When Web 2.0 Attacks!

Understanding Ajax, Flash and other highly interactive web technologies…

Rafal M. Los
AppSec Specialist
Hewlett-Packard, ASC
Rafal@HP.com
+1 (404) 606-6056
http://twitter.com/RafalLos

AppSec DC
November 10-13th, 2009

The OWASP Foundation
http://www.owasp.org

Fire! … Aim! Ready?

Question 1: Web 2.0 content is being developed primarily by the same developers that write traditional web code. True or False?

Question 2: Everyone understands the idea of "Web 2.0" and there are concrete standards. True or False?

Question 3: Your company has deployed "Web 2.0 stuff" already. True or False?

Answers…

Question 1: False! Web 2.0 is being developed in a large part not by traditional developers, but by "marketing or media folks"…

Question 2: False! Ask 2 different people to define "Web 2.0"… listen to their answers.

Question 3: (*most likely*) True! … and if you don't know it, it's even worse.

Browser Evolution

Render *simple* HTML content

Render *complex, synchronous* content

Render *complex, asynchronous* content

Perform *complex, asynchronous* interactions

Perform *complex, asynchronous, offline* interactions

online

Let's start by thinking offensively

Understanding Web 2.0 Motivations

2 reasons "Web 2.0" happened…

1. Processing power requirement moved off to client
2. Decrease bandwidth required for interactions

What happened…

- Logic moved from server → client
- Invention of asynchronous transaction
- The "offline web" application

Examples – What Could Possibly Go Wrong?

…

what could possibly go wrong?

- Manipulation of business logic
- Client-side data validations
- Exposure of sensitive information

→ *so why bother with XSS, SQLi?*

Client-Side Logic Manipulation

```
try {
            strURI = ExternalInterface.call("getLittleServer");
            nGameId = gameID;
            nScore = score;
            nTime = ExternalInterface.call("getSrvrTime");
            strTime = toString();
            strN1 = substr(253, 3);
            strN2 = substr(252, 3);
            n1 = parseInt(strN1);
            n2 = parseInt(strN2);
            nAlgo = n1 * n2 * nScore + nScore;
            strToPass = nGameId + "," + nScore + "," + nTime + "," + nAlgo;
            encrypted_data = MD5.hash(strToPass);
            submission_data = "score=" + nScore + "|gameId=" + nGameId + "|timestamp=" + nTime + "|key="
    + encrypted_data;
            variables = new URLVariables();
        variables.attr1 = submission_data;
            request = new URLRequest(strURI);
        request.data = variables;
        navigateToURL(request, "_self");
        return submission_data;
    …
```

Examples – What Could Possibly Go Wrong?

…

what ^else could possibly go wrong?

- Manipulation of business logic
- Client-side data validations
- Exposure of sensitive information

→ *so why bother with XSS, SQLi?*

Client-Side Data Validations

```
...
button 9 {

    on (release, keyPress '<Enter>') {
      if (password eq ` PASSWORD ') {
        getURL('http://www.SomeCompany.tld/client_pages/CUSTOMER_REMOVED/778.html', '');
      } else {
        if (password eq ' PASSWORD ') {
          getURL('http://www.SomeCompany.tld/client_pages/CUSTOMER_REMOVED/781.html', '');
        } else {
          if (password eq ' PASSWORD ') {
            getURL('http://www.SomeCompany.tld/client_pages/CUSTOMER_REMOVED/783.html', '');
          } else {
            if (password eq ` PASSWORD ') {
              getURL('http://www.SomeCompany.tld/client_pages/CUSTOMER_REMOVED/771.html', '');
            } else {
              if (password eq ` PASSWORD ') {
                getURL('http://www.SomeCompany.tld/client_pages/CUSTOMER_REMOVED/799.html', '');
              } else {
...
```

Examples – What Could Possibly Go Wrong?

…

what ^else could possibly go wrong?

- Manipulation of business logic
- Client-side data validations
- Exposure of sensitive information

→ *so why bother with XSS, SQLi?*

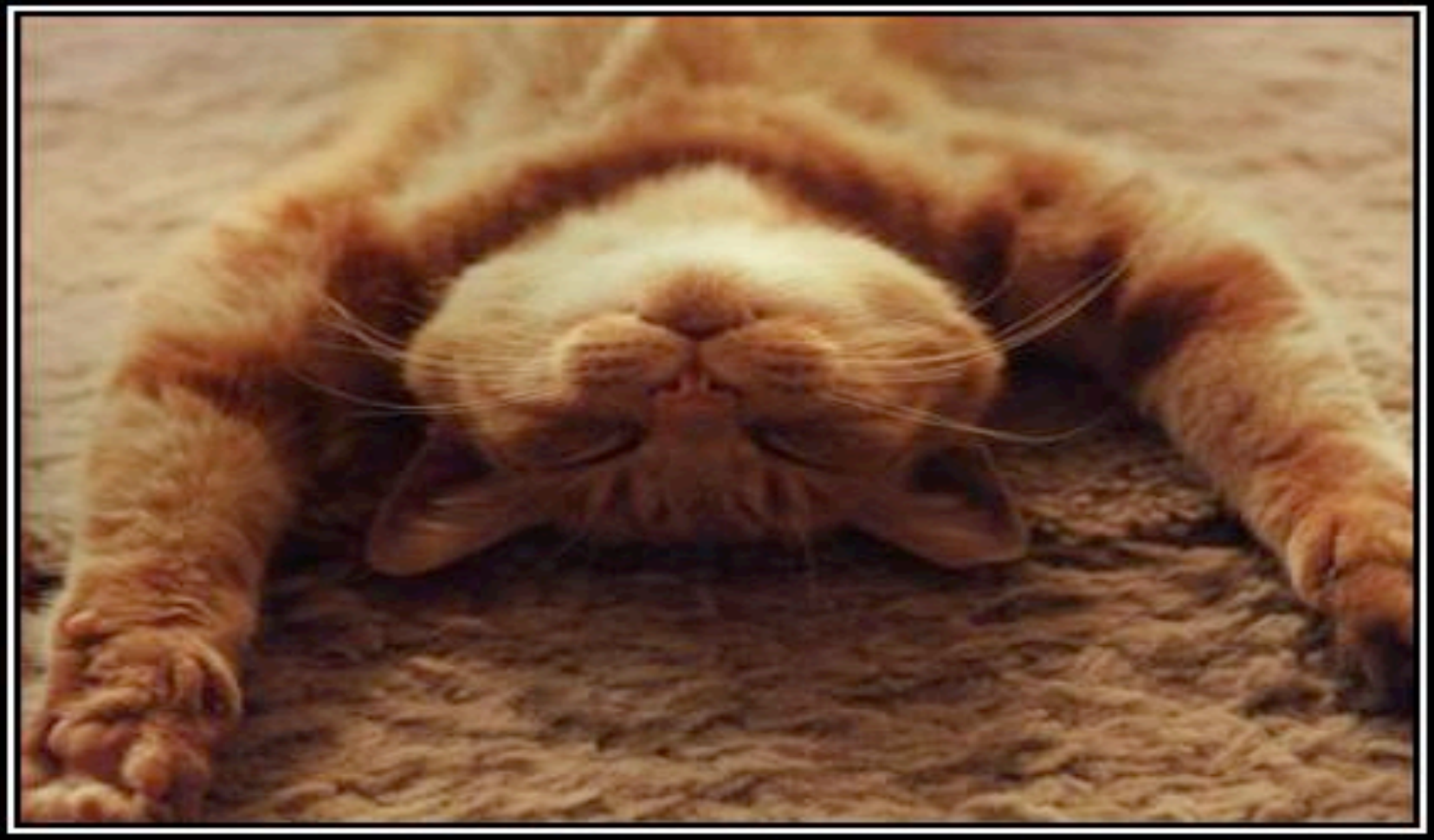
Thinking Web 2.0 Offense

```
private static function query(arg0:String, arg1:flash.events::EventDispatcher = null)
   {
      st = null;
      token = null;
      statement = arg0;
      dispatcher = arg1;
      trace("2:MySQL Query: " + statement);
      if(this.connection == null)
      {
         try {
            this.connection = new Connection(irrcrpt("dqgurjudgh.frp", 3), 3306, irrcrpt("icog_nqikp", 2),
     irrcrpt("d1su4y", 1), irrcrpt("jdph", 3));

         } catch (e:SecurityError) {
            var loc1:* = e;
                  statement = null;
            Alert.show(statement.message, "Security Error");
            if(dispatcher)
            {
               dispatchEvent(new Event(Event.CANCEL));
            }
            return;
      }
```

Let's decompile some flash!

… wait, I thought you couldn't do that!

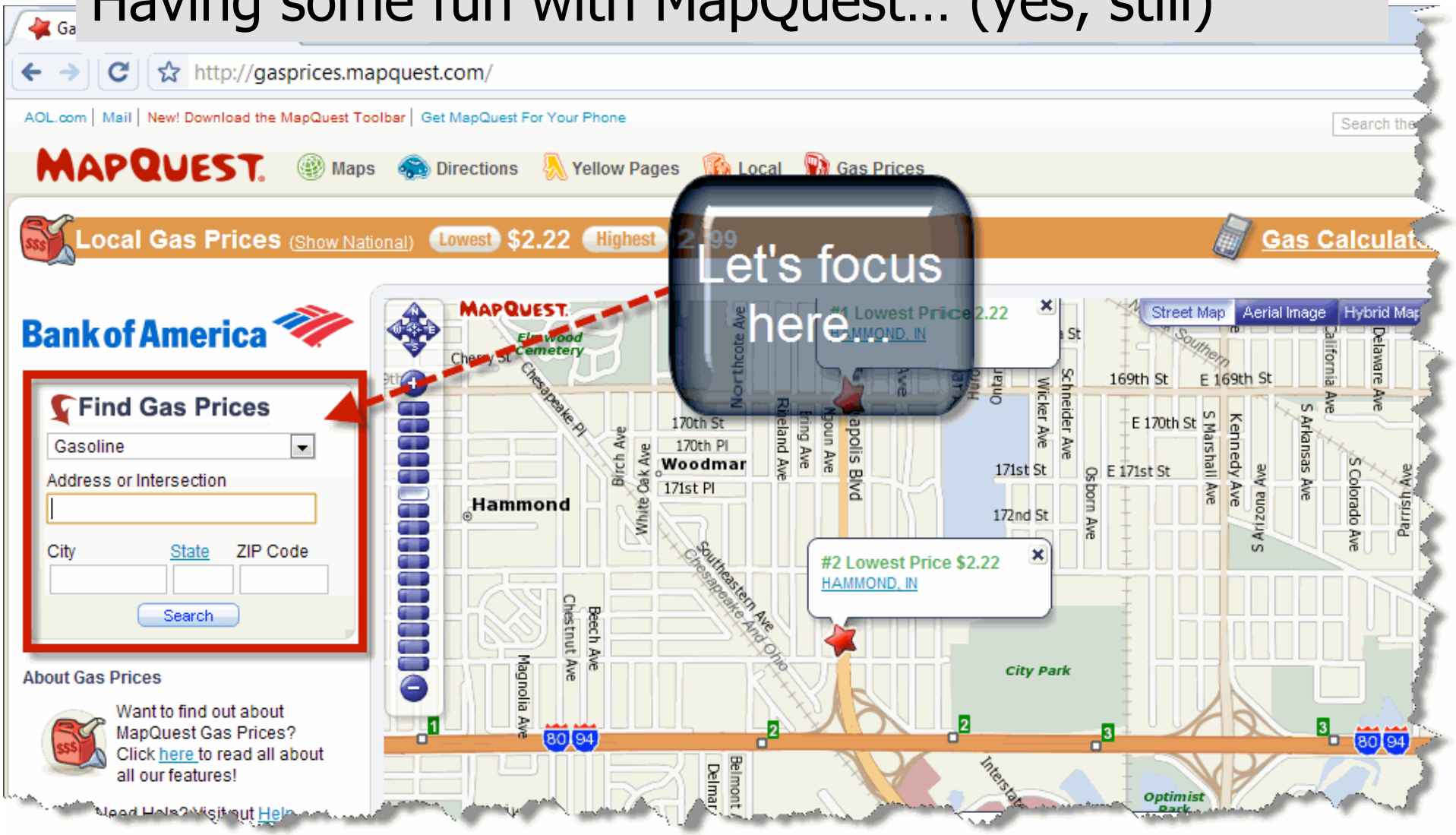


SURRENDER

The hacker always wins anyway...

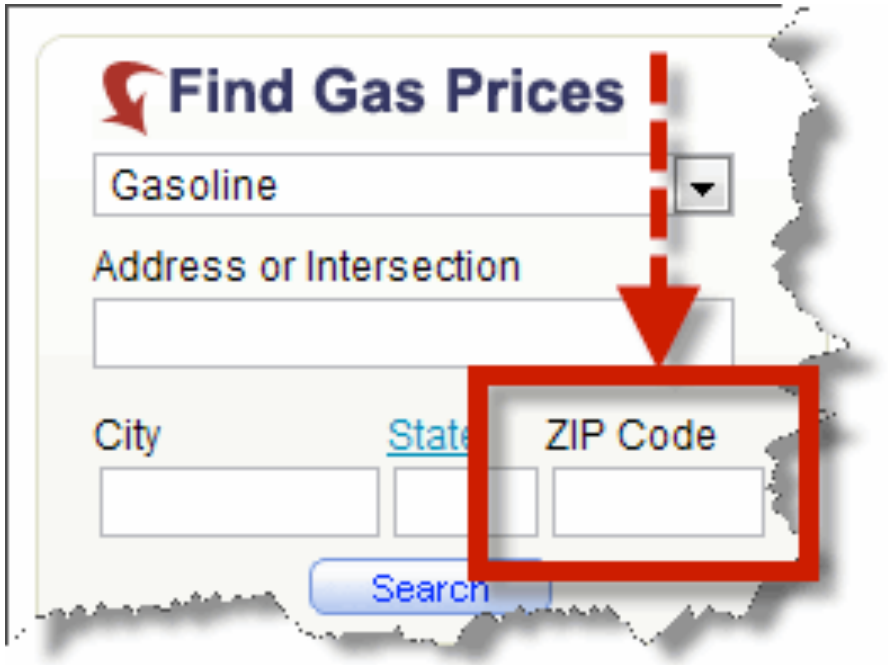
Attacking Web 2.0 Sites

Having some fun with MapQuest… (yes, still)



Attacking Web 2.0 Sites

Having some fun with MapQuest… (yes, still)



We insert the infamous iFrame

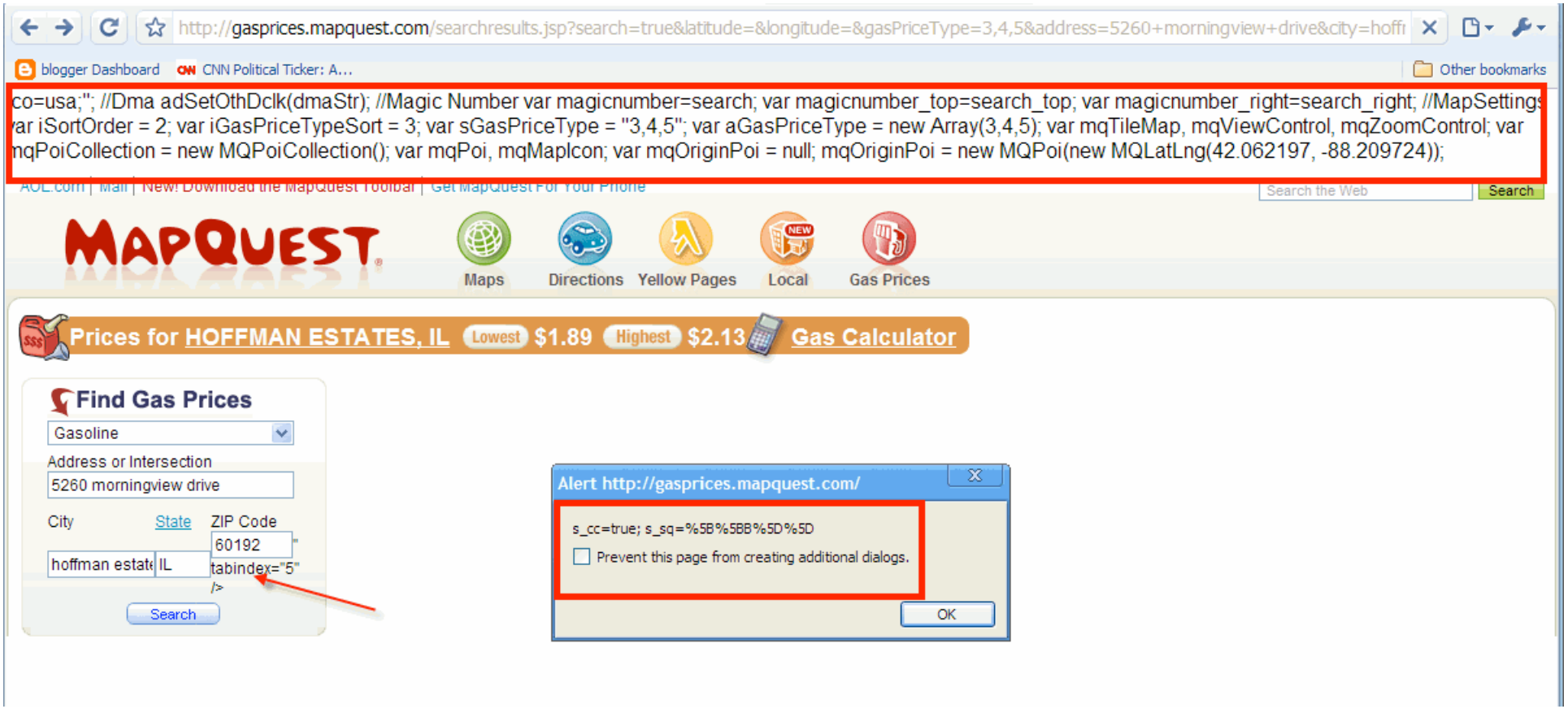
</iframe><script>
alert(document.cookie)
</script>

Let's ENCODE it to get past black-listing filters…

%22%3e%3cframe%20src%3dhttp%3a%2f%2fgoogle.com%3e
%3c%2fiframe%3e%3cscript%3ealert(document.cookie)%3c
%2fscript%3e

Attacking Web 2.0 Sites

… and then this happens!



What Did We Just Learn?

Web 2.0 isn't some magical new "thing"; it's a conglomeration of old technologies…

…and yes, all the old bugs are back.

The HTML v5 Specification

Standards rule.

Consider this…

- ✓ ClickJacking was an *abuse of standards*
- ✓ HTML v5 now has local database specification
- ✓ HTML v5 has an offline application specification
- ✓ HTML v5 is *so big* few people have read it all

Specification for Offline Web Apps

From W3.org → http://www.w3.org/TR/offline-webapps/

Users of typical online Web applications are only able to use the applications while they have a connection to the Internet. When they go offline, they can no longer check their e-mail, browse their calendar appointments...

The HTML 5 specification provides two solutions to this: a SQL-based database API for storing data locally, and an offline application HTTP cache for ensuring applications are available even when the user is not connected to their network.

Mechanisms for Offline Apps

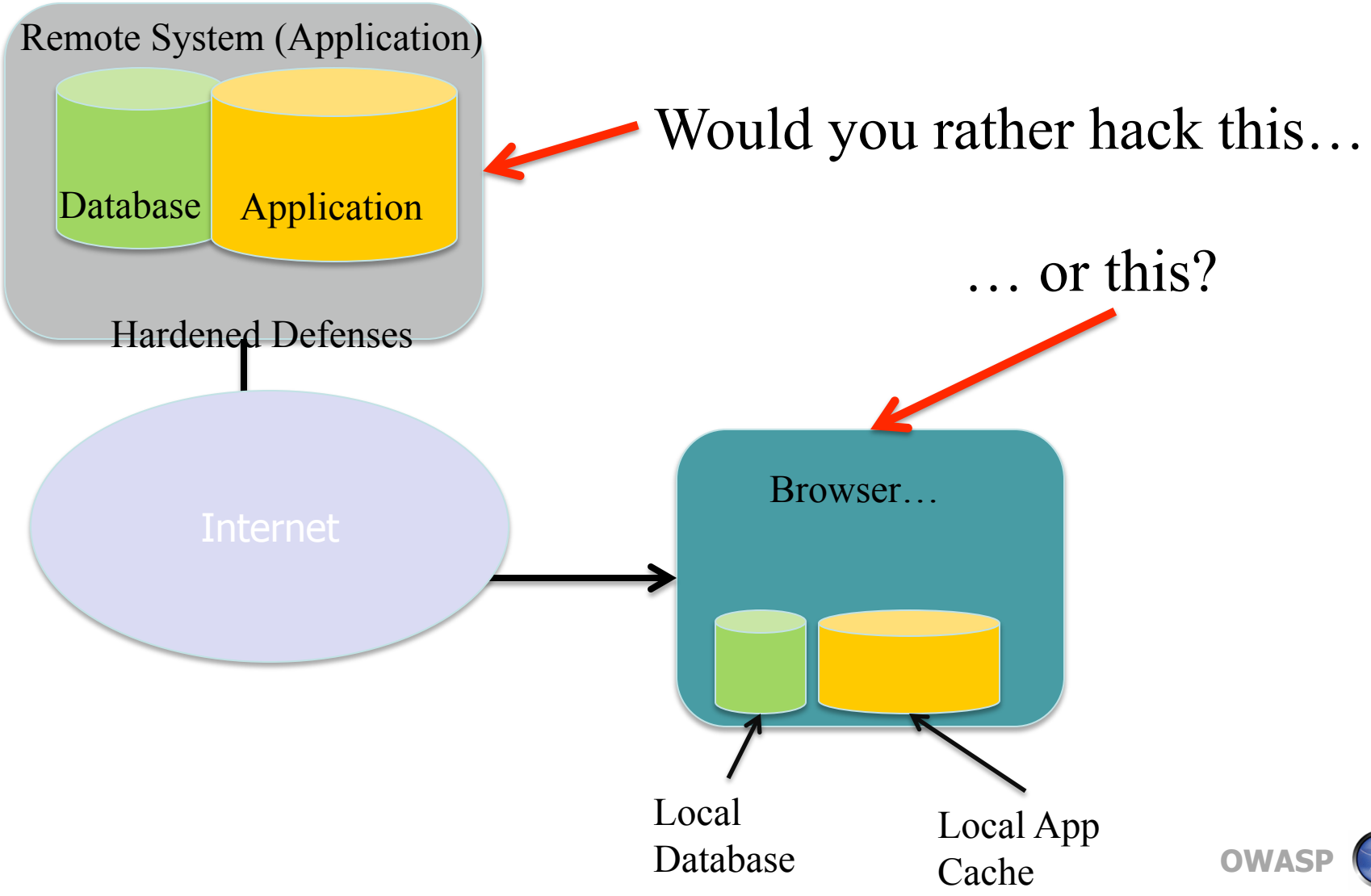
SQL-based database API for storing data locally

and a

offline application HTTP cache

What could *possibly* go wrong?

Implementing Offline App Concepts



Simple Problems with Offline Apps

Online Application	Offline Application
Remote data storage	Local data storage
Enterprise DB typically "secured"	Local DB "forgotten"
Enterprise DB difficult to access	Local DB … on local filesystem
Attack trips security mechanisms	No local security mechanisms
Remote Logic	Local "Cached" Logic
Manipulate at run-time, remotely	Manipulate code, locally
Remote validation of logic	Fully control/manipulate logic

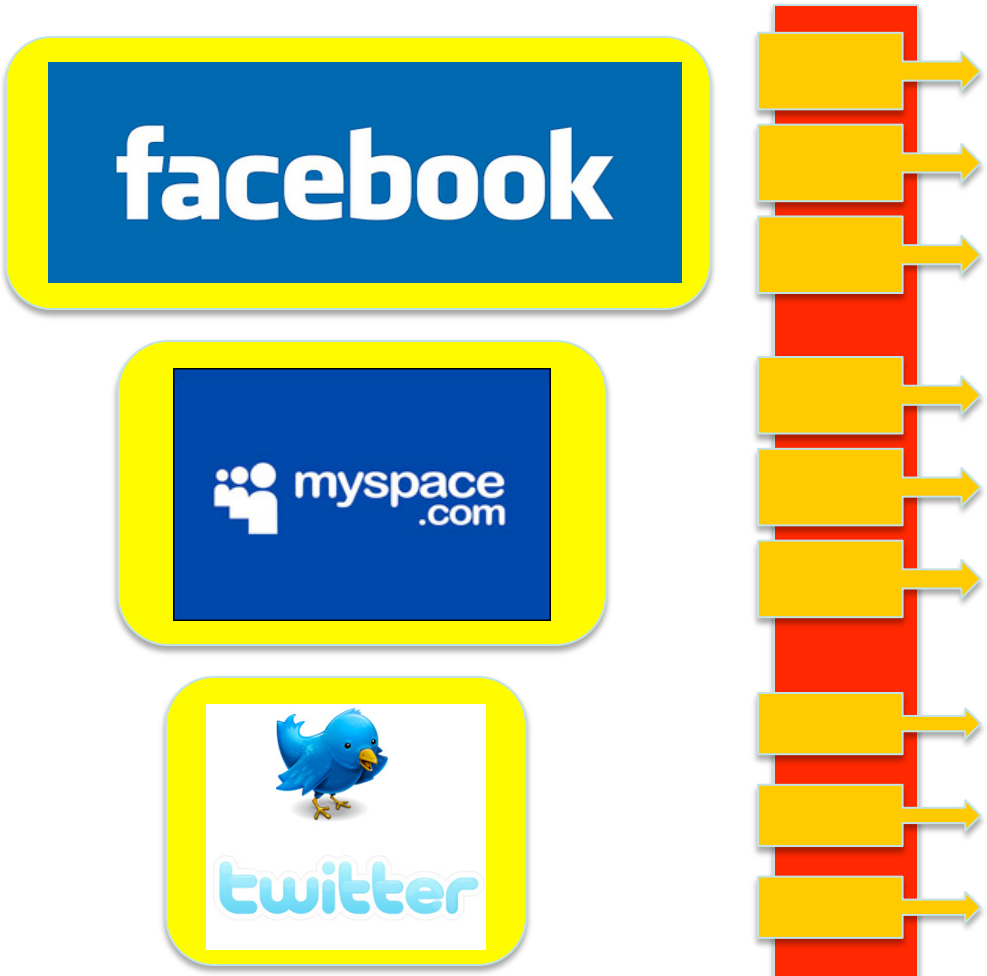
Then Came Social Media…

First, came the applications…

They were attacked.

Then they were hardened.

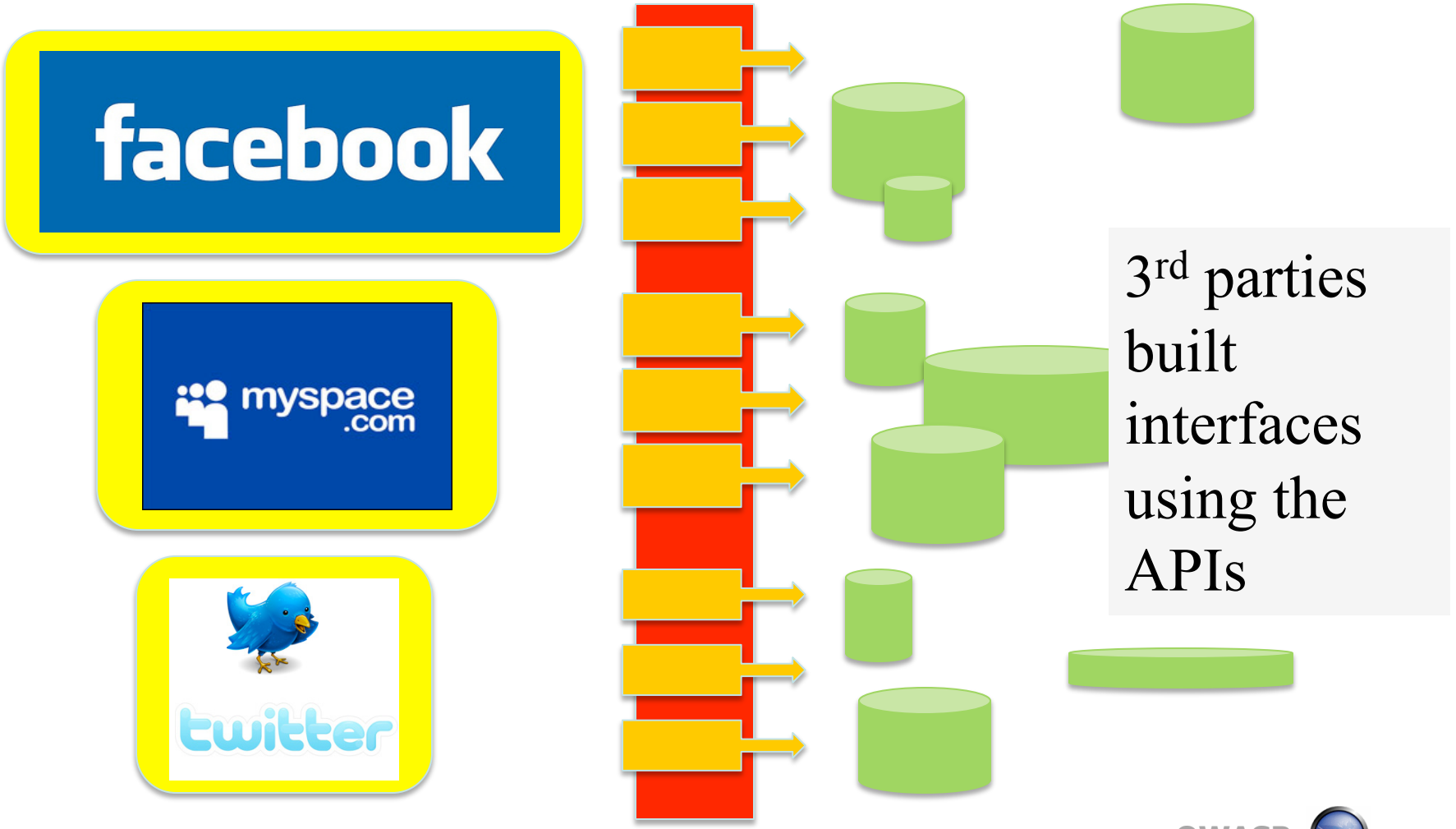
Users Demanded More



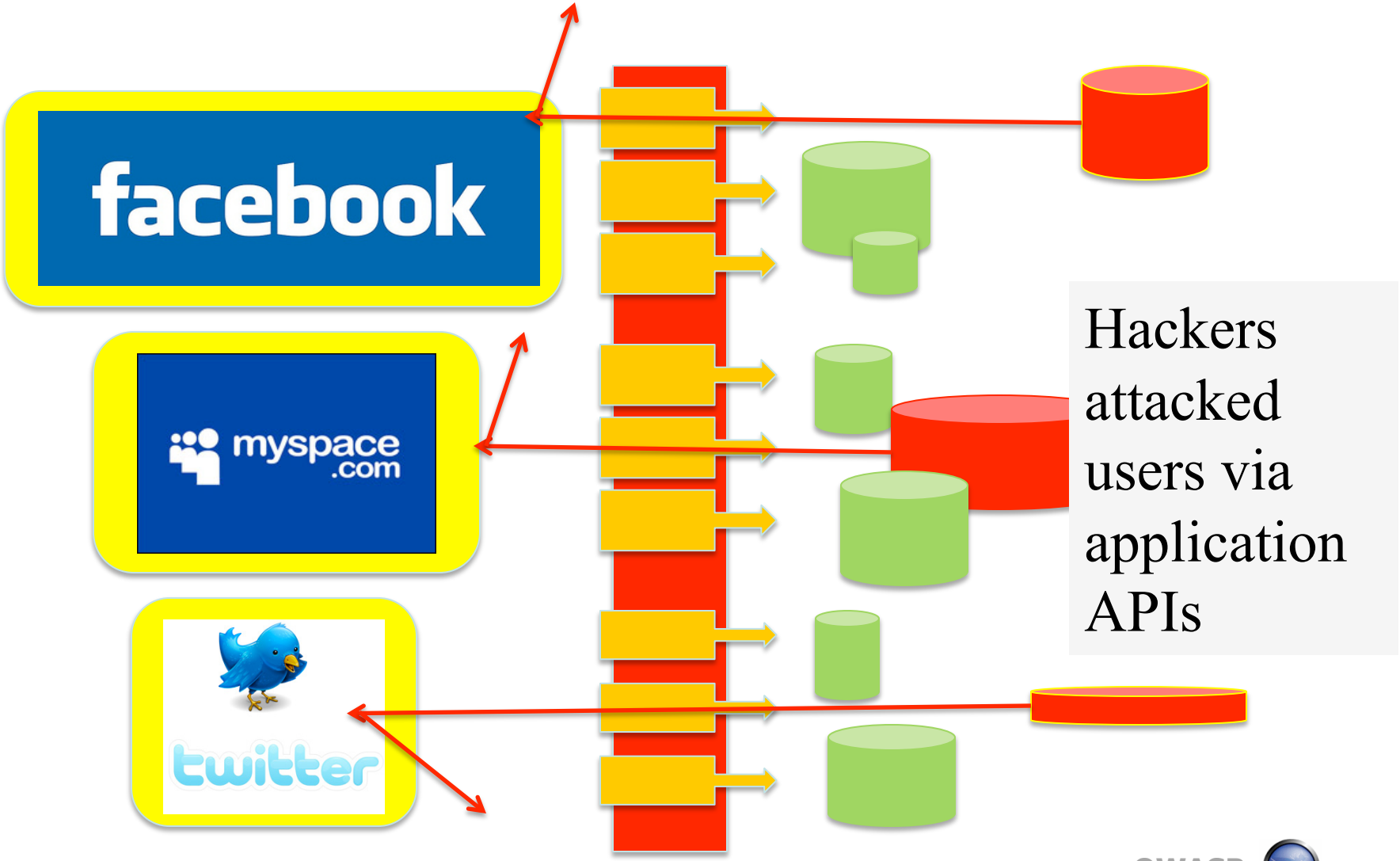
Users wanted more.

Applications were extended via APIs.

Social Sites Were Extended…



Hackers Exploited Extensions/APIs



Web 2.0 Attacked Via Extension

FaceBook *still* fighting worms and hacks against users via extensions (or plug-ins) built using legal API extensions (Koobface?)

Twitter API continually being abused by worms and "bots" to spam and seed trojan malware

Why attack a hardened resource/site when a hacker can use APIs to write malicious plug-ins?

So what do we do about it?!

The 3½ Keys to Success

- Perform all control logic server-side
- Validate all data at ingress & egress
- Build zero-trust interfaces

… and remember, "the user will always choose dancing bears over security". -Schnier

Perform All Control Logic Server-Side

Application-critical logic must always be performed on the server side, where it is less likely to be manipulated

- Remember you can never trust code once it leaves your control
- Web code can and will be reverse-engineered (flash, java, etc)
- Never push critical information (passwords, connection strings) to the client

Validate All Data at Ingress/Egress

Validate all data as it comes into your application, and also as it leaves

- Validate every single piece of data, always
- Mix white-list and black-list, focusing on minimum required data sets
- Make sure you know what's leaving your application…

Build Zero-Trust Interfaces

Assume the APIs or web-services you expose will be attacked

- Never trust the interface to provide clean data, legal calls, or valid requests
- Authenticate interfaces when ever possible
- Never trust your own code once it's in the user's browser (least-privilige)
- Adopt the mentality of …"If you were sticking your hand into a dark, unknown box"

Save the User, Save the World

Usable security is a myth on the web.

Web 2.0+ focuses on usability, over security.

“Cool” wins over “secure” every time.

Never trust to user to make a decision.

Thank You

Rafal Los

Twitter: @RafalLos
Email: Rafal@HP.com
Direct: +1 (765) 247 - 2325

Blogs:
"Following the White Rabbit"
http://www.communities.hp.com/securitysoftware/blogs/rafal/default.aspx
"Digital Soapbox"
http://preachsecurity.blogspot.com

Oh! … and I work at HP's Application Security Center (ASC)