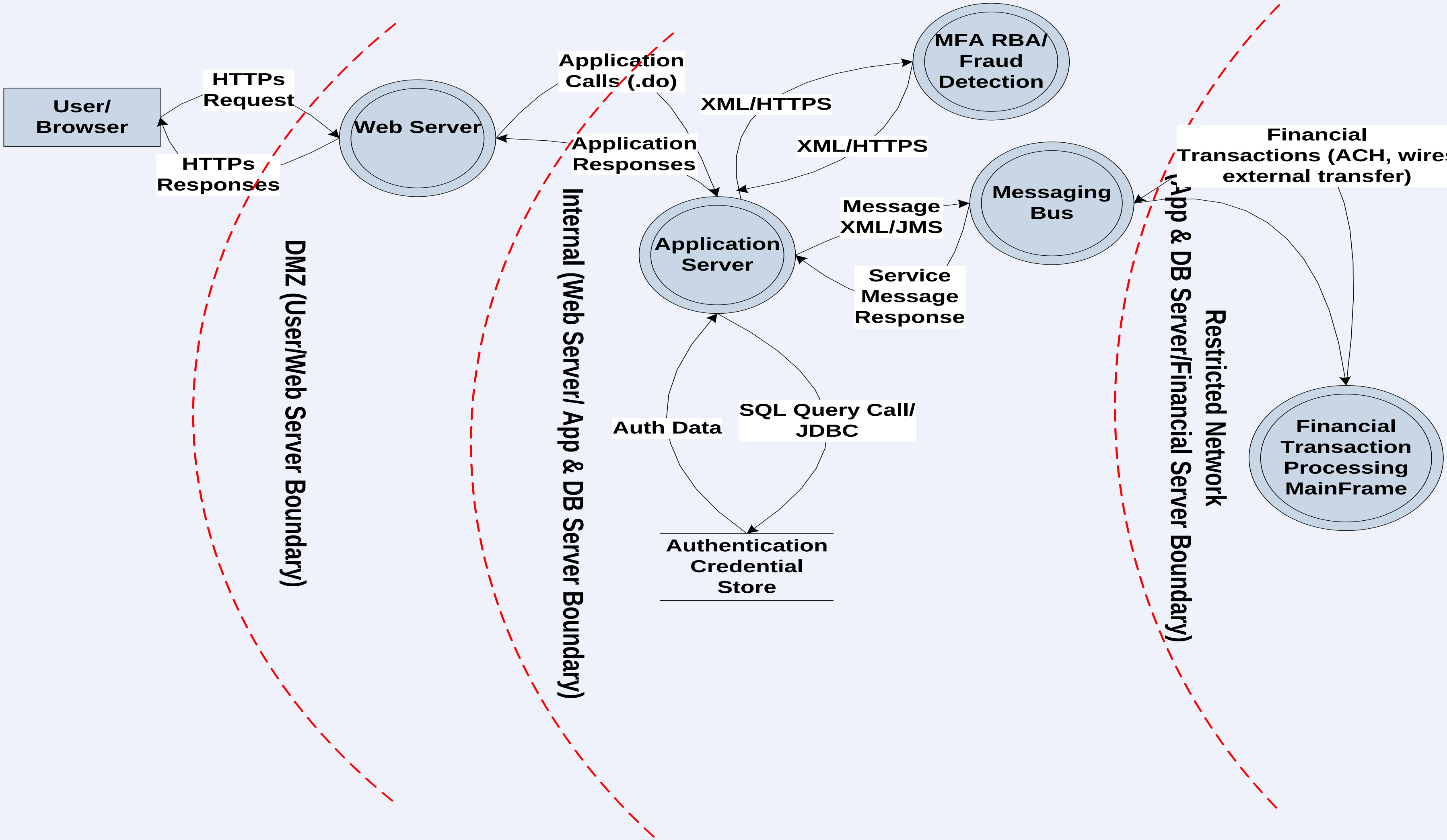
Transactional Security Control Sprint

OWASP
The Open Web Application Security Project

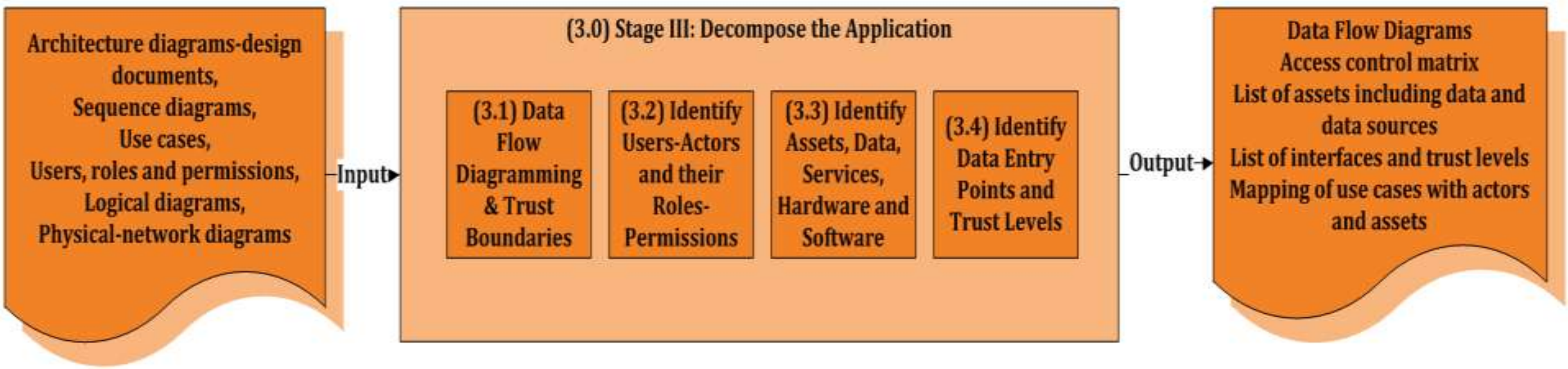
Online Banking Application Transaction Analysis			Data Input Validation (Initiation)	Authentication/ Identification	Authorization	Session Management	Cryptography (data in rest and transit)	Error Handling	Logging/Auditing /Monitoring
Transaction	Risk	Data Classification	Security Functions Invoked						
Password Reset	HIGH	Sensitive	Debit Card, PIN,Account#	Challenge/ Questions Risk Interdicted	Pre-Auth/Bank Customer	Pre-auth SessionID/ Cookie	HTTPS	Custom Errors & Messages	Application, Fraud Detection
Username Recovery	HIGH	Sensitive	Debit Card, PIN,Account#	Challenge/ Questions Risk Interdicted	Pre-Auth/Bank Customes	Pre-auth SessionID/ Cookie	HTTPS	Custom Errors & Messages	Application, Fraud Detection
Registration	MEDIUM	Confidential PII & Sensitive	Debit Card, PIN,Account#, PII (e.g. SSN), Demographics	OOB/ Confirmation	Visitor	Pre-auth SessionID/ Cookie	HTTPS	Custom Errors & Messages	Application
Logon	HIGH	Confidential PII & Sensitive	Username /Password	Single Auth + Challenge/ Questions Risk Interdicted	Post-Auth/Bank Customer	Post-auth SessionID Mgmt	HTTPS/ 3DES Token	Custom Errors & Messages	Application, Fraud Detection
Wires	HIGH	Confidential PII & Sensitive	Amount,Account#, IBAN/BIC	Single Auth + C/Q Risk Interdicted + OTP	Post-Auth/Bank Customer	Post-auth SessionID Mgmt	HTTPS	Custom Errors & Messages	Application, Fraud Detection
Bill Pay	HIGH	Confidential PII & Sensitive	Amount, Payee Account#	Single Auth + C/Q Risk Interdicted + OTP	Post-Auth/Bank Customer	Post-auth SessionID Mgmt	HTTPS	Custom Errors & Messages	Application, Fraud Detection

Visualizing of Trust Boundaries in Design Phases

OWASP
The Open Web Application Security Project



Stage 3 : Application Dissection Parallels DESIGN SLDC Phase



OWASP
The Open Web Application Security Project

STAGE IV - Threat Analysis

"Identifying and extracting threat information from sources of intelligence to learn about threat-attack scenarios used by web focused attack agents"

Threat Intelligence is Key

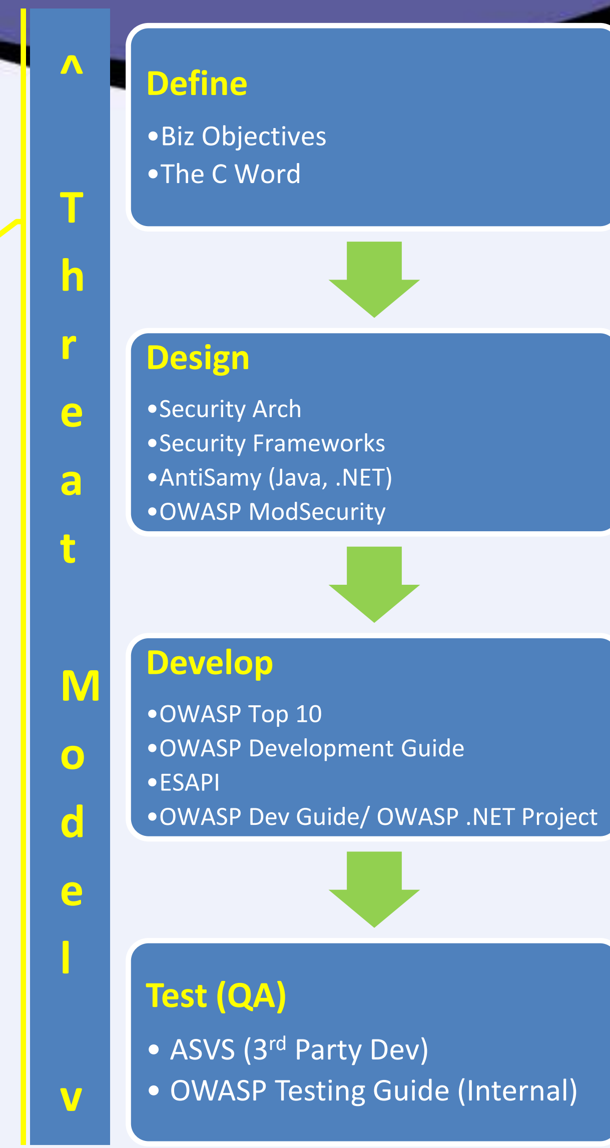
OWASP
The Open Web Application Security Project

- Traditional Sources
  - Isolated server/ app / network logs
  - Syslogs
  - General threat feeds/ news
  - SIEM products
  - SOC/ MSSP
  - Threat aggregation/ tailored threat intelligence
- Non-Traditional Sources
  - Physical security incidents
  - Third party incidents
  - Counter-intelligence subscriptions
  - Internal security testing
    - Security Testing: If it works here, how much more will it work within adversary circles

Blind Threat Model: Worst Case Scenario

OWASP
The Open Web Application Security Project

- A blind threat model says ‘I have no threat information’ but relies on global governance examples for inherent mitigation
  - Requires org to humbly agree on ‘security ignorance’ and consume the benefits of a baked-in secure SDLC.
- Business owners can consume prescriptive security governance (Definition Phase)
- Architects and IT Leaders speak to architectural design and platform solutions (Design Phase)
- Governance leaders inject compliance & standards requirements for during he design phase; BIA affects security objectives
- Aforementioned buys time to build Intelligence fed Threat Model



Stairway to Better Threat Model

OWASP
The Open Web Application Security Project

Blind Threat Model

- Industry 'Best Practice' Fed
- News helps shapes perception
- Internal testing may help legitimize probabilistic analysis

Event Driven Threat Model

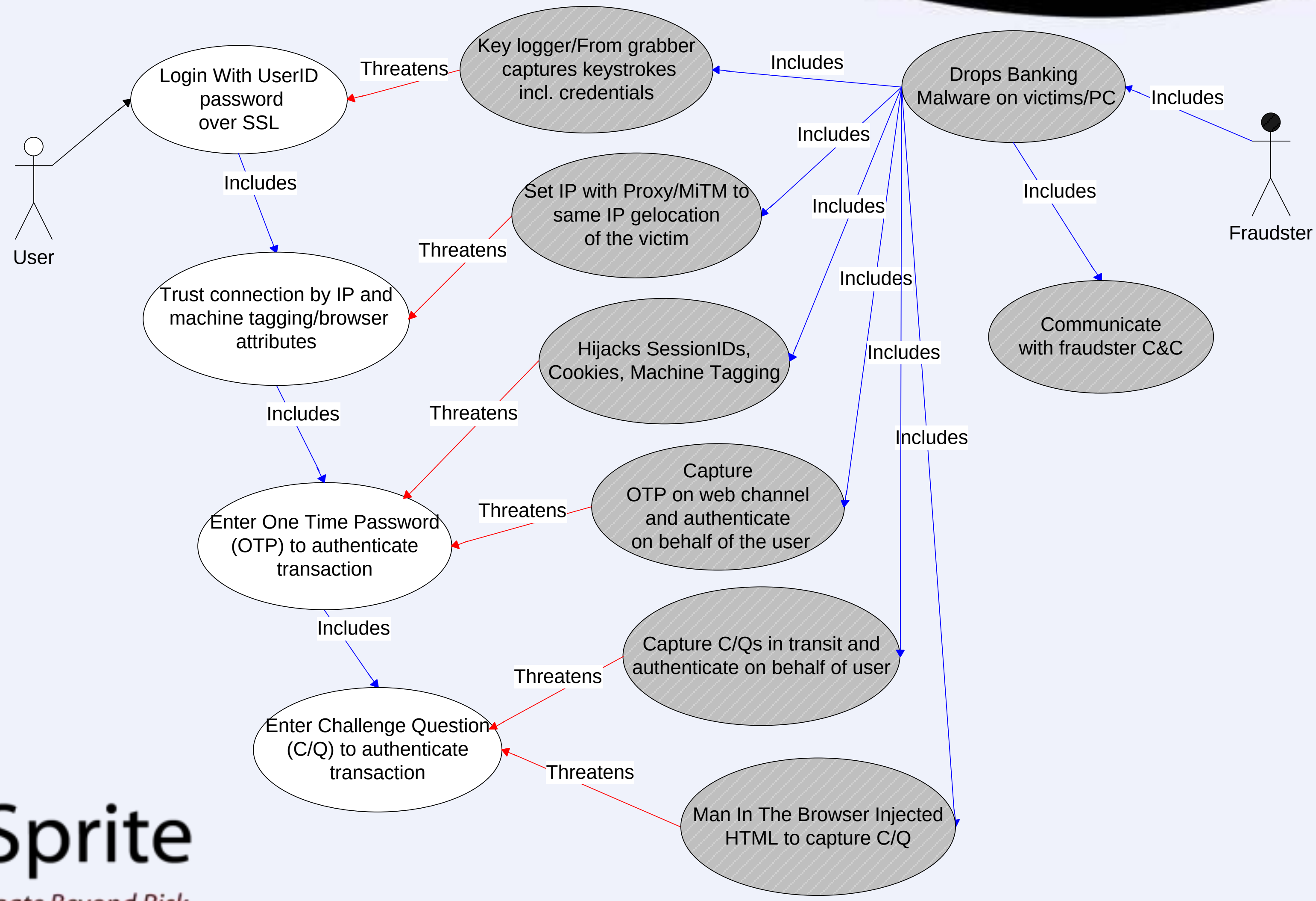
- Log centralization & analysis
- Begins with network and platform; app logs lag behind
- Correlation is game changer: client, server, network events

Advanced Threat Model

- Bakes in non-traditional threat intelligence sources
- Physical events correlated (email, phone, in-person)
- Counter threat intelligence

Threat Analysis

OWASP
The Open Web Application Security Project



VerSprite
Navigate Beyond Risk

Threat Analysis Process

OWASP
The Open Web Application Security Project

Perceived Motive

- Identified Targets
- Identified gains
- Identified risks (for attacker)
- Difficult to achieve, not essential but helpful

Understood Threats

- Data extraction
- DoS
- Attacking data integrity
- STRIDE/ DREAD mention

Asset Mapping

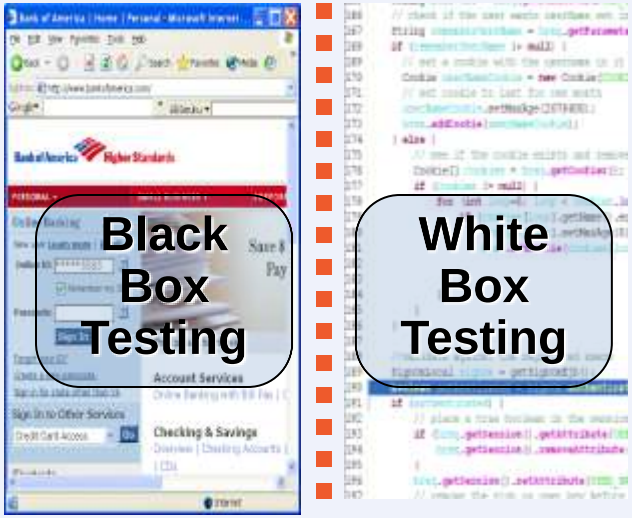
- High Level Data
- More Detail Data Asset Enumeration
  - Leads into next phase: vulnerability analysis

Organizing Threat via MITRE, SANS, OWASP

OWASP
The Open Web Application Security Project

WASC Threat Classification v2	OWASP Top Ten 2010 RC1
WASC-19 SQL Injection	A1 - Injection
WASC-23 XML Injection	
WASC-28 Null Byte Injection	
WASC-29 LDAP Injection	
WASC-30 Mail Command Injection	
WASC-31 OS Commanding	
WASC-39 XPath Injection	
WASC-46 XQuery Injection	
WASC-08 Cross-Site Scripting	A2 -Cross Site Scripting (XSS)
WASC-01 Insufficient Authentication	A3 - Broken Authentication and Session
WASC-18 Credential/Session Prediction	
WASC-37 Session Fixation	
WASC-47 Insufficient Session Expiration	
WASC-01 Insufficient Authentication	A4 - Insecure Direct Object References
WASC-02 Insufficient Authorization	
WASC-33 Path Traversal	
WASC-09 Cross-site Request Forgery	A5 - Cross-Site Request Forgery
WASC-14 Server Misconfiguration	A6 - Security Misconfiguration
WASC-15 Application Misconfiguration	
WASC-02 Insufficient Authorization	A7 - Failure to Restrict URL Access
WASC-10 Denial of Service	
WASC-11 Brute Force	
WASC-21 Insufficient Anti-automation	
WASC-34 Predictable Resource Location	
WASC-38 URL Redirector Abuse	A8 - Unvalidated Redirects and Forwards
WASC-50 Insufficient Data Protection	A9 - Insecure Cryptographic Storage
WASC-04 Insufficient Transport Layer Protection	A10 -Insufficient Transport Layer Protection

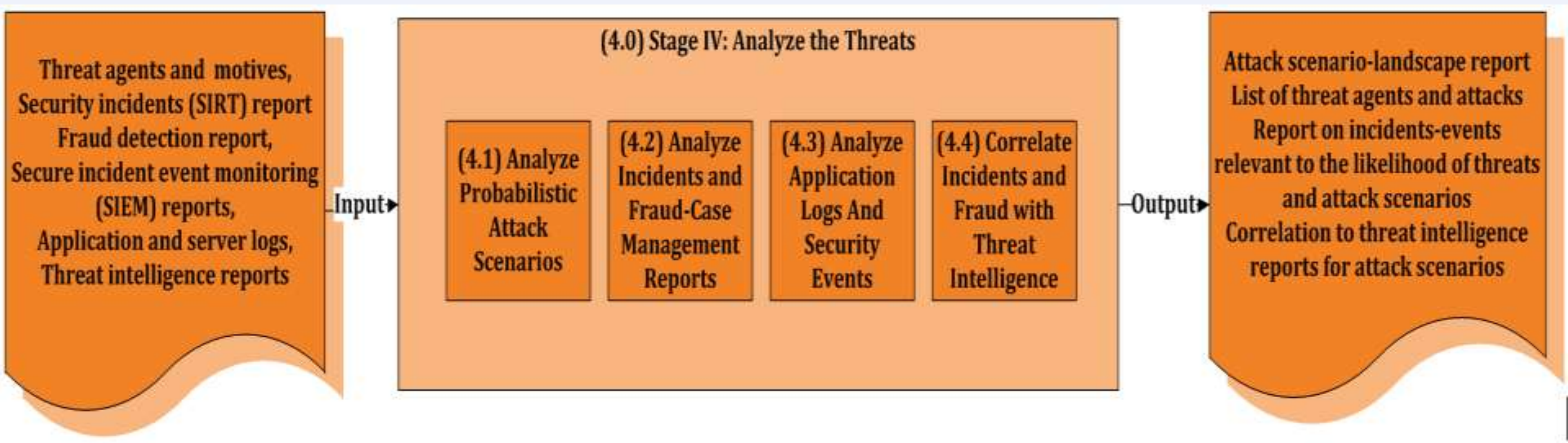
OWASP Top Ten 2010 RC1	2010 Top 25
A1 - Injection	CWE-89 (SQL injection), CWE-78 (OS Command injection)
A2 - Cross Site Scripting (XSS)	CWE-79 (Cross-site scripting)
A3 - Broken Authentication and Session Management	CWE-306, CWE-307, CWE-798
A4 - Insecure Direct Object References	CWE-285
A5 - Cross Site Request Forgery (CSRF)	CWE-352
A6 - Security Misconfiguration	No direct mappings; CWE-209 is frequently the result of misconfiguration.
A7 - Failure to Restrict URL Access	CWE-285
A8 - Unvalidated Redirects and Forwards	CWE-601
A9 - Insecure Cryptographic Storage	CWE-327, CWE-311
A10 - Insufficient Transport Layer Protection	CWE-311



Categories - into which the problem types are divided for diagnostic and resolution purposes.	Problem Types - (i.e., basic causes) underlying security-related vulnerabilities.	Description	Consequences - of exploited vulnerabilities for basic security services. Can be failures in these basic security services: Authorization (resource access control), Confidentiality (of data or other resources), Authentication (identity establishment & integrity), Availability (denial of service), Accountability, & Non-repudiation	SDLC Phase - Exposure Period	Exposure Period - (i.e., SDLC phases) in which vulnerabilities can be inadvertently introduced into application source code.	SDLC Phase - Avoidance & Mitigation	Avoidance & Mitigation - (i.e., SDLC phases) in which preventative measures and countermeasures can be applied.	Platforms - which may be affected by a vulnerability.	Required Resources - prerequisites for exploit[illegible] attack vulnerabilities in application's source cod[illegible]
Range & Type	Buffer Overflow	A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers.	•Availability: Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop. •Access control (instruction processing): Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. •Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.	Requirements	•Requirements specification: The choice could be made to use a language that is not susceptible to these issues.	Requirements	•Pre-design: Use a language or compiler that performs automatic bounds checking.	•Languages: C, C++, Fortran, Assembly •Operating platforms: All, although partial preventatise measures may be deployed, depending on environment.	Any
Range & Type	Buffer Overflow	A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers.	•Availability: Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop. •Access control (instruction processing): Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. •Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.	Design	•Design: Mitigating technologies such as safe-string libraries and container abstractions could be introduced.	Design	•Design: Use an abstraction library to abstract away risky APIs. Not a complete solution.	•Languages: C, C++, Fortran, Assembly •Operating platforms: All, although partial preventatise measures may be deployed, depending on environment.	Any
Range & Type	Buffer Overflow	A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers.	•Availability: Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop. •Access control (instruction processing): Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. •Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.	Implementation	•Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.	Requirements	•Pre-design through Build: Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.	•Languages: C, C++, Fortran, Assembly •Operating platforms: All, although partial preventatise measures may be deployed, depending on environment.	Any
Range & Type	Buffer Overflow	A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers.	•Availability: Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop. •Access control (instruction processing): Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. •Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.	Implementation	•Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or misuse of mitigating technologies.	Operational	•Operational: Use OS-level preventative functionality. Not a complete solution.	•Languages: C, C++, Fortran, Assembly •Operating platforms: All, although partial preventatise measures may be deployed, depending on environment.	Any
Range & Type	"Write-what-where" condition	Any condition where the attacker has the ability to write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.	•Access control (memory and instruction processing): Clearly, write-what-where conditions can be used to write data to areas of memory outside the scope of a policy. Also, they almost invariably can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. •Availability: Many memory accesses can lead to program termination, such as when writing to addresses that are invalid for the current process. •Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.	Requirements	•Requirements: At this stage, one could specify an environment that abstracts memory access, instead of providing a single, flat address space.	Requirements	•Pre-design: Use a language that provides appropriate memory abstractions.	•Languages: C, C++, Fortran, Assembly •Operating platforms: All, although partial preventatise mea[illegible] may be deployed depe[illegible]	Any
Range & Type	"Write-what-where" condition	Any condition where the attacker has the ability to write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.	•Access control (memory and instruction processing): Clearly, write-what-where conditions can be used to write data to areas of memory outside the scope of a policy. Also, they almost invariably can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. •Availability: Many memory accesses can lead to program termination, such as when writing to addresses that are invalid for the current process. •Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.	Design	•Design: Many write-what-where problems are buffer overflows, and mitigating technologies for this subset of problems can be chosen at this time.	Design	•Design: Integrate technologies that tr[illegible] consequences of this problems.	[illegible]	[illegible]

OWASP
The Open Web Application Security Project

Stage 4 : Threat Analysis
Parallels DESIGN SDLC Phase



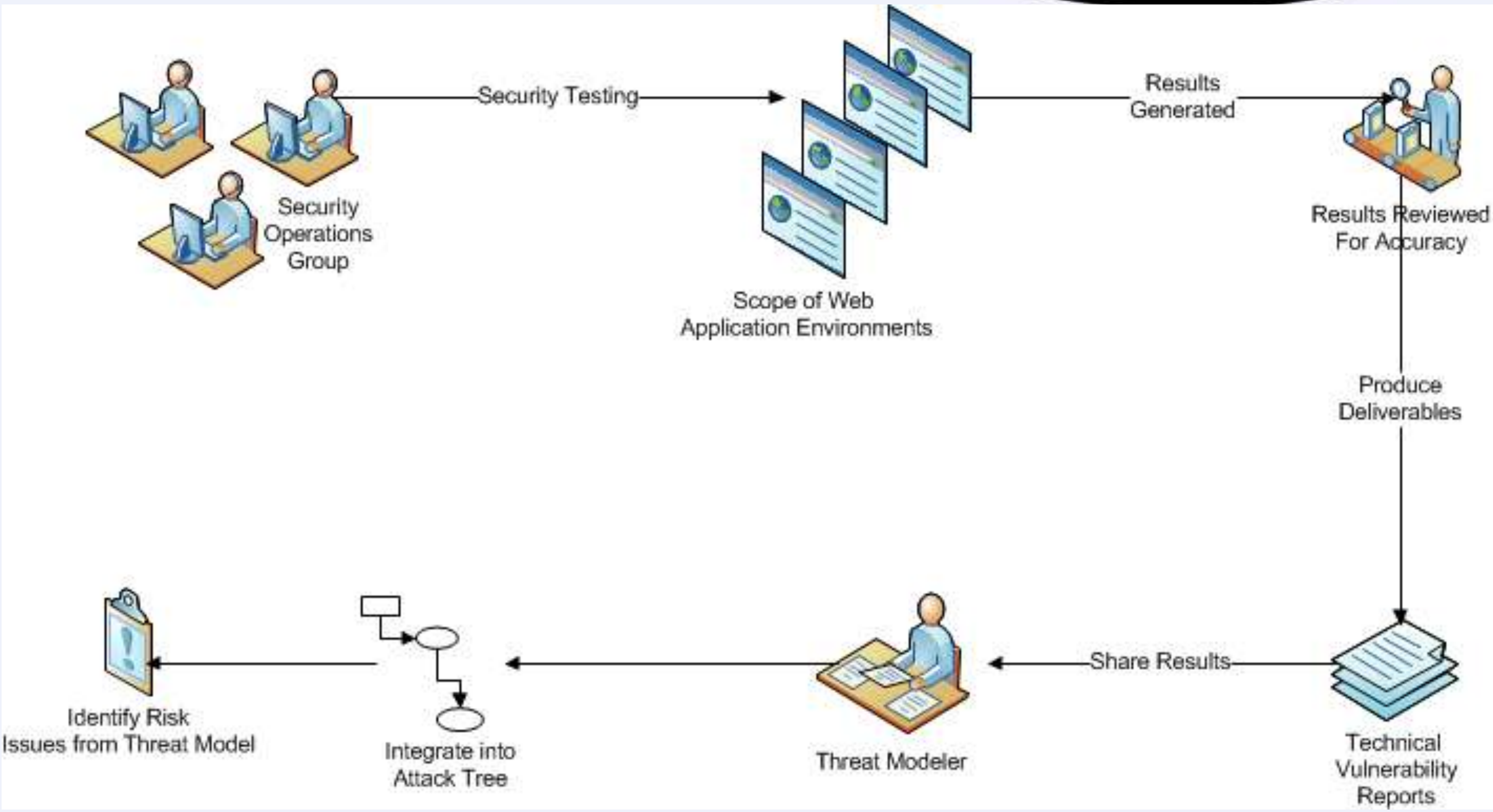
OWASP
The Open Web Application Security Project

STAGE V - Weakness and Vulnerabilities Analysis

Analyzing the weaknesses and vulnerabilities of web application security controls

Identifying Weaknesses & Vulnerabilities

OWASP
The Open Web Application Security Project



Mapping/ Prioritizing Vulnerabilities to Application Asset Targets

- Absolute Path Traversal (CWE-36)
- Cross-site scripting (XSS) (CWE-79)
- Cross-Site Request Forgery (CSRF) (CWE-352)
- CRLF Injection (CWE-93)
- Error Message Information Leaks (CWE-209)
- Format string vulnerability (CWE-134)
- Hard-Coded Password (CWE-259)
- Insecure Default Permissions (CWE-276)
- Integer overflow (wrap or wraparound) (CWE-190)
- OS Command Injection (shell metacharacters) (CWE-78)
- PHP File Inclusion (CWE-98)
- Plaintext password Storage (CWE-256)
- Race condition (CWE-362)
- Relative Path Traversal (CWE-23)
- SQL injection (CWE-89)
- Unbounded Transfer ('classic buffer overflow') (CWE-120)
- UNIX symbolic link (symlink) following (CWE-61)
- Untrusted Search Path (CWE-426)
- Weak Encryption (CWE-326)
- Web Parameter Tampering (CWE-472)

Vulnerabilities that affect both Design and Coding Flaws

OWASP
The Open Web Application Security Project

- **Design-Related**
  - High Algorithmic Complexity (CWE-407)
  - Origin Validation Error (CWE-346)
  - Small Space of Random Values (CWE-334)
  - Timing Discrepancy Information Leak (CWE-208)
  - Unprotected Windows Messaging Channel ('Shatter') (CWE-422)
  - Inherently Dangerous Functions, e.g. gets (CWE-242)
  - Logic/Time Bomb (CWE-511)
- **Low-level coding**
  - Assigning instead of comparing (CWE-481)
  - Double Free (CWE-415)
  - Null Dereference (CWE-476)
  - Unchecked array indexing (CWE-129)
  - Unchecked Return Value (CWE-252)
  - Path Equivalence - trailing dot - 'file.txt.‘ (CWE-42)
- **Newer languages/frameworks**
  - Deserialization of untrusted data (CWE-502)
  - Information leak through class cloning (CWE-498)
  - .NET Misconfiguration: Impersonation (CWE-520)
  - Passing mutable objects to an untrusted method (CWE-375)
- **Security feature failures**
  - Failure to check for certificate revocation (CWE-299)
  - Improperly Implemented Security Check for Standard (CWE-358)
  - Failure to check whether privileges were dropped successfully (CWE-273)
  - Incomplete Blacklist (CWE-184)
  - Use of hard-coded cryptographic key (CWE-321)

MITRE

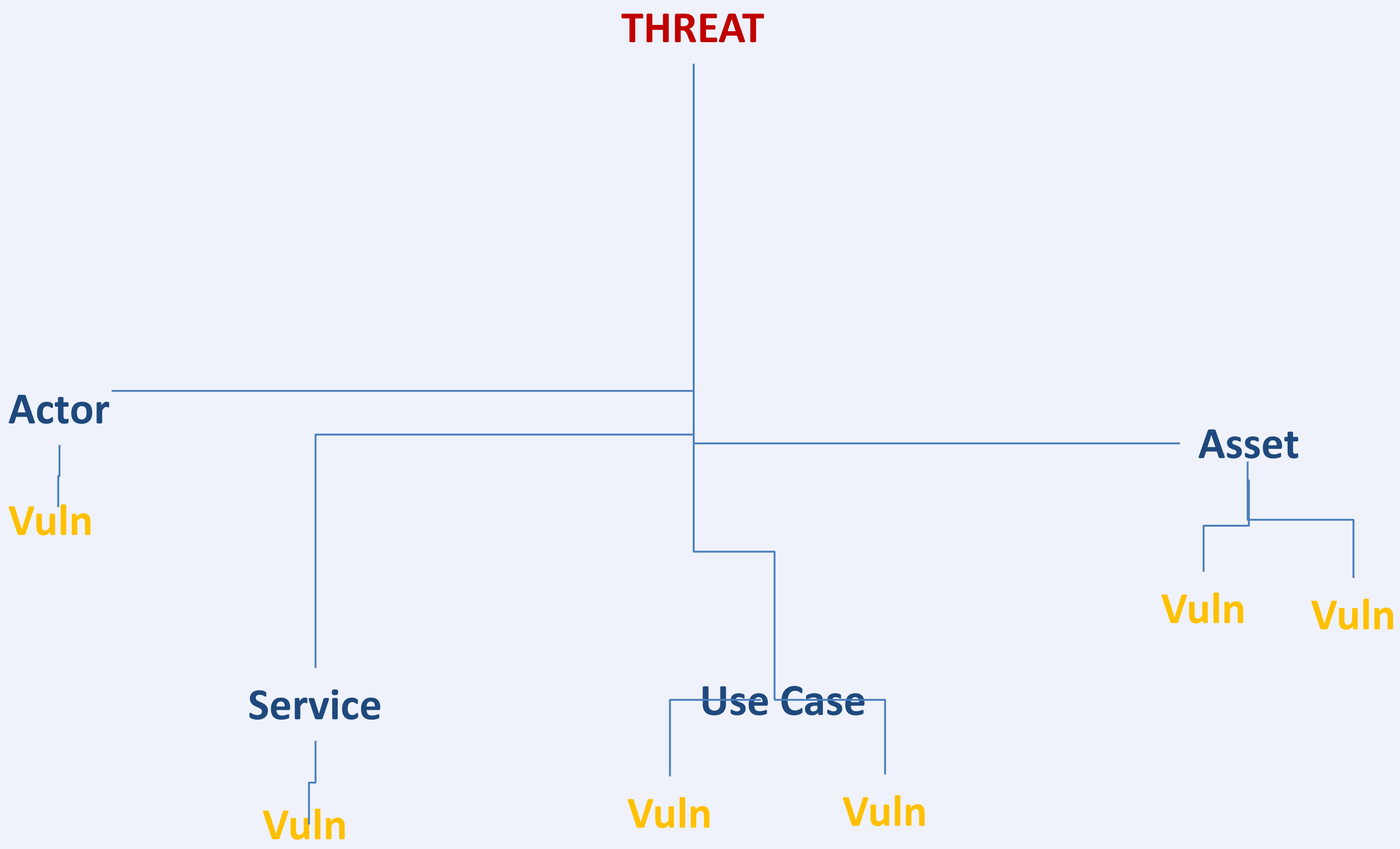
… and about 550 more

Identifying & Classifying Vulnerabilities

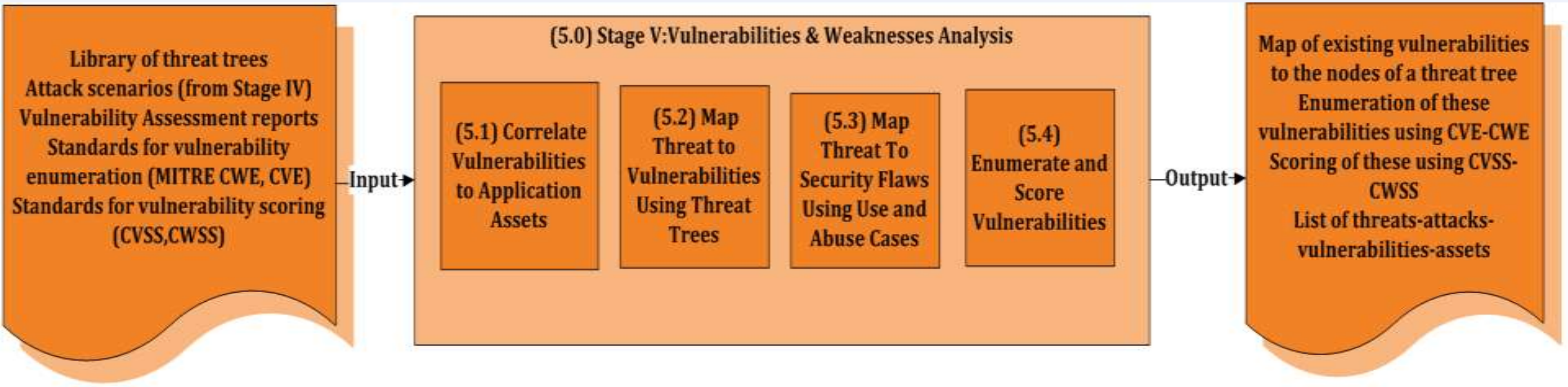
OWASP
The Open Web Application Security Project

- Easiest part of PASTA as most places have vulnerability detection capabilities
- More advance form of this stage looks beyond vulnerabilities identified by configuration gaps, insecure versioning, missing patches, known vulns
- Advance Stage V looks at design flaws
  - Should have actually been caught under Stage III
- Integration w/ SOC or those running vuln detection is preferable.
  - Request specific vulnerabilty checks based upon threat analysis

Simple Threat Tree Under PASTA's Stage V



Stage 5 : Vulnerability Analysis
Parallels SDLC DEVELOP & TEST Phase



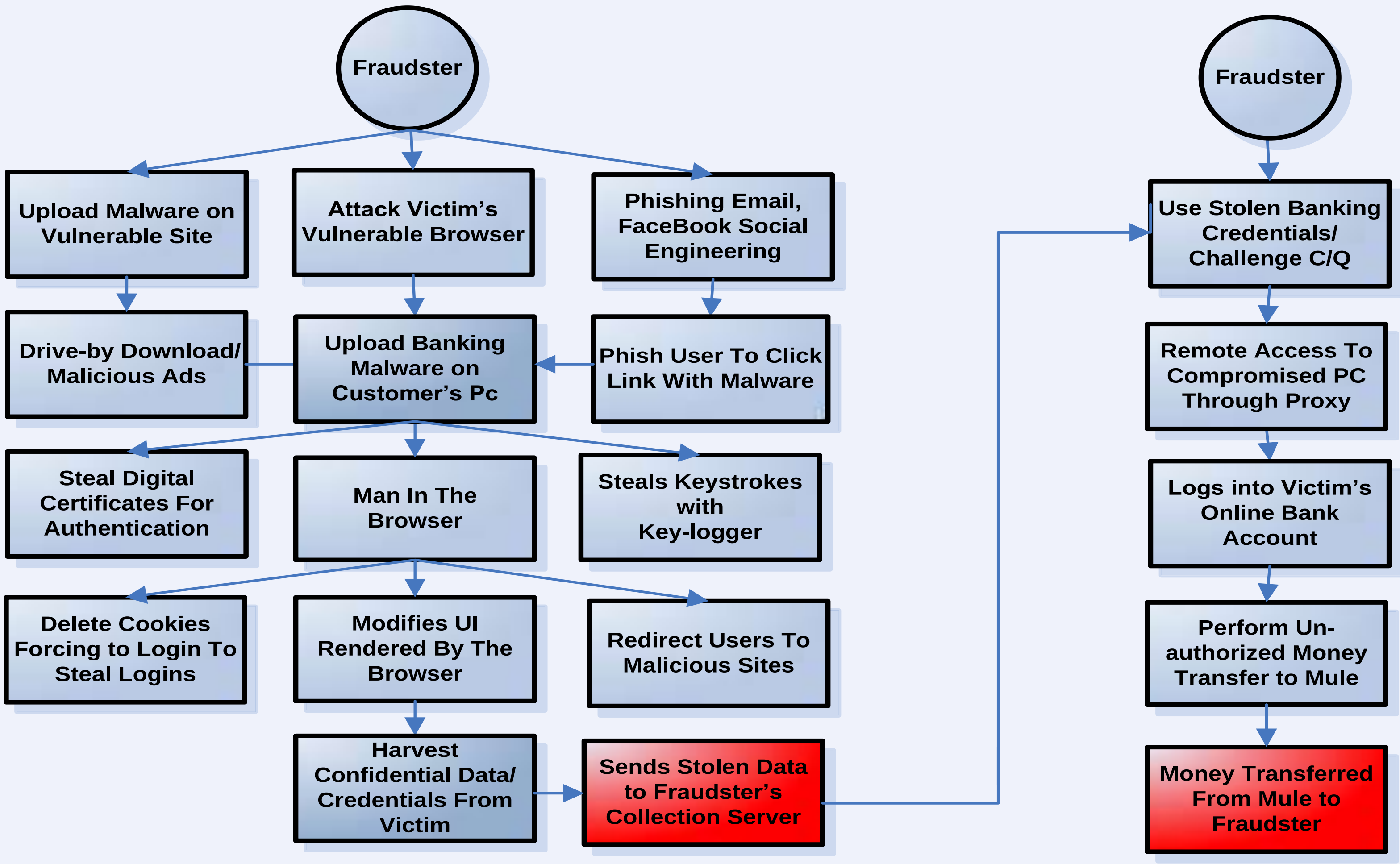
OWASP
The Open Web Application Security Project

STAGE VI

Model The Attacks/Exploits

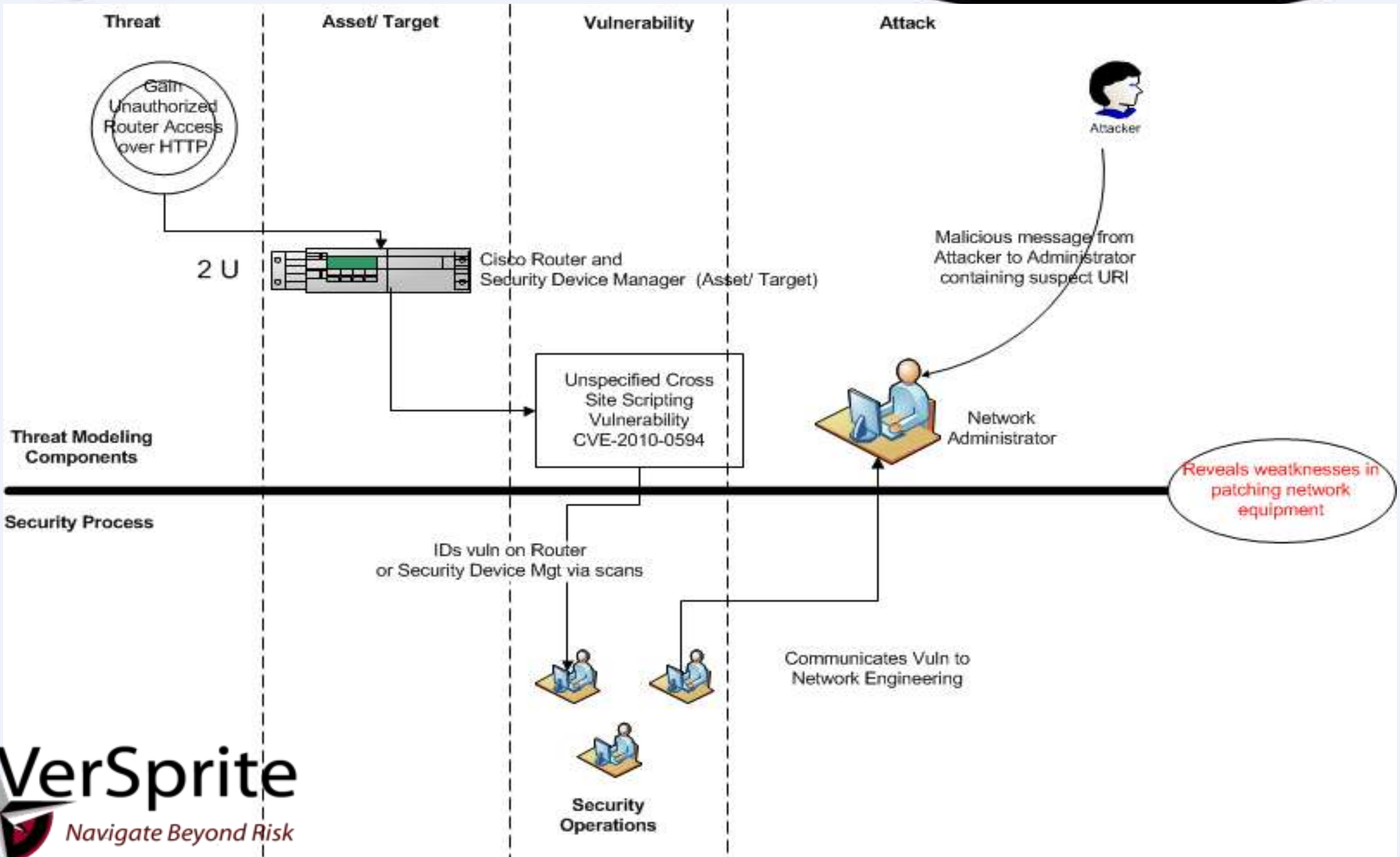
Analysis Of Attacks Using Attack Trees

OWASP
The Open Web Application Security Project



Identifying Attack Surfaces & Vectors

OWASP
The Open Web Application Security Project



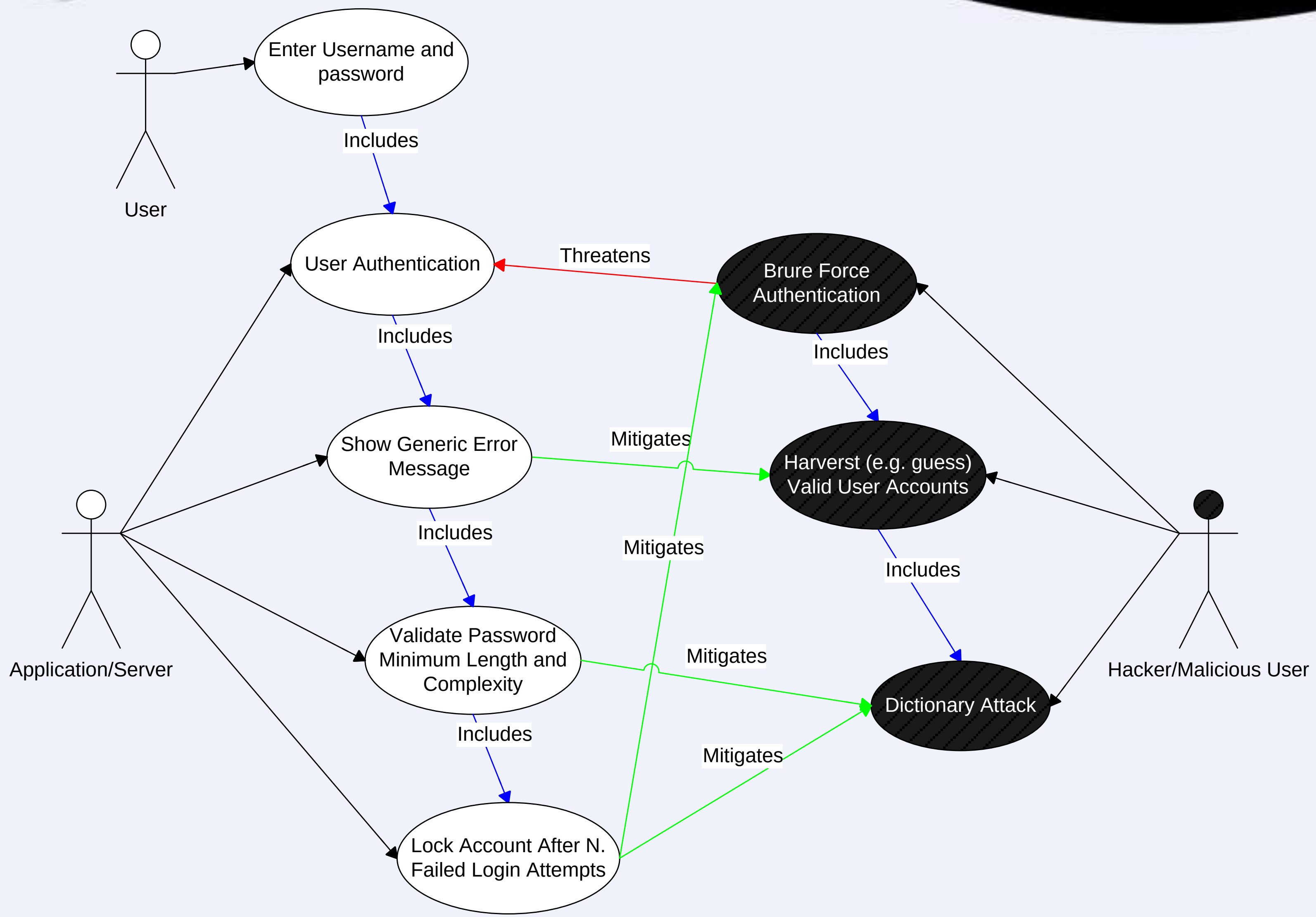
VerSprite
Navigate Beyond Risk

Banking Perspective: Attack Vectors via Malware Agents

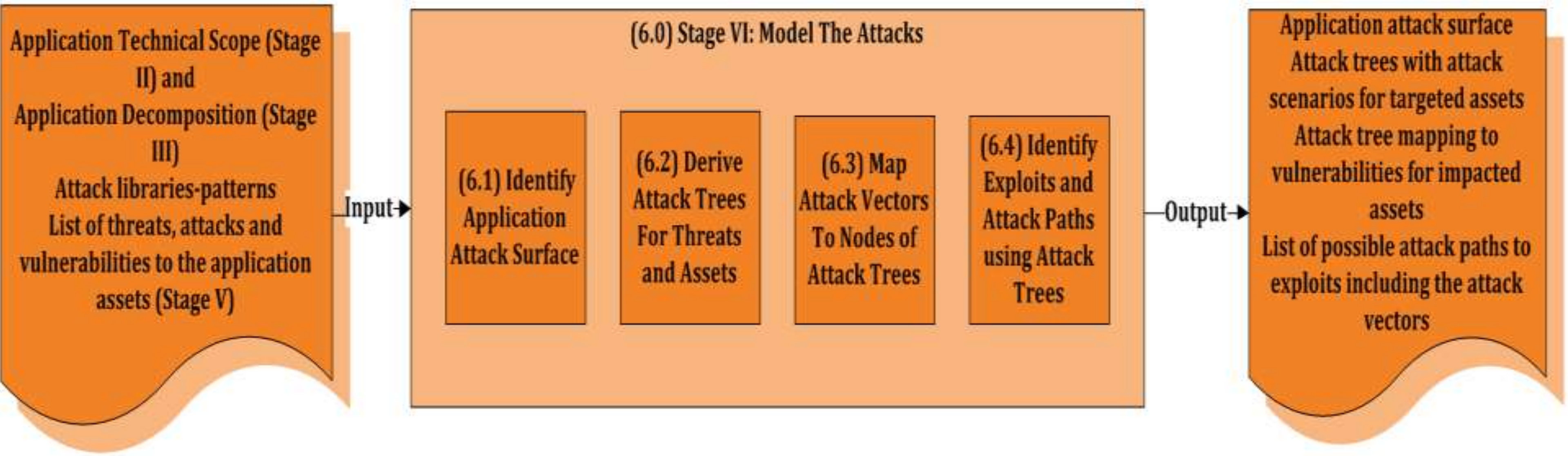
OWASP
The Open Web Application Security Project

Trojan	Phishing	Drive-by Download	Malicious Web Link	Malicious Ad	Virus Infection	HTTP Injection	Browser Redirect	Form Grabbing	Credential Theft	Keystroke Logging	By Pass MFA	Screen Capture/Video	Certificate Theft	Install Backdoor	Instant Message	Real-Time	Out of Band	Automated	Manu
	Infection Method					Attack Capabilities										Timing		Type	
MB- MitB MM-MitM B-Both O-Other						MB	MM	B	B	B	B	O	O	O	O				
ZeuS	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
SpyEye	*	*	*	*	*	*	*	*	*	*	*	*		*		*	*	*	*
InfoStealer	*	*	*	*	*	*		*	*	*	*	*	*	*			*		*
Silent Banker	*	*	*	*	*	*	*	*		*	*	*	*	*		*	*		*
URLZone	*	*	*	*	*	*		*		*	*	*		*		*	*	*	*
Clampi/Bugat/Gozi	*	*	*	*	*	*				*							*		*
Haxdoor	*	*	*	*	*	*		*		*				*			*		*
Limbo	*	*	*	*	*	*		*		*	*			*			*		*

Rise of Countermeasures from Attack Enumeration



Stage 6 : Attack Modeling
Parallels SDLC TESTING Phase



OWASP
The Open Web Application Security Project

STAGE VII - Risk And Impact Analysis:

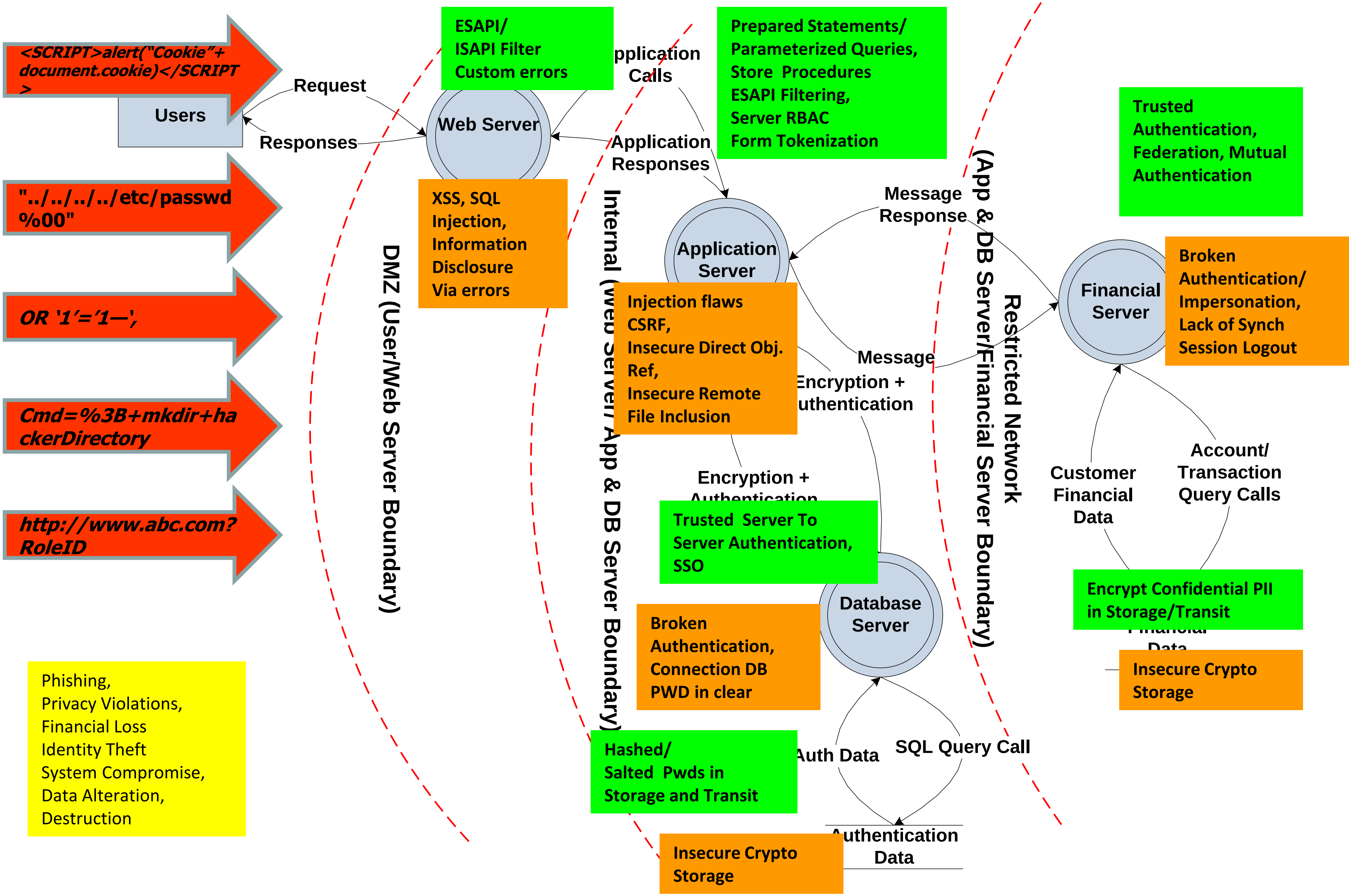
Impact Analysis, Residual Risk, and Countermeasure Development

Exploitation fosters Countermeasures

OWASP
The Open Web Application Security Project

- Unacceptable risks give way to countermeasure development
- Develop countermeasures based upon the net risk of an application environment at multiple levels
  - Baseline configuration
  - Design and programmatic controls
  - 3rd party software/ COTS

Countermeasure Development



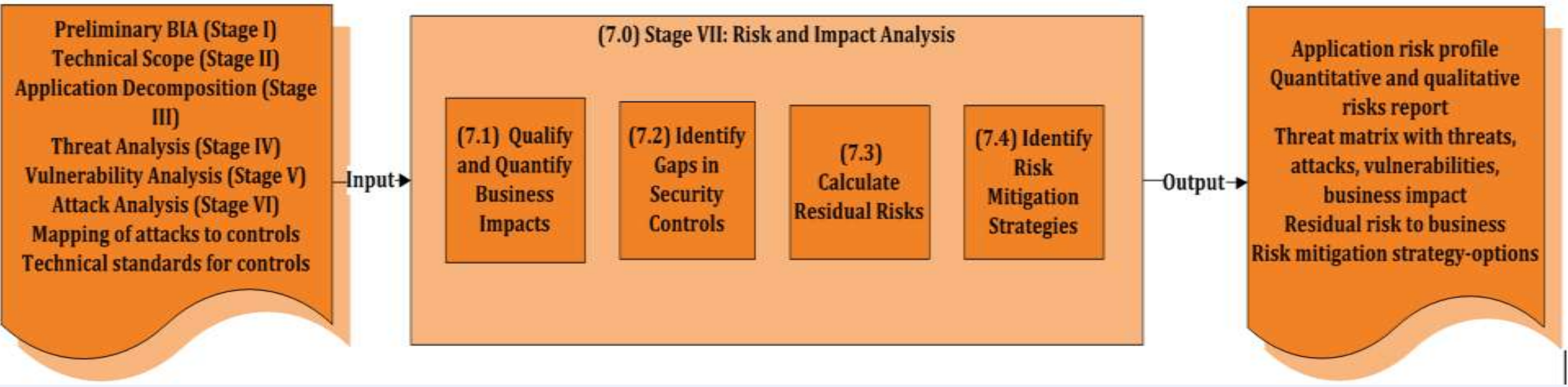
The PASTA™ Recipe For Risk Analysis of Web Apps

OWASP
The Open Web Application Security Project

- Remediate in commensuration to identified Risk
- Risk !=t * v * i
- Risk! = t * v * i * p
- $[(t_p * v_p)/c] * i = R_{risk}$
- Attack simulation enhances (*p*) probability coefficients
- Considers both inherent countermeasures & those to be developed
- Focused on minimizing risks to applications and associated impacts to business

Right Amount of Countermeasures

Stage 7 : Risk Analysis
Parallels SDLC MAINTAINANCE Phase



VerSprite
Navigate Beyond Risk

The Beneficiaries of PASTA™

OWASP
The Open Web Application Security Project

- **Business managers** can incorporate which security requirements that impact business
- **Architects** understand security/design flaws and how countermeasure protect data assets
- **Developers** understand how software is vulnerable and exposed
- **Testers** can use abuse cases to security tests of the application
- **Project managers** can manage security defects more efficiently
- **CISOs** can make informed risk management decisions; leverage maturity modeling (SAMM) to map progress

VerSprite
Navigate Beyond Risk

Q&

QUESTIONS
ANSWERS

Contact Me:
tonyuv@versprite.com
tonyuv@owasp.org
@versprite