Web Security

Rattikorn Hewett

NSF-SFS Workshop
Jul 14-18, 2014

Outline

• Basic Concepts
• Security Types and Perspectives
• Attacks and Threats
• Some Tools and Research Snapshots
• Conclusions

Web Security in Contexts

Cyber Security

Web Security

Computer Security

Protects against Malicious Acts

Malicious Acts + Activities on the Web

• Assumes no Malicious Acts
• Protects HW/SW (e.g., OS), Operations, Network, Data

Why should “Web Security” receive special attention?

• High publicity
  ➔ Attractive target as attack is a public event
• High exposure/connectivity to users
  ➔ Vulnerable
• Increasingly crucial for business functions
  ➔ Loss of identity/sensitive information/money/reputation
• Increased Complexity
  ➔ Security flaws
• More expensive to fix than to prevent
  ➔ Economic impacts
Web Security
• Involves technologies and practices to protect
  • Web Servers (computers that run the websites)
  • Web Users (sensitive/proprietary information)

What is a Web Server
• A Web Server runs a Website
  • Has a corresponding Name and IP Address
• A Web Server delivers
  • Web pages to Web Browsers (to public)
  • Files to Web applications

A Web Server = A Computer (with Site content files)
+ Internet Access
+ Web Server Software (e.g., Apache)

Web Server Architecture
• Most Web Services are based on the Client/Server model
  • Web Servers
    • Retrieve files from the server’s hard drive
    • Format the files for the Web browser (by CGI)
    • Send them via the network

Extended Web Server Architecture
• Web Server: provides Static document (web page, image)
  • Application Server: supports Dynamic contents by
    • Running Web Application programs (e.g., looking up stock price, processing transactions) to conduct business
  • Data Base Server: supports Data Base access
Outline

- Basic Concepts
- Security Types and Perspectives
- Attacks and Threats
- Some Tools and Research Snapshots
- Conclusions

Security Types

- Securing Web Servers
  - Server is in operation
  - Data on it is secured
- Securing Data Transmission between Server and Users
  - No eavesdropping (read/listen)
  - No modification
- Securing Web User’s computer
  - Safe download
  - Safe service transactions
  - Protect sensitive information

Popular Counter Measures

- Securing Web Servers
  - By Anti-Virus
- Securing Data Transmission between Server and Users
  - By Cryptographic protocols
    - E.g., SSL (secure socket layers)
- Securing Web User’s computer
  - By Anti-Virus

Two Perspectives

- User Perspectives:
  - How to help users protect themselves during activities on the Web such as:
    - Web browsing
    - Online transactions
    - Social networking
  - Tools to alert users, educate users of potential attacks
- Developer Perspectives: Assume traditional computer security technologies (e.g., authentication, authorization), find techniques to
  - Prevent/Defend attacks on the Web Server
  - Tools to detect compromised Web sites/applications
  - Develop secure software
  - Software security including Web applications
Outline

- Basic Concepts
- Security Types and Perspectives
- Attacks and Threats
- Some Tools and Research Snapshots
- Conclusions

Well-known Web Security Attacks

![Diagram of Web Security Attacks]

- Client
- Web Server
- Web Browser
- Web Application
- Internet Crime

- Social Engineering
- Malware
- CrossSite Scripting
- Click Fraud
- HeartBleed
- Buffer Overflow
- SQL Injection
- CrossSite Scripting

Social Engineering Attacks

Attacker engages victims thru online social interactions to obtain useful information

- Spam – unsolicited bulk emails
  - Intentional – e.g., advertising through fraud links
  - Unintentional – e.g., by infected computers
- Spoofing – impersonated e-mails/websites hoping to "phish" the recipients or for them to open attachment
  - E.g., replace ".org" with ".com"
  - replace "i" with "1"
- Phishing – spams or malicious websites intended to get sensitive information from recipients

Malware

Malicious Software - various types:

- Virus: Malicious code that makes copies of itself and attaches to other programs
- Worm: Malicious code that makes copies of itself and attaches to other computers
- Trojans: Program that appears legitimate but has hidden damages (e.g., viruses, delete files, create back door for attacks)
- Bots: Automated software to interact with other network services (e.g., IM). Malicious bots can infect networks like worms

Heartbleed

Internet Communication

Plain Text

“My social ID is 123456789”

Internet

“My social ID is 123456789”

Alice

Bob

Alice sent “My social ID is 123456789” to Bob

Sniffer

“Secured” Internet Communication

Encrypted/Decrypted Texts

“My social ID is 123456789”

Internet

“My social ID is 123456789”

Encrypt

Decrypt

Alice

Bob

Alice sent “XYABSZEDMI123456” to Bob

Sniffer

SSL (Secure Socket Layer)

Cryptographic protocols that
• Provide secured communication over the Internet
• Are widely used, e.g.,
  • web browsing
  • electronic mails
  • Internet faxing
  • instant messaging
  • voice-over-IP (VoIP)

NO SSL
http://www.cnn.com
http://www.google.com

SSL enabled web browsing
https://www.bankofamerica.com
https://www.facebook.com
**“Secured” Internet Communication**

- Encrypted/Decrypted Texts:
  - "My social ID is 123456789"
  - "XYABSZED MI123456"

- Alice sent "XYABSZED MI123456" to Bob

**OpenSSL**

- Open-source implementation of the SSL (Secure Socket Layer) and TLS (Transport Layer Security) protocols
- Implements basic cryptographic functions
- >50% of web servers on Internet use functions from OpenSSL to provide HTTPS (secure connection) protocol

**Heartbeat**

- OpenSSL extension proposed as a standard in February 2012 by RFC 6520
- Provides tests to keep secure communication links alive without renegotiation the connection (handshaking)

**Heartbleed**

- Failure of bounds checking in SSL Heartbeat request
- Server Memory Space
  - Has 500 letter word, "bird"
  - User Bob has connected. User Alice asks for more than 500 letters for "bird".

- Leaked
  - Server key
  - User Password, etc.

**Server Memory Space**

- Has connected. User Bob has connected.
- User Alice wants more letters for "bird".
- Server Master key is 31331498531054, User Carol wants to change password to "Maddog123".
Post Heartbleed Recovery

- Revocation of the compromised keys
- Reissuing and redistributing new keys
  - Contact Certificate Authority
  - Reset passwords after new certificates are authorized

How to check Heartbleed

- Establish Client-Server Handshaking
- Send Heartbleed request*
  - No output return or Return Error → SAFE
  - Return content → UNSAFE

* Code to check Heartbleed implemented in NodeJS programming
  https://github.com/kphongph/heartbleedjs.git

Example of buffer overflow

A program with buffer of size 10 bytes

```c
void foo(char *s) {
    char buf[10];
    strcpy(buf,s);
    printf("buf is %s\n",s);
}
...  
foo
("thisstringistoolongforfoo");
```
# Layout of Program Call

```c
x = 2;
foo(“12345678901”);
```

```c
y = 3;
void foo(char *s) {
    int a, b;
    char buf[1000];
    strcpy(buf, s);
    printf("buf is %s\n", s);
}
```

# Smashing the Stack

**Goal:** To overwrite the Return Address

```c
x = 2;
foo(“12345678901”);
```

```
```c
y = 3;
void foo(char *s) {
    int a, b;
    char buf[1000];
    strcpy(buf, s);
    printf("buf is %s\n", s);
}
```

### Smashing the Stack in Web Server

**Attacker:**
- Generates malicious content (+malicious code) >1000 bytes
- Sends it as the user input to overflow (and take control) the web server

**Example:** Request

```
"user:ASDFBSDFSAF......
&password:...................."
```

```
```

### Smashing the Stack: Example

```javascript
var post_data = querystring.stringify({
    'username': 'ASDFBSDFSAF......', %% > 1000 bytes
    'password': 'XXX123' });

var post_options = {
    host: 'eraider.ttu.edu',
    path: '/authenticate.asp',
    method: 'POST',
    headers: { 'Content-Type': 'application/x-www-form-urlencoded', 'Content-Length': post_data.length },
};

var post_req = http.request(post_options, function(res) {
    res.setEncoding('utf8');
    res.on('data', function(chunk) {
        console.log('Response: ' + chunk);
    });
});

post_req.write(post_data);
```

**How?**

**From source of web page**

**Send data via “post” command**
SQL* Injection Attack

Malicious SQL statements are inserted into a data entry field for execution

* SQL (Structured Query Language) is a programming language for querying and managing data in the database

Login Scenario

**Login Scenario Attack**

SQL syntax:

```
SELECT password
FROM profiles
WHERE user_id = '<user input>'
```

SQL injection attack eliminates Table “profiles”

Assumption: Attacker knows the target, e.g., Table “profiles”

SQL Injection Defense

SQL syntax:

```
SELECT password
FROM profiles
WHERE user_id = 'foo'
```

→ SQL Error since TABLE profiles does not exist

→ Moving Target Defense = Randomly generate TABLE names

What is the target in this example?
Other injection possibilities

Attackers can
• Inject new data to the database, e.g.,
  • Sell politically incorrect items
  • INSERT command to input in the data entry
• Modify data in the database
  • Discount price on an expensive item
  • UPDATE command in the injected SQL
• Gain access to other user’s system by obtaining their passwords

Cross-Site Scripting (XSS)

• Enables attackers to inject client-side script into Web pages viewed by other users
• Enables attackers to bypass access controls from the Client’s policy

  Cross-Site  Scripting

  “Foreign” scripts sent to the client
  Attacker makes Web-Server delivered malicious script

  Cross-Site

  Scripting

  Languages (e.g., JavaScript, VBScript, ActiveX)
  Embedded in Web page (html)
  Executed by Web Browser

Example

Browser
Welcome alice
user_name="alice"

Server
PHP "Welcome ' + user_name;"
Welcome alice

Web Application
**Example of Simple Attack**

```
Welcome
<script>alert('hijack');</script>
```

Browser interprets `<script>alert('hijack');</script>` as the executable script and execute it.

**Simple Attack in Blogging**

```html
POST Forum Message
Subject: GET Money for FREE!!!
Body: <script>alert('hijack')</script>
```

**Impact of XSS attacks**

- **Web server:** Access to authentication credentials for Web applications, Cookies, Username and Password
  → Loss customer trust/reputation/money
- **Web clients:**
  - **Normal Users:** Access to personal data (Credit card, Bank Account), business data, Misuse of the account (e.g., order expensive goods)
  - **High privileged users**
    - Control over Web application
    - Control/Access on Web server machine
    - Control/Access on Backend/Database systems

XSS is a very harmful flaw!
Outline

• Basic Concepts
• Security Types and Perspectives
• Attacks and Threats
• Some Tools and Research Snapshots
• Conclusions

Tools

• Web application scanners:
  • SpikeProxy (Debian Project)
  • WebInspect (SPI Dynamics)
• Penetration Testing tools: conducting security testing for web applications
  • Enable crafting of http request/responses
  • Analysis of session ids, cookies, tokens, etc. for randomness features and
  • WebScarab (OWASP project) Samspade (Samspade)

Counter Measures

Web Server
• HeartBleed
• Buffer Overflow
• SQL Injection

Web Browser
• Social Engineering
  • Phishing
  • Malware
  • CrossSite Scripting

Internet Crime
• Click Fraud

Detection

Goal: Identify malicious URLs to prevent

• Phishing, Spamming and Malicious Download

Approaches:

• Dynamic Analysis: running script and verify if it is safe
  accurate but expensive
• Static Analysis:
  • Compare with URL blacklist (e.g., Phishtank) and
    whitelist (customized)
  • URL Analysis
  • Page Analysis
URL Analysis

Use known attack signatures as heuristics for detecting malicious sites, e.g.,

• **Obfuscated URLs**
  
  ![Example URL](http://193.08.123.30/www.ebay.com/ws/htm)
  
  or more than four periods in directory name space

• **Redirection**
  
  ![Example URL](http://usa.visa.com/track/dyredir.jsp?rDirl=http://200.251.251.10/verified/)

• **Mismatched Domain Names**
  
  ![Example Link](http://secure.bank.com/Ebanking/logon/)

Counter Measures

- **Web Server**
  - HeartBleed
  - Buffer Overflow
  - SQL Injection

- **Web Browser**
  - Social Engineering
  - Phishing
  - Malware

- **Web Application**
  - CrossSite Scripting

- **Internet Crime**
  - Click Fraud

Automotive Click Fraud Detection in Web Advertisement

Rattikorn Hewett & Abhishek Agarwal

ASE International Conference on Cyber Security
Washington D.C.

Motivations

- **Web advertisement is a major source of revenues for online services**
- **Pay-per-click model:** For each click on an advertisement

Broker passively allows fraud for profits

High # clicks ➔ pays

Broker (e.g., Google)

High $ $$

Advertiser

High $ $$

Competitors deplete an advertiser's budget

High $ $$

Publishers earn illegal profit

High $ $$
Click Frauds

• Internet crimes where clicks are deliberately performed with no real interest in the target ad to mimic a legitimate user for generating a charge on the ad

• Detection of click frauds is desirable
  For Broker → To protect reputation
  For Advertisers → To protect revenues

Click Fraud Detection

• Why is it hard?
  • Fraud behaviors evolve over time
  • Frauds can be performed by both humans and software bots → may require distinctive behavioral characteristics
  • Hard to track user identity (IP addresses can change)

• How?
  • Detection at the broker’s side (e.g., Google, Yahoo, etc.)
    • Detection at the advertiser’s side
      → More limited data than that of the broker’s

Our Research

Problem: Given a web server’s log data on an advertiser’s site. Is there any click frauds on this site?

Issues: Most existing automated techniques rely on
  • Signature-based
    → can’t detect unseen new patterns
  • Classifier-based
    → needs a historical training data set that may be outdated
  • Broker server’s data
    → not publically accessible

Proposed Approach

Dempster-Shafer Theory, a Theory of Evidence, that allows:
  • Reasoning about uncertainty based on combined evidences
  • Probability assignment to a set of hypotheses, e.g.,
Proposed Approach

Dempster-Shafer Theory, a Theory of Evidence, that allows:

- Reasoning about uncertainty based on combined evidences
- Probability assignment to a set of hypotheses
- Combination of mass functions (measuring a probability and a belief of a set of hypotheses) from different evidences

Evidence for Click Fraud Detection

Evidence 1: On number of clicks
  High number of clicks in a short time → likely fraud

Evidence 2: On time spent at ad-site
  Spent a long time at the ad-site → likely ~fraud

Evidence 3: On visit patterns
  Visit ad-site via ad click after visit via URL → likely fraud

Evidence 4: On click origin
  Click where advertiser has no business → likely fraud

Evidence 5: On time of click
  Click during suspicious time of the day → likely fraud

Illustration

- Experiments with Synthetic Data containing information on:
  - User IP address (also Google’s ID – eliminate redundancy)
    → infer click location
  - Click number (only in a specified time window)
  - Time of click
  - Page requested
  - Page referred
    → infer whether the visit is via ad or non-ad

Data Sample

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Click No</th>
<th>Time of click</th>
<th>Page</th>
<th>Referrer</th>
</tr>
</thead>
<tbody>
<tr>
<td>172.16.276.3</td>
<td>1</td>
<td>3/5/2012 1:50</td>
<td>index.htm</td>
<td>adsite.htm</td>
</tr>
<tr>
<td>172.16.276.3</td>
<td>2</td>
<td>3/5/2012 1:56</td>
<td>index.htm</td>
<td>adsite.htm</td>
</tr>
<tr>
<td>172.16.276.3</td>
<td>3</td>
<td>3/5/2012 2:01</td>
<td>page1.htm</td>
<td>index.htm</td>
</tr>
<tr>
<td>172.16.276.3</td>
<td>4</td>
<td>3/5/2012 2:07</td>
<td>page2.htm</td>
<td>page1.htm</td>
</tr>
<tr>
<td>172.16.276.3</td>
<td>5</td>
<td>3/5/2012 2:13</td>
<td>index.htm</td>
<td>google.com</td>
</tr>
<tr>
<td>172.16.276.3</td>
<td>6</td>
<td>3/5/2012 2:18</td>
<td>page1.htm</td>
<td>index.htm</td>
</tr>
<tr>
<td>172.16.276.3</td>
<td>7</td>
<td>3/5/2012 2:23</td>
<td>page2.htm</td>
<td>page1.htm</td>
</tr>
<tr>
<td>172.16.276.3</td>
<td>8</td>
<td>3/5/2012 2:33</td>
<td>index.htm</td>
<td>null</td>
</tr>
<tr>
<td>172.16.276.3</td>
<td>9</td>
<td>3/5/2012 2:35</td>
<td>page1.htm</td>
<td>index.htm</td>
</tr>
<tr>
<td>172.16.276.3</td>
<td>10</td>
<td>3/5/2012 2:37</td>
<td>page2.htm</td>
<td>page1.htm</td>
</tr>
</tbody>
</table>
A scenario

Three clicks during three short visits, each is a sec apart
One click during a 4 sec visit via URL with “login” + “shopping”
Four clicks during four short visits, each is a sec apart

Experimental Results

Combined belief values

Limitations

Experiments with synthetic data since no public data available
Incomplete set of mass functions can be extended easily
No comparison with related systems not available
Conclusions & Future work

- Present an automated approach to click fraud detection
  - Based on well-established theory
  - Can detect new unseen patterns
  - On-line computation on new incoming data
  - Does not require large database
  - Easily extensible framework
- Future work: more experiments to gain understanding of
  - Effectiveness of the proposed approach
  - Characteristics of other click fraud behaviors

Counter Measures

Motivations

- Modern enterprises increasingly rely on information systems to provide their functionalities
- Managing access authorities is critical to security of the information systems
- Role-based access control (RBAC) models are widely used to enforce access authorization
**Access Control Models**

Three types of models:

- **Mandatory Models**
  - Prior to run-time: Role Assignment
  - E.g., clerk $\rightarrow$ (Ann, Bob)

- **Discretionary Models**
  - At run-time: Role Activation
  - E.g., clerk $\rightarrow$ Bob

- **Role-based Models**
  - Gain flexibility since users’ roles can be reassigned easily without changing access control

**Issues**

Most existing work in RBAC deals with

- **Specifications** of security constraints
  - The constraints may or may not be satisfied

- Access authorization enforcement assumed by logical inferences at run-time
  - When violation occurs, it may be too late to fix
  - Delays or operation failures

- Logic-based systems that tend to be difficult to understand and do not scale easily
Our research

A systematic technique that supports

• Automated verification of security constraints (instead of specification)

• Analysis for authorization enforcement prior to runtime (i.e., before execution of action/operation, role activation)

• Comprehensibility and scalability by employing set-based algorithms

Separation of Duties (SoDs)

• Common security policy constraints

• Protect frauds by ensuring that no single user receives too many authorities

• Realized by Mutually Exclusive Roles (MER): a set of pairs of conflicting roles

  E.g., (Claim clerk, Auditor) ∈ MER

  Type & print / Approve & sign

  A check payment

  x should not be both a claim clerk & an auditor ⇒ Static SoD

Types of SoD constraints

For any mutually exclusive roles r and r' and any user x

- Strong
  - Static
    - Simple Dynamic
      - Object-based
      - Operational
  - Weak
    - History-based

- For any mutually exclusive roles r and r' and any user x
  - x can’t be both r and r'
  - x can be both r and r' but not at the same time
  - x can be both r and r' at the same time but not on the same object
  - x can be both r and r' at the same time but not on the same operation

- Simple Dynamic (SD)-SoD Verification

  • Input: a design of RBAC information system

  • Output: either yes - the design satisfies SD-SoD or a counter example of role activations causing a violation - to be fixed

  • Basic idea:
    - Identify mutually exclusive roles in MER
    - Annotate each activity of the system operation with authorized access (role, action, objects)
    - For each operation, use MER to search for a valid role activation for each activity in order of the operation path in a depth first search manner

Simple Dynamic (SD)-SoD Verification
A Health Insurance Claim Process:

**Roles:** Clerk, Examiner, Supervisor, Manager, Director

**Actions:** read, write, update

**Objects:** claim-info, insurer-info, EOB, check

**User Role Assignment (UR):**

- **Clerk:** write, claim-info, EOB
- **Examiner:** write, claim-info
- **Manager:** write, claim-info
- **Supervisor:** write, claim-info
- **Director:** write, claim-info

**Mutually exclusive roles:**

- Examiner, Supervisor
- Supervisor, Manager
- Manager, Director
- Clerk, Examiner, Supervisor, Manager

**Simple role hierarchy:**

- Clerk, Examiner, Supervisor, Manager
- Claim Processing Director

**Our approach:**

- Applies the same way when role hierarchy is employed
- Is useful for verification especially when changes occur due to delegation

**Conclusions:**

- Present a systematic approach to analyzing SoD constraints in RBAC information systems
- Advantages of our technique:
  - Supports automated verification of security constraints prior to run-time
  - Changes prior to commitment of role activations are easy to implement → helps prevent failures of critical operations
  - General for reuse in multiple local contexts (e.g., in distributed systems) → provides building block capability for scalable enforcement of access control authorization
Outline

- Basic Concepts
- Security Types and Perspectives
- Attacks and Threats
- Some Tools and Research Snapshots
- Conclusions

Conclusions

- This talk has covered
  - Basic concepts of Web Security
  - Attacks
  - State of the arts
  - Some Research Snapshots
- Not likely to find tools to prevent “Attacks”
- Evolving techniques are necessarily to combat attacks
- Recent trends are to make attackers work harder to attack by moving targets → Moving targets defense