Getting web authentication right

Joseph Bonneau
jcb82@cl.cam.ac.uk

Security Protocols Workshop
March 28, 2011
A parable of obsolescent technology

Credit: freeyellow.com
Web authentication has evolved very little...

Wall Street Journal, 1996

Wall Street Journal, 2010
Goals for this talk

- An outline for how secure web-based password authentication can be
  - As secure as possible
  - As simple as possible
    - No new software
    - No change to user experience

- How secure is this?
- Why aren’t implementations anywhere close?

---

1But a healthy dose of HTML 5 and other modern tricks
Goals for this talk

- An outline for how secure web-based password authentication can be
  - As secure as possible
  - As simple as possible
    - No new software
    - No change to user experience

- How secure is this?
  - Why aren’t implementations any where close?

---

1 But a healthy dose of HTML 5 and other modern tricks
Goals for this talk

- An outline for how secure web-based password authentication can be
  - As secure as possible
  - As simple as possible
    - No new software
    - No change to user experience

- How secure is this?
- Why aren’t implementations any where close?

---

1But a healthy dose of HTML 5 and other modern tricks
Keyloggers
Phishing
Persistent login cookies

...
Password recovery questions
Password re-use
Password database compromise

...
Cookie stealing
Password guessing
Registration (TLS)

Transmitted:

\[ y = H^{Y}_{\ell_2}(u\|s), \quad x = H^{X}_{\ell_1}(u\|p\|s) \]

Stored:

\[ y = H^{Y}_{\ell_2}(u\|s), \quad z = H^{Z}(u\|x) \]

- **s**: site identifier
- **u**: username
- **p**: password
- **x**: “authenticator”
Login (TLS)

Transmitted:

\[ u, \quad x = H_{\ell_1}^X(u \parallel p \parallel s) \]

Verified to exist in-database:

\[ H^Z(u \parallel x) \]

Returned:

\[ K_u, \quad a = AE_{K_S}(K_u, u, x, t, d) \]

- \( s \): site identifier
- \( u \): username
- \( p \): password
- \( x \): “authenticator”
- \( K_S \): Server master key
- \( a \): session cookie
- \( K_u \): session key
- \( t \): expiration date
- \( d \): additional data
Site interaction (Plain HTTP)

Transmitted as a cookie:

\[ a = \text{AE}_{K_s}(K_u, u, x, t, d) \]

Appended to requests:

\[ \text{AE}_{K_u}(\text{data}) \]

- **s**: site identifier
- **u**: username
- **p**: password
- **x**: “authenticator”
- **K_S**: Server master key
- **a**: session cookie
- **K_u**: session key
- **t**: expiration date
- **d**: additional data
Site interaction (Plain HTTP)

Transmitted as a cookie: **HTTP-only**

\[ a = \text{AE}_{K_s}(K_u, u, x, t, d) \]

Appended to requests:

\[ \text{AE}_{K_u}(\text{data}) \]

- **s**: site identifier
- **u**: username
- **p**: password
- **x**: “authenticator”
- **K_s**: Server master key
- **a**: session cookie
- **K_u**: session key
- **t**: expiration date
- **d**: additional data
Site interaction (Plain HTTP)

Transmitted as a cookie: \text{HTTP-only}

\[ a = \text{AE}_{K_s}(K_u, u, x, t, d) \]

Appended to requests: \text{JavaScript \& HTML5 localStorage}

\[ \text{AE}_{K_u}(\text{data}) \]

- $s$: site identifier
- $u$: username
- $p$: password
- $x$: “authenticator”
- $K_s$: Server master key
- $a$: session cookie
- $K_u$: session key
- $t$: expiration date
- $d$: additional data
Site interaction (Plain HTTP)

Transmitted as a cookie:  

\[ a = AE_{K_s}(K_u, u, x, t, d) \]

Optional cookie:  

\[ a_{secure} = AE_{K_s}(K_u, u, x, t_2 > t, d) \]

Appended to requests:  

\[ AE_{K_u}(\text{data}) \]

- **s**: site identifier
- **u**: username
- **p**: password
- **x**: “authenticator”
- **K_S**: Server master key
- **K_u**: session key
- **a**: session cookie
- **t**: expiration date
- **d**: additional data
Server verification

- Verify & decrypt $a = \text{AE}_{K_s}(K_u, u, x, t, d)$
- Verify & decrypt $\text{AE}_{K_u}(\text{data})$
- Verify that $z = \text{H}^z(u||x)$ is stored (optional)
- Check timestamp $t \geq \text{now}$
- Check ACL for $u, d, \text{data}$

- $s$: site identifier
- $u$: username
- $p$: password
- $x$: "authenticator"
- $K_S$: Server master key
- $a$: session cookie
- $K_u$: session key
- $t$: expiration date
- $d$: additional data
Server verification

- Verify & decrypt \( a = \text{AE}_{K_s}(K_u, u, x, t, d) \)
- Verify & decrypt \( \text{AE}_{K_u}(\text{data}) \)
- Verify that \( z = H^2(u || x) \) is stored (optional)
- Check timestamp \( t \geq \text{now} \)
- Check ACL for \( u, d, \text{data} \)

- \( s \): site identifier
- \( u \): username
- \( p \): password
- \( x \): “authenticator”
- \( K_S \): Server master key
- \( a \): session cookie
- \( K_u \): session key
- \( t \): expiration date
- \( d \): additional data
Performance analysis

- **Login**
  - **Server**
    - 1 hash
    - 1 DB lookup
    - 1 AE + 1 RNG
  - **Browser**
    - 1 iterated hash ($\leq 0.1$ s, PC; $\sim 1$ s, mobile)

- **Interaction**
  - **Server**
    - 2 AE
    - 1 DB lookup (optional)
  - **Browser**
    - 2 AE ($\leq 10$ ms, PC; $\leq 0.1$ s, mobile)
Security analysis—many attacks prevented

- rainbow tables
- online password guessing
- cookie modification
  
  ...  
- session key theft (XSS)
- session cookie theft (sidejacking)
- read-only DB access
- user probing
  
  ...  
- XSS + sidejacking
- DB access + cookie theft
- malware in browser
- password theft
- phishing
- persistent log-in
Security analysis-many attacks prevented

- rainbow tables
- online password guessing
- cookie modification

... 

- session key theft (XSS)
- session cookie theft (sidejacking)
- read-only DB access
- user probing

... 

- XSS + sidejacking
- DB access + cookie theft
- malware in browser
- password theft
- phishing
- persistent log-in
Security analysis—many attacks prevented

- rainbow tables
- online password guessing
- cookie modification
- ... 
- session key theft (XSS)
- session cookie theft (sidejacking)
- read-only DB access
- user probing
- ... 
- XSS + sidejacking
- DB access + cookie theft
- malware in browser
- password theft
- phishing
- persistent log-in
Some sobering facts

- Over 90% of the top 500 websites collect passwords
- 29-50% store them in the clear
- 84% do not prevent brute force attacks at all
- 40% implement TLS correctly (20% incorrectly, 40% not at all)
- hashing in browser, HTTP-only cookies extremely rare...
Even the frameworks get it wrong!

<table>
<thead>
<tr>
<th>Language</th>
<th>Framework</th>
<th>Plugin</th>
<th>ver.</th>
<th>Algorithm</th>
<th>Iteration count</th>
<th>Salt (bits)</th>
<th>Output (bits)</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NET</td>
<td>ASP.NET</td>
<td></td>
<td>4</td>
<td>SHA-1</td>
<td>1</td>
<td>none</td>
<td>160</td>
<td>also supports cleartext storage</td>
</tr>
<tr>
<td>PHP</td>
<td>built-in</td>
<td></td>
<td>5.3</td>
<td>MD5</td>
<td>1,000</td>
<td>72</td>
<td>132</td>
<td>MD5 crypt()</td>
</tr>
<tr>
<td>PHP</td>
<td>CakePHP</td>
<td></td>
<td>7</td>
<td>SHA-1</td>
<td>1</td>
<td>none</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>PHP</td>
<td>Drupal</td>
<td></td>
<td>7</td>
<td>SHA-512</td>
<td>16,384</td>
<td>48</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>PHP</td>
<td>Joomla!</td>
<td></td>
<td>1.5</td>
<td>MD5</td>
<td>1</td>
<td>48</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>PHP</td>
<td>WordPress</td>
<td></td>
<td>3.1</td>
<td>Blowfish</td>
<td>256</td>
<td>48</td>
<td>132</td>
<td>uses PHPPass</td>
</tr>
<tr>
<td>Python</td>
<td>Django</td>
<td></td>
<td>1.2</td>
<td>SHA-1</td>
<td>1</td>
<td>20</td>
<td>160</td>
<td>also supports unsalted MD5</td>
</tr>
<tr>
<td>Python</td>
<td>generic WSGI</td>
<td>repoze.w</td>
<td>2.0</td>
<td>SHA-1</td>
<td>1</td>
<td>none</td>
<td>160</td>
<td>Recommended for Pylons</td>
</tr>
<tr>
<td>Ruby</td>
<td>Rails</td>
<td>restful_auth</td>
<td></td>
<td>SHA-1</td>
<td>10</td>
<td>160</td>
<td>160</td>
<td>salt has only 80 bits of entropy</td>
</tr>
</tbody>
</table>
Is it worthwhile to fix password authentication?

jcb82@cl.cam.ac.uk