Eradicating DNS Rebinding with the Extended Same-Origin Policy

Martin Johns, Sebastian Lekies and **Ben Stock**

USENIX Security
August 16th, 2013

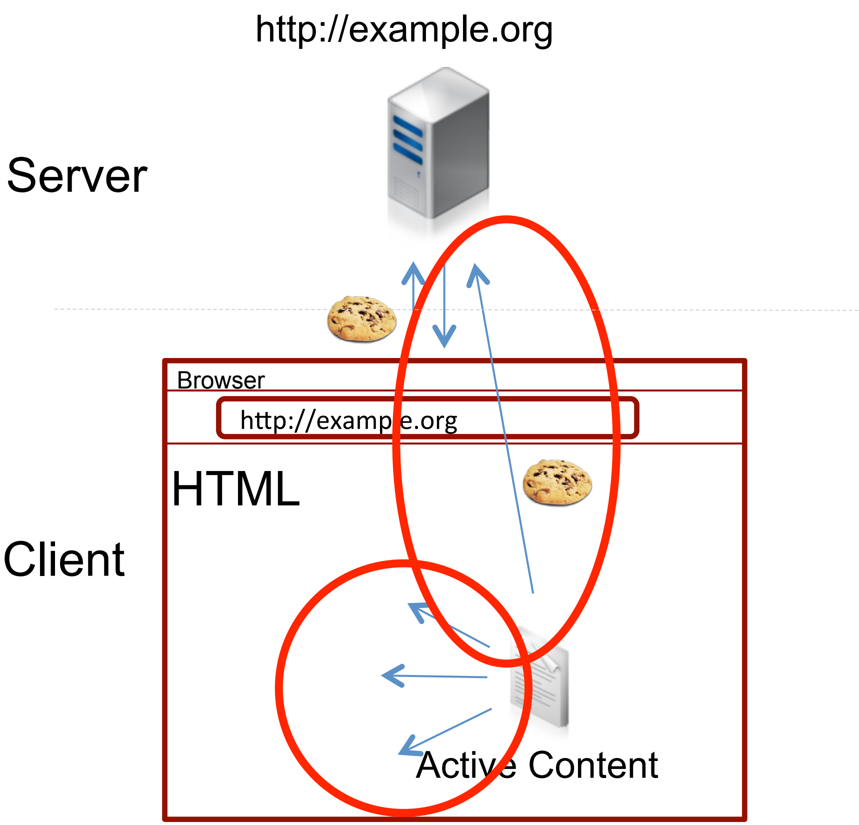
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Agenda

- DNS Rebinding
  - The basic attack
  - History repeating
- HTML5 Offline Application Cache Attack
- Extending the Same-Origin Policy
  - The three principals of Web Interaction
  - Extending the SOP with server-provided information
- Conclusion & Future Work

Technical Background

Web Application 101



- Active Content enables Web Apps to
  - interact with the Document (via the DOM)
  - interact with the Server (via XMLHttpRequests, Iframes, etc)
- ... in the name of the user
  - security sensitive
  - sensitive data and active content may originate from different sources
- Access is governed by the **Same-Origin Policy**

The Same-Origin Policy

> The Same-Origin Policy **restricts access** of active content to objects that share the same origin. The origin is, hereby, defined by the **protocol**, the **domain** and the **port** used to retrieve the object. *

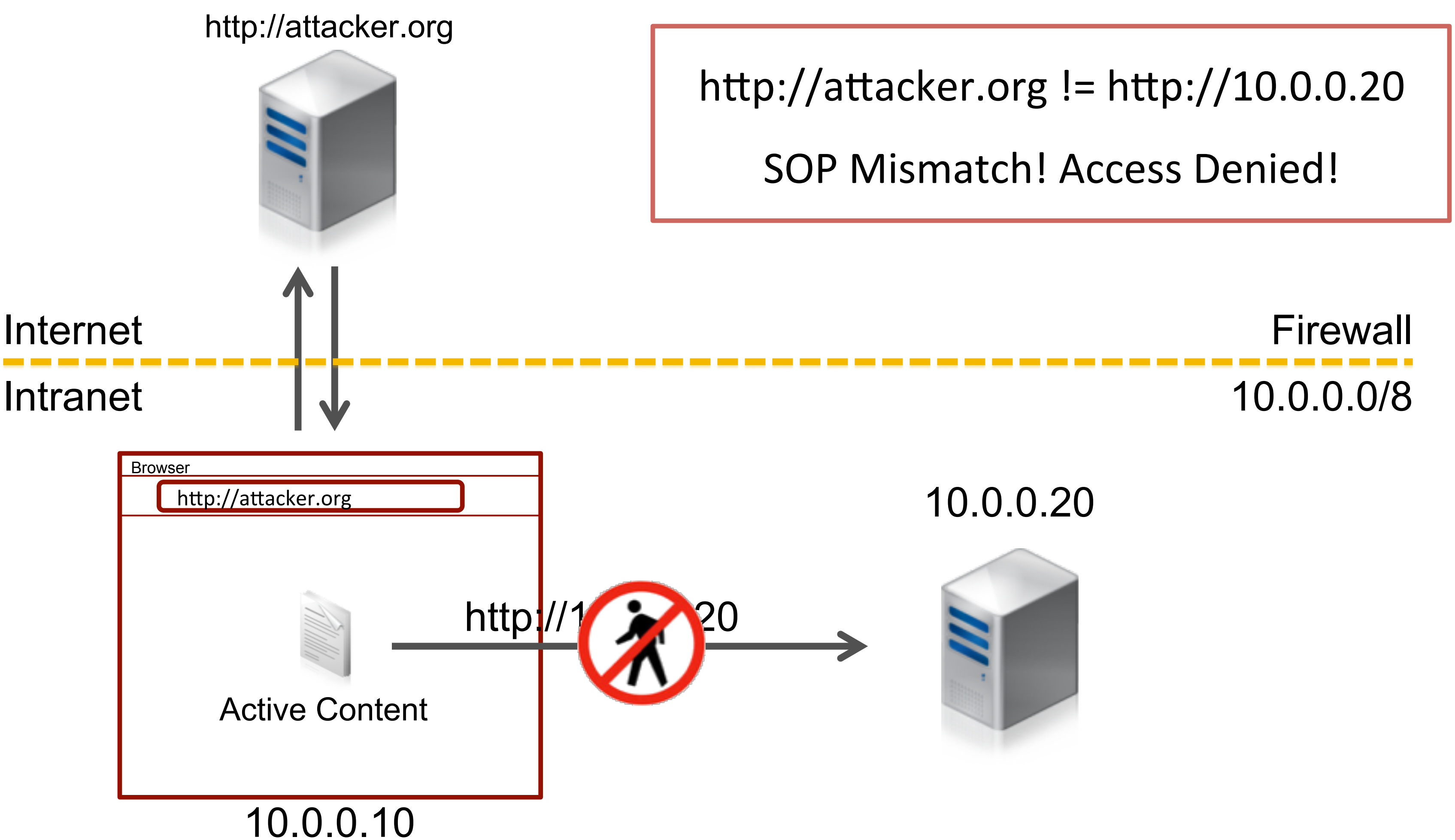
* Paraphrasing RFC 6454

http://example.org:80/some/webpage.html

protocol domain port

Target host	Access	Reason
http://example.org	Yes	---
https://example.org	No	Protocol mismatch
http://example.org**:8080**	No	Port mismatch
http://**facebook.com**	No	Domain mismatch

Protecting the Intranet

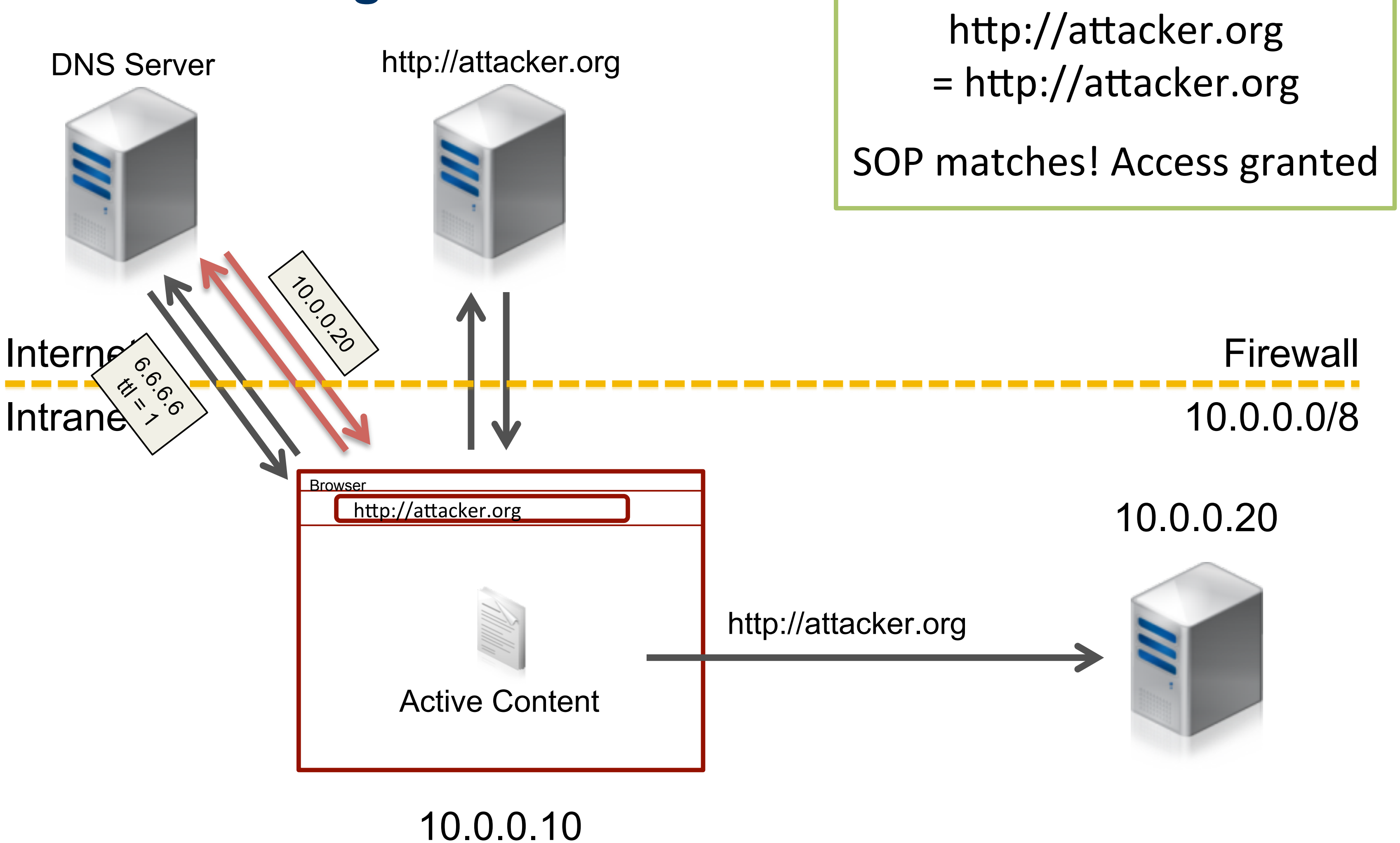


DNS Rebinding

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

DNS Rebinding



History Repeating

1996: The Princeton Attack

- in 1996 Java applets offered sophisticated networking capabilities
- DNS server returned two IP address for the same host
  1. The IP the applet was loaded from
  2. The IP of the target host

Countermeasure: Strict IP-based access control for Java applets

- Java applets are only allowed to connect to their server‘s IP address
- Maintainted over the entire lifetime of the applet
  - even inside the Browser‘s Java Cache

History Repeating

<u>2002: JavaScript</u>

- DNS Rebinding via domain relaxation
  - Domain 1 attacker.org ➔ 10.0.0.20
  - Domain 2 evil.attacker.org ➔ 6.6.6.6
- Quick-Swap DNS

Countermeasure: <u>Explicit domain relaxation</u>

- Both involved frames need to use domain relaxation

Countermeasure: <u>DNS-Pinning</u>

- Browser caches domain-to-IP mapping
- Browser resolves mapping only once per session

History Repeating

<u>2006: The full browser experience</u>

- FF & IE dropped domain-to-IP mapping on connection resets
- Leading to many DNS Rebinding vulnerabilities
  - JavaScript, Flash, Java, ...
  - Even allowing socket communication

Countermeasure: <u>Host-header checking</u>

- In HTTP 1.1, the browser attaches an additional header containing the hostname
- Applications need to check this header for correctness

Countermeasure: <u>Restrictive Networking Capabilities for plug-ins</u>

- Plugins are only allowed to connect to a limited set of ports

HTML5 Offline Application Cache Attack

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Abusing the Cache

- Idea: use the cache to store resource until domain-to-IP mapping is lost

- Abusing the cache for DNS Rebinding as such is straight-forward
  - However, „normal“ caching is not reliable

- HTML5 AppCache enables a
  - controllable caching behaviour
  - and thus, a way for content to easily exceed DNS pinning times

HTML5 AppCache

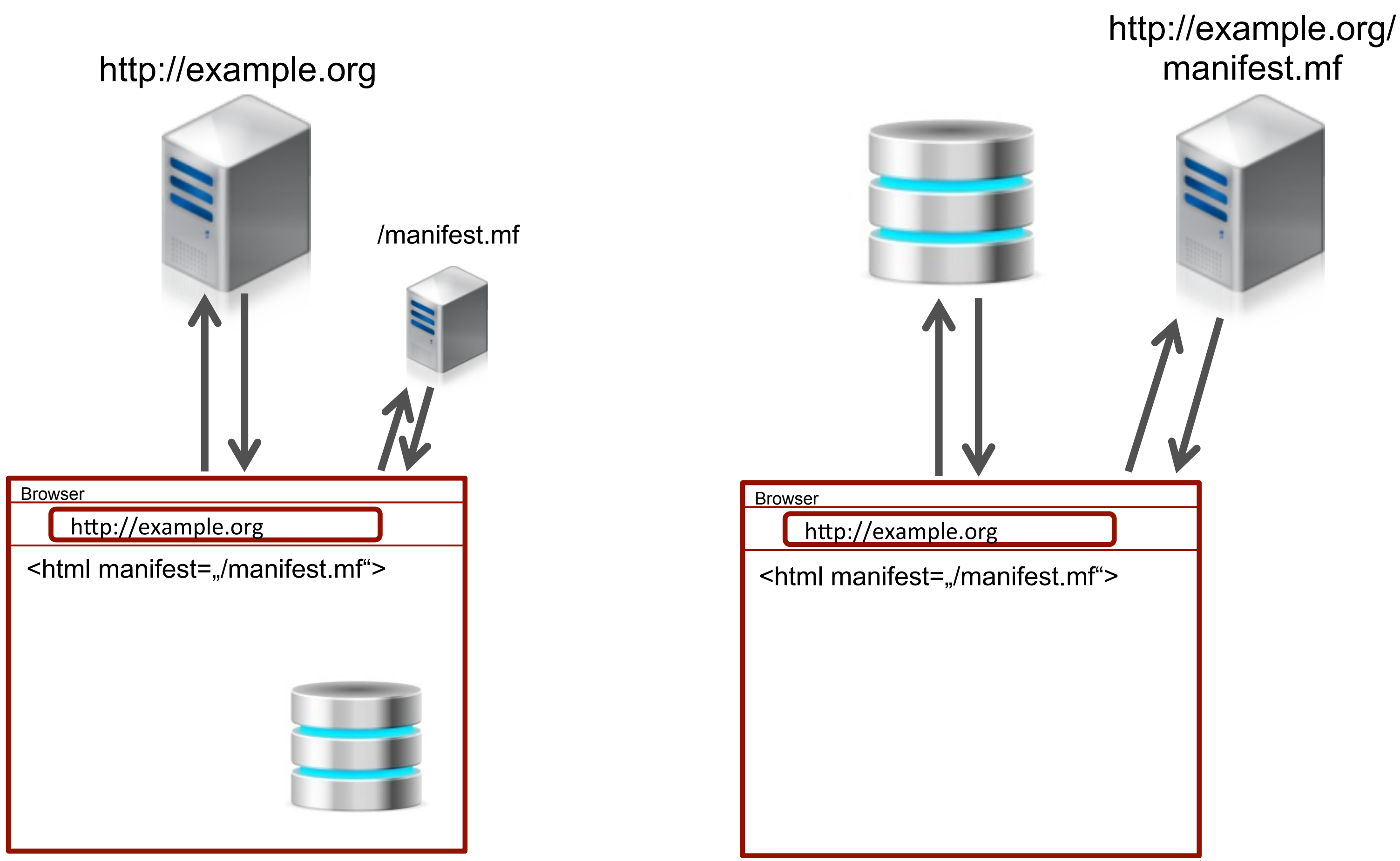
- Used to store parts of an application in the Cache
  - e.g. to reduce bandwidth consumption

- New attribute „manifest“ added in HTML5
  - URL to a file containing resources the browser should cache

```
CACHE MANIFEST

http://example.org/index.php
http://example.org/flash.swf
```

How the AppCache works



Abusing the HTML 5 AppCache

1. Store resources from http://attacker.org in the AppCache
2. Let the victim close the browser
3. Lure the victim to attacker‘s site again, resolve hostname to intranet server
4. Retrieve sensitive data and send it to attacker
5. manifest is downloaded again (will result in 404)
   - **We only have one shot**

Solution: Cross-domain caching

- AppCache allows us to store cross-domain resources
  - Have two domains – one for rebinding, one for manifest

- Domain attacker1.org hosts manifest and iframe with source attacker2.org/index.php

```
CACHE MANIFEST

http://attacker2.org/index.php
http://attacker2.org/flash.swf
```

- attacker2.org is rebound
- In the final step, manifest is retrieved from http://attacker1.org (still working)

History Repeating

2013: HTML5 Offline Application Cache

- Circumvents pinning abusing the application cache
- can reliably be used to scan ranges of IP addresses
- Works on almost all desktop browsers
  - IE does not allow for cross-domain caching

2013: Filling up the DNS Cache with bogus entries

- FireDrill by Dai & Resig (WOOT 13)

Countermeasure:

The extended Same-Origin Policy

The extended Same-Origin Policy

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

The three principals of Web interaction

- The Same-Origin Policy‘s duty is
  - to isolate *unrelated* Web applications from *each other*
  - based on the *origin* of the interacting resources

- Semantics of the SOP are built around two entities:
  - The ***Web client (browser)*** enforces the policy
  - The ***Web server*** provides the resources subject to the policy decision

- However, the involved entities differ:
  - The ***Web client (browser)*** enforces the policy
  - The ***DNS server*** provides the information used in the policy decision

Principal mismatch: Web server is not involved in the decision

Design Goals

- (DG1) Client-side enforcement
  - SOP is a client-side security policy and thus checking should be conducted in the browser
- (DG2) Protocol layer
  - Applications must not to be changed, only the protocol layer should be modified
- (DG3) Dedicated security functionality
  - Host header as such is not a security functionality
- (DG4) Non-disruptive
  - Our approach should not break existing browsers or applications

Extending the SOP with server-provided information

- Only the server should be capable of settings its trust boundary
  - Currently, the browser is guessing this boundary
  - based on information delivered by the network

- Therefore, we propose to extend the Same-Origin Policy
  - with server-provided input
  - delivered through an HTTP response header to be

{ protocol, domain, port, *server-origin* }

Extended Same-Origin Policy decision logic

The eSOP is satisfied iff:

$$\{\textbf{protocol}, \textbf{domain}, \textbf{port}\}_A == \{\textbf{protocol}, \textbf{domain}, \textbf{port}\}_T$$

and

$$\textbf{domain}_A \in \textbf{server-origin}_T$$

If the **server-origin**$_T$ property is empty, the second criterion always evaluates as "true".

Example

- 10.0.0.20's server-origin = { 10.0.0.20, wiki.corp }
- 2. part of the SOP decision: attacker.org ∈ of { 10.0.0.20, wiki.corp } → false
- Many edge cases are explained in the paper

Analysis of the eSOP

- The eSOP, summarized
  - client-side enforcement (DG1)
  - HTTP header used, no change to applications necessary (DG2)
  - HTTP header only used for security (DG3)
  - browsers fall back to „old“ SOP when header is not sent (DG4)

- We implemented a prototype into Chromium
  - consists of header extraction (array access) and string matching
    - actually in two separate places, but similar method
  - ➔ overhead not noticable

Conclusion

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Conclusion

- The Same-Origin Policy is the most basic security policy in the browser
  - it isolates unrelated Web applications from each other
  - based on the origin of the interacting resources (protocol, domain, port)
- DNS Rebinding circumvents the SOP
  - by associating a domain name with two unrelated IPs
  - vulnerabilities discovered in 1996, 2002, 2006 and 2013
- DNS Rebinding is a protocol-level flaw
  - Network governs the server's security characteristics
  - ➔ We enhanced the SOP with explicit server-origin to eridicate DNS Rebinding
- our approach was implemented within Chromium and proofed to have no overhead
- Opt-in, but on the target server-side

Future Work

- Rethink the notion of origins in the browser
  - Use the server-provided origin instead of the domain
- Adopt the newly developed SOP to other parts of the browser
  - password manager (e.g. defeats certain phishing attacks)
  - postMessage (currently only URL is known by recipient)
- Adopt policy for plugins
- Rethink CORS-like preflight requests
  - Different attacker model

Thank you for your attention

ben.stock@cs.fau.de
@kcotsneb

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT