The unsatisfied Security Requests of the Web

Bastian Braun

OWASP Stammtisch Munich

Oct 20 2015
Jan 19 2016
about: me

• Diploma Computer Science 2006
  – RWTH Aachen
  – Bachelor in Economics (nobody is perfect ;-) )

• University of Hamburg 2006-08
  – Security in Distributed Systems Research Group

• University of Passau 2008-15
  – Institute of IT-Security and Security Law
  – finished with PhD

• mgm security partners since March 2015
  – TCFKA SecureNet
  – Senior Consultant IT Security
Background: The Web Then and Now

• The web 1990
  – network of few trusted peers
  – providing static public content for read-only access

• The web today
  – stateful web applications
    • ongoing trend: business logic moves to the browser
  – distributed application platform
  – personalized accounts
The Web – A Success Story

• The beginning: hypertexts as “a single user interface to large classes of information” [Tim Berners-Lee, 1990]

• Evolvement driven by use cases & business models
  – CGI, database access → dynamic web pages
  – JavaScript, cookies, plug-ins → powerful clients
The Web – A Success Story

- Protocol: evolutionary steps since HTTP 0.9 (1991)
  - methods, e.g. POST, PUT, OPTIONS, HEAD, DELETE (v1.0)
  - performance improvements by re-usable TCP connections (v1.1)
  - request multiplexing, improved data compression, server-side content pushing (v2.0)
  - compensations for increased web traffic

- HTTP’s statelessness as its secret of success
  - simplicity
  - expandability
  - fault tolerance
Security Requests of the Web

• personalized accounts
  – Mutual app-user authentication
  – Message integrity & confidentiality

• distributed application platform
  – Secure cross-domain communication

• stateful web applications
  – Control-flow integrity
Mutual App-User Authentication

• The user and the web application can mutually verify the other’s identity.
• Improper user authentication gives attackers access to personal user accounts.
• Improper web application authentication facilitates spoofing attacks.
Message Integrity & Confidentiality

• Message integrity
  – The recipient can verify that the message has not been tampered with.

• Message confidentiality
  – Only the authorized recipient can read the message content.
Secure Cross-domain Communication

• All messages
  – invoked by a web application on behalf of the user
  – targeting another web application
  must be authorized by the user.

• Insecure cross-domain communication allows
  user impersonation by cross-site request forgery and clickjacking attacks.
Control-flow Integrity (CFI)

• Processing HTTP requests may change a web application’s state. A web application preserves CFI if it prevents arbitrary state changes.

• A successful attack can bypass the application’s business logic.
Web-based Secure Application Control

**Higher Layers**
- mutual app-user authentication
- sec. cross-domain communication
- control-flow integrity

**TLS/SSL**
- app2browser authentication
- message integrity
- message confidentiality
Mutual App-User Authentication

• User authentication: client-side SSL possible but failed because of bad usability
  – client-side SSL cannot prevent website spoofing

• App authentication: browser authenticates the domain, the user authenticates the content; missing recognizability
Mutual App-User Authentication

• Consequences of missing user authentication + gap in app authentication:

Active Phishing Sites (dark grey) and Email Phishing Campaigns (pale grey),
src: Anti-Phishing Working Group (APWG) Reports
Approach: “Lightweight Client-side SSL”

- **Init**: store cryptographic token in the browser’s localStorage
- **For each login**: deliver challenge with login form
- **compute response** as HMAC of challenge and token
- **only correct** (user,pwd,response) allow account access
PhishSafe: Evaluation

• Implementation: 2 static JS files + application-specific server-side AccountManager module

• Features
  – does not suffer the chicken-and-egg problem
  – binds authentication token to browser-verifiable domain

• Downside
  – relies on the confidentiality + availability of the user’s email account
Web-based Secure Application Control

**Higher Layers**
- mutual app-user authentication
- sec. cross-domain communication
- control-flow integrity

**TLS/SSL**
- app2browser authentication
- message integrity
- message confidentiality
Secure Cross-domain Communication

• Every webpage can embed content from other websites – visibly or invisibly.
• A server-side function call is triggered
  – upon loading the content
  – upon clicking (also invisible, transparent) page elements.
• The web browser automatically authenticates those function calls on behalf of the user.
Approach: User-level Authentication of Function Calls

- Function calls classified as critical must be explicitly authenticated by the user.
Session Imagination: Evaluation

• Implementation: one server-side module, included into critical functions

• Features
  – does not suffer the chicken-and-egg problem
  – as secure as password-based re-authentication but more user-friendly (proven by case study)

• Downside
  – displaying the images on mobile devices has bad usability
Web-based Secure Application Control

**Higher Layers**
- mutual app-user authentication
- sec. cross-domain communication
- control-flow integrity

**TLS/SSL**
- app2browser authentication
- message integrity
- message confidentiality
Control-flow Integrity

• Every user can trigger server-side function calls in random order and arbitrarily often
  – repeated executions (i.e. exceeding limits)
  – omitted workflow steps
  – functions and parameters technically not limited

• The web browser cannot limit the set of available functions and parameters.
Approach: Replicate User Actions to Determine Crafted Function Calls

- Server-side replica whitelists function calls triggered by mouse clicks
Ghostrail: Evaluation

• Implementation: Node.js-based reverse proxy, WebKit-based headless browser as replica

• Features
  – no policy definition necessary
  – integration into web application firewalls possible

• Downside
  – scaling difficult for high-traffic and real-time applications
Summary: The Browser’s Role on Meeting Higher-layer Needs

• The browser authenticates the domain while the user authenticates the content.
• It transparently authenticates cross-domain function calls and allows UI manipulations.
• It does not limit the available functions according to the application’s state.
Approach: Complement the Browser with Secure Authentication Module

• Browser forwards critical function calls to external authentication module
• Module authenticates web app
• User acknowledges function calls via unspoofable interface
• Module authorizes function call on behalf of the user by adding authentication proof
Approach: Complement the Browser with Secure Authentication Module

Unique URL

Compute Shared Secret using Diffie-Hellman Key Exchange
Provide Comprehensible Description of Critical Functions

Usual Web Surfing

Request Critical Function + signed server challenge
Ask for user consent using function description

Enter URL, password
Discard password

Request Critical Function + signed client response

Request Critical Function + signed client response
MobileAuthenticator: Evaluation

• Implementation: client-side app, server-side module, JavaScript-based broker

• Features
  – avoids sending credentials, e.g. passwords
  – device-specific credentials, no more password entry
  – one client-side installation for all web apps

• Downside
  – requires installation on client side + server-side support
The Big Picture

• We used a combination of browser-level token, user-level knowledge, and SSL to
  – approach the authentication gap for mutual app-user authentication
  – approach secure cross-domain communication

• The latter case has a negative impact on usability while the first case is transparent for the user.
The Big Picture

• Control-flow integrity cannot be preserved on the client side.

• Single-purpose apps can achieve similar security properties as our multi-purpose MobileAuthenticator – but not solve the CFI problem because of HTTP
  – e.g. online banking software, mobile apps
  – MobileAuthenticator challenges might serve as a control mechanism
Conclusion

• Omitting client-side SSL leaves a gap in app authentication
• Re-usable credentials cannot provide secure user authentication
  – e.g. passwords, token-generators like RSA SecurID
• Secure cross-domain communication requires user involvement
• Due to their RESTful nature, web apps need no service description – but an enforcement to only legal next function calls
Outlook

• redesign of code distribution
  – e.g. AngularJS, gwt

• new basis: Quic & IPv6

• scenarios in smart worlds
  – worlds ⊆ \{home, city, car, industry\}

• frameworks with built-in security features
  – CSRF tokens well-established, XSS increasing, more to come
References

• thesis: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:739-opus4-3048

• papers: https://web.sec.uni-passau.de/members/bastian/index.php
Q & A

bastian.braun@owasp.org