Why Web Security Is Fundamentally Broken

Jeremiah Grossman
Founder & Chief Technology Officer

WhiteHat SECURITY

Web Security

Rule #1:

A website must be able to defend itself against a hostile client [browser].

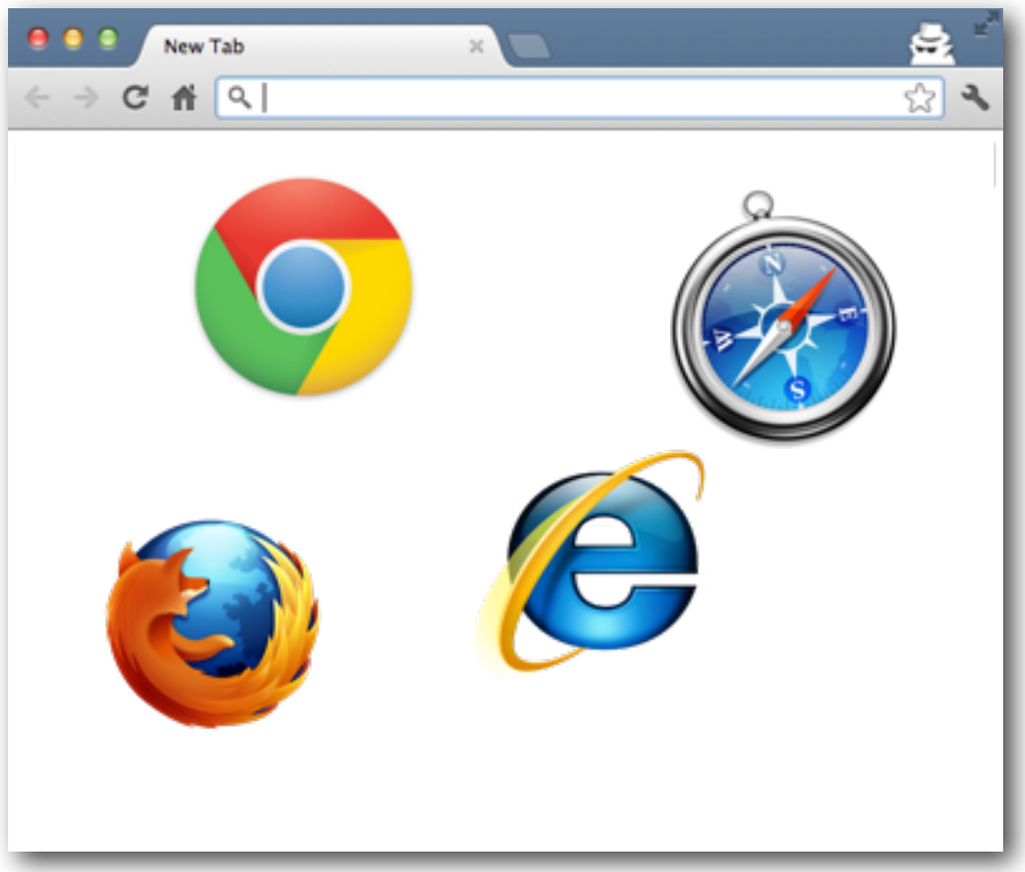
Challenging, but possible to follow.

WhiteHat SECURITY

Web Security
Rule #2:

A browser must be able to defend itself against a hostile website.

Impossible.



WhiteHat SECURITY

Today's browsers make availabe to every website you visit:

Passive access to your operating system information, various system settings, browser type / version, installed add-ons & plug-ins, geographic location, websites currently logged-into, etc.

Give a website just 1 mouse-click — Then it gets access to:

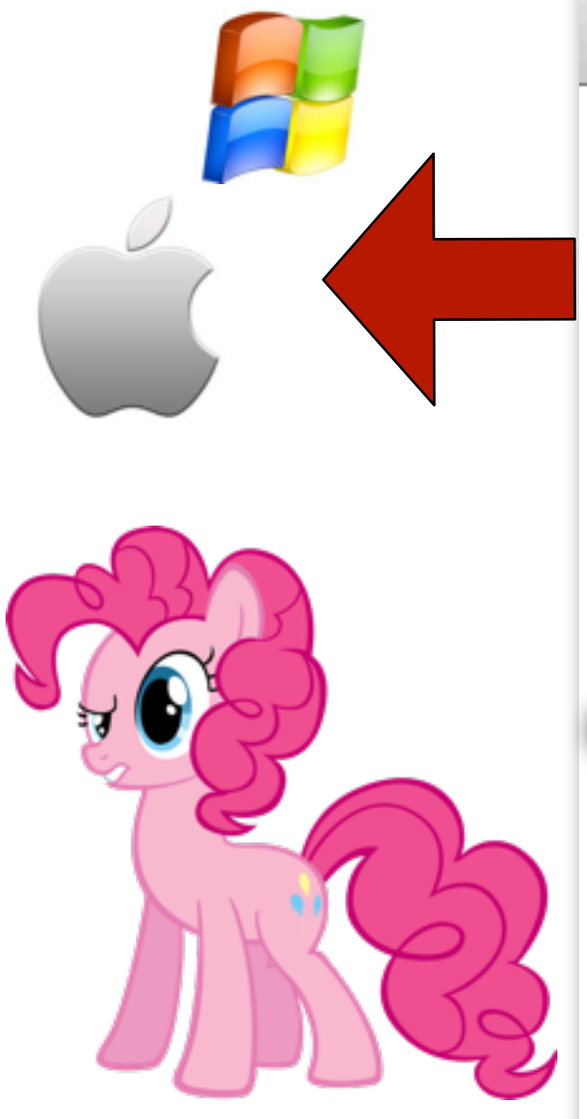
Your name, where you live, where you've been, town you grew up in and went to school, martial status, photos, and in some cases, the browser's auto-complete data and surfing history.

All browsers also allow a [malicious] website to:

Force your browser to send self-incriminating Web requests, hack your Intranet, auto-XSS / CSRF you on any website, etc.

WhiteHat SECURITY

The 2 Types of Browser Attacks



1) Attacks designed to escape the browser walls and infect the operating system with malware.
(a.k.a. Drive-by-Downloads)

Security: Sandboxing, silent and automatic updates, increased software security, anti-phishing & anti-malware warnings, etc. [Enabled by default]

**2) Attacks that remain within the browser walls and compromise cloud-based data.
XSS, CSRF, Clickjacking, etc.**

Security: SECURE Cookies, httpOnly, X-Frame-Options, Strict-Transport-Security, X-Content-Type-Options, Content Security Policy, EV-SSL, etc.
[Opt-In by website, users can't protect themselves]

Common use-case:

DEMO

```
<img src="http://someotherwebsite/image.png">

<img src="http://someotherwebsite/image.png"
onload="successful()" onerror="error()">
```

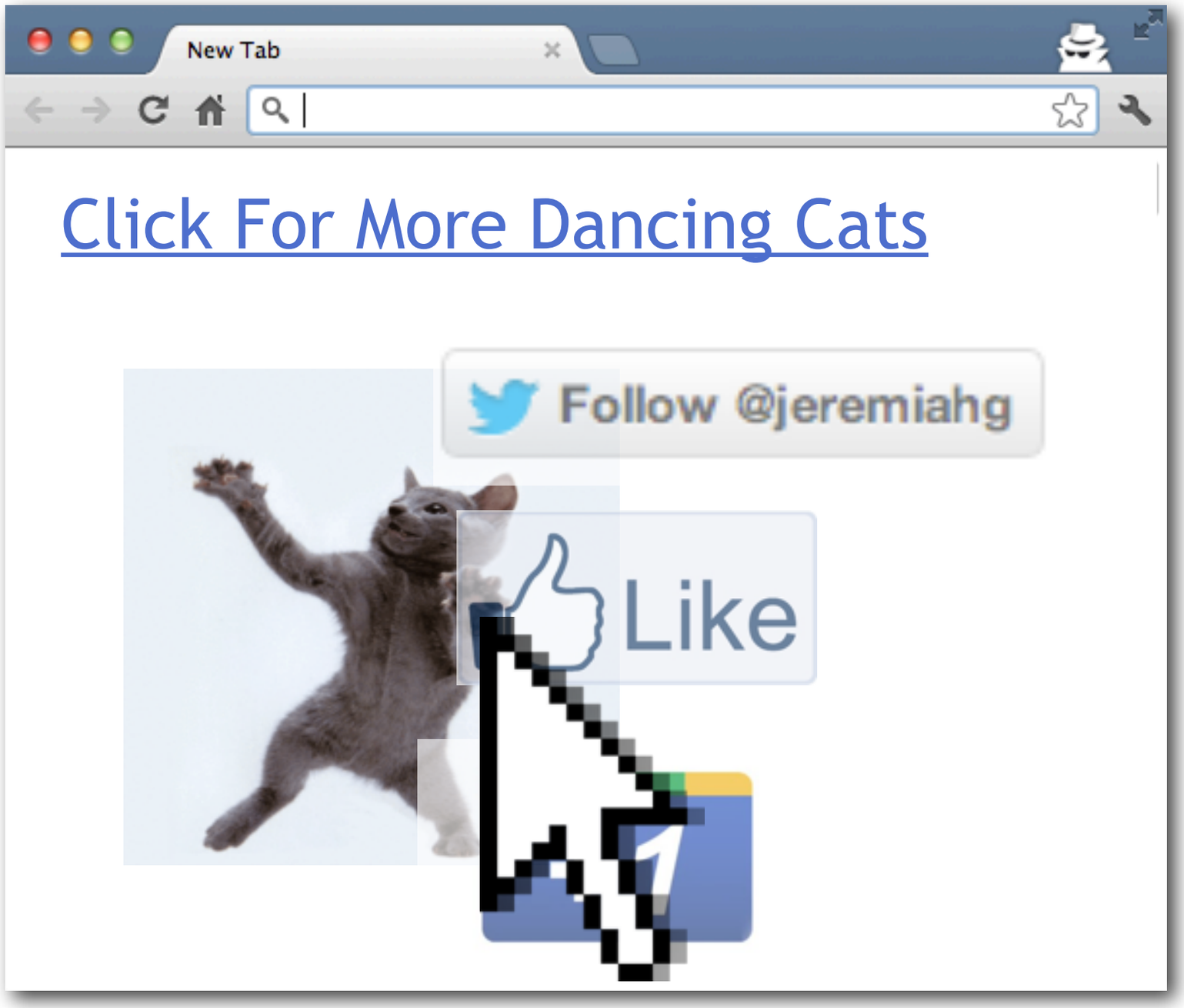
If the image file loaded correctly, the “successful” Javascript function executes.
If some error occurred, obviously the “error” function executes.

Login-Detection (via CSRF):

```
<img src="http://someotherwebsite/loggedin.png"
onload="loggedIn()" onerror="notLoggedIn()">.
```

If the user is logged-in, the image file loads successfully, which executes the “loggedIn.” If they’re not logged-in, “notLoggedIn” is executed.

Deanonymize (via Clickjacking)



"A mashup is a self-inflicted XSS attack."
Douglas Crockford

DEMO

http://mayscript.com/blog/david/clickjacking-attacks-unresolved

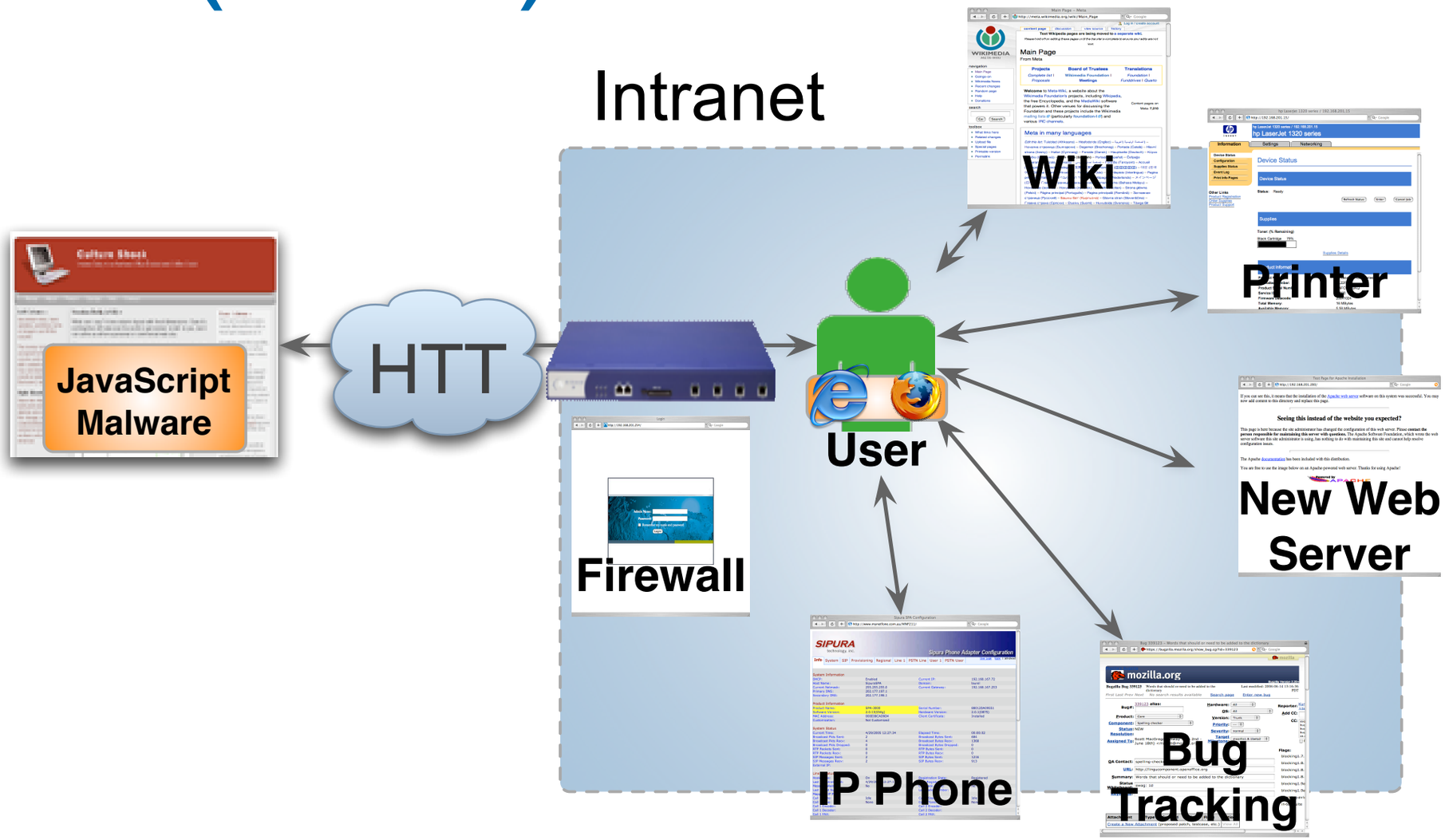
WhiteHat SECURITY

“Unless you've taken very particular precautions, assume every website you visit knows exactly who you are, where you’re from, etc.”

Jeremiah Grossman

WhiteHat SECURITY

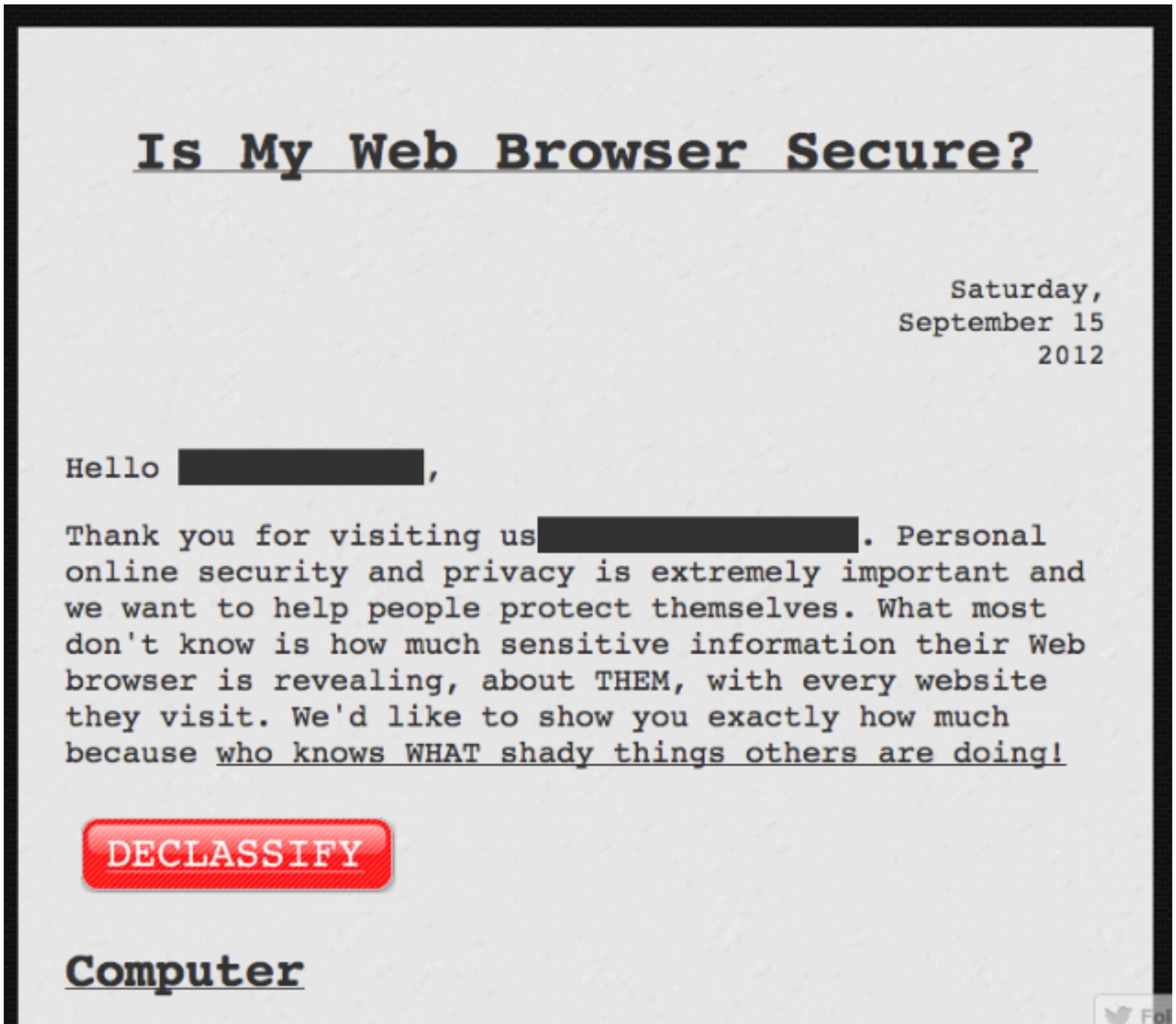
Browser Intranet Hacking Circa (2006)




```
<iframe src="http://192.168.1.1/" onload="detection()"></iframe>
```

WhiteHat SECURITY

DEMO



WhiteHat SECURITY

Possible Solutions?

WhiteHat SECURITY

Login-Detection

***Idea**: Do not send the Web visitors cookie data to off-domain destinations, destinations different from the hostname in the URL bar, along with the Web requests.*

Breaks the Web

Not sending cookies off-domain would break websites using multiple hostnames to deliver authenticated content. Breaks single-click Web widgets like Twitter "Follow," Facebook "Like," and Google "+1" buttons. Also breaks visitor tracking via Google Analytics, Coremetrics, etc.

Deanonymization

***Idea**: Ban IFRAMEs entirely, or at least ban transparent IFRAMEs. Ideally, browser users should be able to "see" what they are really clicking on.*

Breaks the Web

Millions of websites current rely upon IFRAMEs, including transparent IFRAMEs, for essential functionality. Notable examples are Facebook, Gmail, and Yahoo! Mail.

WhiteHat SECURITY

Browser Intranet Hacking

***Idea**: Create a barrier in the browser between "public" and "private" networks by prohibit the inclusion of RFC-1918 on non-RFC-1918 websites.*

Breaks the Web

Some organizations actually do include intranet content on public websites, for their employees, which does not violate RFC specification.

Vulnerabilities are required by Web standards.

bigger problem

KNOWN “WONT-FIX” ISSUES

WhiteHat SECURITY

Browser vendor's choice is simple:

Be less secure and more user adopted, or secure and obscure.

Browser War

=

Trench Warfare

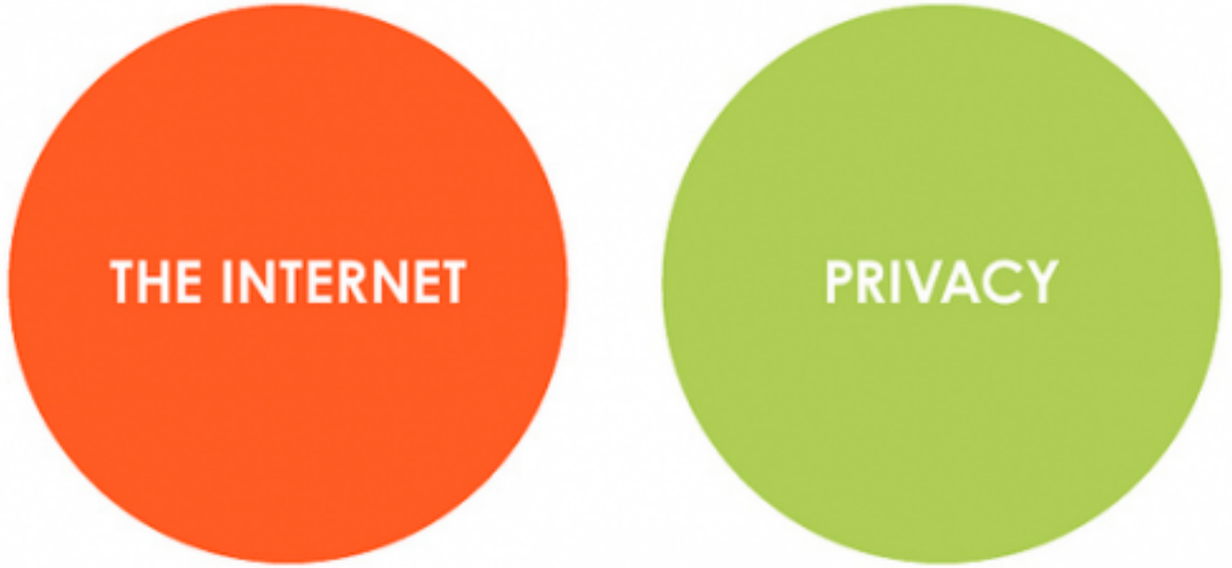
“[N]obody's breaking the web, dude. Not now, not ever.”

Dan Kaminsky to Jeremiah Grossman, December 21, 2010

Security: SECURE Cookies, httpOnly, X-Frame-Options, Strict-Transport-Security, X-Content-Type-Options, Content Security Policy, EV-SSL, etc.

- Opt-In security, by website owners
- Measurably low adoption rates
- Do not allow for Web users to protect themselves

Web browsers are NOT “safe.”
Web browsers are NOT “secure.”
Web browsers do NOT protect your “privacy.”



A HELPFUL VENN DIAGRAM

WhiteHat SECURITY

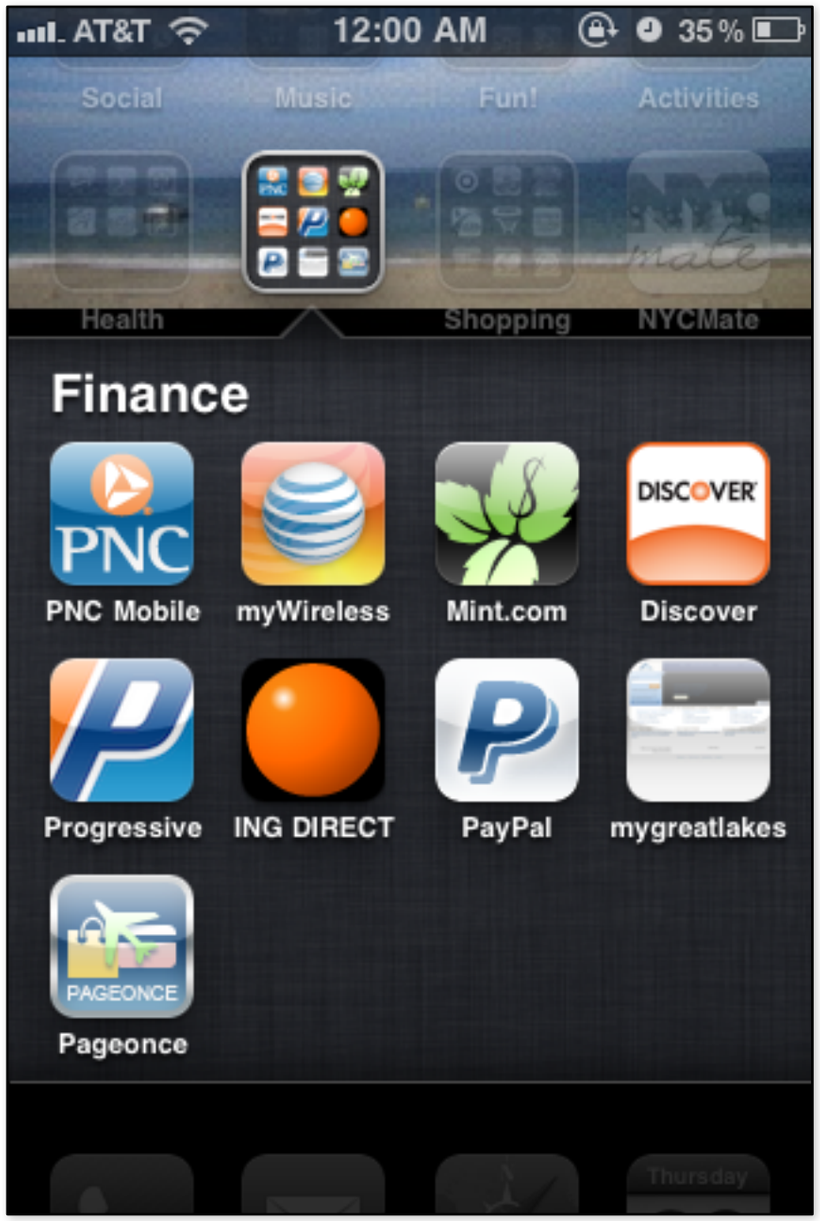
What do we do now?

Geek meditation session.

1) Status Quo
2) .SECURE
3) Break the Web

• • •

Mobile Apps



Mini-browsers, where each site / app is isolated. No issues with Login Detection, Denaonymization, etc.

“DesktopApps”

Custom browsers’ designed to automatically launch to a website and go no further.

WhiteHat SECURITY

Thank You!

Blog: http://blog.whitehatsec.com/
Twitter: http://twitter.com/jeremiahg
Email: jeremiah@whitehatsec.com

I was not in your threat model.

1:53 PM Apr 28th via TweetDeck
Retweeted by 1 person

jeremiahg
Jeremiah Grossman

WhiteHat SECURITY