

# 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...

 **Choose a User Name and Password**

Choose a User Name that's no more than 15 characters and doesn't include punctuation, symbol characters or spaces. You'll need this each time you sign on to the Interactive Journal. →

Choose a Password, which you'll also enter each time you use this service. Your password should be 5-15 characters in length and shouldn't include punctuation, symbol characters or spaces. →

**Important:** We'll record your User Name and Password EXACTLY as you type them, so make a note if you enter in upper and lower case.

**User Name:**

**Password:**

**Re-enter Password:**

Wall Street Journal, 1996

Please register to gain free access to WSJ tools.

First Name  Last Name

Email (your email address will be your login)

Confirm Email

Create a Password  Confirm Password

From time to time, we will send you e-mail announcements on new features and special offers from The Wall Street Journal Online.

[REGISTER NOW ▶](#)

[Why Register? ▼](#) [Privacy Policy](#) | [Terms & Conditions](#)

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<sup>1</sup>
    - No change to user experience
- How secure is this?
- Why aren't implementations any where close?

---

<sup>1</sup>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<sup>1</sup>
    - No change to user experience
- How secure is this?
- Why aren't implementations any where close?

---

<sup>1</sup>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<sup>1</sup>
    - No change to user experience
- How secure is this?
- Why aren't implementations any where close?

---

<sup>1</sup>But a healthy dose of HTML 5 and other modern tricks

# How password authentication goes wrong

- Keyloggers
- Phishing
- Persistent login cookies
- ...
- Password recovery questions
- Password re-use
- Password database compromise
- ...
- Cookie stealing
- Password guessing

# Registration (TLS)

Transmitted:

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

Stored:

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

- **s**: site identifier
- **u**: username
- **p**: password
- **x**: “authenticator”



# Login (TLS)

Transmitted:

$$u, \quad x = \mathbf{H}_{\ell_1}^X(u||p||s)$$

Verified to exist in-database:

$$\mathbf{H}^Z(u||x)$$

Returned:

$$K_u, \quad a = \mathbf{AE}_{K_S}(K_u, u, x, t, d)$$

- $s$ : 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:

$$a = \mathbf{AE}_{K_S}(K_U, u, x, t, d)$$

Appended to requests:

$$\mathbf{AE}_{K_U}(\text{data})$$

- $s$ : 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

$$a = \mathbf{AE}_{K_S}(K_U, u, x, t, d)$$

Appended to requests:

$$\mathbf{AE}_{K_U}(\text{data})$$

- $s$ : 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

$$a = \mathbf{AE}_{K_S}(K_U, u, x, t, d)$$

Appended to requests: JavaScript & HTML5 localStorage

$$\mathbf{AE}_{K_U}(\text{data})$$

- $s$ : 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

$$a = \mathbf{AE}_{K_S}(K_U, u, x, t, d)$$

Optional cookie: HTTP-only, SECURE

$$a_{\text{secure}} = \mathbf{AE}_{K_S}(K_U, u, x, t_2 > t, d)$$

Appended to requests: JavaScript & HTML5 localStorage

$$\mathbf{AE}_{K_U}(\text{data})$$

- $s$ : 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

# Server verification

- Verify & decrypt  $a = \mathbf{AE}_{K_s}(K_u, u, x, t, d)$
  - Verify & decrypt  $\mathbf{AE}_{K_u}(\text{data})$
  - Verify that  $z = \mathbf{H}^Z(u||x)$  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  $a = \mathbf{AE}_{K_s}(K_u, u, x, t, d)$
  - Verify & decrypt  $\mathbf{AE}_{K_u}(\text{data})$
  - Verify that  $z = \mathbf{H}^Z(u||x)$  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

- 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!

Default parameters in common web frameworks and CMSes

Language	Framework	Plugin	ver.	Algorithm	Iteration count	Salt (bits)	Output (bits)	notes
.NET	<a href="#">ASP.NET</a>		4	SHA-1	1	none	160	also supports cleartext storage
PHP	<a href="#">built-in</a>		5.3	MD5	1,000	72	132	MD5 crypt()
PHP	<a href="#">CakePHP</a>		7	SHA-1	1	none	160	
PHP	<a href="#">Drupal</a>		7	SHA-512	16,384	48	256	
PHP	<a href="#">Joomla!</a>		1.5	MD5	1	48	128	
PHP	<a href="#">WordPress</a>		3.1	Blowfish	256	48	132	uses <a href="#">PHPPass</a>
Python	<a href="#">Django</a>		1.2	SHA-1	1	20	160	also supports unsalted MD5
Python	generic WSGI	<a href="#">repoze.who</a>	2.0	SHA-1	1	none	160	Recommended for <a href="#">Pylons</a>
Ruby	Rails	<a href="#">restful_auth</a>		SHA-1	10	160	160	salt has only 80 bits of entropy

# Is it worthwhile to fix password authentication?

jcb82@cl.cam.ac.uk