KORNERSTONE
Training. Makes a differece.
a TRAINOCATE company

Certified Cloud Security Professional, CCSP®

D4 - Cloud Application Security

Domain 4 –
Cloud Application Security Domain

Key Areas of Knowledge (1)

A. Recognize the need for Training and Awareness in Application Security

- A.1 Cloud Development Basics (e.g., RESTful)
- A.2 Common Pitfalls
- A.3 Common Vulnerabilities (e.g. OWASP Top 10)

B. Understand Cloud Software Assurance and Validation

- B.1 Cloud-based Functional Testing
- B.2 Cloud Secure Development Lifecycle
- B.3 Security Testing (e.g., SAST, DAST, Pen Testing)

Key Areas of Knowledge (2)

C. Use Verified Secure Software

- C.1 Approved API
- C.2 Supply-Chain Management
- C.3 Community Knowledge

D. Comprehend the Software Development Life-Cycle (SDLC) Process

- D.1 Phases & Methodologies
- D.2 Business Requirements
- D.3 Software Configuration Management & Versioning

Key Areas of Knowledge (3)

E. Apply the Secure Software Development Life-Cycle

- E.1 Common Vulnerabilities (e.g., SQL Injection, XSS, XSRF, Direct Object
- Reference, Buffer Overflow)
- E.2 Cloud-Specific Risks
- E.3 Quality of Service
- E.4 Threat Modeling

Key Areas of Knowledge (4)

F. Comprehend the Specifics of Cloud Application Architecture

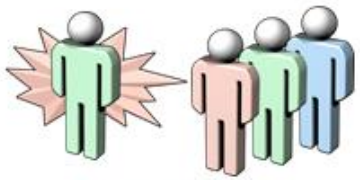
- F.1 Supplemental Security Devices (e.g., WAF, DAM, XML firewalls, API gateway)
- F.2 Cryptography (e.g. TLS, SSL, IPSEC)
- F.3 Sandboxing
- F.4 Application Virtualization

G. Design Appropriate Identity and Access Management (IAM) Solutions

- G.1 Federated Identity
- G.2 Identity Providers
- G.3 Single Sign-On
- G.4 Multi-factor Authentication

A. Recognize the need for Training and Awareness in Application Security

Challenges & Complexities



Attackers vs. Defenders

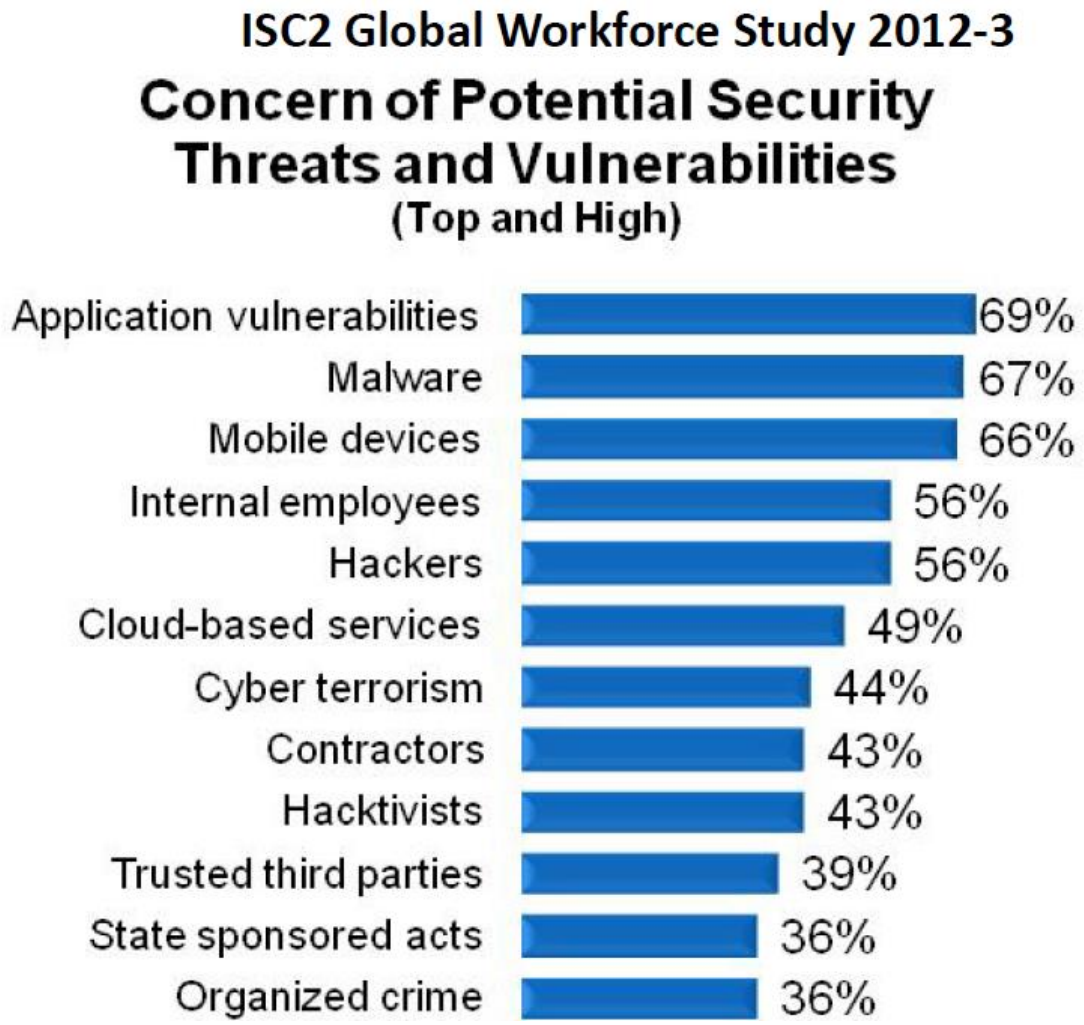
Security vs. Usability

Security as an afterthought

Iron Triangle Constraints

What advantages does the attacker have over the defender?

Security Trends



Top Concerns:

-App Sec

-Malware

-Mobile Devices

Why application rank the highest?

Security – Not just for Networks

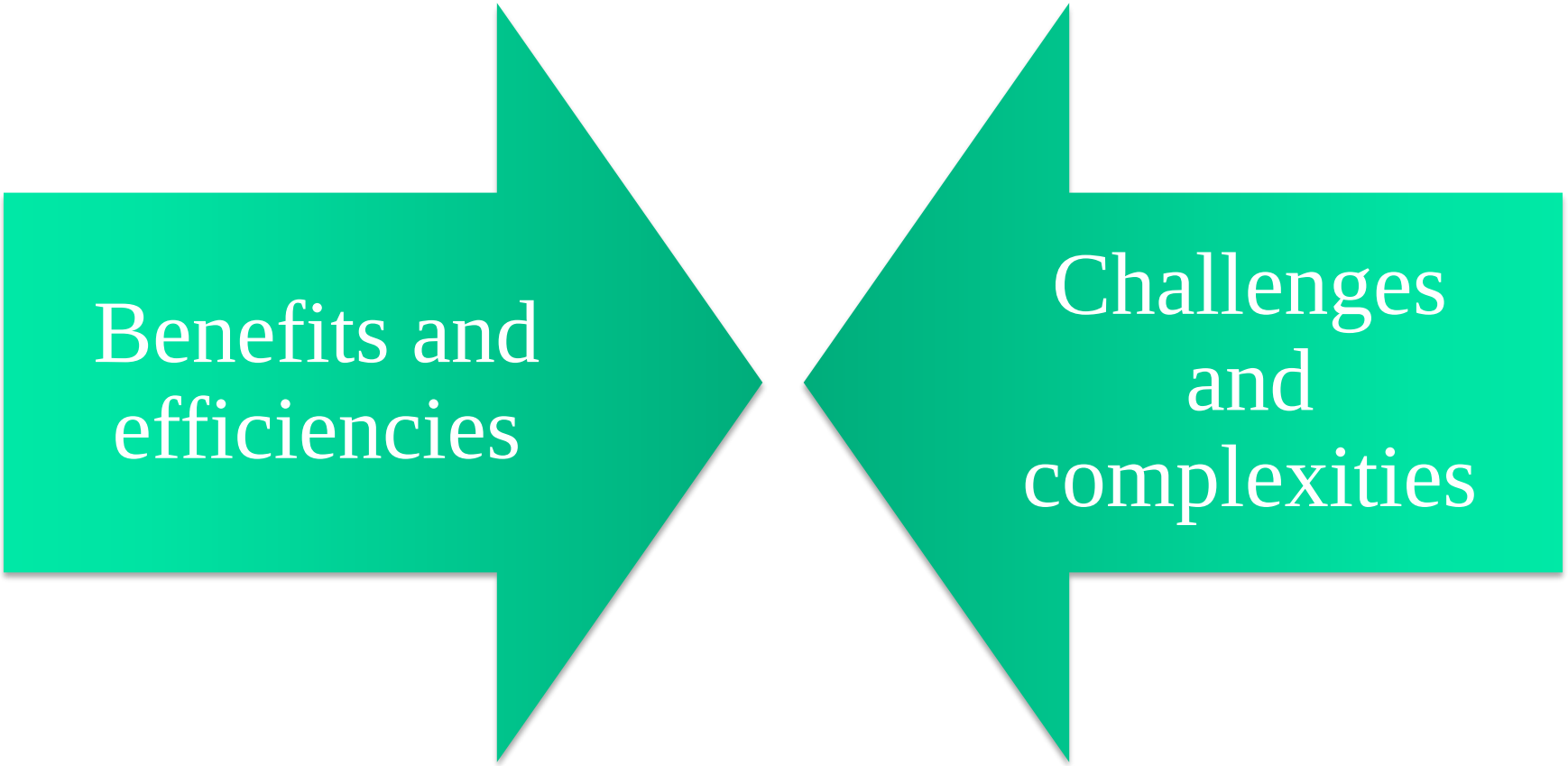
- Criminals are targeting software.
- Hackers know that you have firewalls – Not convenient to hack the network.
- Not enough to *just* build a robust system. You must include security.
- Hackers are targeting a new way to 'hack' into your systems
  - *The new way is through applications.*

Why Applications?

People are using software in every part of their lives

Additional loss of control when outsourcing development work

Introduction



Cloud Development Basics

What is the impact?

- Data become public and distributed
- Employee can access your application
- Manipulated by an outsider
- Process or function manipulated by outsider
- Process or function failed to expected results
- Application was unavailable

- Discuss with System Owner
- Information Gathering Exercise on CIA

Application Programming Interface

- Representational State Transfer (REST)
- Simple Object Access Protocol (SOAP)

- Web Services : REST VS SOAP
- IAM (OAuth, SAML, Token)
- API Gateway
- JSON vs XML

Common Pitfalls

- **On Premise Does Not Always Transfer (and Vice Versa)**
- **Not All Apps Are "Cloud Ready"**
- **Lack of Training and Awareness**
- **Documentation and Guidelines**
- **Complexities for Integration**
- **Overarching Challenges**

On Premise Does Not Always Transfer (and Vice Versa)

- Performance & Functionality
- Not Cloud-based in development
- Maintenance & Support

- Not all “forklifted” = migrating to Cloud
- Porting to Cloud introduce additional change, e.g. IaaS -> PaaS

Not All Apps are “Cloud-Ready”

- Development & Testing
- Challenging:
  - Confidentiality and integrity verified and assured system
  - Security & Regulatory Restrictions, e.g. COBOL
  - Highly secure systems and hardened security controls

Lack of Training and Awareness

- Cloud based development is different
- Maybe unfamiliar with Cloud
- DevOps / Agile / PaaS

Documentation and Guidelines

- Fast development lifecycle
- Open Source
- Fast cloud services release
- Outdated documentation

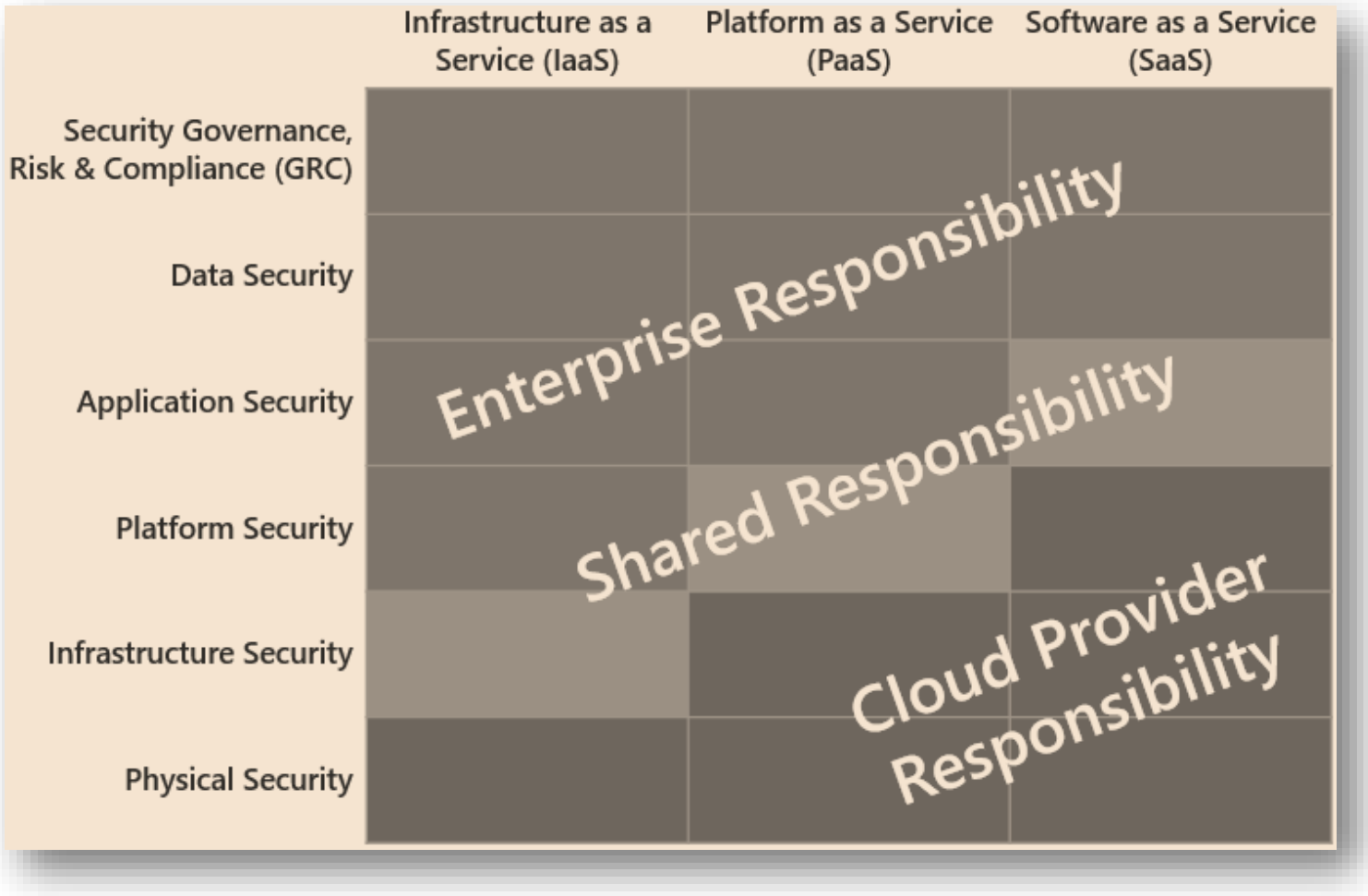
Complexities for Integration

- Cloud <-> Ground (on-premise)
- Cloud API
- Cloud provider manages infrastructure, applications and integration platforms
- Troubleshooting, e.g. Log analytics

Overaching Challenges

- Two key risks in cloud application
  - Multitenancy Risk
  - 3rd party administration
- Public, Private, Community, Hybrid
- IaaS, PaaS, SaaS
- Security is different for different model

Security Responsibility for Cloud



Awareness of Encryption Dependencies

- At Rest, e.g. SAN, NAS, SSD, Hard-disk
- Data in Transit, SSL, HTTPS, Internet
- Data Obfuscation / Masking

- Compliance / Industry Certifications

D. Comprehend the Software Development Life-Cycle (SDLC) Process

Ice-Breaking Activity

- Work in teams of three.
- Your team will be assigned a phase of the software development life cycle: planning, designing, developing, acquiring, testing, deploying

As a team:

- Define the phase
- Identify at least one activity that can increase security during the phase
- Identify at least one activity that can reduce security during the phase

- • Prepare to share your work with the class

Secure SDLC Phases & Methodology

- Define: Business & Security Requirements
- Design: Threat Modeling, Secure Design
- Develop: Code Review, Unit Testing, Static Analysis
- Testing: Functional Tests, QA, Vulnerability Assessment, Dynamic Analysis

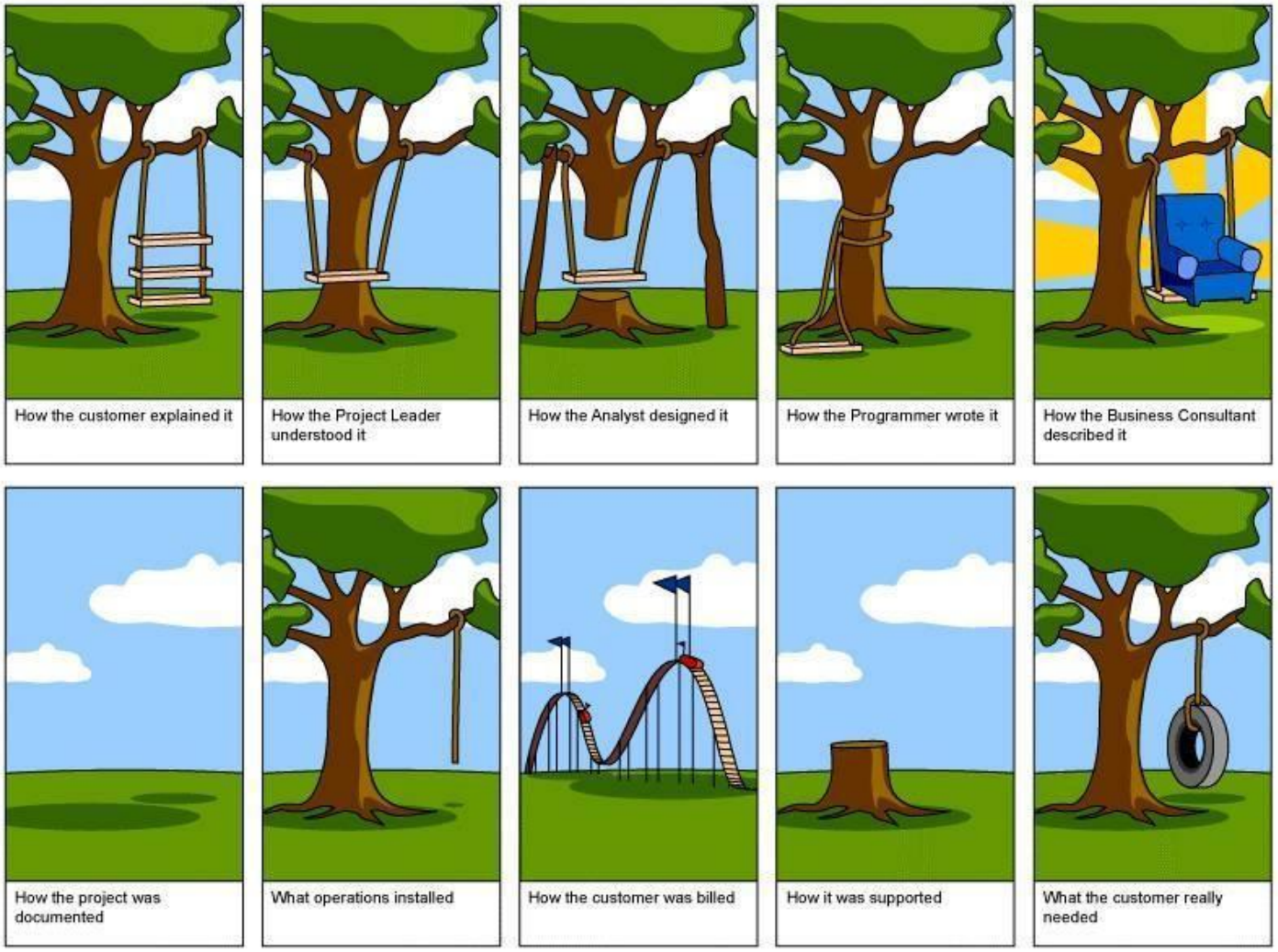
10 Key Secure Design Principles (for reference only)

1. Least Privilege
2. Separation of Duties
3. Defense-in-Depth
4. Fail Secure (Safe)
5. Economy of Mechanism
6. Complete Mediation
7. Session Management
8. Open Design
9. Least Common Mechanism
10. Leveraging Existing Components

Secure Operations Phase

- Dynamic Analysis
- Vulnerability Assessments and Penetration Testing
- Continuous Monitoring
- Activity Monitoring
- Web Application Firewall (WAF)

Secure Business Requirements



Software Security Testing

Criterea	Security Testing Approach	
	Black Box	White Box
Determination of root cause	Most likely to address the symptoms than the root cause.	Exact line of code or design issue causing the vulnerability can be identified.
Extent of code coverage	Limited as the analysis is behavioral; not all code paths may be covered.	Greater as the source code and configuration is available for review.
Detection of logic flaws	Not knowing the normal behavior of the software, anomalous behavior may not necessarily indicate flaws in logic.	The availability of design and architectural documents besides code can be used to detect logic flaws.
Issues with deployment	Assessment can be performed in pre- as well as post- deployment production or production-like simulated environment, giving insight into any potential issues after deployment.	Assessment is performed in pre-deployment environments usually providing limited issues pertaining to configuration and change management.

Software Configuration Management & Versioning

- Versioning
- Configuration Management
  - Puppet
  - Chef
  - Git
  - DevOps

E. Apply the Secure Software Development Life-Cycle

Common Vulnerabilities

- OWASP Top 10
- https://www.owasp.org/index.php/Top_10_2013-Top_10

OWASP Top 10 2013

OWASP Top 10 – 2010 (Previous)	OWASP Top 10 – 2013 (New)
A1 – Injection	A1 – Injection
A3 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A6 – Security Misconfiguration	A5 – Security Misconfiguration
A7 – Insecure Cryptographic Storage – Merged with A9 →	A6 – Sensitive Data Exposure
A8 – Failure to Restrict URL Access – Broadened into →	A7 – Missing Function Level Access Control
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<buried in A6: Security Misconfiguration>	A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards	A10 – Unvalidated Redirects and Forwards
A9 – Insufficient Transport Layer Protection	Merged with 2010-A7 into new 2013-A6

OWASP Top 10 Mobile (Reference Only)

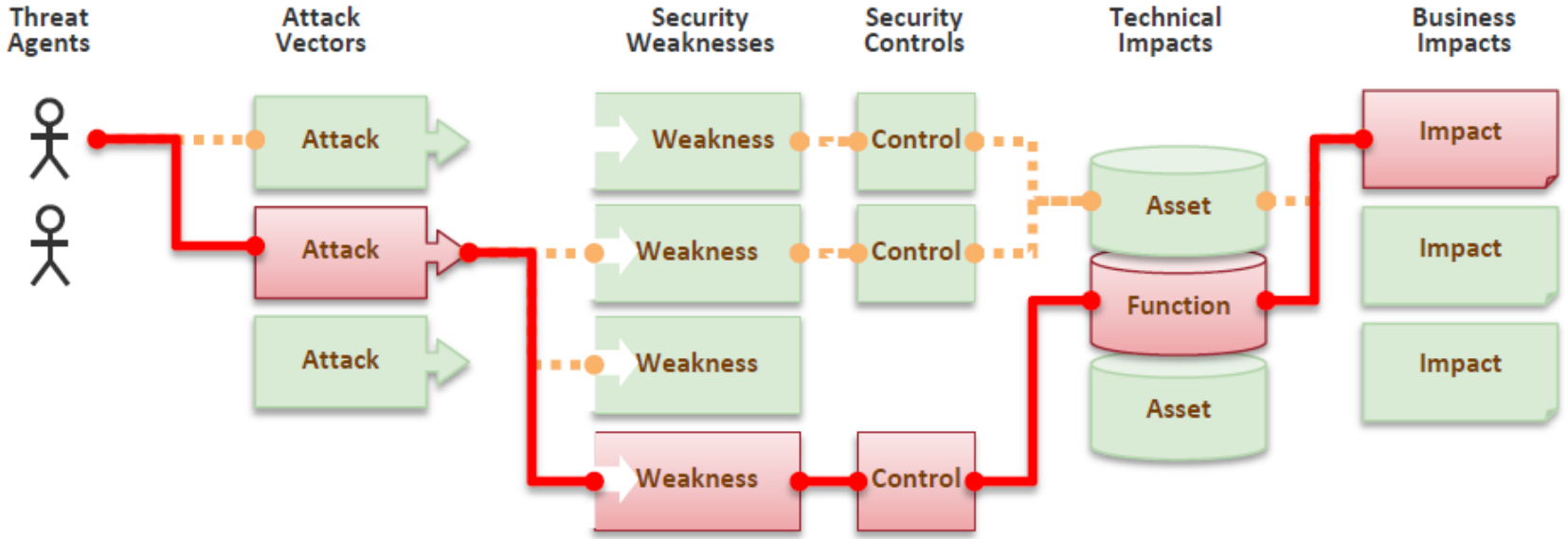
M1 - Improper Platform Usage	This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk.
M2 - Insecure Data Storage	This new category is a combination of M2 + M4 from Mobile Top Ten 2014. This covers insecure data storage and unintended data leakage.
M3 - Insecure Communication	This covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.
M4 - Insecure Authentication	This category captures notions of authenticating the end user or bad session management. This can include: • Failing to identify the user at all when that should be required • Failure to maintain the user's identity when it is required • Weaknesses in session management
M5 - Insufficient Cryptography	The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that probably belongs in M2. This category is for issues where cryptography was attempted, but it wasn't done correctly.
M6 - Insecure Authorization	This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.). If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure.
M7 - Client Code Quality	This was the "Security Decisions Via Untrusted Inputs", one of our lesser-used categories. This would be the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.
M8 - Code Tampering	This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.
M9 - Reverse Engineering	This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.
M10 - Extraneous Functionality	Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.

https://www.owasp.org/index.php/Mobile_Top_1

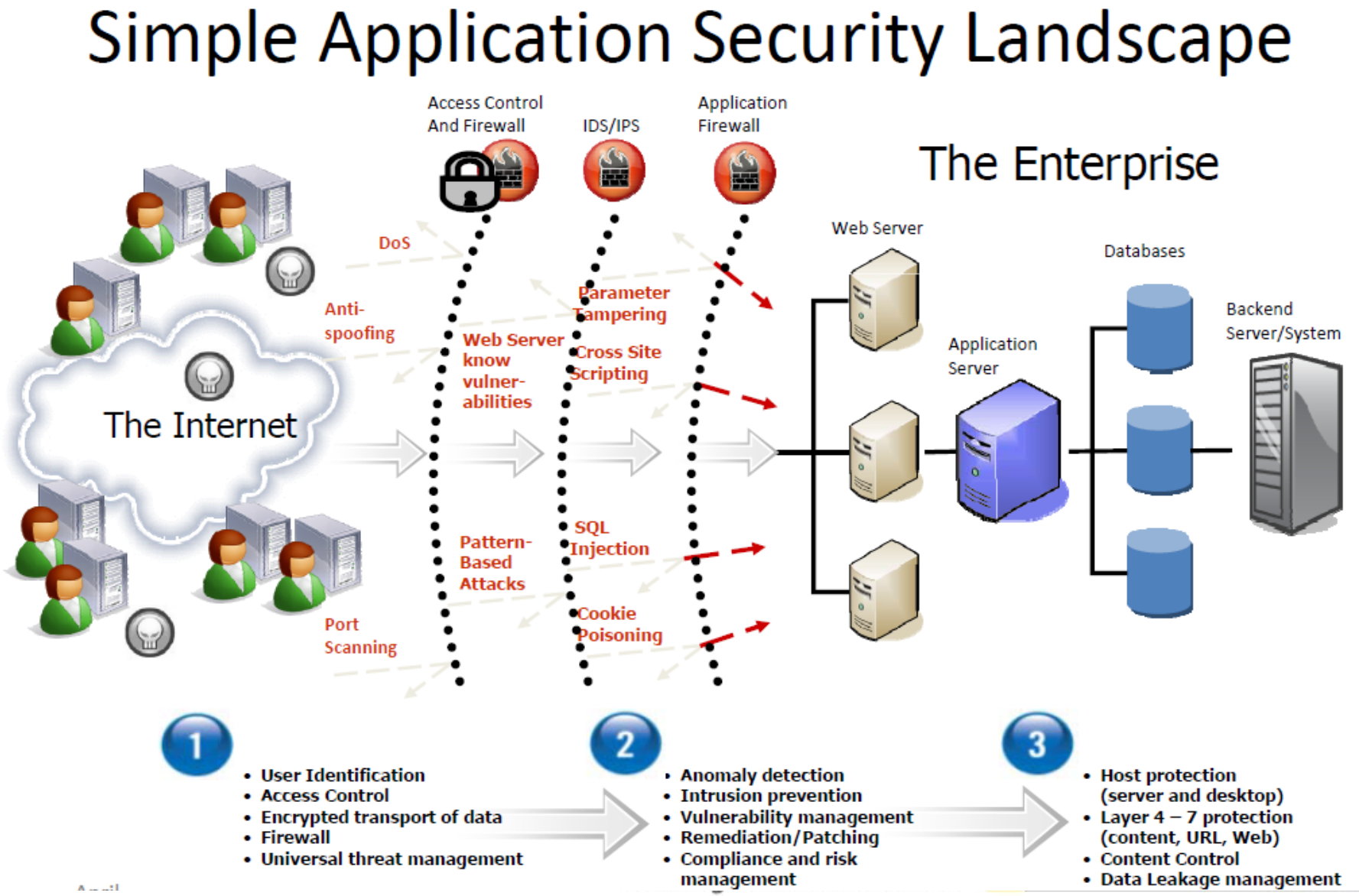
Web Application Security

What Are Application Security Risks?

Attackers can potentially use many different paths through your application to do harm to your business or organization. Each of these paths represents a risk that may, or may not, be serious enough to warrant attention.



Simple Web App Example



OWASP A1: Injection

Example Attack Scenarios

Scenario #1: The application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in her browser to send: **' or '1'='1**. For example:

http://example.com/app/accountView?id=' or '1'='1

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify data or even invoke stored procedures.

Controls:

- Input validation
- Input Encoding
- Dynamic SQL prevented
- Parameterized queries
- Generic error messages
- Redirect to generic error page
- Replace single quotes with double
- Remove unused functions / procedures
- Use principle of least privilege

OWASP A2: Broken Authentication and Session Management

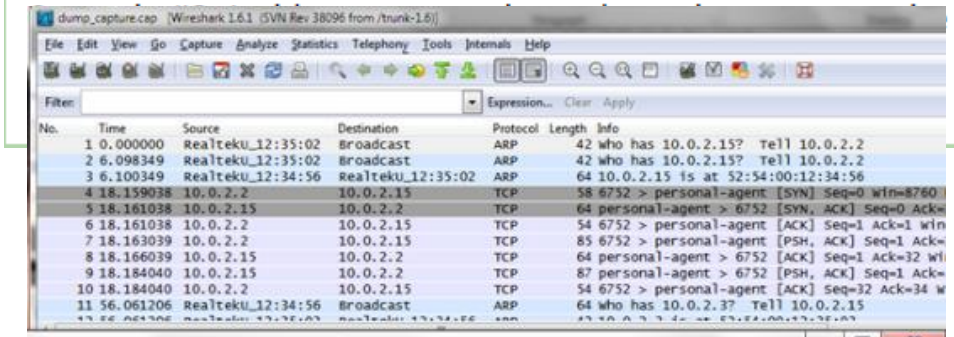
Example Attack Scenarios

Scenario #1: Airline reservations application supports URL rewriting, putting session IDs in the URL:

http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii

An authenticated user of the site wants to let his friends know about the sale. He e-mails the above link without knowing he is also giving away his session ID. When his friends use the link they will use his session and credit card.

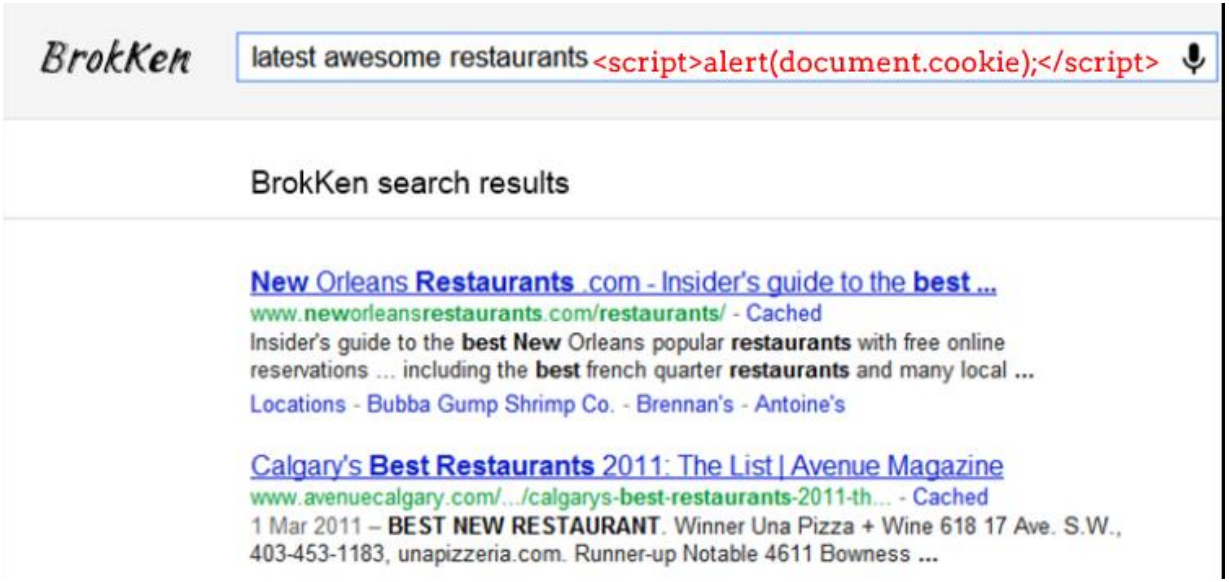
Scenario #2: Application's timeouts aren't set properly. User uses a public computer to access site. Instead of selecting "logout" the user simply closes the browser tab and walks away. Attacker uses the same browser an hour later, and that browser is still authenticated.



Controls:

- Only use the built-in, session-management mechanism
- Do not accept new, preset, or invalid session identifiers
- Limit or rid your code of custom cookies for authentication or session management
- Use a single authentication mechanism
- Do not allow the login process to start from an unencrypted page
- Regenerate a new session upon successful authentication
- Ensure every page has a logout link
- Use a timeout period
- Use only strong, ancillary, authentication functions
- Do not expose any session identifiers in URLs or logs
- Check the old password when the user changes to a new password

OWASP A3: Cross-Site Scripting (XSS)



Controls:

- Input validation / filtration
- Output handling / encoding
- Request validation
- Use text properties
- Usage of secure libraries
- Disallow active scripting

Once XSS found on a site, Hackers take it and send to unaware users

http://brokken.com/search.aspx?query=Latest%20awesome%20restaurants<script>alert(document.cookie);</script>

OWASP A4: Insecure Direct Object Reference

Example Attack Scenario

The application uses unverified data in a SQL call that is accessing account information:

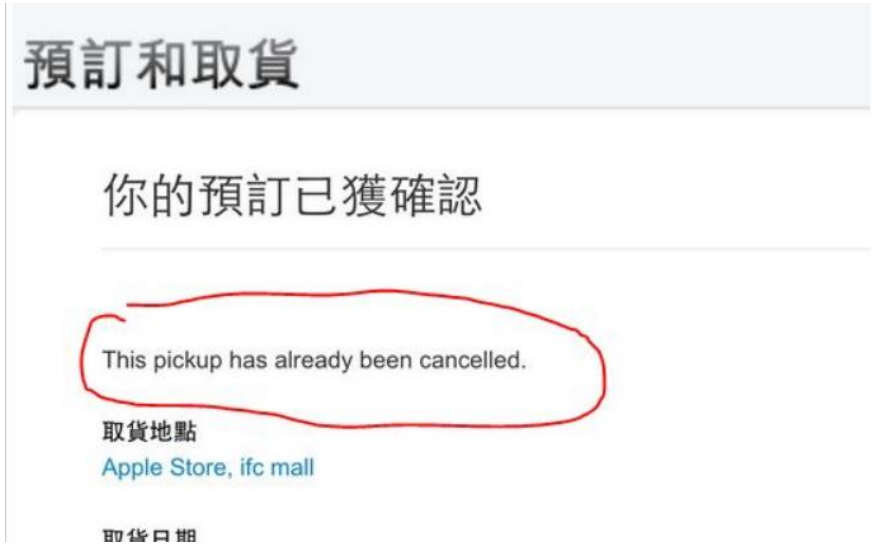
```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt =
connection.prepareStatement(query , ... );

pstmt.setString( 1, request.getParameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

The attacker simply modifies the 'acct' parameter in her browser to send whatever account number she wants. If not properly verified, the attacker can access any user's account, instead of only the intended customer's account.

```
http://example.com/app/accountInfo?acct=notmyacct
```

Controls:

- Reference mapping /indexing
- Encapsulate internal objects
- Encryption/hashing
- Input validation
- Complete mediation



OWASP A5: Security Misconfiguration

Example Attack Scenarios

Scenario #1: The app server admin console is automatically installed and not removed. Default accounts aren't changed. Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over.

Scenario #2: Directory listing is not disabled on your server. Attacker discovers she can simply list directories to find any file. Attacker finds and downloads all your compiled Java classes, which she decompiles and reverse engineers f... your custom code. She then finds a serious access cor... flaw in your application.

Scenario #3: App server configuration allows stack tra... be returned to users, potentially exposing underlying ... Attackers love the extra information error messages p...

Scenario #4: App server comes with sample applicatio... are not removed from your production server. Said sa... applications have well known security flaws attackers ... to compromise your server.

How Do I Prevent This?

The primary recommendations are to establish all of the following:

1. A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically (with different passwords used in each environment). This process should be automated to minimize the effort required to setup a new secure environment.
2. A process for keeping abreast of and deploying all new software updates and patches in a timely manner to each deployed environment. This needs to include **all code libraries as well (see new A9)**.
3. A strong application architecture that provides effective, secure separation between components.
4. Consider running scans and doing audits periodically to help detect future misconfigurations or missing patches.

OWASP A6: Sensitive Data Exposure

Example Attack Scenarios

Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this means it also decrypts this data automatically when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text. The system should have encrypted the credit card numbers using a public key, and only allowed back-end applications to decrypt them with the private key.

Scenario #2: A site simply doesn't use SSL for all authenticated pages. Attacker simply monitors network traffic (like an open wireless network), and steals the user's session cookie. Attacker then replays this cookie and hijacks the user's session, accessing the user's private data.

Scenario #3: The password database uses unsalted hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password file. All of the unsalted hashes can be exposed with a rainbow table of precalculated hashes.

How Do I Prevent This?

The full perils of unsafe cryptography, SSL usage, and data protection are well beyond the scope of the Top 10. That said, for all sensitive data, do all of the following, at a minimum:

1. Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all sensitive data at rest and in transit in a manner that defends against these threats.
2. Don't store sensitive data unnecessarily. Discard it as soon as possible. Data you don't have can't be stolen.
3. Ensure strong standard algorithms and strong keys are used, and proper key management is in place. Consider using FIPS 140 validated cryptographic modules.
4. Ensure passwords are stored with an algorithm specifically designed for password protection, such as bcrypt, PBKDF2, or scrypt.
5. Disable autocomplete on forms collecting sensitive data and disable caching for pages that contain sensitive data.

OWASP A7: Missing Function Level Access Control

Example Attack Scenarios

Scenario #1: The attacker simply force browses to target URLs. The following URLs require authentication. Admin rights are also required for access to the "admin_getappInfo" page.

http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo

If an unauthenticated user can access either page, that's a flaw. If an authenticated, non-admin, user is allowed to access the "admin_getappInfo" page, this is also a flaw, and may lead the attacker to more improperly protected admin pages.

Scenario #2: A page provides an 'action 'parameter to specify the function being invoked, and different actions require different roles. If these roles aren't enforced, that's a flaw.

How Do I Prevent Forced Access?

Your application should have a consistent and easy to analyze authorization module that is invoked from all of your business functions. Frequently, such protection is provided by one or more components external to the application code.

1. Think about the process for managing entitlements and ensure you can update and audit easily. Don't hard code.
2. The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific roles for access to every function.
3. If the function is involved in a workflow, check to make sure the conditions are in the proper state to allow access.

OWASP A8: CSRF

Example Attack Scenario

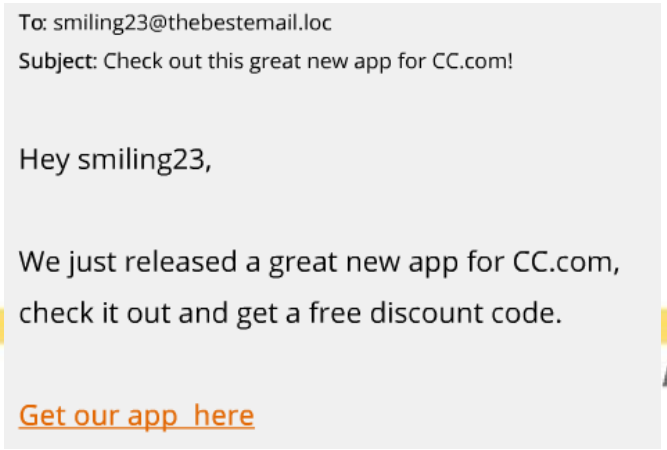
The application allows a user to submit a state changing request that does not include anything secret. For example:

```
http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243
```

So, the attacker constructs a request that will transfer money from the victim's account to the attacker's account, and then embeds this attack in an image request or iframe stored on various sites under the attacker's control:

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#"
width="0" height="0" />
```

If the victim visits any of the attacker's sites while already authenticated to example.com, these forged requests will automatically include the user's session info, authorizing the attacker's request.


To: smiling23@thebestemail.loc

Subject: Check out this great new app for CC.com!

Hey smiling23,

We just released a great new app for CC.com, check it out and get a free discount code.

Get our app here

Controls (Users):

- Log off immediately
- Do not allow browsers or sites to save credentials
- Do not use the same browser to access sensitive applications and to surf freely

Controls (Developers):

- Add session-related information to the URL
- Use CAPTCHA to establish token IDs
- Randomize session tokens
- Do not use GET requests for sensitive data or to perform value transactions
- Don't trust validation on the client end alone
- Re-authenticate
- Log out explicitly
- Test the referrer tag

OWASP A10: Invalidated Redirects and Forwards

Example Attack Scenarios

Scenario #1: The application has a page called "redirect.jsp" which takes a single parameter named "url". The attacker crafts a malicious URL that redirects users to a malicious site that performs phishing and installs malware.

http://www.example.com/redirect.jsp?url=evil.com

Scenario #2: The application uses forwards to route requests between different parts of the site. To facilitate this, some pages use a parameter to indicate where the user should be sent if a transaction is successful. In this case, the attacker crafts a URL that will pass the application's access control check and then forwards the attacker to administrative functionality for which the attacker isn't authorized.

http://www.example.com/boring.jsp?fwd=admin.jsp

How Do I Prevent This?

Safe use of redirects and forwards can be done in a number of ways:

1. Simply avoid using redirects and forwards.
2. If used, don't involve user parameters in calculating the destination. This can usually be done.
3. If destination parameters can't be avoided, ensure that the supplied value is **valid**, and **authorized** for the user.

   It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL.

   Applications can use ESAPI to override the sendRedirect() method to make sure all redirect destinations are safe.

Ice-Breaking Activity

Match the threat with the corresponding symptom.

A. Broken authentication
B. CSRF
C. Failure to restrict URL
D. Improper error handling
E. Session attacks
F. Insecure Direct Object Reference
G. Injection flaws
H. Insufficient transport layer protection
I. XSS
J. Overflow
K. Race condition
L. Security misconfiguration
M. Insecure cryptographic storage
N. Unvalidated redirects and forwards
O. Malicious file execution

1. ___	A user account keeps getting locked out repeatedly.
2. ___	Database connection information in an include file (db.inc) has been disclosed, even though there is no direct menu access to the "includes" directory from the Web application.
3. ___	The search criteria input makes an alert box appear in the browser.
4. ___	An attacker copies data into contiguous computer memory locations.
5. ___	Even after the user is logged out, one can access his past interactions with the software.
6. ___	By manipulating the parameters, internal object representations are disclosed without any mediation.
7. ___	Even though the data is being stored in an encrypted form, an attacker has been able to see the data by placing a sniffer.
8. ___	A user who is authenticated into his bank account finds out after the fact that when he clicked on a link promising a sports store discount in an email he received, money from his bank was transferred.
9. ___	When the software fails, it presents the user with the full stack trace.
10. ___	The active directory data store can be enumerated by anyone because the query to search it is not being validated.
11. ___	The update to the sales price of an item is observed to have been overwritten by another sales person between the time you checked it and the time you used it.
12. ___	The key that is used for encryption is stored in clear text within a configuration file.
13. ___	An uploaded file's behavior does not match its file name extension.
14. ___	The service account information, including account name and password, is hard-coded within the code.
15. ___	Every time a user navigates to a particular Web page on her customer portal, a script executes that redirects her to a different Web site causing her a denial of service.

Cloud Specific Risks

- E.g. Encryption in PaaS, Logging in PaaS
- Application Isolation
- Dockers & Containers
- Quality of Service (QoS)
  - Service Limit
  - Malware Slow down whole system
  - Encryption degrading performance

Cloud Computing Top Threats 2013

1. Data Breaches
2. Data Loss
3. Account Hijacking
4. Insecure APIs
5. Denial of Service
6. Malicious Insiders
7. Abuse of Cloud Services
8. Insufficient Due Diligence
9. Shared Technology Issues

https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf

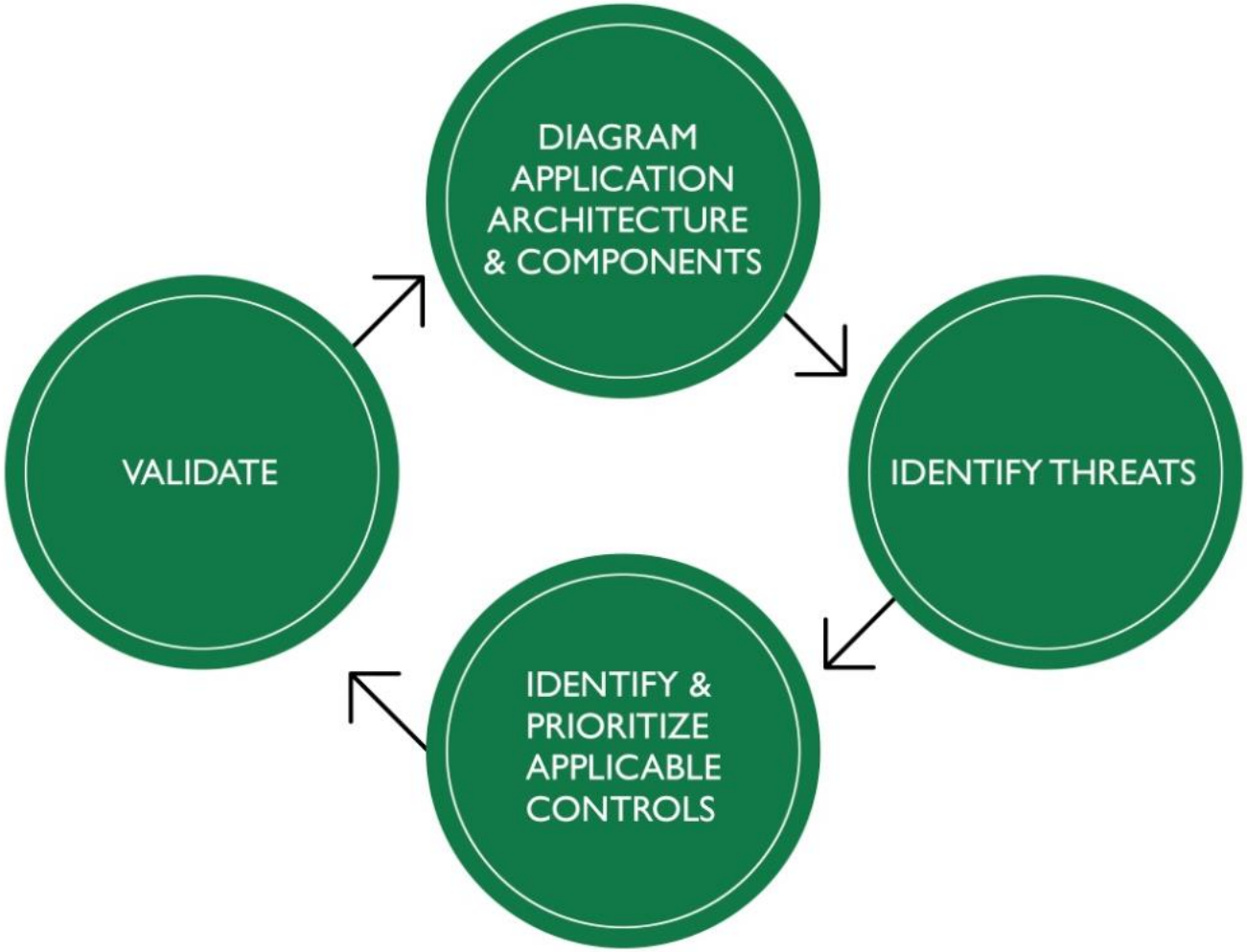
No developer goes to work with the intention of writing bad code.

- Developers are often not trained or experienced in secure coding techniques, and have **never needed to worry** about this before

- Developers face **pressures** of demands for quality and functionality, and are often short on timeline, resources, information, budget, quality assurance tools investment.

- Plus heavy demands on outsourcing parties ….

Tools: Threat Modeling



Tools: STRIDE & DREAD

STRIDE and DREAD

STRIDE is a threat modeling methodology that is performed in the design phase of a software development project, where applicable threats are identified and grouped into six categories

- **S**poofing
- **T**ampering
- **R**epudiation
- **I**nformation disclosure
- **D**enial of service
- **E**levation of privilege

DREAD is a risk calculation and rating methodology that is often used in conjunction with STRIDE. To overcome inconsistencies and qualitative risk ratings, the DREAD methodology aims to arrive at ratings for the identified (and categorized) threats by using the following five dimensions

- **D**iscoverability
- **R**eproducibility
- **E**xploitability
- **A**ffected users
- **D**amage potential

Tools: Threat Modeling

In step 1, participants should diagram:

- Physical topology (intranet, DMZ, Internet)
- Logical topology (layers)
- Services, ports, and protocols
- Identities and authentication (forms, Web app)
- Technologies (Web, database, middleware)
- Dependencies (credit card processor, shipping company, etc.)
- Actors (human, non-human, etc.)
- Authorization mechanisms (RBAC, resource-based)
- Auditing controls

In step 2, participants should consider and identify applicable threats:

- Identity trust boundaries
- Identify data flows
- Identify entry and exit points
- Introduce mis-actors
- Use a categorized threat list (e.g., STRIDE) or an attack tree methodology
- Consider OWASP top 10 and CWE Top 25 most dangerous programming errors

At the end of step 2, participants should have compiled a list of threats.

In step 3, participants must identify controls to mitigate the identified threats (output of step 2).

In step 4, participants validate the threat model to ensure it meets the requirements.

C. Use Verified Secure Software

Service Oriented Architecture

- Black box software components
- Heterogeneous Systems
- Application Programming Interface (API)
  - Representational State Transfer (REST)
  - Simple Object Access Protocol (SOAP)
- Ease of Integration
  - JSON
  - XML
- Agile Programming

Cloud Application Architecture (For Reference)

- Software-Defined Everything
- Service Architecture
- Micro-Services
- API Gateway
  - Governance
  - Throttling
  - Security (Authentication & Authorization)
  - Service Control Gateway
- Docker / Container

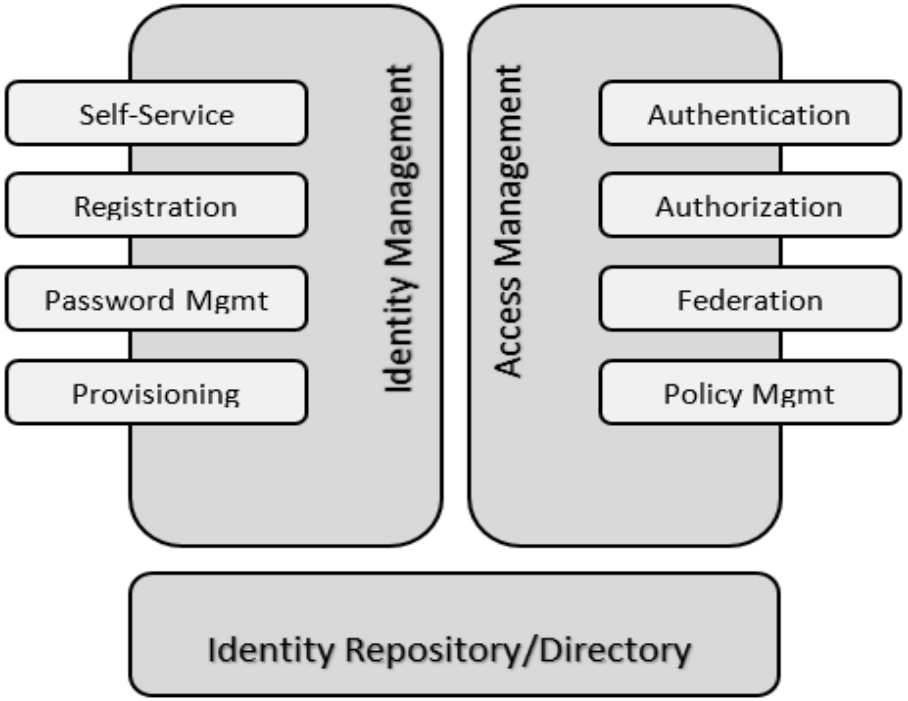
Supply-Chain API Management

- Complex Development Process composed by APIs
- API Economy (cloud-to-cloud)
- Secure end-to-end transmission
- Open API initiative
- API Standard: Swagger / RAML
- Open Source / Open Data
- ISO 27034-1 standard

G. Design Appropriate Identity and Access Management (IAM) Solutions

What is IAM?

- Identity Management
  - Self-Service
  - Registration
  - Password Management
  - Provisioning
- Access Management
  - Authentication
  - Authorization
  - Federation
  - Policy Management
- Identity Repository / Directory



What is IAM?

- Entitlement Management
- Enforcement of logical access controls
- Identity individuals
- Access Management
  - Who, what, how
- LDAP-enabled / Active Directory Repository

Federated Identities

- Policies, processes, mechanisms to manage identity and trusted access to systems across organizations
- Standards
  - SAML
  - WS-Federation
  - OpenID (authentication)
  - OAuth (authorization)

Federated Identities

- Identity Provider (IdP)
  - Hold all of the identities and token generation
- Relying Party (RP)
  - Consume tokens
- Single Sign-On (SSO)

Multi-Factor Authentication (MFA)

- Knowledge (Something you know)
- Ownership (Something you have)
- Characteristics (Something you are)

Step-up Authentication

- One-time password
- Challenge Questions
- Out-of-band authentication (call / SMS)
- Dynamic knowledge-based authentication (unique questions)

F. Comprehend the Specifics of Cloud Application Architecture

Security Devices

- Web Application Firewall (WAF)
- Database Activity Monitoring (DAM)
- XML
- Firewalls
- API Gateway

ICE-BREAKING ACTIVITY

1. ___ A layer-7 monitoring device that understands SQL commands
2. ___ A device that filters API traffic. It can be installed as a proxy or as a specific part of your applications stack before data is processed
3. ___ Transform how services and sensitive data are exposed as APIs to developers, mobile, and cloud
4. ___ A layer-7 firewall, one that can understand HTTP traffic
5. ___ Can be distributed or configured across the SaaS, PaaS, and IaaS landscapes

a. WAF
b. DAM
c. XML
d. Firewalls
e. API gateway

Crytography

Data-in-transit Encryption:

- Transport Layer Security (TLS)
- Secure Sockets Layer (SSL)
- VPN / IP Sec Gateway

Data-at-rest Encryption:

- Instance
- Volume
- File/Directory

Tokenization & Data Masking

Tokenization

- Stored in secure location
- Temporary Use with expiry
- PCI DSS Compliant
- OAuth, API Token, SLS Token, etc.

Data Masking

- Social Security Numbers
- Retain its original format
- Cannot intercept the data

Sandboxing & Application Virtualization

Sandboxing

- Separation, e.g. personal and corporation information in different components
- Run untested or untrusted code in a tightly controlled environment

Application Virtualization

- Wine, App-V, XenApp, etc.

B. Understand Cloud Software Assurance and Validation

Software Assurance & Validation

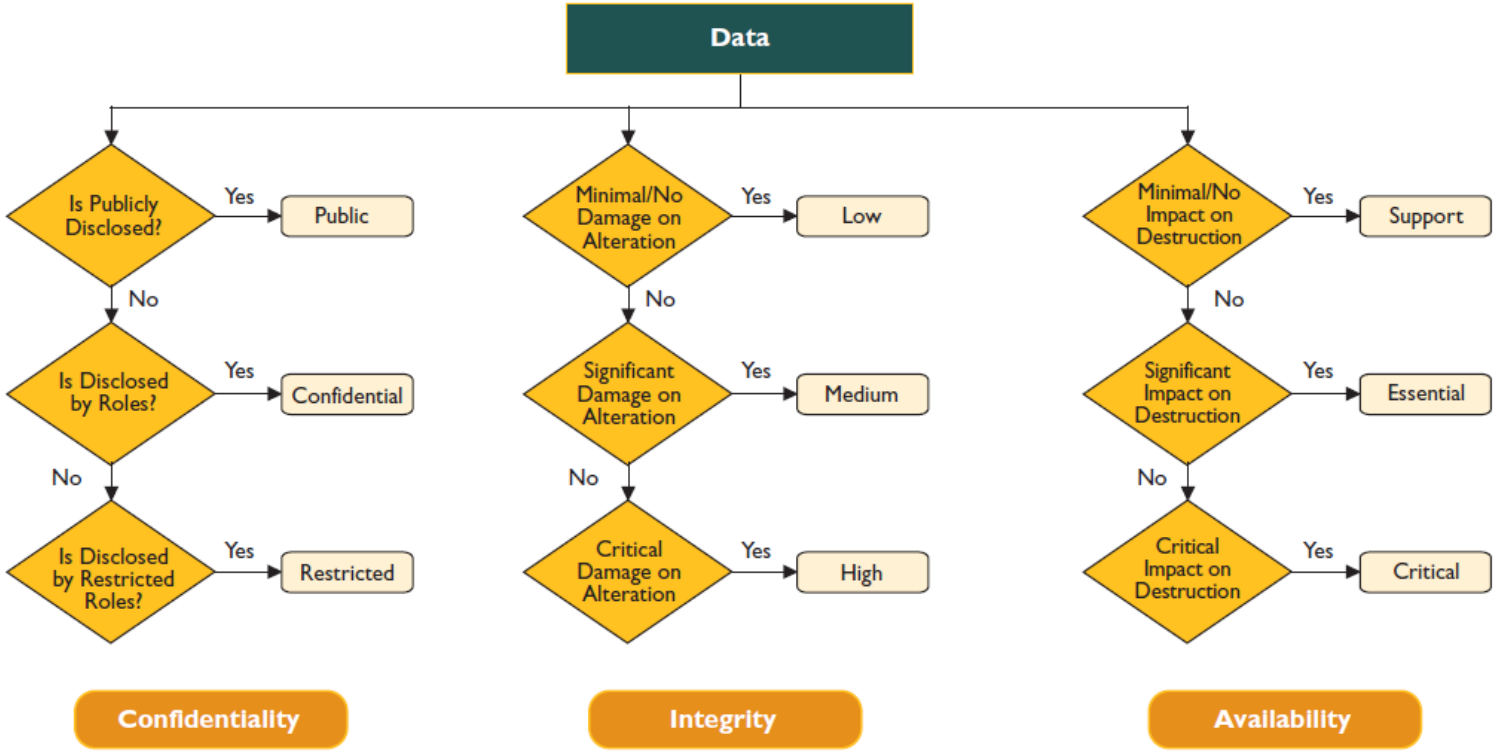
Software Assurance

- Ensure software functions as expected
- Mitigating the risk of vulnerabilities, malicious code, or defects
- Address assurance through entire SDLC

Verification & Validation

- Follow best practice guidelines
- Verification and validation of coding
- Segregation of duties
- Independent Measurable Review
- V&V each stage of SDLC

Cloud-based Functional Data



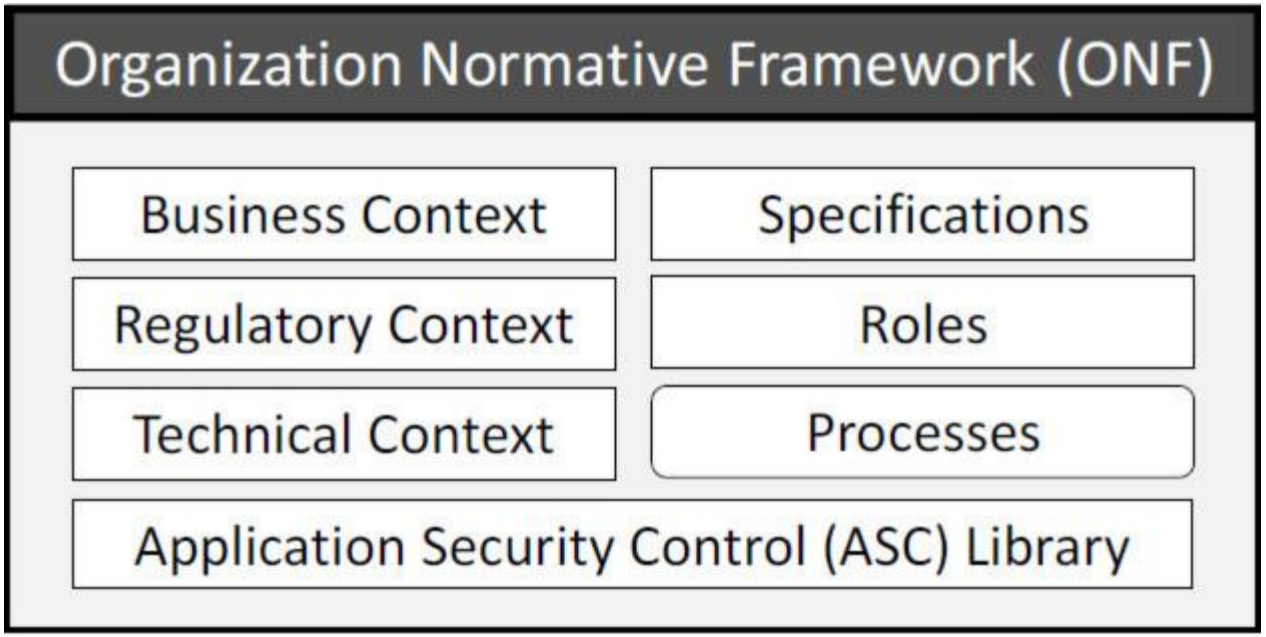
Cloud-based Function Data

- Data Collected, processed, and transferred by different functions
- Legal implications on how data is used, presented and stored
- Breakdown the legal implications of the data
- Regulatory Requirements, e.g. EU privacy law

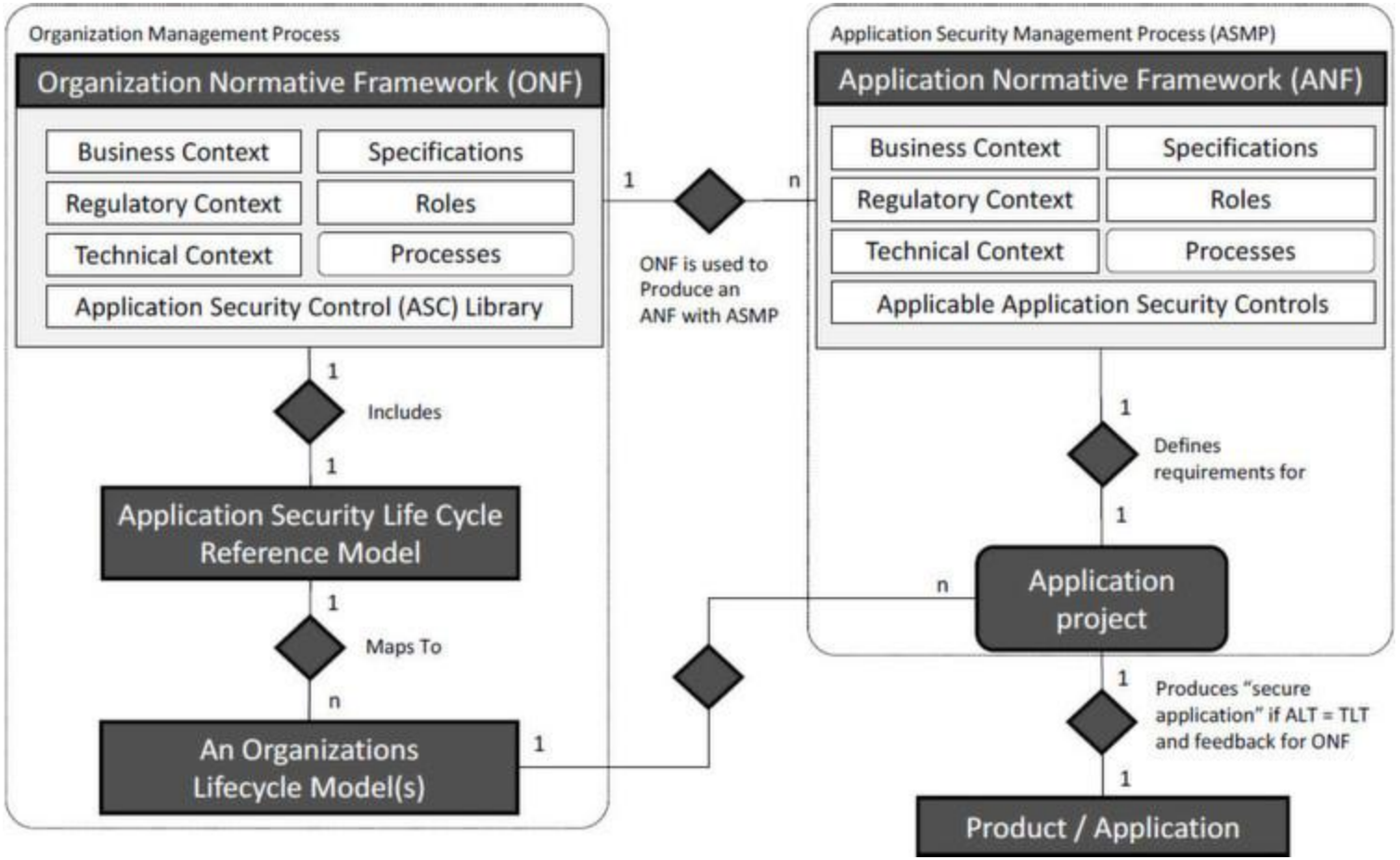
Cloud Secure Development Lifecycle

- ISO/IEC 27034-1
  - Defines concepts, frameworks, and processes to help organizations integrate security within their software development life cycle.
- Requirements
- Design
- Implementation
- Verification
- Release

Organizational Normative Framework (ONF)



- Framework for application security best practices components



- Application Normative Framework (ANF)
- Application Security Management Process (ASMP)
- ONF 1-to-many to ANF
- Specific application scope

Application Security Testing

- Static & Dynamic Application Security Testing (SAST)
- Runtime Application Self-Protection (RASP)
- Vulnerability Assessments (VA) and Pen-Test
- OWASP Active Security Testing

Static Application Security Testing (SAST)

A set of technologies designed to analyze application source code, byte code and binaries for coding and design conditions that are indicative of security vulnerabilities.

SAST solutions analyze an application from the "inside out" in a nonrunning state.

Dynamic Application Security Testing (DAST)

DAST are set of technologies that are designed to detect conditions indicative of a security vulnerability in an application in its running state.

Most DAST solutions test only the exposed HTTP and HTML interfaces of Web-enabled applications; however, some solutions are designed specifically for non-Web protocol and data malformation (for example, remote procedure call, Session Initiation Protocol [SIP] and so on).

Vulnerability Assessments (VA)

Vulnerability assessment tools discover which vulnerabilities are present, but they do not differentiate between flaws that can be exploited to cause damage and those that cannot.

Vulnerability scanners alert companies to the preexisting flaws in their code and where they are located.

Penetration-Testing (Pen-Test)

Pen-Test attempt to exploit the vulnerabilities in a system to determine whether unauthorized access or other malicious activity is possible and identify which flaws pose a threat to the application.

Penetration tests find exploitable flaws and measure the severity of each. A penetration test is meant to show how damaging a flaw could be in a real attack rather than find every flaw in a system..

OWASP Testing Guide

https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents

- Information Gathering
- Configuration and Deployment Management Testing
- Identity Management Testing
- Authentication Testing
- Authorization Testing
- Session Management Testing
- Input Validation Testing
- Testing for Error Handling
- Testing for Weak Crytography
- Business Logic Testing
- Client Side Testing