

# Biting the hand that serves you: A closer look at client-side Flash proxies for cross-domain requests

Martin Johns and Sebastian Lekies  
DIMVA 2011 / 8<sup>th</sup> July 2011

**SAP**

 **WebSand**

# Agenda

---

1. The how and why of client-site cross-domain HTTP requests
2. Client-side Flash proxies & identified security problems
3. Countermeasures



# Technical Background



# Client-side HTTP requests

---

## Client-side HTTP requests

- JavaScript initiated HTTP requests
- Originally introduced by Microsoft via the XMLHttpRequest-object

## Same-origin policy

- Common security policy that applies to all active browser-based technologies (JS, Flash, Java, Silverlight)
  - (slight variations might occur)
- General principle
  - Same origin == full trust, full access
  - Different origin == no trust, no access
- This rule applies to explicit HTTP requests that are created by script code

## → Only same-origin requests are permitted

- But, wouldn't it be great to do this cross-domain?

# Use-cases for client-side cross-domain HTTP requests

---

**The need for this mechanism is not immediately obvious**

Alternative: Server-side proxies

- Capable of cross-domain data retrieval
- Compliant with the same-origin policy
  - Requests are routed through the script's original host

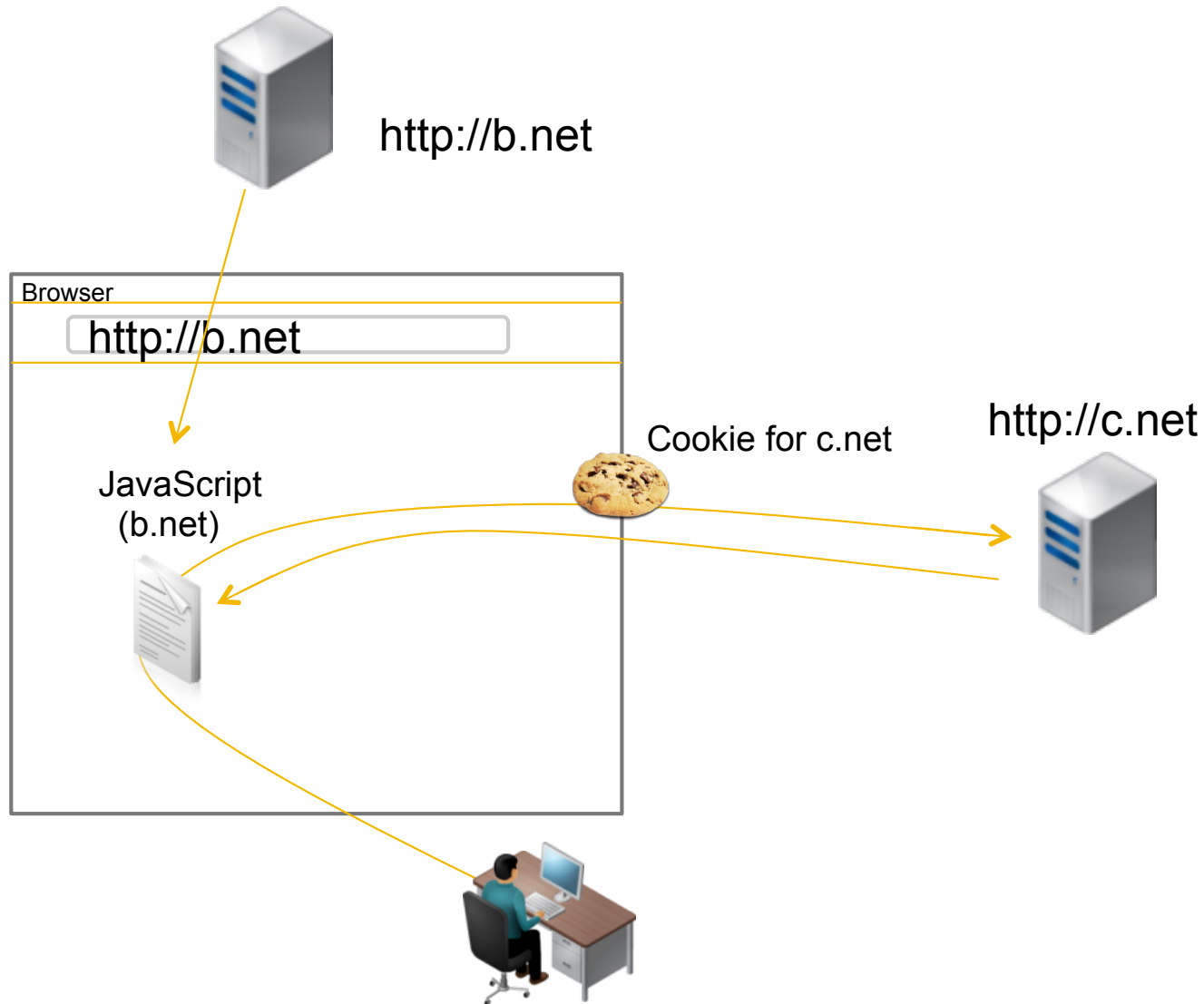
**Advantage of client-side cross-domain requests:**

The HTTP requests are created in the user's current authentication context

- Cookies
- Creation within the current intranet

This allows application scenarios which are impossible with server-side proxies

# Use-case for client-side cross-domain HTTP requests



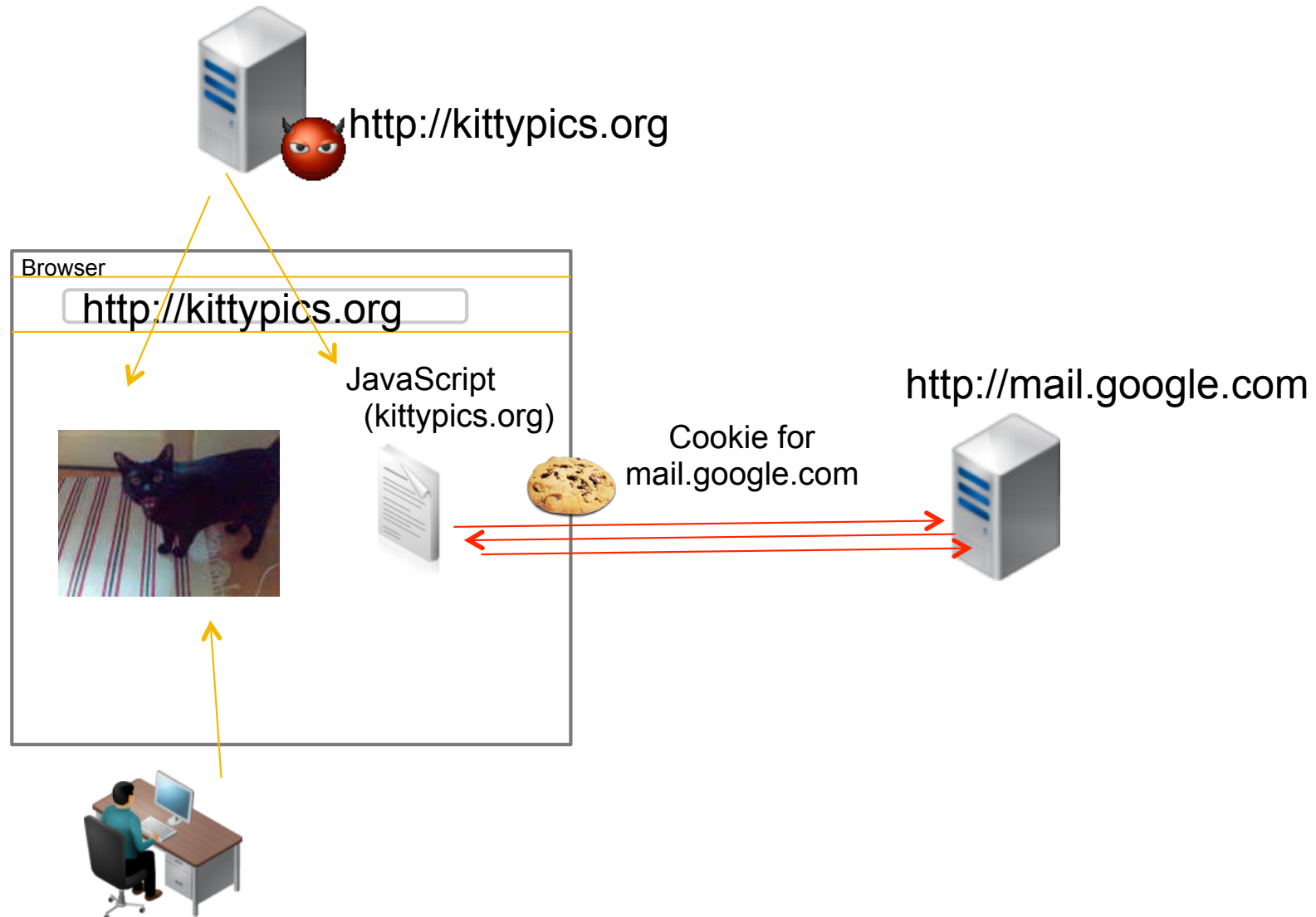
# Security implications

---

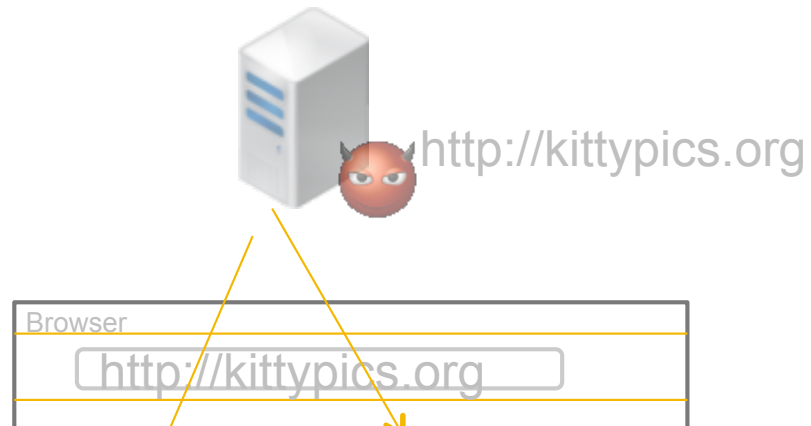
## **Scenario: Let's pretend client-side cross-domain requests are permitted**

1. An adversary controlled client-side script is permitted to create cross-domain HTTP requests and receive the corresponding HTTP responses
2. These requests are created in the user's current authentication context
  - I.e., the requests carry the user's session cookies

# Security implications



# Security implications



## 1. Leakage of sensitive information

- The adversary can request sensitive web resources

## 2. Circumvention of CSRF protection

- Token-based CSRF protection relies on the fact, that the adversary cannot read cross-domain data

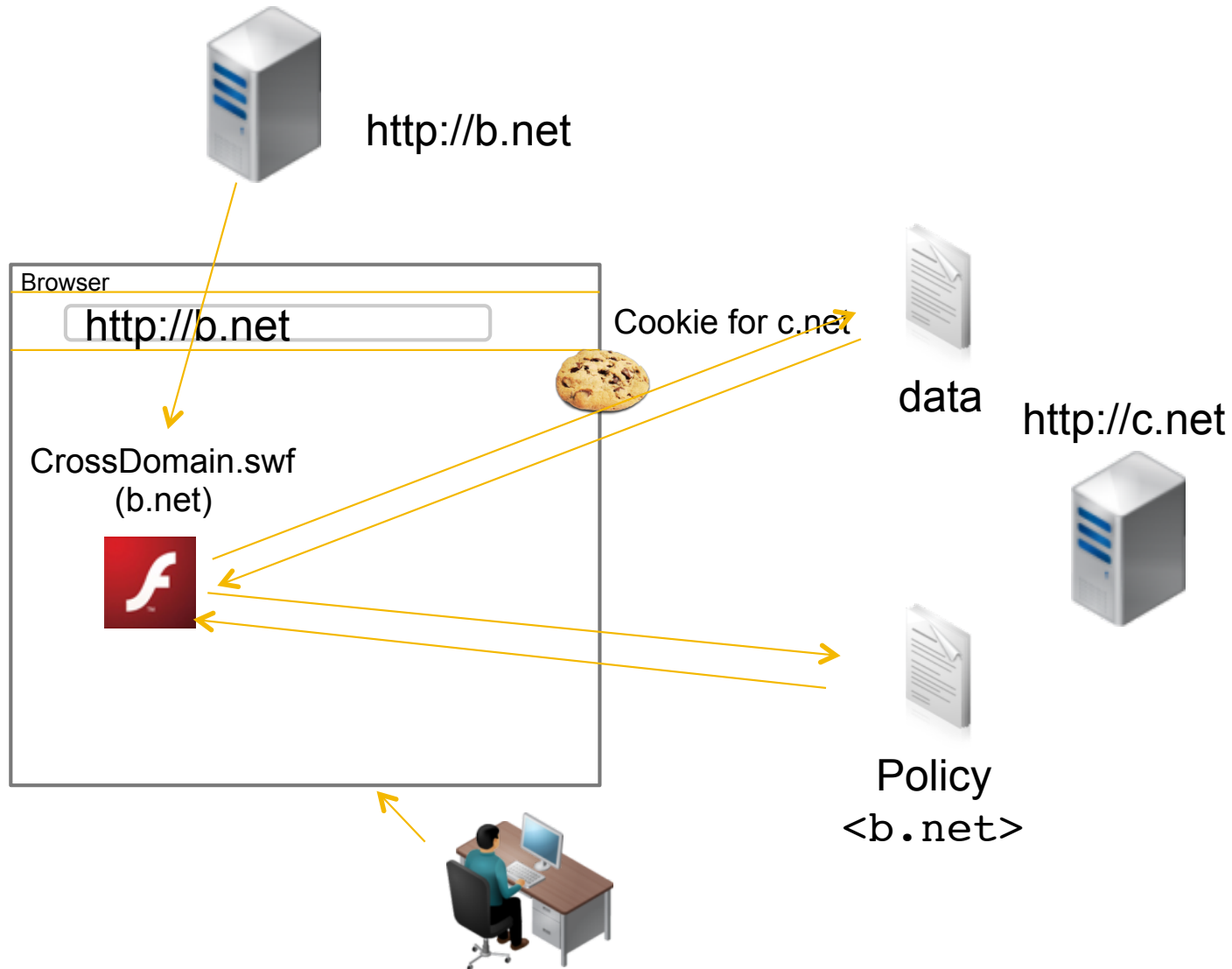
## 1. + 2. = XSS-based Session hijacking

- Chaining requests & reading responses -> capabilities equal to XSS session hijacking

http://mail.google.com



# Opt-in model for client-side cross-domain requests



# Policy-files for opt-in

---

**A list of all domains that are trusted to send client-side cross-domain requests on behalf of the user**

## **Example: Flash**

- crossdomain.xml
- List of domain names
- Hosted in the Web root

```
<cross-domain-policy>  
  <allow-access-from domain="google.com" />  
  <allow-access-from domain="facebook.com" />  
</cross-domain-policy>
```

**Attention:** The domains listed in this policy now have potential harmful capabilities!

# Is this really used in practice?

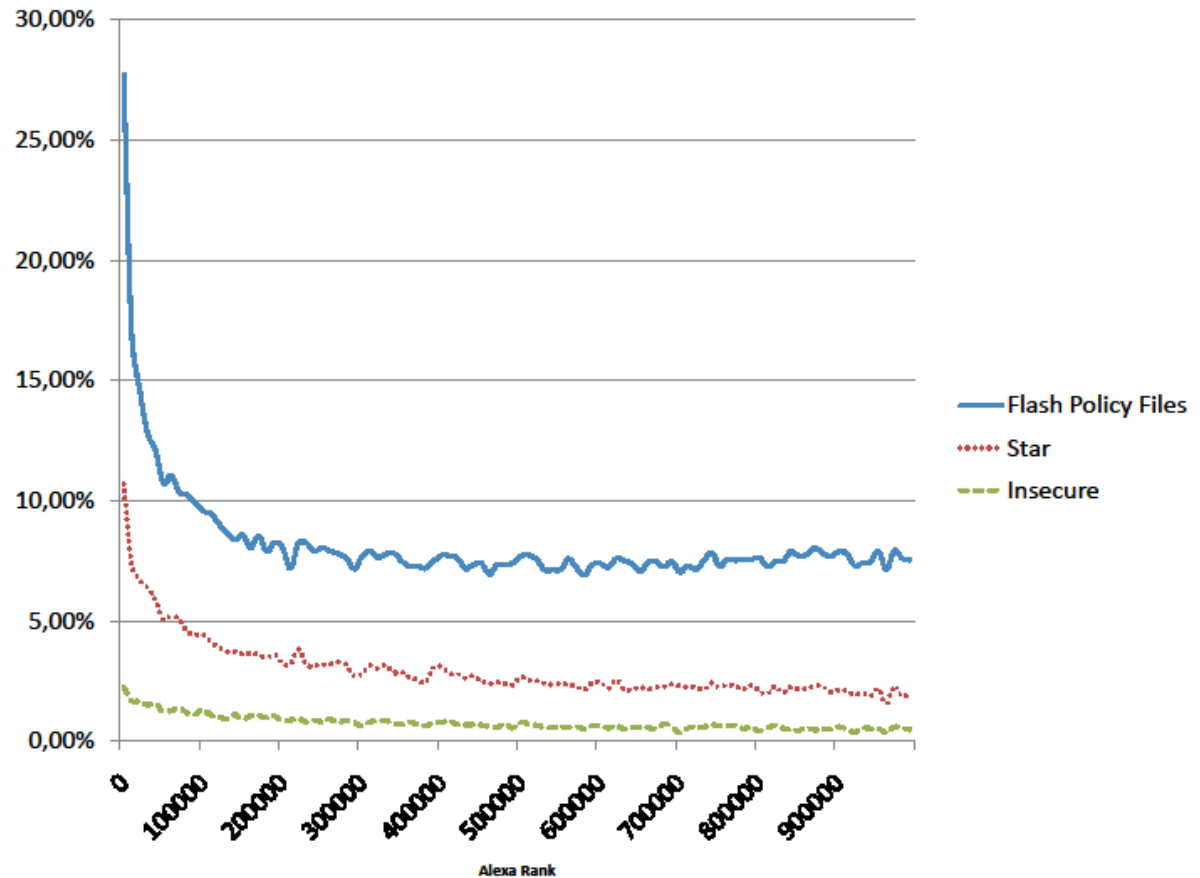
## Survey: Penetration / Security

### Alexa top 1.000.000

- 7,5% provide policy files

### Wildcard-policy

- 31.011 files (37,7% of all crossdomain.xml) resulting in 2,8% potentially insecure sites
- When checking for authentication
  - 15.060 sites (1,3% of all analyzed sites)



[Lekies & Johns, W2SP'11]

Collected crossdomain.xml files

# The move towards JS only applications

---

## **Observable trend towards reduced reliance on browser plugins**

- Motivated by, e.g., mobile browsers that do not support plugins (iOS, Windows Phone 7)
- Enabled through current rapid development of native browser capabilities (“HTML5”)

## **Cross-origin resource sharing (CORS)**

- Native JavaScript capabilities for creating client-side cross-domain requests
- Server-opt-in via HTTP headers

## **Today: Transitional phase**

Developers have to maintain two separate versions of their client-side functionality

- Flash-based version for legacy browsers
- CORS version for browsers that do not support Flash

Length of this transitional phase is unknown...



# Client-side Flash proxies



# Client-side Flash proxies for cross-domain HTTP requests

---

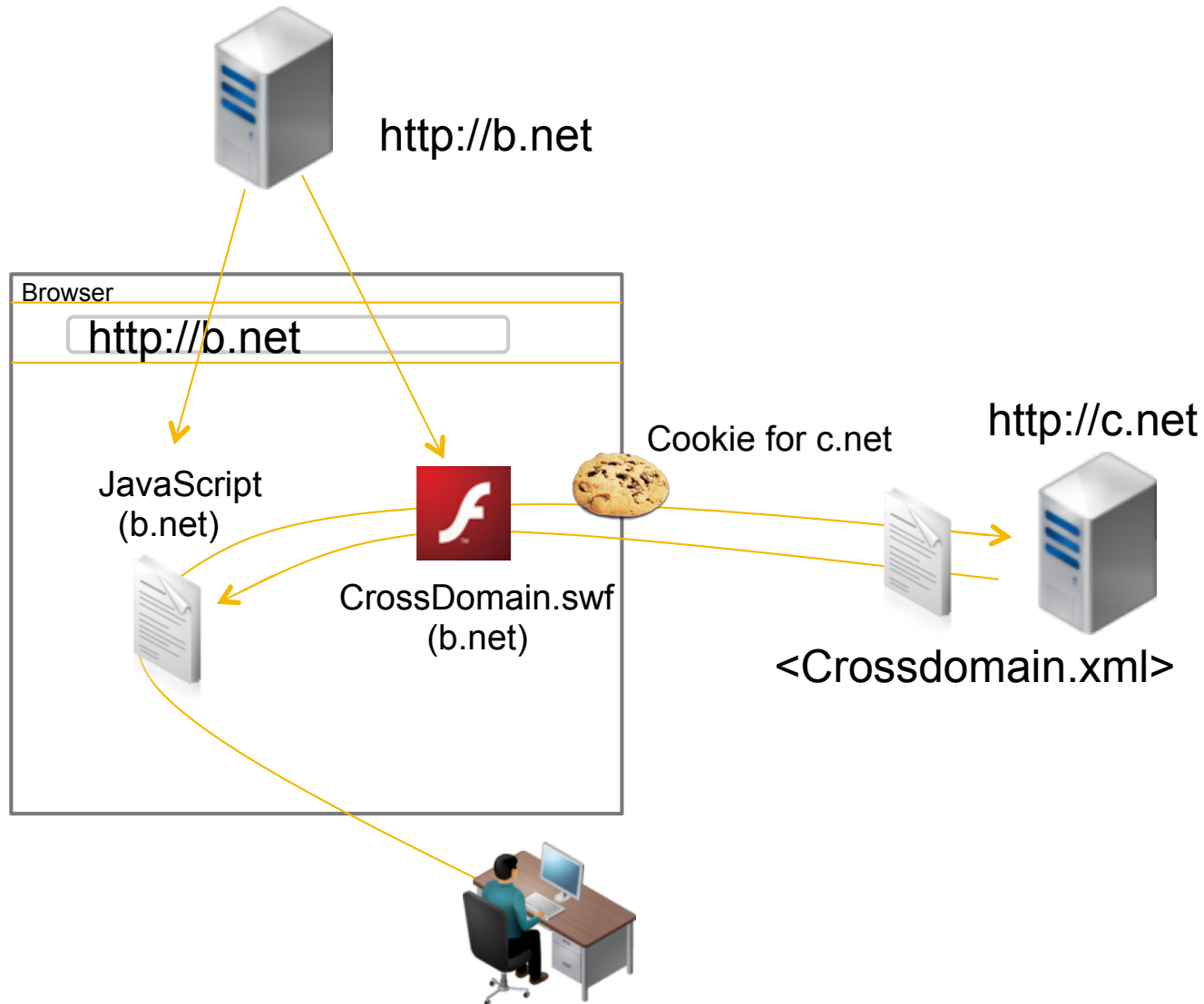
## Potential solution for the transitional phase

- Create application that fully relies on JS
- Utilize single purpose Flash applet only for the cross-domain communication

## Client-side Flash proxies for cross-domain requests

- Offer this functionality
- Offer small JS-libs that provide the interface to the hosting JS code
- Several ready-made implementations available
  - We identified five interesting candidates
  - Most work by replacing the native XMLHttpRequest object and/or integrate into popular JS frameworks

# Client-side Flash proxies for cross-domain HTTP requests



# A subtle difference in the SOP

---

## HTML allows the inclusion of cross-domain elements

- `<img>`, `<style>`, `<script>`, (`<audio>`, `<video>`)
- The included cross-domain content inherits the origin of the including site
- This means, cross-domain JavaScript is executed as if it belongs to the domain of the including site
- However, this does not apply to the `<object>`, `<embed>` and `<applet>` tags
  - Even though they (mostly) behave like standard HTML elements

## Inclusion of cross-domain plugin content

- Unlike native HTML/JS content, plugin content retains the origin from where it was retrieved

## In consequence:

- Outgoing cross-domain requests are permitted based on the site from which the Flash was retrieved and **not** based on the site which initiates the requests

# Attack vectors

---

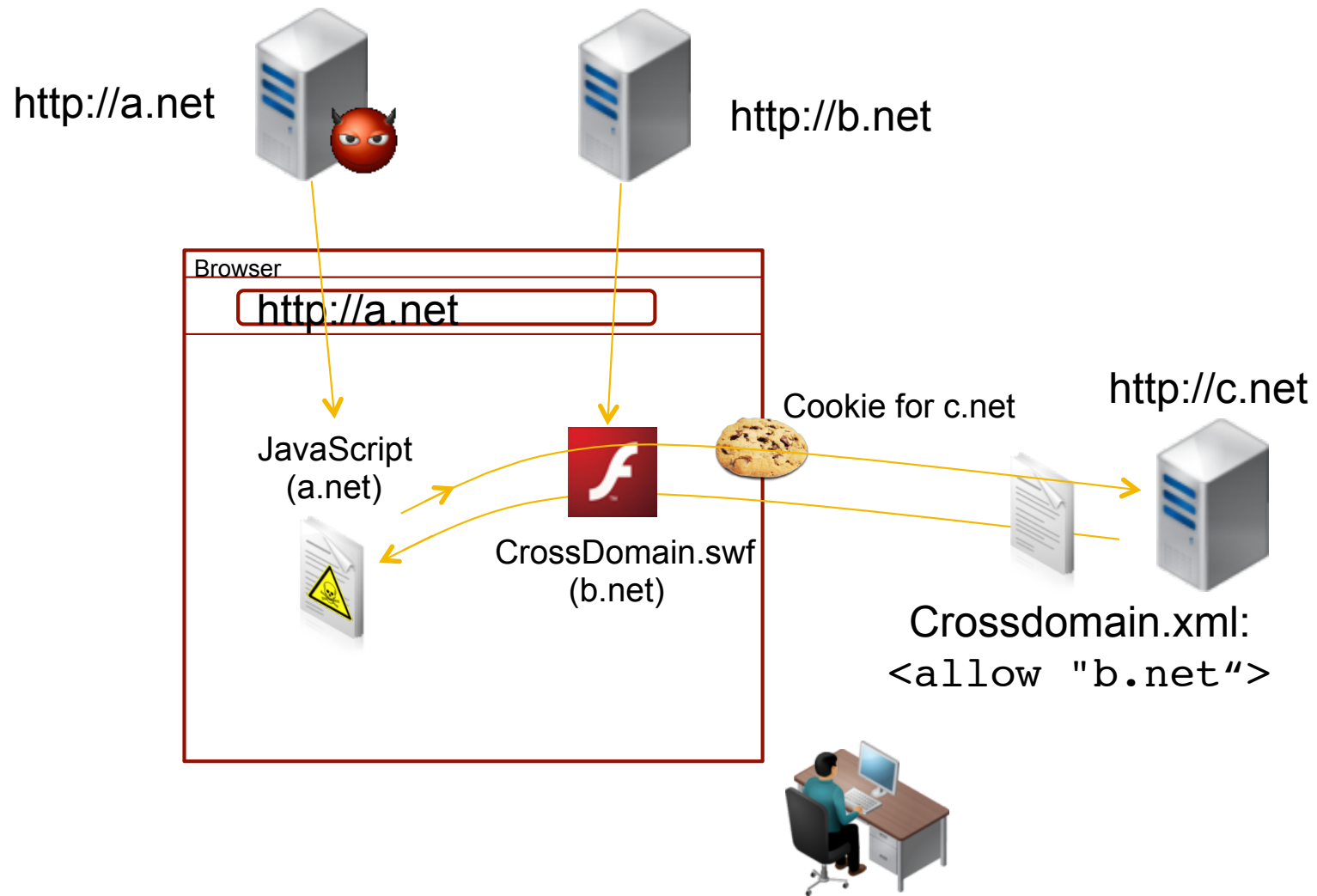
**Observation: Cross-domain inclusion of client-site Flash proxies provides the **including** site with elevated capabilities**

- The including site is allowed to initiate HTTP communication with all sites that *trust* the proxies original host

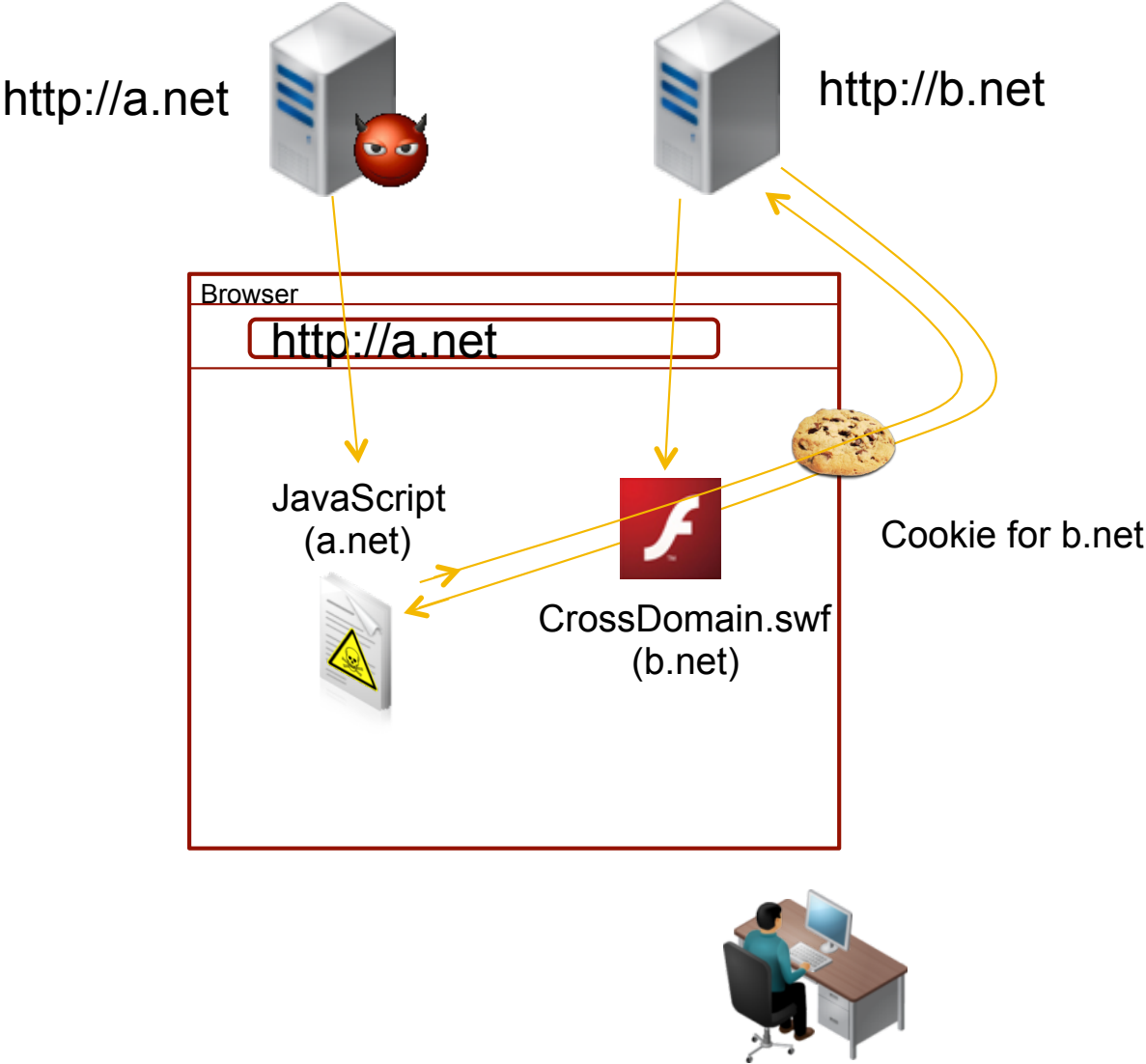
**We could identify two potential attack vectors**

1. Transitivity of trust
2. Return to sender

# Attack vectors: Transitivity of trust



# Attack vectors: Return to sender



# Survey of existing implementations

## We could identify five candidates

- Out of these 5 implementations, 3 were vulnerable to the attacks
- The remaining 2 proxies restricted their external interface in a fashion that prohibited cross-domain access
- This can be done via setting the matching `allowdomain()` directive in the proxy's source code

Name	Year	Source	JS-lib	Vulnerable
flXHR [30]	2010	No	1,2,3,4	Yes, despite of countermeasures
SWFHttpRequest [34]	2007	Yes	1,2,3	No
FlashXMLHttpRequest [8]	2007	No	3	No
CrossXHR [24]	2010	Yes	1,2	Yes
F4A	unknown	No	-	Yes

**Legend:**

**Year:** Year of last recorded activity

**Source:** Is source code available?

**JS-lib:** Available plug-in for popular JavaScript frameworks: 1) JQuery, 2) Prototype, 3) Dojo, 4) Mootools

# Analysis

---

## Why didn't the vulnerable implementations protect themselves?

- Via using the `allowdomain()` directive to restrict the set of permitted domains that are allowed to interact with the Flash

### Variant 1:

- **Only the Flash's own domain can interact**
- Cannot be used easily in setups that incorporate subdomains
  - Already serving the same site with and without the “www” prefix might cause problems

### Variant 2:

- **Providing a full list of permitted domains**
- Hardcode in the proxy's source code
- Requires knowledge of the application's details on compile time

**Both variants are unsuitable for general purpose proxy implementations**



# Countermeasures



# Mission statement

---

## Two separate tasks

1. Secure way to create a general purpose Flash proxy implementation
2. Reliably allow verifying the origin of the enclosing Web page (in the paper)

# 1. Securely create a general purpose Flash proxy implementation

---

Two variants:

## A. CSRF protecting the swf-delivery

Allows the usage of ready-made proxy implementations

```
<?php
if ($_GET["anti_csrf_nonce"] == $_SESSION["nonce"]){
    $swf = [...] // binary data of the .swf file
    header("Content-type: application/x-shockwave-flash");
    echo $swf;
} else {
    ... // Generate 500 internal server error
}
?>
```

## B. Dynamic parameterizing the `allowdomain()` directive

- The parameter of `allowdomain()` can be dynamically set on run time
- This allows to load a separate policy file from the Flash's original host which contains all hosts which are allowed to interact with the proxy



# Conclusion



# Conclusion

---

## Outlook

- Transitional phase towards JS-only applications has just started
- Empirical evidence suggests that client-side cross-domain requests are used by a significant number of sites
- → We probably will see more Flash fallback solutions in the future

**TL; DR: Be careful when using general purpose Flash proxies**



# Thank You!

Contact information:

Martin Johns  
SAP Research  
[martin.johns@sap.com](mailto:martin.johns@sap.com)