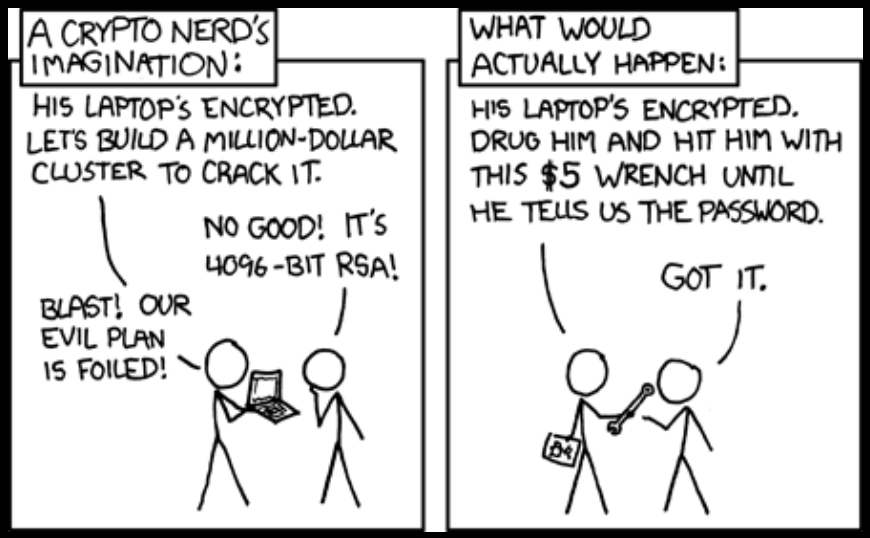
hackH4RV4RD

Web Application Security

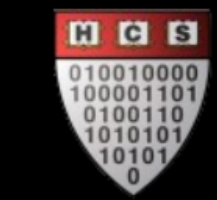


Sources of Risk

- Trusting the browser/user
  → XSS, CSRF
- Confused delimiters
  → XSS, SQL Injection, buffer overflow
- Insecure storage
  → Password compromise
- Insecure channels
  → Session hijacking

Delimiting strings

- Quoting, whether characters or tags
  e.g. HTML, SQL, many languages
  "Hello, world!"
  Hello, world!
  - Requires escaping
    - Backslash: 'Don\'t panic!'
    - Encoding: "5 > 3"



Delimiting strings

- Encode the content
  Base64: SGVsbG8sIHdvcmxkIQ==
  URLencode: http://foo.bar/a%2Fb
- Declare the length
  e.g. C arrays — vulnerable to overrun
  char buf[13];
- Use a special symbol at the end
  e.g. C strings, end of transmission
  Hello, world!\0

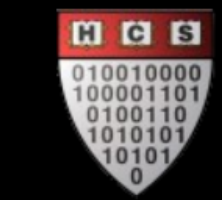
XSS Attacks

- “Cross-Site Scripting Attacks.”
- The injection of malicious client-side scripts (such as javascript) into pages viewed by other users
- Ex:

```
<div id='post'>
   <script>alert("Hello!")</script>
</div>
```

XSS Attacks

- Who cares? It won't happen to me!
- Successful exploits against most major web companies at some time, including:
  - Google, Facebook, Twitter, MySpace
- Attackers can write worms, steal information, impersonate others, etc.
- XSS now outnumbers buffer overrun as the most exploited flaw on the Internet.



XSS Attacks

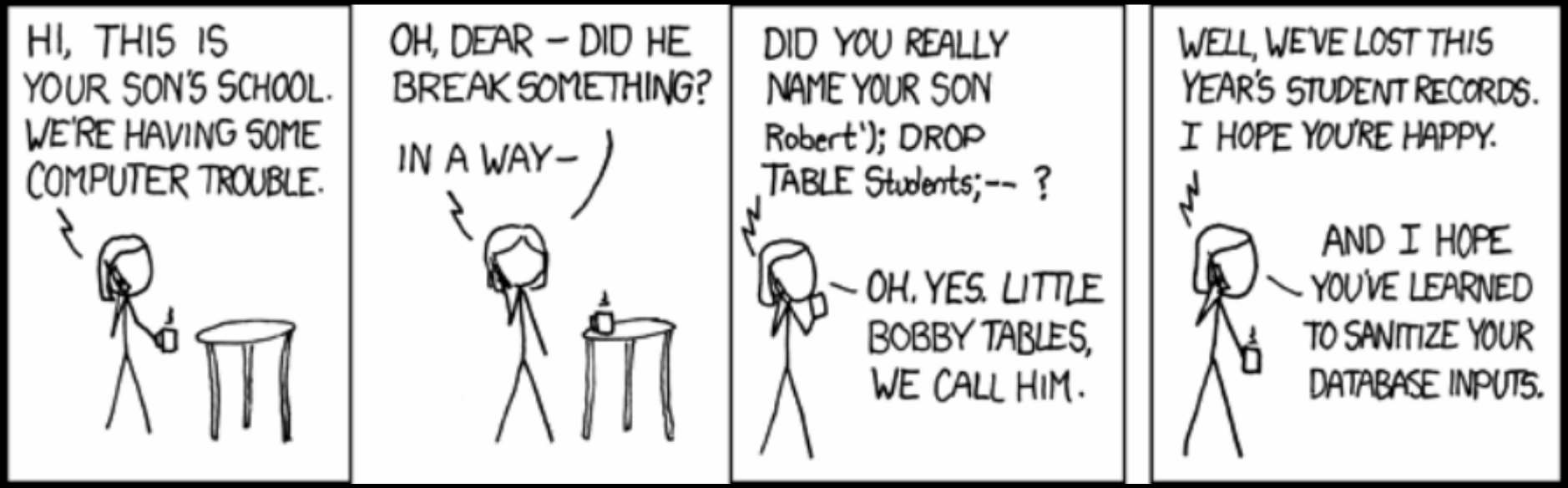
- How to protect yourself:
  - Escape all output destined for HTML!
  - Filter "javascript:" calls from attributes!
- Web frameworks and templating languages usually have built-in support.
- In plain PHP:

```
htmlspecialchars($s, ENT_QUOTES)
rawurlencode($url)
```

SQL Injection

- When input is not sanitized, and the input is used to execute queries directly, you can inject SQL into the query.
- For instance
  `"SELECT * FROM users WHERE username='$user'"`
- What happens when:
  `$user = " '; DROP TABLE users; --"`

SQL Injection



SQL Injection

- PHP is not so friendly here. Be careful!
  - Disable magic quotes if it's enabled.
  - Use parameterized queries if at all possible, using mysqli or PDO.
    - It may seem like more work at first, but it's better than hundreds of mysql_real_escape_string() calls.

SQL Injection

```
$db = new PDO("mysql:host=localhost;
dbname=db", $user, $pw);
```

Using parameters:

```
$q = "SELECT * FROM users WHERE name=?";
$c = $db->prepare($q);
$c->execute(array('Andy'));
print_r($c->fetch());
```

A simple query with no parameters:

```
$q = "SELECT * FROM posts";
foreach ($db->query($q) as $row) {...}
```

CSRF

- You are authenticated (via a session cookie) on some site.
- Cross-Site Request Forgery is a request to the site triggered from a different site.
- Examples using GET and POST:

```
<img src="https://bank.com/withdraw?amount=100000"
/>
<form action="https://bank.com/withdraw" method="
post">...</form>
```

CSRF


i'm in ur browzer
stealin ur cookiez
ICANHASCHEEZBURGER.COM

CSRF

- This is an example of a confused deputy.
  - No amount of encryption or input validation will protect you.
- In *every form* on your site, add a hidden token that is specific to each user's session. Only a genuine form that came from your site will be able to send it back. If you get a POST without the token, it's illegitimate.
- Still doesn't protect against clickjacking.

Insecure Channels

- Encryption is great. Use it!
  - CSRF protection is basically moot if you're not using SSL.
  - Don't fall prey to Firesheep & friends.
- If you only encrypt the login page, you're still vulnerable to session hijacking.
  - How important is this? It depends on how damaging impersonation would be.

Insecure Channels

Insecure Storage

- Never store passwords in plaintext.
  - Use a hashing function like SHA1.
  - Even better, use a salted hash.
    - It's an easy drop-in replacement. Just search for "salted sha1".
    - Gawker used a weak hash that is easily brute-forced. Don't do that!

Insecure Storage

- Most popular gawker passwords:

```
4162  123456
3332  password
1444  12345678
 861  lifehack
 765  qwerty
 529  abc123
 503  12345
 471  monkey
```

General Principles

- Don't ever trust the user/browser.
  - Sanitize *all* data from the user.
- Avoid security by obscurity.
- In general, use libraries.
- Frameworks can help prevent XSS and CSRF attacks.
- Keep your software up-to-date.

hack Harvard

http://agb.me/blag/

source: https://github.com/abrody/blag

hack Harvard

CSRF:
http://hcs.harvard.edu/abrody/csrf.html

Further reading:
- http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
- Wikipedia pages for XSS, CSRF, SQL Injection, etc.
- http://chadselph.com/wget/Slidy/csg_www2.html
- http://en.wikipedia.org/wiki/Samy_(XSS)
- http://www.php.net/manual/en/pdo.prepare.php