Hacking's Gilded Age: How APIs Will Increase Risk And Chaos

K. Scott Morrison
CTO & Chief Architect
Layer 7 Technologies

Session ID: ASEC-402
Session Classification: Intermediate

RSA CONFERENCE 2012

What is the gilded age?

"To gild refined gold, to paint the lily,
To throw a perfume on the violet,
To smooth the ice, or add another hue
Unto the rainbow, or with taper-light
To seek the beauteous eye of heaven to garnish,
Is wasteful and ridiculous excess."

William Shakespeare
King John (1595)

The Gilded Age: A Tale of Today

Mark Twain

Charles Dudley Warner

1873

"During the 1870s and 1880s, the U.S. economy rose at the fastest rate in its history, with real wages, wealth, GDP, and capital formation all increasing rapidly."

Economic growth between 1865 and 1898

Wheat output increases	256%
Corn	222%
Coal	800%
Miles of railway track	567%

Source: *Wikipedia* http://bit.ly/xxHKnK

Rockefeller
Carnegie
Morgan
Astor
Vanderbilt
Mellon

Who will be the robber barons of the 21st century?

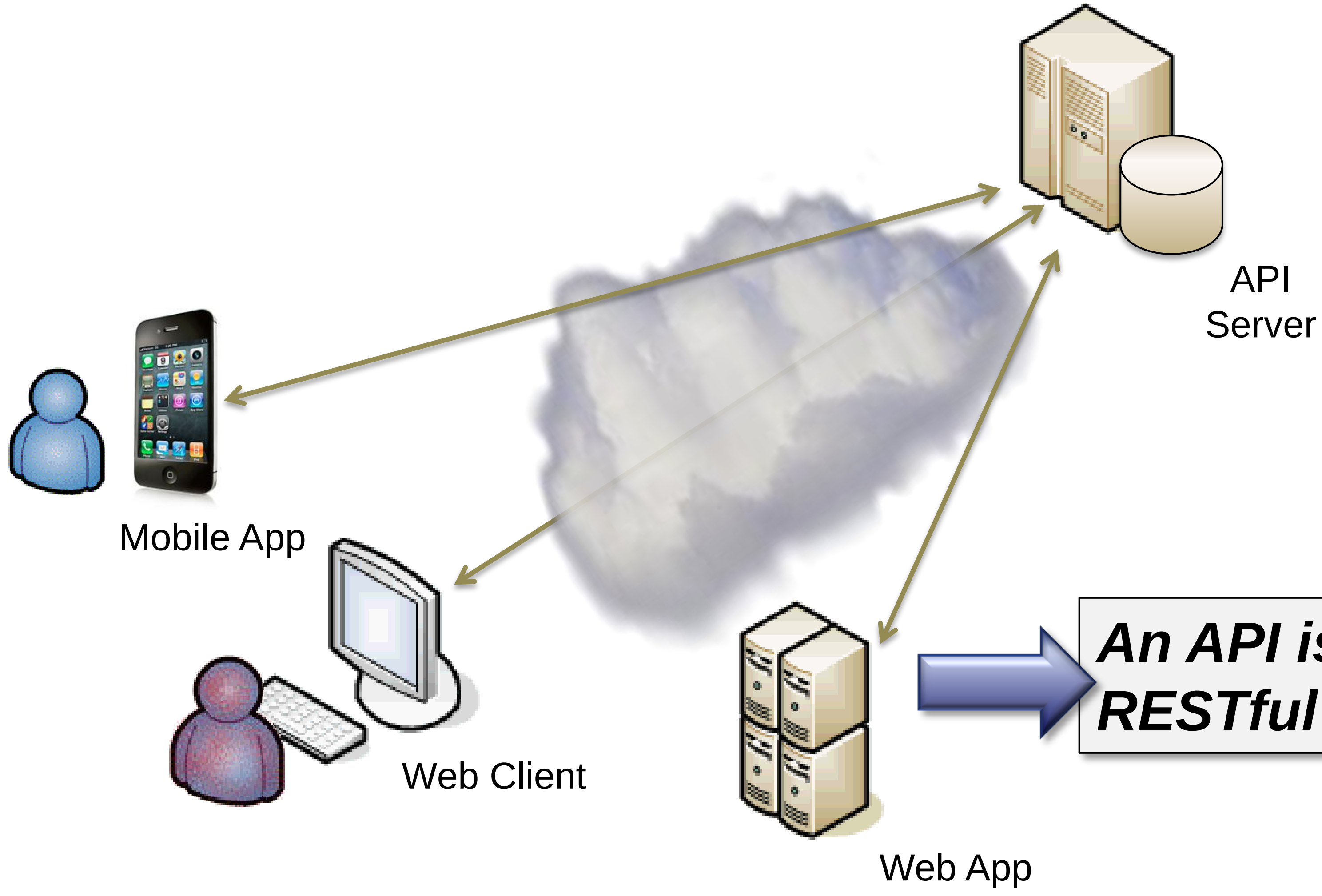
We don’t know yet

And APIs could be a rich avenue of exploitation

Here Is What This Talk Is About:

- The new API threat
  - …and the potential rise of the hacker-robber-baron
- Are APIs just like the Web? Or are they different?
  - Look at three important areas:
    1. Parameterization
    2. Identity
    3. Cryptography
- How to apply the lessons of this talk

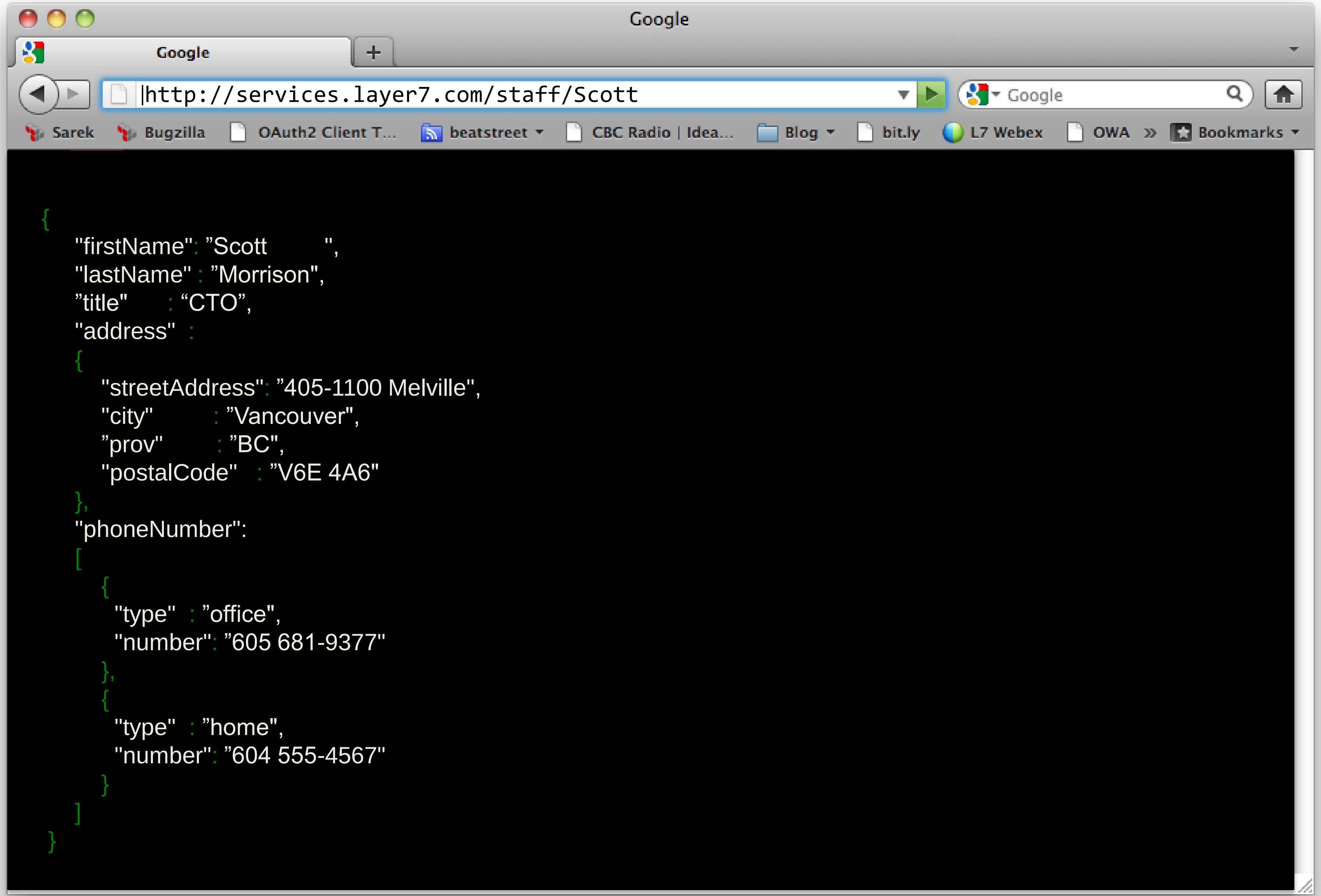
What is an API?



An API is a RESTful service

For Example:

```
GET http://services.layer7.com/staff/Scott
```



```
http://services.layer7.com/staff/Scott
```

```
{
   "firstName": ”Scott        ",
   "lastName" : ”Morrison",
   ”title"     : “CTO”,
   "address"  :
   {
      "streetAddress": ”405-1100 Melville",
      "city"          : ”Vancouver",
      ”prov"          : ”BC",
      "postalCode"   : ”V6E 4A6"
   },
   "phoneNumber":
   [
      {
        "type"  : ”office",
        "number": ”605 681-9377"
      },
      {
        "type"  : ”home",
        "number": ”604 555-4567"
      }
   ]
}
```

LAYER 7 TECHNOLOGIES

RSA CONFERENCE 2012

This is a Technological Sea Change

	Old	New
Transport	HTTP	HTTP
Data	XML	JSON
Authentication	Basic, X.509, Kerberos, SAML	OAuth
Confidentiality & Integrity	WS-Security	SSL

Where Simple Won

<u>SOAP + WS-*</u>

- Complex
- Highly standardized
- Vendor driven
- Barriers

<u>RESTful APIs</u>

- Simple
- Informal
- Grassroots
- Frictionless

“Sounds great. So what’s the problem?”

The Real Problem Is This:

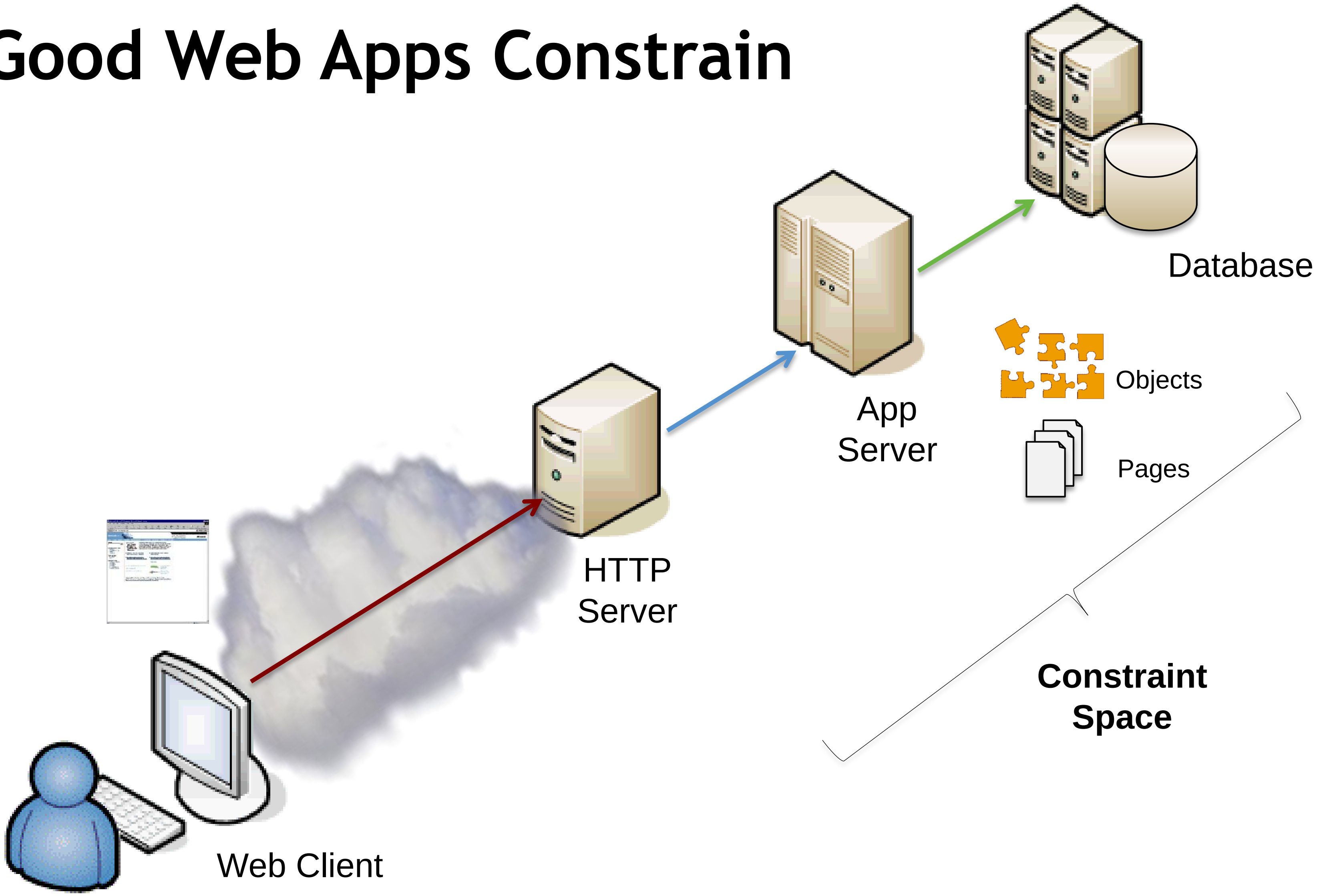
API Development !=
Web Development

In Particular:
We need to be wary of bad web development practices migrating to APIs…

Problem Area #1: *API Parameterization*

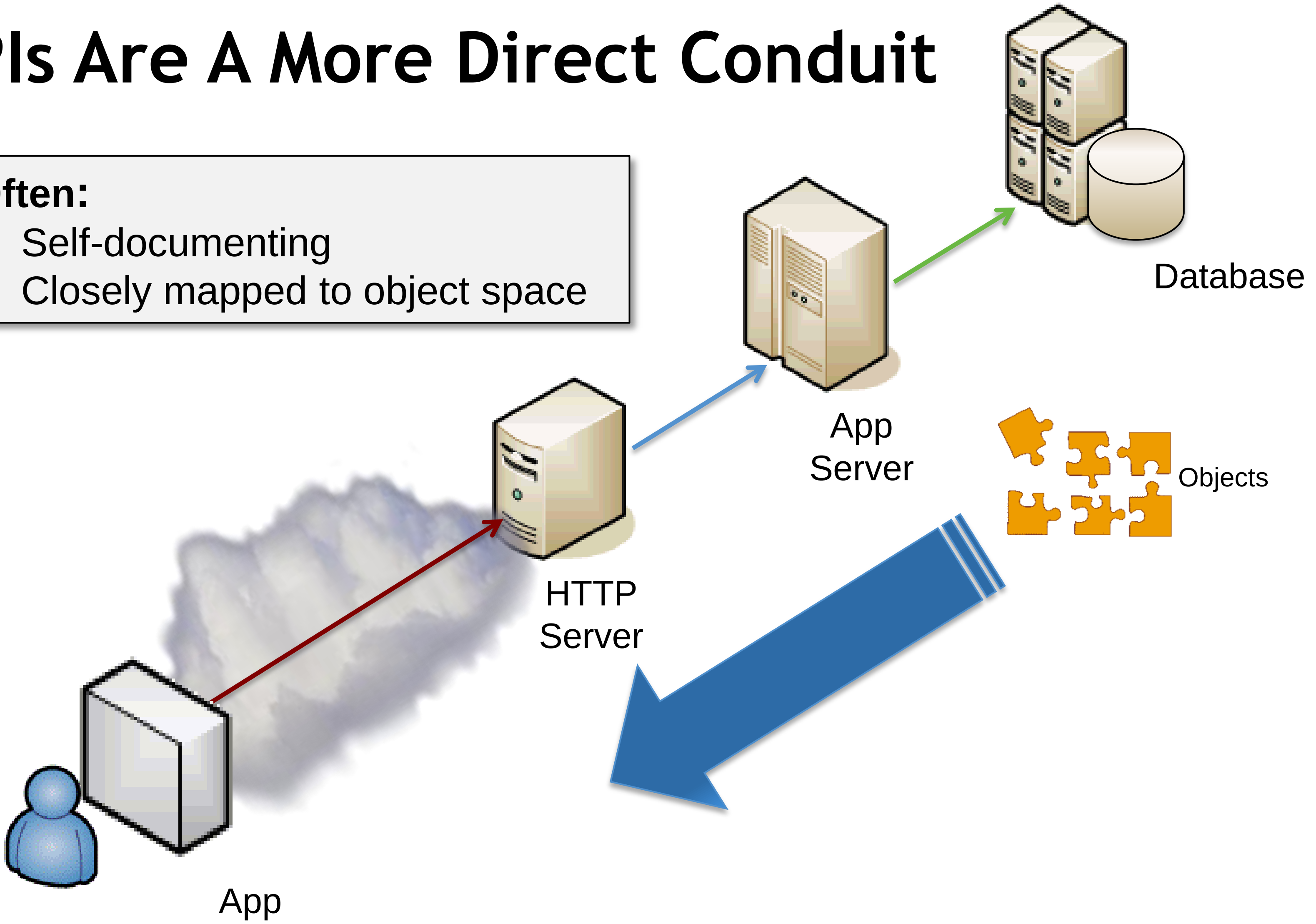
- In the web world, parameterization was limited and indirect
  - Subject to the capabilities of URLs and forms
- APIs in contrast and offer much more explicit parameterization
  - The full power of RESTful design: GET, POST, PUT, DELETE
    - (And don't stop there… what about effects of HEAD, etc)?
- This is a greater potential attack surface
  - Injection, bounds, correlation, and so on

Good Web Apps Constrain

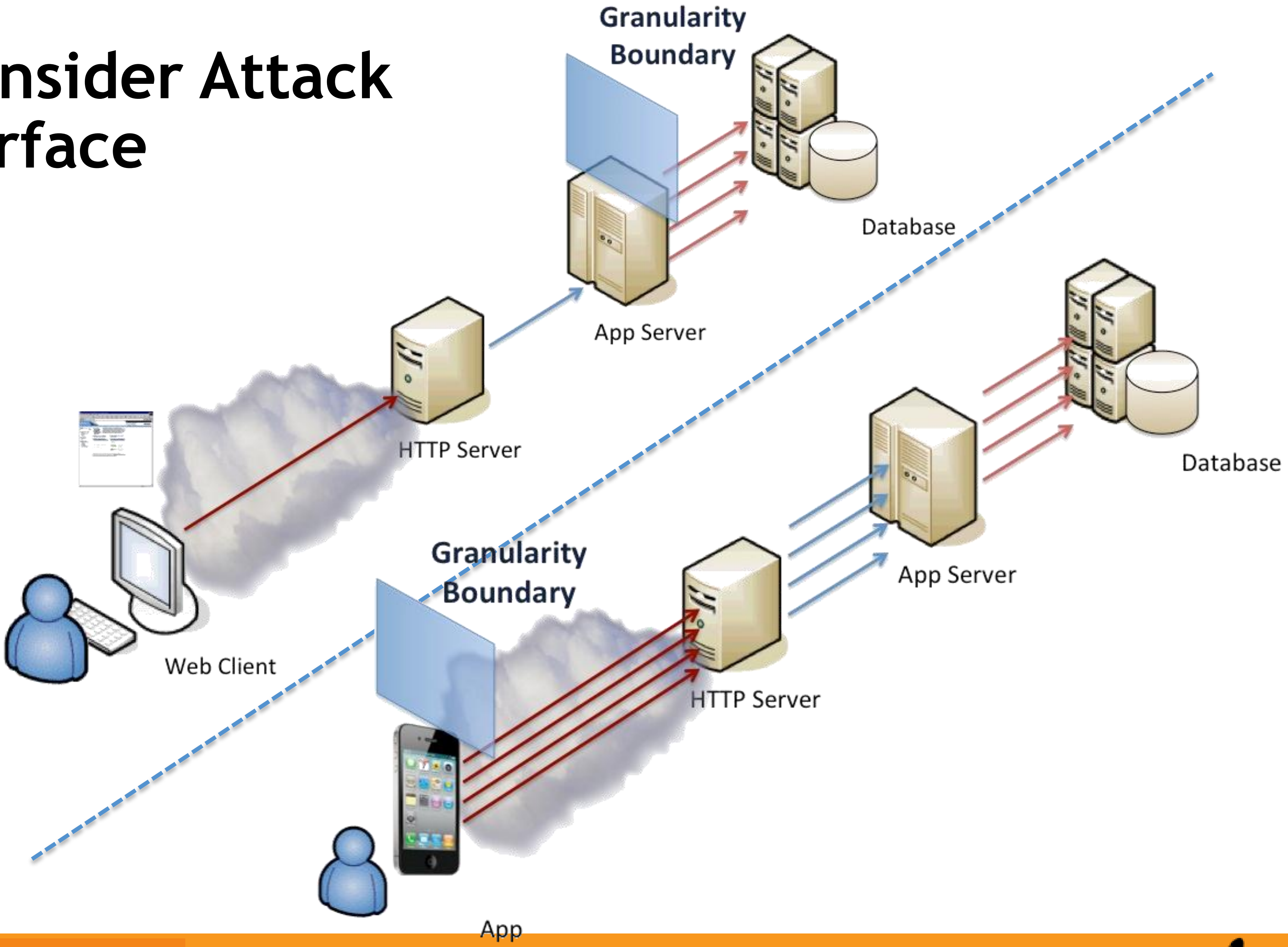


APIs Are A More Direct Conduit

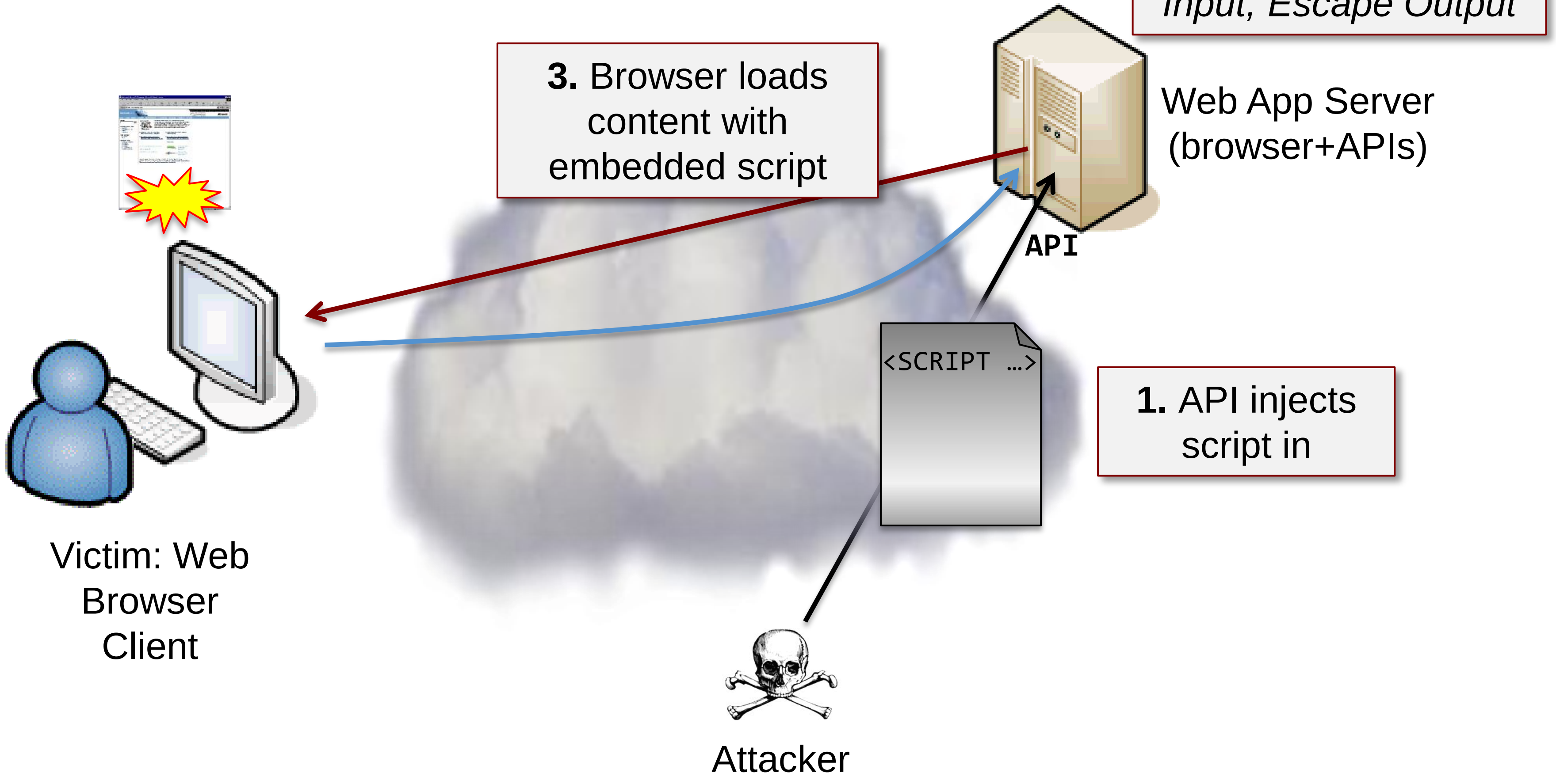
Often:
- Self-documenting
- Closely mapped to object space



Consider Attack Surface

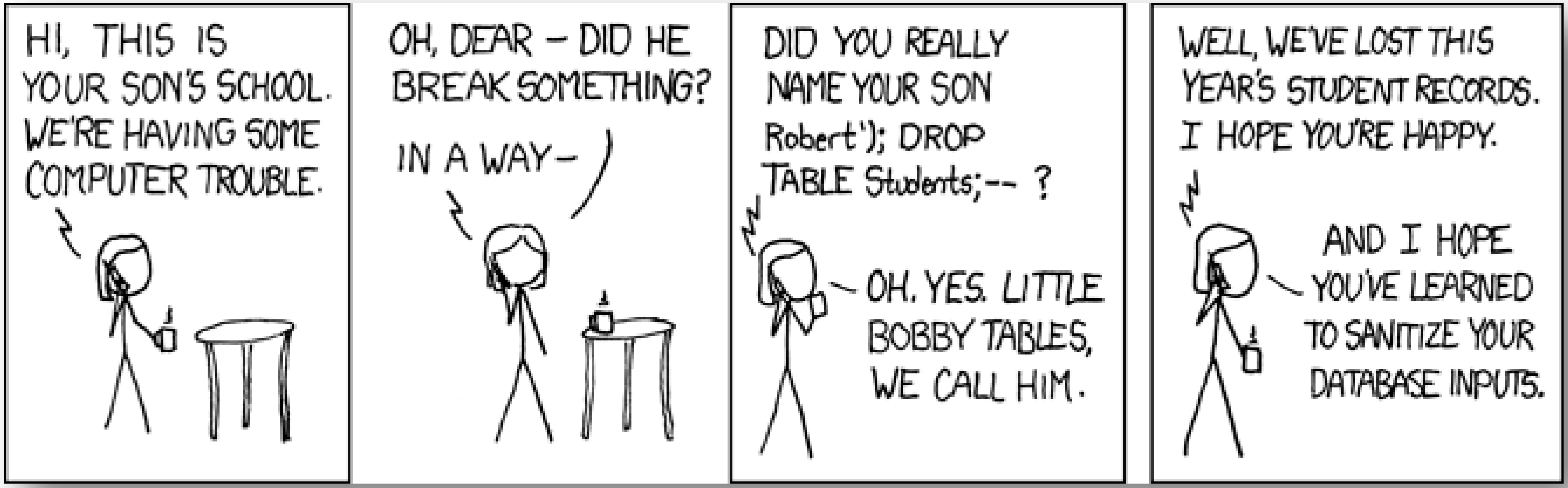


Script Insertion is Just One Potential Exploit


2. Server fails to perform **FIEO:** *Filter Input, Escape Output*
3. Browser loads content with embedded script
Web App Server (browser+APIs)
API
<SCRIPT …>
1. API injects script in
Victim: Web Browser Client
Attacker

SQL Injection is Another

Exploits of a Mom


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Source: https://xkcd.com/327/

Mitigation Strategy

- Rigorous validation of user supplied input
  - Stronger typing
  - Sets and ranges
  - Avoid auto-generated schemas that make everything a string
- Use schema validation
  - XML Schema, RELAX-NG, Schematron
    - Please no DTDs!
  - JSON schema validation
  - WADL’s second coming

Mitigation Strategy *(cont.)*

- Regex scanning for signatures
- Tune scanning for the API
  - Sometimes SELECT is OK
- Virus scanning of attachments
  - Don’t forget B64’d message content
- Library, service, or gateway solutions
  - Decoupled security
  - What are the tradeoffs?

Mitigation Strategy

- Whitelist tags if you can (i.e. where the validation space is small and concise)
  - Not always practical
  - (Note that I'm referring to whitelisting tags not IPs.)
- Blacklist dangerous tags like `<SCRIPT>`
- Always perform FIEO (Filter Input, Escape Output)
- Learn more: http://xssed.com

Problem Area #2: *Identity*

- We had it surprisingly good in the Web world
  - Browser session usually tied to human
  - Dealing with one identity is not so tough
    - Security tokens abound, but solutions are mature
      - – Username/pass, x.509 certs, multi-factor, Kerberos, SAML, etc
  - APIs rapidly becoming more difficult
    - Non-human entities
    - Multiple layers of relevant identities
      - – Me, my attributes, my phone, my developer, my provider…

API Keys

"An **application programing interface key** (API key) is a code generated by websites that allow users to access their application programming interface. API keys are used to track how the API is being used in order to prevent malicious use or abuse of the terms of service.

API keys are based on the UUID system to ensure they will be unique to each user."

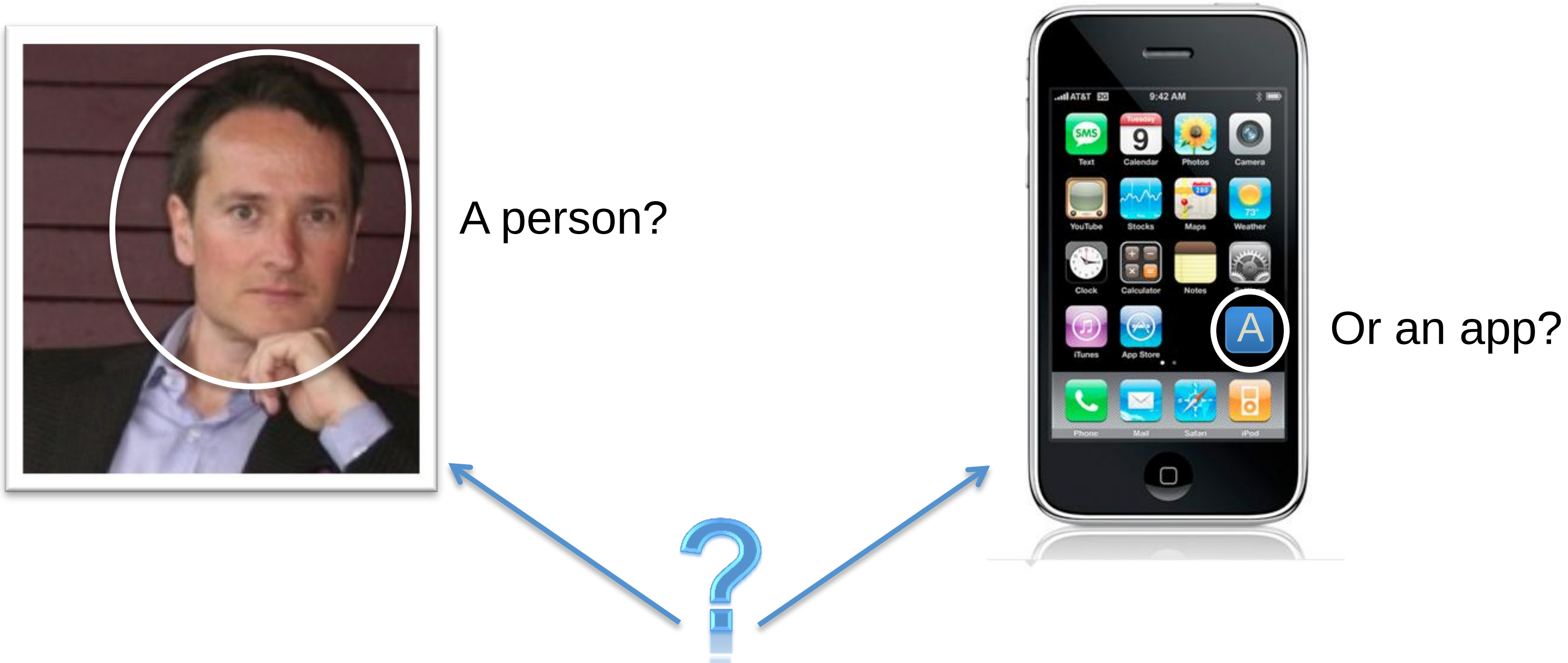
(Source: wikipedia http://en.wikipedia.org/wiki/Application_programming_interface_key)

For Example:

```
GET http://example.layer7.com/services/staff
    ?APIKey=15458617-7813-4a37-94ac-a8e6da6f6405
```

Seriously? WTF.

How Does An API Key Map To Identity?



15458617-7813-4a37-94ac-a8e6da6f6405

It is entirely inconsistent

The Identity Profile

Increasingly we need to move toward large number of claims (multiple identity profile)

- appID
- userID
- deviceID

- User attributes
  - Roles
  - Geo location
  - IP
  - User agent
  - Time of day
  - etc

Where Did API Keys Come From?

- API keys came from Google APIs like maps, early Yahoo APIs, early Twitter APIs etc.
  - Originally meant for loose, non-authoritative tracking
  - Rate limits, approximate usage profiling
- Google geocoding v3.0 API deprecates API keys
  - Uses IP instead to track and throttle
  - This has its own set of problems
    - IP address spoofing
    - Problems with legitimate clients like cloud servers
- Google Premier uses a public `client_id` and requires signed URLs
  - (Strips domain leaving only path and query parameters)

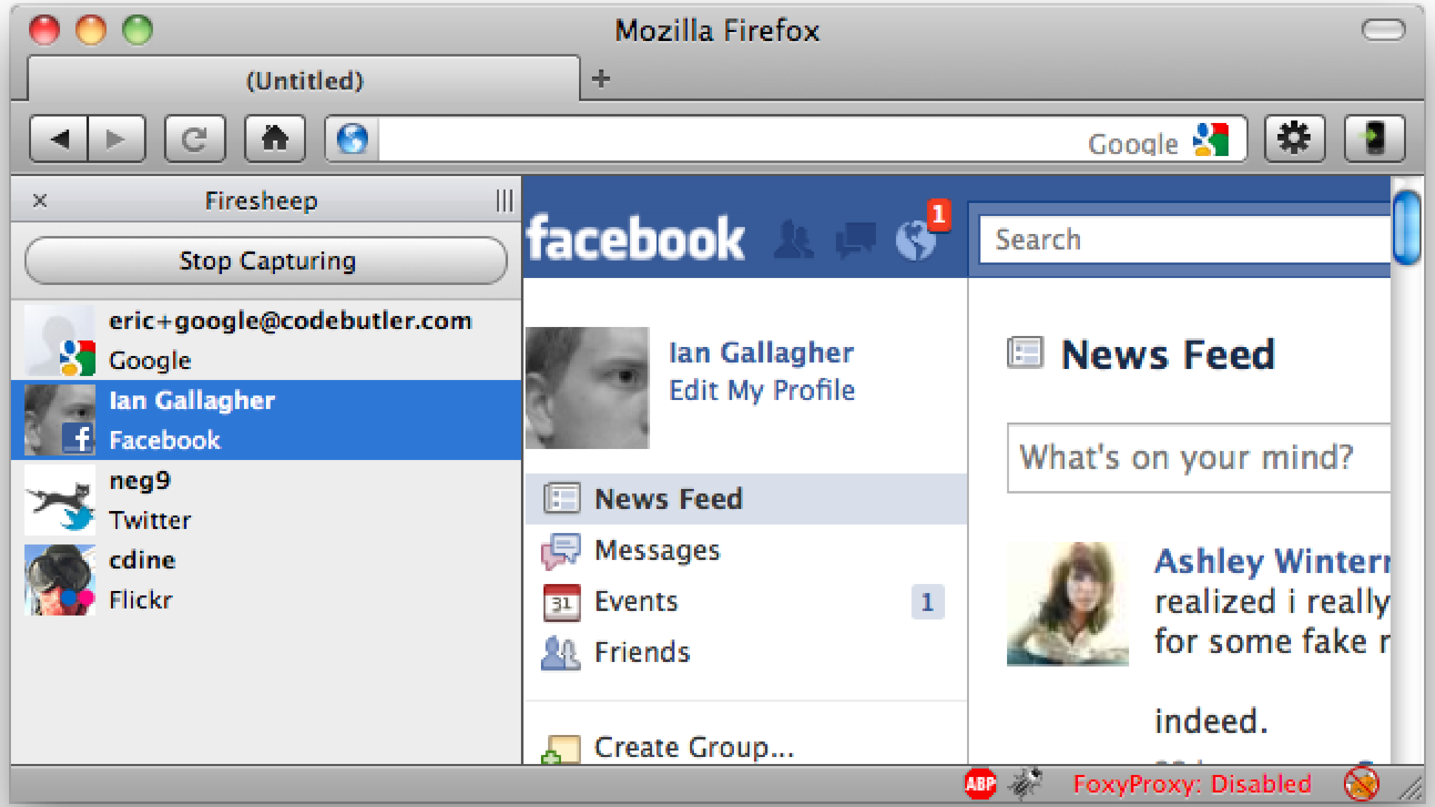
The Real Issue Is That This Is Ad-hoc Sessioning

- This is a web developer cultural issue
  - On the web, session tokens are rarely secured
    - **Step 1:** Sign in with SSL
    - **Steps 2 to n:** Track session using cookie without SSL
- There are two bad things going on here:
  1. No protection of security token
  2. No binding of token to message content

→ These two things are very related

- In 2012, this is a huge problem…

One Word: *Firesheep*



eric+google@codebutler.com
Google
Ian Gallagher
Facebook
neg9
Twitter
cdine
Flickr

Ian Gallagher
Edit My Profile

News Feed
Messages
Events 1
Friends
Create Group...

News Feed
What's on your mind?
Ashley Winterr
realized i really
for some fake r
indeed.

In the post Firesheep world, unprotected session tracking should never be tolerated…

Source: http://codebutler.com/firesheep

Side Note: Cookie Sessioning May Also Be Indicating Design Problems

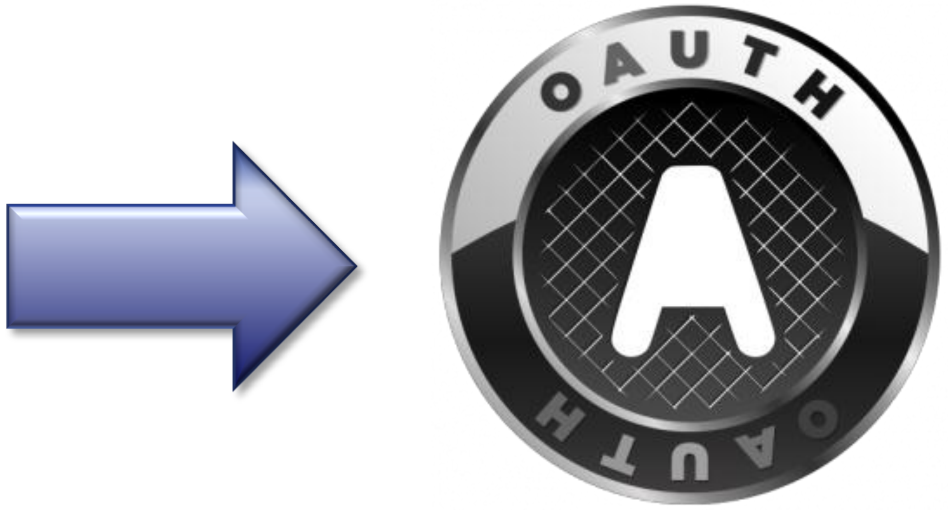
- Good RESTful design is stateless
- Authentication session tokens—as long as they are protected—are OK
- State tokens, however, are not very RESTful
  - i.e. Tokens that index server-side state objects
  - Makes it hard to scale
  - But it could also be an exploit vector

Bottom Line: The API Key Was *Never* Meant To Be Authoritative

- Strange hybrid of HTTP's `USER-AGENT` and sessioning
- OK only for general tracking
- Anything that matters should use real security tokens
    - Anywhere where identity is important:
        - APIs that provide access to sensitive data
        - APIs that change things that matter
        - APIs that charge for use
        - etc.

Twitter Gets It

API Keys,
Basic
Authentication



Will OAuth survive adolescence?

- Complexity and confusion
- Interop is a nightmare right now
- Enterprise token management
  - Beyond the timeout
  - Lifecycle, revocation, distribution, etc.
  - Talk, but not a lot of action
- SSL everywhere
  - Protect bearer tokens
- Phishing vulnerabilities
  - Server authentication and loss of trust in the CA infrastructure

Mitigation

- **Protect the tokens!**
- HTTPS everywhere
  - This is another web design cultural issue
  - It’s just not that expensive any more
- Need HTTP Strict Transport Security (HSTS) for APIs
  - http://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security

Problem Area #3: *Cryptography and PKI*

- Cryptography is reasonably mature on the web
  - Surprisingly limited use patterns
  - SSL/TLS
  - Very little tough PKI (like client-side)
- So what’s wrong with APIs?

It’s Like We Forgot Everything We Knew

- Emailing keys
  - API, shared secrets, etc.
- Bad storage schemes
  - Security through obscurity
  - Toy ciphers
- No life cycle management
  - Limits on use
  - Time limits
  - Revocation
  - Audit

The Issues

- Key management
  - Especially across farms
- Nobody takes web PKI seriously
  - CRLs and OCSP aren’t much good in the browser world
    - Fail open—seriously
- CA trust breakdown
  - Comodo fraud in March 2011

The Issues *(cont.)*

- Cipher suite restrictions
  - Avoiding downgrades
- Client-side certificate authentication is hard
- The alternatives for parameter confidentiality and/or integrity are hard:
  - XML encryption is still there
    - Not for the faint of heart
  - OAuth 1.0 gave you parameter signing
    - Only optional in 2.0

Mitigations

- Use real PKI
  - I know it’s painful
- Use HSMs to protect keys that really matter
- Otherwise use real key material protection schemes
  - PKCS #12, etc
  - Libraries abound

Mitigations

- You must do CRLs and OCSP for APIs
- Google maintains a certificate registry: http://googleonlinesecurity.blogspot.com/2011/04/improving-ssl-certificate-security.html
- Google safe browsing API: http://code.google.com/apis/safebrowsing/developers_guide_v2.html
  - Blacklist of phishing and malware sites
- DANE and DNSSEC

Where Does This All Leave Us?

- SOAP, the WS-* stack dealt with much of this very rigorously
  - But it was just too hard.
- We need to learn from this, but make it easier to implement
- Here’s how…

How Do I Apply This Today?

- Use SSL for all API transactions
  - Hides many sins
    - Confidentiality, integrity, replay, binding token+message, server authentication, etc.
- Use real PKI
  - Yes, it's hard
  - *But you can't skimp here*
- Use OAuth for distributed authentication
- Use real frameworks, don't reinvent
  - They exist for most languages
  - Consider gateways to encapsulate security

Hacking's Gilded Age: How APIs Will Increase Risk And Chaos

RSACONFERENCE2012