mozilla

Building and Breaking the Browser

Window Snyder
Mike Shaver

Overview

Who the @#&^@#$ are we?

A security process tested by millions

Lies, damned lies, and statistics

New security goodies in future Firefoxen

Tools you can use

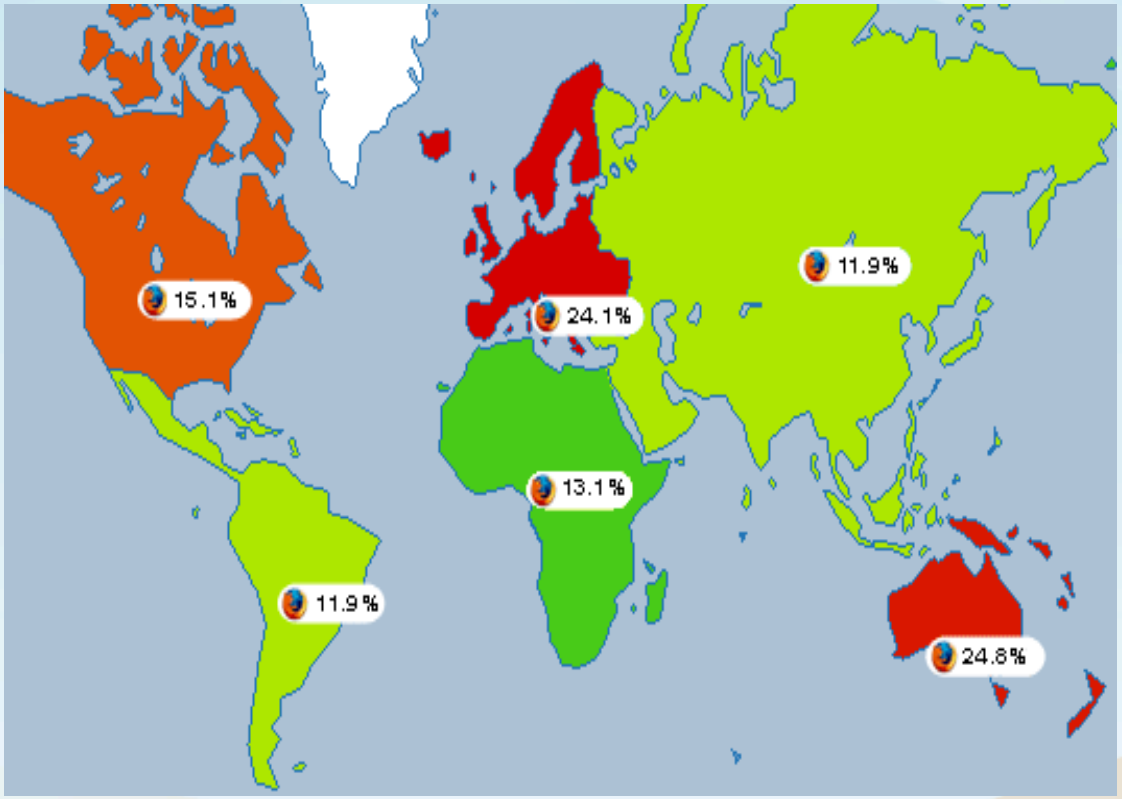
About Mozilla

Mozilla is...

- a global effort to promote choice & innovation on the Internet
- the foremost advocate for users on the Web
- an open source project with thousands of code contributors and tens of thousands of non-code contributors
- home of the Firefox Web browser
- more than 100 million users worldwide

Who runs Firefox?

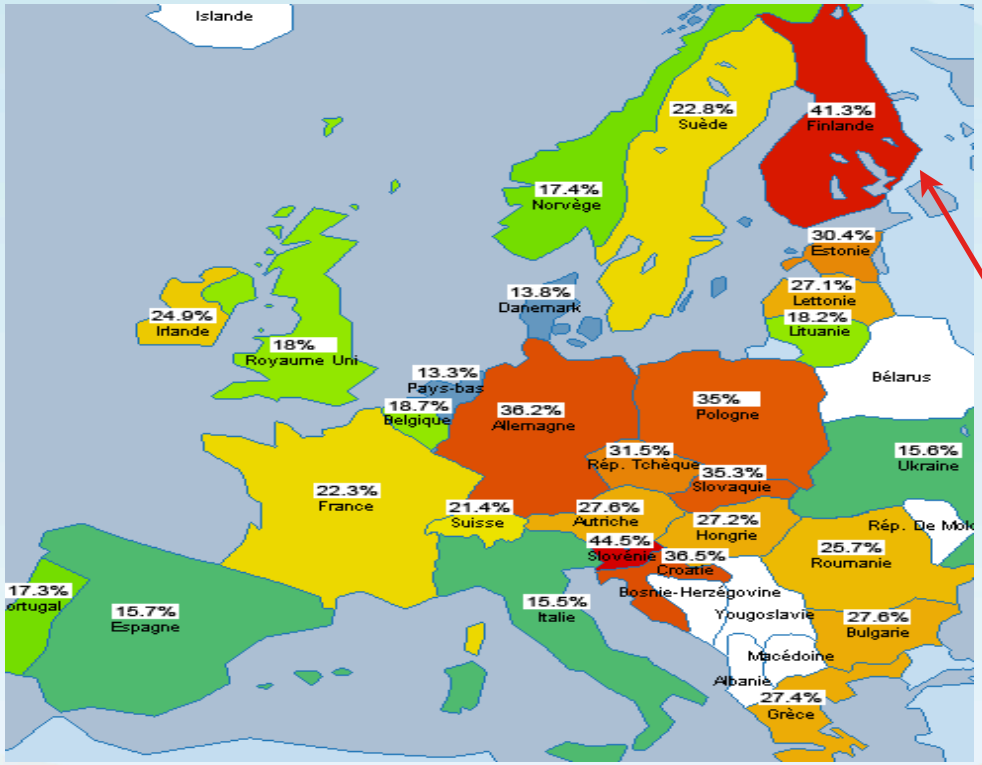
18% of Internet users worldwide; 100 million people.



http://www.xitimonitor.com April 2007

Who runs Firefox?

Almost 25% of Europe!



(Finland loves us: 41%!)

Aliens run Firefox…

(Market share numbers unavailable.)

A security process tested by millions

Opening up to lock it down

Approach to Security - Transparency

- Community supports security testing and review efforts
- Code and developer documentation is available to anyone
- Security researches can spend their time in analysis and not in reconnaissance
- External parties can check our work, do not need to rely on what we tell them
- Design online, open meetings (MSFT take great notes!)
- Real time updates on vulnerabilities

Security Process

Self-organizing Security Group is about 85 people representing all aspects of the community

Features are security reviewed to ensure compatibility with the overall security model

Designed with security in mind

Security testing is continuous throughout development process

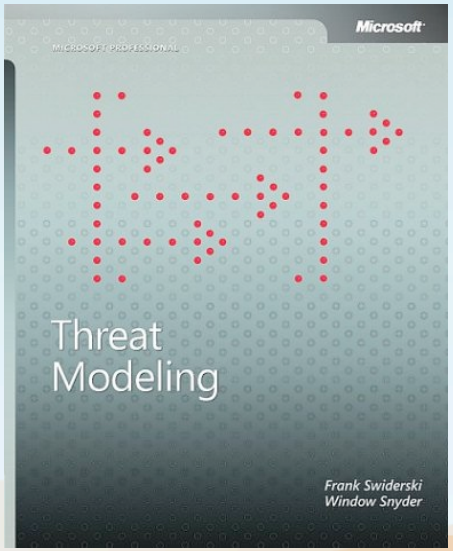
Security updates every 6-8 weeks

Threat Modeling

Identify entry points into the system

Trace data flows through the application

Focuses penetration testing effort on specific components



Component Security Review

Review new features to determine how they impact the security of the product. Sometimes effects can be indirect!

Determine if they introduce new vectors

Evaluate existing mitigations

Determine if mitigations are sufficient

Write tests to prove it

Develop additional mitigations when your tests find things you missed!

Code Review

Focused on components that:

- are most likely to handle user input directly
- perform complex memory management
- perform pointer arithmetic
- parse complex formats

Looking for:

- Improper string handling
- Integer arithmetic errors
- Uninitialized variable utilization (esp. in error cases)
- Memory allocation/deallocation errors
- Defense in depth

Make Code Review Scale

Include these checks as part of the peer-review system required before check-in

Develop a level of confidence in the new code. Over time code at that confidence level grows, replaces lower confidence code

(Unless you keep all your legacy code…)

Make Code Review Scale (cont.)

Many environments have peer-review systems in place – never too late to start

Train the developers to recognize the kinds of code constructs that often result in vulnerabilities

Humans, and even software developers, are good at recognizing patterns

Engaging security consultants

Work with some of the best application security experts

Different perspective

Experience with other projects that have had to solve similar problems

Not personally invested in any design, decision, architecture, etc

We've worked with Matasano, Leviathan, IOActive, and others; ask around for references and good (and bad!) experiences

Automated Penetration Testing

Custom fuzzing code automates destruction

Specific to targeted components

- Leverage existing frameworks and libraries where possible
- Mimics normal format of input: attackers don't care about standards!

Our targets include

- FTP protocol and list formats
- HTTP server responses
- JavaScript
- URI methods
- Content parsing and DOM: HTML, SVG, XUL, MathML
- Goal: all untrusted data sources

Manual Penetration Testing

Individual test cases

Negative testing

Validating issues identified through source code analysis

Scratch those hard to reach areas!

Identify new vectors of attack

Mostly by hand, but some tools are useful:

- Netcat – The network swiss army knife
- Snark – Attack proxy and request/response editor
- Windbg – Runtime editing of variables and data injection

Security Updates

Most vendors ship security updates for vulnerabilities reported externally

- The bugs found internally (though QA, engaging penetration testers, etc) are rolled up in service packs in major releases
- Bugs get the benefit of a full test pass
- Takes a very long time for the fix to reach the user
- Can't tell from the outside how many bugs get fixed this way

Mozilla is continuously looking for vulnerabilities, shipping security updates on a regular schedule

Don't have to wait for a major release to get the benefit of the security work we're doing

Try this at home…please!

Evaluate whether the benefit of the monster test pass for service packs and major revisions is really required for security fixes

It's not nice to force customers to pay for an upgrade to get security fixes

Just because they were found internally doesn't mean they are not known externally

Customers shouldn't have to be exposed for a year if the fix is already checked in and just waiting for the right ship vehicle to be ready

Lies, damned lies, and statistics

Using numbers makes you smarter

Managers Need Data

Answers questions like:

“Should I be worried?” (Yes.)

“Are we getting better?”

“What is the top priority?”

“When will we get there?”

Metrics for Success

"Show me how you'll measure me, and I'll show you how I'll perform." – Eli Goldratt; physicist

How should we measure success and prioritize effort?

Just counting bugs doesn't work.

And it doesn't help the industry:

- Provides incentive to group bugs unhelpfully
- Provides incentive to keep quiet about bugs not otherwise disclosed

You don't want those incentives!

Metrics for Success (cont.)

What metrics describe user safety for Mozilla?

Mozilla's metrics:

- Severity
- Find Rate/Fix Rate
- Time to Fix
- Time to Deploy

What are your metrics?

Severity

Helps us prioritize what to fix first, and when to ship an emergency update

Every bug with any security risk gets fixed, even low – often easier to fix than prove exploitable

No industry standard for severity ratings – but there probably should be!

Consistent with ourselves over time

Mozilla Severity Ratings

Critical: Vulnerability can be used to run attacker code and install software, requiring no user interaction beyond normal browsing

High: Vulnerability can be used to gather sensitive data from sites in other windows or inject data or code into those sites, requiring no more than normal browsing actions

Mozilla Severity Ratings (cont.)

Moderate: Vulnerabilities that would otherwise be High or Critical except they only work in uncommon non-default configurations or require the user to perform complicated and/or unlikely steps

Low: Minor security vulnerabilities such as Denial of Service attacks, minor data leaks, or spoofs

Find Rate

How many security bugs have we found? How severe in aggregate?

What methods were most productive? Quantity and severity both count

Are some methods inefficient?

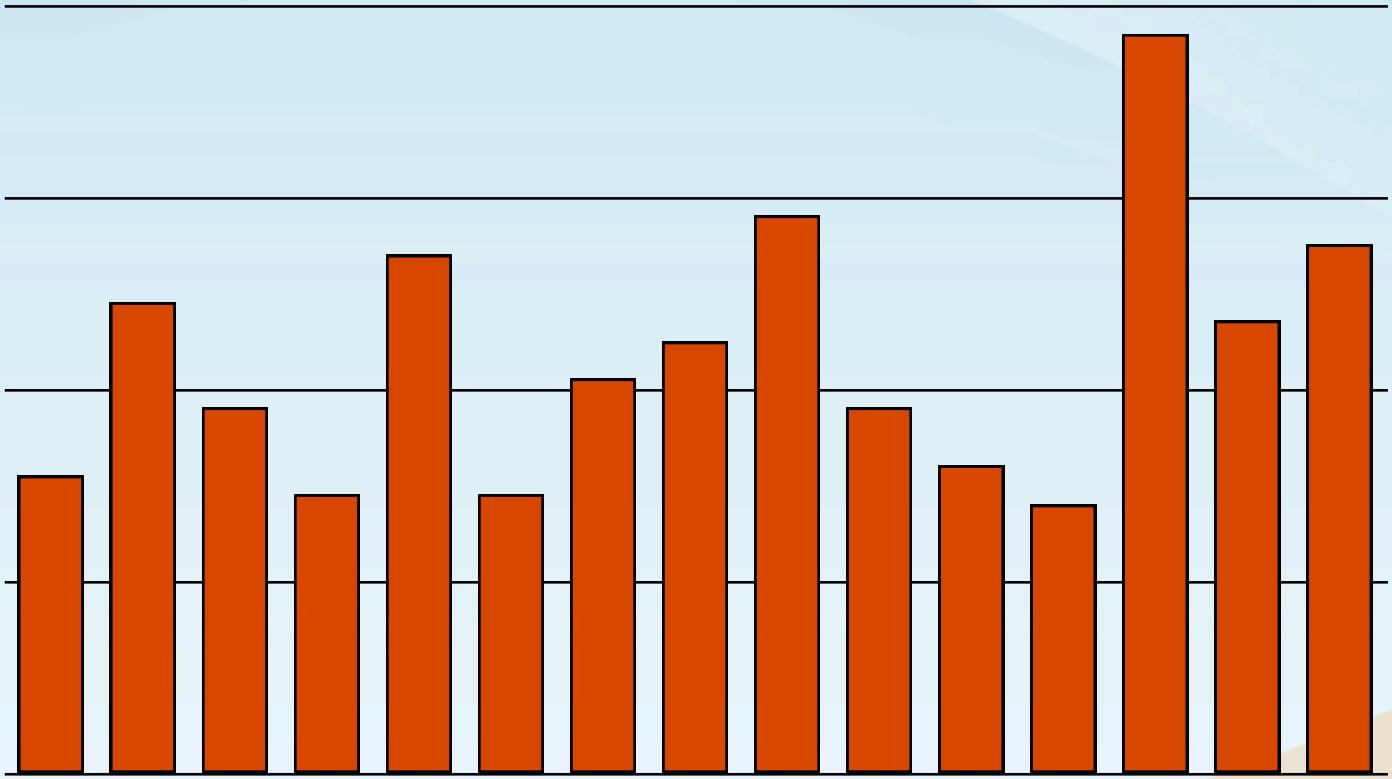
- Automated source code analysis: high number of false positives (one tool was 0 for ~300!)

Who is really good at finding security bugs?

How do we scale?

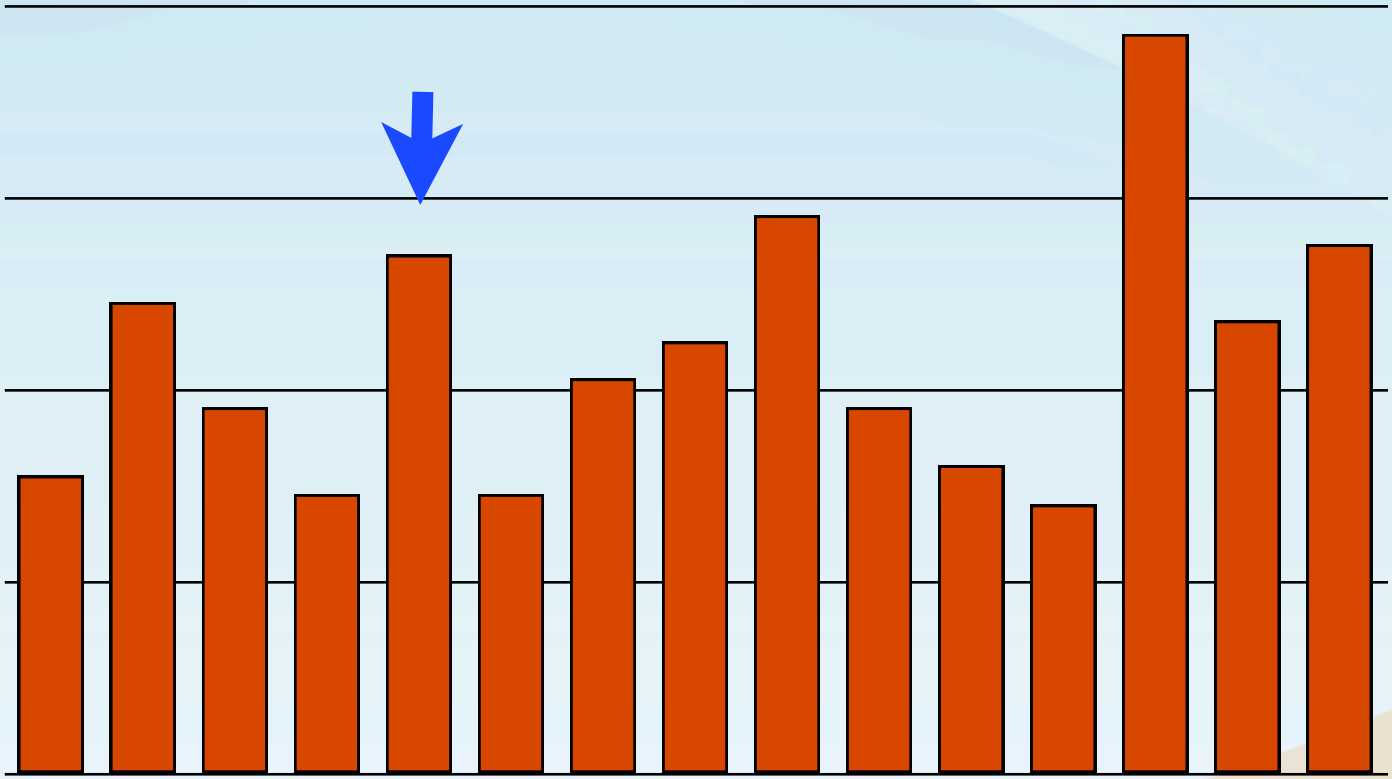
Pretty Chart: Find Rate

Find rate by month, Jan 06 – Mar 07



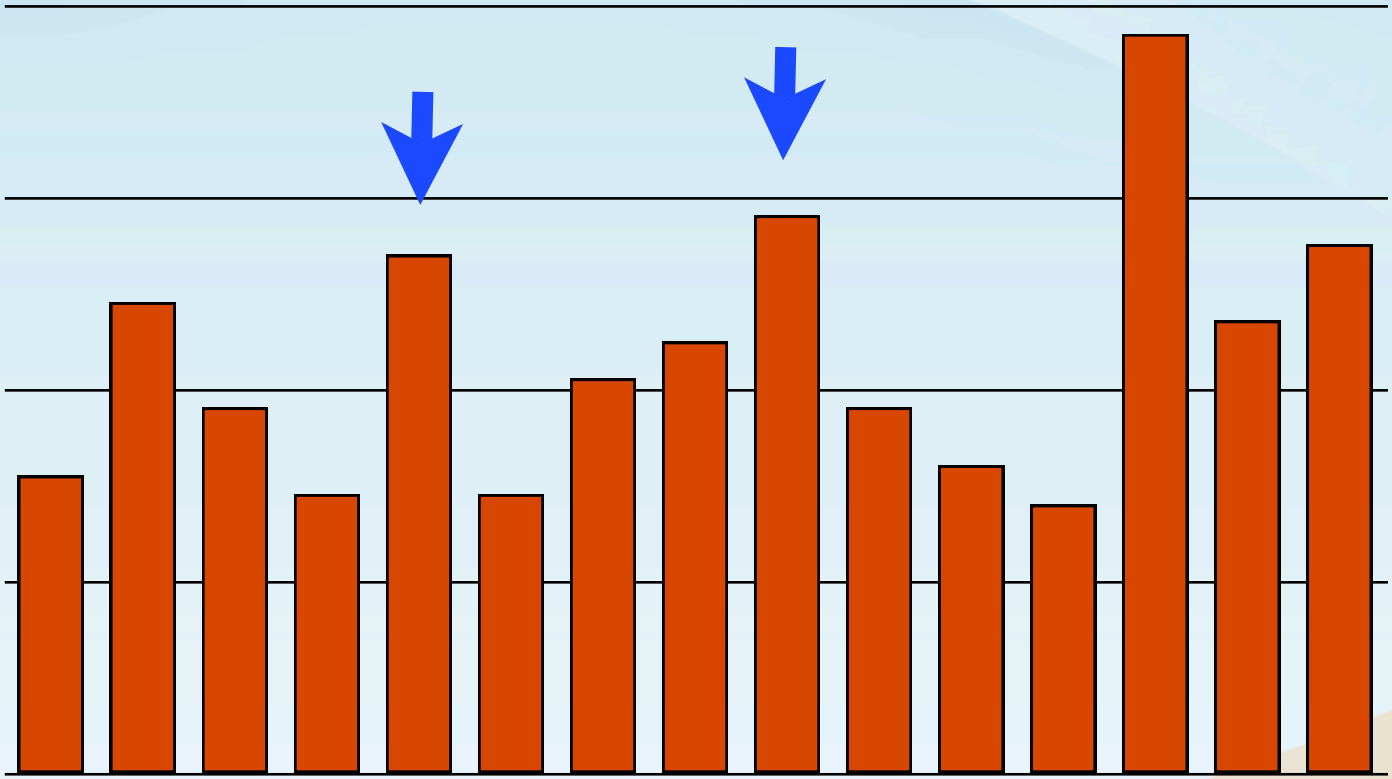
Pretty Chart: Find Rate

Find rate by month, Jan 06 – Mar 07



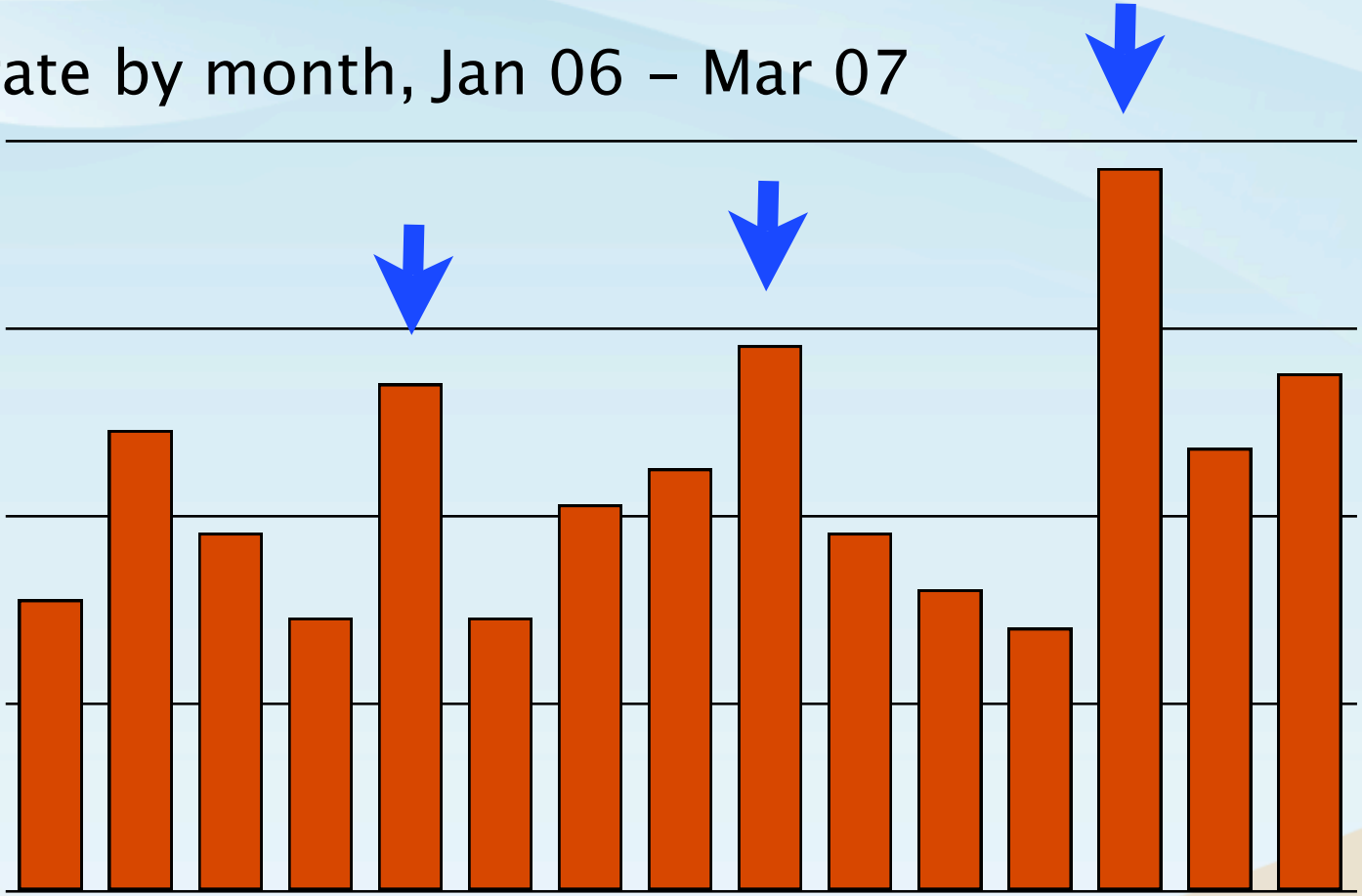
Pretty Chart: Find Rate

Find rate by month, Jan 06 – Mar 07



Pretty Chart: Find Rate

Find rate by month, Jan 06 – Mar 07



(A brief interlude about tools)

"What methods were most productive?"
- Window Snyder

"What happens when I press here?"
- Jesse Ruderman

"Why do we even have that button?"
- Various Mozilla hackers

Tools capture expertise so that non-experts can behave more like experts

Fix Rate

How long does it take to fix bugs?

Which are hardest to fix?

Which components have the highest concentration of bugs?

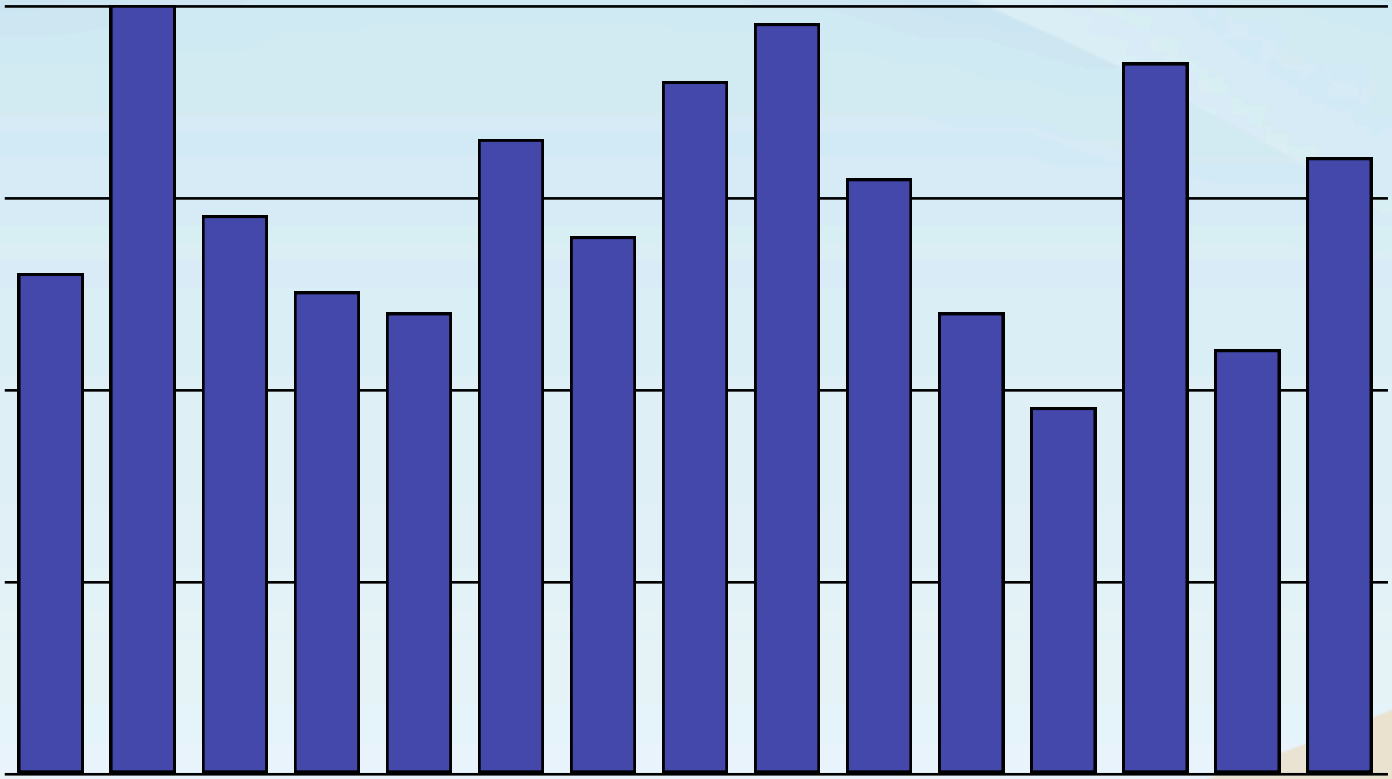
Can we fix many bugs with a single architecture change?

Are we finding faster than we can fix?

Regressions? (part of the cost of the fix)

Pretty Chart: Fix Rate

Fix rate by month, Jan 06 – Mar 07



Window of Risk

Two factors:

1. How long does it take to fix the security vulnerability?

2. How long does it take for users to get the patch installed?

Users don't care why they're vulnerable, and neither do attackers

Time to Fix

Once a vulnerability is identified, how long does it take a vendor to ship a patch?

Are we getting better over time?

Community Support

- Nightly builds tested by 20,000 people
- Users, developers, security researchers

Time to Deploy

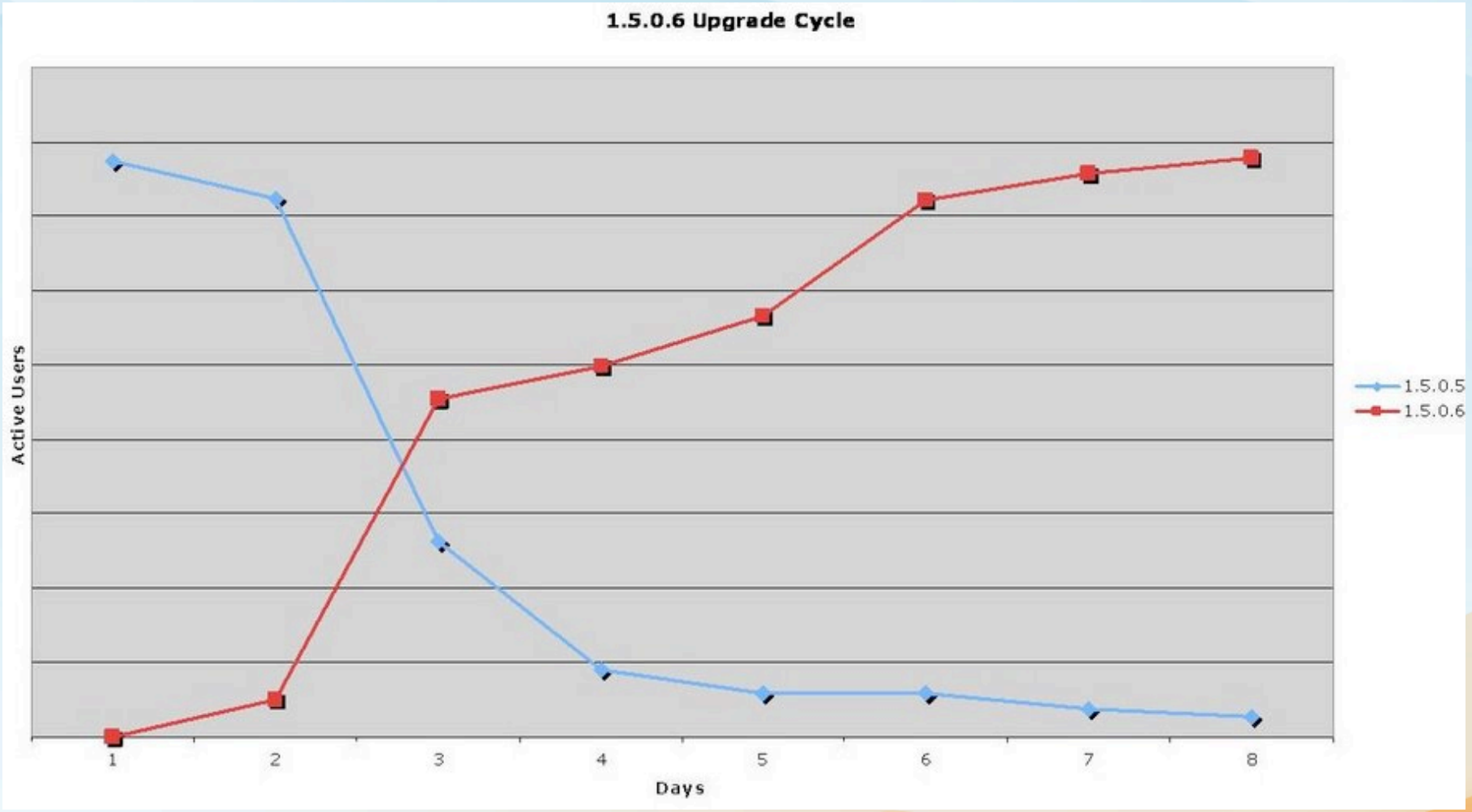
How long does it takes for users to get a patch installed once the fix is available from the vendor?

Auto-update is:

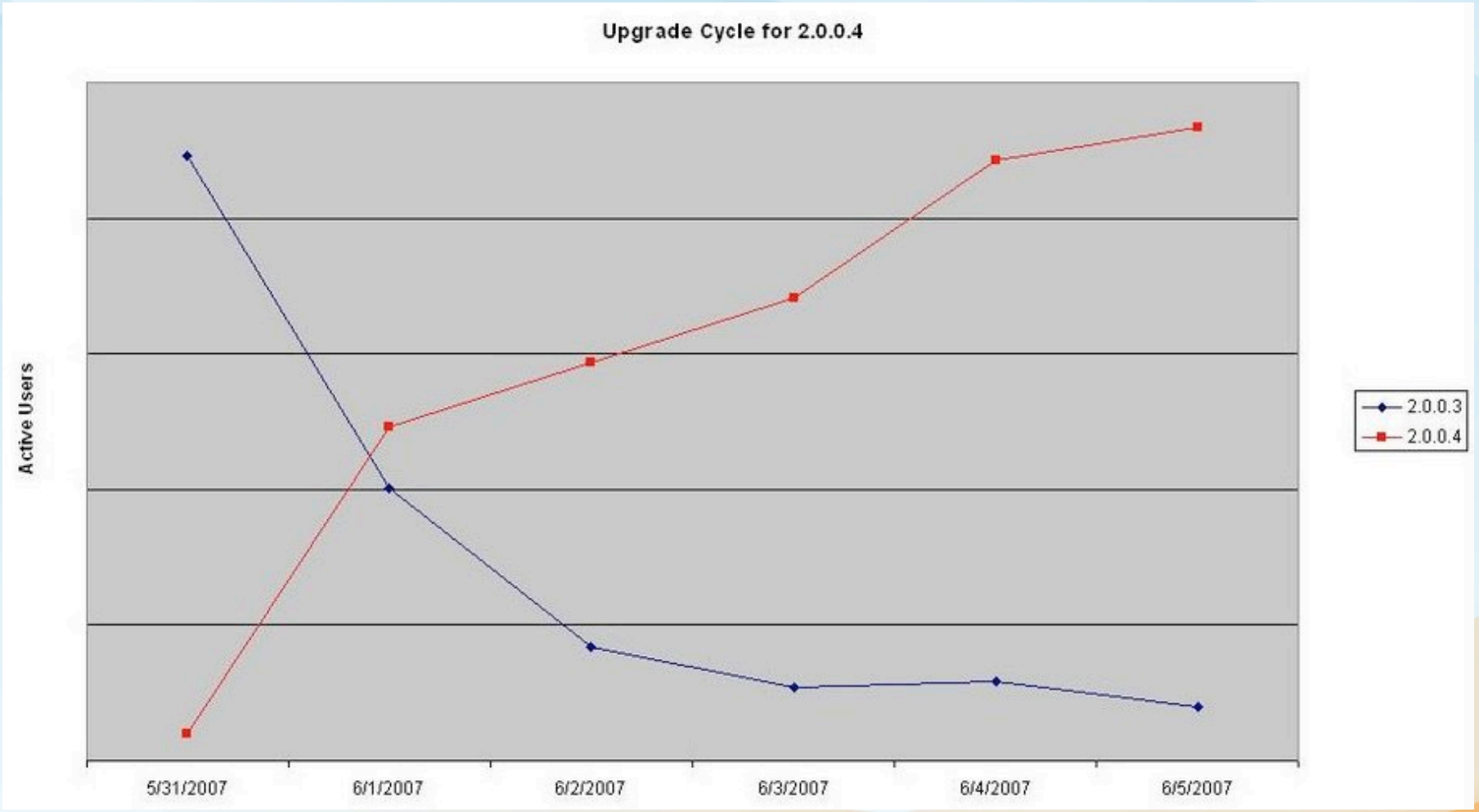
- vital for users; and
- a source of useful data for us

Measuring active users via AUS requests

Upgrade Cycle for 1.5.0.6



Upgrade Cycle for 2.0.0.4



Time to Deploy

Reduced time to deploy by 25% this year

Users get patches faster, stay safer

90% of active users updated within six days

In your development environment

These metrics apply to most software projects

Reduce FUD about number of vulnerabilities

Maybe there are more because you've gotten better at finding them…

Track progress over time – make pretty charts

Predict the future!

Security stuff from the future

A product designer's work is never done

Designing Firefox for Security

What are the key user tasks for security?

How can we make them better?

How can we help users help us help users?

Key User Task: Apply an Update

We want to optimize time-to-deploy, remember!

The “last mile” is in the hands of the user

Why do users decline updates?

- Too intrusive (“when I’m done with this blog post”)
- Worried about things breaking

Session restore is a security feature

API stability is a security feature

Security in Firefox 3

Enhanced phishing and malware protection

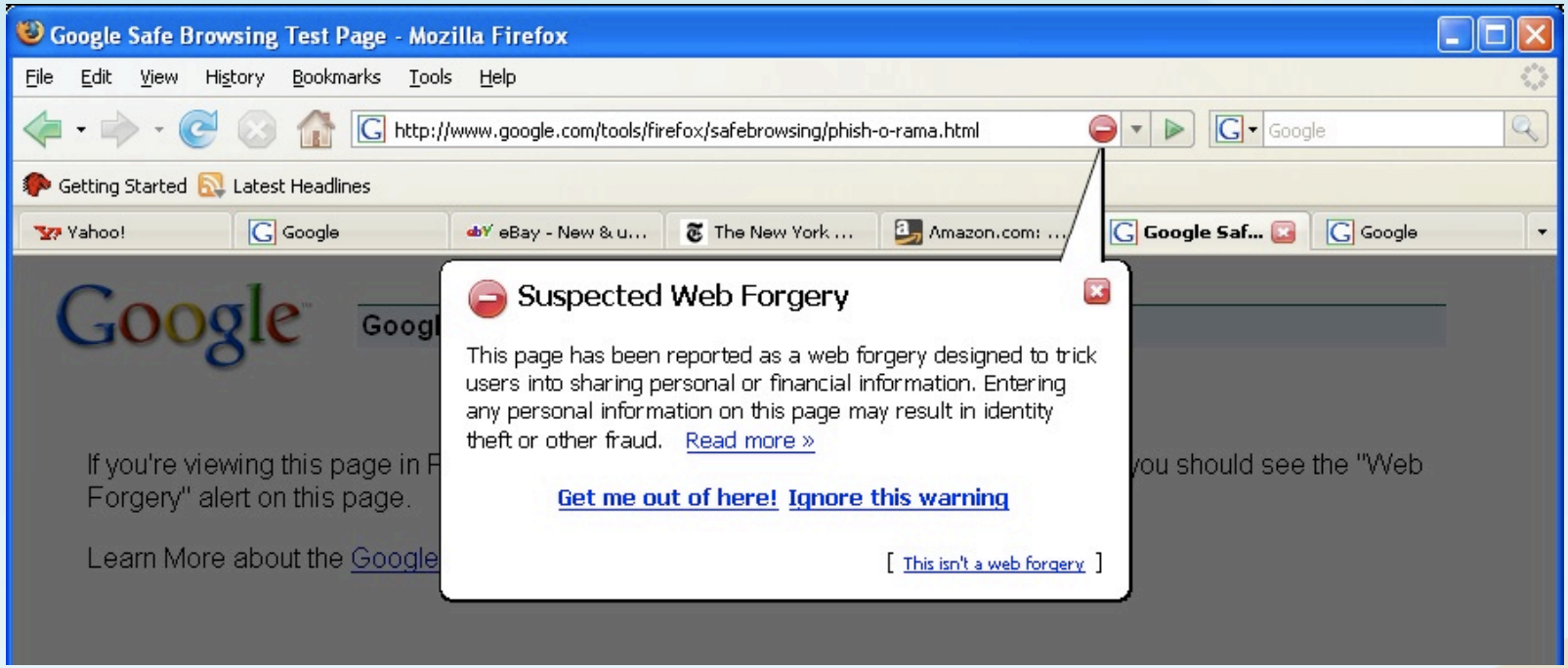
Extended Validation Certificates

Moving components to managed code

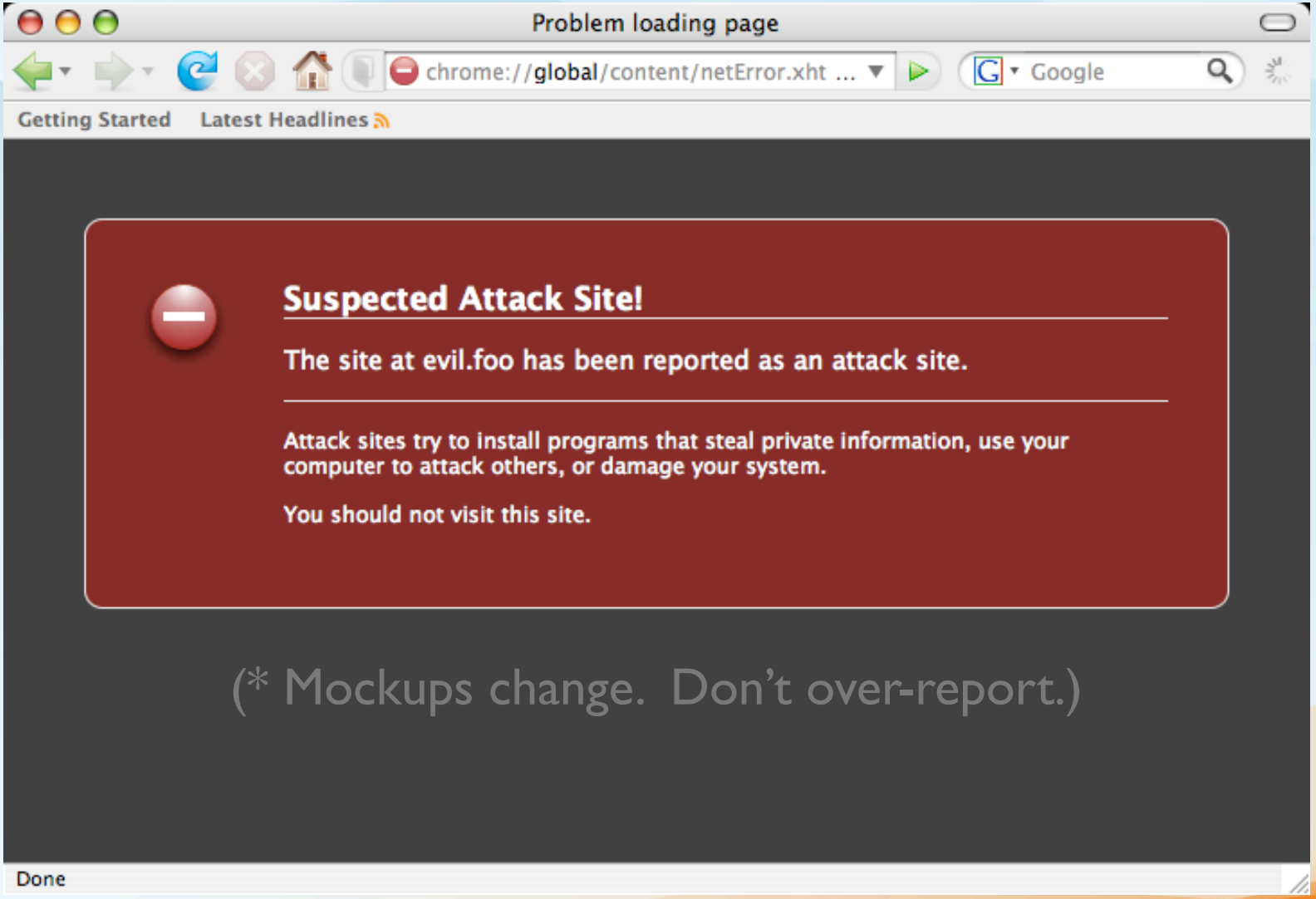
Security UI

Under the hood

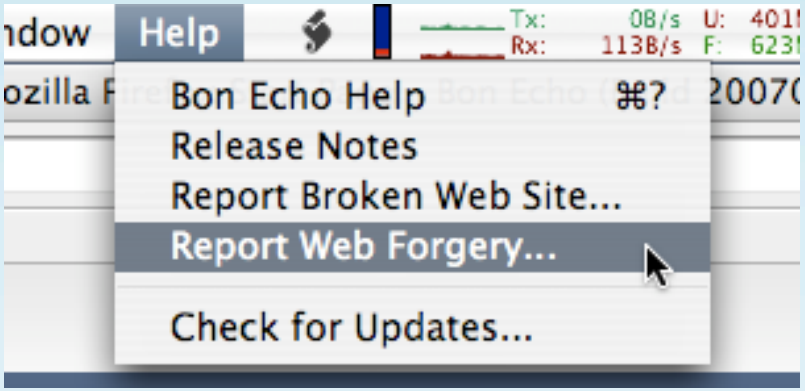
Protect against phishing



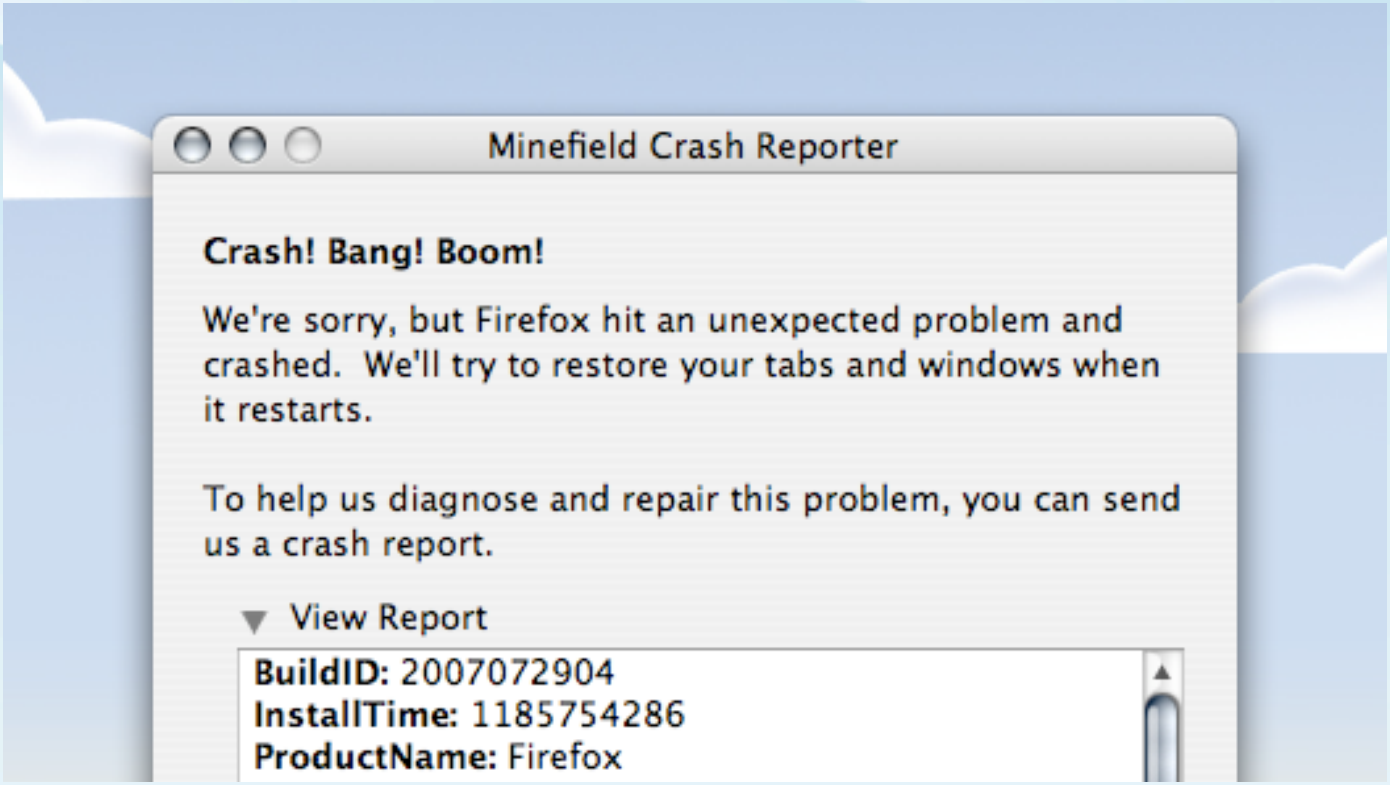
...and malware/attack sites



Help users help us help users (!)



Help users help us help users (!)



Extended Validation Certificates

SSL certificates intended to verify identity

Except that not verifying very well improves business for CAs (lower cost, high margin)

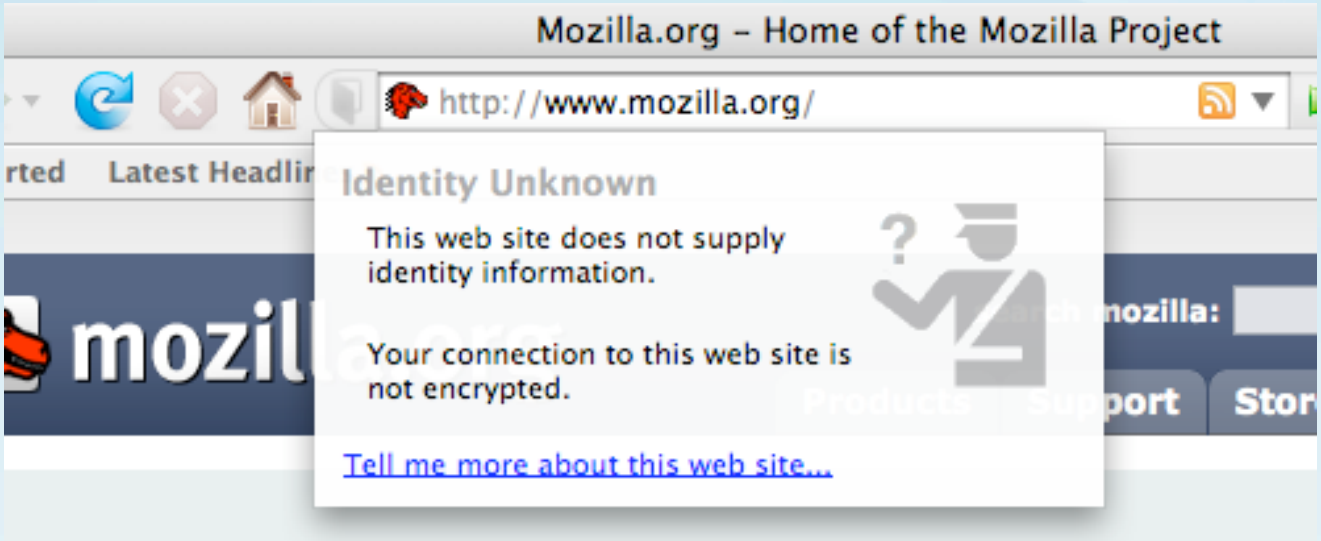
EV Certs are more thoroughly validated (higher confidence in site identity)

Meet Larry

Larry shows site identity

(* Mockups change. Don't over-report.)

...or that we don't know much



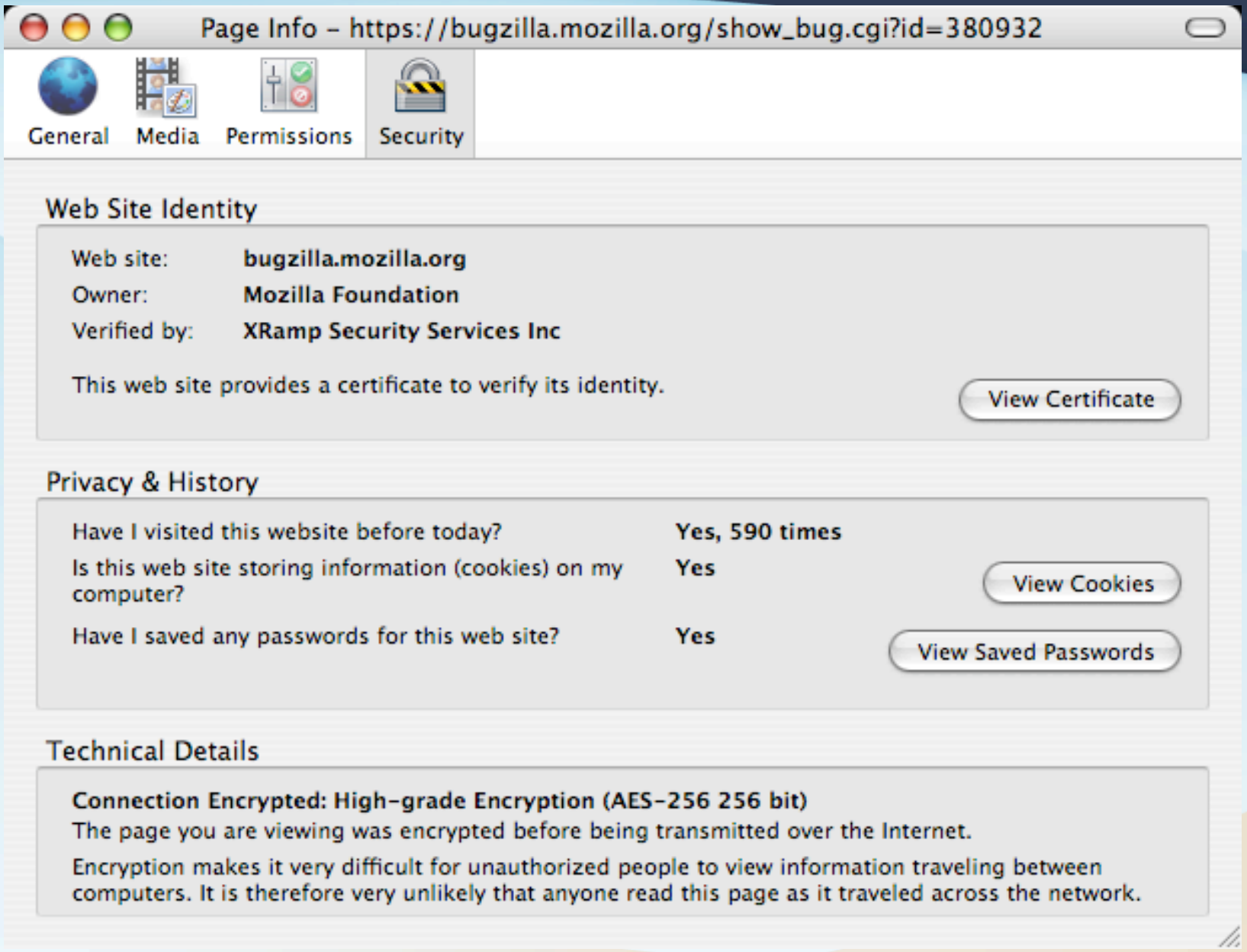
(* Mockups change. Don't over-report.)

Security User Interface

Better indication of

- Encryption
- Identity
- Previous interaction
- Knowledge of site
- Security/privacy context

- Summary of security signals
- Certificate presentation
- Dialogs and alerts


Page Info – https://bugzilla.mozilla.org/show_bug.cgi?id=380932
General
Media
Permissions
Security
Web Site Identity
Web site: bugzilla.mozilla.org
Owner: Mozilla Foundation
Verified by: XRamp Security Services Inc
This web site provides a certificate to verify its identity.
View Certificate
Privacy & History
Have I visited this website before today? Yes, 590 times
Is this web site storing information (cookies) on my computer? Yes
View Cookies
Have I saved any passwords for this web site? Yes
View Saved Passwords
Technical Details
Connection Encrypted: High-grade Encryption (AES-256 256 bit)
The page you are viewing was encrypted before being transmitted over the Internet.
Encryption makes it very difficult for unauthorized people to view information traveling between computers. It is therefore very unlikely that anyone read this page as it traveled across the network.

Under the Hood

Reflow rewritten, large test suite added (improve content and DOM resilience)

Simplifying and robustificating handling of events (defend against race condition attacks)

Cross-Origin wrappers (block “chrome” escalation)

Cycle collector (centralized memory management for correctness)

Moving to cairo (shared resource with other projects, large test base)

Mozilla2

- JS2 via Tamarin provides JITing VM
  - move more code from fragile C++ to managed JS
- Replacing Mozilla-only C++ with standards
  - libraries safer, easier ramp-up by new developers
- DeCOMtamination via Oink and friends
  - better performance and static analysis
- Tool- and run-time security properties
- Even faster and fancier text and graphics
- Whitens teeth, still low-carb

Tools: free to every attendee!

Tools

Mozilla creates security tools to test Mozilla products.

HTTP Fuzzer

FTP Fuzzer

Collaboration with Leviathan and Matasano

Javascript Fuzzer

Mozilla Internal Tools

But they can be useful to other environments!

Sharing Tools

- Securing large software projects is difficult
- Most commercial vendors build internal tools, but are reluctant to make public
- Other development environments can benefit from security work at Mozilla
- This is the first set of security testing tools to be released

Sharing tools responsibly

Engaged other browser vendors in May (Microsoft, Apple, Opera)

Give everyone a chance to protect their users

Release tools once everyone has time to evaluate, react, and respond

Solicit and incorporate feedback

Iterate with other tools and updates

Protocol Fuzzers

These tools can be used to identify problems in code that implements HTTP or FTP

Not specific to Firefox

HTTP Protocol Fuzzer – Michael Eddington

Emulates an HTTP server to test how an HTTP client handles unexpected input.

Written in Python on top of the Peach Fuzzing Framework.

http.py
– Test case generation

Httpfuzzer.py
– Serves test cases to browser

Httpfuzzer.html
– Drives browser to test server

Gentestcases.py
– Creates file for each test case containing HTTP message

FTP Protocol Fuzzers- Michael Eddington

Client and Listing fuzzers

Emulate server to test how an FTP client handles unexpected data

Built using RACKET ruby fuzzing framework

server.rb FTP fuzzing server

fuzz.rb RACKET fuzzing library

list.rb FTP listing module

JavaScript Fuzzer – Jesse Ruderman

jsfunfuzz creates JavaScript function bodies and runs them. (Also decompiles them!)

Creates the functions using a bunch of mutually recursive functions:

makeStatement
makeExpr
makeFunction
makeSwitchBody
makeTryBlock
...

JavaScript Fuzzer – Jesse Ruderman

Found 280 bugs in Firefox (~27 exploitable):

- It knows a lot about the JavaScript language
- It breaks all the rules
- It is not scared to nest very deeply
- It can accumulate state
- It tests correctness, not just crashes
- It works when Jesse is sleeping

JavaScript Fuzzer – Jesse Ruderman

Bug 352606

```
y = ({toString: gc});
new Function("y--;")();
```

Bug 353079

```
for (let a in [1]) let (x) { for(let y in
((function(id2) { return id2; })('')))
{ } }
```

Bug 361346

```
this.x setter= new Function;
this.watch('x', function(){});
gc();
x = {};
```

Get Mozilla security tools

Permanent home coming soon! Watch the Mozilla Security Blog for details: http://blog.mozilla.com/security/

JavaScript fuzzer lives in bug “jsfunfuzz”:

https://bugzilla.mozilla.org/show_bug.cgi?id=jsfunfuzz

Mozilla Security Sites

Security Blog

http://blog.mozilla.com/security/

Security Advisories

http://www.mozilla.org/projects/security/known-vulnerabilities.html

Security Projects

http://www.mozilla.org/projects/security/

Get Involved

How?

- Spread the word!

spreadfirefox.com

- Give us feedback
- Write an add-on

developer.mozilla.org

- Become a contributor
- Join MoCo!

Security folks like you

- Design
- Implementation
- Code review and penetration testing
- Develop tools
- Report bugs
- Run nightlies

Thank You

window@mozilla.com

shaver@mozilla.com