Security of Rich Internet Applications

This document makes use of information provided by MSDN online document titled: "Improving Web Application Security: Threats and Countermeasures".

Security of Rich Internet Applications

June 2012

Table of Contents

1- Document Overview 6
Supporting Architecture 6
Rich Internet Application Life Cycle 6
Built-In Security Measures 6
Security Threats and Countermeasures 6
Recommendations 6
2- Supporting Architecture 7
Distributed Modules 7
The RIA Client 7
WWW Service Capabilities 7
Magic xpa Internet Requester 7
Magic xpa Request Broker 7
Magic xpa Enterprise Server 8
Distribution of the Modules 8
Communication Between the Distributed Modules 9
The Life Cycle of a Request 9
Client to Web Server 9
Requester to Broker 9
Requester to Engine 9
Web Server to RIA Client 9
3- RIA Life Cycle 10
RIA Task Initialization 10
The Server Response 10
Presentation Layer 10
Logic Layer 10
Data Layer 10
Ongoing Interaction 11
RIA Client to Enterprise server 11
Enterprise Server to RIA Client 11
4- Magic xpa RIA Built-In Security Measures 12
HTTPS Support 12
Trusted Vendor Certification 12
Encrypted Internal Communication 12
Scrambled Transmitted Content 12

Code Obfuscation 12
Streamed-only Data 13
Encrypted Local Cache 13
5- Network & Host-related Threats and Countermeasures 14
Network Threats 14
Information Gathering 14
Sniffing 15
Spoofing 15
Session Hijacking 16
Denial of Service 16
Host Threats 17
Viruses, Trojan Horses, and Worms 17
Footprinting 18
Password Cracking 18
Denial of Service 19
Arbitrary Code Execution 19
Unauthorized Access 20
6- Application-related Threats and Countermeasures 21
Input Validation 22
Buffer Overflows 22
Cross-site Scripting (XSS) 23
SQL Injection 23
Canonicalization 25
Authentication 26
Network Eavesdropping 26
Brute Force Attacks 26
Dictionary Attacks 26
Cookie Replay Attacks 27
Credential Theft 27
Authorization 28
Elevation of Privilege 28
Disclosure of Confidential Data 28
Data Tampering 29
Luring Attacks (Phishing) 29
Configuration Management 30
Unauthorized Access to Administration Interfaces 30
Unauthorized Access to Configuration Stores 30

Retrieval of Plaintext Configuration Secrets 31
Lack of Individual Accountability 31
Over-privileged Application and Service Accounts 31
Session Management 32
Session Hijacking 32
Session Replay 32
Man in the Middle Attacks 33
Cryptography 34
Poor Key Generation or Key Management 34
Weak or Custom Encryption 34
Checksum Spoofing 34
Parameter Manipulation 36
Query String Manipulation 36
Form Field Manipulation 36
Cookie Manipulation 37
HTTP Header Manipulation 37
Exception Management 38
Attacker Reveals Implementation Details 38
Denial of Service 38
Auditing and Logging 39
User Denies Performing an Operation 39
Attackers Exploit an Application without Leaving a Trace 39
Attackers Cover Their Tracks 39
7- Recommendations 40
Securing a Magic xpa Application 40
Secured Layer 40
Encrypted Data 40
Direct SQL 40
Error Handling 40
LDAP Facility 40
Rights Mechanism 40
Vendor Signature 40

1-
Document Overview

This document is intended to unravel the basic architecture that supports a Rich Internet application (RIA) confronted with Internet-related security issues.

Supporting Architecture

Chapter 2 outlines the architecture that supports any Magic xpa Internet application, including RIA.

Rich Internet Application Life Cycle

Chapter 3 deals with the general life cycle of a Magic xpa RIA. The chapter specifies what type of information is transmitted between the client and the server as the user interacts with the application.

Built-In Security Measures

Chapter 4 describes and highlights some of the security-related features that make Magic xpa a highly secured platform for RIA implementation.

Security Threats and Countermeasures

Chapter 5 and Chapter 6 deal with known security threats. Some security matters that are known to pose a threat for Internet applications are intrinsically avoided when you create a Magic xpa RIA. For the remaining threats, the chapter presents the recommended countermeasures.

The security threats listed in this document have been collected from various external sources. Most of the information in Chapters 5 and 6 is quoted from a Microsoft online document on MSDN titled Improving Web Application Security: Threats and Countermeasures.

The Magic xpa perspective on every security threat is provided where relevant.

Recommendations

The Magic xpa platform intrinsically handles and covers most of the security issues leaving the developer to focus on the actual business-related functionality of the application. However, there are some extra features and capabilities that tighten the application security level. Chapter 7 lists these features with recommended implementation tips.

2-
Supporting Architecture

The Magic xpa Enterprise Server deployment environment is constructed using the Distributed Application Architecture modules provided by Magic xpa. The Magic xpa Enterprise Server provides your front end with automatic and optimized context management designed for handling large-scale concurrent users.

Distributed Modules

The basic modules required to construct a Magic xpa RIA are:

The RIA Client

Magic xpa provides a browser-free graphical client module. This client module is a generic module that can reflect the UI and client-side logic of any Magic xpa RIA. The Magic xpa RIA client is a self-installed module that is automatically installed on the client side.

Desktop deployment of the RIA client is done using Microsoft's ClickOnce deployment technology to install a copy of the client signed by the application vendor. ClickOnce installs the RIA Client files in the following folder:

- On Windows Vista and Windows 7:
  c:\users\username\AppData\Local\Apps\2.0\obfuscatedfoldername\obfuscatedfoldername\appname
- On Windows XP:
  C:\Documents and Settings\username\LocalSettings\Apps\2.0\obfuscatedfoldername\obfuscatedfoldername\appname

For each of the RIA client files, ClickOnce generates an additional two files: a .manifest file and a .cdf-ms file. These files contain information about the respective dll, of which the auto updater of the ClickOnce will make use of.

Mobile deployment of the RIA mobile client is done by installing the client module signed by the application vendor.

WWW Service Capabilities

A Web server is required to receive an HTTP request from remote RIA clients. Using the Magic xpa Internet Requester, the Web server forwards the HTTP request to the Magic xpa Enterprise Server.

Magic xpa Internet Requester

Magic xpa provides an Internet Requester module that extends the Web server. When a RIA client makes a request to the Requester, the module passes the request, with its accompanying data, to the Magic xpa Enterprise Server. The Internet Requester module can locate an available Enterprise Server using the pool of server engines maintained by the Magic xpa Request Broker.

Magic xpa Request Broker

The Magic xpa Request Broker handles all the available Magic xpa Enterprise Server engines and directs each request from an Internet Requester to the available Enterprise Server engine. The Request Broker provides load balancing and recovery capabilities to handle any fail over.

Magic xpa Enterprise Server

The Magic xpa Enterprise Server lies at the heart of Rich Internet Application deployment. It is the central deployment unit, which handles each request and executes the server-side application logic for each type of request it receives. The Magic xpa Enterprise Server needs to know the location of the Request Broker, and connect to it, thus making itself available to the Internet Requester.

The Magic xpa Enterprise Server engine is designed to handle multiple requests using a single engine process. This is achieved using the server's multi-threading capabilities.

Distribution of the Modules

The above modules can be installed on the same machine or distributed among different machines, even on machines using different operating systems.

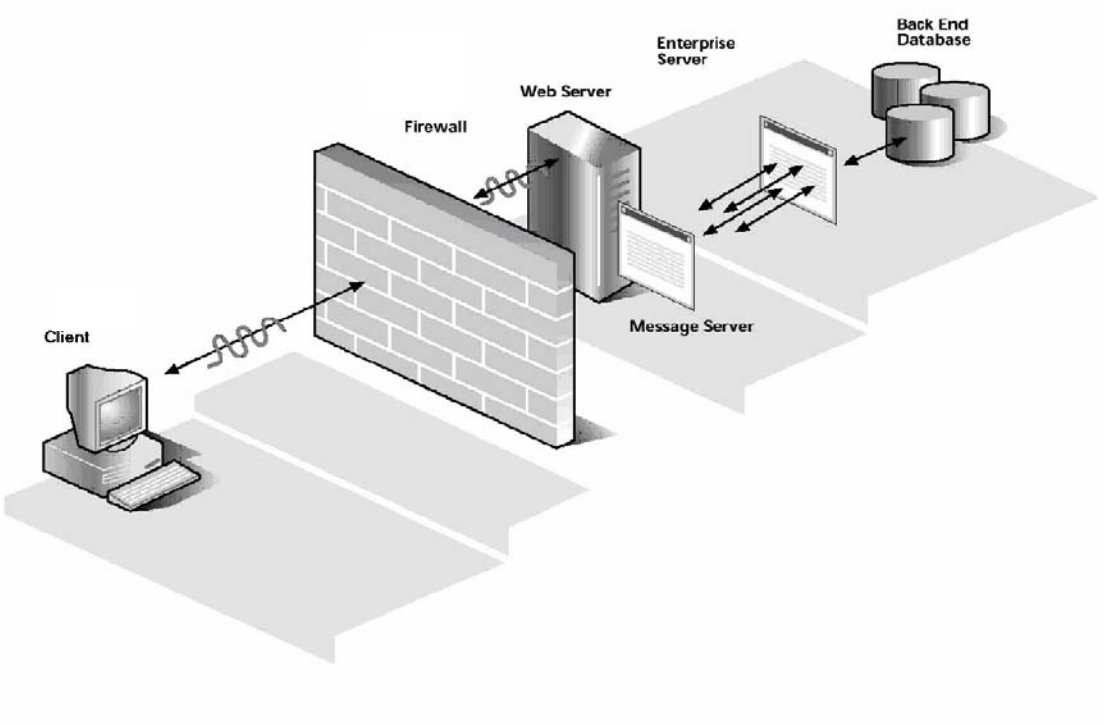

Figure 1: This image illustrates how a RIA client communicates with the enterprise server, and how the back end interacts with the distributed Magic xpa modules.

Communication Between the Distributed Modules

The Life Cycle of a Request

Every request issued by the RIA client goes through a defined procedure from the time it is sent through to the time a response is accepted. The life cycle is the same for each request, regardless of its context of use or content.

Client to Web Server

Once a request is triggered, it is passed to the Web server over HTTP or secured HTTP (HTTPS). Once the request reaches the Web server, the Requester that is executed within the Web server gets the request data.

Requester to Broker

The Requester queries the Broker for an available Magic xpa Server engine. The Broker informs the Requester of the host and port of the available server engine. This communication is done over TCP/IP.

Requester to Engine

Once the Requester gets the details of the available server engine, it passes the request details to the server engine. The server engine processes the request and sends the result back to the Requester. This communication is done over TCP/IP.

Web Server to RIA Client

Once the Requester receives the result for the processed request, the response is transmitted back to the RIA client via the Web server over the HTTP or HTTPS layer.

3-
RIA Life Cycle

Every Rich Internet application has its entry point that is usually represented by the first program that is executed to begin the application flow.

RIA Task Initialization

Upon the initial request to execute a RIA program, the server engine opens a context, and a unique context ID is generated. A short response is immediately returned to the client, informing it of the created context.

As the context response completes its short process on the client, a second request is made back to the Enterprise Server, using the context ID generated for this context, to authenticate the client (anonymously or by providing credentials, according to the *Input Password* environment setting).

The third and last request of the initialization sequence loads and executes the RIA program.

The Server Response

On initializing a RIA program, the response of the server is an XML-based response including the following elements:

Presentation Layer

Part of the response that the client receives when calling a RIA program is the task's presentation layer. This XML portion page contains the exact design defined for the program in the Studio.

Logic Layer

Another part of the response that the client receives when calling a RIA program, is the task's logic layer. This XML portion page contains all of the client-side logic that is derived from the RIA program definition together with instructions on when to turn to the server engine when a server-side intervention is required. The presentation and logic parts of the XML-based response, being static throughout the duration of the application, are generated by the server and kept hidden from the Web server and the outside world. The logic and presentation XML files are streamed directly to the client whenever they client reaches specific parts of the application. The client then caches the transmitted files. Whenever the application definition is modified, the modified XML parts will be retransmitted to the client on demand.

Data Layer

An additional XML portion in the server response defines the initial data that is required for the task, according to the view definition of the task.

Ongoing Interaction

When the end user interacts with the RIA program, the interaction may require server-side intervention, such as committing a transaction or performing server-side logic.

RIA Client to Enterprise server

For each server-side activity, the RIA client issues an HTTP request that is directed to the Enterprise Server in the flow described in Chapter 2. This type of request posts the modified data and the requested server-side logic in XML format.

Enterprise Server to RIA Client

The Enterprise Server processes the received XML and returns an HTTP response back to the client. The HTTP response includes new data for the client to display and further instructions for client-side logic.

4- Magic xpa RIA Built-In Security Measures

The Magic xpa platform employs various built-in means to achieve tighter security with minimal development and deployment effort:

- HTTPS Support
- Trusted Vendor Certification
- Encrypted Internal Communication
- Scrambled Transmitted Content
- Code Obfuscation
- Streamed-only data
- Encrypted Local Cache

HTTPS Support

The Magic xpa platform transparently supports the deployment of Magic xpa RIA over secured HTTP layer (HTTPS). Aside from the required Web Server configuration there is no need to handle the matter explicitly in the application.

Trusted Vendor Certification

Desktop client RIA deployment is done using Microsoft ClickOnce technology. For every Rich Internet application a specific manifest is produced and signed by a certificate that is acquired by the application provider. When ClickOnce deploys or updates the application, it computes the hash of each file as it is downloaded from the server and compares the hash to the one embedded in the downloaded application manifest. Since the application manifest is signed and cannot be tampered with to change the hash values for application files, there is no way to tamper with any of the application files, because ClickOnce will refuse to launch the application if the application file hashes do not match after they have been downloaded.

Encrypted Internal Communication

The communication between the Magic requester, Magic broker and Magic xpa engine is encrypted using Asymmetrical and Symmetrical encryption. No hard-coded keys are used for the encryption so it is possible to define the Symmetric encryption mechanism and the key length.

Asymmetrical encryption is only being used for the handshake (RSA with key size of 1024 bits) and from that point on a Symmetrical encryption is being used. The algorithm and key length can be set in the environment file of the Magic xpa platform (i.e. mgreq.ini) by the developer based on the Magic xpa supported cipher algorithms and key lengths. Each opened connection between any of the Magic xpa modules has its own Symmetrical key, which was generated on the fly in the handshake process (under the Asymmetrical encryption).

Scrambled Transmitted Content

In addition to the secured HTTP layer, Magic xpa adds another level to the security of the actual content that is transmitted to and from the RIA client and Magic xpa server. The data between the RIA client and the Magic xpa server engine is scrambled. The scrambling is done using the Magic xpa proprietary scrambling algorithm.

Code Obfuscation

The code of the RIA client module is obfuscated to prevent any attempt of revealing the inside works of the client especially when it comes to the communication, authentication and encryption layers of the client.

Streamed-only Data

All the information pertaining to the application definition and the application data is streamed directly to the client. The information of this kind is not kept in a shared folder on the Web server and it is available only in the Magic xpa server process, which can be far behind the Web server.

Encrypted Local Cache

For improved performance, the Magic xpa RIA client caches the application meta-data that is transmitted to it from the server. The client also caches some of the data views that are used in a given session. To prevent any attempt to reveal or tamper with the locally cached files, the cached information is encrypted under DES with a 56 bits key.

5-
Network & Host-related Threats and Countermeasures

Internet applications serve as a target for malicious threats. These threats are categorized into three main categories: Network threats, Host threats, and Application threats. This chapter deals with the first two categories.

Note: Most of the information in this chapter is taken from the Microsoft online document on MSDN titled Improving Web Application Security: Threats and Countermeasures.

Network Threats

The network infrastructure that serves the Internet application consists of the following primary components: "routers, firewalls, and switches. They act as the gatekeepers guarding your servers and applications from attacks and intrusions. An attacker may exploit poorly configured network devices.

Top network level threats include:

- Information gathering
- Sniffing
- Spoofing
- Session hijacking
- Denial of service

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."

Magic xpa Perspective

Assuming there is a secure mechanism between the client and the Web server, and between the Magic xpa components (the Requester, the Broker and the Enterprise Server), this threat is intercepted by the secure mechanism.

The secure mechanism can be either SSL and/or external encryption/decryption mechanisms implemented on the client side and on the server side.

Sniffing

"*Sniffing* or *eavesdropping* 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."

Magic xpa Perspective

The Magic xpa RIA client supports a secured connection between the client and the server. This prevents the ability to monitor the actual data being transmitted from the client to the Web server and back. In most cases, the Web server, the Requester, the Magic Broker, and the Magic xpa Server engine are placed within and behind a demilitarized zone in a subnet that is not accessible by unauthorized parties.

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 traced 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."

Magic xpa Perspective

In addition to the network level filtering, the Magic xpa RIA enables the developer to design the application in a way that additional client-side identification means are added for enhanced security measures. The ability of the RIA client module to fully interact with the client machine lets the developer extract client imprinted information to be matched and validated by the Magic xpa server. The *ClientGetUniqueMachineID* built-in function of Magic xpa is one way to authorize specific machines to access the application or to access more sensitive parts of the application.

Note that this ID can be changed upon the next restart/connection to the Internet.

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."

Magic xpa Perspective

Utilizing the Magic xpa ability to communicate over a secured channel (HTTPS) together with the Magic xpa specific syntax (unlike regular HTML-based submission and responses) makes Session Hijacking an impossible challenge.

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:

- 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.
- Use a reverse proxy to allow enhanced web server obfuscation."

Magic xpa Perspective

This threat is not an application or deployment tool related threat; hence, it should be addressed solely by network configurations.

Host Threats

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

Top host level threats include:

- Viruses, Trojan horses, and worms
- Footprinting
- Profiling
- Password cracking
- Denial of service
- Arbitrary code execution
- Unauthorized access

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. The Code Red Worm is one of the most notorious to afflict IIS; it relies upon a buffer overflow vulnerability in a particular ISAPI filter.

Although these three threats are actually attacks, together they pose a significant threat to Internet 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 against viruses, Trojan horses, and worms include:

- Block all unnecessary ports at the firewall and host.
- Disable unused functionality including protocols and services.
- Harden weak, default configuration settings."

Magic xpa Perspective

In addition to utilizing external security utilities, such as malware detectors, you can configure the Port usage of Magic xpa using the environment settings of the deployment engine.

Footprinting

"Examples of footprinting are port scans, ping sweeps, and NetBIOS enumeration 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 your Web server to prevent information disclosure through banner grabbing.
- Use an IDS that can be configured to pick up footprinting patterns and reject suspicious traffic."

Magic xpa Perspective

This threat is not an application or deployment tool related threat; hence, it should be addressed solely by server machine configurations.

Password Cracking

"If the attacker cannot establish an anonymous connection with the server, the attacker will try to establish an authenticated connection. For this, the attacker must know a valid user name 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 Internet applications.
- Audit failed logins for patterns of password hacking attempts."

Magic xpa Perspective

Magic xpa can utilize any standard user authentication facilities, such as Active Directory or LDAP. However, this threat is not an application or deployment-tool related threat; hence, it should be addressed solely by proper user account policies.

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.
- 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 an IDS that can detect potential denial of service attacks."

Magic xpa Perspective

Magic xpa request filtering enables you to have more control over the service level allocated for each type of request. Moreover, the scalable nature of the Magic xpa platform enables it to overcome potential irregular increases in service demand. Nevertheless, most of these threats should be handled on the network and server machine configuration.

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 the Web server to reject URLs with "../" to prevent path traversal.
- Lock down system commands and utilities with restricted ACLs."

Magic xpa Perspective

Magic xpa RIA requires only a common HTTP port to be made available from a client machine. In addition, the above countermeasures should be followed as this threat is not an application or deployment tool related threat; hence, it should be addressed solely by host configurations.

Unauthorized Access

"Inadequate access controls could allow an unauthorized user to access restricted information or perform restricted operations. Common vulnerabilities include weak Web access controls, including Web permissions and weak NTFS permissions.

Countermeasures to help prevent unauthorized access include:

- Configure secure Web permissions.
- Lock down files and folders with restricted NTFS permissions."

Magic xpa Perspective

This threat is not an application or deployment tool related threat; hence, it should be addressed solely by host configurations.

6-
Application-related Threats and Countermeasures

The application design in terms of authentication, encryption of data and data management at large, plays a significant role in making your Internet application a secured one.

Note: Most of the information in this chapter is taken from the Microsoft online document on MSDN titled Improving Web Application Security: Threats and Countermeasures.

“A good way to analyze application-level threats is to organize them by application vulnerability category. The various categories used in the subsequent sections of this chapter and throughout this document, together with the main threats to your application, are summarized in the following table:

Table 1: Threats by Application Vulnerability Category

Category	Threats
Input validation	Buffer overflow; Cross-site scripting; SQL injection; Canonicalization
Authentication	Network eavesdropping; Brute force attacks; Dictionary attacks; Cookie replay; Credential theft
Authorization	Elevation of privilege; Disclosure of confidential data; Data tampering
Configuration management	Unauthorized access to administration interfaces; Unauthorized access to configuration stores; Retrieval of clear text configuration data; Lack of individual accountability; Over-privileged process and service accounts
Session management	Session hijacking; Session replay; Man in the middle
Cryptography	Poor key generation or key management; Weak or custom encryption; Checksum spoofing
Parameter manipulation	Query string manipulation; Form field manipulation; Cookie manipulation; HTTP header manipulation
Exception management	Attacker reveals implementation details; Denial of service
Auditing and logging	User denies performing an operation; Attacker exploits an application without trace; Attacker covers his or her tracks

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 your application become the only source of attack. The input to your application is a way to both test your system and execute code on an attacker's behalf. Your application may be susceptible to the following:

- Buffer overflows
- Cross-site scripting
- SQL injection
- Canonicalization

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. The following code fragment illustrates a common example of buffer overflow vulnerability.

```
void SomeFunction( char *pszInput )
{
char szBuffer[10];
// Input is copied straight into the buffer when no type checking is
performed
strcpy(szBuffer, pszInput);
. . .
}
```

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 your 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."

Magic xpa Perspective

The developer of a Magic xpa application does not need to deal with overflows of primitive data types. The Magic xpa developer handles simplified data types and the Magic xpa engine pre-emptively prevents any possible buffer overflow.

Cross-site Scripting (XSS)

"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.

Example of Cross-site Scripting

To initiate the attack, the attacker must convince the user to click a carefully crafted hyperlink, for example, by embedding a link in an email sent to the user or by adding a malicious link to a newsgroup posting. The link points to a vulnerable page in your application that echoes the un-validated input back to the browser in the HTML output stream. For example, consider the following two links.

Here is a legitimate link:

www.yourwebapplication.com/logon.aspx?username=bob

Here is a malicious link:

www.yourwebapplication.com/logon.aspx?username=<script>alert('hackercode')</script>

If the application takes the query string, fails to properly validate it, and then returns it to the browser, the script code executes in the browser. The preceding example displays a harmless pop-up message. With the appropriate script, the attacker can easily extract the user's authentication cookie, post it to his or her site, and subsequently make a request to the target Web site as the authenticated user."

Magic xpa Perspective

Being completely browser free, the RIA client module allows no client-side manipulation of the code or script. There is no "View Source" option and there is no way that an end user can add additional script.

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.

Example of SQL Injection

Your application may be susceptible to SQL injection attacks when you incorporate unvalidated user input into database queries. Particularly susceptible is code that constructs dynamic SQL statements with unfiltered user input. Consider the following code:

```
SELECT * FROM Users
WHERE UserName ='" + txtuid + "'
```

"Attackers can inject SQL by terminating the intended SQL statement with the single quote character followed by a semicolon character to begin a new command, and then executing the command of their choice. Consider the following character string entered into the **txtuid** field.

```
'; DROP TABLE Customers -
```

This results in the following statement being submitted to the database for execution.

```
SELECT * FROM Users WHERE UserName=''; DROP TABLE Customers --'
```

This deletes the Customers table, assuming that the application's login has sufficient permissions in the database (another reason to use a least privileged login in the database). The double dash (--) denotes an SQL comment and is used to comment out any other characters added by the programmer, such as the trailing quote.

"Other more subtle tricks can be performed. Supplying this input to the **txtuid** field:

```
' OR 1=1 -
```

builds this command:

```
SELECT * FROM Users WHERE UserName='' OR 1=1 -
```

Because 1=1 is always true, the attacker retrieves every row of data from the Users table.

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."

Magic xpa Perspective

In Magic xpa RIA deployment, input strings are never treated as executable statements unless the programmer deliberately uses these strings as part of Direct SQL statements. Moreover, any SQL range or locate procedure that is based on alphanumeric data is passed to the database server within single quotes, thereby eliminating the possibility of 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. File names are also problematic. For example, a single file could be represented as:

```
c:\temp\somefile.dat
somefile.dat
c:\temp\subdir\..\somefile.dat
c:\ temp\ somefile.dat
..\somefile.dat
```

Ideally, your 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 file names 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."

Magic xpa Perspective

To keep relative paths on the server side so that they are related as full paths on the client side, it is recommended to utilize the Magic xpa Logical Names facility.

Authentication

"Depending on your 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. The top threats that exploit authentication vulnerabilities include:

- Network eavesdropping
- Brute force attacks
- Dictionary attacks
- Cookie replay attacks
- Credential theft

Network Eavesdropping

If authentication credentials are passed in plain text 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 the Kerberos protocol or Windows authentication.
- Make sure passwords are encrypted (if you must transmit passwords over the network) or use an encrypted communication channel, for example with SSL."

Magic xpa Perspective

The Magic xpa RIA client supports a secured connection between the client and the server. This prevents the ability to monitor the actual data being transmitted from the client to the Web server and back. In most cases, the Web server, the Requester, the Magic Broker, and the Magic xpa Server engine are placed within and behind a demilitarized zone in a subnet that is not accessible by unauthorized parties. Moreover, Magic xpa authentication credentials that are transmitted from the client to the server are encrypted and scrambled along with the context-ID, to prevent Network eavesdropping as well as Replay Attack.

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.

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 a 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.

Countermeasures to prevent dictionary attacks include:

- Use strong passwords that are complex, contain irregular 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."

Magic xpa Perspective

Magic xpa enables you to easily utilize strong authentication facilities, such as LDAP and Active Directory.

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.
- Set 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."

Magic xpa Perspective

Magic xpa RIA implementation is completely browser free and therefore does not require any cookies-related support. The session and context management is performed internally by the RIA client.

Credential Theft

"If your 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 Microsoft Active Directory® directory service or Security Accounts Manager (SAM) user store. 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 log-in information 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 log-in access, create functionality that either allows the user to choose to not save credentials, or force this functionality as a default policy."

Magic xpa Perspective

Being completely browser free, the Magic xpa RIA implementation does not produce any client-side logging or history of past sessions.

Authorization

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

Top threats that exploit authorization vulnerabilities include:

- Elevation of privilege
- Disclosure of confidential data
- Data tampering
- Luring attacks

Elevation of Privilege

When you design 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 you can use to prevent elevation of privilege is to use least privileged process, service, and user accounts."

Magic xpa Perspective

Magic xpa offers very flexible support for user roles and user groups to govern every activity in the application. Utilize the user role support functionality to enable each activity for users with privileges only.

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 Windows resources.
- Use standard encryption to store sensitive data in configuration files and databases."

Magic xpa Perspective

Magic xpa built-in support for user role implementation, data encryption, and seamless integration with strong authentication facilities, enables you to fully secure your data.

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."

Magic xpa Perspective

Magic xpa built-in support for user role implementation, data encryption, and seamless integration with strong authentication facilities, enables you to fully secure your data. Moreover, being browser free, there is no way to view the "source" of a RIA, which is possible in browser-based applications.

Luring Attacks (Phishing)

"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 .NET Framework code access security helps in this respect by authorizing calling code whenever a secure resource is accessed or a privileged operation is performed."

Magic xpa Perspective

A Magic xpa RIA client module can be generated as a signed module using an authenticated signature. Whenever the application is installed, the end user is prompted with the signature details. Only after verifying the source of the application the user proceeds to complete the automatic client module installation.

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.

Top configuration management threats include:

- Unauthorized access to administration interfaces
- Unauthorized access to configuration stores
- Retrieval of plaintext configuration secrets
- Lack of individual accountability
- Over-privileged process and service accounts

Unauthorized Access to Administration Interfaces

Administration interfaces are often provided through additional Web pages or separate Internet applications that allow administrators, operators, and content developers to manage 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, such as Machine.config and Web.config.
- 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.

Session Management

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

Top session-management threats include:

- Session hijacking
- Session replay
- Man in the middle

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 log-out 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."

Magic xpa Perspective

In addition to the optional SSL, Magic xpa employs its own scrambling techniques to reduce the chances of unveiling the actual content of the messages sent to and from the RIA client. Moreover, the Magic xpa server and client modules keep a tight trace and validity check on the session progress. An additional instance of a Magic xpa RIA client cannot hook on a session already in use by another client instance.

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."

Magic xpa Perspective

Magic xpa's RIA full context management facility removes the need to use cookies as a means of context management . As such, once a session is concluded, there are no traces left on the client side that "remember" the application flow or state. Moreover, Magic xpa authentication credentials that are transmitted from the client to the server are encrypted and scrambled along with the context ID, to prevent Network eavesdropping and Replay Attack.

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, Distributed Component Object Model (DCOM) 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, they 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."

Magic xpa Perspective

In addition to the ability to utilize the HTTP secured layer, Magic xpa also provides the means for encrypted internal communication. Every message between the distributed modules in the Magic xpa architecture is encrypted. This encryption prevents internal attackers from manipulating the content of Magic xpa RIA messages.

Cryptography

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

Top threats surrounding your application's use of cryptography include:

- Poor key generation or key management
- Weak or custom encryption
- Checksum spoofing"

Magic xpa Perspective

The three issues discussed in this section are irrelevant to a Magic xpa developer, since the Magic xpa platform automatically handles all encryptions, message validity and integrity.

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. Data Protection application programming interface (DPAPI) is an example of an encryption service provided on Windows® 2000 and later operating systems where the operating system manages the key.
- 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 using DPAPI for added security.
- 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 Safe Hash Algorithm (SHA1) and Message Digest compression algorithm (MD5) can be intercepted and changed. Consider the following base 64 encoding UTF-8 message with an appended Message Authentication Code (MAC).

```
Plaintext: Place 10 orders.
Hash: T0mUNdEQh13IO9oTcaP4FYDX6pU=
```

If an attacker intercepts the message by monitoring the network, the attacker could update the message and recompute the hash (guessing the algorithm that you used). For example, the message could be changed to:

```
Plaintext: Place 100 orders.
Hash: oEDuJpv/ZtIU7BXDDNv17EAHeAU=
```

When recipients process the message, and they run the plaintext (“Place 100 orders”) through the hashing algorithm, and then recompute the hash, the hash they calculate will be equal to whatever the attacker computed.

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 rely on the modification of the parameter data sent between the client and Internet application. This includes query strings, form fields, cookies, and HTTP headers.

Top parameter manipulation threats include:

- Query string manipulation
- Form field manipulation
- Cookie manipulation
- HTTP header manipulation

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."

Magic xpa Perspective

Since Magic xpa RIA is completely browser free, there is no way for the end user to pick up on the request format issued by the RIA client module.

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."

Magic xpa Perspective

Being completely browser and HTML free, with Magic xpa RIA there is no way to access and manipulate the application data and controls outside the permitted application flow.

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 or use an HMAC with the cookie.”

Magic xpa Perspective

The Magic xpa RIA full context management facility and browser-free implementation removes the need to use cookies as a means of context management.

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.

Top exception handling threats include:

- Attacker reveals implementation details
- Denial of service

Attacker Reveals Implementation Details

If rich exception details are 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.
- Handling 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 an Internet 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 Internet application process. This can occur if exceptions are not properly caught and handled.

Countermeasures to help prevent application-level denial of service:

- 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.

Top auditing and logging related threats include:

- User denies performing an operation
- Attackers exploit an application without leaving a trace
- Attackers cover their tracks

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.
- 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."

Magic xpa Perspective

Magic xpa provides a built-in logging mechanism for the Enterprise Server, the Broker and the Requester. You can create your own application level logs as suggested in this section.

7- Recommendations

This chapter summarizes the recommendations for secured RIA application deployment that relate to the Magic xpa modules.

Securing a Magic xpa Application

Secured Layer

Secured HTTP is the recommended protocol to use between the client and the Web server. Furthermore, you can utilize Magic xpa SSL support for behind-the-scenes communication. You can configure the Enterprise Server, Broker and Requester to intercommunicate with each other over SSL.

Encrypted Data

Utilize Magic xpa built-in support for sensitive data encryption to ensure confidentiality.

Direct SQL

If you choose to implement the Magic xpa Direct SQL feature, avoid creating an SQL statement that can be modified by user input in a way that can hinder the integrity of the statement.

Error Handling

Use the Magic xpa error handling mechanism and make the application proactive in cases of runtime errors.

LDAP Facility

Utilize the Magic xpa ability to interact with authorization facilities, such as LDAP and Microsoft Active Directory®.

Rights Mechanism

Use the Magic xpa Rights mechanism to properly grant application privileges to each identified user.

Vendor Signature

Obtain a signature to prove the authenticity of the application vendor and utilize the signature in the building procedure of the application client module.