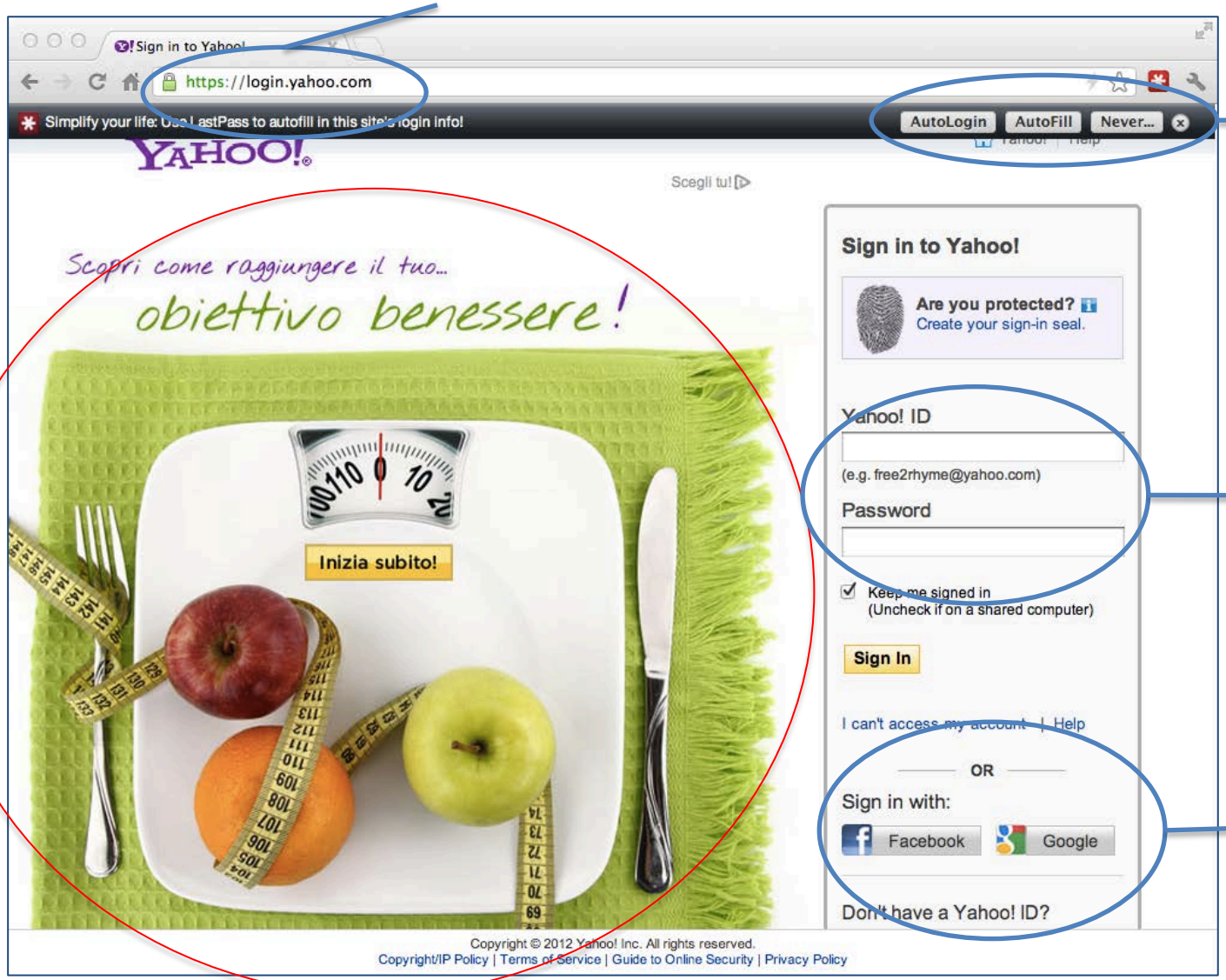
Design, Verification and Implementation of Secure Web Applications

Antoine Delignat-Lavaud
PROSECCO

Security on the Web

HTTPS Protocol *(HTTP over TLS)*



Advert *(Untrusted)*

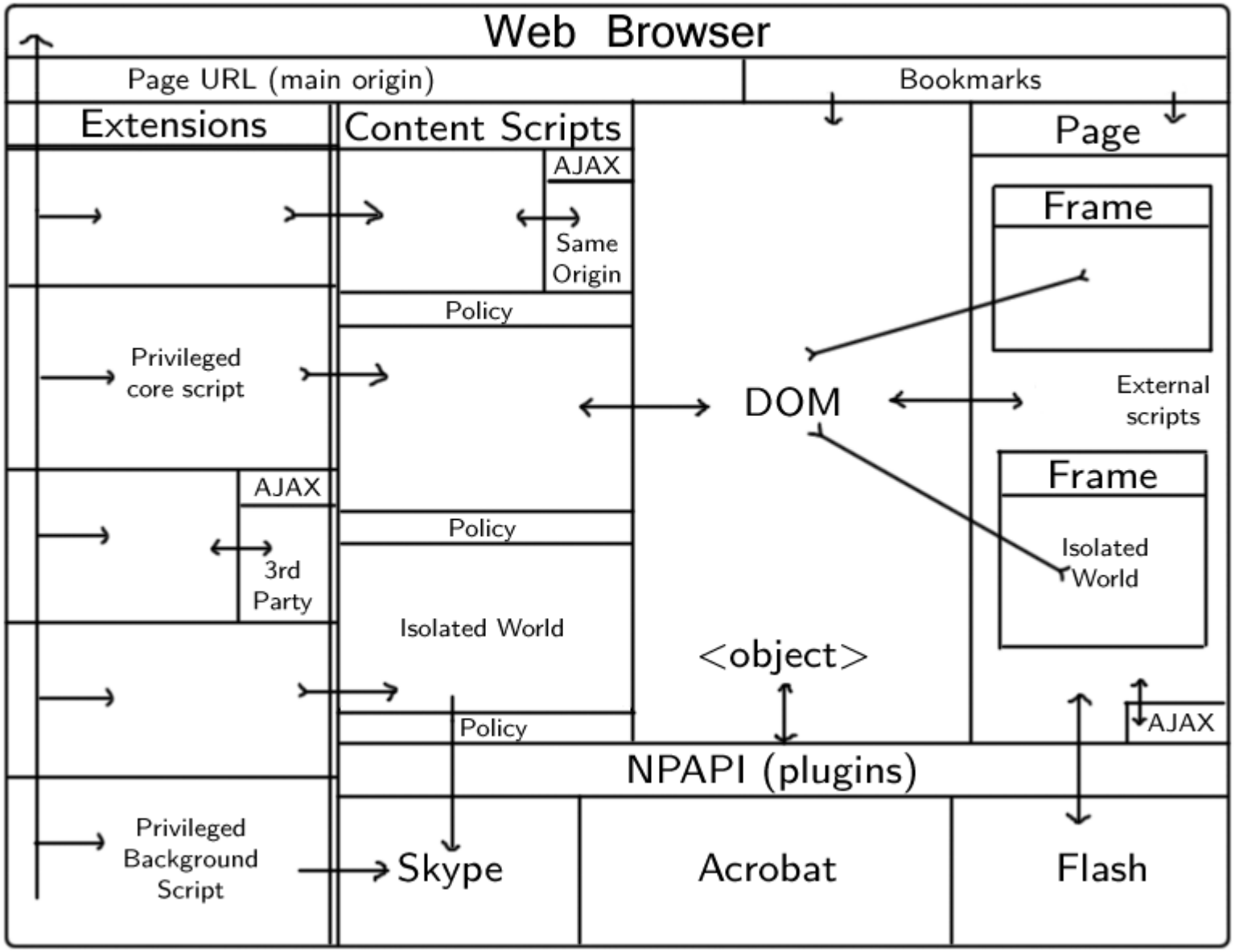
Password Manager *(Encrypted Cloud Storage)*

Login Form *(password)*

Single Sign-On *(OAuth, OpenID)*

Why is Web Security Difficult?

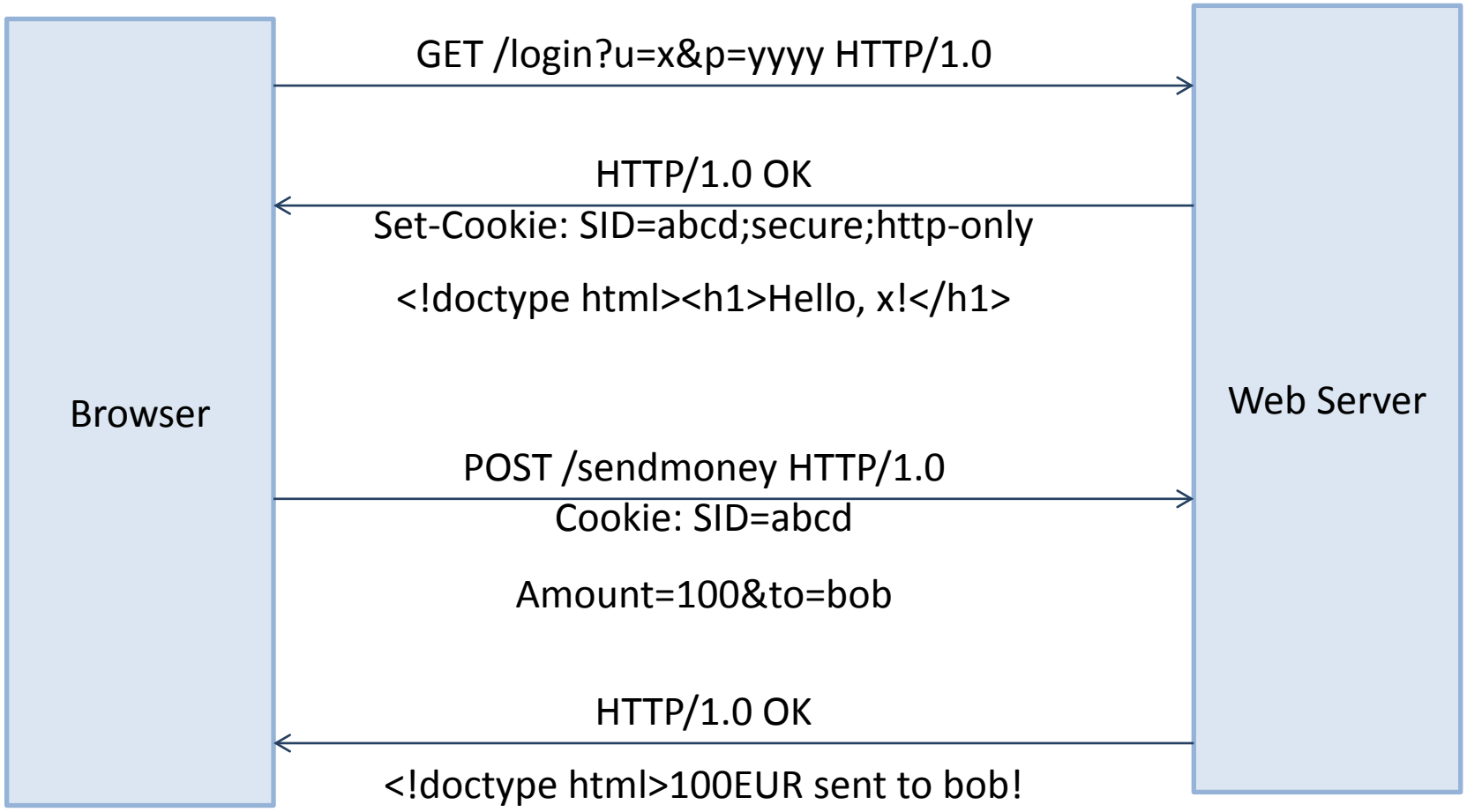
- Extremely powerful attacker
- Confusion between code and data
- Interaction between many principals


Web Browser
Page URL (main origin)
Bookmarks
Extensions
Content Scripts
Page
AJAX
Same Origin
Frame
Policy
Privileged core script
DOM
External scripts
AJAX
3rd Party
Policy
Frame
Isolated World
Isolated World
<object>
Policy
AJAX
NPAPI (plugins)
Privileged Background Script
Skype
Acrobat
Flash

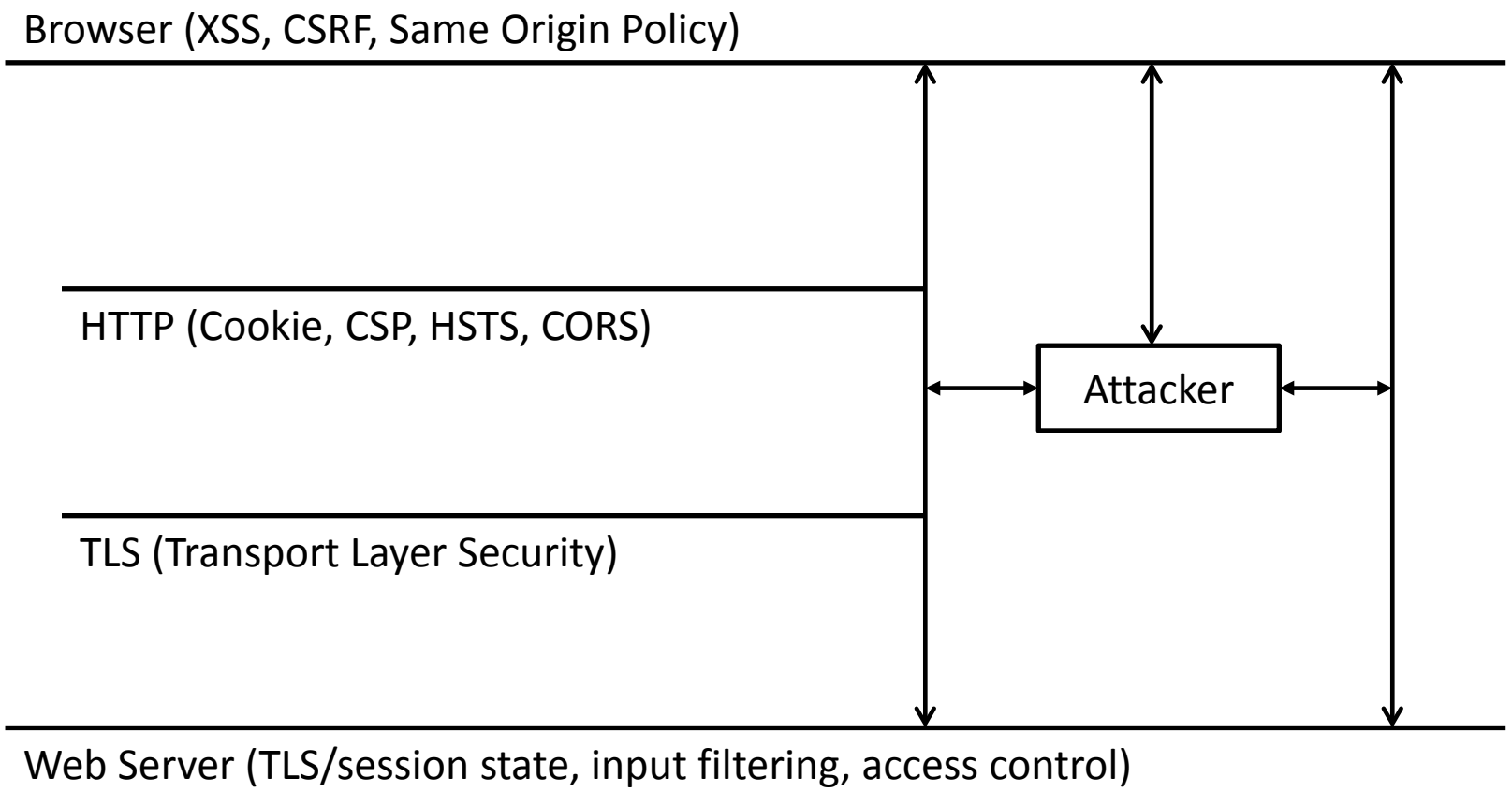
Same Origin Policy

- Origin = Protocol + Domain + Port
  https://www.dropbox.com:443/login
- Frames from different origins can only communicate by text messages
- Cannot download (get the raw bytes of) a file from a different origin
- Possible to “load” pictures and scripts across origins. They become part of the “host” origin
  - Still cannot access their raw value

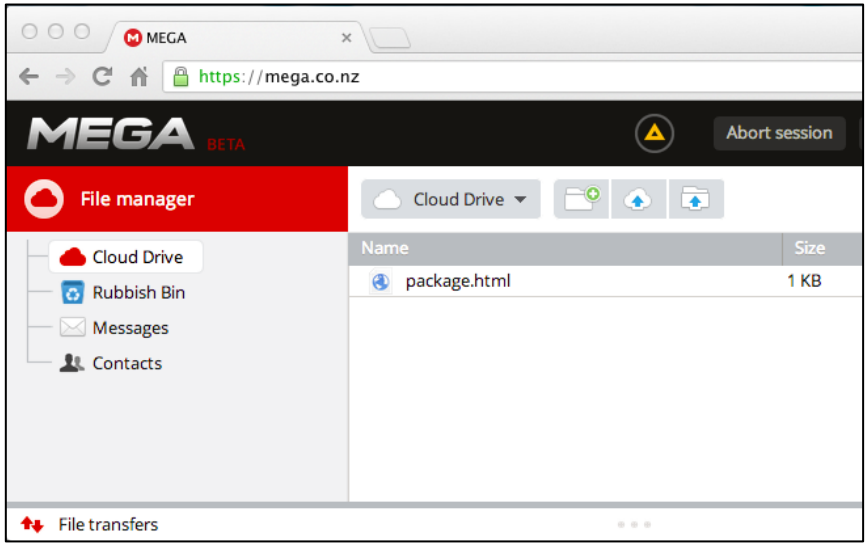
HTTP and Cookies Overview

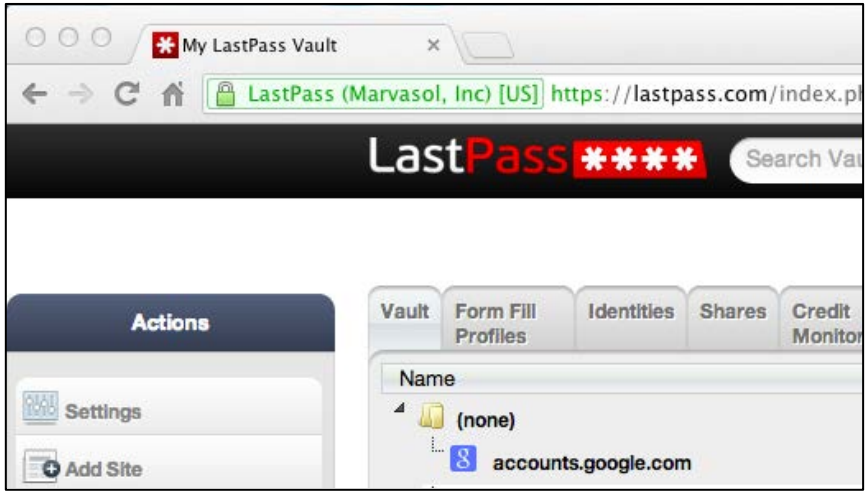


Web Security is a Multilayer Problem



Motivation: Client-Side Encryption





- Storage and retrieval of encrypted data using a **client-side encryption**
  - Cloud storage services
  - Password managers
- Long-term encryption keys never leave the client
- How to protect against encryption key leaks?
  - by other scripts on page

Motivation: Single Sign-On

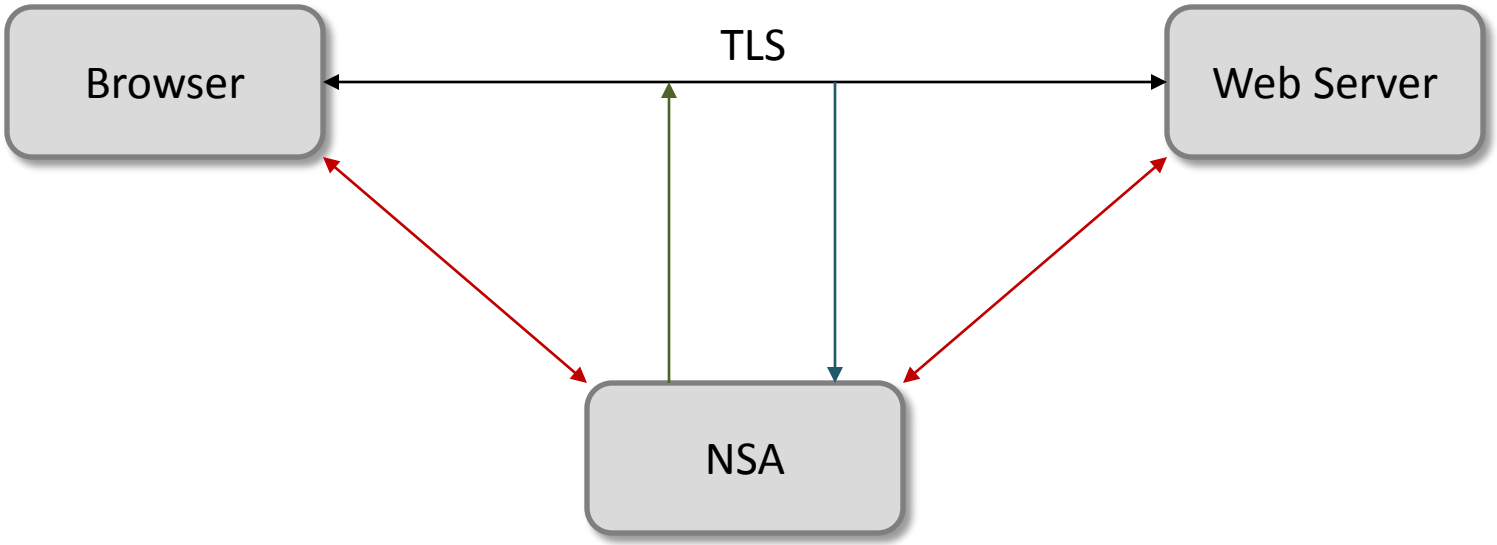
- Provides access to user's identity and social data
- **3-party authentication and authorization protocol**
- Holds **secret access** token

- How to prevent access token leaks?
  - to unauthorized hosts
  - by malicious, buggy scripts on honest hosts

PROBLEMS RELATED TO TLS

Expected TLS Guarantees

Authentication > Integrity > Confidentiality



TLS Sessions

- Handshake is expensive but HTTP requires many short exchanges
- Sessions are created at TLS level for fast resumption
- HTTP Session on top of TLS session

Attacks against TLS

- Renegotiation / cipher downgrade
- CRIME (encryption+compression)
- Problems with block ciphers
  (padding oracle / mac-and-encrypt, IV…)
- Problems with the only stream cipher (RC4)

Truncation attacks against TLS

- A. Pironti, B. Smyth (PROSECCO)
  Truncating TLS Connections to Violate Beliefs in Web Applications, WOOT'13

- A. Delignat-Lavaud, K. Bhargavan
  Truncation of HTTP headers in Chrome, Safari and Opera ($3133.7 Google bounty)

Truncation attacks against TLS

> *"failure to properly close a connection no longer requires that a session not be resumed [...] to conform with widespread implementation practice"*

RFC 5246 – TLS specification

```
POST /wire_transfer.php HTTP/1.1
Host: mybank.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 40
amount=1000&recipient=Jeanne
```

Two TLS fragments:
1)POST […] recipient=Jean
2)ne

Attack: Drop the 2nd fragment to transfer money to Jean.

Server ignores:
- termination mode
- Content-Length field

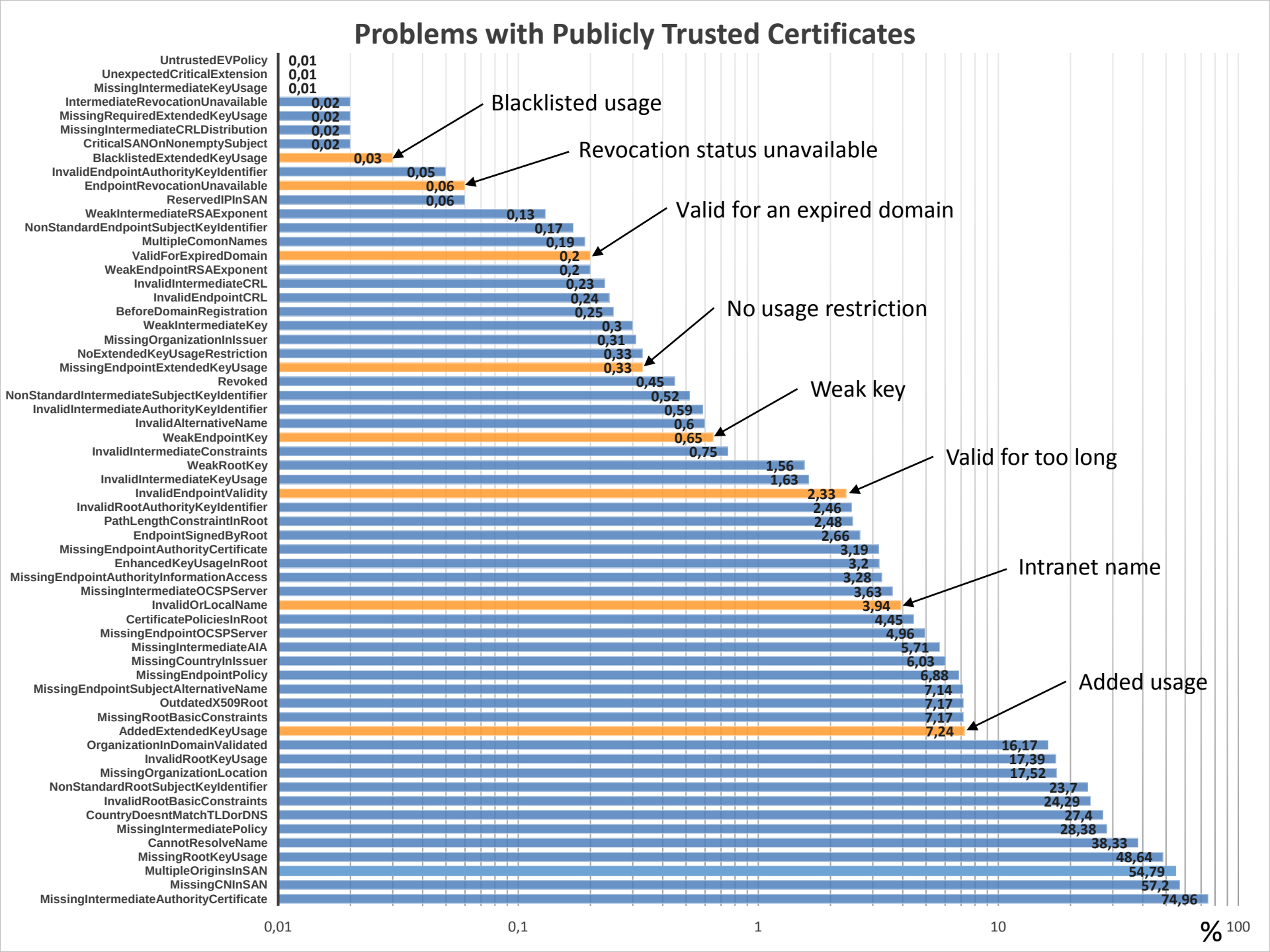
Fix:
- wire transfers upon *graceful closure* only
- check lengths

Attack works against Apache

Most browsers only offer integrity of any content prefix!

Authentication / Certificates Issues

- Untrustworthy CA (Trustwave)
- Compromised CA (DigiNotar)
- Careless delegation by CA (Comodo, …)
- Failure in purpose restriction (Turktrust, FLAME…)
- Failure of crypto (FLAME, Debian OpenSSL, …)


Problems with Publicly Trusted Certificates
UntrustedEVPolicy 0,01
UnexpectedCriticalExtension 0,01
MissingIntermediateKeyUsage 0,01
IntermediateRevocationUnavailable 0,02
MissingRequiredExtendedKeyUsage 0,02
MissingIntermediateCRLDistribution 0,02
CriticalSANOnNonemptySubject 0,02
BlacklistedExtendedKeyUsage 0,03
InvalidEndpointAuthorityKeyIdentifier 0,05
EndpointRevocationUnavailable 0,06
ReservedIPInSAN 0,06
WeakIntermediateRSAExponent 0,13
NonStandardEndpointSubjectKeyIdentifier 0,17
MultipleComonNames 0,19
ValidForExpiredDomain 0,2
WeakEndpointRSAExponent 0,2
InvalidIntermediateCRL 0,23
InvalidEndpointCRL 0,24
BeforeDomainRegistration 0,25
WeakIntermediateKey 0,3
MissingOrganizationInIssuer 0,31
NoExtendedKeyUsageRestriction 0,33
MissingEndpointExtendedKeyUsage 0,33
Revoked 0,45
NonStandardIntermediateSubjectKeyIdentifier 0,52
InvalidIntermediateAuthorityKeyIdentifier 0,59
InvalidAlternativeName 0,6
WeakEndpointKey 0,65
InvalidIntermediateConstraints 0,75
WeakRootKey 1,56
InvalidIntermediateKeyUsage 1,63
InvalidEndpointValidity 2,33
InvalidRootAuthorityKeyIdentifier 2,46
PathLengthConstraintInRoot 2,48
EndpointSignedByRoot 2,66
MissingEndpointAuthorityCertificate 3,19
EnhancedKeyUsageInRoot 3,2
MissingEndpointAuthorityInformationAccess 3,28
MissingIntermediateOCSPServer 3,63
InvalidOrLocalName 3,94
CertificatePoliciesInRoot 4,45
MissingEndpointOCSPServer 4,96
MissingIntermediateAIA 5,71
MissingCountryInIssuer 6,03
MissingEndpointPolicy 6,88
MissingEndpointSubjectAlternativeName 7,14
OutdatedX509Root 7,17
MissingRootBasicConstraints 7,17
AddedExtendedKeyUsage 7,24
OrganizationInDomainValidated 16,17
InvalidRootKeyUsage 17,39
MissingOrganizationLocation 17,52
NonStandardRootSubjectKeyIdentifier 23,7
InvalidRootBasicConstraints 24,29
CountryDoesntMatchTLDorDNS 27,4
MissingIntermediatePolicy 28,38
CannotResolveName 38,33
MissingRootKeyUsage 48,64
MultipleOriginsInSAN 54,79
MissingCNInSAN 57,2
MissingIntermediateAuthorityCertificate 74,96
Blacklisted usage
Revocation status unavailable
Valid for an expired domain
No usage restriction
Weak key
Valid for too long
Intranet name
Added usage
0,01
0,1
1
10
% 100

PROBLEMS RELATED TO THE HTTP PROTOCOL

Attacks at HTTP level

- Any website can cause the browser to send a request to any other website (**CSRF**):

- Cookies are **attached by the browser**
- How to tell whether a request was caused by the user or a malicious website?

CSRF Protection

- CSRF token (stored in cookie/local storage)
  Can it be stolen?

- **Origin** HTTP header
  Added to POST-over-HTTPS and AJAX requests

Problems with Cookies

- **Access policy is not based on origin:** domain suffix + path + secure flag

- SID=xxx;domain=.dropbox.com, path=/, expires=0, secure, http-only

- Secure flag is for reading the cookie (can set secure cookie over HTTPS)

Cookie Issues

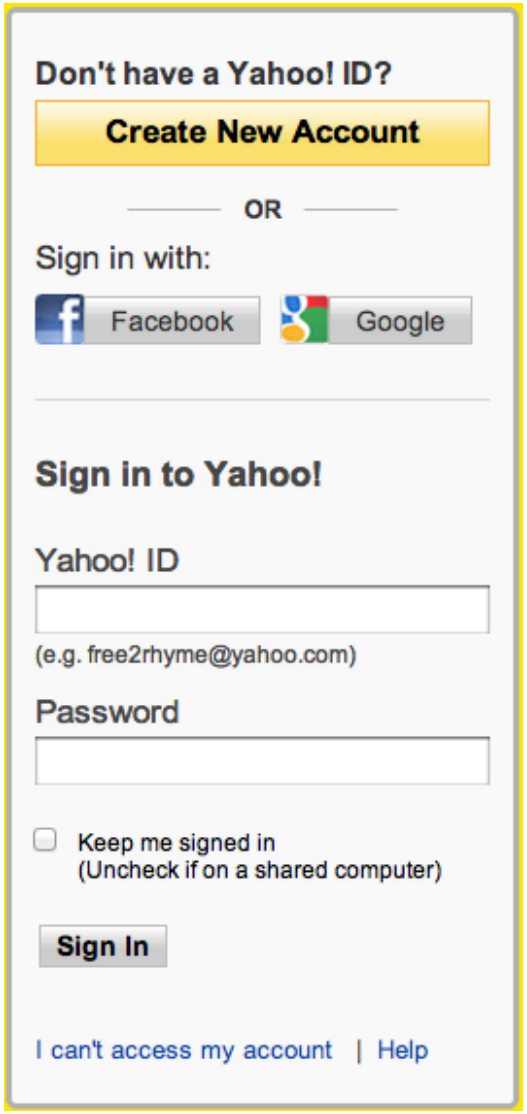
- If multiple cookies are applicable with the same name, **any can be picked**
- A page on dl.dropbox.com (user contents) can set cookies for .dropbox.com (service site)
- Attacker can set .dropbox.com cookies over HTTP

New Session Design

- Cookies are an **unfixable disaster for security**
- **Why not use the existing TLS session in the application?**
  - Strong integrity protection
  - Can be matched with same-origin policy in a webserver middleware

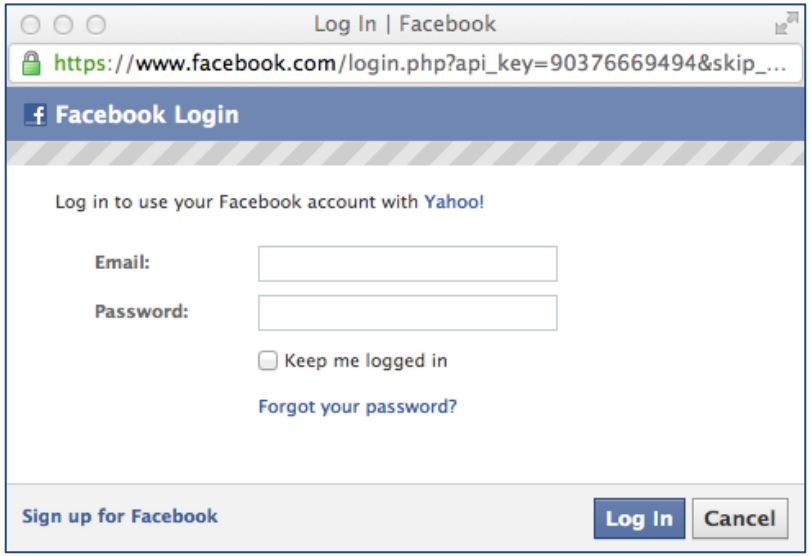
WEAKNESSES OF WEB PROTOCOLS AND THEIR IMPLEMENTATIONS

Social Sign-On with OAuth 2.0



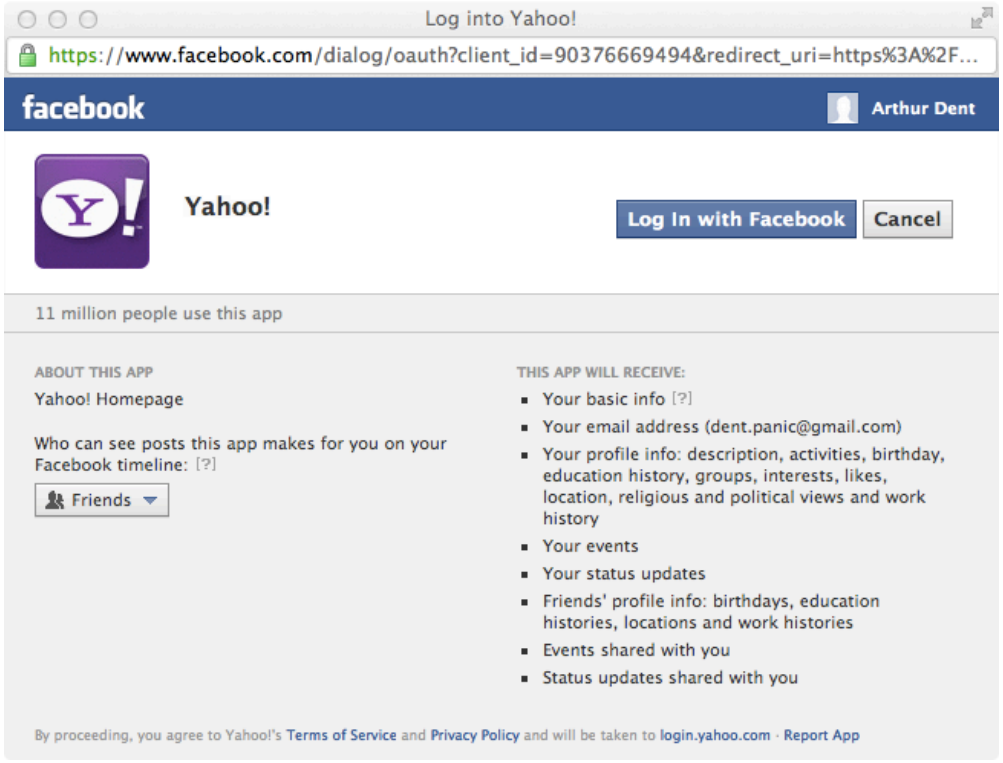
- Alice browses to https://login.yahoo.com
- Alice clicks on "Sign in with Facebook"

Social Sign-On with OAuth 2.0



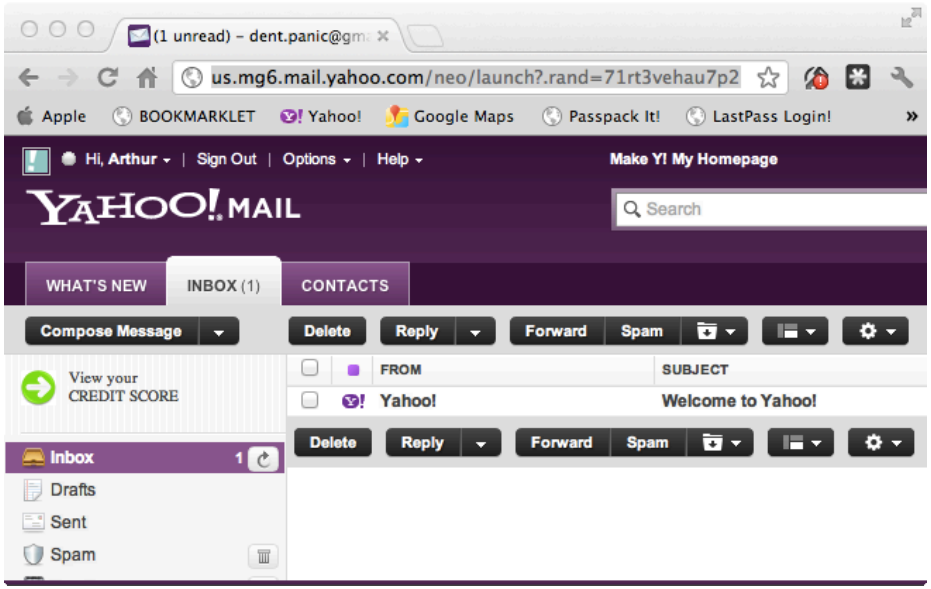
- Alice's browser is redirected to https://facebook.com/login.php
- Alice authenticates herself with a username and password
  - If she is already logged in, this step is skipped

Social Sign-On with OAuth 2.0



- Alice's browser redirected to https://facebook.com/oauth?app_id=(Yahoo)&perms=email,name,…&**redirect_uri=login.yahoo.com**
- Alice authorizes Yahoo to access her Facebook data
  - If she has previously authorized Yahoo, this step is skipped

Social Sign-On with OAuth 2.0



- Alice's browser redirected to https://**login.yahoo.com**?access_token=XXX
- Yahoo calls Facebook's REST API with the token XXX to read Alice's identity
  - Possessing the token authorizes Yahoo
- Yahoo logs Alice in who can now read her Yahoo mail

Stealing OAuth Access Tokens

1. Access token requests are not authenticated
2. Tokens are not bound to a particular client
3. Tokens are sent as part of an ordinary URI
4. Long-lived tokens are as good as passwords

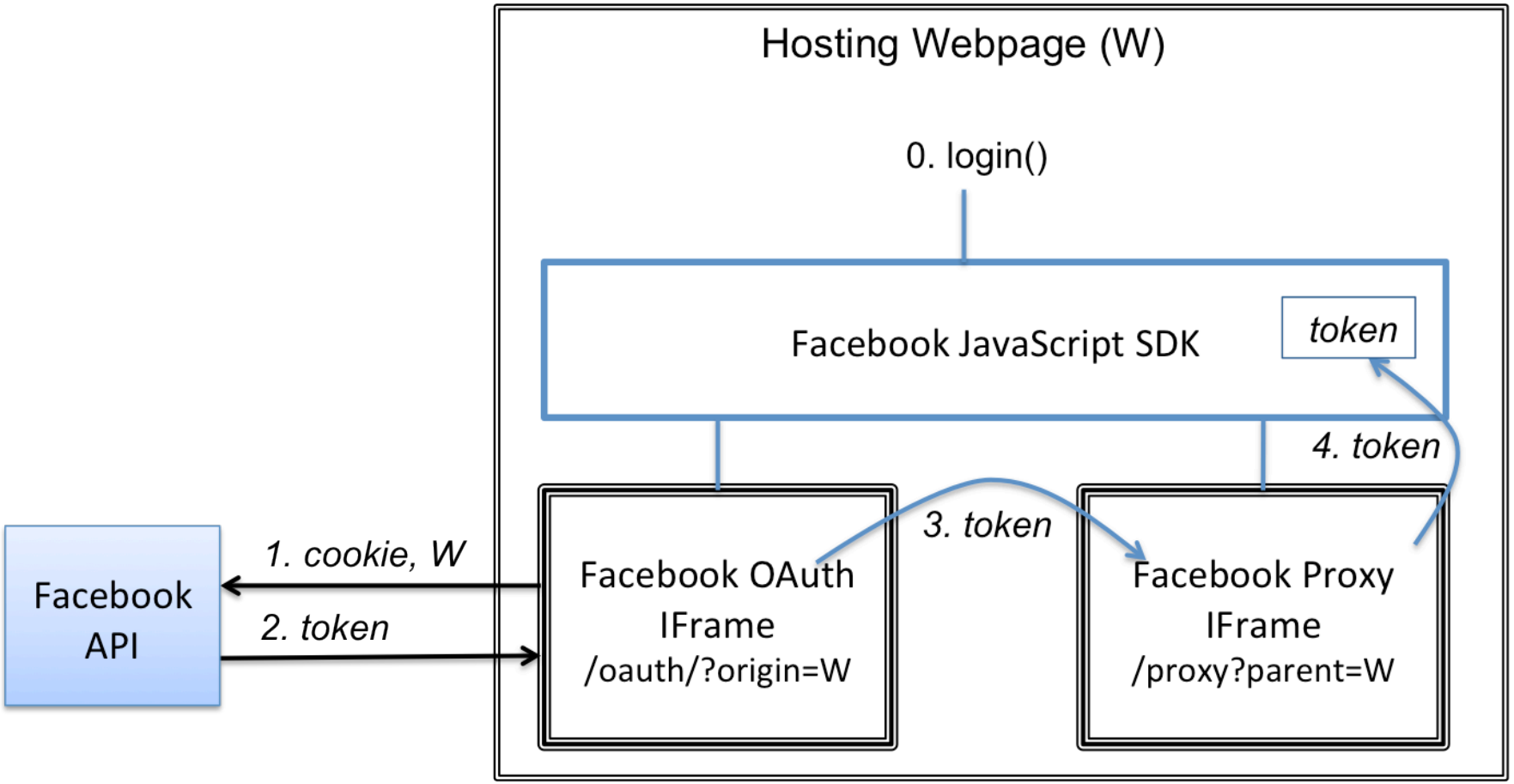
- We found **a dozen ways of stealing OAuth 2.0 access tokens** from popular websites

- Stealing a user's access token lets me
  - impersonate the user
  - steal user data from OAuth provider

Token Redirection Attack: Yahoo

- Suppose a malicious website redirects a user to
  - https://facebook.com/oauth?app_id=(Yahoo) &perms=email,name,… &**redirect_uri=search.yahoo.com/redirect/attacker.com**
- Facebook redirects the browser to
  - http://**search.yahoo.com/redirect/attacker.com/**?access_token=XXX
- Yahoo will then redirect the browser to
  - http://attacker.com/?access_token=XXX
- Attacker obtains the token XXX

Origin Spoofing: Facebook JS SDK

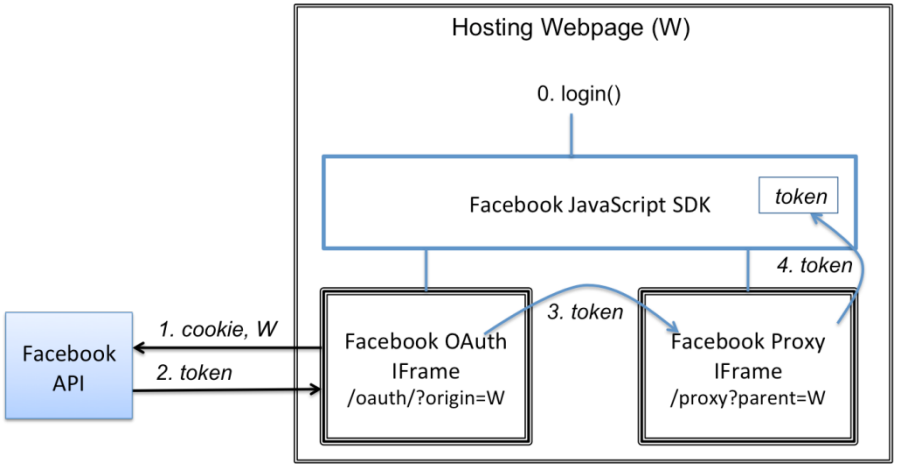


Origin Spoofing: Facebook JS SDK

- 4 instances of the Same Origin Policy:
  - *iframe*: W cannot access content of OAuth or Proxy
  - *redirection*: OAuth token redirection is invisible to W
  - *AJAX*: W cannot directly access Facebook API
  - *postMessage*: W cannot read token sent to Yahoo

Origin Spoofing: Facebook JS SDK

- A malicious website W can still break origin authentication
- OAuth iframe with origin=Yahoo, Proxy iframe with parent=W
- OAuth token XXX for Yahoo is passed it to Proxy, which sends it to W by postMessage
  - Bug #1: OAuth and Proxy do not compare **origin == parent**
  - Bug #2: Proxy does not **parse its parent URI correctly**
  - Bug #3: OAuth does not **parse multiple params in origin correctly**



Many Attacks on Social Sign-On

Website	Role(s)	Preexisting Vulnerabilities			New Social CSRF Attacks			New Token Redirection Attacks		
		Login CSRF	Form CSRF	Token Redirector	Login CSRF	Automatic Login	Sharing CSRF	Resource Theft	Unauthorized Login	Cross Social-Network Request Forgery
Twitter	AS, RS	Yes								Yes
Facebook	AS, RS						Yes	Yes		Yes
Yahoo	Client			Yes				Yes	Yes	
WordPress	Client	Yes			Yes	Yes		Yes	Yes	
CitySearch	Client	Yes		Yes	Yes	Yes	Yes			
IndiaTimes	Client	Yes		Yes	Yes	Yes	Yes			
Bitly	Client				Yes	Yes		Yes		
IMDB	Client	Yes			Yes	Yes				
Posterous	Client				Yes			Yes		
Shopbot	Client	Yes			Yes	Yes				
JanRain	Client lib									Yes
GigYa	Client lib									Yes

- *Discovering Concrete Attacks on Website Authorization by Formal Analysis*, C. Bansal, K. Bhargavan, S. Maffeis. CSF 2012.
- *Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services*, R. Wang, S. Chen, and X. Wang, *IEEE S&P 2012*

Parsing Issues

```
strict: /^(?:([^:\/?#]+):)?(?:\/\/((?:(([^:@]*)
    (?::([^:@]*))?)?@)?([^:\/?#]*)(?::(\d*))?)
    )?((((?:[^?#\/]*\/)*)([^?#]*))(?:\?([^#]*)
    )?(?:#(.*))?)/
```

- Most popular JavaScript URL parsing library (parseUri)
- Suppose href = https://attacker.com/#@google.com
  - parseUrl returns google.com
  - it should return attacker.com

- Phishing attack: attacker.com can pretend to be Google
- Solutions:
  - Use the browser's window.location.host whenever possible
  - Implement the URI grammar completely (multiple regular expressions)

Survey of attacks against Security-Sensitive Web Applications

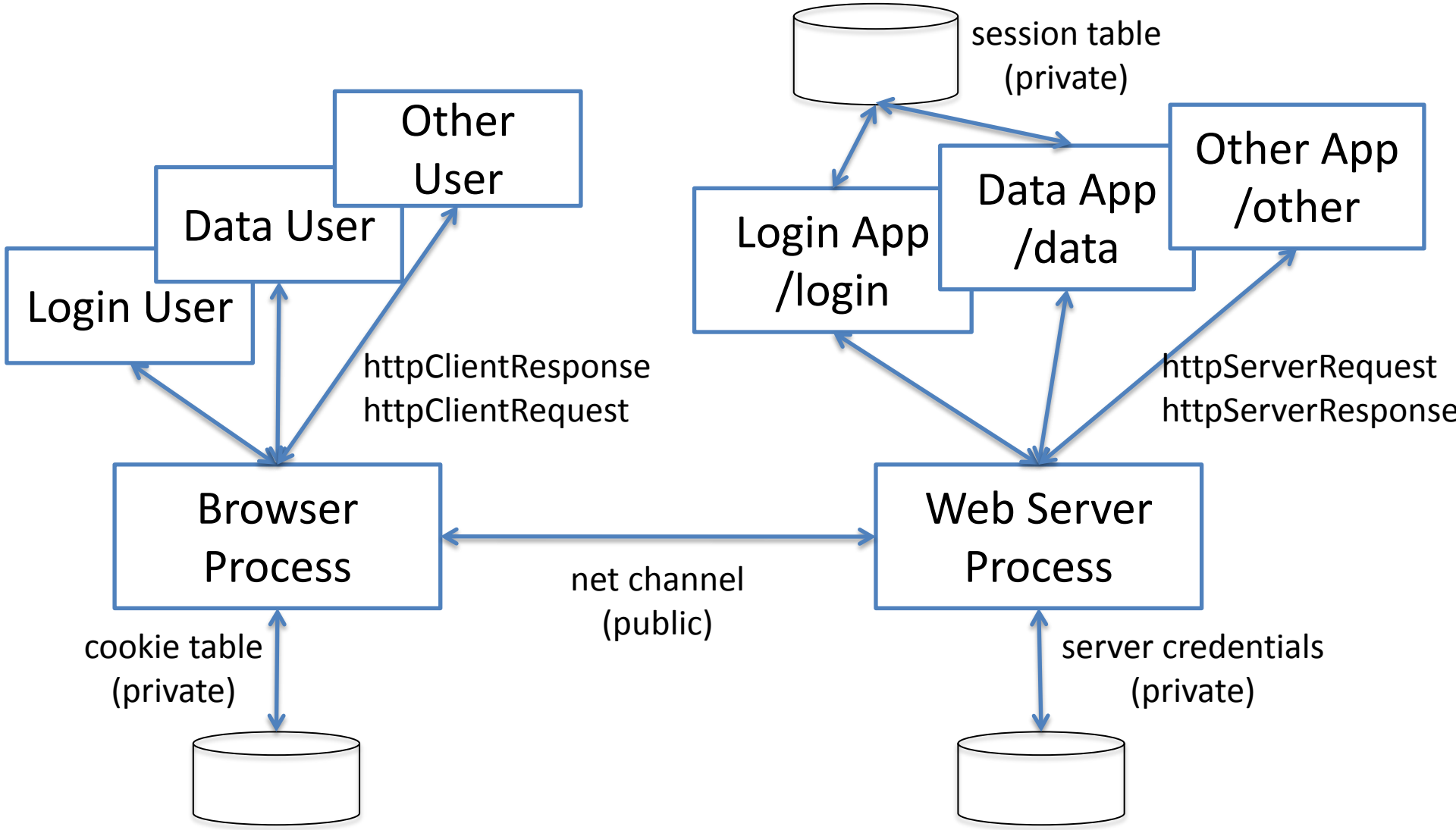
Product	Category	Protection Mechanism	Attack Vectors Found	Secrets Stolen
Facebook	Single Sign-On Provider	Frames	Origin Spoofing, URL Parsing Confusion	Login Credential, API Access Token
Helios, Yahoo, Bitly WordPress, Dropbox	Single Sign-On Clients	OAuth Login	HTTP Redirector, Hosted Pages	Login Credential, API Access Token
Firefox	Web Browser	Same-Origin Policy	Malicious JavaScript, CSP Reports	Login Credential, API Access Token
1Password, RoboForm	Password Manager	Browser Extension	URL Parsing Confusion, Metadata Tampering	Password
LastPass, PassPack Verisign, SuperGenPass	Password Manager	Bookmarklet, Frames, JavaScript Crypto	Malicious JavaScript URL Parsing Confusion	Bookmarklet Secret, Encryption Key
SpiderOak	Encrypted Cloud Storage	Server-side Crypto	CSRF	Files, Encryption Key
Wuala	Encrypted Cloud Storage	Java Applet, Crypto	Client-side Exposure	Files, Encryption Key
Mega	Encrypted Cloud Storage	JavaScript Crypto	XSS	Encryption Key
ConfiChair, Helios	Crypto Web Applications	Java Applet, Crypto	XSS	Password, Encryption Key

VERIFICATION OF WEB APPLICATIONS

WebSpi: a Formal Model of the Web

- A web security library for ProVerif
  - browsers, cookies, HTTP(s) sessions
  - web forms, HTTP redirection, JavaScript
  - TLS sessions, encrypted databases, user credentials
  - malicious websites, CSRF attacks, open redirectors, malicious users, compromised servers
- Papers:
  - *Discovering Concrete Attacks on Website Authorization by Formal Analysis*, C. Bansal, K. Bhargavan, S. Maffeis. CSF 2012
  - A. Delignat-Lavaud: *Keys to the cloud: Formal analysis and concrete attacks on encrypted web storage*. C. Bansal, K. Bhargavan, A. Delignat-Lavaud, S. Maffeis, POST 2013

Structure of WebSpi



Example: Login Page Browser Process

```
let LoginUserAgent() =
 in(httpClientResponse,(b:Browser,sid:Cookie,sp:Principal,
                        u:Uri,d:bitstring));
 let up = principal(b) in
 let uri(proto,h,app) = ep(u) in
 if proto = https() then
 if loginForm = formTag(d) then
 get credentials(=h,=app,=p,uid,pwd) in
 if Assume(Login(up,b,sid,ep(u),uid)) then
 out(httpClientRequest,
 (b,sid,formAction(d),u,httpPost(loginFormReply(d,uid,pwd))));
 in(httpClientResponse,(=b,=sid,=sp,formAction(d),loginSuccess()));
 event Expect(ValidSession(up,b,sid,ep(u)))
```

Login form loaded

Check HTTPS is used

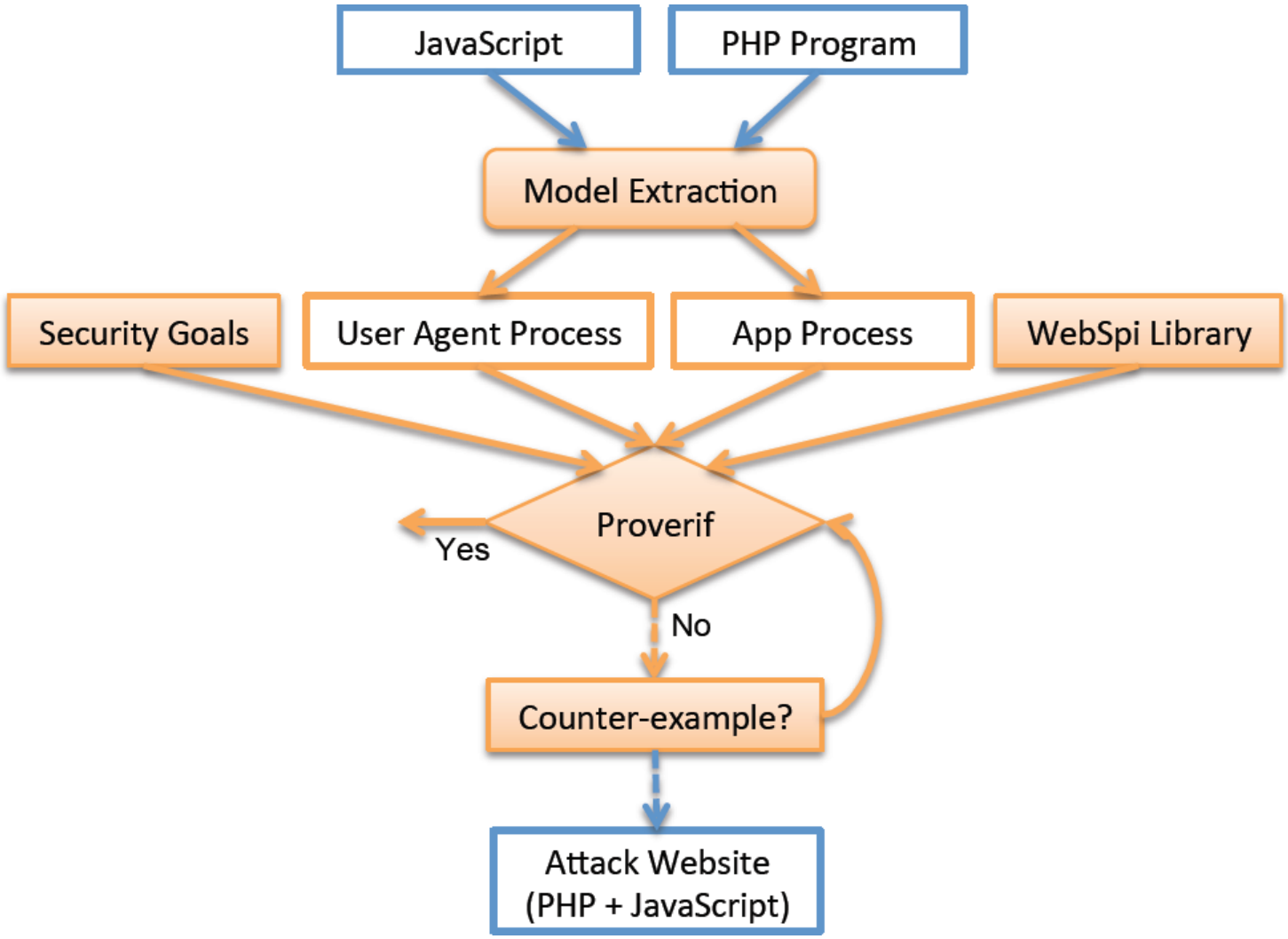
Retrieve user id, password

Declare login intention

Send form

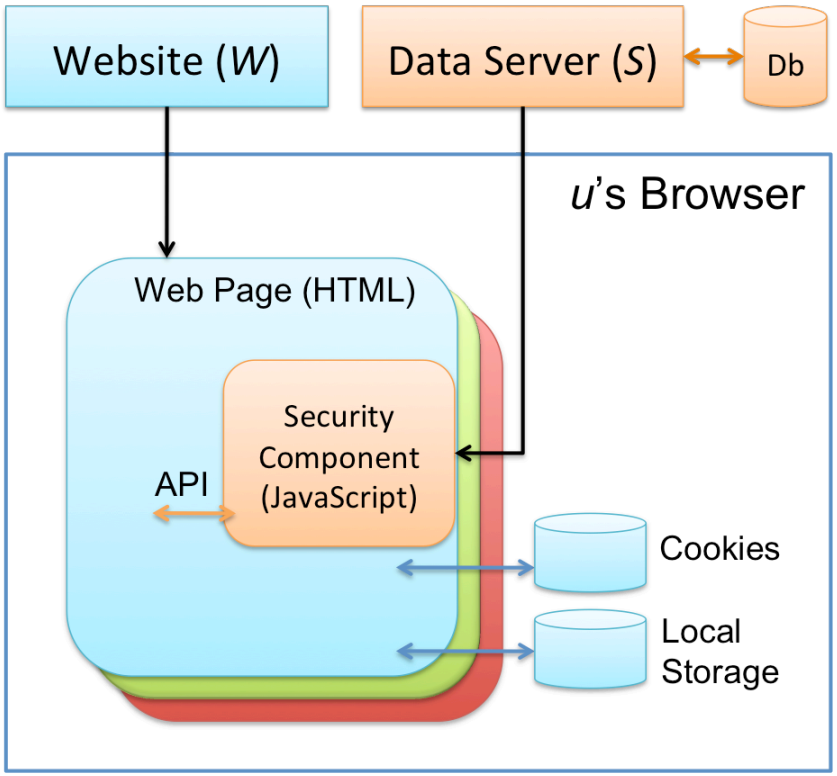
Expect valid session

Our Verification Flow



DJS: Defensive Web Components

- How do we write security-sensitive JavaScript components that may be safely executed within partially-trusted websites?



Threats:
- Malicious host server
- Buggy or malicious scripts
- XSS attacks

Component Goals:
- Its functionality cannot be tampered with
- Its secrets cannot be stolen

DJS: Robust Component Security

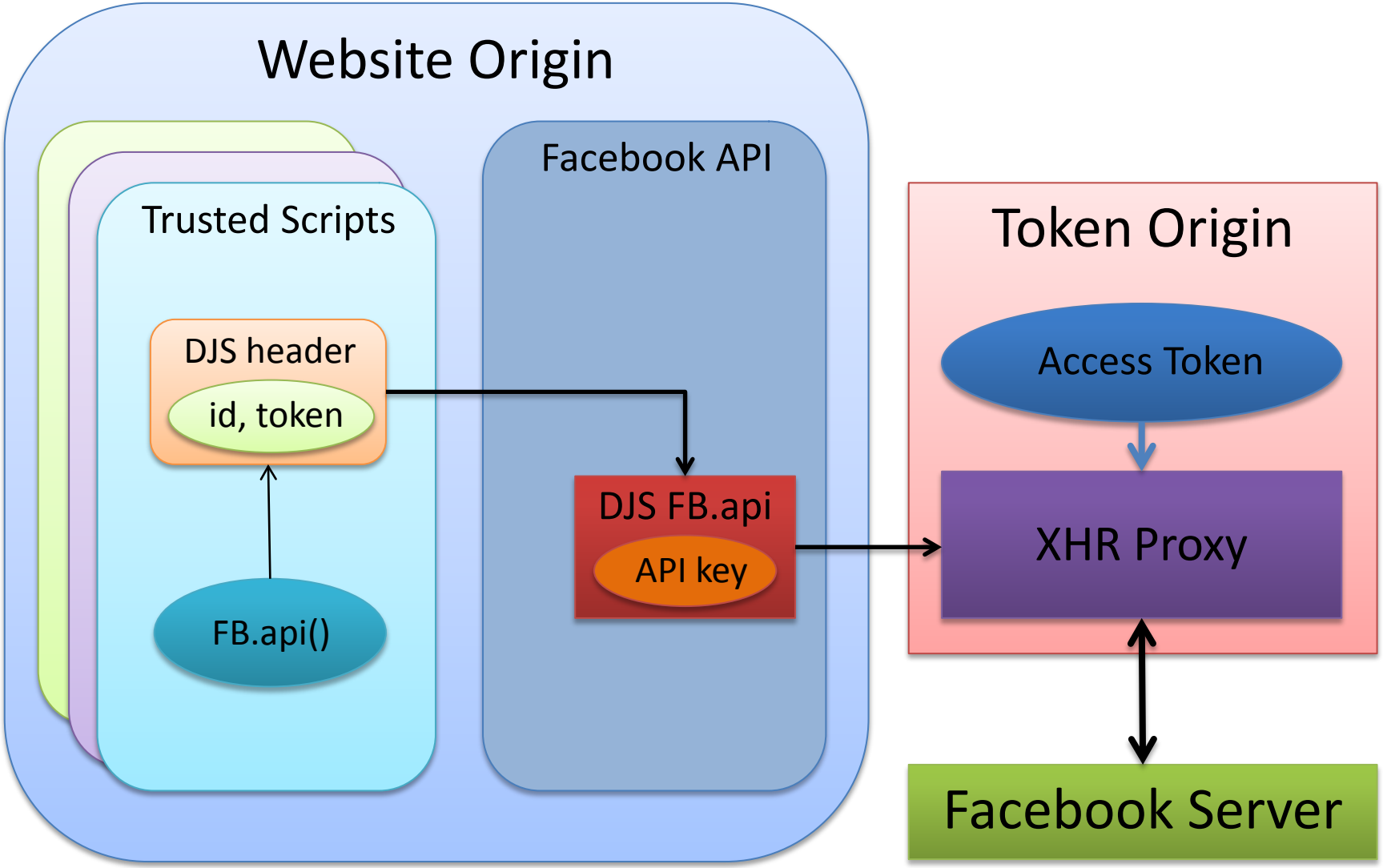
- **Component security is *fragile* against same-origin attackers**
  - every buggy script presents a potential attack
  - every XSS attack is fatal and leaks all secrets
- **Getting component security right against cross-origin attackers is *hard***, even with strong isolation mechanisms
  - flaws in authorization logic
  - incorrect use of crypto
  - incorrect assumptions about the same origin policy
- Need for a component programming framework that affords **stronger isolation guarantees** and supports **automated formal analysis**

DJS: Defensive JavaScript subset

- A **sound static type system** that identifies a formal subset of JavaScript and enforces our defensive idioms
  - fully self-contained, no external references
  - all code wrapped in a closure and exposed through a typed first-order API served from a trusted origin

- Type safety guarantees:
  - *Independence:* The input-output functionality of well-typed programs is the same in all JavaScript contexts
  - *Encapsulation:* The only way a context can discover the content of a typed program is by calling its API

DJS Example: Facebook Login



Conclusions

- Designing secure Web applications is *hard*
- Implementing them correctly is *even harder*
- **We aim to make automatic model extraction and verification tools easy enough to be used by average web developers**
  - Help fix browsers, standard protocols and large websites along the way

QUESTIONS