faresweb.net

WEB & MOBILE SECURITY BEST PRACTICES

Contents

EXECUTIVE SUMMARY

As new technologies emerge, both legitimate and malicious, and as defenses mature, cybercriminals are enhancing their modus operandi. The past year saw the proliferation of organized cybercrime, funded by real-life organized crime and, potentially, even hostile foreign states.

Security and risk professionals need to understand what Internet threats they are facing and to what extent their organizations are at risk. This analysis is not only important to understanding the efficacy of current security mechanisms but is also essential if we as an industry are to prepare for what's to come.

Attack statistics suggest that malware attackers infiltrated benign Web sites by exploiting Web-server vulnerabilities or leveraging third-party content. High-profile Web sites are popular infiltration targets because they have a large amount of traffic.

It's worth noting that a rising number of attacks leveraged Web advertising as a delivery platform. The Web advertising ecosystem creates a complex relationship among ad-hosting, ad-placement, and ad-writing entities: Web sites that host third-party-placed advertisements via Google or Yahoo!, for example, often don't know where the ads come from or whether they contain malicious content. This is also true for Smartphone's application distribution process via Apple App Store or Android Market.

Sun Tzu said in his famous work, *The Art of War*, that a fundamental principle for successful battle strategy is "know thy enemy, know thy self." Understanding the nature of cyber threats is the first step in crafting a strategy against the criminals.

PREFACE

WHAT IS IN THIS BOOK

The risk analysis presented in this book should be viewed as a tool, a framework, for each web project to identify its own security measures and produce its own risk analysis depending on its specificities. It should concerns:

- The business functionalities to be supported by the application;
- The IT infrastructure and deployment architecture;
- The specific regulatory requirements.

The book shows developers how applications can be developed by keeping security considerations in focus and by taking advantage of web architectures to application programming whenever possible to develop highly maintainable, extensible applications for Web, intranet and mobile use.

HOW THIS BOOK IS ORGANIZED

The document is organized into six parts.

PART 1: PROJECTS SECURITY MANAGEMENT

Part 1 provides an overview of task projects to fulfill mandatory security requirements requested by web projects.

PART 2: WEB SECURITY

Part 2 provides an overview of Web security principles based on threat analysis. It helps in taking decision over security measures that must be implemented depending on application risk profile.

PART 3: MOBILE SECURITY

Part 3 describes functional and logical views of the native mobile security architecture including the description of additional components and security services within the platform. It details the security measures and provides a description of each security measures that must be implemented within mobile applications.

PART 4: SECURING THE INFRASTRUCTURE

Part 4 describes infrastructure components to help securing web environments at infrastructure level.

PART 5: SECURING THE APPLICATIONS

Part 5 is all about designing applications while understanding and avoiding security risks. In this part, you learn about practical design and implementation considerations, best practices, and security risks and the techniques you can take to avoid them.

PART 6: APPENDIXES

The four appendixes of last part provide information about different attacks scenario and detailed risk analysis based on different user authentication method use cases.

TELL US WHAT YOU THINK

I am always very interested in learning what my readers are thinking about and how this book could be made more useful. If you are interested in contacting me directly, please send e-mail to fwbooks@faresweb.net. I will do my best to respond promptly. The most updated discussions related to this book can be found at http://www.faresweb.net/forum/mybooks.

PROJECTS SECURITY MANAGEMENT

INTRODUCTION

Organizations that develop applications in-house have a decision to make: you can wait until someone exploits vulnerability in your system and fix it, or you can proactively build security early on in your development process — mitigating vulnerabilities before attackers find them. A proactive application security program should extend to every relevant phase of the application life cycle, from conception to operation: program success hinges on commitment and support from executive management. Security personnel need to work with application owners and business stakeholders to prioritize resources and to ensure proper measures are implemented throughout the life cycle.

SECURITY MANAGEMENT LIFECYCLE

A typical application life cycle contains four phases:

1) Design;
2) Implementation;
3) Quality Assurance (QA) testing; and
4) Operation.

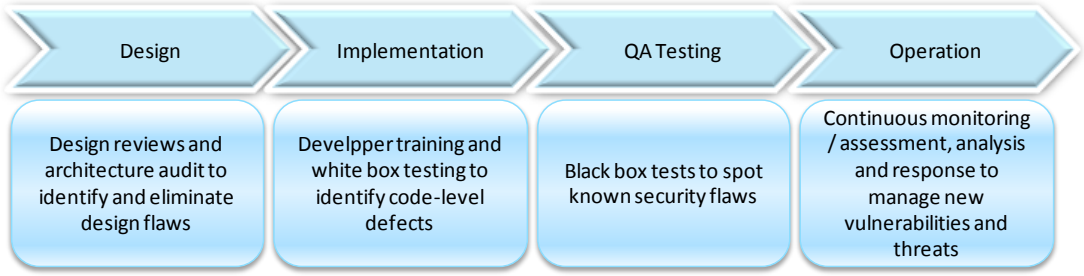


Figure 1 - Application lifecycle

Traditionally, the application lifecycle has been primarily about design and implementation. But the recent proliferation of attacks against Web applications brought a new focus to the post-development phases. To effectively manage the security of in-house developed applications, you must understand the lifecycle of these applications and the relevant security issues in each stage.

Vulnerabilities can arise in each stage of the life cycle. Evidence suggests that the payoff for eliminating flaws early on in the life cycle is high. Security flaws are but a special class of software defects. The earlier you identify vulnerabilities (or their sources), the more time you have for corrective actions. The longer a flaw exists within a piece of code, the more costly it is to repair. During the undetected period, other code might be developed based on the flawed assumption or around the buggy code — to undo entire layers of code is a costly proposition.

DESIGN PHASE

In the design phase, the application exists conceptually through ideas or architectural diagrams. Design flaws that arise here can result in critical vulnerabilities in later phases. To eliminate flaws and errors in the design stage, conducting design reviews and architecture audits is essential.

REVIEWS AND AUDITS

Every web application must draft a specific security form requirement in order to establish the expression of the security need by the project ownership.

It is important that security team conduct or participate in the review and audit process. In these reviews:

- **Clearly document design assumptions and enumerate intended usage scenarios**. This should include expected input values, intended communication interfaces, and assumed operational conditions;
- **Build an architectural representation of the application system.** This should include components (e.g., users, applications, data stores, etc.) trust boundaries, and more importantly, modes of interactions and dependencies between components;
- **Build and analyze a threat model.** This should include all applicable threats, both internal and external, to the application system. The model should detail how a particular threat can arrive at the system, the interaction thereafter, and possible consequences. Pay special attention to security-related features such as cryptographic modules, sensitive data communications, and interactions across trust boundaries;
- **Identify potential vulnerabilities that violate risk policies.** Vulnerabilities that create acceptable levels of risk may not require action;
- **Modify the design to mitigate the vulnerabilities.** Mitigation can occur in the form of implementing additional controls such as strong authentication, or reducing attack surface such as eliminating problematic interface calls.

IMPLEMENTATION PHASE

Code-level defects are introduced during the implementation phase. A number of approaches are available to minimize vulnerabilities during implementation, including developer education and code-level security tools.

Static analysis tool applies a set of rules to the code and attempts to deduce flaws without actually executing the program. Static analysis can identify and eradicate flaws significantly before deployment, which usually results in less costly remediation. However, it is fundamentally limited to the capability of compile-time analysis — certain run-time behavior, either data dependent or environment dependent, will not be visible to a static analyzer.

THE DEVELOPMENT PROCESS

Integrating security as of the development phase serves to ensure good adequacy between the security needs and requirements.

All developers taking part in the preparation of a Web application must be sensitized and trained with regard to development best practices and security rules. It is strongly recommended that the development process should include a code analysis, whether automated or requiring the participation of security experts.

Implementing security mechanisms

The implementation of security mechanisms must be as industrialized as possible and, ideally, it should be part of a service offer that is common to the group. If certain implementations require a case-by-case development, they will have to be qualified by the development security teams.

It is also important to consider that, in so far as possible; the management of jurisdictions must be carried out on all application layers, including the database. Authentication must take place on top level tiers. This measure makes it possible to limit unauthenticated distributed denial of service (DoS) risks being propagated to internal business or data layers.

Running remote code

Running code in the user's browser is a common feature in order to provide more flexible and dynamic interaction with the user. However, since this execution is delocalized to an uncontrolled environment, the processes and the results cannot be considered as trusted. This code must therefore necessarily be validated by the development security teams.

It must under no circumstances take part in business line processes or be used in order to model security mechanisms. This remote code must therefore be limited to web pages expected to extend user experience.

API and external frameworks usage

Any usage of API or external frameworks must be approved in terms of security aspects. Any usage must be referenced such as to ensure that, in the event that software faults are detected, they will be updated as quickly as possible.

Data flows management

All communication rules must be referenced within a document. The related document must be updated after any modification and validated by the production security architects.

Rules for communication with the web

Administration or supervision services coming from the Internet network are formally forbidden. Only those services needed for an application's proper operation from a user standpoint must be accessible.

Moreover, any information received from the public network must be carefully analyzed before being routed to the application server such as to comply with the application separation rules. The objective of this provision is to require, on each application layer, an analysis and transformation of the data received prior to any processing.

Rules for communication between application layers

It is strongly recommended that protocol separation should be applied. This involves, between each application layer, establishing communications that are based on different protocols. This principle is intended to limit the successive usage of protocol vulnerability, and therefore the risk of hacking within the information system.

Any communication from a Web server to another server must be filtered by security hardware and opened only after a security analysis. Each communication must be carried out in peer-to-peer mode.

RULES FOR COMMUNICATING WITH THE INFRASTRUCTURE SERVICES

Communications from Internet applications to the infrastructure services must be studied on a case-by-case basis. It may also be required that certain critical infrastructure services should be dedicated to Internet applications (for example, authentication services).

QA TESTING PHASE

The QA phase is where testers verify that the code does not contain any known security flaws. These tests are analogous to the functional tests performed here, where testers investigate code in an emulated operational environment. The security tests in this phase are called “black box” tests.

True to its name, a black box test usually does not have knowledge of the internal state of the code. Instead, a black box test uses input and output from the system to deduce the existence of vulnerabilities and flaws.

THE SECURITY ACCEPTANCE PROCESS

Like the functional tests during the acceptance phase, the application's security level must be verified before any commissioning. This acceptance must at least include the countermeasures linked to the major vulnerabilities identified. This security acceptance set-up implies the creation of a report that must be approved by the security manager.

Moreover, the acceptance environment must be the same as the production environment. This implies the set-up of the security mechanisms, configuration and hardware before the start of this phase. The objective of this measure is to identify any impacts of the security mechanisms on the application's proper operation.

CODE AUDITS

Particular attention must be paid to security code audits.

Before any production integration, any outsourced or sensitive development must undergo an audit of its security code. This must be carried out on the basis of non-compiled source code, in order to verify that:

- The security mechanisms are present and efficient;
- The management rules for these mechanisms are suited to the application's criticality, and up-to-date;
- The source code contains no vulnerability that could jeopardize the application or the hosting environment.

OPERATION PHASE

The deployment of a software system and shift to the operational phase is the testing point for all of the upstream security work. During this phase, custom application is folded into the greater application vulnerability management process for deployed applications, regardless of provenance. This process encompasses several steps. First, vulnerability monitoring (periodic black box scanning and penetration testing) and threat monitoring ensure the ongoing health of the application. Next, the analysis phase processes this information, assessing the business risk and determining the nature of response. Finally, the response tools manage the change and configuration of systems and software.

Traditional firewalls and intrusion detection systems (IDS) technologies target network-level information and therefore are blind to application-level attacks. In contrast, Web application firewalls understand application semantics — input format, buffer lengths, field values — and can provide in-depth filtering of potentially malicious application input.

Deployment Management

Security lifecycle management (i.e. software updates, components upgrades, etc.) should not impact applications or platforms.

The deployment approach (including initial installation & configuration and subsequent version upgrades) should support operation standards. In particular, it should be possible to perform deployment via existing Operating System and Application installation scripts.

It should be possible to flexibly re-configure the security components on the basis on self-contained configuration files that can be loaded into the system to restore the baseline setup in the production environment.

Hosting

Since a Web application is also one of the public world's doorways to the group's internal environment, it is crucial for the infrastructure that hosts the Web applications to provide a minimum security service, namely:

- Equipment to detect hacking and a permanent watch cell;
- Hardware for filtering Web application content (Web Application Firewall);
- A security patch management policy;
- A policy for the strengthening of network, system and application configurations.

Provisions must be made for the organizational elements needed for the implementation of these principles.

The choice of the hosting environment for the application and its component modules must be made by the production security architects attached to the relevant production.

Outsourced hosting

The outsourcing of a Web application's hosting must comply with the security.

The application of the security requirements described in this document can take the shape of contractual clauses that can be integrated into the contract with the service provider.

Hosting of an intranet application with sensitive processes

All sensitive application code must be protected before any commissioning. This measure means that it is forbidden to introduce code that is interpreted in production, which must systematically be compiled before any deployment.

Pooling between several applications of different types

Applications of different types can be hosted within a single hosting infrastructure if and only if they meet all of the most sensitive application's security rules.

Availability and Disaster Recovery Requirements

Below redundancy requirements should be support to enhance the resilience of the security components:

- High-Availability with redundant nodes (both in Active/Active or Hot-Standby modes) and dual datacenters. In a typical deployment architecture we would have:
  - **Local Redundancy**: 2 nodes (in Active/Active mode) in each datacenter;
  - **Site Failover**: In case of unavailability of one datacenter, it should be possible to use the other redundant security nodes in the disaster site.
- N-1 security nodes should be able to sustain the total load assumed by the overall n nodes;
- To be able to apply upgrades and housekeeping tasks without service interruption, n-1 nodes should be able to sustain the total load assumed by the overall n nodes;
- Recovery mode should support all the requirements in case of failure;
- Transparent Session Recovery: sessions already established in one failing security node should be recoverable by the other node, with the least possible impact on the user, preferably not requiring the user to re-authenticate. *Session stickiness* is desirable, whereby an established session for a given user is entirely handled by a single node.

Patch management

The patch management must align with the production environment's operating processes. The timeframe for applying security patches must be inversely proportional to the severity of the vulnerabilities. Just like the SLA, this timeframe must be the subject of a service contract regarding the commitment to implement security updates.

Moreover, organizational and technical systems must be implemented in order to be able to update the elements that comprise the hosting infrastructure and the Internet applications, without impacting production. These measures must make the application of security updates independent of the availability constraints of the hosted Internet applications.

Administration

Web applications must only be administered by authorized persons mandated by the production manager, and this must be carried out by means of a strong authentication mechanism and encryption on the transport.

All administration actions must be the subject of detailed tracing, accommodated within a dedicated environment. This environment cannot be administered by the same team.

Supervision

Though the security mechanisms integrated into the development and operational phases may well be working, it is still probable that an attacker will be able to work around or overcome these various security mechanisms. It is therefore essential to implement, on all infrastructures that host Internet sites:

- Intrusion detection systems (IDS);

- Event aggregation and correlation systems.

These mechanisms must be maintained by a dedicated security team, and continuously monitored. Security incident management system must be updated for each detected intrusion.

Moreover, and in order to detect any application malfunction (application bug, denial of service, etc.), monitoring of each application must be implemented.

INTRUSION AND VULNERABILITY TESTS

For every Internet site, regular intrusion and vulnerability tests must be scheduled involving all of the Group's Web applications, irrespective of the selected hosting mode (internal or external).

These tests must under no circumstances result in temporary or permanent deterioration of the audited environments.

VULNERABILITY TESTS

Vulnerability tests must be mandatory for all Group Internet sites, irrespective of their criticality.

They must also apply to the elements that support the applications (network, system, middleware, etc.) and be carried out *in situ*, independently of the filtering mechanisms.

INTRUSION TESTS

As described in the policy, the performance of every intrusion test must be supervised by an *ad-hoc* follow-up committee.

This committee must include:

- A representative of the sponsor;
- A security representative (from the division, territory, Project Manager...), depending on the audit perimeter;
- A representative of the Group ISS.

These intrusion tests can be carried out in 3 steps according to the asset's criticality:

- The first, known as the "black box test", involves simulating an external attack without any authorization or specific internal knowledge of the application;
- The second, known as the "grey box test", involves simulating an attack by a malicious user authorized to use the application (the various clearance levels may also be considered);
- The last one, known as the "white box test", involves simulating an attack by a person who has detailed knowledge of the application (access to source code, detailed security mechanisms, etc.).

SECURITY INCIDENT MANAGEMENT

The management of security incidents must be compliant with the policy of the organization.

Moreover, every detected security incident must be the subject of a specific analysis process by the security teams in order to determine the best response, as quickly as possible.

After each security incident, all Internet applications must be reviewed in order to determine whether or not the incident has affected any other applications, or if the latter could be vulnerable.

At the end of the intrusion or vulnerability tests, if security-related defects have been detected, each of them must then be addressed in a retrofitting action plan.

Should an incident be detected, an assessment will have to be provided by the CSIRT in order to propose suitable measures for the incident's resolution (for example, possibly resulting in the temporary closing of external communications relative to the application during the retrofitting phase).

END-OF-LIFE PHASE

When a Web application arrives at the end of its life, all of the adherences to the enterprise Information System must be terminated.

They include:

- The flows and configurations specifically created on the security and filtering hardware;
- The application's authorizations on the infrastructure services or any other group asset.

Before any reuse, all of the configurations applied to each of the servers and systems that supported the application must be reinitialized to the standard configurations defined by the production manager.

GUIDELINES TO DEVELOPING SECURE WEB APPLICATIONS

While you want users to come to your site and interact with the application, you need to be paranoid and build your site assuming the user will try to manipulate the application. Only information that is derived from the enterprise can be trusted and everything else (e.g.. from the client) is always potentially dangerous.

Therefore, the first guideline for developing a secure web application is: never trust any information that comes from the client, and never assume anything about it. All security decisions must have the underlying assumption that anything that can theoretically be manipulated by the user will be manipulated in reality. Never assume that just because a user supposedly uses a specific tool, that this tool will put any constraints on his actions.

For example, the fact that the browser does not show the hidden fields in the HTML of the page does not mean they cannot be seen by looking at the traffic, and manipulated when sent back to the server.

The second guideline: it is always easier to secure simple logic than complex logic. This actually complements the first guideline as it relates to the usage of logic on the client. The use of client side logic (such as scripts written in JavaScript) might enhance the user experience with the application, but it has serious affects on the site's security.

Using client side logic as a security mechanism by checking information on the client is easily subverted since the mechanism can be bypassed. For example, having JavaScript verification of constraints on form parameters can easily be bypassed by directly creating the request outside the browser. In addition, any changes to the application logic or flow by changing links or parameter

values, creates a complex program with multiple sources of flow instead of a single unified point or a simple flow structure.

The guidelines for securing a site's applications can be organized in a hierarchical structure in order to address security at each level. The first level, the single transaction, is the smallest piece of logic in a web application. The second level is the complete session and is made up of multiple transactions. The uppermost level is the complete application including a large number of different sessions.

SECURING THE CORE

ENCODING THE TRANSACTION

The first step to providing a secure transaction is bringing it to a standard format that can be understood with no ambiguities or multiple representations. Encoding transformation brings the transaction into a canonical format, and is crucial before any additional security measures are performed on the request. The same transaction passed using different encodings may have different lengths and reflect different patterns, making security measures ineffective. The application must establish a single encoding scheme for each request (preferably common to all requests within an application) during the design stage, which is non-ambiguous and has a single representation. The decoding function should be the first to be invoked during transaction processing.

It is crucial to protect the transaction parameters enforcing their validity and fit with the application logic. An attacker of the web application may change the value of any parameters sent to the server. As a result, nothing can be assumed about this input. All input must be checked for maximum number of characters on the server side before any use of the parameters. Setting the maximal side on the HTML page or verifying the input on the client via a scripting language may be useful from a functionality standpoint but cannot serve as a security measure. The client can easily remove the client-side tests by changing the page on his browser or by creating the request outside of the browser.

Moreover, even in parameters where the input is not free text, such as a pull-down menu or hidden field, no assumptions should be made on the parameter length since they can assume any value. The validation check for all of the parameters should always be done following the canonization of the encoding of the parameters to avoid changes to the parameter value that may result due to changes in the encoding. Most fields should contain only characters that are used in the specified language and not any meta-characters such as <>"& etc. that can be used to encode special attacks. Filtering such characters out of the parameters to avoid the attacks should enforce this point.

It is important to realize during the design stage which parameters might receive special characters and make them the exception. It is important that for these parameters, only the specific characters are allowed and potentially dangerous sequences are eliminated. It is preferable to start from the "positive" characters such as identifying the legal characters (i.e. A-Z and 0-9) and add as needed, instead of removing the illegal characters. This will ensure getting a minimal subset of needed characters and ensure unwanted characters are being filtered.

Parameters should typically have as many constraints as possible attached to them. Always attempt to avoid free format input wherever possible. Instead, define the most limiting structure. The definition of the correct input should be defined in the design stage and should be enforced by adding the specific attributes during the coding stage. Such constraints include the maximal size and the valid characters for the field using the CHARSET HTML attribute. In addition, whenever possible try to

choose specific values from a list rather than a free choice of input. For example, instead of typing the two characters of the state within a specified area provide a list of possible location.

OBSCURITY

Obscurity does not provide security by itself. However, it is typically better to keep things hidden instead of exposing them to easy access and manipulation. Obscurity may be easily improved by using HTTP POST method instead of the GET method. This causes the parameters to be passed in the body of the request instead of being visibly seen and potentially changed within the URL itself. This eliminates many of the naïve URL changes by users trying to poke around the application. By itself this will not stop the advanced hackers that typically have tools for seeing and manipulating the POST parameters.

Improving obscurity may also be performed using a closely related technique: removal of all parameters from links. Links, which contain parameters, should either be encrypted or signed for improved security or at least passed as hidden parameters for improved obscurity (see hidden parameter guidelines separately).

Typically, the best way of maintaining the transaction information is by keeping it on the backend and away from the client. However, in some cases the simplest and sometimes only possibility is by passing the information through parameters that are passed through the client usually as hidden parameters. As before, the parameter names and values should be encrypted to avoid revealing their value or at least signed so they cannot be manipulated.

Another important factor for concealing information is removal of meta-information. Meta-information that exists within the web pages, such as comments sent to the client, can provide valuable clues to the hacker and aid in exposing more vulnerability. Therefore, all comments should be stripped from web pages in the production environment. Any client side code or parts of the HTML, which are commented out, should be removed as well.

DYNAMIC PAGE CREATION

As part of the process of tailoring a rich user experience, delivering adapted services for his specific profile and session, aspects of the user input must be taken into account to create the dynamic pages sent back to the user.

However, this should be done with care: Never use values received from the client to directly create dynamic pages. Implanting users' input directly within pages can lead to severe consequences. For example, a parameter received from the user that contains his/her name might be used to create a page containing a "welcome user" phrase. However, the client might submit a JavaScript code within this parameter that when implanted in the page will cause cross-site scripting and lead to a virtual hijacking of the client's communication. This implanting of the script leads to it running in the user's browser and copying or altering data that is passed between the client and the site. The input should always be verified to be "script free" by removing any dangerous characters. Care should be taken to verify that any encoding is transformed into canonical structure before removal of the characters.

USAGE OF HTTP HEADERS

HTTP headers raise security issues. Similar to all other information from the client, these fields are easily manipulated. Therefore, while they might be used to provide certain functionality, they should never be used to provide security. A good example is the REFERER header, which reflects the page leading to the transaction. It can be manipulated to contain any desired URL. Whenever a header needs to be used it must be signed and preferably encrypted, such as in the case with cookies.

Although such headers should never be used for security, they can be used to make hacking attempts more complicated. For example, by using the REFERER to disqualify some requests from outside the site a hacker will need to work harder and make manual changes to overcome the obstacle.

STANDARDS

All web protocols and standards are well defined within specifications; however most web servers and web applications accept a large number of deviations from the standard. Using standard HTML and HTTP will help avoid the possibility of misbehavior by any of the components interpreting the information. Whenever a non-standard protocol is used, it may lead to ambiguity and ill-defined responses. Since it is important at each stage to understand the expected transaction, it is best to avoid potential ambiguities.

THE SECURITY SESSION MANAGEMENT

Multiple requests are organized together in sessions, which are tied to a logical entity representing a single user who is using the application. The preliminary condition to securing the session is to secure all its components i.e. the single transactions. Without this preliminary condition, no security efforts for securing the session will succeed.

Web based protocols and standards such as HTTP and HTML are context-less. Therefore, a context mechanism needs to be created using the application level. A session is typically initiated using an authentication process where the user is identified, usually via a simple user name/password mechanism. Following successful authentication, the user is given a token that identifies him/her to the application and provides the context in which all his interactions with the server are evaluated.

AUTHENTICATION

The session creation stage is especially prone to security attacks. Once a session is created, all further actions are associated with this “legal” session without the need for further authentication. There are several rules for securing this process. The most basic is to always pass the authentication information (such as user name / password) over a secure medium, such as SSL. This verifies that the information will not fall into the wrong hands due to tapping. In addition, a strong password scheme is needed to prevent enumerations over the possible passwords due to a short password or obvious ones, which could be found in a dictionary.

In addition, no default users and passwords should exist in the application such as demo and administrative users. Such default users tend to remain accessible after deployment making them easy prey for attackers.

The authentication process should assume that it might be attacked by enumeration, and therefore use defensive measures to address this case. The authentication mechanism should contain a delay to slow down enumeration. In addition, a threshold should be given for multiple failed authentications for a specific account, both as a measure of consecutive attempts and regarding the number of failed attempts over a longer period of time. Finally, a secondary authentication mechanism, separate from the first, is strongly recommended to perform crucial transactions. For example, having a first set of credentials for entering the online bank, and a second internal set of credentials for transferring money out of the account.

SESSION MAINTENANCE

Following the authentication process, a session is created and must be associated with the user authenticating into the system. It is crucial to create a secure session identifier since comparison to this identifier actually bypasses the whole authentication mechanism. The session identifier must be

cryptographically strong, signed and time stamped. It is better to use well-known algorithms and avoid "inventing" new algorithms since they are prone to design mistakes. The identifier itself has historically been passed as a cookie sent to the client and submitted back to the server with every request. However, due to privacy concerns some applications have moved to URL mangling which adds the identifier as either a parameter to the URL or even as part of the path part of the URL. Note, as the identifier in the URL, it will become part of the REFERER header when the user accesses a different site and will therefore be exposed to other sites. All private areas, following the login procedure of the site, should be associated with the session identifier. Switching to a weaker session ID or counting on IP/SSL information to maintain the user's identity causes vulnerability for the whole session.

In addition, utilizing a session identifier for all the public areas is a good obstacle for automatic and manual attack methods used by hackers. It will help slow them down or cause them to leave the site. Using this method, a session ID is attached to the user whenever he accesses the application even prior to the authentication. Following the authentication, a second identifier may be used or the public identifier may be associated with the login credentials via the backend. In any case, all session IDs must be secure (e.g. signed and encrypted via a strong crypto mechanism). The session identifier should never be changed by the client. Any code running on the client cannot guarantee a secure ID change without opening security vulnerabilities.

Session Termination

Session termination is an important aspect of session maintenance. Unattended sessions left open over time provide a huge security hole for potential breaches. Non-terminated sessions enable attacks on the specific session and potential identity theft of the user owning it. All attempts should be made to shorten the vulnerability period of the session.

The session should be terminated under any of the following conditions:

- Inactive session – a session that has not been active over a reasonable time, which varies between sites and applications, typically 15-30 minutes;
- Long session – a session, which despite being active has reached the maximal allowed time and must be re-authenticated or created. The time can vary according to the application type, typically several hours;
- Logoff / Logout – a session should always enable the user to logoff/logout of the session as the most secure option;
- Security error – any security error in the application should immediately result in termination of the session.

Flow Maintenance

By default, web based protocols have no flow, and the transactions have no inherent order and sequencing. However, they often do explicitly demand some flow such as forcing an authentication point before accessing private data. More importantly, many web applications contain a large number of implicit constraints on the flow of the application.

For example, applying for an account might require filling multiple forms sequentially without the ability to jump to the next form before filling in the prior form. Enforcing this flow is crucial to prevent hackers from getting to parts of the application that they should not reach. During the design stage, it

is important to define all of the possible states of the site that describe its flow and assign every single web page to one of the possible states.

The state (or flow) should be maintained by the server and might be sent to the user as a cookie or a field. It is important that the state indicator should be encrypted and signed for the specific session to avoid its hijacking and usage during attacks on the site. All subsequent web pages must have a different state with a direct or indirect flow from the initial state.

Application security

During the application design stage, the areas accessed without session information must be determined. These areas may be accessed by tools that include search engines and other non-session dependent crawlers. It is recommended that these areas be separated into individual directories or servers to avoid mixing the public data with private areas of the application. The simpler the application structure, the greater the chance for achieving good security. Removing cross dependencies within the applications is an important step.

All linkages between the applications can increase the security complexity. Therefore, the number of cross-links between the applications should be limited and clearly mapped with their security considerations understood.

Entry Points

Providing security for the application is impossible unless it is clear which entry points exist to the application. These are the points where users can access the site from the outside. Once defined, the entry points can be secured and the flow from them determined.

The first stage, therefore, is the definition of entry points, which fall into four main categories:

- User access - these are the root points of the different applications. Any increase in the number of entry points causes a similar increase in the needed effort for securing the application during all of the development stages: design, coding and testing;
- Search and Index agent access- such agents do not maintain a session, and therefore must consider any page they access as an entry point. This is a crucial design consideration to avoid publicly exposing indexing of pages. The pages that should be indexed must be defined during the design stage and should be considered as public and session-less. It is important to differentiate these areas of the application preferably by assigning them to different areas of the site hierarchy. Another measure, which might prevent a mistaken indexing of a confidential section, is the usage of the robots.txt file, which can limit access of the search agents or robots.
- Bookmark access- during the design stage, it is important to define the behavior of the application when it receives a bookmark. Bookmarks, which are public and session-less, should be accessible as entry points. Bookmarks into session-based pages are typically those following authentication and should redirect to a legal entry point instead of to the requested object.
- Secure entry points - typically given to partners allowing them to access private areas of the site. These areas should not be treated as simple entry points. They should always be identified early in the design stage and be limited in their number to minimized the threat that they will be used to bypass authentication mechanisms. Such links to non-public areas should always be signed to provide identification of the partner. The signature will typically

be within the path of the URL or as one of the parameters. Such entry points should also be passed over SSL to prevent their interception and re-use by an attacker.

There should be as few entry points as possible to maintain a reasonable review process of pathways to the application on an ongoing basis. All other entry points should be disabled.

ENCRYPTION

Encryption is a key aspect in providing security to the web application. Encryption may be performed on the information stream using SSL or on specific parts of the transaction. The complete transaction should be encrypted using SSL for all private areas of the application. These transactions, which follow the authentication process, are unique to the user and should be kept from interception or change. However, designing the application should not create dependencies on the encryption itself, which could enable the addition of a SSL proxy between the clients and the web site. The links should be relative and not contain the https:// prefix. It is also important never to use IP addresses in the URLs but rather to use host names to allow flexibility and the addition of secure applications (or better yet, to keep the links relative).

Applications, which include proxies and SSL accelerators, might change the IP addresses of the server. It is important to remember that SSL acts as a stream encryption mechanism thereby protecting the transport layer and is not specific to the application. Therefore, in addition to the encryption of the data stream, it is important to encrypt specific fields in the HTML forms and mangle links (URLs) within the pages. This is necessary to obscure the content and structure of the application and avoid its probing. All the fields and links used by the server should be signed, and preferably encrypted, or compared with values stored in the backend to avoid their manipulation.

CACHING

By allowing proxy servers to cache information, a site actually transfers part of its logic to other servers outside the organization (be that proxy servers or user browsers). External caching servers are important for the performance of the information retrieved from the application. They provide an important method of speeding the delivery of multimedia information to the user. However, their use might cause a security hazard. As a guideline, never put any content pages (e.g. non multimedia) on external unprotected cache servers. These pages, which are part of the application, will expose the application to changes in the pages, cause loss of control over the application flow and add complexity in the security design if put on the external server.

Submitting private information with the “no-cache” indication can also protect the privacy of the application’s user. This will prevent records from remaining on the user’s computer, cached for future use, after the web page is served. Removal of the page by setting this indicator not only adds to the obscurity, but also protects the user’s private data from being copied or manipulated.

WEB APPLICATION ENVIRONMENT

Large number of vulnerability threats can be added to the application environment. Only by deploying the application securely within this web environment, is it possible to have a secure web application.

PRODUCTION SITE

The server running the production version of the application must always be separate from the rest of the internal servers (typically in a DMZ, see following section). This server should not run any other software that might disrupt the web application and should never be used to develop the application code. This avoids temporary, old, saved files, etc., which are saved both automatically and manually as part of the development and maintenance cycles. The production server should maintain a sterile

environment cleaned thoroughly before every new release. The application should then be copied into the sterile production environment from an internal computer. This will ensure that only the minimal needed parts of the application actually reside in the production environment.

The production site should never be administrated from outside the organization in order to prevent an abuse of the management application. It is best not to use remote administration even from within the organization. Administration should optimally be executed locally on the production computer.

DMZ

The DMZ is a crucial component of the periphery and network defense for the organization. In addition to providing the network level protection, it is also a vital part in creating a safe environment for the web application. The DMZ serves as a way of separating external facing machines from the internal machines thereby separating the web applications from each other. It is also an important structure for enabling the use of the same applications for dealing with internal vs. external users of the system. Using the DMZ, the application's front end lies within the DMZ while the backend resides in the internal part of the network. By maintaining such a structure it is easier to build differential access for external users accessing through the DMZ and internal users accessing the application through the intranet or other access methods.

A more secure configuration, from the application standpoint, can be achieved by separating the DMZ into two parts. The first part will contain the public areas of the application and the multimedia files on separate servers. The second part will contain the private areas of the application which will also be the only part accessing the back-end system. In such a configuration, a compromise to the public part of the site is still prevented from spreading to the more critical parts of the application.

3RD PARTY TOOLS

The web application typically contains a mix of 3rd party software coming from large vendors to freeware including software for the web servers, application servers, ecommerce packages, etc. Although they are not part of the internal development cycle of the organization they still affect the overall security of the web application.

SECURE CONFIGURATION

The installation of 3rd-party tools tends to lead to less-than-secure configuration by default. All effort should be made to achieve the most secure configuration of any of the products used as part of the web application. In many cases, the vendors will have a guideline for creating a maximum-security installation. Such guidelines should be followed as closely as possible. In addition, it is crucial to verify that all the default accounts are removed from the system and that any default passwords are changed. This prevents attackers from simply going into any of the publicly available lists or users and passwords for any of the infrastructure products. Most 3rd party tools will come with demos and sample applications associated with them. These sample applications often prove to be the Achilles heel of the web application, since attackers of the application may abuse them. All demo and sample applications not needed must be removed from the production server and preferably, never installed in the first place. Moreover, the most secure installation of 3rd party software is installing it on separate servers from the rest of the application (whenever possible), thereby minimizing the potential hazards that it might cause.

PATCH PROCESS

The majority of the vulnerabilities in 3rd-party tools affect a large number of organizations. Some, like Apache Web server, are a part of millions of web applications. Therefore, when vulnerability is found

in one of these products, a patch is released to prevent an attack. Patch latency, or the length of exposure to the vulnerability, is the most critical issue during this process. This is the time between the initial discoveries of the vulnerability, until the actual deployment of the patch in the production environment which prevents the vulnerability from being used. The patch latency period contains many stages and minimizing the length of time affected by this process should be a priority. There are many web sites and mailing lists available that make released vulnerabilities and patches available publicly.

Once a patch has been discovered, the following process should be performed as soon as possible. Install the patch on all the product installations that might be affected. Following the initial installation of the patch, any additional copies of the product in the company (added after the patch is released) must have the patch installed as part of the initial installation process. It is important to remember that even in the best case and under extremely fast cycles of patch installations, the site will still be vulnerable for at least the period between the hacker discovery of the vulnerability and the patch roll-out by the application vendor.

WEB SECURITY

INTRODUCTION

It is business critical for web applications to rely on a robust, highly available, scalable and cost effective security processes and platforms.

Exposing more and more very high value business applications on the Internet made them very attractive for ill-intentioned hackers. While the perimeters of the organizations using these applications are more and more secured, hackers are now attacking the application layer and the clients directly, instead of the infrastructure. It is of survival importance to respond with applications that are as attack proof as possible.

Security architecture should ensure that the internet platform is protected against multiple attack vectors. The impact of these attacks on a business level could lead to reputational damage, financial loss or non-compliance with legal or regulatory requirements.

Main business threats facing are:

- **Fraudulent transactions (integrity issues):**

  An application performing transactions can be targeted for personal benefits of the threat agent. These fraudulent transactions become more and more prevalent, and up to today, there are several attack techniques that can be used to gain unauthorized access to transaction capabilities, or to intercept and modify authorized transactions. For example: Man-in-the-middle, Browser-in-the-middle attacks, Credential stealing, web site spoofing by DNS attacks, phishing, DoS, buffer overflow…;

- **Unauthorized disclosure of business information (confidentiality issues):**

  Platforms can be manipulated; inadequately protected or configured that could lead to the disclosure of information used within the business processes. The disclosed information could be personal customer information, transactional information, or internal organization information abuse.

- **Unavailability of the application (availability issues):**

  Platforms can become unavailable due to intentional or unintentional actions. The following are examples of threats that could happen:

  - Web servers overload or network congestion due to unforeseen volume of transactions (unintentional and intentional);
  - Failure due to unreliable software/hardware or operational mistakes (unintentional);
  - Exploiting conditions in the platform that crash the application (intentional).

- **Integrity issues with information (integrity issues):**

  The content of the platforms can be deleted or modified by malicious people hereby affecting organization's public image, commonly, these attacks are known as defacements.

These modifications of the platform could also be targeted in order to publicize falsified information.

WEB THREATS ANALYSIS

In order to perform a useful threat analysis of a web application platform, some architectural assumptions about such applications must be made. This section describes a generic architecture for typical 3-tier web applications. It serves as the basis for analyzing the threats in the most important infrastructural components in that architecture.

Web applications are commonly targeted for security attacks on the Internet. They are easily accessible through the HTTP-protocol, and often company-critical assets are part of the web application infrastructure. Although web applications infrastructures are fairly complex, basic technology for building web applications is easily accessible. Hence, web applications are often designed and built by developers with little or no distributed system security background.

A variety of technologies are used today for building web applications. Older technologies such as CGI (common gateway interface) provided a simple standardized interface between a web server and an existing application. The application was started on the web server for every dynamic request in order to process the request, introducing a big startup and shutdown overhead. Newer technologies handle dynamic requests. The real processing work can be delegated to a separate application server, leading to better performance and manageability.

Moreover, the application server can offer support for non-functional requirements of the application such as transactional behavior, synchronization, access control and so forth.

The following set of core terms are defined to avoid confusion and to ensure they are used in the correct context:

- **Asset**: A resource of value such as the data in a database or on the file system, or a system resource;
- **Threat**: A potential occurrence of malicious threat that may harm an asset;
- **Vulnerability**: A weakness that makes a threat possible;
- **Attack**:(or exploit): An action taken to harm an asset;
- **Countermeasure**: A safeguard that addresses a threat and mitigates risk.

THREAT CATEGORIES

Different threat categories affecting web applications are listed below. It should be noted that a given vulnerability may be exploited by more than one type of threat. For the sake of simplicity, only the primary vulnerabilities are presented for each threat.

As mentioned in the introduction section, the ultimate risk rating and recommended mitigation actions will be analyzed based on, amongst other things, detailed functional analysis and architecture design.

This part identifies a non exhaustive set of common network, host, and application level threats, and the recommended countermeasures to address each one. The main components of the threat environment affecting a web application are depicted hereafter.

Threats faced by web applications can be categorized based on the goals and purposes of the attacks:

- **Spoofing** is attempting to gain access to a system by using a false identity. This can be accomplished using stolen user credentials or a false IP address. After the attacker successfully gains access as a legitimate user or host, elevation of privileges or abuse using authorization can begin;
- **Tampering** is the unauthorized modification of data, for example as it flows over a network between two computers;
- **Repudiation** is the ability of users (legitimate or otherwise) to deny that they performed specific actions or transactions. Without adequate auditing, repudiation attacks are difficult to prove;
- **Information disclosure** is the unwanted exposure of private data. For example, a user views the contents of a table or file he or she is not authorized to open, or monitors data passed in plaintext over a network. Some examples of information disclosure vulnerabilities include the use of hidden form fields, comments embedded in Web pages that contain database connection strings and connection details, and weak exception handling that can lead to internal system level details being revealed to the client. Any of this information can be very useful to the attacker;
- **Denial of service** is the process of making a system or application unavailable. For example, a denial of service attack might be accomplished by pealing a server with requests to consume all available system resources or by passing it malformed input data that can crash an application process;
- **Elevation of privilege** occurs when a user with limited privileges assumes the identity of a privileged user to gain privileged access to an application. For example, an attacker with limited privileges might elevate his or her privilege level to compromise and take control of a highly privileged and trusted process or account.

NETWORK THREATS

The primary components that make up network infrastructure are routers, firewalls, and switches. They act as the gatekeepers guarding servers and applications from attacks and intrusions. An attacker may exploit poorly configured network devices. Common vulnerabilities include weak default installation settings, wide open access controls, and devices lacking the latest security patches.

INFORMATION GATHERING

Network devices can be discovered and profiled in much the same way as other types of systems. Attackers usually start with port scanning. After they identify open ports, they use banner grabbing and enumeration to detect device types and to determine operating system and application versions. Armed with this information, an attacker can attack known vulnerabilities that may not be updated with security patches.

Countermeasures to prevent information gathering include:

- Configure routers to restrict their responses to footprinting requests;
- Configure operating systems that host network software (for example, software firewalls) to prevent footprinting by disabling unused protocols and unnecessary ports.

SNIFFING

Sniffing is the act of monitoring traffic on the network for data such as plaintext passwords or configuration information. With a simple packet sniffer, an attacker can easily read all plaintext traffic. Also, attackers can crack packets encrypted by lightweight hashing algorithms and can decipher the payload that you considered to be safe. The sniffing of packets requires a packet sniffer in the path of the server/client communication.

Countermeasures to help prevent sniffing include:

- Use strong physical security and proper segmenting of the network. This is the first step in preventing traffic from being collected locally;
- Encrypt communication fully, including authentication credentials. This prevents sniffed packets from being usable to an attacker. SSL and IPSec (Internet Protocol Security) are examples of encryption solutions.

SPOOFING

Spoofing is a means to hide one's true identity on the network. To create a spoofed identity, an attacker uses a fake source address that does not represent the actual address of the packet. Spoofing may be used to hide the original source of an attack or to work around network access control lists (ACLs) that are in place to limit host access based on source address rules.

Although carefully crafted spoofed packets may never be tracked to the original sender, a combination of filtering rules prevents spoofed packets from originating from your network, allowing you to block obviously spoofed packets.

Countermeasures to prevent spoofing include:

- Filter incoming packets that appear to come from an internal IP address at your perimeter;
- Filter outgoing packets that appear to originate from an invalid local IP address.

SESSION HIJACKING

Also known as man in the middle attacks, session hijacking deceives a server or a client into accepting the upstream host as the actual legitimate host. Instead the upstream host is an attacker's host that is manipulating the network so the attacker's host appears to be the desired destination.

Countermeasures to help prevent session hijacking include:

- Use encrypted session negotiation;
- Use encrypted communication channels;
- Stay informed of platform patches to fix TCP/IP vulnerabilities, such as predictable packet sequences.

DENIAL OF SERVICE

Denial of service denies legitimate users access to a server or services. The SYN flood attack is a common example of a network level denial of service attack. It is easy to launch and difficult to track. The aim of the attack is to send more requests to a server than it can handle. The attack exploits a

potential vulnerability in the TCP/IP connection establishment mechanism and floods the server's pending connection queue.

Countermeasures to prevent denial of service include:

- Apply the latest service packs;
- Harden the TCP/IP stack by applying the appropriate registry settings to increase the size of the TCP connection queue, decrease the connection establishment period, and employ dynamic backlog mechanisms to ensure that the connection queue is never exhausted;
- Use a network Intrusion Detection System (IDS) because these can automatically detect and respond to SYN attacks.

HOST THREATS

Host threats are directed at the system software upon which your applications are built.

VIRUSES, TROJAN HORSES, AND WORMS

A virus is a program that is designed to perform malicious acts and cause disruption to your operating system or applications. A Trojan horse resembles a virus except that the malicious code is contained inside what appears to be a harmless data file or executable program. A worm is similar to a Trojan horse except that it self-replicates from one server to another. Worms are difficult to detect because they do not regularly create files that can be seen. They are often noticed only when they begin to consume system resources because the system slows down or the execution of other programs halt.

Although these three threats are actually attacks, together they pose a significant threat to Web applications, the hosts these applications live on, and the network used to deliver these applications. The success of these attacks on any system is possible through many vulnerabilities such as weak defaults, software bugs, user error, and inherent vulnerabilities in Internet protocols.

Countermeasures that you can use against viruses, Trojan horses, and worms include:

- Stay current with the latest operating system service packs and software patches;
- Block all unnecessary ports at the firewall and host;
- Disable unused functionality including protocols and services;
- Harden weak, default configuration settings.

FOOTPRINTING

Examples of footprinting are port scans, ping sweeps that can be used by attackers to glean valuable system-level information to help prepare for more significant attacks. The type of information potentially revealed by footprinting includes account details, operating system and other software versions, server names, and database schema details.

Countermeasures to help prevent footprinting include:

- Disable unnecessary protocols;
- Lock down ports with the appropriate firewall configuration;

- Use TCP/IP and IPSec filters for defense in depth;
- Configure IIS to prevent information disclosure through banner grabbing;
- Use IDS that can be configured to pick up footprinting patterns and reject suspicious traffic.

Password Cracking

If the attacker cannot establish an anonymous connection with the server, he or she will try to establish an authenticated connection. For this, the attacker must know a valid username and password combination. If you use default account names, you are giving the attacker a head start. Then the attacker only has to crack the account's password. The use of blank or weak passwords makes the attacker's job even easier.

Countermeasures to help prevent password cracking include:

- Use strong passwords for all account types;
- Apply lockout policies to end-user accounts to limit the number of retry attempts that can be used to guess the password;
- Do not use default account names, and rename standard accounts such as the administrator's account and the anonymous Internet user account used by many Web applications;
- Audit failed logins for patterns of password hacking attempts.

Denial of Service

Denial of service can be attained by many methods aimed at several targets within your infrastructure. At the host, an attacker can disrupt service by brute force against your application, or an attacker may know of a vulnerability that exists in the service your application is hosted in or in the operating system that runs your server.

Countermeasures to help prevent denial of service include:

- Configure your applications, services, and operating system with denial of service in mind;
- Stay current with patches and security updates;
- Harden the TCP/IP stack against denial of service;
- Make sure your account lockout policies cannot be exploited to lock out well known service accounts;
- Make sure your application is capable of handling high volumes of traffic and that thresholds are in place to handle abnormally high loads;
- Review your application's failover functionality;
- Use IDS that can detect potential denial of service attacks.

Arbitrary Code Execution

If an attacker can execute malicious code on your server, the attacker can either compromise server resources or mount further attacks against downstream systems. The risks posed by arbitrary code execution increase if the server process under which the attacker's code runs is over-privileged. Common vulnerabilities include weak IIS configuration and unpatched servers that allow path traversal and buffer overflow attacks, both of which can lead to arbitrary code execution.

Countermeasures to help prevent arbitrary code execution include:

- Configure IIS to reject URLs with "../" to prevent path traversal;
- Lock down system commands and utilities with restricted ACLs;
- Stay current with patches and updates to ensure that newly discovered buffer overflows are speedily patched.

Unauthorized Access

Inadequate access controls could allow an unauthorized user to access restricted information or perform restricted operations.

Countermeasures to help prevent unauthorized access include:

- Configure secure Web permissions;
- Lock down files and folders with restricted file system permissions;
- Use native frameworks for access control mechanisms enablement, including URL authorization and principal permission demands.

Application threats

Application-level threats are organized by application vulnerability category, as listed below:

- Input validation;
- Authentication;
- Authorization;
- Configuration management;
- Sensitive data;
- Session management;
- Cryptography;
- Parameter manipulation;
- Exception management;
- Auditing and logging.

Input Validation

Input validation is a security issue if an attacker discovers that your application makes unfounded assumptions about the type, length, format, or range of input data. The attacker can then supply carefully crafted input that compromises your application.

When network and host level entry points are fully secured; the public interfaces exposed by web applications become the only source of attack. The input to web applications is a means to both test the system and a way to execute code on an attacker's behalf.

The following section examines these vulnerabilities in detail, including what makes these vulnerabilities possible.

Buffer Overflows

Buffer overflow vulnerabilities can lead to denial of service attacks or code injection. A denial of service attack causes a process crash; code injection alters the program execution address to run an attacker's injected code.

Well managed code is not susceptible to this problem. This makes the threat of buffer overflow attacks on managed code much less of an issue. It is still a concern, however, especially where managed code calls unmanaged APIs or COM objects.

Countermeasures to help prevent buffer overflows include:

- Perform thorough input validation. This is the first line of defense against buffer overflows. Although a bug may exist in an application that permits expected input to reach beyond the bounds of a container, unexpected input will be the primary cause of this vulnerability. Constrain input by validating it for type, length, format and range;
- When possible, limit application's use of unmanaged code, and thoroughly inspect the unmanaged APIs to ensure that input is properly validated;
- Inspect the managed code that calls the unmanaged API to ensure that only appropriate values can be passed as parameters to the unmanaged API.

Cross-Site Scripting

An XSS attack can cause arbitrary code to run in a user's browser while the browser is connected to a trusted Web site. The attack targets your application's users and not the application itself, but it uses your application as the vehicle for the attack.

Because the script code is downloaded by the browser from a trusted site, the browser has no way of knowing that the code is not legitimate. Internet Explorer security zones provide no defense. Since the attacker's code has access to the cookies associated with the trusted site and are stored on the user's local computer, a user's authentication cookies are typically the target of attack.

Countermeasures to prevent XSS include:

- Perform thorough input validation. Web applications must ensure that input from query strings, form fields, and cookies are valid for the application. Consider all users' input as possibly malicious, and filter or sanitize for the context of the downstream code. Validate all input for known valid values and then reject all other input. Use regular expressions to validate input data received via HTML form fields, cookies, and query strings;

- Use HTML en URL encoding functions to encode any output that includes users' input. This converts executable script into harmless HTML.

For more information see http://www.faresweb.net/web/web-security-cross-site-scripting-xss.

SQL Injection

A SQL injection attack exploits vulnerabilities in input validation to run arbitrary commands in the database. It can occur when your application uses input to construct dynamic SQL statements to access the database. It can also occur if your code uses stored procedures that are passed strings that contain unfiltered user input. Using the SQL injection attack, the attacker can execute arbitrary commands in the database. The issue is magnified if the application uses an over-privileged account to connect to the database. In this instance it is possible to use the database server to run operating system commands and potentially compromise other servers, in addition to being able to retrieve, manipulate, and destroy data.

Countermeasures to prevent SQL injection include:

- Perform thorough input validation. Your application should validate its input prior to sending a request to the database;
- Use parameterized stored procedures for database access to ensure that input strings are not treated as executable statements. If you cannot use stored procedures, use SQL parameters when you build SQL commands;
- Use least privileged accounts to connect to the database.

For more information see http://www.faresweb.net/web/web-security-sql-injection.

Canonicalization

Different forms of input that resolve to the same standard name (the canonical name), is referred to as canonicalization. Code is particularly susceptible to canonicalization issues if it makes security decisions based on the name of a resource that is passed to the program as input. Files, paths, and URLs are resource types that are vulnerable to canonicalization because in each case there are many different ways to represent the same name.

Ideally, applications code should not accept input file names. If it does, the name should be converted to its canonical form prior to making security decisions, such as whether access should be granted or denied to the specified file.

Countermeasures to address canonicalization issues include:

- Avoid using file names as input where possible and instead use absolute file paths that cannot be changed by the end user;
- Make sure that file names are well formed (if you must accept file names as input) and validate them within the context of your application. For example, check that they are within your application's directory hierarchy;
- Ensure that the character encoding is set correctly to limit how input can be represented.

Authentication

Depending on requirements, there are several available authentication mechanisms to choose from. If they are not correctly chosen and implemented, the authentication mechanism can expose vulnerabilities that attackers can exploit to gain access to your system.

Network Eavesdropping

If authentication credentials are passed in plaintext from client to server, an attacker armed with rudimentary network monitoring software on a host on the same network can capture traffic and obtain user names and passwords.

Countermeasures to prevent network eavesdropping include:

- Use authentication mechanisms that do not transmit the password over the network such as Kerberos protocol or others;
- Make sure passwords are encrypted (if you must transmit passwords over the network) or use an encrypted communication channel, for example with SSL.

Brute Force Attacks

Brute force attacks rely on computational power to crack hashed passwords or other secrets secured with hashing and encryption. To mitigate the risk, use strong passwords. Additionally, use hashed passwords with salt; this slows down the attacker considerably and allows sufficient time for countermeasures to be activated.

Dictionary Attacks

This attack is used to obtain passwords. Most password systems do not store plaintext passwords or encrypted passwords. They avoid encrypted passwords because a compromised key leads to the compromise of all passwords in the data store. Lost keys mean that all passwords are invalidated.

Most user store implementations hold password hashes (or digests). Users are authenticated by re-computing the hash based on the user-supplied password value and comparing it against the hash value stored in the database. If an attacker manages to obtain the list of hashed passwords, a brute force attack can be used to crack the password hashes.

With the dictionary attack, an attacker uses a program to iterate through all of the words in a dictionary (or multiple dictionaries in different languages) and computes the hash for each word. The resultant hash is compared with the value in the data store. Weak passwords will be cracked quickly.

Countermeasures to prevent dictionary attacks include:

- Use strong passwords that are complex, are not regular words, and contain a mixture of upper case, lower case, numeric, and special characters;
- Store non-reversible password hashes in the user store. Also combine a salt value (a cryptographically strong random number) with the password hash.

Cookie Replay Attacks

With this type of attack, the attacker captures the user's authentication cookie using monitoring software and replays it to the application to gain access under a false identity.

Countermeasures to prevent cookie replay include:

- Use an encrypted communication channel provided by SSL whenever an authentication cookie is transmitted;
- Use a cookie timeout to a value that forces authentication after a relatively short time interval. Although this doesn't prevent replay attacks, it reduces the time interval in which the attacker can replay a request without being forced to re-authenticate because the session has timed out.

Credential Theft

If the web application implements its own user store containing user account names and passwords, compare its security to the credential stores provided by the platform, for example, a LDAP directory service. Browser history and cache also store user login information for future use. If the terminal is accessed by someone other than the user who logged on, and the same page is hit, the saved login will be available.

Countermeasures to help prevent credential theft include:

- Use and enforce strong passwords;
- Store password verifiers in the form of one way hashes with added salt;
- Enforce account lockout for end-user accounts after a set number of retry attempts;
- To counter the possibility of the browser cache allowing login access, create functionality that either allows the user to choose to not save credentials, or force this functionality as a default policy.

Authorization

Based on user identity and role membership, authorization to a particular resource or service is either allowed or denied.

Elevation of Privilege

When designing an authorization model, you must consider the threat of an attacker trying to elevate privileges to a powerful account such as a member of the local administrators group or the local system account. By doing this, the attacker is able to take complete control over the application and local machine.

The main countermeasure that can be used to prevent elevation of privilege is to use least privileged process, service, and user accounts.

Disclosure of Confidential Data

The disclosure of confidential data can occur if sensitive data can be viewed by unauthorized users. Confidential data includes application specific data such as credit card numbers, employee details, financial records and so on together with application configuration data such as service account credentials and database connection strings. To prevent the disclosure of confidential data you should secure it in persistent stores such as databases and configuration files, and during transit over the network. Only authenticated and authorized users should be able to access the data that is specific to them. Access to system level configuration data should be restricted to administrators.

Countermeasures to prevent disclosure of confidential data include:

- Perform role checks before allowing access to the operations that could potentially reveal sensitive data;
- Use strong ACLs to secure system resources;
- Use standard encryption to store sensitive data in configuration files and databases.

Data Tampering

Data tampering refers to the unauthorized modification of data.

Countermeasures to prevent data tampering include:

- Use strong access controls to protect data in persistent stores to ensure that only authorized users can access and modify the data;
- Use role-based security to differentiate between users who can view data and users who can modify data.

Luring Attacks

A luring attack occurs when an entity with few privileges is able to have an entity with more privileges perform an action on its behalf.

To counter the threat, you must restrict access to trusted code with the appropriate authorization. Using native frameworks code access security helps in this respect by authorizing calling code whenever a secure resource is accessed or a privileged operation is performed.

Configuration Management

Many applications support configuration management interfaces and functionality to allow operators and administrators to change configuration parameters, update Web site content, and to perform routine maintenance.

Unauthorized Access to Administration Interfaces

Administration interfaces are often provided through additional Web pages or separate Web applications that allow administrators, operators, and content developers to managed site content and configuration. Administration interfaces such as these should be available only to restricted and authorized users. Malicious users able to access a configuration management function can potentially deface the Web site, access downstream systems and databases, or take the application out of action altogether by corrupting configuration data.

Countermeasures to prevent unauthorized access to administration interfaces include:

- Minimize the number of administration interfaces;
- Use strong authentication, for example, by using certificates;
- Use strong authorization with multiple gatekeepers;
- Consider supporting only local administration. If remote administration is absolutely essential, use encrypted channels, for example, with VPN technology or SSL, because of the sensitive nature of the data passed over administrative interfaces. To further reduce risk, also consider using IPSec policies to limit remote administration to computers on the internal network.

Unauthorized Access to Configuration Stores

Because of the sensitive nature of the data maintained in configuration stores, you should ensure that the stores are adequately secured.

Countermeasures to protect configuration stores include:

- Configure restricted ACLs on text-based configuration files;
- Keep custom configuration stores outside of the Web space. This removes the potential to download Web server configurations to exploit their vulnerabilities.

Retrieval of Plaintext Configuration Secrets

Restricting access to the configuration store is a must. As an important defense in depth mechanism, you should encrypt sensitive data such as passwords and connection strings. This helps prevent external attackers from obtaining sensitive configuration data. It also prevents rogue administrators and internal employees from obtaining sensitive details such as database connection strings and account credentials that might allow them to gain access to other systems.

Lack of Individual Accountability

Lack of auditing and logging of changes made to configuration information threatens the ability to identify when changes were made and who made those changes. When a breaking change is made either by an honest operator error or by a malicious change to grant privileged access, action must first be taken to correct the change. Then apply preventive measures to prevent breaking changes to be introduced in the same manner. Keep in mind that auditing and logging can be circumvented by a shared account; this applies to both administrative and user/application/service accounts. Administrative accounts must not be shared. User/application/service accounts must be assigned at a level that allows the identification of a single source of access using the account, and that contains any damage to the privileges granted that account.

Over-privileged Application and Service Accounts

If application and service accounts are granted access to change configuration information on the system, they may be manipulated to do so by an attacker. The risk of this threat can be mitigated by adopting a policy of using least privileged service and application accounts. Be wary of granting accounts the ability to modify their own configuration information unless explicitly required by design.

Sensitive Data

Sensitive data is subject to a variety of threats. Attacks that attempt to view or modify sensitive data can target persistent data stores and networks.

Access to Sensitive Data in Storage

Sensitive data must be secured in storage to prevent a user — malicious or otherwise — from gaining access to and reading the data.

Countermeasures to protect sensitive data in storage include:

- Use restricted ACLs on the persistent data stores that contain sensitive data;
- Store encrypted data;

- Use identity and role-based authorization to ensure that only the user or users with the appropriate level of authority are allowed access to sensitive data. Use role-based security to differentiate between users who can view data and users who can modify data.

Network Eavesdropping

The HTTP data for Web application travels across networks in plaintext and is subject to network eavesdropping attacks, where an attacker uses network monitoring software to capture and potentially modify sensitive data.

Countermeasures to prevent network eavesdropping and to provide privacy include:

- Encrypt the data;
- Use an encrypted communication channel, for example, SSL.

Data Tampering

Data tampering refers to the unauthorized modification of data, often as it is passed over the network.

One countermeasure to prevent data tampering is to protect sensitive data passed across the network with tamper-resistant protocols such as hashed message authentication codes (HMACs).

Session Management

Session management for Web applications is an application layer responsibility. Session security is critical to the overall security of the application.

Session Hijacking

A session hijacking attack occurs when an attacker uses network monitoring software to capture the authentication token (often a cookie) used to represent a user's session with an application. With the captured cookie, the attacker can spoof the user's session and gain access to the application. The attacker has the same level of privileges as the legitimate user.

Countermeasures to prevent session hijacking include:

- Use SSL to create a secure communication channel and only pass the authentication cookie over an HTTPS connection;
- Implement logout functionality to allow a user to end a session that forces authentication if another session is started;
- Make sure you limit the expiration period on the session cookie if you do not use SSL. Although this does not prevent session hijacking, it reduces the time window available to the attacker.

Session Replay

Session replay occurs when a user's session token is intercepted and submitted by an attacker to bypass the authentication mechanism. For example, if the session token is in plaintext in a cookie or URL, an attacker can sniff it. The attacker then posts a request using the hijacked session token.

Countermeasures to help address the threat of session replay include:

- Re-authenticate when performing critical functions. For example, prior to performing a monetary transfer in a banking application, make the user supply the account password again;
- Expire sessions appropriately, including all cookies and session tokens;
- Create a "do not remember me" option to allow no session data to be stored on the client.

Man in the Middle Attacks

A man in the middle attack occurs when the attacker intercepts messages sent between you and your intended recipient. The attacker then changes your message and sends it to the original recipient. The recipient receives the message, sees that it came from you, and acts on it. When the recipient sends a message back to you, the attacker intercepts it, alters it, and returns it to you. You and your recipient never know that you have been attacked.

Any network request involving client-server communication, including Web requests and calls to remote components and Web services, are subject to man in the middle attacks.

Countermeasures to prevent man in the middle attacks include:

- Use cryptography. If you encrypt the data before transmitting it, the attacker can still intercept it but cannot read it or alter it. If the attacker cannot read it, he or she cannot know which parts to alter. If the attacker blindly modifies your encrypted message, then the original recipient is unable to successfully decrypt it and, as a result, knows that it has been tampered with;
- Use Hashed Message Authentication Codes (HMACs). If an attacker alters the message, the recalculation of the HMAC at the recipient fails and the data can be rejected as invalid.

Cryptography

Most applications use cryptography to protect data and to ensure it remains private and unaltered.

Poor Key Generation or Key Management

Attackers can decrypt encrypted data if they have access to the encryption key or can derive the encryption key. Attackers can discover a key if keys are managed poorly or if they were generated in a non-random fashion.

Countermeasures to address the threat of poor key generation and key management include:

- Use built-in encryption routines that include secure key management;
- Use strong random key generation functions and store the key in a restricted location — for example, in a registry key secured with a restricted ACL — if you use an encryption mechanism that requires you to generate or manage the key;
- Encrypt the encryption key;
- Expire keys regularly.

Weak or Custom Encryption

An encryption algorithm provides no security if the encryption is cracked or is vulnerable to brute force cracking. Custom algorithms are particularly vulnerable if they have not been tested. Instead,

use published, well-known encryption algorithms that have withstood years of rigorous attacks and scrutiny.

Countermeasures that address the vulnerabilities of weak or custom encryption include:

- Do not develop your own custom algorithms;
- Use the proven cryptographic services provided by the platform;
- Stay informed about cracked algorithms and the techniques used to crack them.

Checksum Spoofing

Do not rely on hashes to provide data integrity for messages sent over networks. Hashes such as Secure Hash Algorithm (SHA1) and Message Digest compression algorithm (MD5) can be intercepted and changed.

To counter this attack, use a MAC or HMAC. The Message Authentication Code Triple Data Encryption Standard (MACTripleDES) algorithm computes a MAC, and HMACSHA1 computes an HMAC. Both use a key to produce a checksum. With these algorithms, an attacker needs to know the key to generate a checksum that would compute correctly at the receiver.

Parameter Manipulation

Parameter manipulation attacks are a class of attack that relies on the modification of the parameter data sent between the client and Web application. This includes query strings, form fields, cookies, and HTTP headers.

Query String Manipulation

Users can easily manipulate the query string values passed by HTTP GET from client to server because they are displayed in the browser's URL address bar. If your application relies on query string values to make security decisions, or if the values represent sensitive data such as monetary amounts, the application is vulnerable to attack.

Countermeasures to address the threat of query string manipulation include:

- Avoid using query string parameters that contain sensitive data or data that can influence the security logic on the server. Instead, use a session identifier to identify the client and store sensitive items in the session store on the server;
- Choose HTTP POST instead of GET to submit forms;
- Encrypt query string parameters.

For more information see http://www.faresweb.net/web/web-security-cross-site-request-forgeries-csrf.

Form Field Manipulation

The values of HTML form fields are sent in plaintext to the server using the HTTP POST protocol. This may include visible and hidden form fields. Form fields of any type can be easily modified and client-side validation routines bypassed. As a result, applications that rely on form field input values to make security decisions on the server are vulnerable to attack.

To counter the threat of form field manipulation, instead of using hidden form fields, use session identifiers to reference state maintained in the state store on the server.

Cookie Manipulation

Cookies are susceptible to modification by the client. This is true of both persistent and memory-resident cookies. A number of tools are available to help an attacker modify the contents of a memory-resident cookie. Cookie manipulation is the attack that refers to the modification of a cookie, usually to gain unauthorized access to a Web site.

While SSL protects cookies over the network, it does not prevent them from being modified on the client computer. To counter the threat of cookie manipulation, encrypt and use an HMAC with the cookie.

HTTP Header Manipulation

HTTP headers pass information between the client and the server. The client constructs request headers while the server constructs response headers. If your application relies on request headers to make a decision, your application is vulnerable to attack.

Do not base your security decisions on HTTP headers. For example, do not trust the HTTP Referer to determine where a client came from because this is easily falsified.

Exception Management

Exceptions that are allowed to propagate to the client can reveal internal implementation details that make no sense to the end user but are useful to attackers. Applications that do not use exception handling or implement it poorly are also subject to denial of service attacks.

Attacker Reveals Implementation Details

Usually, providing rich exception details are invaluable to developers. If the same information is allowed to fall into the hands of an attacker, it can greatly help the attacker exploit potential vulnerabilities and plan future attacks. The type of information that could be returned includes platform versions, server names, SQL command strings, and database connection strings.

Countermeasures to help prevent internal implementation details from being revealed to the client include:

- Use exception handling throughout your application's code base;
- Handle and log exceptions that are allowed to propagate to the application boundary;
- Return generic, harmless error messages to the client.

Denial of Service

Attackers will probe a Web application, usually by passing deliberately malformed input. They often have two goals in mind. The first is to cause exceptions that reveal useful information and the second is to crash the Web application process. This can occur if exceptions are not properly caught and handled.

Countermeasures to help prevent application-level denial of service include:

- Thoroughly validate all input data at the server;
- Use exception handling throughout your application's code base.

Auditing and Logging

Auditing and logging should be used to help detect suspicious activity such as footprinting or possible password cracking attempts before an exploit actually occurs. It can also help deal with the threat of repudiation. It is much harder for a user to deny performing an operation if a series of synchronized log entries on multiple servers indicate that the user performed that transaction.

User Denies Performing an Operation

The issue of repudiation is concerned with a user denying that he or she performed an action or initiated a transaction. You need defense mechanisms in place to ensure that all user activity can be tracked and recorded.

Countermeasures to help prevent repudiation threats include:

- Audit and log activity on the Web server and database server, and on the application server as well, if you use one;
- Log key events such as transactions and login and logout events;
- Do not use shared accounts since the original source cannot be determined.

Attackers Exploit an Application Without Leaving a Trace

System and application-level auditing is required to ensure that suspicious activity does not go undetected.

Countermeasures to detect suspicious activity include:

- Log critical application level operations;
- Use platform-level auditing to audit login and logout events, access to the file system, and failed object access attempts;
- Back up log files and regularly analyze them for signs of suspicious activity.

Attackers Cover Their Tracks

Your log files must be well-protected to ensure that attackers are not able to cover their tracks.

Countermeasures to help prevent attackers from covering their tracks include:

- Secure log files by using restricted ACLs.
- Relocate system log files away from their default locations.

Summary of Threats and Vulnerabilities

The table below provides a synthesis of the most important threats and vulnerabilities.

Threat ID	Threat	Vulnerabilities	Description
1	Input validation	• Attacks performed by embedding malicious strings in query strings, form fields, cookies, and HTTP headers. These include command execution, cross-site scripting (XSS), SQL injection, and buffer overflow attacks	• How do you know that the input that your application receives is valid and safe? Input validation refers to how your application filters, scrubs, or rejects input before additional processing
2	Authentication	• Identity spoofing, password cracking, elevation of privileges, and unauthorized access.	• "Who are you?" Authentication is the process where an entity proves the identity of another entity, typically through credentials, such as a

			user name and password.
3	Authorization	• Access to confidential or restricted data, tampering, and execution of unauthorized operations.	• "What can you do?" Authorization is how your application provides access controls for resources and operations.
4	Configuration Management	• Unauthorized access to administration interfaces, ability to update configuration data, and unauthorized access to user accounts and account profiles.	• Who does your application run as? Which databases does it connect to? How is your application administered? How are these settings secured? Configuration management refers to how your application handles these operational issues.
5	Sensitive Data	• Confidential information disclosure and data tampering.	• Sensitive data refers to how your application handles any data that must be protected either in memory, over the wire, or in persistent stores.
6	Session Management	• Capture of session identifiers resulting in session hijacking and identity spoofing.	• A session refers to a series of related interactions between a user and your Web application. Session management refers to how your application handles and protects these interactions.
7	Cryptography	• Access to confidential data or account credentials, or both.	• How are you keeping secrets, secret (confidentiality)? How are you tamper proofing your data or libraries (integrity)? How are you providing seeds for random values that must be cryptographically strong? Cryptography refers to how your application enforces confidentiality and integrity.
8	Parameter Manipulation	• Path traversal attacks, command execution, and bypass of access control mechanisms among others, leading to information disclosure, elevation of privileges, and denial of service.	• Form fields, query string arguments, and cookie values are frequently used as parameters for your application. Parameter manipulation refers to both how your application safeguards tampering of these values and how your application processes input parameters.
9	Exception Management	• Denial of service and disclosure of sensitive system level details.	• When a method call in your application fails, what does your application do? How much do you reveal? Do you return friendly error information to end users? Do you pass valuable exception information back to the caller? Does your application fail gracefully?
10	Auditing and Logging	• Failure to spot the signs of intrusion, inability to prove a user's actions, and difficulties in problem diagnosis.	• Who did what and when? Auditing and logging refer to how your application records security-related events.

WEB SECURITY PRINCIPLES

Principle #1: End-to-end approach for security architecture.

Issue	Run security through multi-layered approach, defense in depth and diversity of defense.
Concern	Security shall focus on infrastructure, Identity Management, Authentication/Authorization, Logging, Non-Repudiation, and Secure Development.

Principle #2: Security functions must be externalized from the application and preferably use off-the-shelf products before developing home-grown security solutions.

Issue	Implement defense in depth. Separation of duty between development teams. Security specialist must concentrate their development on external system and not dive in business application. Critical security component to review are centralized. Secure system/application development is a specialized area and the risk of introducing vulnerabilities and giving a false sense of security is high with custom developed solutions.
Concern	Dedicated components must provide security functionalities.

Principle #3: Network communications must be limited to needed protocols. In-depth perimeter defense must be implemented.

Issue	The environment must be under control and must be used for what it's intend to. Limiting the number of protocols and services reachable from untrusted networks makes it more controllable and manageable.
Concern	The protocols opened must be limited and segregated by DMZ and server/application type. Systems must be hardened.

Principle #4: All internet communications must be inspected for malicious service requests.

Issue	The application must be under control and must be used for what it's intend to. Application attacks can use tunnels embedded inside common protocols used on the internet, for instance HTTP. This type of attack can be detected by protocol inspection.
Concern	For web applications this must be done by implementing http protocol inspection in using a Web Application Firewall in front of web applications. Communication between client-side and server-side web applications must use http/https protocol with a standard and readable data format.

Principle #5: Authentication enforcement and SSO functionality must be provided with no application adherence.

Issue	Implement defense in depth regarding authentication. Integration of applications which rely on different platforms becomes less complex when externalizing Single Sign On and authentication functionality. Separation of duty between development teams "full authentication chain does not rely on one business developer". Security specialist must concentrate their development on external system and not dive in business application. Critical security component to review are centralized.
Concern	Authentication enforcement point must be externalized from the applications. *Web-SSO* solution can provide such functionality. The platform must be compatible with an external Web-SSO solution.

Principle #6: Authentication mechanism must be reusable by different websites or commercial packages.

Different authentication mechanisms & devices (OTP, SMS, Challenge-response…) must be supported.

Issue	Application shall not implement its own authentication method and authentication validation.
Concern	Dedicated components must provide authentication enforcement, authentication validation and authentication interface functionality. Web-SSO, security services abstraction layer can provide such functionality.

Principle #7: Transaction signature functionality must be externalized from the application. Depending on authentication mechanism & device capabilities, WYSIWYS (What You See Is What You Sign) should be supported.

Issue	Application shall not implement its own transaction signature method and validation. WYSIWYS should be implemented to protect financial application against Financial Trojan threats (MITB attacks) when it's possible.
Concern	Dedicated components must provide transaction signature enforcement, validation and interface functionality. Security service abstraction layer can provide such functionality.

Principle #8: Coarse grained authorization (control access to the business application) shall be implemented and the authorization controls shall preferably be externalized from the application.

Middle grained authorization aiming to control access to the business functionalities shall be provided within the

application and based on Role Based Access Control. Fine grained authorization aiming to control the accesses to the data/functionalities shall be provided by the application and based on user specific information or specific rules based algorithm.	
Issue	Implement defense in depth regarding authorization. Make it easy to integrate new applications to the coarse grained authorization services. Separation of duties between business developer and security developer.
Concern	Coarse grained authorization control shall be externalized from the applications. *Web-SSO* solution must provide such functionality.
Principle #9: Each event concerning financial transactions shall be logged using the following "6W[1]" rule and kept for an agreed period of time.	
Issue	Logs recording security events must be produced to assist in future investigations. Preserve this logging information, which will make it valuable in regard of the law or local regulation authority.
Concern	Provide configurable functionalities to log user events, application events and security events. Provide dedicated storage system for retaining logs for an extended period of time. Security logs and technical logs must not be mixed together. Security logs are confidential and must be protected against data leak. Logging must be centralized with all "6W rule" information at middle end servers layer, no concern's for back-end core system. Purpose is to manage non repudiation feature.
Principle #10: The integrity and non-repudiation of critical transactions should be ensured. The signature mechanism used to provide this assurance, shall provide integrity and non-repudiation over all information processed by the transaction.	
Issue	In any case you have/want to proof actions of the customers. Logging facilities and security log information must be protected against tampering to keep them valuable toward the law.
Concern	A digital signature should be implemented using transaction information as an input and other security components like timestamp and transaction reference number….
Principle #11: Application must implement security best practices regarding applications security (OWASP, SANS, WASC, etc.).	
Issue	Application must be protected against web application attacks, incoming threats (web 2.0, mobile …) and e-financial hacking techniques.
Concern	Dedicated libraries must provide functionalities for building secure and robust web applications.
Principle #12: If technical channels (mobile and internet devices for instance) offer different business services protected by different security mechanism, they must be isolated.	
Issue	Avoid cross channel impact due to different security protection mechanism (authentication method for instance) and provided business services.

[1] 6W Rules: Who initiated the action; What was the action; When did the action take place; Where from was it initiated; What was the communication channel used; Was the action successful.

Concern	Front-end Servers should physically or virtually be segregated from similar channels. SSO and session context must not be shared between different channels.

Principle #13: Servers with similar services must be combined in specific DMZ depending on local requirements, compliance and risk acceptance.

Horizontal zoning (n-tier deployment) and vertical zoning (transactional/public deployment) must be supported by the technical reference architecture.

Issue	When servers with similar services are combined in a specific DMZ then access to these servers is limited. This limits the number of protocol (ports) to dedicated purposes. Avoid that attackers can directly exploit vulnerabilities of the web- and application servers. Use firewalls to block outgoing connections initiated from a web- or application server. Anonymous application exposes a greater surface of attack. In order to avoid that anonymous application vulnerability expose transactional or other critical business services, it can be required to separate the public and transactional application on different DMZ.
Concern	Technical application architecture must be flexible and deployable on different DMZ architecture. Inter-applications communication protocol (between tiers) must be compatible with network control point components (network firewall, IDS). Public and transactional services must be deployable on different servers (physically or virtually). Inter-application communication (between public and transactional applications) must not rely on context sharing mechanism.

AN ARCHITECTURE TYPE REFERENCE

The reference security architecture provides a common and flexible end-to-end secure solution based on security principles defined above.

The deployment security model could depend on business requirements, the level of risk accepted by organizations and local regulations.

Flexibility of secure deployment must be taken into account at the technical architecture level. Deployment scenario will impact configurations, application packaging, inter-applications communications, identity propagation …

The Web reference security architecture intends to describe the end-to-end logical view and highlight key security functionalities, services and components.

KEY COMPONENTS AND FUNCTIONALITIES

A generic web application architecture consists of a client at the end-user side, and a 3-tier processing server side (presentation tier, business tier and back-end tier) as shown in figure below. Firewalls are placed between every two tiers to enable network perimeter security. Often, this architecture is simplified by omitting FW3 and/or FW2, and by implementing two or more tiers on the same server. Also, in some deployments the authentication server is directly connected to the web server.

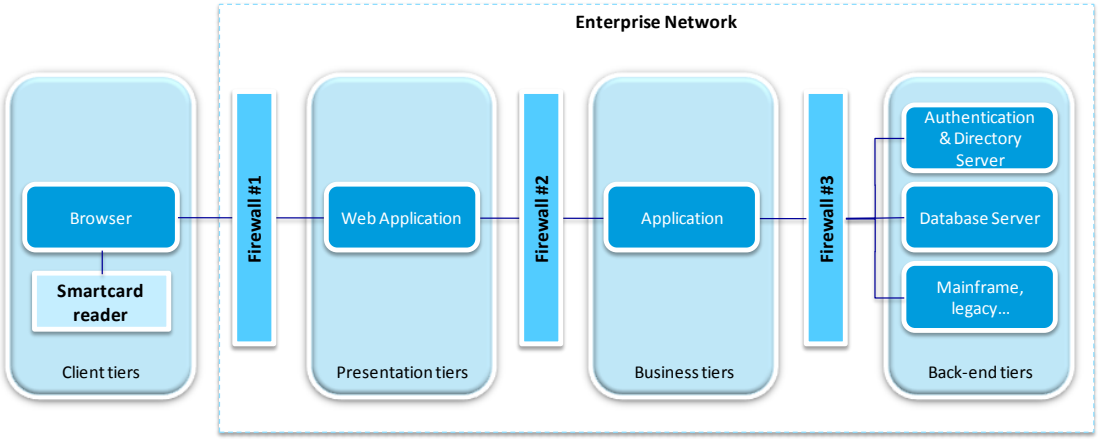


Figure 2 - Web Application Architecture reference

CLIENT TIERS

Basically, the client tier consists of a web browser, possibly extended with client-side application components. Moreover, the client is referred to as a rich client that interacts with the web server through simple HTML over HTTP in a REST-style, or through more complex exchange with JSON or SOAP over HTTP interactions with the web server.

Furthermore, the client can be equipped with a smartcard reader to interface with a smartcard or other security token. Such tokens can be used among others for authenticating the user and protecting the requests.

PRESENTATION TIERS

The presentation tier is responsible for formatting the processed information before returning it to the client, and for handling client requests by performing input validation and delegating them to the appropriate units within the business tier. Usually, the processed information is formatted using the markup languages HTML (in case of a client browser), JSON or XML (in case of a rich client).

Infrastructural components within the presentation tier are typically the web server, whether or not accompanied by a web connector of the application server (PHP like case).

BUSINESS TIERS

The business tier is instantiated in the application server. The application server implements the actual business logic. In order to achieve its functionality, several services can be provided to the application server from the back-end tier.

The web server from the presentation tier can interact with the application server by using remote procedure calls, web services or a proprietary application server protocol.

BACK-END TIERS

The back-end tier provides some basic services to the business tier, such as a database system and an authentication and directory service. The SQL query language is mostly used in requests towards the database system, and both LDAP and X.509 in communication with the directory service. The communication protocol for the authentication service depends on the authentication system used (e.g. Kerberos…).

The back-end tier can also contain back-end systems including mainframes, wrapped legacy applications and interfaces to remote application servers.

KEY COMPONENTS AND RESPONSIBILITIES

Different security functionalities must be provided by the reference security architecture to comply with predefined principles. Functionalities can be grouped within two major categories:

- **Front layers defense:**

  Group of functionalities which provide protection in front of applications: content inspection, Authentication enforcement, Single Sign On, Access control.

  The reverse proxy functionality is not (only) about security. It provides some functional features for an internet platform. However, this component has to be considered within the security chapter as well.

- **Security services:**

  Group of functionalities which provide security services to applications: authentication and authorization interface, transaction signing interface, user context and cross application context management, functional logging interface and enterprise application development security API.

The reference security architecture recommends a scenario where the *reverse proxy, Authentication enforcement, SSO* and *Access control (coarse grained)* functionalities are co-deployed on a unique and dedicated component so called *Web-SSO*.

The purpose of *Web-SSO* is to authenticate and provide access control to web based resources. *Web-SSO* market products may also include IAM, role/rule management and audit capabilities. *Web-SSO* solution provides a policy enforcement solution including Single Sign On (SSO).

REVERSE PROXY

Responsibility:

- Provide a homogenous view to a group of web- and application servers, without leaking distribution over several machines;
- Provide single application entry point to end-user;
- Can offload the Web servers by caching static content as well as dynamic content.

WEB APPLICATION FIREWALL

Responsibility:

- Provides a common entry point for a collection of servers
  - It avoids that attackers have direct access from the outside to web- or application servers: i.e. to any vulnerabilities of the underlying platform (web- and application server software, DB, OS ...);
  - It is hiding the network and host internals;
  - Protects the server software on the level of the application protocol at the network perimeter;

- By filtering all requests, for attacks on protocol level, it allows that only filtered requests will reach the application servers;
- The filtering is detecting attacks (worms, viruses, SQL injection, XSS...) from the internet, trying to exploit vulnerabilities on operating systems (Windows, AIX, Linux...) or server software (DB, Applications server, Web server...) via different protocols (HTTP, FTP...) and languages (JavaScript, Java, PHP…).

- SSL endpoint
  - Identify the server to the customer;
  - Store the public server certificate (and associated private key) in an Hardware Security Module.
- Caching and compression of HTTP payload
  - Increases availability by offloading this functionality from the backend.

AUTHENTICATION SERVICE

Responsibility:

- Perform the authentication or be an interface to multiple authentication servers (provide support to different authentication systems).

Collaboration:

- Handle messages from Logon server (e.g. create the challenge in case of a challenge-response authentication)

LOGON PAGES

Responsibility:

- Web application which provides the authentication and authentication step-up pages.

Collaboration:

- Answer to requests from the Browser for the logon page.

SECURITY SERVICES

Responsibility:

- The Security services are centralized all over the Internet Platform, as for:
  - Transaction signature interface;
  - Authorization interface;
  - Functional and security logging interface;
  - XlogID generation (unique logging ID across logs);
  - Build user context during the authentication process;

- Analyze and log suspicious user events and activities (IP switching and country switching, User-agent switching, Δt between user logging and the first transaction…).

Security services does not host context managers or cross application context needed upon user sessions (cross application context allows sharing data between different applications).

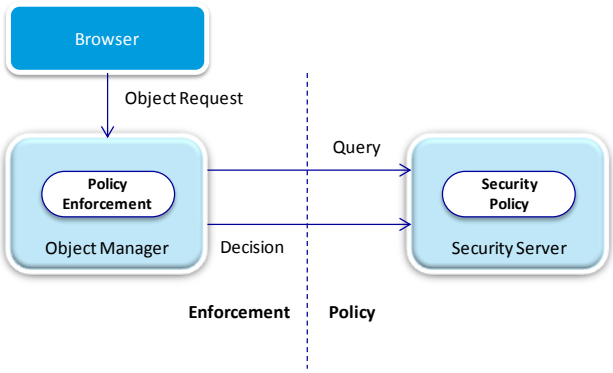


Figure 3 - Security service enforcement

Collaboration:

- Security services validates transaction signature;
- Security services validates user authorization;
- Security services connects with the functional logging solution;
- Portal, Presentation and Business layers accesses Security services to enforce security when required.

WEB-SSO (WAM + REVERSE PROXY FUNCTIONALITIES)

Responsibility:

- Provide a homogenous view to a group of web and application servers, without leaking distribution over several machines, to end users:
  - Easy integration, and transparent for end users, of web applications and different technologies of web- and application servers;
  - Make it easy to integrate new applications into the authentication and authorization services.
- Implement access control to a set of web applications:
  - Authentication (enforcement point)
    - Implement a single user identity for all applications, even when existing web applications already carry their own user data base;
    - Applications need to be independent of the type of authentication;
    - Enforce authentication of users, using different type of authentication;

        - If required, impose different types of authentication depending on the services accessed, the users...;
        - Enforce users to re-authenticate to avoid misuse of a user's session left alone for some time;
        - Centralize authentication enforcement point for all applications (low operational impact);
    - Authorization (coarse grained authorization enforcement point):
        - Only authenticated users are authorized to access a defined subset of the application;
        - If required, impose different access rights depending on the users being authenticated (role-based authorization).
    - Security (authentication) session management:
        - Implement single sign-on: avoid users have to authenticate for each application separately;
        - Pass user identity and additional session related attributes to all backend servers that require this information;
        - Setup a security context for each user and use a security cookie to maintain that session;
        - Protect (some) backend servers from unauthenticated users;
        - Handle single sign-off (user only keeps his session as long as he is active);
        - Strongly hardened server with only security features (no complexity).

Protections:

- External Web-SSO is an important system and network protection level. Indeed all traffic which needs to be authenticated and authorized before to enter the internal network zone must go through this reverse proxy. This system is a highly hardened system running the minimum services necessary;
- In the case of the reverse proxy, this is the first business system that is touched by external traffic prior to authentication. If a reverse proxy is not used, then the web server itself is the front line for authentication, but has many more services running and potential vulnerabilities.

SECURITY MEASURES

Business functionalities provided through web application can be broadly classified under four risk profiles:

- **Public content:** Web sites such as company's institutional information providing static content (such as product information), marketing campaigns (banners) or online tools (such as of credit simulation, insurance) that doesn't make usage of sensitive data;

- **Private content:** Personal user data or relevant transactional operations (such as account overview and balances) can be processed by such applications. This category is related to "query" operations, where no changes or updates occurs;

- **Transactional limited:** Operations can be executed within a limited scope (e.g. fund transfer to pre-registered list of targeted account or trusted beneficiaries) or specific business limitations (e.g. fund transfer based on a maximal allowed amount);

- **Transactional Extended:** fully qualified scope of transactional applications that can be executed.

The approach adopted to describe security measures required for application in this analysis intends to align the recommended security measures to be implemented with the inherent business and technical risks of applications. It is therefore articulated on:

- **A baseline security measures** that must be implemented whatever the functionalities provided by the web application;

- **Additional security measures** depending on the risks profile of the business functionalities made available through the application. For each risk profile, additional security measures may be required to reach an acceptable level of security.

The security approach looks at the complete lifecycle used to deliver and execute the application on the web.

- ***Design***: conceiving the design of the application taking into account critical vulnerabilities in later phases;

- ***Implementation***: developing the application taking into account best practices related to adopted technology;

- ***QA testing***: testing the application;

- ***Operation***: running the application on the web.

Furthermore, the identified security measures (whether baseline or additional) are mapped onto one of the four phases of the web application development lifecycle: Design, Implementation, QA testing and Operation.

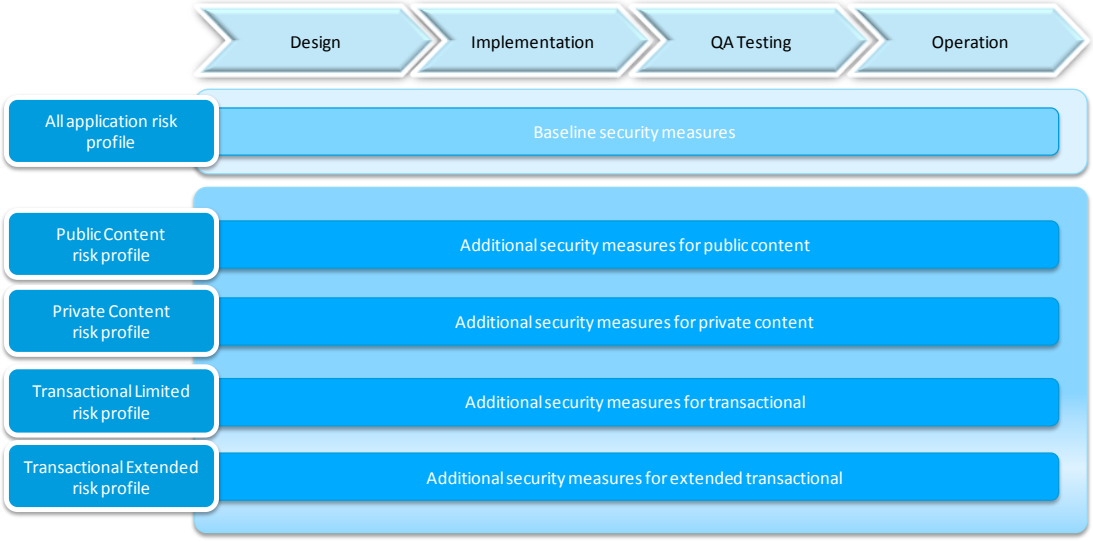


Figure 4 - Security approach lifecyle

MOBILE SECURITY

INTRODUCTION

Most businesses have a comprehensive acceptable use policy, travel and expense policy, telecom and networking policy, and security and risk management policy. And all of these incorporate some aspects of mobility into them. But organizations are finding that these disparate policies don't meet their needs for managing all aspects of mobility. Mobility brings together many aspects of these existing policies, and finding a way to integrate it all into a mobile policy isn't easy. Compounding the problem is that there is no dedicated group or function within IT that is responsible for managing and securing it all. Additionally, the technology and vendor landscape evolves quickly; you need to keep your business's mobile policy current, but it's a challenge finding people with the bandwidth to revisit the policy quarterly.

IT SHOULD MAKE DEVICE MANAGEMENT AND SECURITY THE CORE OF ITS MOBILE POLICY

As lame or uninteresting device management is, it needs to be a top priority for any organization that strives to be smarter about mobility. In fact, a well-thought-out and heavily collaborative mobile device management policy should be the foundation of any organization's business mobility strategy because it: 1) ensures all the moving parts are functioning soundly, and 2) serves as the console that your mobile operations team uses to manage and secure all of your mobile assets.

SECURITY POLICY

When devices are lost or stolen, the data needs to stay secure. Your highest priority should be the security policies within the mobile device management solution, which include strong password enforcement, power-on passwords, authentication, multiple encryption options (e.g., the ability to use and encrypt data on removable storage cards), multiple user authentication options, and credential expiration options. And of course the No. 1 feature that any mobile device management point product or suite should deliver is the ability for IT — and even in some cases, the end user — to remotely lock and (selectively or completely) wipe a lost or stolen device.

MOBILE CONNECTIVITY BECOMES A BUSINESS PRIORITY

Following the trend of cell phones becoming smart computers, mobile devices for business require advanced business functions to enable users to conduct business regardless of location.

A native mobile application is a thick client application installed on mobile device (Smartphone / tablets) downloaded from an Application Store (such as Apple App Store, Google Android Market, Microsoft Market Place and so forth) leveraging the enriched ergonomical and user interface (UI) capabilities of these devices compared to what could be achieved in a more casual web mobile application relying on a web browser.

As businesses embrace mobility, IT operations professionals are facing new challenges. Most organizations today are simply trying to get smarter about how to manage and secure their increasingly mobile population and distributed assets.

The key is to make mobile device management and security the foundation of your business's mobile strategy. Security is a key feature for mobility to allow end-users to conduct business in a public setting.

SECURITY MANAGEMENT IS KEY FEATURE TO ALLOW BUSINESS GOES MOBILE

Developing an appealing mobile offer for customers is one of the top priorities for many organizations.

One of the key business requirements of Internet and mobile platform is to support multiple presentation technologies and devices – going from pure web-based interfaces to the different and emerging mobile devices (Smartphones and tablets).

Security risks are certainly one of the key challenges raised when publishing applications.

Main risks and mitigation measures to address security regarding mobile applications are described here below:

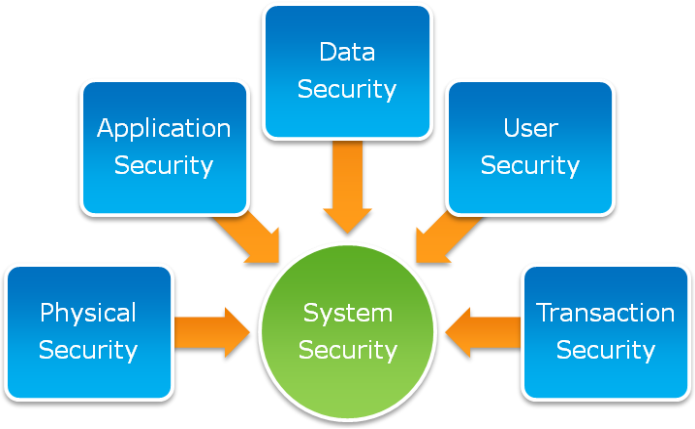


Figure 5 - Mobile risks and mitigation measures

- ***Physical security***: protecting from unauthorized device access to the mobile service;
- ***Application security***: protecting from application modification through verification and trust;
- ***Data security***: protecting from the modification or disclosure of the data;
- ***User security***: Protecting from unauthorized user access to the mobile service – ensuring users get the access they should have;
- ***Transaction security***: protecting from unauthorized transaction execution – ensuring transactions are sent and received by the parties claiming to have sent and received the message (non repudiation).

The purpose of this section is to provide a generic risk analysis of native mobile applications. Follows a risk-oriented approach distinguished by application risk profiles and a proposal for a set of risk-mitigation measures related to the involved risk for each one of them.

Risk acceptance decisions must nevertheless be fully aligned with a group-based security risk management strategy for mobile applications, in order to globally manage group-wide security risks such as reputational risks. This document aims to help defining and enforcing such a strategy.

MOBILE SECURITY MANAGEMENT SUPPORTS MOBILE PROJECTS LIFECYCLE

The security approach looks at the complete lifecycle used to deliver and execute the application on the mobile phone device.

- ***Application distribution***: publishing the application on the Application store and make it available for customers;
- ***Application installation and start-up***: installation of a application's instance on the phone/ tablet device; start-up of the application on the phone/tablet device;
- ***Operation***: execution of the application on the phone/tablet device.

Furthermore, the identified security measures (whether baseline or additional) are mapped onto one of the three phases of the mobile application development lifecycle: Distribution, Installation, and Execution.

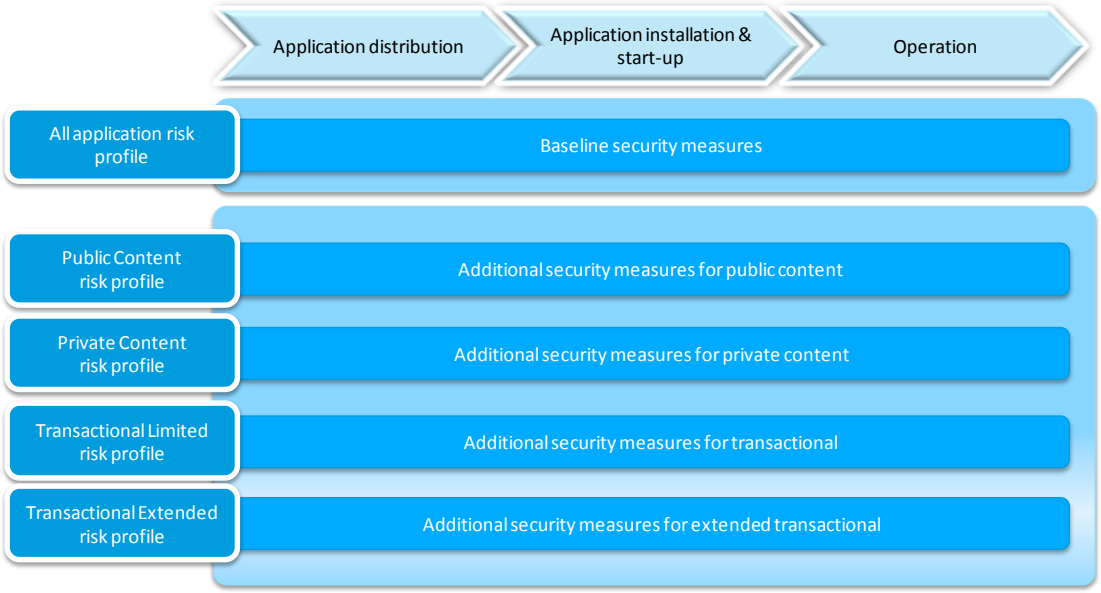


Figure 6 - Mobile security lifecycle management

MOBILE THREATS ANALYSIS

The aim of this section is to describe the threats against mobile applications running on a Smartphone or Tablet devices, as opposed to a purely browser-based mobile solution.

The scope of the threat analysis provided in this section is based on the following assumptions:

- Mobile device and Operating System: iPhone and iPad / IOS;
- Client software type: native client application.

This generic threat analysis is positioned within the context of a generic project, such as:

- Regulatory requirements;
- Business and functional requirements specification;

- Native client application distribution model;
- Customer registration and initial credential distribution processes;
- Enterprise specific deployment architecture.

These aspects are required to have a concrete view on underlying vulnerabilities, associated business and regulatory risks and will determine relevant mitigation measures.

This threat analysis does not cover aspects below:

- Threats for purely browser-based mobile (e.g. no client side code or native application);
- Threats exploiting server-side vulnerabilities, such as OWASP vulnerabilities XSS, CSRF, SQL Injection, XML poisoning, directly affecting the Presentation layer, Execution Engine, Application Layer, or Backend Services Abstraction Layer

This section focuses on threats exploiting vulnerabilities in the Smartphone device, client-side application code, client data, or network communication protocols.

THREAT CATEGORIES

Different threat categories affecting the Smartphone / Tablet are structured according to the asset types listed below:

To structure the threat analysis, a list of assets that might be potential target of those threats have to be explained. It should be noted that a given vulnerability may be exploited by more than one type of threat. For the sake of simplicity, only the primary vulnerabilities are presented for each threat.

As mentioned in the introduction section, the ultimate risk rating and recommended mitigation actions will be analyzed based on, amongst other things, detailed functional analysis and architecture design.

The main components of the threat environment affecting the mobile Application are depicted below.

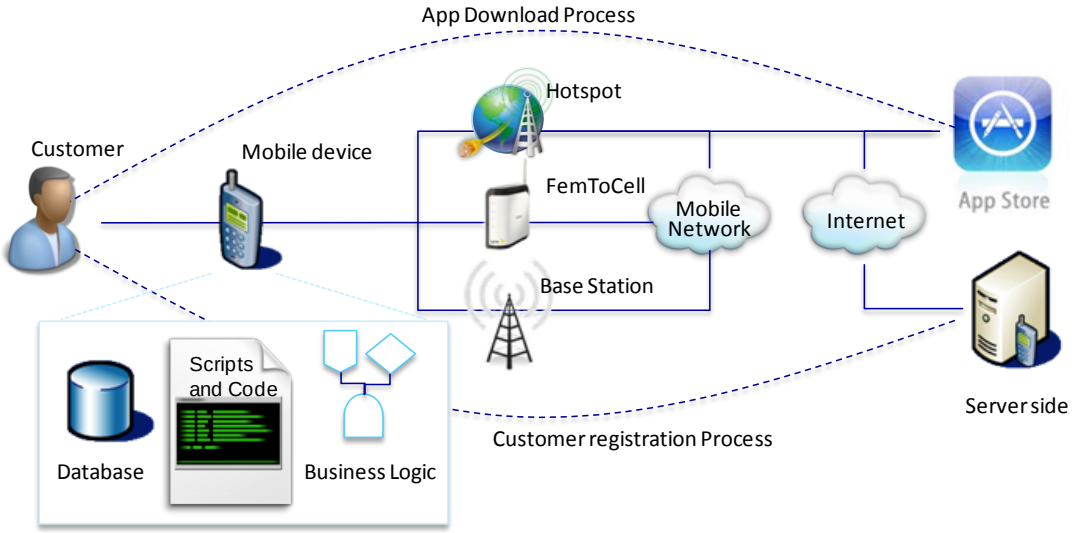


Figure 7 - Mobile threat environment

Asset	Description	Example Threats
Users, Customers	Customer identification	Social engineering, privilege abuse.
Device	Smartphone used	Loss, stealing of device
Code	Device side code	Reverse engineering, fingerprinting
Business Logic	Business logic driving security decisions for transactions	Abuse of authorization sequence flow required by the business logic / rules
Local data	Private data, log files, temporary files, static credentials	Disclosure of sensitive data to gain unauthorized access to user services
Application Store	Vendor-managed application stores: Apple Apps Store, Google Android Market Place, etc.	Fraudulent applications posing as legitimate and stealing credentials
Access Points	Network Access devices such as Hot spots, base station, picocell macrocell, microcell, femtocells	Rogue devices intercepting customer traffic to disclose sensitive data
Network	Mobile Network, Internet, local network	Eavesdropping communications to disclose customer private data, credentials, transaction details, or other sensitive data

SUMMARY OF THREATS AND VULNERABILITIES

The table below provides a synthesis of the most important threats and vulnerabilities.

Threat ID	Threat	Vulnerabilities	Description
1	Exposure of Sensitive Data / Credentials	• Lost or stolen Smartphone • Stealing users credentials • Cached passwords by mobile app • Transferring users credentials to a PC • Remote Control of the Smartphone	• Customers data / credentials could be recovered from Smartphone • Impersonate customer to access account and/or execute fraud transactions • Passwords can be easily recovered from cache files • Obtain user credentials stored on the Smartphone by transferring the contents to pc/laptop from the phone or memory card • A Smartphone can be remotely controlled through a Trojan, possibly part of a Botnet
2	Interception of sensitive data / credentials	• 3G man-in-the-middle (MITM); • Sniffing user data and credentials through a rogue hot-spot • Sniffing transactions	• A 3G receiver device can intercept communications in a MITM attack • A user may connect to a free hot-spot in public places that sniff traffic • Transactions can be intercepted in transit via rogue hot-spots
3	Impersonation of mobile application	• Phishing apps • Insufficient control by store provider • Varying level of controls depending on store (Android, Apple, Blackberry…)	• Customer could download rogue app from an app store that collects credentials, the same way as for traditional phishing sites • Apple App Store imposes more stringent controls that Google Android Market before releasing new app to public • Over 50 malicious apps were distributed from Google Android Market
4	Unauthorized Access	• Limited complexity of passwords • Auto-complete feature • Weak password • Bypassing authentication controls • Unauthorized access to customers account details • Hijacking left open session	• Organization enables enforced password complexity rules • Customer ID and/or passwords can be automatically filled-in by app • Weak passwords are likely as Smartphones have small keyboards • Authentication flow/logic could bypassed or abused • Customer account details may be cached or stored in device • Legitimate sessions can be overtaken remotely if left open too long or when session keys are not protected
5	Transaction repudiation	• Loss transaction traceability	• Audit trail are criticial in case a customer dnies a transaction. The auditing system should record relevant transaction data (amounts, date, beneficiary, etc.)
6	Unauthorized transactions	• Malware infection • On-the-fly transactions modification	• An increasing number of Trojans / malware target Smartphones and more specifically mobile banking applications (e.g. Zeus Trojan)

		• Modification of beneficiary • Modification of amount • Bulk transactions abuse	• A Trojan can modify transaction data before submission • Modification of beneficiary in fund transfers to criminals • Risk is even higher if organization accepts bulk transactions using a weak transaction authentication mechanism
7	Mobile application code hacking	• Insecure mobile application distribution • Client-side vulnerabilities • Inadequate server-side error handling	• Mobile application code can be modified and repackaged under legitimate organization name if the distribution mode is not secure enough • Client node could contain buffer overflow or other vulnerabilities • Bad or inadequate error handling could create break by exposing sensitive details in the application code
8	**Identity spoofing**	• Registration process vulnerabilities	• Hacker can trick Contact Center to register as legitimate customer and get credentials by using false identity
9	Privilege misuse	• By employees • By partners	• Employees can abuse privileges to get access or steal customer credentials to commit fraud • Partners employees involved in the registration or distribution process can similarly abuse privileges to steal credentials or sensitive information and later on use it to commit fraud

MOBILE SECURITY PRINCIPLES

Security principles for web applications also apply for Mobile.

The multi-layered approach principle referenced in web architectures is also relevent for native mobile applications. This means that all protocol flows for mobile must follow the same layering and filtering as Internet applications:

- Protocol flows used for mobile must be inspected by WAF mechanism using protocol specific filters and rules available on the WAF component;
- Access to services by mobile must always be orchestrated through the WAM;
- Logging must mention the channel to distinguish flows coming from the mobile;
- Static and dynamic code analysis must be done for the mobile, even for client-side code.

Therefore, mobile security architecture should ensure that the native mobile application is protected against multiple attack vectors as described in Chapter 1. The impact of these attacks on a business level could lead to reputational damage, financial loss or non-compliance with legal or regulatory requirements.

Principle #1: Mobile services can be shut down independently from other channels (more conventional applications for instance).	
Issue	The application must be under control and application attacks can target the mobile channel without any additional risks to another one. Threats can be specific to the mobile channel.
Concern	Shut down mechanism can be activated / deactivated for the mobile channel.

Principle #2: User's access to mobile can be blocked independently from the access to other channels.	
Issue	User's account can be only at risk for mobile channel. Threats can be specific to the mobile channel for one user.
Concern	Shut down mechanism about a user can be activated / deactivated for the mobile channel.

Principle #3: Mobile application must be securely distributed to customers. This can be done via trusted

application store. Application code must be signed.	
Issue	User should be educated to only download signed application from a trusted source.
Concern	Agreement between application store providers and financial institution.
Principle #4: Mobile application must follow client side secure engineering practices (OWASP, SANS, WASC, etc.).	
Issue	Application store only accepts application hosting based on list of criteria from a standard "Review guidelines". Client application must implement security best practices.
Concern	No data offline, no multitasking, application sandboxing, client side cache clean-up, …
Principle #5: Business decision and security decision must be taken at server side.	
Issue	Client side logic could be bypassed.
Concern	Authentication, authorization, user's session management, transaction signature validation, business macro flow must be performed at server side.
Principle #6: Transactions issued via the mobile channel should be identifiable as such. An *after the fact* analysis of transactions should be able to determine which of these were issued through the mobile channel.	
Issue	In case of customer dispute or in the course of investigating a security incident, it is crucial to determine which channel is affected.
Concern	Logging module must explicitly record the channel used to create transaction. Transactions issued through the mobile channel must be clearly identifiable in the transaction audit trail.
Principle #7: No customer sensitive data or credentials should be stored on the mobile client. Stored data must be protected.	
Issue	Credentials may be retrieved for unauthorized access. Sensitive customer data can be used for social engineering or may create privacy issues
Concern	Customer credentials should never be stored on the client. Customer data must be protected, either via sandboxing or encryption. Client-side data must be kept to the strict minimum required. No client related persistent data.
Principle #8: The customer should have a reliable means to ascertain that s/he actually using the legitimate mobile application.	
Issue	Customer should be protected against phishing attacks and rogue applications
Concern	Distributed applications should be assigned a certificate (or other equally reliable means) that can used by the customer to verify authenticity of the mobile application
Principle #9: The architecture should foresee a secure mechanism for deploying updates (major and minor) for the mobile application, whereby installed app is able to regularly check if new updates are available and propose	

to download them.	
Rationale	In case of discovered security vulnerability, security fixes must be "pushed" to mobile clients, as soon as possible.
Concern	A type of polling and secure delivery of security fixes must be supported in the architecture
Principle #10: A given authentication method should not be systematically enforced on both Internet and mobile channel. In other words, it should be possible to restrict the availability and use of a given authentication method to one channel and not the other	
Issue	In case security vulnerability or a security incident affects one authentication method, the exposure is confined only to one channel and does not have a cross channel impact.
Concern	The authentication methods are managed separately per channel and can be selectively activated / deactivated on a channel basis.
Principle #11: Customer authentication credentials should not be shared across channels. In case there is a business driver to still share credentials then this sharing should be done only for strong authentication methods	
Issue	In case customer credentials are stolen, the breach can affect simultaneously both Internet and Mobile channels. This is especially important for DAC2 3 & DAC 4.
Concern	Customer credentials must be managed (issued, reset, renewed, etc.) per channel. Passwords for mobile and web applications should be different.
Principle #12: A security weakness in the mobile application should not affect the security of Internet and mobile channels, even by a legitimate customer with valid credentials.	
Issue	It is almost impossible to exclude the presence of security vulnerabilities in the client code. The security of the front-end should assume a hostile mobile client.
Concern	Session management, business logic and business rules must be enforced on server side. Techniques like application sandboxing should not be relied upon to ensure security of the mobile.

AN ARCHITECTURE TYPE REFERENCE

Different security functionalities must be provided by the reference security architecture for native mobile application to comply with predefined principles. Security services and functionalities should address:

- ***Front layers defense***: same as web front layer defense;
- ***Security services***: additional services to be provided - device authentication interface, application authentication interface;
- ***Client side security****:* application signing, ESAPI client side.

[2] DAC Security level : Discretionary Access Control.

The reference security architecture intends to map security features on different components within the mobile architecture.

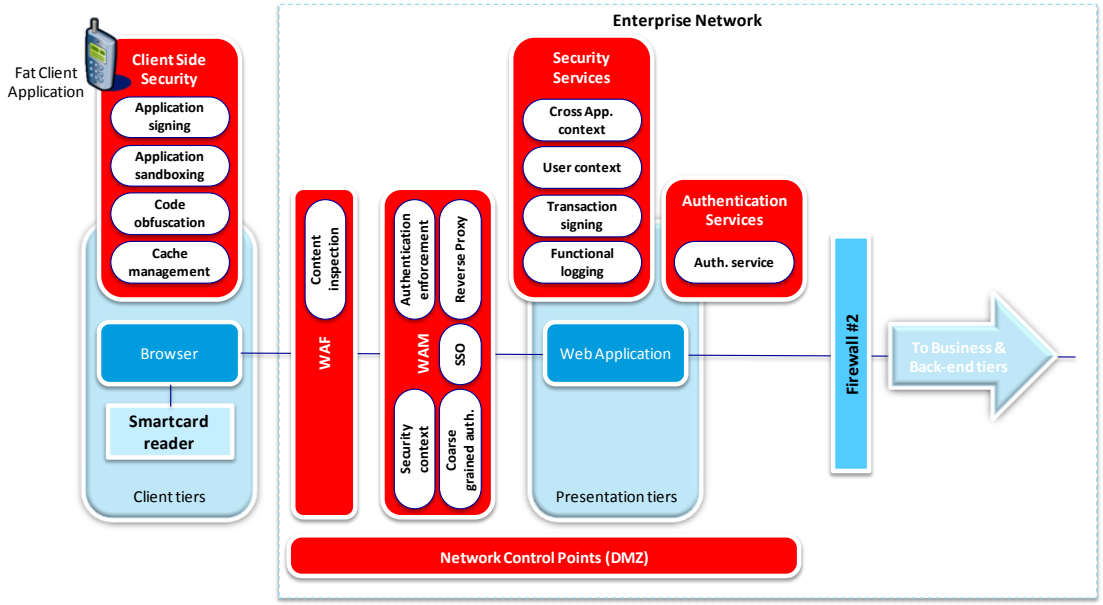


Figure 8 - Mobile security Architecture reference

KEY COMPONENTS AND FUNCTIONALITIES

A description about additional components and functionalities related to the native mobile application are listed below.

LOGON PRESENTATION LOGIC

Part of the native client mobile application which provides the user authentication screen(s). It asks for user credential and forward credential to the Web Access Manager which will starts the authentication flow.

CLIENT SIDE SECURITY

Responsibility is to ensure that native client mobile application is signed and provide verification screen to the user:

- Ensure application sandboxing mechanism on the user's device (could be based on device OS built in capabilities);
- Ensure data at rest mechanism is not used. Ensure full clean-up of cached data on application closure;
- Provide API for calling device authentication interface;
- Provide API for calling application authentication interface.

SECURITY MEASURES

Security measures aim for addressing the threats and vulnerabilities that can potentially affect key assets in the mobile eco-system such as device, application code, and data stored on device, application store, communication networks, etc. Each threat and vulnerability is described here after

in this chapter along with a global overview of security enforcement provided by main architecture principles that support business.

Baseline security measures

Application distribution

The main objective of application distribution measure is to protect from application modification through verification and trust.

Monitor application store regarding rogue applications

This is similar to "traditional" phishing attacks targeting on-line Internet sites, hackers could install malicious or phishing mobile applications in the application store providers. Should a customer fall victims of this threat then he/she might provide credentials or other sensitive information to the rogue applications, which then could be used by hackers to gain unauthorized access to customer accounts and initiate fraudulent transactions. For this reason, the business should extend anti-phishing service provider service to also cover the monitoring of Application Stores foreseen in the scope of the distribution process. The business is then alerted whenever a malicious app is installed in the store and can take measures to remove it in coordination with the store provider.

Establish contractual clauses with Application Store providers

It is important to include contract Terms and Conditions in the contract with the Application Store provider to clearly stipulate provider responsibilities and obligations with regards to (non-exhaustive list):

- Service Level Agreements concerning the response time for deploying mobile application versions, bug fixes, and more importantly security hot fixes;
- Continuity of Service: to ensure the provider does not unilaterally discontinue the service by discarding the mobile application from the application store;
- Financial terms to ensure that the provider is not entitled to financial payment that directly depend on the number of customer downloads or volume of transactions or other "scalable" aspects;
- Any other terms and conditions that mitigates the risks that arise from the strong dependency on the application store provider service provision and performance.

Accelerated and smooth security patch deployment

One cannot completely rule out the discovery of security vulnerabilities in the mobile client after it has been deployed on a larger-scale to the customer community. For this reason, an accelerated and smooth security patch deployment process needs to be defined, designed, and deployed. Tests of the process, in coordination with the application store provider, need to be performed to ensure readiness to close security vulnerabilities in the shortest possible timeframe.

Application installation and start-up

The user authenticates the application

First, the user has to be educated to only use signed application downloaded from a trusted application store. When the user launches the application, s/he will receive a prompt indicating that the application is distributed by the trusted financial institution and can authenticate the application.

An additional measure can be implemented to help the user to authenticate the application.

An adaptive authentication identifier (for example, a picture and/or a passphrase – a captcha) is chosen by the user during the user registration process. The client application displays this authentication identifier to prove the financial institution's identity.

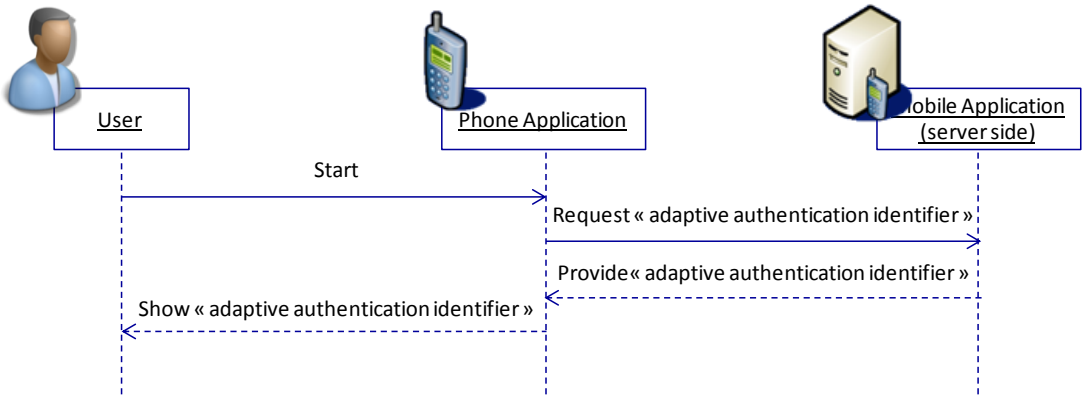


Figure 9 - Mobile authentication process

SHUTDOWN MOBILE CHANNELS INDEPENDENTLY FROM OTHER CHANNELS

In case the mobile channel is under threat (either potential threat or an ongoing attack), it is critical to have the capability to shutdown the mobile channel *without* affecting the full operations of the other channels.

CLIENT SOURCE CODE IS OBFUSCATED

Depending on the type of programming language and the development framework used, an attacker has the opportunity to make a thorough analysis of the code base and exploit possible weaknesses. Decompilation and disassemblies can also be used to determine the framework or libraries used. Based on publicly available vulnerabilities in these frameworks and libraries, an attacker may be able to exploit them. At a minimum, developer comments should be automatically removed from source code before distribution. Variable should not use meaningful names in order to hide business logic.

APPLICATION EXECUTION

NO STORAGE OF SENSITIVE DATA ON DEVICE

Customer private data stored on the Smartphone might be compromised. Temporary files store locally on the Smartphone by the mobile application may reveal sensitive data about previous sessions or even customer credentials if not protected properly. In addition, the customer may have recorded his/her credentials in local files (e.g., in text files, e-mail sent box, or SMS Outbox). Mobile client software should be designed such that no security-sensitive data is stored persistently on the device (log files, security cookies, temporary files, auto-complete authentication, transaction history, etc.). As soon as a session is closed or the application exits, these types of data should be deleted beyond recovery by a determined hacker.

URL TO ACCESS SERVER-SIDE SERVICES

The mobile client usually uses a built-in URL to locate and use server-side resources. It is then obvious to protect these URL from modification by hacker as this can result into requests been redirected to a hostile server that steals sensitive data and/or credentials without the user knowing.

PERFORM SECURITY CODE REVIEW WITH A STRONG FOCUS ON CLIENT SIDE APPLICATION PART

It is a good practice to perform code review, both on client and on server-side to detect and remove vulnerabilities before live operations. Mobile client code review must be done with even more scrutiny because the code is readily available to hackers for full analysis and scanning to exploit vulnerabilities. For this reason, special attention should be devoted to correctly scope and perform the code review, both via static and dynamic code analysis. Input validation and error handling are the most relevant in this case.

MOBILE APPLICATION SHOULD NOT INTERPRET CODE (JAVASCRIPT, JAVA, OBJECTIVE C)

The mobile application should not accept code as input to ensure that the mobile client does not interpret and execute code that might be injected as a valid user input. Otherwise malicious code can be run and steal or modify memory areas that manipulate sensitive data (transaction data or credentials).

STRONG USER SESSION MANAGEMENT

This could be caused by exploiting weaknesses in the session management scheme, through session fixation, predictable session ID, stealing of session cookies, or in case of a weak implementation of the session management in the backend. For this reason the following items should be adhered to:

- State transition management must enforced at the server side;
- Robust session management design and implementation;
- Security cookies must never be sent to the client;
- Use an as-short as possible session timeouts;
- Session ID should not be predictable.

THE DISTRIBUTED APPLICATION IS SIGNED (CODE SIGNING)

The application code signing must be integrated within each build of the distributed application.

Apple, for example, acts as a certificate authority and signs applications distributed on the app-store.

When the user launches the application, he will receive a prompt indicating that the application is distributed by the trusted financial institution.

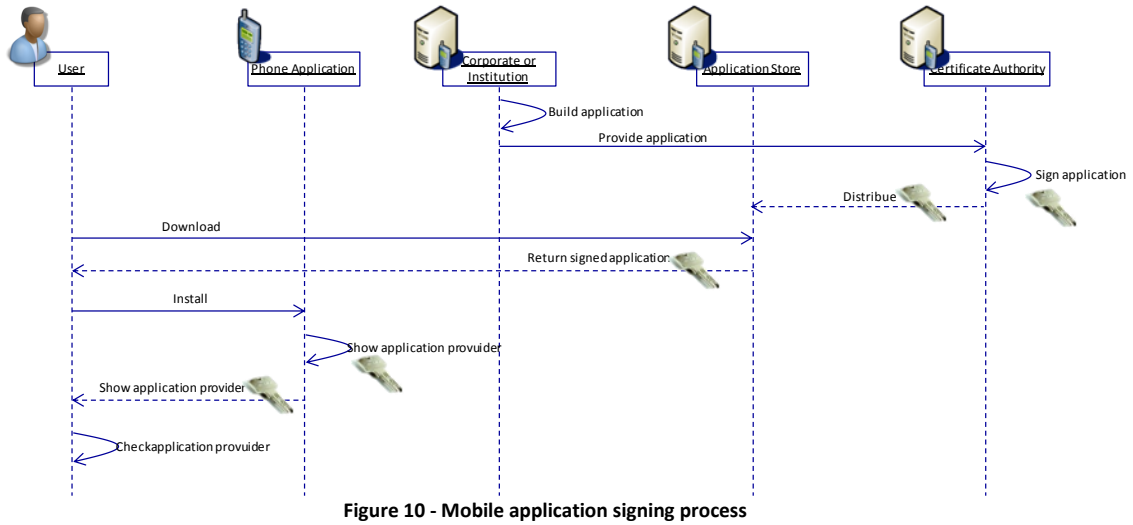


Figure 10 - Mobile application signing process

APPLICATION SANDBOXING PRINCIPLE IS USED

Applications are isolated from each others.

At runtime or at rest, all data handled by the application (user input, data received from the server, client-side session data) are only accessible by the application itself. Other application can never access these data.

This behavior could be based on the device OS capabilities (for example Mac OSX provides such built-in mechanism).

THE APPLICATION DOES NOT KEEP DATA AT REST

All data handled by the application are never cached into the file system, local database, temporary file or any other mechanism to keep data at rest.

An application session cache can be used to keep data on client-side for performance reason. However, these data are completely deleted when the application is closed.

The user Id could be stored on the user device (for usability reason). It must follow the sandboxing principle described in Section 0.

The main objective is to protect from unauthorized device access to the mobile service.

SECURITY AND BUSINESS DECISIONS/VALIDATIONS ARE PERFORMED AT SERVER-SIDE

All security decisions (like user authentication, credentials validation, user authorization, signature requested for a money transfer, check amount limit is reached …) are performed at server side. The client application requests server-side services and does not make any security validation.

Business decisions and validations are also performed at server-side and are not implemented in the client application. Business logic is not exposed on the client application.

USE HTTPS BETWEEN MOBILE CLIENT AND SERVER-SIDE

All communications between mobile client and server-side must be based on HTTPS protocol.

ADDITIONAL SECURITY MEASURES

APPLICATION DISTRIBUTION

No additional measures apply here compared to baseline security measures.

APPLICATION INSTALLATION AND START-UP

CUSTOMER AWARENESS

Raise customer awareness about the risks of stolen devices and exposed sensitive data, especially credentials. Encourage customers to always keep an eye on their device and enable password protection for the device. Raise customer awareness about selecting strong passwords and PIN codes. Offer guidelines to customers on how to distinguish a legitimate mobile application from a rogue/phishing application, based on the appearance (logo, menus, URLs, etc.) and expected behavior (access permission request for privileged device resources, strange requests for customer private data in the same way as traditional phishing sites). Customers should be educated to download applications only from trusted sources. Several anti-virus tools, and anti-spyware as well as other mobile security tools are now maturing so customers should be advised to use them.

USER OUT-OF-BAND REGISTRATION

The out-of-band registration process is done through an authenticated channel which can provide strong authentication.

During the registration process, the user provides his phone number used for mobile application. This phone number can be used by the financial institution to send information to the user (like one time password, transaction details …).

Depending on the authenticated method used to log in, the user initiates his credentials (password for example).

A bootstrap URL is sent via SMS to the user. It will be used by the user to download the trusted application.

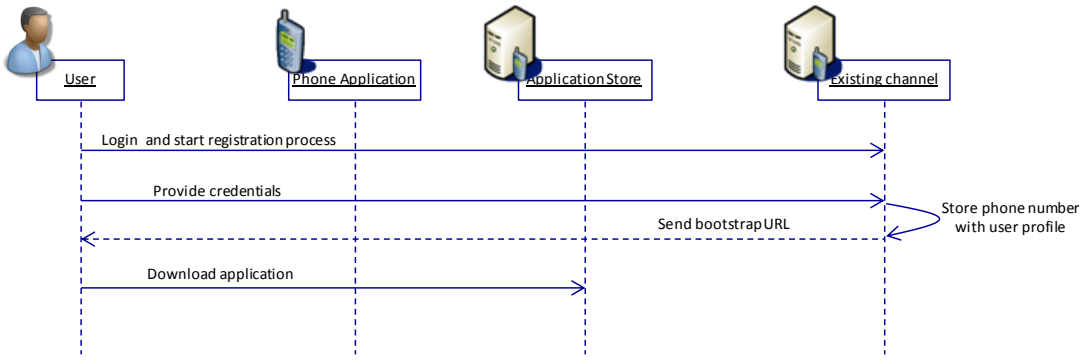


Figure 11 - Mobile out-of-brand registration process

BLOCK USER ACCESS TO MOBILE INDEPENDENTLY OF OTHER CHANNELS

In case a mobile customer account is compromised (as for stolen device), the service should have the ability to (temporarily) block a customer mobile account while maintaining other channels account fully active.

CUSTOMER AUTHENTICATION AT DAC 3 OR DAC 4 DEPENDING ON REGULATIONS AND RISKS

The selection of the right authentication mean must follow a structured process that takes into account the type and sensitivity of the service being accessed (public, account consultation, or transactional service), the risks and business impacts involved in case of fraud and the regulator requirements for on-line services. The ultimate decision must be risk-based with very active business involvement for risk acceptance and mitigation.

Below guidelines should be used to select the right DAC level according to the application risk profile presented in Section **Erreur ! Source du renvoi introuvable.**:

- **Informational content**: should require DAC 3 as a minimum depending on regulatory requirements. In case DAC 3 is used, device and user registration should be used to mitigate the risk;
- **Transactional limited**: DAC 4 or DAC 5 should be used. To limit the risk, DAC 4 could be considered adequate if it is complemented by setting a limit on transaction amounts and also by restricting transfers only to pre-registered recipients;
- **Transaction Extended**: DAC 5 (2-factor authentication) should be required in case the limit on amounts and recipients cannot be enforced.

ABILITY TO INCREASE DAC LEVEL REQUIRED IN CASE OF ATTACK

In some cases it might be necessary to increase the security level in the context of high alert of new security frauds or newly discovered vulnerabilities. It is then very useful for the business to have the flexibility to raise the minimum DAC level required for accessing an Internet Application or for performing critical type of operations. Raising the DAC required should be as flexible as changing a configuration file or executing tailor-made pre-programmed routines. It should not require any redesign work or development.

As for examples:

- Force user authentication via SMS (DAC 4) instead of SMID/SMP (DAC 3);
- Force user to sign all transactions via UCR (unconnected card reader), independently of the amount and recipient;
- Force signatures on all international, non-SEPA transfers.

OPERATION

STRONG USER SESSION MANAGEMENT

User session creation is initiated and maintained at server side just after the user is correctly authenticated.

A session token is sent back and kept on the client side application in using best practices recommendations defined in this document. The token is then attached on each service requests. It will be validated and used during server side process to perform user transactions.

Session timeout is implemented at server side.

Inactivity timeout is implemented at client side and triggers a clean server side session closure process. Each time the client application is closed, the same server side session closure process is called.

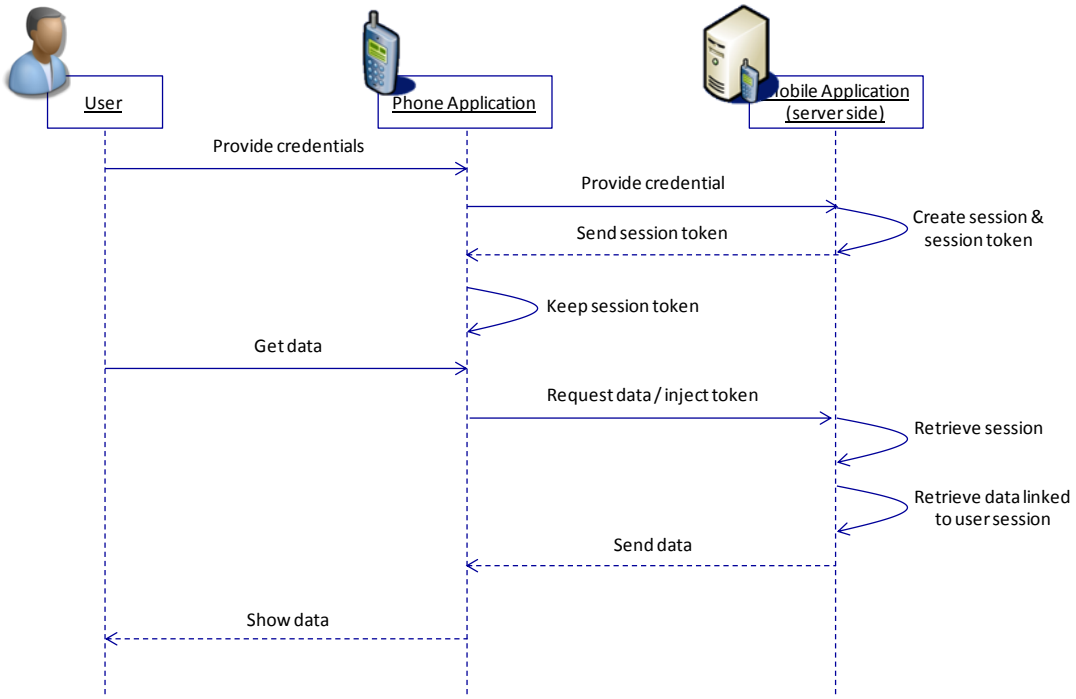


Figure 12 - Mobile session management flow

AMOUNT LIMIT ON TRANSACTIONS PER CHANNEL

As the risk profile for mobile channel differs from conventional web channel, it is very useful for the business to have the flexibility to set amount limits on fund transfers depending on the used channel. For example, amount limit (per transaction, per day, week, etc.) could be set to a conservative amount on the mobile channel and to a higher amount for the web channel. As the risk profile for the 2 channels evolves differently over time (depending on the emergence of new threats on mobile) it would be appropriate to adapt the limit in order to lower the financial loss in case of fraud until the organization has more in depth risk analysis and assessment of the threats and vulnerabilities.

AUDIT TRAILS

Disabled or otherwise non-effective transaction audit trails could result in unauthorized transactions going undetected or in valid customer transactions been repudiated. Detailed and well-designed audit trails must be produced, taking into account all transaction details, including customer identity, transaction attributes and transaction timing for future reference as part of dispute / claim resolution and / or incident investigation following an internal or external fraud.

For this reason:

- All transactions, failed or successful, should be logged with a timestamp;
- All data required for proving transaction authenticity must also be kept as well to be able to resolve the dispute that may arise several months after the transaction.
- Regulatory requirements for data retention must be adhered too;
- The transaction audit trail must indicate which channel the transaction originated from.

TRANSACTION SIGNING VIA 2-FACTOR UNCONNECTED DEVICE

Legitimate transactions created by a customer could be intercepted on the fly, modified and injected in the session. Modifications could involve changing the beneficiary account number, transfer amount and / or currency. This threat is more likely to be effective is case transaction signing support is not provided as part of the transaction integrity validation. Transaction signing must be based 2-factor mechanism, preferably using unconnected device (UCR):

As a consequence:

- WYSIWYS (**w**hat **y**ou **s**ee **i**s **w**hat **y**ou **s**ign) principle should be applied;
- Transaction signing must be applied on all components of the transaction;
- Transaction integrity checking must be implemented at the server-side.

SECURITY MEASURES DECISION MATRIX

The decision matrix helps in taking decision over security measures that must be implemented depending on application risk profile. It can be read as follow:

- ***From left to right***: security measures to apply during each lifecycle phase of the mobile application;
- ***From top to bottom****:* Accumulated security measures from the baseline security measure (for any risk application profile) till the higher risk application profile.

		Application distribution	Application installation & start-up	Application execution
Baseline security measures	Any application risk profile	**Business measures** • Monitor app store regarding Rogue applications • Establish contractual clauses with app store provider **IT measures** • Application is distributed through app store • Application is signed (binary code) • Accelerate and smooth security patch deployment	**Business measures** • Shutdown mobile service independently from other channel **IT measures** • Customers able to authenticate on application (branding, trusted certificates) • Client source code is obfuscated	**IT measures** • No sensitive data stored on the device • Perform all security business decision only at server side • Apply application sandboxing in using OS capabilities • Full data clean-up on application closure • URLs to access server side services cannot be changed • Plan security code review with a strong focus on client side part • No code consumption on device (JavaScript, java, objective C) • Use HTTP request between client and server side
Additional security measures	Public content	No additional security measures	No additional security measures	**No additional security measures**
	Private content	No additional security measures	**Business measures** • Customer security awareness; • Customer registration through secure channel; • Block user's access to mobile service independently from the access to other channels; • User authentication DAC3 or DAC4 depending on regulatory requirements and unless risk is accepted for service.	**IT measures** • Strong user session management.

		Application distribution	Application installation & start-up	Application execution
	Transaction Limited	No additional security measures	**Business measures** • Ability to increase DAC level requirement in case of attack..	**Business measures** • Amount limit on transactions per channel; • Pre-registered recipients; • Audit trail.
	Transaction Extended	**No additional security measures**	**Business measures** • Two factors unconnected user authentication unless risk is accepted for the service.	**Business measures** • Transaction signing through two unconnected device factors.

SECURING THE INFRASTRUCTURE

INTRODUCTION

A secure Web application relies upon a secure network and host infrastructure. The network infrastructure consists of routers, firewalls, and switches. The role of the secure network is not only to protect itself from TCP/IP-based attacks, but also to implement countermeasures such as secure administrative interfaces and strong passwords. The secure network is also responsible for ensuring the integrity of the traffic that it is forwarding. If you know at the network layer about ports, protocols, or communication that may be harmful, counter those potential threats at that layer.

This chapter aims to present the architectural options of using independent dedicated Web Access Management (WAM) technology as opposed to the usage of frameworks/plugins within web applications. The advantages and disadvantages of each scenario will be presented to facilitate a decision for the most appropriate solution.

The described architecture should be flexible enough to be applicable to any web project.

BACKGROUND

The term Web Access Management (WAM) applies generically to technologies that use access control engines to provide authentication and authorization capabilities for Web applications.

Traditionally the purpose of WAM is to authenticate and provide access control to web based resources. WAM products may also include IAM, role/rule management and audit capabilities. As products have evolved WAM solutions have moved towards a more general policy enforcement solution including Single Sign On (SSO), and support for more general web services.

There are two different types of architectures when it comes to implementing WAM:

Coupled - Frameworks/Plug-ins are programs that are installed on every Web/application server, register with those servers, and are called at every request for a Web page. They intercept the Web request in order to make a policy decision and communicate with an external policy server in order to make these decisions.

One of the benefits of a plug-in (or agent) based architecture is that they can be highly customized for unique needs of a particular Web server.

One of the drawbacks is that a different plug-in is required for every Web server on every platform (and potentially for every version of every server). Further, as technology evolves, upgrades to agents must be distributed and compatible with evolving host software.

Decoupled - Proxy-based architectures differ in that all Web requests are routed through the Proxy server to the back-end Web/application servers.

One of the benefits of a proxy-based architecture is a more universal integration with Web servers since the common standard HTTP protocol is used instead of vendor-specific Application programming interfaces (APIs). The drawback is that additional hardware is usually required to run the proxy servers.

AN ARCHITECTURE TYPE REFERENCE

An architecture based on an internet facing firewall, Web Application Firewall (WAF), inner firewall, reverse proxy (RP), inner firewall before the presentation layer offers multiple levels of protection. (Defense in depth).

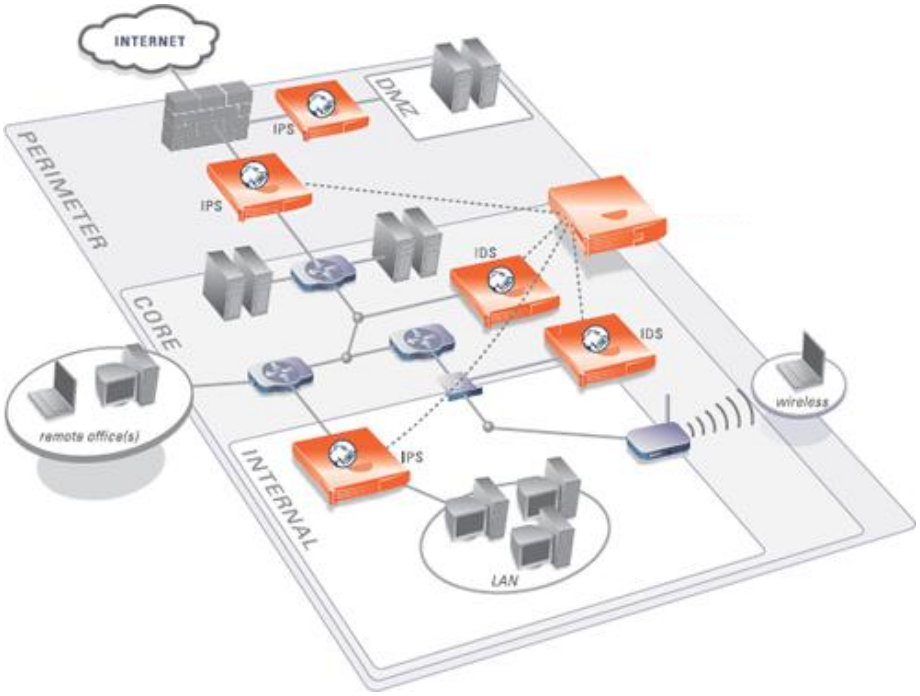


Figure 13 - The infrastructure architecture reference

The firewalls protect against network level attacks (and simpler higher level attacks using deep packet inspection). Using different vendors for the outer and inner firewalls protects against vulnerabilities in one or other set of firewall software. The outer firewall will permit the minimum number of protocols to pass through from the internet.

The WAF and the reverse proxy protect against application level attacks – the WAF because it is able to analyze application traffic in detail and the reverse proxy because it only sends HTTP RFC compliant request to the backend.

All traffic which needs to be authenticated and authorized before to enter the internal network zone must go through the reverse proxy which is indeed an important system and network protection level. RP is the first component that is accessible by external traffic prior to authentication. If not used, the web server is the front line for authentication. As it has more services to run, this will weakness the system and may lead to potential vulnerabilities.

This topology conforms to standard architectural security principle that requires one major function protection component per system or server.

SECURITY INFRASTRUCTURE REQUIREMENTS

Requirement	Description
Segregation	Authentication should be implemented separate from the business

	application. Support multiple business entities
Defense in depth	Validation of data and authority should take place within the DMZ, Only validated and authorized traffic should be permitted into the 'trusted zone'
Provide Centralized Authentication	Leverages existing authentication mechanisms to make it easier to integrate with existing environments
Singe Sign On capability	Ensures that existing sign-on mechanisms can be leveraged across Web applications, federated partners, and Web Services
Real Time Management	Provides a centralized view of who is accessing resources
Provide centralized policy that defines access control policies	Ensures only authorized users access protected resources
Open Standards based	Uses standards such as SAML and ID-FF to create and share security for federation and easy upgrade and redeployment
Access Auditing and Event Logging	Provides a trail of access for auditing violations
Scalability	Solutions should be scalable to meet future business evolution
Cost Effective	Solutions should be cost effective across all lifecycle phases
Compliant	Ease of meeting compliance requirements (e.g. PCI)

NETWORK THREATS COUNTERMEASURES

ROUTER

A router is a device that forwards data packets between telecommunications networks, creating an overlay internetwork. A router is connected to two or more data lines from different networks. When data comes in on one of the lines, the router reads the address information in the packet to determine its ultimate destination. Then, using information in its routing table or routing policy, it directs the packet to the next network on its journey or drops the packet. A data packet is typically forwarded from one router to another through networks that constitute the internetwork until it gets to its destination node (Cf. Wikipedia).

Routers are your outermost network ring. They channel packets to ports and protocols that applications needs. Common TCP/IP vulnerabilities are blocked at this ring.

External networks must be carefully considered as part of the overall security strategy. Separate from the router may be a firewall or VPN handling device, or the router may include these and other security functions.

FIREWALL

A firewall is a device or set of devices designed to permit or deny network transmissions based upon a set of rules and is frequently used to protect networks from unauthorized access while permitting legitimate communications to pass (Cf. Wikipedia).

The firewall blocks those protocols and ports that the application does not use. Additionally, firewalls enforce secure network traffic by providing application-specific filtering to block malicious communications.

Switch

A network switch or switching hub is a computer networking device that connects network segments. The term commonly refers to a multi-port network bridge that processes and routes data at the data link layer (layer 2) of the OSI model. Switches that additionally process data at the network layer (Layer 3) and above are often referred to as Layer 3 switches or multilayer switches (Cf. Wikipedia).

Switches are used to separate network segments. They are frequently overlooked or over-trusted.

Reverse Proxy

A Reverse Proxy is a type of server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client as though it originated from the reverse proxy itself. It is usually situated close to the servers and will only return a configured set of resources.

Reducing the load on the firewall

A reverse proxy can offload the Web servers by caching static content (such as images, css or JavaScript files…) as well as dynamic content (such as a HTML generated pages). Proxy caches of this sort can often satisfy a considerable amount of website requests, greatly reducing the load on the originated servers and improving their effective performance. It can also optimize content by compressing it in order to speed up loading times.

A Reverse Proxy can also distribute the load from incoming requests to several servers, with each server serving its own application area. In the case of reverse proxying in the neighborhood of web servers, the reverse proxy may have to rewrite the URL in each incoming request in order to match the relevant internal location of the requested resource.

Caching on the DMZ reverse proxy reduces the number of requests on the DMZ firewall since cached content does not pass through the firewall. This is desirable since the DMZ firewall will have a more complex traffic model than the first line Internet facing firewall it needs to allow for more protocols and rules (for management and monitoring of systems for example) – they will in turn limit the throughput performance.

The proxy server can optimize and compress content to speed up the load time and reduce bandwidth consumption.

Enabling security

In term of security it can hide the existence and characteristics of originated servers and protect against common web-based attacks. Although, SSL encryption is sometimes not performed by the web server itself, but is instead offloaded to a reverse proxy that may be equipped with SSL acceleration hardware.

Web application firewall

A Web Application Firewall is a form of firewall which controls input, output, and/or access from, to, or by a web application or service. It operates by monitoring and potentially blocking the input, output, or system service calls which do not meet the configured policy of the firewall. The application firewall is typically built to monitor one or more specific applications or services (such as a

web or database service), unlike a stateful network firewall which can provide some access controls for nearly any kind of network traffic.

HTTP TRAFFIC INSPECTION

Furthermore, the reverse proxy can rewrite HTTP requests to conform to the HTTP standard, guaranteeing that web servers only ever see compliant traffic. This also helps to safeguard the infrastructure against invalid or malformed HTTP traffic.

In addition to detecting and blocking such traffic, some types of WAF can also rewrite traffic in flight as required for security or application purposes.

A WAF (Web Application Firewall) is able to inspect HTTP traffic in real time for attacks, vulnerabilities and errors. It operates at the application level rather than the network level. It can detect and alert on a wide range of known attacks and compromise and generally goes beyond the deep packet inspection capabilities of normal firewalls.

A WAF implements an improved level security through black listing and/or white listing. It protects against most common "out of the box" web vulnerabilities. It is highly configurable and can be adapted on the fly to cope with new attacks and vulnerabilities.

A WAF can address situation where the web server is provided by 3rd party (as appliance for example) and where it requires additional security that might be difficult or impossible to configure.

DEVELOPMENT CYCLE

The deployment of a WAF can improve development and break/fix cycles. Since attacks and vulnerabilities can be quickly blocked at the WAF, developers have time to test and evaluate their changes properly (because it is not necessary to patch application code). There is therefore less risk to the availability of the application, since changes are tested and made in a properly controlled and scheduled manner.

PCI COMPLIANCE

The current Payment Card Industry Data Security Standard (PCI DSS version 1.2) requirement can be delivered to any public web application by installing a WAF.

The payment brands and acquirers are responsible for enforcing such compliance by protecting and assuming review code (regularly and on every changes – verifying that all vulnerabilities are removed).

For large applications, the resource and cost required to perform adequate code reviews on a regular basis could easily exceed the cost of a WAF.

WEB ACCESS MANAGEMENT

AUTHORIZATION AND AUTHENTICATION

An effective WAM (Web Application Management) solution offloads the authentication and authorization functions from the web server or application itself. It becomes an external service that the application or web server calls when required. This concentrates the functionality and development of these functions at a single point of deployment and use.

Use of web application management systems removes the authentication and some authorization functions from the domain of the application or web server and moves them to a central logical

function for management and support. This contributes to a reduction in development time and costs because authorization and authentication are now delivered as a standard reusable service with standard interfaces which any new project or application can use.

Interface to enterprise directories

The WAM has the ability to interface directly to the enterprise authentication and authorization sources (such as LDAP databases), instead of holding and managing credentials locally. It can of course also cache credentials as required. It can also interface with the logging, reporting and alerting mechanisms of these systems. One can then upgrade authorization mechanisms or change authorization methods as required at this point of use, without major change or disruption to the application portfolio.

Flexibility

Reduction in development time and costs because authorization and authentication are now delivered as a standard service with standard interfaces such a way that any project or application can plug into.

The WAM can be closely integrated with the group's principal authentication and/or authorization systems, to provide new functionality as required. It can also interface with the logging, reporting and alerting mechanisms of these systems. Different authentication mechanisms can be introduced for customer groups with different requirements.

WAM Architectural Options

The Web Access Management must be a scalable and reusable solution component.

Coupled environment

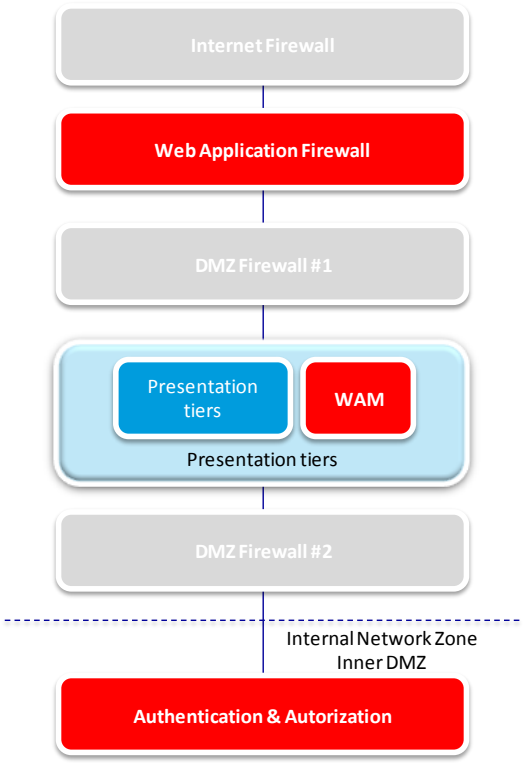


Figure 14 - Plug-in based WAM deployment topology

The principles covered by a coupled environment are provided below.

Requirement	Description
Segregation	By combing the presentation and authentication layer, the goal of segregation are not met, risking the identification functionality
Defense in depth	Filtering and scanning by the WAF
Provide Centralized Authentication	Authentication is distributed potentially across many servers
Singe Sign On capability	Integration of multiple technologies from multiple vendors
Real Time Management	Difficult to obtain a real-time status due to the decentralization of authentication
Provide centralized policy that defines access control policies	Access control distributed across multiple servers, no centralization. Nevertheless all plug-ins could connect to a central policy server
Open Standards based	Will require bespoke plug-in or frameworks for the specific technology and version deployed on the web server or package provider
Access Auditing and Event Logging	The coupled WAM functionality installed within distinct web and application servers distributes the logging which then has to be aggregated for management
Scalability	Limits are imposed by the sharing of physical infrastructure between the web/application servers and the WAM functionality; neither can be independently upgraded
Cost Effective	Delivering the same level of functionality on a per application / server basis increases costs for projects and operational costs for applications overall, however there is no need for additional servers
Compliant	A key PCI compliance requirement can be delivered

DECOUPLED ENVIRONMENT

The use of a reverse proxy and WAF/WAM architecture enables the decoupling of the application environment from the client entry points. This means that back-end web servers can be changed, replaced or added to, transparently as far as the external user is concerned. Only the proxy configuration needs to be changed to redirect traffic as desired. It gives business flexibility to integrate applications or test packages independently of the software used. This is then independent of the technology used at the back end or new packages introduced – whether based on Java, PHP, .NET, etc.

By designing a proxy enabled solution, it is possible to move web servers (for authenticated traffic, public server's remains in a dedicated 2nd line DMZ) back into an internal network zone (or an additional DMZ) and put the heavyweight security around the proxies and their DMZ as required. This also gives flexibility around security depending on the criticality of the system – whereas if the web servers reside in a shared DMZ, they need to be secured to the same level as any other resource in that DMZ.

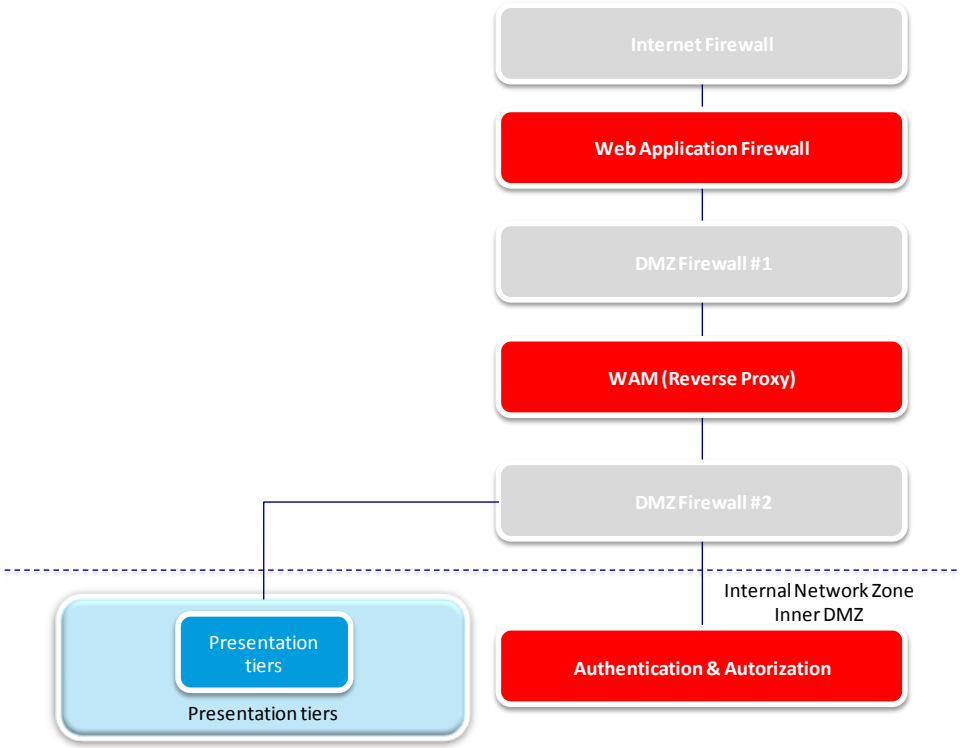


Figure 15 - Proxy based WAM deployment topology

Patching and management is simpler for systems in internal or trusted zones since out of band management connectivity should be in place and there is no need for management traffic to transit a firewall (as there would be if these systems were in a front line DMZ). This also reduces the complexity of the rule base for the DMZ firewall.

The principles covered by a decoupled environment are provided below.

Requirement	Description
Segregation	Supports the separation of security services from business functionality
Defense in depth	Filtering and scanning by the WAF + dedicated security system hardening(reverse proxy)
Provide Centralized Authentication	Authentication check is centralized on the reverse proxy. When secure websites are created, encryption is typically not performed by the web server itself, but by a reverse proxy that is equipped with dedicated acceleration hardware or software, and can offload this task from the web server itself. The reverse proxy can handle traffic encryption in either or both directions (client <–> proxy and proxy <-> server). Performance of the web server is optimized since the tasks of encryption are offloaded to the reverse proxy. This reduces traffic and CPU loading on the web server itself. Encryption key and certificate management (for client side encryption) is also localized around the proxy rather than being distributed across all the web-servers Where authentication will be provided by the certificate (DN), this is available

	for the Proxy based WAM, but not the plug-in based
Singe Sign On capability	Integration of single technology reduces overall costs
Real Time Management	Centralization should allow a real-time status of users which are active and authenticated
Provide centralized policy that defines access control policies	Yes build in most SSO Reverse Proxy product
Open Standards based	Depends on the implemented credential propagation technology
Access Auditing and Event Logging	The reverse proxy is a logical central point to collect logs and statistics for user traffic analysis
Scalability	The reverse proxy can distribute the load around multiple physical or virtual servers. Additional back end resources can be added as required, transparently as far as the end user and internet are concerned since this involve only change the proxy configurations internally. Additional proxies can also be added if required, assuming load-balancing capabilities have been built into the solution architecture. Thus for systems expecting growing user populations or growing traffic levels, new resources can be brought on-line without disruption to the existing service or users, if necessary and at short notice. Reverse proxies also help deliver higher performance of the system by ensuing no server is over or under utilized
Cost Effective	Delivering the same level of functionality on application / server basis would increase costs for projects and operational costs for applications. The management effort is now concentrated on a small number of dedicated systems, rather than spread across multiple presentation layer systems but one more system in the loop and must be managed (reverse proxy)
Compliant	A key PCI compliance requirement can be delivered

POTENTIAL ISSUES

Network latency - Any system inserted into the network will introduce some degree of latency. However, note that the effect of caching, offload of encryption and authentication, etc will offset this; therefore the additional latency should balance this disadvantage.

Additional points of failure - If any of the individual components (firewall, WAF, reverse proxy, WAM) fails, this will result in service interruption. This can be addressed by standard high availability and disaster recovery mechanisms. This is already done for internet facing routers and firewalls and other critical infrastructure.

HOSTS THREATS COUNTERMEASURES

To secure a host, whether it is a Web server, an application server, or a database server, breaking down the various secure configuration settings into separate categories help in focusing on specific category to review or apply security settings that relate to that specific category.

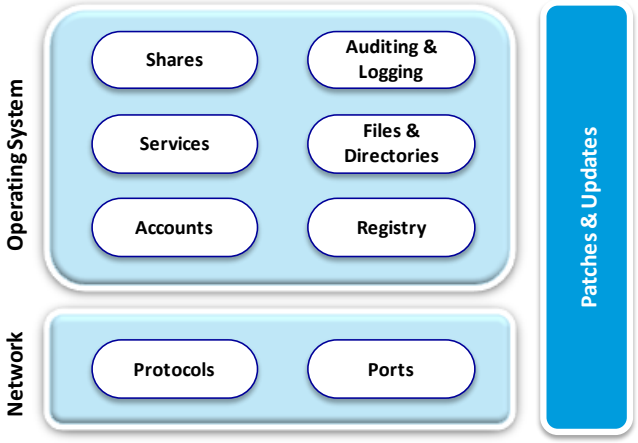


Figure 16 - Host threats

With the provided segmentation for these categories, server's configuration can systematically evaluated or secured instead of applying security settings on an ad-hoc basis. The rationale for these particular categories is shown here below.

Category	Description
Patches and Updates	Many top security risks exist because of vulnerabilities that are widely published and well known. When new vulnerabilities are discovered, exploit code is frequently posted on Internet bulletin boards within hours of the first successful attack. Patching and updating your server's software is the first step toward securing the server. If you do not patch and update your server, you are providing more potential opportunities for attackers and malicious code
Services	The service set is determined by the server role and the applications it hosts. By disabling unnecessary and unused services, you quickly and easily reduce the attack surface area.
Protocols	To reduce the attack surface area and the avenues open to attackers, disable any unnecessary or unused network protocols.
Accounts	The number of accounts accessible from a server should be restricted to the necessary set of service and user accounts. Additionally, you should enforce appropriate account policies, such as mandating strong passwords.
Files and Directories	Files and directories should be secured with restricted file system permissions that allow access only to the necessary service and user accounts.
Shares	All unnecessary file shares, including the default administration shares if they are not required, should be removed. Secure the remaining shares with restricted file system permissions.
Ports	Services running on a server listen on specific ports to serve incoming requests. Open ports on a server must be known and audited regularly to make sure that an insecure service is not listening and available for communication. In the worst-case scenario, a listening port is detected that was not opened by an administrator.
Auditing and logging	Auditing is a vital aid in identifying intruders or attacks in progress. Logging

	proves particularly useful as forensic information when determining how an intrusion or attack was performed.
Registry	Many securities related settings are maintained in the registry. Secure the registry itself by applying restricted ACLs and blocking remote registry administration.

SECURING THE APPLICATIONS

Before you can design secure Web applications, you have to understand the security risks involved and know how to deal with them. In this chapter, we will discuss the most common risks involved with Web-based applications.

While the perimeters of organizations are more and more secured, hackers are now attacking the application layer and the clients directly, instead of the infrastructure. It is of survival importance to respond with applications that are as attack proof as possible.

IDENTIFYING THE SOURCES OF RISK

The sources of most security problems are user input, unprotected security information, and unauthorized access to applications. Among these risk factors, user input stands out the most, and it is also the most exploited to make unauthorized, unintended use of applications. A poorly written application that handles user input as safe data provides ample opportunity for security breaches quite easily.

Sensitive data is often made available unintentionally by programs to people who should not have any access to the information. Such exposure can result in disaster if the information falls in the hands of a malicious hacker. Unauthorized access is difficult to deal with if users can't be authenticated using user names and passwords and/or hostname/IP address based access control cannot be established. In the following sections, we discuss these risks and potential solutions in detail.

ENTERPRISE SECURITY API

OWASP Enterprise Security API provides a list of libraries and functionalities for building secure and robust web applications. The goal is to decouple security aspects from business aspects within the applications.

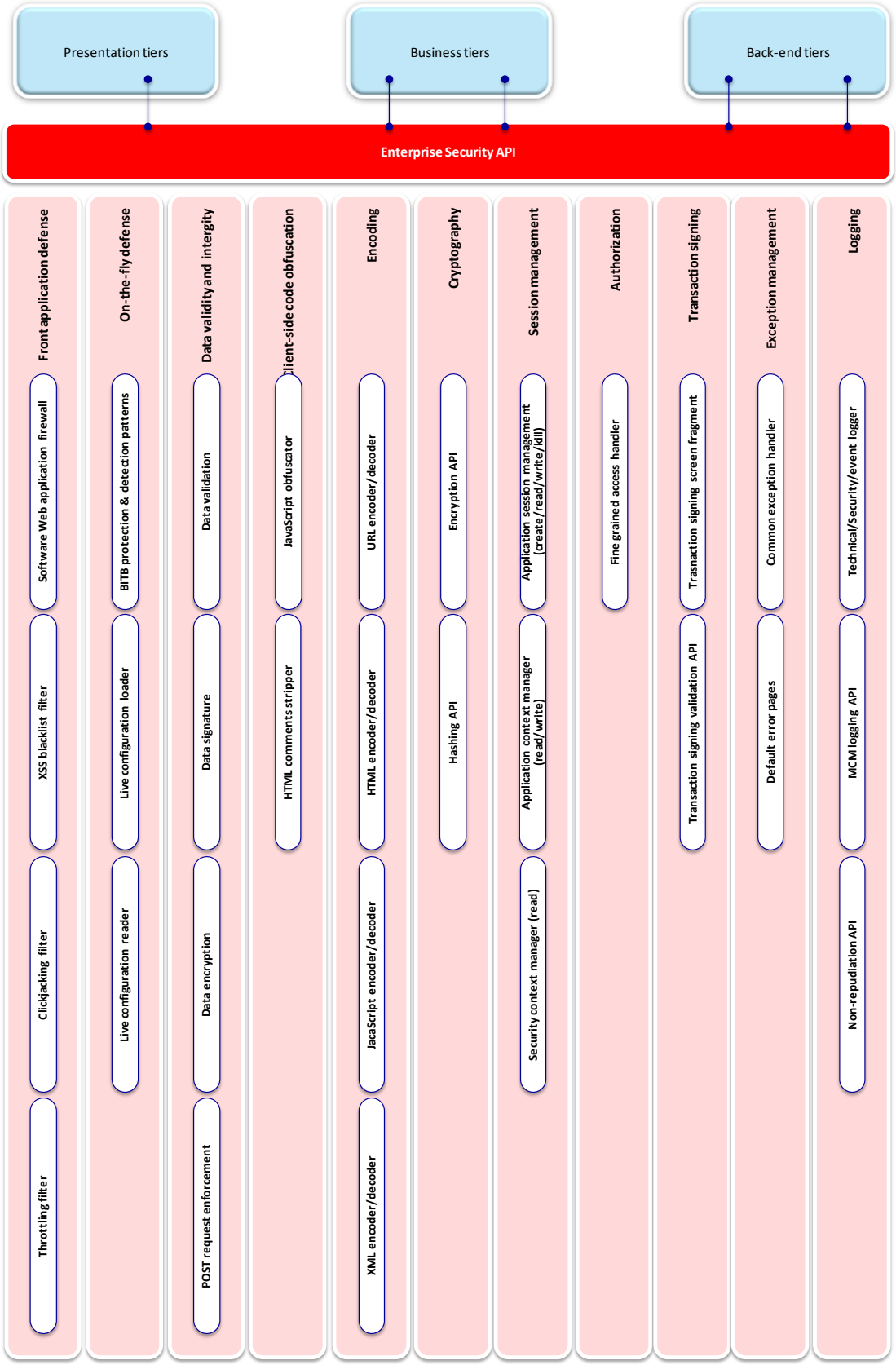


Figure 17 - Enterprise Security API features

Main features are:

- Front application defense :

  It provides external front application layer defense to protect the MIB internet platform against application threats:

  - Web Application Firewall: software application component which is embedded within the application itself. It could be more or less intrusive and specific with each application. Specific countermeasures can be implemented;
  - XSS black list filter: blacklisted XSS protection and detection component;
  - Click jacking filter: click jacking threat protection component;
  - Throttling filter: limit the number of requests supported by the application – protect against DoS application attacks.

- On-the-fly defense:

  - BITB protection & detection patterns: It provides protection and detection patterns against financial Trojan threats. These protections can be activated “on-the-fly” without any application deployment (under attack for example);
  - Live configuration loader / reader: It provides flexible and configurable mechanism to activate and deactivate functionalities within the internet platform (for instance, deactivates money transfer application when platform is under attack, or only allows transfer to trusted beneficiaries…).

- Data validity & integrity:

  - Data validation: It provides white list input validation API;
  - Data signature: It provides XSRF token protection, parameter tampering protection, technical field modification protection;
  - Data encryption: It provides sensitive data protection.

- Client side code obfuscation:

  - JavaScript obfuscator: It provides functionality to obfuscate client side code;
  - HTML comments stripper: It provides functionality to remove client side comments (HTML and JavaScript comments).

- Encoding:

  - It provides common encoder and decoder to protect the platform against code injections (XML, HTML, LDAP, JavaScript, URL…).

- Cryptography:

  - It provides API to support types of encryptions and hashing functions.

- Session management:
    - It provides API to manage application session lifecycle (create, invalidate);
    - It provides API to access application session (read, write).
- Authorization:
    - It provides API to handle fine grained authorization.
- Transaction signing:
    - It provides API to validate signature of a transaction when it is required;
    - It provides screen fragment to build the transaction signing block within the user interface.
- Exception management:
    - It provides common exception handler to centralize exception management and avoid information disclosure within the exception handling process;
    - It provides default error pages on different components within the internet platform;
- Logging:
    - It provides API to log security events;
    - It provides API to log technical events;
    - It provides API to ensure non repudiation in different log records;
    - It provides way to manage XlogID propagation between application components;
    - XlogID is a unique ID shared between all different logs (technical, security and transactional logs). The XlogID is unique for a user during his session. It allows correlations and helps for investigations during the incident handling process.

HARDENED APPLICATION SECURITY

Based on application security capabilities provided in the Enterprise Security API, the following diagram describes a hardened application sample (at Presentation Layer level) regarding web application attacks vectors.

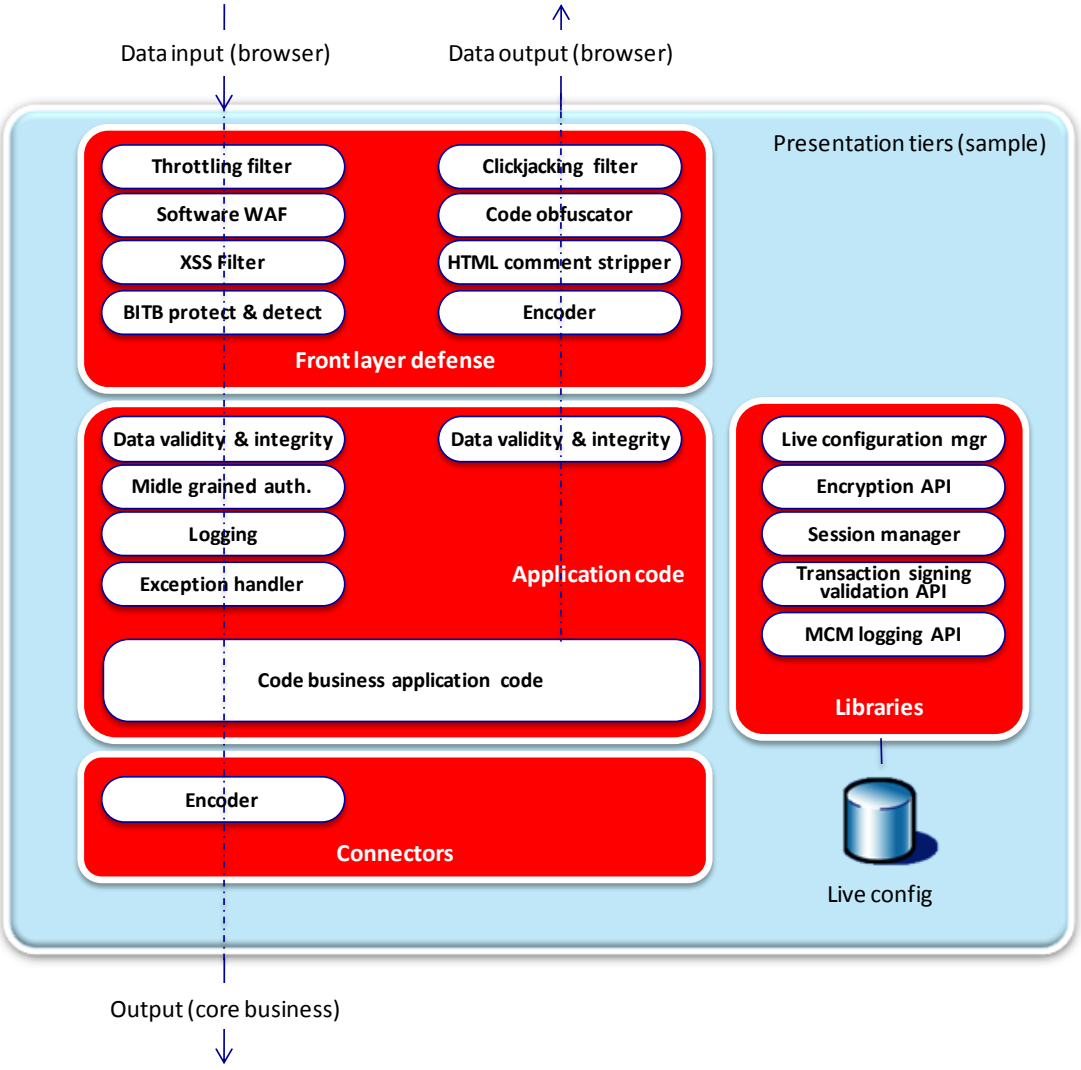


Figure 18 - Application security hardening sample

APPLICATION THREATS AND COUNTERMEASURES

The stateless nature of HTTP means that tracking per-user session state becomes the responsibility of the application. As a precursor to this, the application must be able to identify the user by using some form of authentication. Given that all subsequent authorization decisions are based on the user's identity, it is essential that the authentication process is secure and that the session handling mechanism used to track authenticated users is equally well protected. Designing secure authentication and session management mechanisms are just a couple of the issues facing Web application designers and developers. Other challenges occur because input and output data passes over public networks. Preventing parameter manipulation and the disclosure of sensitive data are other top issues.

Some of the top issues that must be addressed with secure design practices are shown below.

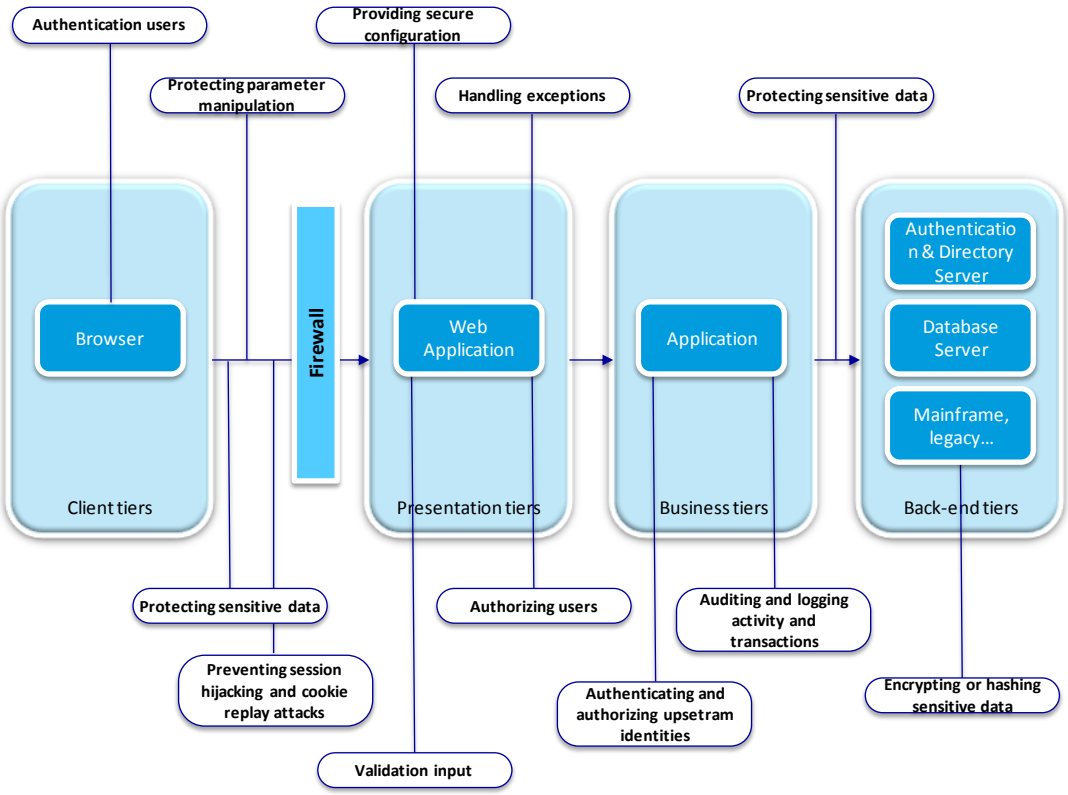


Figure 19 - Web application threats and countermeasures

The design guidelines in this chapter are organized by application vulnerability category. Poor design in these areas leads to security vulnerabilities.

Deployment Considerations

During the application design phase, corporate security policies should be reviewed and procedures together with the infrastructure to be deployed on. Application design must reflect the restrictions of the target environment. Design tradeoffs are merely required, because of protocol or port restrictions, or specific deployment topologies. Involve members of the network and infrastructure teams to help with the process of identification of such constraints.

Figure below shows the various deployment aspects that require design time consideration.

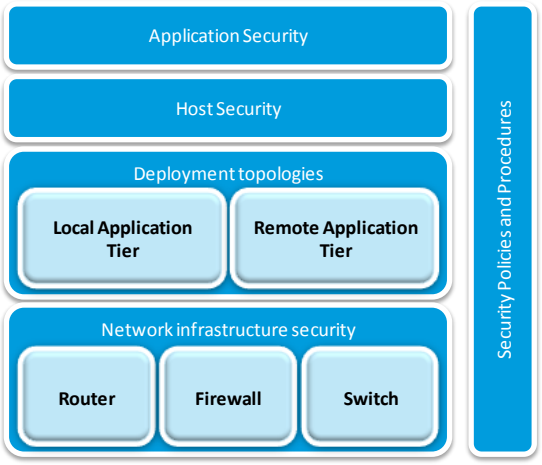


Figure 20 - Deployment time requirements

Security Policies and Procedures

Security policy determines what your applications are allowed to do and what the users of the application are permitted to do. More importantly, they define restrictions to determine what applications and users are not allowed to do. Identify and work within the framework defined by your corporate security policy while designing your applications to make sure you do not breach policy that might prevent the application being deployed.

Network Infrastructure Components

Make sure you understand the network structure provided by your target environment and understand the baseline security requirements of the network in terms of filtering rules, port restrictions, supported protocols, and so on.

Identify how firewalls and firewall policies are likely to affect your application's design and deployment. There may be firewalls to separate the Internet-facing applications from the internal network. There may be additional firewalls in front of the database. These can affect your possible communication ports and, therefore, authentication options from the Web server to remote application and database servers. For example, Windows authentication requires additional ports.

At the design stage, consider what protocols, ports, and services are allowed to access internal resources from the Web servers in the perimeter network. Also identify the protocols and ports that the application design requires and analyze the potential threats that occur from opening new ports or using new protocols.

Communicate and record any assumptions made about network and application layer security and which component will handle what. This prevents security controls from being missed when both development and network teams assume that the other team is addressing the issue. Pay attention to the security defenses that your application relies upon the network to provide. Consider the implications of a change in network configuration. How much security have you lost if you implement a specific network change.

Deployment Topologies

Your application's deployment topology and whether you have a remote application tier is a key consideration that must be incorporated in your design. If you have a remote application tier, you need to consider how to secure the network between servers to address the network eavesdropping threat and to provide privacy and integrity for sensitive data.

Also consider identity flow and identify the accounts that will be used for network authentication when your application connects to remote servers. A common approach is to use a least privileged process account and create a duplicate (mirrored) account on the remote server with the same password. Alternatively, you might use a domain process account, which provides easier administration but is more problematic to secure because of the difficulty of limiting the account's use throughout the network. An intervening firewall or separate domains without trust relationships often makes the local account approach the only viable option.

Intranet, Extranet, and Internet

Intranet, extranet, and Internet application scenarios each present design challenges. Questions that you should consider include: How will you flow caller identity through multiple application tiers to back- end resources? Where will you perform authentication? Can you trust authentication at the

front end and then use a trusted connection to access back-end resources? In extranet scenarios, you also must consider whether you trust partner accounts.

Input Validation

Input validation is a challenging issue and the primary burden of a solution falls on application developers. However, proper input validation is one of your strongest measures of defense against today's application attacks. Proper input validation is an effective countermeasure that can help prevent XSS, SQL injection, buffer overflows, and other input attacks.

Input validation is challenging because there is not a single answer for what constitutes valid input across applications or even within applications. Likewise, there is no single definition of malicious input. Adding to this difficulty is that what your application does with this input influences the risk of exploit. For example, do you store data for use by other applications or does your application consume input from data sources created by other applications.

Assume All Input Is Malicious

Input validation starts with a fundamental supposition that all input is malicious until proven otherwise. Whether input comes from a service, a file share, a user, or a database, validate your input if the source is outside your trust boundary. For example, if you call an external Web service that returns strings, how do you know that malicious commands are not present? Also, if several applications write to a shared database, when you read data, how do you know whether it is safe?

Centralize Your Approach

Make your input validation strategy a core element of your application design. Consider a centralized approach to validation, for example, by using common validation and filtering code in shared libraries. This ensures that validation rules are applied consistently. It also reduces development effort and helps with future maintenance.

In many cases, individual fields require specific validation, for example, with specifically developed regular expressions. However, you can frequently factor out common routines to validate regularly used fields such as e-mail addresses, titles and names, postal addresses including ZIP or postal codes, and so on.

Do Not Rely on Client-Side Validation

Server-side code should perform its own validation. What if an attacker bypasses your client, or shuts off your client-side script routines, for example, by disabling JavaScript? Use client-side validation to help reduce the number of round trips to the server but do not rely on it for security. This is an example of defense in depth.

Be Careful with Canonicalization Issues

Data in canonical form is in its most standard or simplest form. Canonicalization is the process of converting data to its canonical form. File paths and URLs are particularly prone to canonicalization issues and many well-known exploits are a direct result of canonicalization bugs.

You should generally try to avoid designing applications that accept input file names from the user to avoid canonicalization issues. Consider alternative designs instead. For example, let the application determine the file name for the user.

If you do need to accept input file names, make sure they are strictly formed before making security decisions such as granting or denying access to the specified file.

Constrain, Reject, and Sanitize Your Input

The preferred approach to validating input is to constrain what you allow from the beginning. It is much easier to validate data for known valid types, patterns, and ranges than it is to validate data by looking for known bad characters. When you design your application, you know what your application expects. The range of valid data is generally a more finite set than potentially malicious input. However, for defense in depth you may also want to reject known bad input and then sanitize the input. The recommended strategy is shown below.

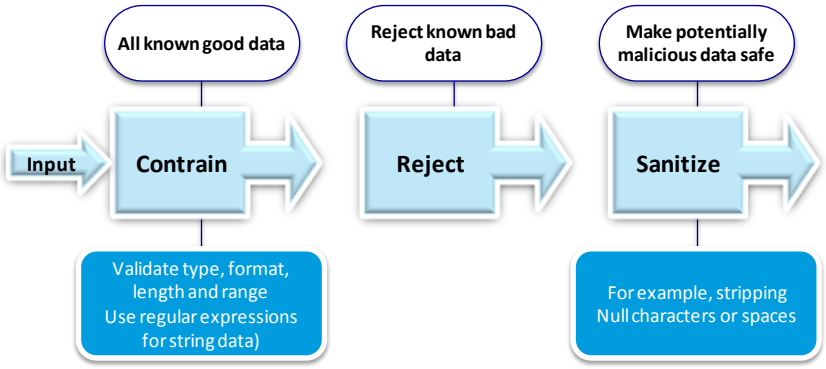


Figure 21 - Input analysis process

To create an effective input validation strategy, be aware of the following approaches and their tradeoffs.

Constraint Input

Constraining input is about allowing good data. This is the preferred approach. The idea here is to define a filter of acceptable input by using type, length, format, and range. Define what acceptable input for your application fields is and enforce it. Reject everything else as bad data.

Constraining input may involve setting character sets on the server so that you can establish the canonical form of the input in a localized way.

Validate Data for Type, Length, Format, and Range

Use strong type checking on input data wherever possible, for example, in the classes used to manipulate and process the input data and in data access routines. For example, use parameterized stored procedures for data access to benefit from strong type checking of input fields.

String fields should also be length checked and in many cases checked for appropriate format. For example, ZIP codes, personal identification numbers, and so on have well defined formats that can be validated using regular expressions. Thorough checking is not only good programming practice; it makes it more difficult for an attacker to exploit your code. The attacker may get through your type check, but the length check may make executing his favorite attack more difficult.

Reject Known Bad Input

Deny "bad" data; although do not rely completely on this approach. This approach is generally less effective than using the "allow" approach described earlier and it is best used in combination. To deny bad data assumes your application knows all the variations of malicious input. Remember that there are multiple ways to represent characters. This is another reason why "allow" is the preferred approach.

While useful for applications that are already deployed and when you cannot afford to make significant changes, the "deny" approach is not as robust as the "allow" approach because bad data, such as patterns that can be used to identify common attacks, do not remain constant. Valid data remains constant while the range of bad data may change over time.

SANITIZE INPUT

Sanitizing is about making potentially malicious data safe. It can be helpful when the range of input that is allowed cannot guarantee that the input is safe. This includes anything from stripping a null from the end of a user-supplied string to escaping out values so they are treated as literals.

Another common example of sanitizing input in Web applications is using URL encoding or HTML encoding to wrap data and treat it as literal text rather than executable script. HTML and URL encode methods escape out HTML characters, and encode a URL so that it is a valid URI request.

AUTHENTICATION

Authentication is the process of determining caller identity. There are three aspects to consider:

- Identify where authentication is required in your application. It is generally required whenever a trust boundary is crossed. Trust boundaries usually include assemblies, processes, and hosts;
- Validate who the caller is. Users typically authenticate themselves with user names and passwords;
- Identify the user on subsequent requests. This requires some form of authentication token.

Many Web applications use a password mechanism to authenticate users, where the user supplies a user name and password in an HTML form. The issues and questions to consider here include:

- **Are user names and passwords sent in plaintext over an insecure channel?** If so, an attacker can eavesdrop with network monitoring software to capture the credentials. The countermeasure here is to secure the communication channel by using Secure Socket Layer (SSL);
- **How are the credentials stored?** If you are storing user names and passwords in plaintext, either in files or in a database, you are inviting trouble. What if your application directory is improperly configured and an attacker browses to the file and downloads its contents or adds a new privileged logon account? What if a disgruntled administrator takes your database of user names and passwords?
- **How are the credentials verified?** There is no need to store user passwords if the sole purpose is to verify that the user knows the password value. Instead, you can store a verifier in the form of a hash value and re-compute the hash using the user-supplied value during the logon process. To mitigate the threat of dictionary attacks against the credential store, use strong passwords and combine a randomly generated salt value with the password hash;
- **How is the authenticated user identified after the initial logon?** Some form of authentication ticket, for example an authentication cookie, is required. How is the cookie secured? If it is sent across an insecure channel, an attacker can capture the cookie and use it to access the application. A stolen authentication cookie is a stolen logon.

Separate Public and Restricted Areas

A public area of your site can be accessed by any user anonymously. Restricted areas can be accessed only by specific individuals and the users must authenticate with the site. Consider a typical retail Web site. You can browse the product catalog anonymously. When you add items to a shopping cart, the application identifies you with a session identifier. Finally, when you place an order, you perform a secure transaction. This requires you to log in to authenticate your transaction over SSL.

By partitioning your site into public and restricted access areas, you can apply separate authentication and authorization rules across the site and limit the use of SSL. To avoid the unnecessary performance overhead associated with SSL, design your site to limit the use of SSL to the areas that require authenticated access.

Use Account Lockout Policies for End-User Accounts

Disable end-user accounts or write events to a log after a set number of failed logon attempts. If you are using authentication based on the Kerberos protocol, these policies can be configured and applied automatically by the operating system. With Forms authentication, these policies are the responsibility of the application and must be incorporated into the application design.

Support Password Expiration Periods

Passwords should not be static and should be changed as part of routine password maintenance through password expiration periods. Consider providing this type of facility during application design.

Be Able to Disable Accounts

If the system is compromised, being able to deliberately invalidate credentials or disable accounts can prevent additional attacks.

Do Not Store Passwords in User Stores

If you must verify passwords, it is not necessary to actually store the passwords. Instead, store a one way hash value and then re-compute the hash using the user-supplied passwords. To mitigate the threat of dictionary attacks against the user store, use strong passwords and incorporate a random salt value with the password.

Require Strong Passwords

Do not make it easy for attackers to crack passwords. There are many guidelines available, but a general practice is to require a minimum of eight characters and a mixture of uppercase and lowercase characters, numbers, and special characters. Whether you are using the platform to enforce these for you, or you are developing your own validation, this step is necessary to counter brute-force attacks where an attacker tries to crack a password through systematic trial and error. Use regular expressions to help with strong password validation.

Do Not Send Passwords Over the Wire in Plaintext

Plaintext passwords sent over a network are vulnerable to eavesdropping. To address this threat, secure the communication channel, for example, by using SSL to encrypt the traffic.

Protect Authentication Cookies

A stolen authentication cookie is a stolen logon. Protect authentication tickets using encryption and secure communication channels. Also limit the time interval in which an authentication ticket remains valid, to counter the spoofing threat that can result from replay attacks, where an attacker captures the cookie and uses it to gain illicit access to your site. Reducing the cookie timeout does not prevent

replay attacks but it does limit the amount of time the attacker has to access the site using the stolen cookie.

Authorization

Authorization determines what the authenticated identity can do and the resources that can be accessed. Improper or weak authorization leads to information disclosure and data tampering. Defense in depth is the key security principle to apply to your application's authorization strategy.

Use Multiple Gatekeepers

On the server side, you can use IP Security Protocol (IPSec) policies to provide host restrictions to restrict server-to-server communication. For example, an IPSec policy might restrict any host apart from a nominated Web server from connecting to a database server. IIS provides Web permissions and Internet Protocol/ Domain Name System (IP/DNS) restrictions. IIS Web permissions apply to all resources requested over HTTP regardless of the user. They do not provide protection if an attacker manages to log on to the server.

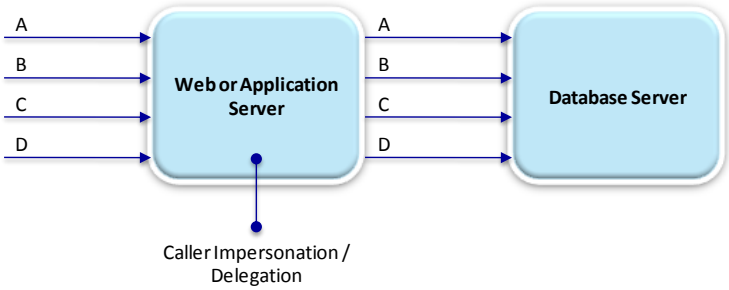
Restrict User Access to System Level Resources

System level resources include files, folders, registry keys, Active Directory objects, database objects, event logs, and so on. Use Windows Access Control Lists (ACLs) to restrict which users can access what resources and the types of operations that they can perform. Pay particular attention to anonymous Internet user accounts; lock these down with ACLs on resources that explicitly deny access to anonymous users.

Consider Authorization Granularity

There are three common authorization models, each with varying degrees of granularity and scalability.

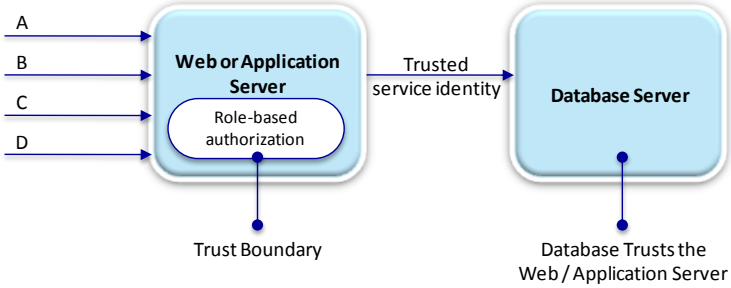
The most granular approach relies on impersonation. Resource access occurs using the security context of the caller. ACLs on the secured resources (typically files or tables, or both) determine whether the caller is allowed to access the resource. If your application provides access primarily to user specific resources, this approach may be valid. It has the added advantage that operating system level auditing can be performed across the tiers of your application, because the original caller's security context flows at the operating system level and is used for resource access. However, the approach suffers from poor application scalability because effective connection pooling for database access is not possible. As a result, this approach is most frequently found in limited scale intranet-based applications.



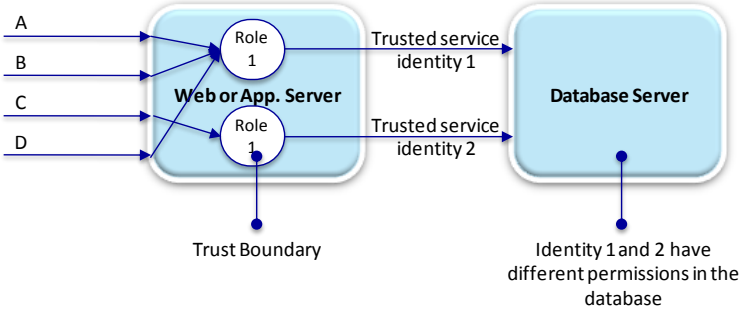
The least granular but most scalable approach uses the application's process identity for resource access. This approach supports database connection pooling but it means that the permissions granted to the application's identity in the database are common, irrespective of the identity of the

original caller. The primary authorization is performed in the application's logical middle tier using roles, which group together users who share the same privileges in the application. Access to classes and methods is restricted based on the role membership of the caller. To support the retrieval of per user data, a common approach is to include an identity column in the database tables and use query parameters to restrict the retrieved data.

This model is referred to as the trusted subsystem or sometimes as the trusted server model. It is shown below.



The third option is to use a limited set of identities for resource access based on the role membership of the caller. This is really a hybrid of the two models described earlier. Callers are mapped to roles in the application's logical middle tier, and access to classes and methods is restricted based on role membership. Downstream resource access is performed using a restricted set of identities determined by the current caller's role membership. The advantage of this approach is that permissions can be assigned to separate logins in the database, and connection pooling is still effective with multiple pools of connections. The downside is that creating multiple thread access tokens used to establish different security contexts for downstream resource access using Windows authentication is a privileged operation that requires privileged process accounts. This is counter to the principle of least privilege. The hybrid model using multiple trusted service identities for downstream resource access is shown below.



Configuration Management

Carefully consider your Web application's configuration management functionality. Most applications require interfaces that allow content developers, operators, and administrators to configure the application and manage items such as Web page content, user accounts, user profile information, and database connection strings. If remote administration is supported, how are the administration interfaces secured? The consequences of a security breach to an administration interface can be severe, because the attacker frequently ends up running with administrator privileges and has direct access to the entire site.

Secure Your Administration Interfaces

It is important that configuration management functionality is accessible only by authorized operators and administrators. A key part is to enforce strong authentication over your administration interfaces, for example, by using certificates.

If possible, limit or avoid the use of remote administration and require administrators to log on locally. If you need to support remote administration, use encrypted channels, for example, with SSL or VPN technology, because of the sensitive nature of the data passed over administrative interfaces. Also consider limiting remote administration to computers on the internal network by using IPSec policies, to further reduce risk.

Secure Your Configuration Stores

Text-based configuration files, the registry, and databases are common options for storing application configuration data. If possible, avoid using configuration files in the application's Web space to prevent possible server configuration vulnerabilities resulting in the download of configuration files. Whatever approach you use, secure access to the configuration store, for example, by using Windows ACLs or database permissions. Also avoid storing plaintext secrets such as database connection strings or account credentials. Secure these items using encryption and then restrict access to the registry key, file, or table that contains the encrypted data.

Separate Administration Privileges

If the functionality supported by the features of your application's configuration management varies based on the role of the administrator, consider authorizing each role separately by using role-based authorization. For example, the person responsible for updating a site's static content should not necessarily be allowed to change a customer's credit limit.

Use Least Privileged Process and Service Accounts

An important aspect of your application's configuration is the process accounts used to run the Web server process and the service accounts used to access downstream resources and systems. Make sure these accounts are set up as least privileged. If an attacker manages to take control of a process, the process identity should have very restricted access to the file system and other system resources to limit the damage that can be done.

Sensitive Data

Applications that deal with private user information such as credit card numbers, addresses, medical records, and so on should take special steps to make sure that the data remains private and unaltered. In addition, secrets used by the application's implementation, such as passwords and database connection strings, must be secured. The security of sensitive data is an issue while the data is stored in persistent storage and while it is passed across the network.

Secrets

Secrets include passwords, database connection strings, and credit card numbers.

Do Not Store Secrets if You Can Avoid It

Storing secrets in software in a completely secure fashion is not possible. An administrator, who has physical access to the server, can access the data. For example, it is not necessary to store a secret when all you need to do is verify whether a user knows the secret. In this case, you can store a hash

value that represents the secret and compute the hash using the user-supplied value to verify whether the user knows the secret.

Do Not Store Secrets in Code

Do not hard code secrets in code. Even if the source code is not exposed on the Web server, it is possible to extract string constants from compiled executable files. Configuration vulnerability may allow an attacker to retrieve the executable.

Do Not Store Database Connections, Passwords, or Keys in Plaintext

Avoid storing secrets such as database connection strings, passwords, and keys in plaintext. Use encryption and store encrypted strings.

Use cryptography API for Encrypting Secrets

To store secrets such as database connection strings or service account credentials, use system kernel cryptography API (such as Linux APIs or DPAPI for Microsoft platform). The main advantage to using those API is that the platform system manages the encryption/decryption key that is no longer an issue for the application. The key is either tied to a user account or to a specific server, depending on flags passed to the API functions.

Cryptography API is best suited for encrypting information that can be manually recreated when the master keys are lost, for example, because a damaged server requires an operating system re-install. Data that cannot be recovered because you do not know the plaintext value, for example, customer credit card details, require an alternate approach that uses traditional symmetric key-based cryptography such as the use of triple-DES.

Sensitive Per User Data

Sensitive data such as logon credentials and application level data such as credit card numbers, bank account numbers, and so on, must be protected. Privacy through encryption and integrity through message authentication codes (MAC) are the key elements.

Retrieve Sensitive Data on Demand

The preferred approach is to retrieve sensitive data on demand when it is needed instead of persisting or caching it in memory.

Cache the Encrypted Secret

Retrieve the secret when the application loads and then cache the encrypted secret in memory, decrypting it when the application uses it. Clear the plaintext copy when it is no longer needed. This approach avoids accessing the data store on a per request basis.

Cache the Plaintext Secret

Avoid the overhead of decrypting the secret multiple times and store a plaintext copy of the secret in memory. This is the least secure approach but offers the optimum performance. Benchmark the other approaches before guessing that the additional performance gain is worth the added security risk.

Encrypt the Data or Secure the Communication Channel

If you are sending sensitive data over the network to the client, encrypt the data or secure the channel. A common practice is to use SSL between the client and Web server. Between servers, an increasingly common approach is to use IPSec. For securing sensitive data that flows through several

intermediaries, for example, Web service Simple Object Access Protocol (SOAP) messages, use message level encryption.

Do Not Store Sensitive Data in Persistent Cookies

Avoid storing sensitive data in persistent cookies. If you store plaintext data, the end user is able to see and modify the data. If you encrypt the data, key management can be a problem. For example, if the key used to encrypt the data in the cookie has expired and been recycled, the new key cannot decrypt the persistent cookie passed by the browser from the client.

Do Not Pass Sensitive Data Using the HTTP-GET Protocol

You should avoid storing sensitive data using the HTTP-GET protocol because the protocol uses query strings to pass data. Sensitive data cannot be secured using query strings and query strings are often logged by the server.

Session Management

Web applications are built on the stateless HTTP protocol, so session management is an application-level responsibility. Session security is critical to the overall security of an application.

Use SSL to Protect Session Authentication Cookies

Do not pass authentication cookies over HTTP connections. Set the secure cookie property within authentication cookies, which instructs browsers to send cookies back to the server only over HTTPS connections.

Encrypt the Contents of the Authentication Cookies

Encrypt the cookie contents even if you are using SSL. This prevents an attacker viewing or modifying the cookie if he manages to steal it through an XSS attack. In this event, the attacker could still use the cookie to access your application, but only while the cookie remains valid.

Limit Session Lifetime

Reduce the lifetime of sessions to mitigate the risk of session hijacking and replay attacks. The shorter the session, the less time an attacker has to capture a session cookie and use it to access your application.

Protect Session State from Unauthorized Access

Consider how session state is to be stored. For optimum performance, you can store session state in the Web application's process address space. However, this approach has limited scalability and implications in Web farm scenarios, where requests from the same user cannot be guaranteed to be handled by the same server. In this scenario, an out-of-process state store on a dedicated state server or a persistent state store in a shared database is required.

You should secure the network link from the Web application to state store using IPSec or SSL to mitigate the risk of eavesdropping. Also consider how the Web application is to be authenticated by the state store. Use Windows authentication where possible to avoid passing plaintext authentication credentials across the network and to benefit from secure Windows account policies.

Cryptography

Cryptography in its fundamental form provides the following:

- **Privacy** (Confidentiality). This service keeps a secret confidential;

- **Non-Repudiation** (Authenticity). This service makes sure a user cannot deny sending a particular message;
- **Tamperproofing** (Integrity). This service prevents data from being altered;
- **Authentication**. This service confirms the identity of the sender of a message.

Web applications frequently use cryptography to secure data in persistent stores or as it is transmitted across networks.

Do Not Develop Your Own Cryptography

Cryptographic algorithms and routines are notoriously difficult to develop successfully. As a result, you should use the tried and tested cryptographic services provided by the platform. Do not develop custom implementations because these frequently result in weak protection.

Keep Unencrypted Data Close to the Algorithm

When passing plaintext to an algorithm, do not obtain the data until you are ready to use it, and store it in as few variables as possible.

Use the Correct Algorithm and Correct Key Size

It is important to make sure you choose the right algorithm for the right job and to make sure you use a key size that provides a sufficient degree of security. Larger key sizes generally increase security. The following list summarizes the major algorithms together with the key sizes that each uses:

- Data Encryption Standard (DES) 64-bit key (8 bytes);
- TripleDES 128-bit key or 192-bit key (16 or 24 bytes);
- Rijndael 128–256 bit keys (16–32 bytes);
- RSA 384–16,384 bit keys (48–2,048 bytes).

For large data encryption, use the TripleDES symmetric encryption algorithm. For slower and stronger encryption of large data, use Rijndael. To encrypt data that is to be stored for short periods of time, you can consider using a faster but weaker algorithm such as DES. For digital signatures, use Rivest, Shamir, and Adleman (RSA) or Digital Signature Algorithm (DSA). For hashing, use the Secure Hash Algorithm (SHA) 1.0. For keyed hashes, use the Hash-based Message Authentication Code (HMAC) SHA1.0.

Secure Your Encryption Keys

An encryption key is a secret number used as input to the encryption and decryption processes. For encrypted data to remain secure, the key must be protected. If an attacker compromises the decryption key, your encrypted data is no longer secure.

Use cryptography API to Avoid Key Management

As mentioned previously, one of the major advantages of using a cryptography API is that the key management issue is handled by the operating system. The key that API uses is derived from the password that is associated with the process account that calls the API functions.

Cycle Your Keys Periodically

Generally, a static secret is more likely to be discovered over time. Questions to keep in mind are: Did you write it down somewhere? Did Bob, the administrator with the secrets, change positions in your company or leave the company? Do not overuse keys.

Parameter Manipulation

With parameter manipulation attacks, the attacker modifies the data sent between the client and Web application. This may be data sent using query strings, form fields, cookies, or in HTTP headers.

Encrypt Sensitive Cookie State

Cookies may contain sensitive data such as session identifiers or data that is used as part of the server-side authorization process. To protect this type of data from unauthorized manipulation, use cryptography to encrypt the contents of the cookie.

Make Sure that Users Do Not Bypass Your Checks

Make sure that users do not bypass your checks by manipulating parameters. URL parameters can be manipulated by end users through the browser address text box. For example, the URL http://www.<***YourSite***>/<***YourApp***>/sessionId=10 has a value of 10 that can be changed to some random number to receive different output. Make sure that you check this in server-side code, not in client-side JavaScript, which can be disabled in the browser.

Validate All Values Sent from the Client

Restrict the fields that the user can enter and modify and validate all values coming from the client. If you have predefined values in your form fields, users can change them and post them back to receive different results. Permit only known good values wherever possible. For example, if the input field is for a state, only inputs matching a state postal code should be permitted.

Do Not Trust HTTP Header Information

HTTP headers are sent at the start of HTTP requests and HTTP responses. Your Web application should make sure it does not base any security decision on information in the HTTP headers because it is easy for an attacker to manipulate the header. For example, the referer field in the header contains the URL of the Web page from where the request originated. Do not make any security decisions based on the value of the referer field, for example, to check whether the request originated from a page generated by the Web application, because the field is easily falsified.

Exception Management

Secure exception handling can help prevent certain application-level denial of service attacks and it can also be used to prevent valuable system-level information useful to attackers from being returned to the client. For example, without proper exception handling, information such as database schema details, operating system versions, stack traces, file names and path information, SQL query strings and other information of value to an attacker can be returned to the client.

A good approach is to design a centralized exception management and logging solution and consider providing hooks into your exception management system to support instrumentation and centralized monitoring to help system administrators.

Do Not Leak Information to the Client

In the event of a failure, do not expose information that could lead to information disclosure. For example, do not expose stack trace details that include function names and line numbers in the case

of debug builds (which should not be used on production servers). Instead, return generic error messages to the client.

Log Detailed Error Messages

Send detailed error messages to the error log. Send minimal information to the consumer of your service or application, such as a generic error message and custom error log ID that can subsequently be mapped to detailed message in the event logs. Make sure that you do not log passwords or other sensitive data.

Catch Exceptions

Use structured exception handling and catch exception conditions. Doing so avoids leaving your application in an inconsistent state that may lead to information disclosure. It also helps protect your application from denial of service attacks. Decide how to propagate exceptions internally in your application and give special consideration to what occurs at the application boundary.

Auditing and Logging

You should audit and log activity across the tiers of your application. Using logs, you can detect suspicious-looking activity. This frequently provides early indications of a full-blown attack and the logs help address the repudiation threat where users deny their actions. Log files may be required in legal proceedings to prove the wrongdoing of individuals. Generally, auditing is considered most authoritative if the audits are generated at the precise time of resource access and by the same routines that access the resource.

Audit and Log Access Across Application Tiers

Audit and log access across the tiers of your application for non-repudiation. Use a combination of application-level logging and platform auditing features, such as Web server and SQL Server auditing.

Consider Identity Flow

Consider how your application will flow caller identity across multiple application tiers. You have two basic choices. You can flow the caller's identity at the operating system level using the Kerberos protocol delegation. This allows you to use operating system level auditing. The drawback with this approach is that it affects scalability because it means there can be no effective database connection pooling at the middle tier. Alternatively, you can flow the caller's identity at the application level and use trusted identities to access back-end resources. With this approach, you have to trust the middle tier and there is a potential repudiation risk. You should generate audit trails in the middle tier that can be correlated with back-end audit trails.

Log Key Events

The types of events that should be logged include successful and failed logon attempts, modification of data, and retrieval of data, network communications, and administrative functions such as the enabling or disabling of logging. Logs should include the time of the event, the location of the event including the machine name, the identity of the current user, the identity of the process initiating the event, and a detailed description of the event.

Secure Log Files

Secure log files using ACLs and restrict access to the log files. This makes it more difficult for attackers to tamper with log files to cover their tracks. Minimize the number of individuals who can manipulate the log files. Authorize access only to highly trusted accounts such as administrators.

Back Up and Analyze Log Files Regularly

There's no point in logging activity if the log files are never analyzed. Log files should be removed from production servers on a regular basis. The frequency of removal is dependent upon your application's level of activity. Your design should consider the way that log files will be retrieved and moved to offline servers for analysis. Any additional protocols and ports opened on the Web server for this purpose must be securely locked down.

APPENDIXES

GLOSSARY AND TERMS

REVERSE PROXY

Ability to act as a reverse proxy server intercepting all access requests to web-resources to enforce authentication, authorization and auditing. The Reverse Proxy functionality should be able to hide internal topology of web-servers and act as a single entry point.

We distinguish between 2 operational modes: *proxy-mode* and *agent-mode*. While both scenarios offer equivalent security capabilities for protecting web-based resources, there are some key differences between the 2 operational modes:

- *Proxy Mode*: In this mode, web-access requests are first intercepted by the reverse proxy component that enforces authentication, authorization, and auditing based on the user role and the resource being accessed. From the user perspective, the reverse proxy appears to be the content server;
- *Agent Mode*: In this mode, agents running on application servers are responsible for managing application-specific web-access requests. Web-SSO agents can be optimized for specific applications, such as Apache HTTP server. Agents communicate access requests to a central authorization server on behalf of application servers. Depending on the outcome of the authorization verification, the authorization server sends back the outcome to the requesting Web-SSO agent.

Protect application servers within a DMZ from Internet attacks. Decouple security services logic from application services logic to achieve a consistent reusability of security services and a central point of enforcement across applications.

To the “outside world” the reverse proxy server will appear as the web server. Newly deployed web services must be made available through the reverse proxy. The latter has to be configured to forward web access requests to the new web server.

Main advantages of using a proxy mode can be summarize by the following:

- Single point of security policy enforcement;
- Increased security, by placing fewer machines in the DMZ;
- Increased security, by hiding application-related tokens as these are never sent to the browser;
- Easy integration and transparency to end users of web applications and different technologies of web- and application servers;
- Makes it easy to integrate new applications into the authentication and authorization services;
- Load balancing of backend servers by the reverse proxy allows us to implement stickiness for (statefull) application servers over TCP and SSL.

Authentication Enforcement

This functionality allows us to force all web-resources access requests from the Internet to be authenticated before granting access. The authentication process should support common authentication methods: basic authentication (username/password), strong authentication based on One-Time-Password, and supporting authentication technologies (smartcards, Vasco DigiPass, RSA SecurID, etc.).

Authentication enforcement should support below principles:

- Flexibility in enforcing different authentication methods depending on the business service at hand and according to security policy and business decisions;
- Flexibility in enforcing a different authentication methods for selected user communities;
- Decisions on which authentication methods should have no impact on the business applications.

The authentication process should support local Identity & Access Management (IAM) and User Directory platforms available. Commonly used standard authentication protocols should be supported, such as Kerberos tickets, X509 Certificate, etc.

Single Sign On

To enhance end-user experience, the SSO functionality allows users to be authenticated only once for multiple web-resources, thereby removing the need for re-authenticating users each time an access request is initiated for other services. Depending on his/her profile, the user is authenticated once and will be allowed access to the business services he/she is authorized for during an entire session. Session timeouts should be supported (e.g., request the user to be re-authenticated beyond a maximum session duration time or in case of idle sessions).

Federated-IAM Support

The Web-SSO should be able to inter-operate with other Web-SSO domains in a Federated Identity & Access Management (F-IAM) context, where certain web-resources could be under control of a separate Web-SSO domain.

In this context, the requirement is that the Web-SSO should be able to pass-on a user security context to another Web-SSO, based on standard protocols, such as SAML (Security Assertion Markup Language).

Single Sign Off

Conversely, in case a user has been authenticated for several services with several active sessions, the Single-Sign-Off requirement means that all active sessions are closed at once when the user signs-off of any one of these sessions. This means that the Web-SSO should be able to maintain a state and track all active sessions for any given user.

Authorization

The technology should support Role-Based Access Control (RBAC) to manage access to web-resources, where roles are defined according to customer service contracts that define which on-line services a customer is entitled to. In addition to scalability benefits, RBAC significantly reduces access

administration overhead as the authorization logic is based on roles (or user groups) rather than individual user identities.

Web-SSO must be able to re-use existing roles, access permissions, and user directories, through minimal integration effort.

Three 3 authorizations level are distinguished:

- Coarse grained authorization (control access to the business application) shall be implemented and the authorization controls shall preferably be externalized from the application;
- Middle grained authorization aiming at controlling access to business functionalities shall be provided within the application and based on Role Based Access Control;
- Fine grained authorization aiming to control the accesses to the data/functionalities shall be provided by the application and based on user specific information or specific rules based algorithm.

As far as this document is concerned, only the coarse and middle-grained authorizations are relevant.

Elevated Authentication Enforcement

In some cases, a customer with an active session might be required to re-authenticate with an authentication method stronger than the one used to open the initial session. The re-authentication process is triggered upon the invocation by the user of a more sensitive business service, function, or web resource for which an *elevated* authentication method is required by the security policy. This requirement enhances user experience because elevated authentication (such as hardware token) is triggered only when needed depending on the risk involved.

Auditing and Logging

The Web-SSO should support flexible auditing and logging by producing an audit trail recording the "6W's" requirement of e-Commerce Policy:

1. Subject: **w**ho initiated the action, as identified by his/her security context and/or session ID;
2. Action: **w**hat was the action (e.g., access to a web-resource or a URL);
3. Timestamp: **w**hen did the action take place;
4. Origin: **w**here was it initiated from (e.g., IP address, range…);
5. Channel: **w**hat was the communication channel used;
6. Result: **w**hether access was successful.

The user identification should be used consistently across web-server resources and application servers so that event correlation across multiple audits trails can be effectively and efficiently used for security incidents investigation purposes.

Credential propagation

This functionality supports the N-tier architecture for web services. In this context, once an access request is authorized by the Reverse Proxy, the user credentials are passed on to higher layers for processing the web access request. The credential propagation requirement means that the Reverse Proxy maintains a security context (user identity, list of services the user is authorized for, user group membership, and other items) for the authenticated user. The Reverse Proxy should offer mechanisms for securely communicating user context to higher layers (e.g., application servers, back-end servers).

The receiving end of the propagation chain makes further authorization decisions on the basis of the received user credentials.

Network Segmentation Requirements (network placement)

The envisioned technology integration requirements include the need for placing different components of the solution in different trust zones within the network environment (typically a multi-DMZ environment). This means that the following components should be decoupled (i.e., not be packaged as a monolithic block):

- Reverse Proxy Server (front-end);
- Authentication Services Abstraction Layer (performing the actual authentication process);
- Authorization (or Policy) Server: containing roles, permissions, and resource mappings;
- Directory Server: containing user identities and other user attributes.

Operational reliability should be maintained when these components are placed in difference trust zones separated by Firewalls. Given the sensitivity of the communications between these components, the Web-SSO solution should support authentication and encryption protocols.

Security of the Web-SSO

As the Secure Reverse Proxy will be exposed to un-trusted and potentially hostile networks, it should be built on solid security capabilities: stripped-down, hardened or purpose-built operating-system.

Fail-safe mode: should any of the Web-SSO component (Reverse Proxy, Authorization/Policy Server) experience a failure (due to a software defect, a human error, or as a result of a denial of service attack) then it should block all access until normal operational status is restored.

Security Management

- Web-SSO Communication Protocols: communication between Web-SSO components (e.g., Reverse Proxy, Policy Server, Authentication Server, and Directory Server) must be based on standard protocols (not vendor-proprietary protocols) and must support mutual authentication. Using standard protocols also allows protocols inspection by Web Application Firewall in front of the Web-SSO;
- Out-of-Band Management: the technology solution should support the use of management traffic through a dedicated network interface to enable management traffic protection and the segregation of management traffic from customer traffic. The management LAN where the Reverse Proxy management client interface is located should be behind a bastion host;

- Protected Management Traffic: management traffic (between the management client on the Management LAN and the Reverse Proxy Server) should be encrypted and strongly authenticated;

- HSM Support: the private key of the Reverse Proxy should be physically protected through Hardware Security Module (HSM), a purpose-built, stripped-down, and hardened operating-system, preferably compliant with internationally recognized certification standards (e.g., FIPS-4…);

- Auditing and Logging Administrative Tasks: critical administrative tasks (such as start/stop of the server services, changes to authentication schemes, administration user profiles, permissions, etc.) should be audited and preferably exportable to separate systems for after-the-fact security monitoring and investigation.

Administration Roles

Various roles with different permissions should be supported to enforce the separation of duties principle. Security administration roles (such policy configuration, modification of critical configuration files, audit log management), should be separated from operational roles (system monitoring and housekeeping, system upgrades, backup/restore, performance management, etc.).

Administrative Accounts Protection

Administrative accounts used for various housekeeping and administration tasks should protected by strongly authenticating administrative accounts logins and protecting stored administrative credential data.

Audit logs

It should be possible to configure the Web-SSO to generate audit logs of selected critical and non-critical events related to the administration and operation of the Web-SSO (stop/start, policy change, administrative accounts usage, and other security-related events).

ATTACK USE CASES

CREDENTIAL STEALING THROUGH A ROGUE MOBILE APPLICATION

OVERVIEW

A customer could be tricked to download a rogue app from an application store app store that collects credentials, the same way as for traditional phishing attacks targeting e-commerce or e-banking sites. Not all Application Store providers apply the same security measures to prevent hackers from installing malicious applications in the application store. For example Apple App Store imposes more stringent controls than Google Android market before releasing new app to the public. Over 50 malicious apps were distributed from Google Android Market.

MODUS OPERANDI

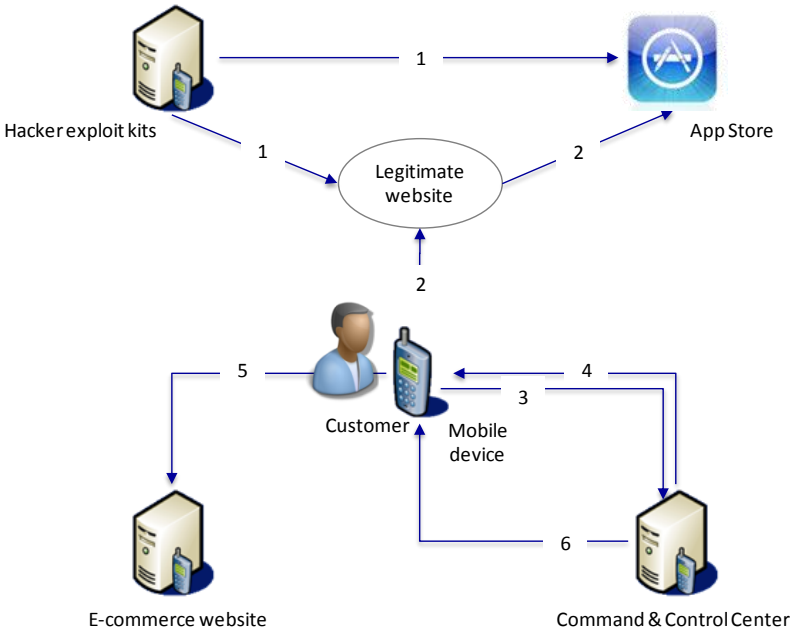


Figure 22 - Mobile attacks modus operandi

The *Modus Operandi* of this attack is described below:

- The hacker uploads to App Store or Android Market a malicious App looking like originated service;
- Customers downloads the App, thinking it is a legitimate mobile application;
- The App confirms to the Command & Control Center (CCC) that it was successfully install on victims Smartphone;
- The malicious app receives instructions from the CCC and waits for a session to be open by the customer;
- A customer open mobile and authenticates via SMID/SMP;
- The malicious app captures SMID/SMP and sends them (via SMS, Internet or any other mean) to the CCC.

ZEUS-LIKE INFECTION CUSTOMER'S SMARTPHONE

OVERVIEW

As outlined in the abuse-case scenario, customer's Smartphone could be infected by a malware that is specially crafted for attacking popular mobile applications. The malware could be downloaded as a malicious mobile app, posing as a legitimate mobile application with appealing functionalities to entice customers. Alternatively, the malware could be automatically installed as customers visit infected sites (drive-by viruses). Likewise, the infection will most probably be part of a botnet, where infected Smartphones are remotely controlled by a command center, either programmatically or interactively.

MODUS OPERANDI

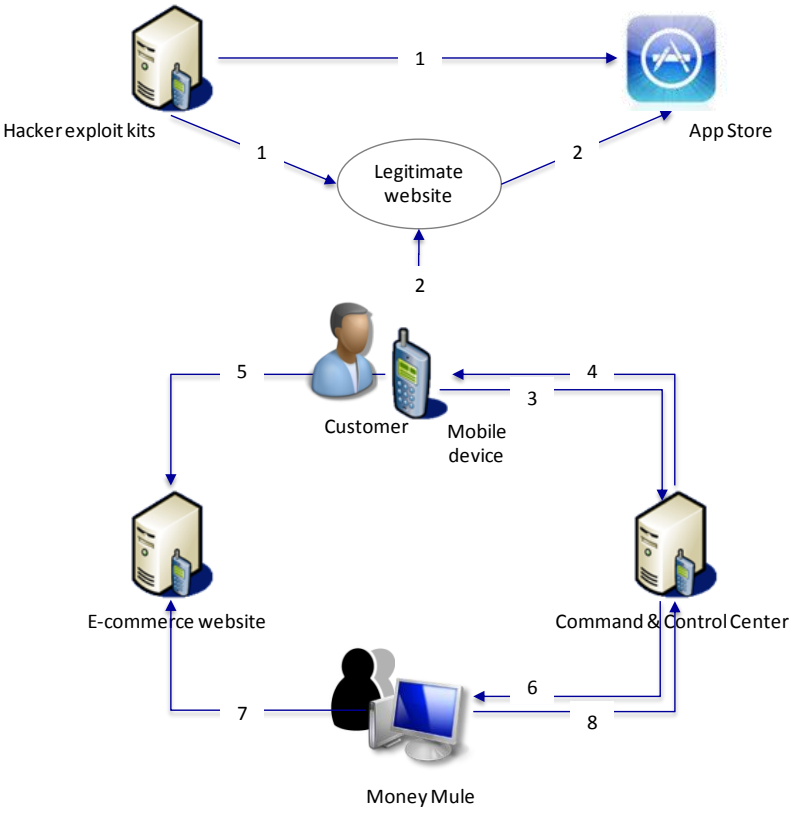
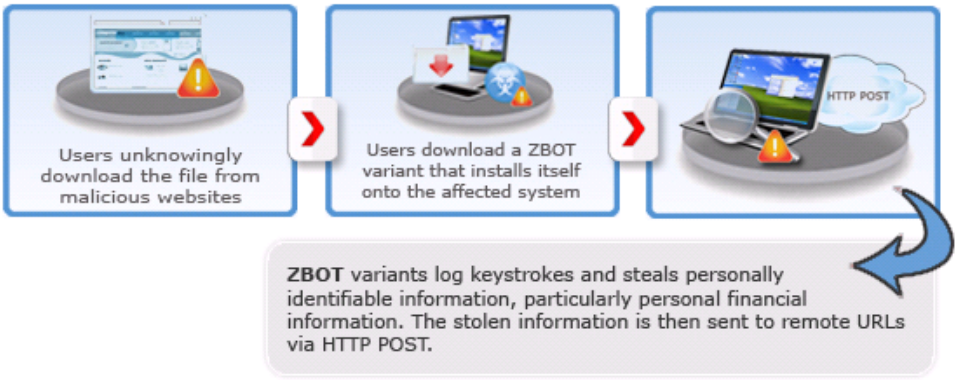


Figure 23 - Zeus-like mobile infection

The Modus Operandi of the Zeus attack against mobile would follow the overall steps described below:

- Hackers upload Zeus-like malware to legitimate websites or even to Apple App Store;
- Customers (unknowingly) downloads the malware onto his/her smartphone by visiting infected websites or by downloading a malicious app;
- The malware reports to the hackers Command & Control Center (CCC);
- Malware receives instructions from the CCC and waits for a session to be open by the customer;
- A customer open mobile and authenticates with UCR, which triggers the Malware to create fraudulent transfers;

- The CCC alerts the Money Mule(s) who stand ready to transfer money abroad;
- The Money Mule monitors his/her account and sends money abroad as soon as received;
- The Money Mule reports back to CCC that the fraud has been successfully completed.



The Top 10 Application Security Risks (OWASP[3])

This non-exhaustive list provides an overview of the major vulnerabilities likely to be present within a Web application. Amongst other things, this list builds on the vulnerabilities listed in the Guide to Building Secure Web Applications published by the OWASP Top 10 for the year 2010.

ID	Risk	Description
A1	Injection	Injection flaws, such as SQL, OS, and LDAP injection, occur when un-trusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data
A2	Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes un-trusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites
A3	Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities
A4	Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these

[3] OWASP: **O**pen **W**eb **A**pplication **S**ecurity **P**roject.

		references to access unauthorized data
A5	Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim
A6	Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date, including all code libraries used by the application
A7	Insecure Cryptographic Storage	Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes
A8	Failure to restrict URL Access	Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks each time these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway
A9	Insufficient Transport Layer Protection	Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly
A10	Invalid redirect and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use un-trusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages

REFERENCES

https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf

http://www.cgisecurity.com/lib/WhitePaper_DevelopingSecureWebApps.pdf

http://msdn.microsoft.com/en-us/library/ff648647.aspx

TABLE OF INDEX