CSE 484 / CSE M 584: Computer Security and Privacy

Autumn 2019

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Franzi Roesner, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
Announcements

• My office hours
  – 11/13 (Wed), 11:30am, CSE1 403
  – 11/20 (Wed), 2:30pm, CSE1 403
  – 11/27 (Wed), None
  – 12/4 (Wed), 12:30pm, CSE1 403

• HW 2 available (due 11/15); extra late day if submitted by Saturday 5pm (11/9)

• Final Project checkpoint on Friday (11/8) (group members, brief description)
  – https://courses.cs.washington.edu/courses/cse484/19au/assignments/final_project.html

• No class on Monday (Veterans Day)
Cross-Site Request Forgery (CSRF/XSRF)
Cross-Site Request Forgery

• Users logs into bank.com, forgets to sign off
  – Session cookie remains in browser state
• User then visits a malicious website containing
  
  <form name=BillPayForm
      action=http://bank.com/BillPay.php>
    <input name=recipient value=badperson> …
  </form>
  
  <script> document.BillPayForm.submit(); </script>

• Browser sends cookie, payment request fulfilled!
• Lesson: cookie authentication is not sufficient when side effects can happen
Cookies in Forged Requests

User credentials automatically sent by browser
XSRF True Story [Alex Stamos]

Internet Exploder

www.cybervillians.com/news.html

Bernanke Really an Alien?

script

Hidden iframes submitted forms that...
- Changed user’s email notification settings
- Linked a new checking account
- Transferred out $5,000
- Unlinked the account
- Restored email notifications

CyberVillians.com

GET news.html

HTML and JS

HTML Form POSTs

StockBroker.com

Java
ticker.stockbroker.com
Impact

• Hijack any ongoing session (if no protection)
  – Netflix: change account settings, Gmail: steal contacts, Amazon: one-click purchase

• Reprogram the user’s home router
  – Change DNS settings (attacker can see/control all DNS responses)

• Login to the attacker’s account
Login XSRF: Attacker logs you in as them!

User logged in as attacker

Attacker’s account reflects user’s behavior
XSRF (aka CSRF): Summary

1. Establish session
2. Visit server
3. Receive malicious page
4. Send forged request

Q: how long do you stay logged on to Gmail? Financial sites?
Broader View of CSRF

• Abuse of cross-site data export
  – SOP does not control data export (we’ve seen this before!)
  – Malicious webpage can initiates requests from the user’s browser to an honest server
  – Server thinks requests are part of the established session between the browser and the server (automatically sends cookies)
XSRF Defenses

- Secret validation token
  
  ```html
  <input type=hidden value=23a3af01b>
  ```

- Referer validation
  
  Referer:
  http://www.facebook.com/home.php
Add Secret Token to Forms

• “Synchronizer Token Pattern”
• Include a secret challenge token as a hidden input in forms
  – Token often based on user’s session ID
  – Server must verify correctness of token before executing sensitive operations
• Why does this work?
  – Same-origin policy: attacker can’t read token out of legitimate forms loaded in user’s browser, so can’t create fake forms with correct token
Referer Validation

- **Lenient** referer checking – header is optional
- **Strict** referer checking – header is required
Why Not Always Strict Checking?

• Why might the referer header be suppressed?
  – Stripped by the organization’s network filter
  – Stripped by the local machine
  – Stripped by the browser for HTTPS → HTTP transitions
  – User preference in browser
  – Buggy browser

• Web applications can’t afford to block these users

• Many web application frameworks include CSRF defenses today
Injection
Injection Attacks

- http://victim.com/copy.php?name=\texttt{username}
- copy.php includes
  \texttt{system(“cp temp.dat $\texttt{name}.dat”)}
- User calls
  http://victim.com/copy.php?name=\texttt{“a; rm *”}
- copy.php executes
  \texttt{system(“cp temp.dat a; rm *.dat”);}
Basic Issues

• User-supplied data is not validated, filtered, or sanitized by application

• User input directly used or concatenated to a string that is used by an interpreter

• Common Injections: SQL, NoSQL, Object Relational Mapping (ORM), LDAP, Object Graph Navigation Library, ...
SQL Injection
Typical Login Prompt

![Typical Login Prompt Image]
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Users " .
    "WHERE Username='".$selecteduser."'";
$rs = $db->executeQuery($sql);

What if ‘user’ is a malicious string that changes the meaning of the query?
User Input Becomes Part of Query

Web browser (Client) → Enter Username & Password

Web server → SELECT passwd FROM USERS WHERE uname IS ‘$user’

DB
Normal Login

- **Web browser (Client)**: Enter Username & Password
- **Web server**: 
  - SQL query: `SELECT passwd FROM USERS WHERE uname IS 'cse484-staff'`
- **DB**
Malicious User Input

![User Login - Microsoft Internet Explorer](image)

- Enter User Name: `'; DROP TABLE USERS; --`
- Enter Password: **Hidden**

CSE 484 / CSE M 584
SQL Injection Attack

SELECT passwd
FROM USERS
WHERE uname IS ‘’;
DROP TABLE
USERS;
--'

Eliminates all user accounts
Security Instruction via XKCD

Hi, this is your son’s school. We’re having some computer trouble.

Oh, dear – did he break something? In a way—

Did you really name your son Robert’); DROP TABLE Students;--?

Oh, yes. Little Bobby Tables, we call him.

Well, we’ve lost this year’s student records. I hope you’re happy.

And I hope you’ve learned to sanitize your database inputs.

http://xkcd.com/327/
SQL Injection: Basic Idea

• This is an input validation vulnerability
  • Unsanitized user input in SQL query to back-end database changes the meaning of query
• Special case of command injection
set UserFound = execute(
    "SELECT * FROM UserTable WHERE
    username=' ' & form("user") & ' ' AND
    password=' ' & form("pwd") & ' ');

User supplies username and password, this SQL query checks if user/password combination is in the database (note: here we’re not thinking about how to actually securely store a password)

If not UserFound.EOF
    Authentication correct
else Fail

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database
Using SQL Injection to Log In

• User gives username ‘ ’ OR 1=1 --
• Web server executes query

```sql
set UserFound=execute(
    SELECT * FROM UserTable WHERE username=‘ ’ OR 1=1 -- ...
);
```

- Always true!
- Everything after -- is ignored!

• Now all records match the query, so the result is not empty ⇒ correct “authentication”!
Preventing SQL Injection

• Validate all inputs
  – Filter out any character that has special meaning
    • Apostrophes, semicolons, percent, hyphens, underscores, ...
    • Use escape characters to prevent special characters from becoming part of the query code
      – E.g.: escape(O’Connor) = O\’Connor
  – Check the data type (e.g., input must be an integer)
Prepared Statements

```java
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, quantity, order_day "
    + "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParameter("month")));
ResultSet res = ps.executeQuery();
```

- **Bind variables**: placeholders guaranteed to be data (not code)
- Query is parsed without data parameters
- Bind variables are typed (int, string, ...)

http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html
Defenses

• Use safe APIs, e.g., prepared statements in SQL with parameterized queries
  – Define all the SQL code, then pass in each parameter
  – Separates code from data
• Whitelist-based server-side input validation
• Escape special characters
• Use LIMIT (and other) SQL controls within queries to prevent mass disclosure of records

• Remember Defense in Depth, Least Privilege, etc.

• Remember OWASP
  https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
  – (though resources now moved elsewhere, this link is to OWASP given value of OWASP in general)
Back to Secure Communications
Problem: How does Alice know that the public key she received is really Bob’s public key?
Threat: Man-In-The-Middle (MITM)
Distribution of Public Keys

• Public announcement or public directory
  – Risks: forgery and tampering

• Public-key certificate
  – Signed statement specifying the key and identity
    • \( \text{sig}_{\text{CA}}(\text{“Bob”}, \text{PK}_B) \)

• Common approach: certificate authority (CA)
  – Single agency responsible for certifying public keys
  – After generating a private/public key pair, user proves his identity and knowledge of the private key to obtain CA’s certificate for the public key (offline)
  – Every computer is pre-configured with CA’s public key
Trusted(?) Certificate Authorities

Keychain Access

Click to unlock the System Roots keychain.

Apple Root CA
Root certificate authority
Expires: Friday, February 9, 2035 at 1:40:36 PM Pacific Standard Time
This certificate is valid

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminCA-CD-T01</td>
<td>certificate</td>
<td>Jan 25, 2016, 4:36:19 AM</td>
</tr>
<tr>
<td>AffirmTrust Commercial</td>
<td>certificate</td>
<td>Dec 31, 2030, 6:06:06 AM</td>
</tr>
<tr>
<td>AffirmTrust Networking</td>
<td>certificate</td>
<td>Dec 31, 2030, 6:08:24 AM</td>
</tr>
<tr>
<td>AffirmTrust Premium</td>
<td>certificate</td>
<td>Dec 31, 2040, 6:10:36 AM</td>
</tr>
<tr>
<td>AffirmTrust Premium ECC</td>
<td>certificate</td>
<td>Dec 31, 2040, 6:20:24 AM</td>
</tr>
<tr>
<td>America Online...cation Authority 1</td>
<td>certificate</td>
<td>Nov 19, 2037, 12:43:00 PM</td>
</tr>
<tr>
<td>America Online...cation Authority 2</td>
<td>certificate</td>
<td>Sep 29, 2037, 7:08:00 AM</td>
</tr>
<tr>
<td>Apple Root CA</td>
<td>certificate</td>
<td>Feb 9, 2035, 1:40:36 PM</td>
</tr>
<tr>
<td>Apple Root CA - G2</td>
<td>certificate</td>
<td>Apr 30, 2039, 11:10:09 AM</td>
</tr>
<tr>
<td>Apple Root CA - G3</td>
<td>certificate</td>
<td>Apr 30, 2039, 11:10:06 AM</td>
</tr>
<tr>
<td>Apple Root Certificate Authority</td>
<td>certificate</td>
<td>Feb 9, 2025, 4:18:14 PM</td>
</tr>
<tr>
<td>Application CA G2</td>
<td>certificate</td>
<td>Mar 31, 2016, 7:59:59 AM</td>
</tr>
<tr>
<td>ApplicationCA</td>
<td>certificate</td>
<td>Dec 12, 2017, 7:00:00 AM</td>
</tr>
</tbody>
</table>
Hierarchical Approach

• Single CA certifying every public key is impractical
• Instead, use a trusted root authority (e.g., Verisign)
  – Everybody must know the root’s public key
  – Instead of single cert, use a certificate chain
    • \( \text{sig}_{\text{Verisign}}(\text{"AnotherCA"}, \text{PK}_{\text{AnotherCA}}), \text{sig}_{\text{AnotherCA}}(\text{"Alice"}, \text{PK}_A) \)

– What happens if root authority is ever compromised?
You encounter this every day...

**SSL/TLS:** Encryption & authentication for connections
Example of a Certificate

GeoTrust Global CA
Google Internet Authority G2

*.google.com
Issued by: Google Internet Authority G2
Expires: Monday, July 6, 2015 at 5:00:00 PM Pacific Daylight Time
This certificate is valid

Details

Subject Name
Country US
State/Province California
Locality Mountain View
Organization Google Inc
Common Name *.google.com

Issuer Name
Country US
Organization Google Inc
Common Name Google Internet Authority G2

Serial Number 6082711391012222858
Version 3

Signature Algorithm SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)
Parameters none

Not Valid Before Wednesday, April 8, 2015 at 6:40:10 AM Pacific Daylight Time
Not Valid After Monday, July 6, 2015 at 5:00:00 PM Pacific Daylight Time

Public Key Info
Algorithm Elliptic Curve Public Key (1.2.840.10045.2.1)
Parameters Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
Public Key 65 bytes: 04 CB DD C1 CE AC D6 20 ...
Key Size 256 bits
Key Usage Encrypt, Verify, Derive
Signature 256 bytes: 34 8B 7D 64 5A 64 08 5B ...
X.509 Certificate
Many Challenges...

- Hash collisions
- Weak security at CAs
  - Allows attackers to issue rogue certificates
- Users don’t notice when attacks happen
  - We’ll talk more about this later in the course
- Etc...

自营邮件: https://mail.google.com/mail/u/0/#inbox
### Colliding Certificates

<table>
<thead>
<tr>
<th>Real Cert</th>
<th>Rogue Cert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Number</td>
<td>Serial Number</td>
</tr>
<tr>
<td>Validity Period</td>
<td>Validity Period</td>
</tr>
<tr>
<td>Domain Name</td>
<td>Domain Name</td>
</tr>
<tr>
<td>RSA Key</td>
<td>????</td>
</tr>
<tr>
<td>X.509 Extensions</td>
<td>X.509 Extensions</td>
</tr>
<tr>
<td>Signature</td>
<td>Signature</td>
</tr>
<tr>
<td><strong>Hash to the same MD5 value!</strong></td>
<td><strong>??</strong></td>
</tr>
</tbody>
</table>

- **Valid for both certificates!**

- **Chosen Prefix** (difference)
- **Collision Bits** (computed)
- **Identical Bytes** (copied from real cert)

[Sotirov et al. “Rogue Certificates”]
Attacking CAs

Security of DigiNotar servers:

• All core certificate servers controlled by a single admin password (Prod@dm1n)
• Software on public-facing servers out of date, unpatched
• No anti-virus (could have detected attack)
Consequences

• Attacker needs to first divert users to an attacker-controlled site instead of the real Google, Yahoo, Skype, but then...
  – For example, use DNS to poison the mapping of mail.yahoo.com to an IP address
• ... “authenticate” as the real site
• ... decrypt all data sent by users
  – Email, phone conversations, Web browsing
Certificate Revocation

• Revocation is very important
• Many valid reasons to revoke a certificate
  – Private key corresponding to the certified public key has been compromised
  – User stopped paying his certification fee to this CA and CA no longer wishes to certify him
  – CA’s private key has been compromised!
• Expiration is a form of revocation, too
  – Many deployed systems don’t bother with revocation
  – Re-issuance of certificates is a big revenue source for certificate authorities
Certificate Revocation Mechanisms

• Certificate revocation list (CRL)
  – CA periodically issues a signed list of revoked certificates
    • Credit card companies used to issue thick books of canceled credit card numbers
  – Can issue a “delta CRL” containing only updates

• Online revocation service
  – When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    • Like a merchant dialing up the credit card processor
Attempt to Fix CA Problems:
Certificate Transparency

• **Problem:** browsers will think nothing is wrong with a rogue certificate until revoked

• **Goal:** make it impossible for a CA to issue a bad certificate for a domain *without the owner of that domain knowing*
  – *(Then what?)*

• **Approach:** auditable certificate logs

[www.certificate-transparency.org](http://www.certificate-transparency.org)
**Attempt to Fix CA Problems:**

**Certificate Pinning**

- **Trust on first access:** tells browser how to act on subsequent connections
- **HPKP – HTTP Public Key Pinning**
  - Use these keys!
  - HTTP response header field “Public-Key-Pins”
- **HSTS – HTTP Strict Transport Security**
  - Only access server via HTTPS
  - HTTP response header field "Strict-Transport-Security"