SLU'S WORKFORCE CENTER

training solutions for the modern workforce

Application Security

SEC500

SAINT LOUIS UNIVERSITY WORKFORCE CENTER

@SLU_Workforce

SLU's Workforce Center

SLUWorkforceCenter

314-977-3226

info@workforcecenter.slu.edu

workforcecenter.slu.edu

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook-Discovering and Exploiting Security Flaws

Chapter 1

Web Application (In)security

WORKFORCE CENTER

Evolution of Web Applications

- Originally, the world wide web consisted of only web sites – information repositories containing static documents.
  - Security threats were largely due to vulnerabilities in web server software, allowing a hacker to change the content of the site or use the server's storage and bandwidth.
- Today, many web applications allow for two-way flow of information between the server and the browser.
  - To deliver this functionality, web applications normally require connectivity to internal computer systems that contain highly sensitive data.
  - Most applications are developed in-house by developers who have little understanding of the security problems in their code.

Common Web Application Functions

- Public Internet Applications
  - Shopping – Amazon
  - Social Networking – MySpace
  - Banking – Citibank
  - Auctions – eBay
  - Gambling – Betfair
  - Web logs – Blogger
  - Web Mail – Hotmail
  - Interactive Information – Wikipedia
- Intranet Applications
  - Accessing human resources services
  - Managing company resources

Benefits of Web Applications

- Commercial incentives
- Technical factors
  - HTTP (hyper-text transfer protocol), which is the core communications protocol used on the web, is lightweight and connectionless.
  - All web users already have a browser installed on their computers, so there is no need for a web application to distribute and manage separate client software.
  - By changing the web site on the server, the change will be reflected for all users the next time they load the page.
  - Browsers enable rich and satisfying user interfaces to be built.
  - Core technologies and languages used to develop web applications are relatively simple. Beginners can easily deploy a web application through the use of development tools.

Web Application Security

- Here is the answer to the frequently asked question (FAQ) of a typical web application asking if the site is secure:
  - "This site is absolutely secure. It has been designed to use 128-bit Secure Socket Layer (SSL) technology to prevent unauthorized users from viewing any of your information. You may use this site with peace of mind that your data is safe with us."
- Unfortunately, SSL does not ensure absolute security on a site, and in fact the majority of web applications are insecure due to factors that have nothing to do with SSL.

Non-SSL Security Issues

- The text authors have tested hundreds of web applications during 2006 and 2007. The following are common categories of vulnerability, including the percentage of tested sites with each vulnerability:
  - Broken authentication (67%)
    - Vulnerability within login mechanism
  - Broken access controls (78%)
    - Application fails to protect access to its data and functionality
  - SQL injection (36%)
    - Application allows crafted input to be submitted that interferes with back-end databases
  - Cross-site scripting (91%)
    - Targets other users of the application, such as performing unauthorized actions on their behalf
  - Information leakage (81%)
    - Application divulges sensitive information through defective error handling or other similar behavior

The Core Security Problem

- Users can submit anything they want in a web form, so the application must assume that all input is potentially malicious.
- The core security problem can exist in the following ways:
  - Users can interfere with any piece of data transmitted between the client and the server, so security controls implemented on the client can be circumvented.
  - Users can send requests in any sequence, so there can be no assumption of how users will interact with the application.
  - Users do not have to use a web browser to access the application.
- Users can submit crafted input to cause some unexpected event by the application
  - Changing a hidden HTML form field's value
  - Modifying a session token transmitted in an HTTP cookie
  - Removing certain parameters that are normally submitted
  - Altering input that will be processed by a back-end database

Key Factors of the Core Security Problem

- Immature Security Awareness
  - There is a less mature level of awareness of web application security issues than there is in longer-established areas such as networks and operating systems.
- In-House Development
  - Most web applications are developed by an organization's staff, which means every application is different and may contain its own unique defects.
- Deceptive Simplicity
  - A novice programmer can create a powerful web application from scratch, but there is a huge difference between producing code that is functional and producing code that is secure.
- Rapidly Evolving Threat Profile
  - New threats for web applications are conceived at a much faster rate than is now the case for older technologies.
- Resource and Time Constraints
  - The need to produce a stable and functional application, by a deadline, normally overrides less tangible security considerations.
- Overextended Technologies
  - Many of the core technologies used in web applications have been pushed far beyond the purposes for which they were originally conceived, which has led to security vulnerabilities.

The New Security Perimeter

- Web applications do not allow the network to be firewalled off completely, as was the defense before the rise of web applications.
- Inbound connections over HTTP/HTTPS must be allowed through the firewall.
- Many web servers must be connected to databases, mainframes, and financial and logistical systems for the web application to function.
- Part of the security perimeter of an organization is still embodied in firewalls, but a significant part of it is now in the organization's web applications.

Stealing Money (Then and Now)

- Before the bank deployed a web application
  - Attacker needs to find a vulnerability in a publicly reachable service
  - Exploit this to gain access to the bank's DMZ
  - Penetrate the firewall that restricts access to internal systems
  - Find the mainframe computer on the network
  - Decipher the protocol used to access it
  - Guess credentials to log in
- After the deployment of a web application
  - Attacker may be able to simply modify an account number in a hidden field of an HTML form if the site is extremely vulnerable

The Future of Web Application Security

- Understanding security threats facing web applications remains immature.
- Substantial current research is focused on developing advanced techniques for attacking more subtle manifestations of vulnerabilities.
- There is a gradual shift in attention from traditional attacks against the server side of the application to those that target other users.
  - Flaws in the server side applications are the first to be understood and addressed, but the client side is not addressed nearly as much.

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook- Discovering and Exploiting Security Flaws

Chapter 15
Exploiting Information Disclosure

Exploiting Error Messages

- Many web applications return informative error messages when unexpected events occur, such as error categories or full-blown debugging information.
- Some security-critical applications have been found to return highly verbose debugging output when a sufficiently unusual error condition is generated.
- In this chapter, we will discuss what information can be retrieved from an application during an actual attack, such as gathering user credentials, technologies in use, and the application's internal structure and functionality.

Script Error Messages

- When an error occurs in an interpreted web scripting language, the application typically returns a simple message disclosing the nature of the error and possibly the line number:

```
Microsoft VBScript runtime error 800a000d
Type mismatch: '[string: "']'
/scripts/confirmOrder.asp, line 715
```

The message above indicates that the value you have modified is probably being assigned to a numeric variable.

However, since the line number is provided, you may be able to determine whether two different malformed requests are triggering the same or different errors.

- You may also be able to determine the sequence in which different parameters are processed by submitting bad input within multiple parameters and identifying the location at which an error occurs.
- This will allow you to map out the different code paths being executed on the server.

Stack Traces

- Many unhandled errors in server side languages generated full stack traces that are returned to the browser.
  - A stack trace is a structured error message that begins with a description of the actual error, followed by a series of lines describing the state of the execution call stack when the error occurred.
- Stack traces provide a large amount of information that can be useful in an attack.
  - The precise reason for an error occurring is described.
  - The call stack makes reference to third party libraries and code components being used by the application.
  - The call stack includes the names of the proprietary code components being used to process the request.
  - The stack trace often includes line numbers that may enable you to understand the internal logic of the application.
  - The error message often includes additional information about the application and the environment in which it is running.

Informative Debug Messages

- Some applications generate custom error messages that contain a large amount of debug information.
- The following items are commonly included in verbose debug messages:
  - Values of key session variables that can be manipulated by user input
  - Hostnames and credentials for back-end components such as databases
  - File and directory names on the server
  - Information embedded within meaningful session tokens
  - Encryption keys used to protect data transmitted via the client
  - Debug information for exceptions arising in native code components
- This information can be used to determine crafted input to manipulate the application's state and control the information retrieved.

Server and Database Messages

- Informative error messages are often returned by back-end components such as a database, mail server, or SOAP server.
- If a completely unhandled error occurs, the application will typically respond with an HTTP 500 status code.
  - However, the application may handle the error gracefully and return a customized message to the user, sometimes including error information generated by the back-end component.
- Database error messages often contain information that you can use to advance an attack, such as:

```
Failed to retrieve row with statement –
SELECT object_data FROM deftr.tblobject
WHERE object_id='FDJE00012' AND
project_id='FOO' and 1=2
```

Using Public Information

- You should frequently expect to encounter unusual messages that you have not seen before and that may not immediately indicate the nature of the error that the application experienced.
  - You can often obtain further information about the meaning of the message from various public sources.
- Many applications employ third-party components to perform specific common tasks, and any error messages that are generated by these components are likely to have arisen in other applications and been discussed elsewhere.
- You can review publicly available source code for third-party components, and you may be able to determine what processing is being performed on your input and how you may be able to manipulate the application to exploit a vulnerability.

Gathering Published Information

- Some web applications give away sensitive data by publishing it directly for various reasons.
  - By design as part of the application's core functionality
  - As an unintended side effect of another function
  - Through debugging functionality that remains present in the live application
  - Because of some vulnerability, such as broken access controls
- Examples of potentially sensitive information that applications often publish include:
  - Lists of valid usernames, account numbers, and document IDs
  - User profile details including user roles and privileges
  - The current user's masked password
  - Log files
  - Application details in client-side HTML source, such as commented links or form fields and comments about bugs

Using Inference

- An application may not divulge any data to you directly, but may behave in ways that enable you to reliably infer information.
  - A registration function that enables you to enumerate registered usernames based on an error message when an existing username is chosen
  - A search engine that allows you to infer the contents of indexed documents that you are not authorized to view
  - A blind SQL injection vulnerability that allows you to extract information one bit at a time
- When different operations take different lengths of time to perform, information may also be disclosed to a user.
  - If lazy loading is used, data that has been recently accessed will be retrieved quickly from a server's cached copy.
  - The amount of processing an application performs on a request may depend on whether a submitted item of data is valid.

Preventing Information Leakage

- Use Generic Error Messages.
  - The application should never return verbose error messages or debug information to the user's browser.
  - A generic error message should be returned regardless of the unexpected event that occurred.
  - Server side log files should contain the detailed information, if necessary.
- Protect Sensitive Information.
  - The application should not publish information that may be of use to an attacker, including usernames, log entries, or user profile details.
  - If this information is needed for certain users, it should be protected by effective access controls.
- Minimize Client-Side Information Leakage.
  - Where possible, service banners should be removed or modified to minimize the disclosure of specific software versions.
  - All comments should be removed from client-side code that is deployed to the live production environment.

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook- Discovering and Exploiting Security Flaws

Chapter 2
Core Defense Mechanisms

Core Defense Mechanisms

- The following are defense mechanisms employed by web applications and will be discussed in more detail in this chapter.
  - Handling user access to the application's data and functionality, to prevent users from gaining unauthorized access
  - Handling user input to the application's functions, to prevent malformed input from causing undesirable behavior
  - Handling attackers, to ensure that the application behaves appropriately when being directly targeted, taking suitable defensive and offensive measures to frustrate the attacker
  - Managing the application itself, by enabling administrators to monitor its activities and configure its functionality
- Understanding these mechanisms thoroughly is the main prerequisite to being able to attack applications effectively.

Handling User Access

- There are often many different types of users of a web site:
  - Anonymous users
  - Ordinary authenticated users
  - Administrative users
- The access each type of user has is based on three components:
  - Authentication
  - Session management
  - Access control
- A defect in any one of the above components may enable an attacker to gain unrestricted access to the application's functionality and data.

User Access Security Components

- Authentication
  - Establishing that a user is in fact who he claims to be
  - Most applications use a username and password.
  - Attackers can identify other users' usernames, guess their passwords, or bypass the login function altogether by exploiting defects in its logic.
- Session Management
  - Web application issues an authenticated user a token that identifies the session because the data in the session is stored on the server.
  - Attackers attempt to compromise the tokens issued to other users by guessing the tokens issued to other users or capturing other users' tokens.
- Access Control
  - Authenticated users may only be able to access specific areas of a site, such as only being able to read their own email after logging in successfully.
  - Attackers can gain unauthorized access to data and functionality by exploiting programmers who have made flawed assumptions about how users will interact with the application.

Handling User Input

- Fundamental security problem – all user input is untrusted, but how do we handle user input?
- Approaches to Input Handling
  - "Reject Known Bad"
    - Match literal strings that are known to be used in attacks.
  - "Accept Known Good"
    - Match literal strings that are known to be only benign input.
  - Sanitization
    - Remove characters that could potentially be malicious but accept everything else.
  - Safe Data Handling
    - Instead of only validating the input, ensure the processing that is performed is inherently safe, such as by parameterizing queries for database access (which prevents SQL injection).
  - Semantic Checks
    - The data submitted is not malformed, but just malicious, such as a user changing the bank account number in a hidden form field to try to access another user's account.

Boundary Validation

- Instead of only validating input on the client side or on the server side, validate input within each individual component or functional unit of the server-side application.
- Example boundary validation application
  - The application receives the user's login details and the form handler validates that each item of input contains only permitted characters.
  - The application performs a SQL query to verify the user's credentials, and any characters that may be used to attack the database are escaped before the query is constructed.
  - If the login succeeds, the application passes certain data to a SOAP service, and any XML metacharacters are suitably encoded.
  - The application displays the user's account information back to the user's browser, HTML-encoding any user-supplied data that is embedded in the return page to prevent cross-site scripting attacks.

Handling Attackers

- If security is remotely important to an application, programmers must work on the assumption that it will be directly targeted by dedicated and skilled attackers.
- To handle attackers, there are four key tasks
  - Handling errors
  - Maintaining audit logs
  - Alerting administrators
  - Reacting to attacks

Handling Attackers (cont.)

- Handling Errors
  - It is inevitable that some unanticipated errors will occur in an application because it is very difficult to anticipate every possible way in which a malicious user may interact with the application.
  - The application should handle unexpected errors in a graceful manner and either recover from them or present a suitable error message to the user.
    - Try/catch blocks in languages provide good error handling.
- Maintaining Audit Logs
  - Audit logs are of value when investigating intrusion attempts against an application.
  - Hopefully the application's owners can understand what has taken place, which vulnerabilities were exploited, whether the attacker gained unauthorized access to data, and evidence as to the intruder's identity.
  - The following events should always be logged
    - Authentication of users and password changes
    - Key transactions, like credit card payments and funds transfers
    - Access attempts that are blocked by access control mechanisms
    - Any requests containing known attack strings that indicate malicious intentions

Handling Attackers (cont.)

- Alerting Administrators
  - Instead of investigating an attack off-line, administrators may want to take immediate action in real-time, such as by blocking the IP address or user account being used by an attacker.
  - Anomalous events monitored by alerting mechanisms include
    - Usage anomalies, such as large numbers of requests being received from a single IP address, indicating a scripted attack
    - Business anomalies, such as an unusual number of funds transfers being made to or from a single account
    - Requests containing known attack strings
    - Requests where data that is hidden from ordinary users has been modified
- Reacting to Attacks
  - Some applications take automatic reactive measures to frustrate the activities of an attacker by slowing down the response to an attacker's requests or terminating an attacker's session.
  - This will buy additional time for administrators to monitor the situation and take more drastic action if desired.

Managing the Application

- Administrators need to be able to manage user accounts and roles, access monitoring and audit functions, perform diagnostic tasks, and configure aspects of the application's functionality.
- The administrative functions are a primary attraction for an attacker for the following reasons:
  - Weaknesses in the authentication mechanism may enable an attacker to gain administrative access.
  - Many applications do not implement effective access control of some of their administrative functions.
  - Administrative functionality often involves displaying data that originated from ordinary users.
  - Administrative functionality is often subjected to less rigorous security testing because its users are deemed to be trusted or because penetration testers are given access to only low-privileged accounts.

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Hacker's Handbook- Discovering and Exploiting Security Flaws

Chapter 3

Web Application Technologies

HTTP

- HTTP – Hypertext Transfer Protocol
  - Core communications protocol used to access the World Wide Web and is used by all of today's web applications
  - Originally developed for retrieving static text-based resources but has been extended to support more advanced applications
  - HTTP is a stateless protocol that uses a request/response transaction, operating over TCP connection.
  - The request and response messages may use different TCP connections.

HTTP Methods

- The HTTP method tells the web application what the client is attempting to do.
- There are six HTTP methods:
  - GET
    - Designed for retrieval of resources
    - Used to send parameters to the requested resource in the URL query string
  - POST
    - Designed for performing actions
    - Request parameters can be sent both in the URL query string and in the body of the message.
  - HEAD
    - Functions similar to GET, but the server should only return the header of the message and not the message body in its response
  - TRACE
    - Designed for diagnostic purposes
    - The server should return the exact contents of the request message, in the response body, that it received.
  - OPTIONS
    - Asks the server to report the HTTP methods that are available for a particular resource
  - PUT
    - Attempts to upload the specified resource to the server, using the content contained in the body of the request
    - Attackers may upload an arbitrary script and execute it on the server if this method is enabled.

HTTP General Headers

- An HTTP header contains information that is used by applications rather than specifically displayed to a user.
- General Headers – likely on most HTTP messages
  - Connection – informs the other end of the communication whether it should close the TCP connection after the HTTP transmission has completed
  - Content-Encoding – specifies the type of encoding being used for the message body, such as `gzip`
  - Content-Length – specifies the length of the message body in bytes
  - Content-Type – specifies the type of content contained in the message body, such as `text/html` for HTML documents
  - Transfer-Encoding – specifies any encoding that was performed on the message body to facilitate its transfer over HTTP, such as chunked encoding when used

HTTP Request Headers

- Request Headers – likely on messages with REQUEST method
  - First line contains three items separated by spaces – the HTTP method, the requested URL, and the HTTP version
  - The rest of the lines are different HTTP headers, such as the following:
    - Accept – tells the server what kinds of content the client is willing to accept, such as image types and office document formats
    - Accept-Encoding – tells the server what kinds of content encoding the client is willing to accept
    - Authorization – submits credentials to the server for one of the built-in HTTP authentication types
    - Cookie – submits cookies to the server which were previously issued by it
    - Host – specifies the hostname that appeared in the full URL being requested
    - If-Modified-Since – specifies the time at which the browser last received the requested resource. If the resource has not changed, the server may instruct the client to use its cached copy.
    - If-None-Match – specifies an entity tag, which is an identifier denoting the contents of the message body
    - Referer – specifies the URL from which the current request originated
    - User-Agent – provides information about the browser or other client software that generated the request

Example

```
GET /books/search.asp?q=wahh HTTP/1.1
Accept: image/gif, image/xxbitmap, application/msword, */*
Referer: http://wahh-app.com/books/default.asp
Accept-Language: en-gb,en-us;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: wahh-app.com
Cookie: lang=en; JSESSIONID=0000tI8rk7joMx44S2Uu85nSWc_:vsnlc502
```

HTTP Response Headers

- Response Headers – likely on messages with RESPONSE method
  - First lines contains three items separated by spaces – HTTP version, HTTP status code, textual "reason phrase" describing the status of the response
  - The rest of the lines are different HTTP headers, such as the following
    - Cache-Control – pass caching directives to the client, such as no-cache
    - ETag – specifies an entity tag, which is used by clients to notify the server which version of the resource it has cached
    - Expires – instructs client how long the contents of the message body are valid
    - Location – specifies the target of the redirect
    - Pragma – passes caching directives to the client, such as no-cache
    - Server – provides information about the web server software being used
    - Set-Cookie – issues cookies to the client that it will submit back to the server in subsequent requests
    - WWW-Authenticate – used in responses with a 401 status code to provide details of the type of authentication supported by the server

Example

```
HTTP/1.1 200 OK
Date: Sat, 19 May 2007 13:49:37 GMT
Server: IBM_HTTP_SERVER/1.3.26.2 Apache/1.3.26 (Unix)
Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Language: en-US
Content-Length: 24246
```


HTTP Status Codes

- HTTP response messages must contain a status code in its first line indicating the result of the request.
- HTTP status codes fall into five groups, according to the first digit of the code
  - 1xx – Informational
  - 2xx – Successful request
  - 3xx – Redirection
  - 4xx – Client error
  - 5xx – Server error
- Here are a few popular HTTP status codes. A complete list can be found on the W3C site – http://www.w3.org by searching for "HTTP Status Codes"
  - 100 Continue
    - The request headers were received and that the client should continue sending the body
  - 200 OK
    - Request was successful and the response body contains the result
  - 304 Not Modified
    - Instructs the browser to use its cached copy of the requested resource
  - 401 Unauthorized
    - Server requires HTTP authentication before the request will be granted
  - 404 Not Found
    - Indicates that the requested resource does not exist
  - 500 Internal Server Error
    - Indicates the server encountered an error fulfilling the request

Server-Side Functionality

- There are three main ways HTTP requests can be used to send parameters to the application:
  - URL query string
  - HTTP cookies
  - Body of requests using POST method
- Web applications have many different technologies for delivering functionality based on these sources of input:
  - Scripting languages
    - PHP, VBScript, Perl
  - Web application platforms
    - ASP.NET, Java
  - Web servers
    - Apache, IIS, Netscape Enterprise
  - Databases
    - MS-SQL, Oracle, MySQL
  - Other back-end components
    - File systems, SOAP-based web services, directory services

Client-Side Functionality

- All web applications are accessed via a web browser and share a common core of technologies.
  - HTML
    - A tag-based language that is used to describe the structure of documents that are rendered within the browser
  - Hyperlinks
    - Allow communication from client to server and frequently contain request parameters
  - Forms
    - The usual mechanism for allowing users to enter arbitrary input via the browser
  - JavaScript
    - Allows processing of data on the client side to improve the application's performance and to enhance usability because the user interface can be dynamically updated in response to user actions
  - Thick Client Components
    - Java applets, ActiveX controls, Macromedia Flash movies

State and Sessions

- Often times, applications need to track the state of each user's interaction with the application across multiple requests.
- Data to uniquely identify the user across requests is typically stored in a server-side structure called a session.
- Applications can store this data on the client in a cookie, but any data transmitted via the client component may be modified by the user.
- HTTP is stateless, so many applications need a means of re-identifying individual users across multiple requests.

Encoding Schemes

- URL Encoding
  - Used to encode any problematic characters within the extended ASCII character set so that they can be safely transported over HTTP
- Unicode Encoding
  - Designed to support all of the writing systems used in the world
- HTML Encoding
  - Used to represent problematic characters so that they can be safely incorporated into an HTML document
- Base64 Encoding
  - Allows binary data to be safely represented using only printable ASCII characters
- Hex Encoding
  - Used when transmitting binary data, using ASCII characters to represent the hexadecimal block

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook-Discovering and Exploiting Security Flaws

Chapter 4

Mapping the Application

WORKFORCE CENTER

Enumerating Content

- In a typical application, the majority of the content and functionality can be identified via manual browsing.
- Web spidering
  - Various web spidering tools work by requesting a web page, parsing it for links to other content, and then requesting these, continuing recursively until no new content is discovered
  - Paros, Burp Spider, WebScarab
  - Limitations to full automated content enumeration
    - Unusual navigation mechanism (complicated JavaScript) are often not handled proplery by these tools.
    - Multistage functionality often implements fine-grained input validation checks which do not accept the values that may be submitted by an automated tool.
    - Automated spiders often use URLS as identifiers of unique content, but many applications use forms-based navigation in which the same URL may return very different content and functions.
    - Some applications place volatile data within URLs that do not actually identify resources or functions, which may cause the spider to run indefinitely.

Discovering Hidden Content

- Many applications contain content and functionality which is not directly linked from the main visible content, such as functionality for testing or debugging purposes that was not removed.
- Applications may have different functionality for different categories of users.
- To find this hidden content, there are a few methods:
  - Brute-force technique
    - Attempt to access common pages, such as account.php, account.php, admin.php, agent.php, etc.
  - Inference from published content
    - By inferring from the resources already identified within the application, it is possible to fine-tune your automated enumerations exercise to increase the likelihood of discovering further hidden content.
  - Use of public information
    - There may be content and functionality that is not presently linked from its main content, but has been linked in the past.
  - Leveraging the web server
    - Vulnerabilities may exist at the web server layer that enable you to discover content and functionality that is not linked within the web application itself, such as listing the contents of a directory or obtaining the raw source for dynamic server-executable pages.

Application Pages vs Functional Paths

- Pre-application days of the World Wide Web had web servers function as repositories of static information, with URLs behaving effectively as filenames.
- Authors would simply create a bunch of files and drop them in a specific directory accessible by the web server.
- Although the evolution of web application has fundamentally changed, the picture is still applicable to the majority of web application content and functionality.
- Applications can be identified using a request parameter rather than a URL, and the URL is the same for each request, which is known as a functional path.

Analyzing the Application

- Analyzing the application's functionality in order to identify key attack surfaces allows a person to probe the application for exploitable vulnerabilities.
- Key areas to investigate are:
  - Core functionality of the application
  - Peripheral behavior of the application, including off-site links, error messages, logging functions, redirects, etc.
  - Core security mechanisms and how they function, including management of session state, access controls, and authentication mechanisms
  - Different locations at which user-supplied input is processed by the application
  - Technologies employed on the client side
  - Technologies employed on the server side
  - Other details that may be gleaned about the internal structure and functionality of the server-side application

Analyzing the Application (cont.)

- The majority of ways in which the application captures user input for server-side processing are:
  - Every URL string up to the query string marker
  - Every parameter submitted within the URL query string
  - Every parameter submitted within the body of a POST request
  - Every cookie
  - Every other HTTP header that in rare cases may be processed by the application
- Further, the application could get input from:
  - A web mail application which processes and renders email messages received via SMTP
  - A publishing application that contains a function to retrieve content via HTTP from another server
  - An intrusion detection application that gathers data using a network sniffer and presents this using a web application interface

Identifying Server-Side Technologies

- Banner Grabbing
  - Many web servers disclose fine-grained version information, both about the web server software itself and about other components that have been installed, such as in the HTTP Server header.
- HTTP Fingerprinting
  - Even if the web server masks the HTTP Server header, it is usually possible to determine information based on the web server's behavior since many web servers deviate from or extend the HTTP specification in various different ways.
  - Httprint is a handy tool that performs a number of tests in an attempt to fingerprint a web server's software.
- File Extensions
  - File extensions often disclose the platform or programming language used to implement the relevant functionality.
- Directory Names
  - Subdirectory names can indicate the presence of an associated technology.
- Session Tokens
  - Session tokens often default with names that provide information about the technology in use.
- Third-party Code Components
  - Many web applications incorporate third-party code components to implement common functionality such as shopping carts, login mechanisms, and message boards.

Identifying Server-Side Functionality

- Dissecting requests
  - If a page has a .jsp extension, we can assume the application is written using Java Server Pages.
  - If the query string has a parameter named OrderBy, chances are the application is using a database and sorting the data by that value.
- Extrapolating Application Behavior
  - An application often behaves in a similar manner across the range of its functionality because different functions were written by the same developer or to the same design specification.
  - For example, if an attacker has identified a blind SQL injection vulnerability, it may be able to be exploited in other functionalities of the same site

Mapping the Attack Surface

- The final stage of the mapping process is to identify the various attack surfaces exposed by the application.
- The following are some key types of behavior and functionality identified:
  - Client-side validation
    - Checks may not be replicated on the server
  - Database interaction
    - SQL injection
  - File uploading and downloading
    - Path traversal vulnerabilities
  - Display of user-supplied data
    - Cross-site scripting
  - Dynamic redirects
    - Redirection and header injection attacks
  - Login
    - Username enumeration, weak passwords, ability to use brute force
  - Multistage login
    - Logic flaws
  - Session state
    - Predictable tokens, insecure handling of tokens
  - Access controls
    - Horizontal and vertical privilege escalation

Mapping the Attack Surface (cont.)

- The following are some key types of behavior and functionality identified (cont.):
  - User impersonation functions
    - Privilege escalation
  - Use of clear text communication
    - Session hijacking, capture of credentials and other sensitive data
  - Off-site links
    - Leakage of query string parameters in the Referer header
  - Interfaces to external systems
    - Shortcuts in handling of sessions and/or access controls
  - Error messages
    - Information leakage
  - Email interaction
    - Email and/or command injection
  - Native code components or interaction
    - Buffer overflow
  - Use of third-party application components
    - Known vulnerabilities
  - Identifiable web server software
    - Common configuration weaknesses, known software bugs

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook-Discovering and Exploiting Security Flaws

Chapter 5

Bypassing Client-Side Controls

WORKFORCE CENTER

Client-Side Controls

- The user has full control over the client and the data it submits and can bypass any controls which are implemented on the client side and not replicated on the server.
- But an application may rely upon client-side controls to restrict user input by using HTML form features, client-side scripts, or thick-client technologies in one of two ways:
  - An application may transmit data via the client component using some mechanism that it assumes will prevent the user from modifying that data.
  - When an application gathers data that is entered by the user, it may implement measures on the client side that control the contents of that data before it is submitted.

Transmitting Data via the Client

- Hidden Form Fields
  - The field's name and value are stored within the form and sent back to the application when the user submits the form.
  - Even though the field is hidden, it is still stored on the client, so it is in the client's control.
  - An intercepting proxy (such as Burp Proxy, WebScarab, or Paros) sits between your web browser and the target application, intercepting every request issued to the application. They have:
    - Fine-grained rules to control which messages are trapped
    - Regular expression based replace of message content
    - Automatic updating of the `Content-Length` header when messages are modified
    - Browsing history and message cache
    - Ability to replay and remodify individual requests
    - Integration with other tools such as spiders and fuzzers

Transmitting Data via the Client (cont.)

- HTTP Cookies
  - These are not typically displayed to the user, but they are stored on the client, so they can be modified by the client.
- URL Parameters
  - Parameters in the URL query string can be trivially modified by any user without the use of any tools.
- Referer Header
  - The Referer header in HTTP requests indicates the URL of the page from which the current request originated.
- Opaque Data
  - Data transmitted from the client can be encrypted or obfuscated in some way, but the application needs to ensure that this has not been modified by the client somehow.

Capturing User Data: HTML Forms

- HTML forms are the simplest and most common mechanism for capturing input from the user and submitting it to the server.
  - An application can impose client-side controls as a security mechanism to defend against malicious input, but the controls can be trivially circumvented.
  - Length Limits
    - The `maxlength` attribute of an input tag can easily be circumvented by intercepting the request containing the form submission or intercepting the response containing the form and removing the `maxlength` attribute.
  - Script-Based Validation
    - JavaScript can be used to validate form values before submitting to the server, but this can easily be circumvented by disabling JavaScript or by intercepting the validated submission with your proxy and modifying the data to the desired value.
  - Disabled Elements
    - Disabled HTML form elements are grayed out, uneditable, and not sent to the server when the form is submitted. Submitting a disabled form field may make the application vulnerable to SQL injection or cross-site scripting.

Capturing User Data: Thick-Client Components

- Thick-client components include Java applets, ActiveX controls, and Shockwave Flash objects.
  - They can perform complex validation and manipulation of data prior to submission to the server, and since their internal workings are less transparently visible than HTML forms and JavaScript, developers are more likely to assume that the validation they perform cannot be circumvented.
  - Java Applets
    - They are cross-platform and run in a sandbox environment which mitigates against various kinds of security problems that can afflict more heavyweight thick-client technologies.
    - The best way to attack a Java Applet is to attempt to decompile the applet code to obtain its source code.

Capturing User Data: Thick-Client Components (cont.)

- ActiveX Controls
  - These are native Win32 executables that, once accepted and installed by the user, execute with the full privileges of that user and can carry out arbitrary actions (including interacting with the operating system).
  - Decompiling ActiveX controls back to source code (which is usually in C or C++) is not trivial, but it is possible for a user to fully scrutinize and control the processing of the component.
    - If the ActiveX component happened to be written in C#, it can normally be decompiled to recover the original source code.
  - ActiveX components often will read various details from the local computer's file system and registry, which can be monitored by the client and arbitrary inputs can be returned.

Capturing User Data: Thick-Client Components (cont.)

- Shockwave Flash Objects
  - Flash provides increased interactivity in informational web sites and is employed in web applications, such as Flash-based user interfaces for online stores or online games.
  - The Flash SWF file contains bytecode that is interpreted by the Flash Virtual Machine, so it can be decompiled to recover the original ActionScript source code.
  - The Flash SWF bytecode itself can also be modified without actually fully decompiling it to source.
  - For more advanced Flash applications (or those that involve money hopefully), obfuscation techniques have been devised in an attempt to hinder decompilation attacks.

Handling Client-Side Data Securely

- The core security problem with web applications is that the client, and all data received from it, is inherently untrustworthy.
- Many applications transmit critical data such as product prices and discount rates via the client in an unsafe manner, though this data should rightfully be stored on the server.
  - If critical data must be transmitted from the client, the data should be signed and/or encrypted to prevent tampering by the user.
    - Replay attacks should be avoided by including sufficient context with the encrypted data.
    - Also, if users know the plaintext value of encrypted strings, they may be able to mount various cryptographic attacks to discover the encryption key being used by the server.

Validating Client-Generated Data

- Data generated on the client and transmitted to the server cannot in principle be validated securely on the client.
  - Html form fields and JavaScript can be very trivially circumvented and provide no assurance about the input received by the server.
  - Controls in thick-client components are sometimes more difficult to circumvent, but this may merely slow down an attacker for a short period.
  - Heavily obfuscated or packed client-side code provides additional obstacles, but a determined attacker will always be able to overcome these.
- Logging and Alerting
  - Server-side logic which performs validation of client-submitted data should be aware of the validation that supposedly already occurred on the client side.
  - If bad data is still sent to the server, the application may be able to conclude that the data is malicious, which should be logged and administrators may be alerted in real time.

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook- Discovering and Exploiting Security Flaws

Chapter 6
Attacking Authentication

WORKFORCE CENTER

Attacking Authentication

- Authentication lies at the heart of an application's protection against malicious attack since it is the front line of defense against unauthorized access.
  - Without robust authentication to rely upon, none of the other core security mechanisms can be effective.
  - Developers often fail to be able to answer the question, "What could an attacker achieve if he were to target our authentication mechanism?"
  - Authentication is often the weakest link in real-world web applications, which could enable an attacker to gain unauthorized access.

Authentication Technologies

- To implement authentication mechanisms, developers can use:
  - HTML forms-based authentication
    - Using a form to capture a username and a password
    - Accounts for over 90% of applications requiring authentication on the internet
  - Multi-factor mechanisms, such as combining passwords with physical tokens
    - One-time passcodes or challenge-response function based on input specified by the application
  - Client SSL certificates or smartcards
    - Due to the overhead, smartcards are typically only used in security-critical contexts where an application's user base is small.
  - HTTP basic and digest authentication
    - Usually encountered in intranet environments
  - Windows-integrated authentication using NTLM or Kerberos
  - Authentication services
    - Microsoft Passport is one example

Design Flaws in Authentication Mechanisms

- Bad Passwords
  - It is quite likely that an application that does not enforce strong password standards will contain a large number of user accounts with weak passwords set, which can be easily guessed by an attacker.
- Brute-Forcible Login
  - If the application allows an attacker to make repeated login attempts with different passwords until the correct one is guessed, then it is highly vulnerable even to an amateur attacker.
    - A serious attacker can use automated techniques to attempt to guess passwords based on lengthy lists of common values.
- Verbose Failure Messages
  - When a login attempt fails, you can infer that at least one piece of information was incorrect, but if the application informs you as to which piece of information was invalid, you can exploit this behavior to considerably diminish the effectiveness of the login mechanism.
  - It may be possible to enumerate usernames based on the time taken for the application to respond to the login request.
    - It may take longer to return a failure message if the username is valid rather than if the username is not valid. This may have to be detected by an automated script rather than a browser.

Design Flaws in Authentication Mechanisms (cont.)

- Vulnerable Transmission of Credentials
  - If an application uses an unencrypted HTTP connection to transmit login credentials, an eavesdropper who is suitably positioned on eht network will be able to intercept them.
  - Some sites load the login page using HTTP and switch to HTTPS at the point where credentials are submitted.
    - Even though the credentials will be encrypted before sent, the user can not verify the authenticity of the login page.
    - An attacker could intercept the login page response and change the action of the form to HTTP instead of HTTPS and compromise the login credentials.
- Password Change Functionality
  - Web applications should provide password change functionality because password changes reduce the threat of password compromises and allow users to change it if they feel it has been compromised.
  - But, some problems that may arise with password change functionality are:
    - Providing a verbose error message indicating whether the requested username is valid
    - Allowing unrestricted guesses of the "existing password" field
    - Only checking whether the "new password" and "confirm new password" fields have the same value after validating the existing password

Design Flaws in Authentication Mechanisms (cont.)

- Forgotten Password Functionality
  - Several design weaknesses can exist in the forgotten password functionality.
    - The answer to the challenge question is often easier for an attacker to guess than attempting to guess the user's password.
    - Developers often overlook brute-force attacking of password recovery questions.
    - In some applications, the recovery challenge is replaced with a password "hint" configurable by users, and this is often extremely obvious.
- "Remember Me" Functionality
  - Allows users to not have to enter their username and password each time they use the application from a specific computer
    - Sometime this is implemented with a simple persistent cookie that contains the username or session ID.
- User Impersonation Functionality
  - Some applications allow a privileged user to impersonate other users in order to access data and carry out actions within their user context, such as helpdesk operators.
    - If an application allows administrative users to be impersonated, then any weakness in the impersonation logic may result in a vertical privilege escalation vulnerability.
- Incomplete Validation of Credentials
  - Some applications truncate passwords, and so only validate the first n characters, or they perform a case-insensitive check of passwords, or they could strip out unusual characters before checking passwords.

Design Flaws in Authentication Mechanisms (cont.)

- Non-Unique Usernames
  - Two users with the same usernames could select the same passwords, so logins by one of the users will result in access to the other's account.
- Predictable Usernames
  - Some applications automatically generate account usernames according to some predictable sequence.
- Predictable Initial Passwords
  - In some vulnerable cases where initial passwords are generated, all users may receive the same password or one closely derived from their username or job function.
- Insecure Distribution of Credentials
  - Many applications send credentials for newly created accounts via post or email, but this may contain both the username and password.
    - Instead, it may contain an "account activation" URL, but if there is any sequence to subsequent URLs, an attacker can identify this by registering multiple users in close succession

Implementation Flaws in Authentication

- Fail-Open Login Mechanisms
  - This is a flaw in programming logic that may allow a user to be logged in when it should not.
- Defects in Multistage Login Mechanisms
  - Designed to provide enhanced security over the simple model based solely on username and password
    - An application may assume that a user who accesses stage three must have cleared stages one and two.
    - An application may trust some of the data being processed at stage two because this was validated at stage one.
    - An application may assume that the same user identity is used to complete each stage, but it might not explicitly check this.
- Insecure Storage of Credentials
  - If an application stores login credentials in an insecure manner, then the security of the login mechanism is undermined, even though there may be no inherent flaw in the authentication process itself.
    - An example would be a web application where the user credentials are stored unencrypted in a database

Securing Authentication

- Implementing a secure authentication solution involves attempting to meet several key security objectives while possibly trading off other objectives such as functionality, usability, and total cost.
- Possible ways to defeat various attacks against authentication mechanisms:
  - Use Strong Credentials
    - Suitable minimum password quality requirements
    - Unique usernames
    - Do not create sequenced or predicted usernames and passwords if system-generated
    - Users should be permitted to set sufficiently strong passwords

Securing Authentication (cont.)

  - Handle Credentials Secretively
    - All credentials should be created, stored, and transmitted in a manner that does not lead to unauthorized disclosure.
    - All client-server communication should be protected using a well-established cryptographic technology.
    - The login form should be accessed over HTTPS.
    - Only POST requests should be used for transmitting credentials.
  - Validate Credentials Properly
    - Passwords should be validated in full and case-sensitively.
    - All authentication logic should be closely code-reviewed.
    - If user impersonation is implemented, it should be strictly controlled to ensure it can not be misused to gain unauthorized access.
    - Multistage logins should be strictly controlled to prevent an attacker from interfering with the transitions between the stages.
  - Prevent Information Leakage
    - The various authentication mechanisms used should not disclose any information about authentication parameters.
    - A single code component should be responsible for responding to all failed login attempts with a generic message.

Securing Authentication (cont.)

- Prevent Brute-Force Attacks
  - Measures need to be enforced within all of the various challenges implemented by the authentication functionality in order to prevent attacks that attempt to meet those challenges using automation.
  - Using unpredictable usernames and preventing their enumeration presents a significant obstacle to completely blind brute-force attacks.
  - Possibly disable an account after a small number of failed logins
- Prevent Misuse of the Password Change Function
  - A password change function should always be implemented.
  - The function should only be accessible from within an authenticated session.
  - The new password should be entered twice to prevent mistakes.
  - Users should be notified out-of-band that their password has been changed.

Securing Authentication (cont.)

- Preventing Misuse of the Account Recovery Function
  - Password "hints" should never be used.
  - The best automated solution for enabling users to regain control of accounts is to email the user a unique, time-limited, unguessable, single-user recovery URL.
  - The application may present users with a secondary challenge that they must complete before gaining access to the password reset function.
- Log, Monitor, and Notify
  - All authentication-related events should be logged by the application.
  - Anomalies in authentication events should be processed by the application's real-time alerting and intrusion prevention functionality.
  - Users should be notified out-of-band of any critical security events.
  - Users should be notified in-band of frequently occurring security events, such as the time and IP of the last login and the number of invalid login attempts since then.

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook- Discovering and Exploiting Security Flaws

Chapter 7

Attacking Session Management

Attacking Session Management

- Session management enables the application to uniquely identify a user across a number of different requests and to handle the data that it accumulates about the state of that user's interaction with the application.
- Session management enables the application to persist its assurance of any user's identity beyond the request in which they supply their credentials.
- If an attacker can break an application's session management, he can effectively bypass its authentication controls and masquerade as other application users without knowing their credentials.

The Need for State

- HTTP is a stateless protocol where each request-response is independent and there is no mechanism for linking together the series of requests made by one user.
- The simplest and most common means of implementing sessions is to issue each user a unique session ID and have the user resubmit this ID on each subsequent request.
  - HTTP cookies are used for this in most cases
- The vulnerabilities that exist in session management fall into two categories:
  - Weaknesses in the generation of session tokens
  - Weaknesses in the handling of session tokens throughout their lifecycle

Alternatives to Sessions

- Some security-critical applications containing authentication mechanisms and complex functionality opt to use other techniques than sessions for managing state.
  - HTTP Authentication
    - The client component interacts with the authentication mechanism directly via the browser using HTTP headers.
    - Once a user has entered his credentials into a browser dialog, the browser effectively resubmits these credentials with every subsequent request to the same server.
  - Sessionless State Mechanisms
    - Instead of transmitting a session ID, some applications transmit all data required to manage that state via the client in a cookie or a hidden form field.
    - The data transmitted via the client must be properly protected though.

Weaknesses in Session Token Generation

- Meaningful Tokens
  - Some session tokens are created using a transformation of some user information, and then may be encoded.
  - Meaningful data in a session token often contains several components separated by a delimiter.
- Predictable Tokens
  - Some session tokens contain sequences or patterns that allow an attacker to extrapolate from a sample of tokens to find other valid tokens recently issues by the application.
  - Concealed Sequences
    - Session tokens that can not be trivially predicted when analyzed in their raw form but that contain sequences that reveal themselves when the tokens are suitably decoded or unpacked.
  - Time Dependency
    - Session tokens that use the time of generation as an input to the token's value.
  - Weak Random Number Generation
    - When a predictable pseudo-random number generator is used for producing session tokens, the resulting tokens are vulnerable to sequencing by an attacker.

Weaknesses in Session Token Handling

- Disclosure of Tokens on the Network
  - This occurs when the session token is transmitted across the network in unencrypted form, enabling an eavesdropper to obtain the token and masquerade as a legitimate user.
    - Even if the application uses HTTPS to protect the user's credentials during login, if any page is loaded via HTTP, the session token can be intercepted by an eavesdropper.
    - Even if every page is accessed over HTTPS, if the user revisits a pre-authentication page after being authenticated, the session token can be transmitted over HTTP.
- Disclosure of Tokens in Logs
  - Session tokens are disclosed to unauthorized view in system logs.
  - If the session token is transmitted in the URL, it will most likely appear in the system log files as well.

Weaknesses in Session Token Handling (cont.)

- Vulnerable Mapping of Tokens to Sessions
  - Various vulnerabilities exist based on how the application maps session tokens to an individual user's session.
    - This can occur if a single user has more than one active session. If both are accessed simultaneously, this usually indicates a security compromise has occurred.
- Vulnerable Session Termination
  - Proper termination of sessions is important for two reasons:
    - Keeping the lifespan of a session as short as necessary reduces the window of opportunity within which an attacker may compromise a valid session.
    - It provides users with the ability to invalidate an existing session, which enables the user to reduce this window further to secure their session.
  - Applications need to provide effective logout functionality that invalidates the session on the server.

Weaknesses in Session Token Handling (cont.)

- Liberal Cookie Scope
  - The cookie mechanism allows a server to specify both the domain and the URL path to which each cookie will be resubmitted, using the `domain` and `path` attributes in the `Set-cookie` instruction.
  - Cookie Domain Restrictions
    - If an application sets a cookie's domain scope too liberally, the cookie will be transmitted more frequently than necessary, which may expose the application to various security vulnerabilities.
  - Cookie Path Restrictions
    - If an application sets a cookie's path scope too liberally, the cookie will be transmitted more frequently than necessary, which may expose the application to various security vulnerabilities.

Securing Session Management

- Generate Strong Tokens
  - Session tokens should be generated in a manner that does not provide any scope for an attacker who obtains a large sample of tokens from the application in the usual way to predict or extrapolate the tokens issues to other users.
  - Effective token generation mechanisms:
    - Use an extremely large set of possible values.
    - Contain a strong source of pseudo-randomness, ensuring an even and unpredictable spread of tokens across the range of possible values.
    - Include some information about the individual request for which the token is being generated, such as IP address, port, `User-Agent` header, and time of request.
      - A highly effective formula for incorporating these fields is to construct a string that concatenates a pseudo-random number, a variety of request-specific data, and a secret string known only to the server and generated new on each reboot.

Securing Session Management (cont.)

- Protect Tokens throughout Their Lifecycle
  - The token should only be transmitted over HTTPS.
  - Session tokens should never be transmitted in the URL.
  - Logout functionality should be implemented.
  - Session expiration should be implemented after a suitable period of inactivity.
  - Concurrent logins should be prevented.
  - The domain and path scope of an application's session cookies should be set as restrictively as possible.
- Per-Page Tokens
  - Finger-grained control over sessions can be achieved by using per-page tokens in addition to session tokens.
    - A new page token is created every time a user requests an application page and is passed to the client in a cookie or a hidden form field. Each time the user makes a request, the page token is validated against the last value issues, in addition to the normal validation of the main session token.

Securing Session Management (cont.)

- Log, Monitor, and Alert
  - The application should monitor requests that contain invalid tokens.
  - Brute-force attacks against session tokens are difficult to block altogether because there is no particular user account or session that can be disable to stop the attack, but the source IP can be blocked for a period of time.
  - Keep detailed logs of brute-force attacks and alert administrators in real-time to investigate the attack.
  - Users should be alerted to anomalous events relating to their session, such as concurrent logins or apparent hijacking (detected using per-page tokens).
- Reactive Session Termination
  - Some applications will terminate a user's session every time the user submits some anomalous request, such as with a modified hidden HTML form field or URL query string parameter.
  - Forcing users to re-authenticate every time they submit an invalid request can slow down the process of probing the application for vulnerabilities by an attacker.

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook- Discovering and Exploiting Security Flaws

Chapter 9
Injecting Code

Injecting into Interpreted Languages

- An interpreted language is one whose execution involved a runtime component that interprets the code of the language and carries out the instructions that it contains
  - SQL, LDAP, Perl, PHP.
- In most applications, the code processed by the interpreter is a mix of instructions written by a programmer and data supplied by a user.
  - An attacker can supply crafted input that breaks out of the data context, usually by supplying some syntax that has a special significance within the grammar of the interpreted language.

Injecting into SQL

- Web applications commonly construct SQL statements that incorporate user-supplied data, which if vulnerable, can enable an attacker to read and modify all data stored in the database.
- If an application does not handle single quotation marks in user-supplied data, it is wide open to SQL injection.
  - An attacker can supply input containing a quotation mark to terminate the string that he controls, and can then write arbitrary SQL to modify the query.
- Look at the following two queries and compare the difference in the output, where the user input is in red (Note that the single quote is part of the user input in the second example):

```
SELECT * FROM books WHERE publisher='Wiley'
SELECT * FROM books WHERE publisher='Wiley' OR 1=1--'
```

Finding SQL Injection Bugs

- For string data or numeric data, the following steps are normally sufficient to identify the majority of SQL injection vulnerabilities:
- String data
  - In order to exploit any SQL injection flaw with user-supplied string data, you need to break out of the quotation marks that encapsulate a string in SQL.
    - Submit a single quotation mark.
    - Submit two single quotation marks together.
- Numeric data
  - The application may handle numeric data as a string by encapsulating it within single quotation marks, so always perform the steps described for string data.
    - Submit a simple mathematical expression.
    - Use SQL-specific keywords and syntax, such as the ASCII command, which returns the numeric ASCII code of the supplied character.

Injecting into Different Statement Types

- SELECT Statements
  - The entry point for SQL injection attacks is normally the WHERE clause of the query since that is where user-supplied items are passed to the database.
- INSERT Statements
  - If any fields in an INSERT statement are vulnerable to SQL injection, an attacker can insert arbitrary data into the table, including values for fields that he should not be able to control.
- UPDATE Statements
  - An UPDATE statement works in a similar way to an INSERT statement, except it usually contains a WHERE clause like a SELECT statement.
- DELETE Statements
  - As with UPDATE statements, a WHERE clause is normally used to tell the database which rows of the table to update, and user-supplied data is most likely to be incorporated into this clause.

Bypassing Filters

- The application may remove or sanitize certain characters or block common SQL keywords, though these types of filters are often vulnerable to bypasses.
  - Avoid blocked characters
    - If the application removes some characters that are often used in SQL injection attacks, remember that the single quotation marks are not required for numeric fields.
    - If the comment symbol is blocked, you can inject values that are always true such as 'a'='a'.
  - Circumventing simple validation
    - Some input validation will block or remove any supplied data which appears on a list.
  - Using SQL comments
    - Comments can be used to simulate whitespace within your injected data.
  - Manipulating blocked strings
    - If the application blocks certain strings that you wish to place as data items in an injected query, the required string can be constructed dynamically using various string manipulation functions.
  - Using dynamic execution
    - Some databases allow SQL statements to be executed dynamically by passing a string representation of a particular statement to the relevant function.

Second-Order SQL Injection

- It is very common for applications to defend themselves against SQL injection by escaping single quotation marks with a second single quotation mark.
- But, this may pose a problem if the same item of data is being passed through several SQL queries, being written to the database and read back more than once
- An attacker could successfully bypass the input validation designed to block SQL injection attacks and execute arbitrary queries within the database and retrieve results.

Escalating the Database Attack

- If the database is shared with other applications, you may be able to escalate privileges within the database and gain access to other applications' data.
- You may be able to compromise the operating system of the database server.
- You may be able to gain network access to other systems.
- You may be able to make network connections back out of the hosting infrastructure to your own computer.
- You may be able to extend the database's existing functionality in arbitrary ways by creating user-defined functions.

Preventing SQL Injection

- Partially Effective Measures
  - Escaping single quotation marks within user input by doubling them up
  - Using stored procedures for all database access
- Parameterized Queries
  - The application specifies the structure of the query, leaving placeholders for each item of user input.
  - The application specifies the contents of each placeholder.
- Defense in Depth
  - The application should use the lowest possible level of privileges when accessing the database.
  - Unnecessary functions should be removed or disabled.
  - All vendor-issued security patches should be evaluated, tested, and applied in a timely manner to fix known vulnerabilities within the database software itself.

Injecting into Web Scripting Languages

- Dynamic execution vulnerabilities
  - Developers can create applications that dynamically modify their own code in response to various data and conditions.
  - If user input is incorporated into code that is dynamically executed, an attacker may be able to supply crafted input that breaks out of the intended data context and specifies commands that are executed on the server.
- File Inclusion Vulnerabilities
  - Include files enable developers to place reusable code components into individual files.
  - The code in the included file is interpreted as if it had been inserted at the location of the include directive.
  - Vulnerabilities often exist in request parameters that specify a language or location and arise when the name of a server-side file is passed explicitly as a parameter.

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook- Discovering and Exploiting Security Flaws

Chapter 12-13
Attacking The User

Attacking Other Users

- The attacks described in this chapter are attempting to attack other users by leveraging some aspect of the application's behavior.
- These actions may result in session hijacking, unauthorized actions, disclosure of personal data, logging of keystrokes, or execution of arbitrary commands on users' computers.
- As the general awareness of security threats has evolved, the front line of the battle between software developers and hackers has moved from the server to the client, with such terms as spyware, phishing, and Trojans.

Cross-Site Scripting

- Cross-site scripting (XSS) is the most prevalent web application vulnerability found, afflicting the vast majority of live applications.
- A very common XSS vulnerability occurs when an application employs a dynamic page to display error messages to users.
  - Typically the page takes a parameter containing the text of the message and simply renders the text back to the user within its response.
  - If no filtering or sanitization is being performed, the application is certainly vulnerable.
- The most common XSS flaw results in the attacker capturing the session token of an authenticated user.

XSS Capturing Session Token

- The user logs in to application and is issued a cookie containing a session token.
- The attacker feeds a malicious XSS URL to the user.
- The user requests from the application the URL from above.
- The server responds to the user's request, which contains the JavaScript created by the attacker.
- The attacker's JavaScript is received by the user's browser.
- The malicious JavaScript created by the attacker causes the user's browser to make a request to a different site, but it contains the user's session token.
- The attacker receives the user's request and uses the captured token to hijack the user's session.

Stored XSS Vulnerabilities

- A stored XSS vulnerability arises when data submitted by one user is stored within the application and then displayed to other users without being filtered or sanitized appropriately.
- These are common in applications that support interaction between end users or where administrative staff access user records and data within the same application.
- If an HTML or text file can be uploaded containing JavaScript and a victim views the file, the payload may normally be executed.
  - Many applications disallow the uploading of HTML files to prevent this kind of attack.

Real-World XSS Attacks

- Web mail application are at risk of stored XSS attacks if the attacker can send a victim an HTML-formatted email containing malicious JavaScript that does not get filtered by the application.
  - The victim's web mail account may be compromised solely by reading the email.
- XSS flaws can sometimes be chained with other vulnerabilities to cause a much more devastating effect.
- Virtual defacement attacks involve injecting malicious data into a page of a web application to feed misleading information to users of the application.

Finding and Exploiting XSS Vulnerabilities

- A basic approach to identifying XSS vulnerabilities is to use a standard proof-of-concept attack string such as:

```
"><script>alert(document.cookie)</script>
```

- This string is submitted as every parameter to every page of the application and responses are monitored for the appearance of this same string.
  - If the attack string appears unmodified within the response, the application is almost certainly vulnerable to XSS.

HttpOnly Cookies

- One of the various payloads for attacking XSS vulnerabilities is to capture a victim's session token by using injected JavaScript to access the document.cookie property.
- HttpOnly cookies are a defense mechanism supported by some browsers and employed by some applications in an attempt to prevent this attack payload from succeeding.
  - When a cookie is flagged as HttpOnly, supporting browsers will prevent client-side JavaScript from directly accessing the cookie.
  - It will not be returned in the string returned by document.cookie.

Preventing XSS Attacks

- Preventing XSS attacks is problematic in practice because of the difficulty of identifying every instance in which user-controllable data is handled in a potentially dangerous way.
- Preventing reflected and stored XSS
  - Identify every instance within the application where user-controllable data is being copied into responses.
- Validate input
  - The application should perform context-dependent validation of this data in as strict a manner as possible.
- Validate output
  - Any responses that originated from some user should be HTML-encoded to sanitize potentially malicious characters.

Redirection Attacks

- Redirection vulnerabilities arise when an application takes user-controllable input and uses this to perform a redirection, instructing the user's browser to visit a different URL than one requested.
- Many applications perform redirects to third-party sites as part of their normal function, which encourages users to perceive that redirection during a transaction is not necessarily indicative of anything suspicious.
- To prevent redirection vulnerabilities, the easiest way is to remove the redirection page and replace it with direct links to relevant target URLs.

Browser Exploitation Frameworks

- A JavaScript hook placed into the browser of a victim via some vulnerability such as XSS allows an attacker to do the following:
  - Log keystrokes and send to attacker
  - Capture clipboard contents and send to attacker
  - Hijack the user's session with the vulnerable application
  - Fingerprint the victim's browser and exploit known browser vulnerabilities
  - Perform port scans of other hosts
  - Attack other web applications accessible via the compromised user's browser
  - Brute forcing the user's browsing history and sending to attacker

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Hacker's Handbook- Discovering and Exploiting Security Flaws

Chapter 11

Attacking Application Logic

Attacking Application Logic

- Flaws in an application's logic are not usually identified by any automated vulnerability scanners, so they are generally not as well appreciated or understood.
  - Logic flaws are therefore of great interest to an attacker.
- The only defining characteristic is that the logic implemented within the application is defective in some way.
- Any serious attacker needs to try to figure out any assumptions that designers and developers are likely to have made and then think imaginatively about how those assumptions may be violated.

Fooling a Password Change Function

- Functionality
  - User had to fill out username, existing password, new password, and confirm new password for password change function
  - Administrators could also change the password of any user without needing to supply the existing password.
- Assumption
  - The administrator's interface only differed by not containing a field for an existing password, so maybe the server-side application used the presence of the existing password to indicate whether the request was from an administrator or an ordinary user.
  - Ordinary users would always provide an existing password.
- Attack
  - An ordinary user can issue a request that does not contain an existing password parameter.
  - An attacker could reset the password of any other user.

Proceeding to Checkout

- Functionality
  - Placing an order involves browsing the product catalog and adding items to the shopping cart, returning to the shopping cart to finalize the order, entering payment information, and entering delivery information.
- Assumption
  - The developers assumed that users would always access the stages in the intended sequence, so any user who completed the order process must have submitted payment details along the way.
- Attack
  - The assumption is flawed because users control every request they make to the application, so they can access any stage of the ordering process in any sequence.
  - By proceeding from finalizing the shopping cart to entering delivery information, an attacker can finalize an order for delivery without paying for it.

Breaking the Bank

- Functionality
  - The application enabled existing customers who did not already use the online application to register to do so.
  - New users were required to supply name, address, and date of birth, but not a password or PIN.
  - An information packet was then mailed to the user's home address, which included instructions for activating their online access via telephone and a one-time password for logging in.
- Assumption
  - The developers reused the existing Customer object, which included the user's unique customer number used in all of the company's systems, and stored the instance of this object in the user's session.
- Attack
  - The same code component was used elsewhere within the application, including with the core functionality, which gave authenticated users access to account details, funds transfers, statements, and other information.
  - In essence, an attacker could obtain an authenticated instance of the Customer object by merely providing the name, address, and date of birth of a user.

Erasing an Audit Trail

- Functionality
  - The application enabled helpdesk personnel and administrators to support and manage a large user base, including creating accounts and resetting passwords.
  - The application recorded every action performed and the identity of the user responsible, and administrators could delete audit trail entries, though this was also recorded.
- Assumption
  - The designers believed it would be impossible for a user to perform an undesirable action without leaving some evidence in the audit trail.
- Attack
  - Log in using your own account and create a second user account.
  - Assign all of your privileges to the new account.
  - Use the new account to perform a malicious action of your choice.
  - Use the new account to delete all of the audit log entries generated by the previous three steps.
  - There is one suspicious entry in the log, indicating that some log entries were deleted by a specific user, but there is nothing linking the attacker to the new user.

Beating a Business Limit

- Functionality
  - Finance personnel had the ability to transfer funds between various bank accounts owned by the company and their key customers and suppliers.
  - Any transfer larger than $10,000 required a senior manager's approval
- Assumption
  - The code for checking the transfer amount merely checked it against the threshold and returned true or false based on those two values.
- Attack
  - The assumption is flawed because he did not take into account a transfer for a negative amount, and the banking module accepted negative transfers and simply processed them as positive transfers in the opposite direction.
  - So, any user wanting to transfer $20,000 from account A to account B could transfer -$20,000 from account B to account A.

Cheating on Bulk Discounts

- Functionality
  - The application allowed users to order software products and qualify for bulk discounts if a suitable bundle of items was purchased.
- Assumption
  - When a user added an item to his shopping cart, the application used various rules to determine whether the bundle of purchases entitled him to any discount.
  - If so, the prices of the relevant items in the shopping cart were adjusted by the discount rate
  - The developers assumed the user would go on to purchase the chosen bundle then.
- Attack
  - The developers assumed that users may remove items from the shopping cart after receiving the discount.
  - An attacker could add to his cart large quantities of every single product on sale from the vendor to attract the maximum possible bulk discount, then remove the items he did not want and still receive the discounts on the remaining products.

Abusing a Search Function

- Functionality
  - The application provided a search function that could be accessed by all users, but an anonymous user would be required to subscribe in order to retrieve any of the protected documents that the query returned.
- Assumption
  - The application's designer assumed that users could not use the search functionality to extract any useful information without paying for it.
- Attack
  - Because the search function indicated the number of documents that matched a query, a user could issue a large number of queries and use inference to extract information that would normally need to be paid for.
  - Although the user could not view the actual document itself, he may be able to build up a fairly accurate understanding of its contents.

Avoiding Logic Flaws

- There is no unique signature by which logic flaws in web application can be identified, and there is also no silver bullet with which to protect them.
- We can reduce the risks of logic flaws
  - Ensure every aspect of the application's design is clearly documented.
  - Mandate that all source code is clearly commented to include the purpose, the assumptions, and references to client code that use it.
  - Reflect upon the assumptions made.
  - Be aware that users control every aspect of every request.
  - A user's identity should only be derived from his/her session.
  - Be wary of any functionality that allows users to delete items, especially from an audit trail.
  - Ensure orders are finalized before applying any discounts.

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook- Discovering and Exploiting Security Flaws

Chapter 17
Attacking the Web Server

WORKFORCE CENTER

Attacking the Web Server

- Vulnerabilities affecting web servers are either because of shortcomings in the server's configuration or security flaws within the web server software.
- Historically, many web servers have shipped with insecure default configuration options.
- Many web servers come with administrative interfaces that may be publicly accessible on a port, such as 8080 or 8443, and default credentials are not required to be changed on installation.
- For example, by default, Apache Tomcat has a username of "admin", "tomcat", or "root", with no password, a password of "tomcat", or a password of "root."

Default Content

- Most web servers ship with a range of default content and functionality that you may be able to leverage to attack either the server or the main application.
- Debug Functionality
  - Functionality designed for diagnostic use by administrators is often of great value to an attacker because it may contain useful information about the configuration and applications running on it.
- Sample Functionality
  - Many servers include by default various sample scripts and pages designed to demonstrate how certain web server functions can be used.
- Powerful Functions
  - Some web server software contains powerful functionality, such as an interface that proxies web requests to a back-end database, allowing arbitrary parameters to be passed.

Directory Listings

- When a web server receives a request for a directory instead of a file, it may respond in one of three ways:
  - It may return a default resource within the directory, such as index.html.
  - It may return an error, such as HTTP status code 403, indicating that the request is not permitted.
  - It may return a listing showing the contents of the directory.
- Directory listings may assist you in attacking an application.
  - Many applications do not enforce proper access control over their resources and rely upon an attacker's ignorance of the URLs used to access sensitive items.
  - Files and directories (such as logs, backup files, and old versions of scripts) are often unintentionally left within the web root of servers.

Dangerous HTTP Methods

- If HTTP methods, other than GET or POST, are accessible by low-privileged users, they may provide an effective avenue for attacking an application.
  - PUT
    - Uploads the attached file to the specified location
  - DELETE
    - Deletes the specified resource
  - COPY
    - Copies the specified resource to the location given in the Destination header
  - MOVE
    - Moves the specified resource to the location given in the Destination header
  - SEARCH
    - Searches a directory path for resources
  - PROPFIND
    - Retrieves information about the specified resource, such as author, size, and content type
  - TRACE
    - Returns in the response body the exact request received by the server
  - OPTIONS
    - Lists the HTTP methods that are permitted in a particular directory

Securing Web Server Configuration

- The most important task to securing a web server is to fully understand the documentation for the software you are using.
- Here are some areas to address:
  - Change any default credentials if possible and remove default accounts that are not required.
  - Block public access to administrative interfaces, typically by firewalling access to nonstandard ports.
  - Remove all default content and functionality that is not strictly required for business purposes.
  - Check all web directories for directory listings and disable directory listings in a server-side configuration.
  - Disable all HTTP methods except those used by the application.
  - Ensure that the web server is not configured to run as a proxy.
  - Ensure that any security hardening is applied on the default host if your web server supports virtual hosting.

Vulnerable Web Server Software

- Web server products range from extremely simple and lightweight software which does little more than serve up static pages to highly complex application platforms that can handle a large variety of tasks.
- Buffer Overflow Vulnerabilities
  - These allow an attacker to take control of execution in the vulnerable process.
  - Achieving arbitrary code execution within a web server will usually enable an attacker to compromise any application that it is hosting.

Vulnerable Web Server Software

- Path Traversal Vulnerabilities
  - The same types of path traversal problems discussed in Chapter 10 can be applied to web applications.
  - This would enable an attacker to read or write arbitrary files outside the web root.
- Encoding and Canonicalization Vulnerabilities
  - Various schemes exist that allow unusual characters and content to be encoded for safe transmission over HTTP.
  - If different components (web server, application platform, APIs, software components, operating system, etc.) handle an encoding scheme in different ways, or perform additional decoding or interpretation of data, then this can often be exploited to bypass filters or cause other anomalous behavior.

Finding Web Server Flaws

- If you are lucky, the web server you are targeting may contain some of the actual vulnerabilities described previously.
- More likely, it will have been patched recently, and you will need to search for something fairly current or brand new to attack the server.
- Some web applications include an open-source web server, and security updates to these bundled servers may be applied more slowly because administrators may view the server as part of the installed application.
- If possible, perform a local installation of the software you are attacking and carry out your own testing to find new vulnerabilities that have not been discovered or widely circulated.

Securing Web Server Software

- Choose software with a good track record.
  - Look at the recent history of different server products to see marked differences in the quantity of serious vulnerabilities found, the time taken by vendors to resolve them, and the resilience of the released fixes to subsequent testing by researchers.
- Apply vendor patches.
  - Any decent software vendor must release security updates periodically, and these should be applied as soon as possible.
- Perform security hardening.
  - Most web servers have numerous configurable options controlling what functionality is enabled and how it behaves.
  - Disable any built-in functionality that is not required, rename functions and resources from their default values, and apply the principle of least privilege throughout the technology stack.
- Monitor for new vulnerabilities.
- Use defense-in-depth.
  - Impose restrictions on the web server's capabilities from other autonomous components of the application.
  - Impose strict network-level filters on traffic to and from the web server.
  - Use an intrusion detection system to identify any anomalous network activity that may indicate that a breach has occurred.

SAINT LOUIS UNIVERSITY

The Web Application Hacker's Handbook- Discovering and Exploiting Security Flaws

Chapter 20

A Web Application Hacker's Toolkit

A Web Application Hacker's Toolkit

- Some attacks on web applications can be performed using only a standard web browser, but the majority of them require you to use some additional tools.
- The most important item in your toolkit runs alongside a browser and modifies its interaction with the target application, acting as an intercepting web proxy and enabling you to view and modify all HTTP messages passing between your browser and the server.
- The web application scanner is a product designed to automate many of the tasks involved in attacking a web application, from initial mapping to probing for vulnerabilities.

Web Browsers

- Web browsers are not exactly a hack tool, but your choice of browser may have an impact on your effectiveness when attacking a web application.
- Internet Explorer
  - 60% of the market uses IE, and virtually all web applications are designed and tested for it.
  - Other browsers do not natively support ActiveX controls.
  - Three useful extensions are available to IE – HttpWatch, IEWatch, and TemperIE.
- Firefox
  - 35% of the market uses Firefox.
  - A large number of extensions are available for Firefox – FoxyProxy, Tamper Data, LiveHTTPHeaders, AddNEditCookies, and CookieWatcher.
- Opera
  - Less than 2% of the market uses Opera.
  - Opera has many features built in, such as F12+x enables or disables the proxy, ALT+CTRL+L displays all the links in the document, CTRL+F3 displays the syntax-highlighted source of the current page, ALT+T+A+C displays cookies and allows them to be edited.

Integrated Testing Suites

- After the web browser, the most useful tool, when attacking a web application, is an intercepting proxy.
- There are three leading intercepting proxies:
  - Burp suite
  - Paros
  - WebScarab
- Each integrated tool contains several other tools that share information about the target application.
- The attacker engages the application via a browser, and the tools monitor resulting requests and responses, storing all relevant details about the target application and providing numerous useful functions.
  - Intercepting proxy
  - Web application spider
  - Application fuzzer or scanner
  - Manual request tool
  - Various shared functions and utilities

Intercepting Proxies

- To make use of an intercepting proxy, you must configure your browser to use as its proxy server a port on the local machine.
- The proxy has access to the two-way communications between the browser and the destination web server, so it can stall each message for review and modification by the user.
  - The browser sends standard HTTP requests to the proxy, with the exception that the URL in the first line of the request contains the full hostname of the destination web server.
  - The proxy parses the hostname and resolves it to an IP address.
  - It converts the request to the non-proxy equivalent and forwards it to the destination web server.
  - When the server responds, it forwards the response back to the client browser.
- HTTPS communication can also be intercepted by a proxy, though the browser will display an alert to the user stating the certificate of the server could not be verified, since the security of the proxy is being used instead.
  - If the proxy and the browser are both under control of the attacker, however, it is really irrelevant that the message is displayed.

Web Application Spiders

- Web application spiders work similarly to traditional web spiders by requesting web pages, parsing them for links to other pages, and then requesting those pages, continuing recursively until all of a site's content has been discovered.
- Application spiders must also address these challenges though:
  - Forms-based navigation, using drop-down lists, text input, and other methods
  - JavaScript-based navigation, such as dynamically generated menus
  - Multistage functions requiring actions to be performed in a defined sequence
  - Authentication and sessions
  - The use of parameter-based identifiers, rather than the URL, to specify different content and functionality
  - The appearance of tokens and other volatile parameters within the URL query string, leading to problems identifying unique content

Application Fuzzers and Scanners

- Automation must be used to enhance speed and effectiveness of an attack.
  - Automated scans detect common vulnerabilities by sending a set of attack strings as each parameter in a given request and analyzing the application's responses to identify signatures of common vulnerabilities.
  - Manually configured scans enable you to control precisely which attack strings are used and how they are incorporated into requests.
  - Built-in attack payloads and versatile functions generate arbitrary payloads in user-defined ways based on malformed encoding, character substitution, brute force, etc.
  - Functions can be written for extracting useful data from the application's responses, which can be useful when you are exploiting various vulnerabilities for flaws in session-handling and access controls.

Manual Request Tools

- The manual request component of the integrated test suite provides the basic facility to issue a single request and view its response.
  - This is extremely beneficial if you need to reissue the same request multiple times, tweaking the elements of the request to determine the effect on the application's behavior.
- The following features are implemented within manual request tools:
  - Integration with other suite components and the ability to refer any request to and from other components for further investigation
  - History of all requests and responses, keeping a full record of all manual requests for futher review

Vulnerability Scanners

- A vulnerability scanner has the benefit of being able to test a large amount of functionality in a relatively short time.
- Web application vulnerability scanners can detect a number of common vulnerabilities.
  - Reflected cross-site scripting vulnerabilities arise when user-supplied input is echoed back in the application's responses without appropriate sanitization.
  - Some SQL injection vulnerabilities can be detected via a signature.
  - Some path traversal vulnerabilities can be detected by submitting a traversal sequence targeting a well-known file such as boot.ini or /etc/passwd.
  - Some command injection vulnerabilities can be detected by injecting a command that will cause a time delay or will echo a specific string into the application's response.
  - Straightforward directory listings can be identified by requesting the directory path.

Inherent Limitations of Scanners

- Every web application is different.
  - Web applications differ from the domain of IT networks and infrastructures, in which a typical installation employs off-the-shelf products in standard configurations.
- Scanners operate on syntax
  - Scanners can not understand the semantic meaning of the content returned, nor can they make normative judgments on the basis of this meaning.
- Scanners do not improvise
  - Many web applications use nonstandard mechanisms for handling sessions and navigation and for transmitting and handling data, for which a human being may quickly notice and deconstruct the unusual mechanism that a computer can not.
- Scanners are not intuitive
  - The approach of today's scanners is to attempt every attack against every function, but they have no intuition about how best to proceed.

Technical Challenges Faced by Scanners

- Authentication and Session Handling
  - A scanner that fails to obtain an authenticated session will miss many detectable flaws.
- Dangerous Effects
  - Running an unrestricted automated scan without any user guidance may be highly dangerous to the application and the data it contains.
  - If the scanner blindly tries to execute a number of functions, it may result in access being denied to all users of the application.
- Individuating Functionality
  - A purely syntactic analysis of an application may fail to correctly identify its core set of individual functions.