Lightweight Integrity Protection for Web Storage-driven Content Caching

Sebastian Lekies and Martin Johns

SAP Research / WebSand Project
martin.johns@owasp.org

OWASP
07.11.2012

WebSand

The OWASP Foundation
http://www.owasp.org

Agenda

Technical Background

- Context
- What is Web Storage
- Use Cases for Web Storage

Attacks

- Insecure Usage
- Attack Scenarios

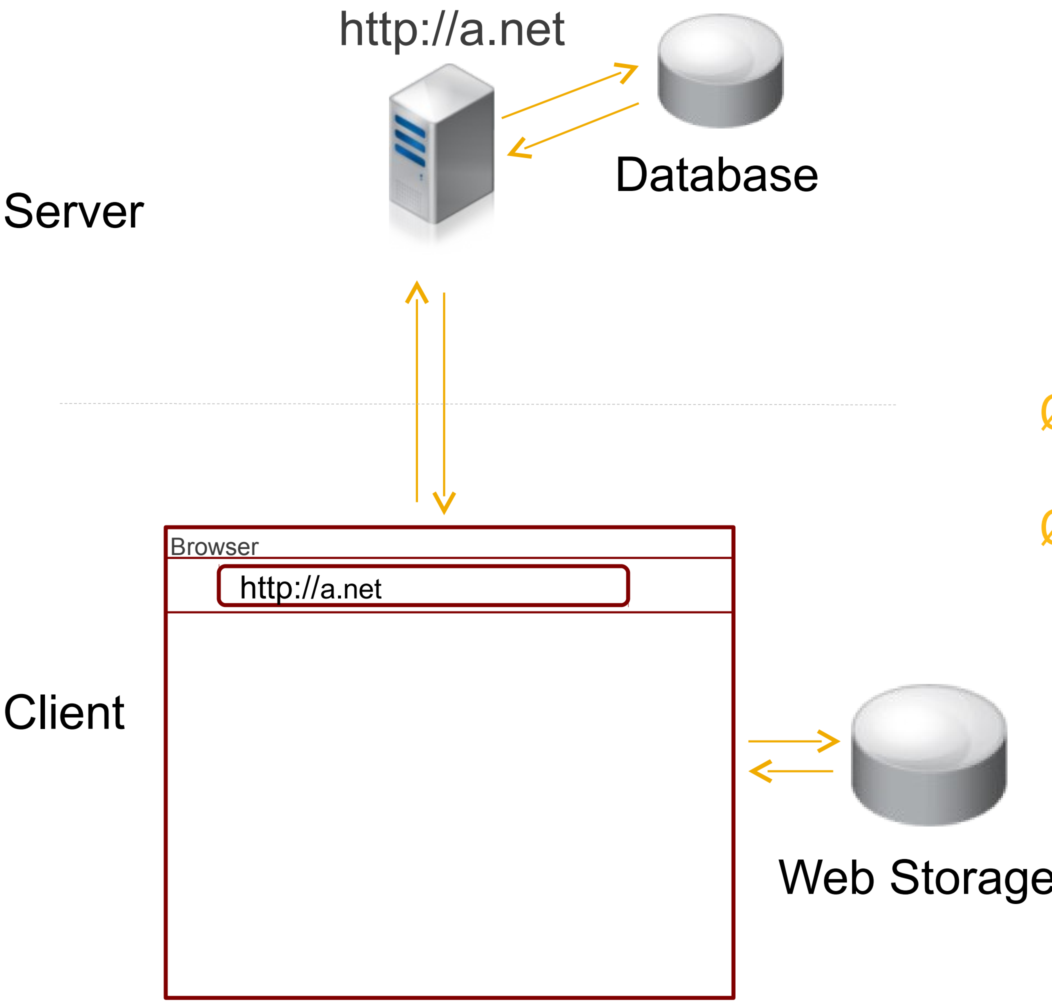
Survey

- Research Questions
- Results

Countermeasure

- Implementation
- Evaluation

Technical Background
Context



- Classical Web Applications…
- Not able to keep client-side state
- State is kept on the server side

- New use cases require client-side Storage
- Web Storage API introduced by HTML5

Technical Background
What is Web Storage?

> Web Storage is a mechanism that allows a piece of JavaScript to store structured data within the user's browser (on the client-side).

- Web Storage consists of three APIs
- Local Storage
- Session Storage
- Global Storage (deprecated)
- Other new client-side storage technologies exist
- IndexedDB
- Web SQL Databases
- File API

- Scope of this Presentation is limited to Local Storage, however our findings apply for all client-side storage technologies

Technical Background
What is Web Storage?

```
<script>
  //Set Item
  localStorage.setItem("foo","bar");
  ...
  //Get Item
  var testVar = localStorage.getItem("foo");
  ...
  //Remove Item
  localStorage.removeItem("foo");
</script>
```

- Access to Web Storage API is restricted by the Same-Origin Policy
- Each origin receives its own, separated storage area

- Origin is defined by

$$\underbrace{\text{http://}}_{\text{protocol}}\underbrace{\text{www.example.org}}_{\text{host}}\text{:}\underbrace{\text{8080}}_{\text{port}}\text{/some/webpage.html}$$

Technical Background
Use Cases for Web Storage

- Client-side state-keeping
- E.g. for HTML5 offline applications
- Store state within Local Storage and synchronize state when online

- Using Web Storage for controlled caching
- Current caching mechanism only allow storage of full HTTP responses
  - Transparent to the application and hence “out of control”
- Web Storage is useful when…
  - only sub-parts of HTML documents needs to be cached e.g. scripts
  - close control is needed by the application
- Especially important in mobile environments

Attacks
Insecure Usage

- Observation: Web sites tend to cache content that will be executed later on
- HTML-Fragments
- JavaScript code
- CSS style declarations

```
<script>
  var content = localStorage.getItem("code")

  if(content == undefined){
    content =
fetchAndCacheContentFromServer("code");
  }

  eval(content);
</script>
```

Attacks
Insecure Usage

- First thought (WebApp Sec Pavlovian reaction): XSS!!!11!
- If the attacker can control the content of the Web storage, he can execute JavaScript!!!

Second thought: This behavior is safe

- Web storage can only be accessed by same-origin resources
- So you need JS execution to cause JS execution

Third thought: What if an attacker is able to circumvent this protection?

- Persistence: This is a persistent attack scenario
  - Even if the causing vulnerability has been resolved in the meantime
- Client-side: Attack payload exists purely in the compromised browser
  - Invisible from the server-based point-of-view
- “WebApp rootkit”

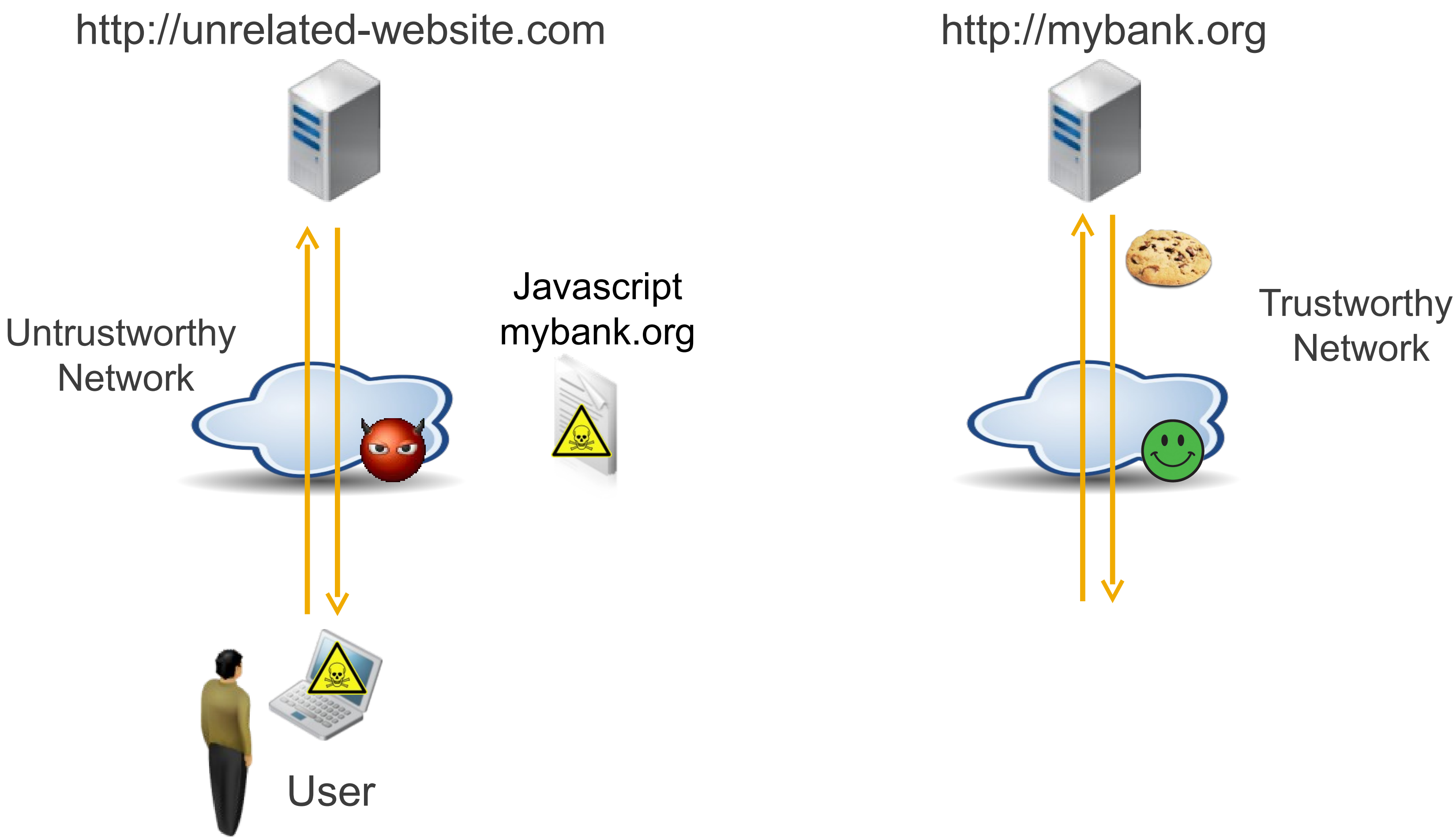
Attacks
Attack scenarios: Cross-Site Scripting

Scenario: Reflected XSS problem somewhere in the site

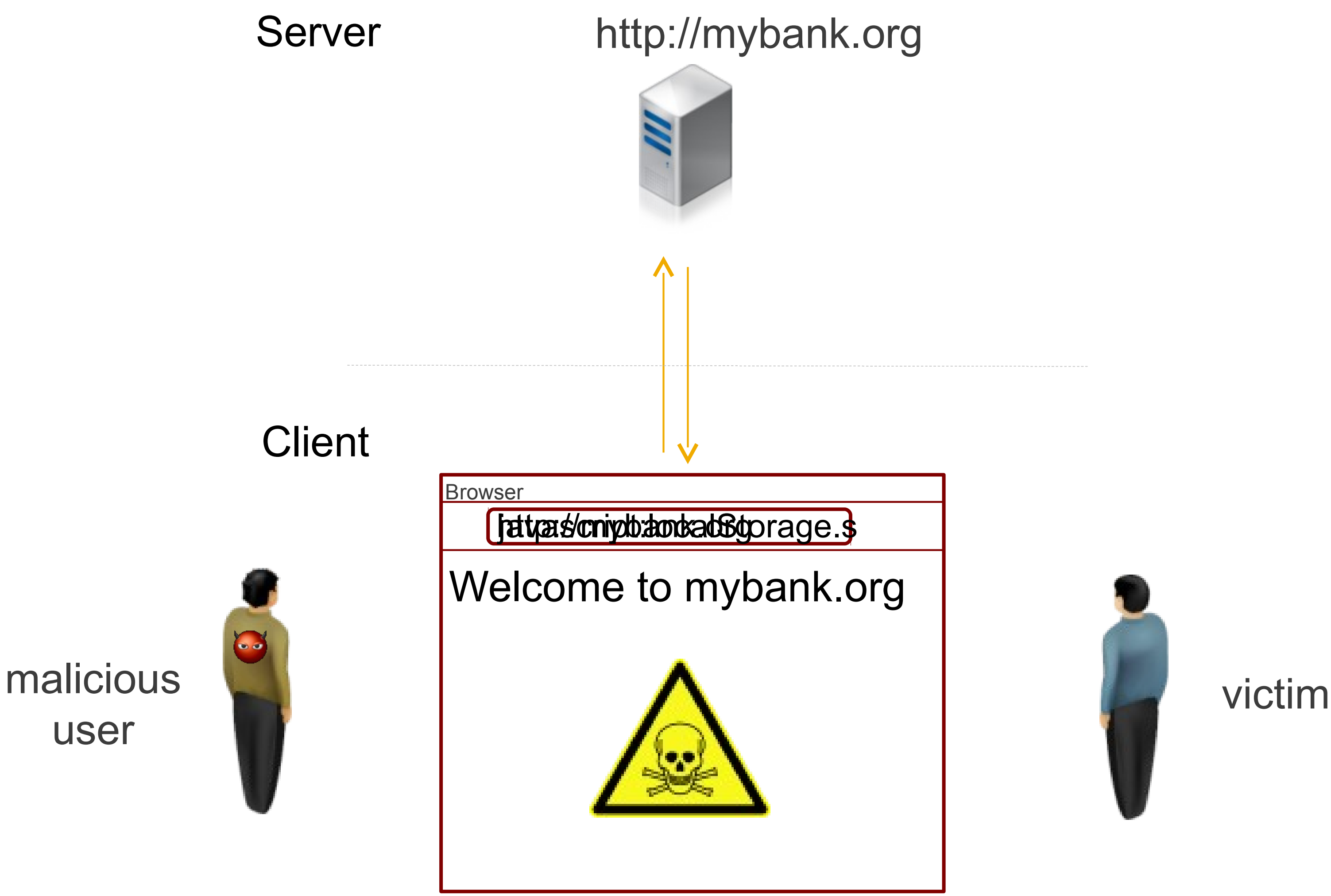
- Vulnerability that does not necessarily require an authenticated context / session
- Attacker can exploit this vulnerability while the user is interacting with an unrelated web site

- E.g., a hidden iFrame pointing to the vulnerable application
  - During this attack, the malicious payload is persisted in the user's browser
- The payload now "waits" to be executed the next time the victim visits the application
  - This effectively promotes a reflected unauthenticated XSS into a **stored authenticated XSS**
- Hence, the consequences are much more severe
  - Furthermore, the payload resides a prolonged time in the victim's browser
- Invisible for the server

Attacks
Attack scenarios: Untrustworthy Network



Attacks
Attack scenarios: Shared Browser



Survey
Methodology

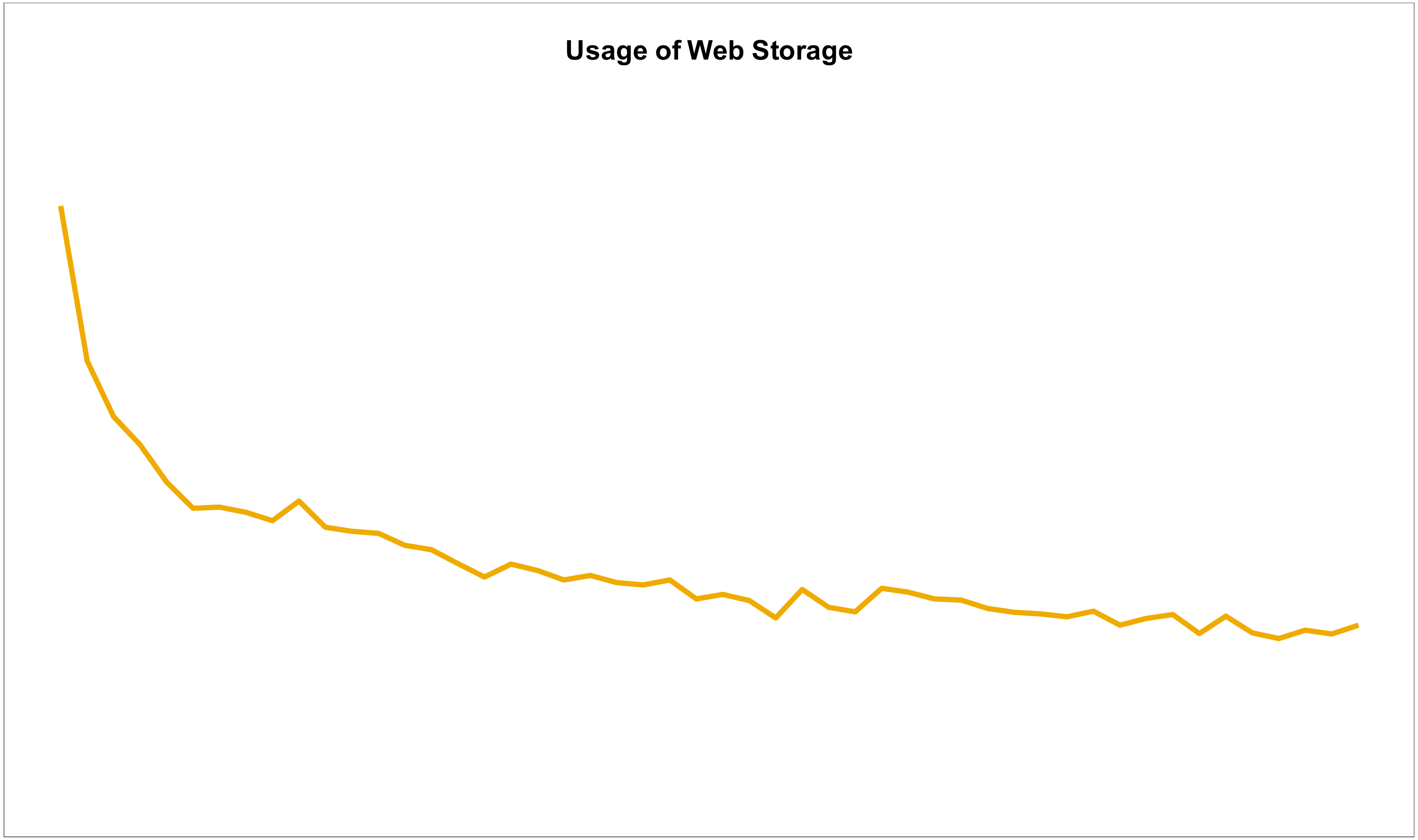
Scope

- Crawl of the Alexa Top 500,000 Web sites

Research Questions

1. Penetration:
2. How many Web sites utilize Web Storage?
3. What kinds of storage APIs are used(Local-, Session- or GlobalStorage)?
4. Does a relation exist between the popularity of a Web site and the usage of Web Storage?
5. Security:
6. How many Web sites utilize Web Storage for storing code fragments?
7. How many Web Sites utilize Web Storage in a secure/insecure fashion?

Survey – Results
Penetration



Survey – Results
Penetration

Name	**Total**	**Web sites**	**% Sites**
Crawled Pages	500,000	500,000	100 %
Total Web Storage Accesses	122,615	20,421	4.08 %
LocalStorage Accesses	82,884	18,811	3.76 %
SessionStorage Accesses	39,068	11,288	2.26 %
GlobalStorage Accesses	663	202	0,04 %
via getItem()	81,811	19,890	3,98 %
via setItem()	35,823	16,169	3,23 %
via removeItem()	4,981	2385	0,48 %

TABLE I: General overview of crawling results

Survey – Results
Security

Categorization

- **Problematic:** Code that is *very likely executed* by the Web site (e.g. HTML, Javascript, CSS)
- **Suspicious:** Code that could *potentially be executed*. (e.g. JSON data: Secure parsing via JSON.parse or insecure execution via eval)
- **Unproblematic**: Content that is *unlikely being executed*. (e.g. numbers, alphanumeric strings, empty values)

- **Methodology**
- Prefiltering: Values not containing “<“,”>”,”{“,”}” were marked unproblematic
- Manual categorization of the remaining items.

Survey – Results
Security

Name	Number
Containing brackets	10,547
Empty JSON ("{}")	5,055
JSON without code or markup	3,408
Code or Markup	2,084

- **An additional interesting attack vector**
- 68 entries of the "JSON without code" category contained URLs to Javascript or CSS files
- Manual inspection revealed that those URLs were used to fetch additional content
  - Manipulation of these URLs leads to code execution capabilities
  - Hence these entries were also marked as problematic

Name	Number
Problematic	2,152 (2,084 + 68)
Suspicious	8,305 (3,340 + 5,055)
Unproblematic	112,158

Countermeasure

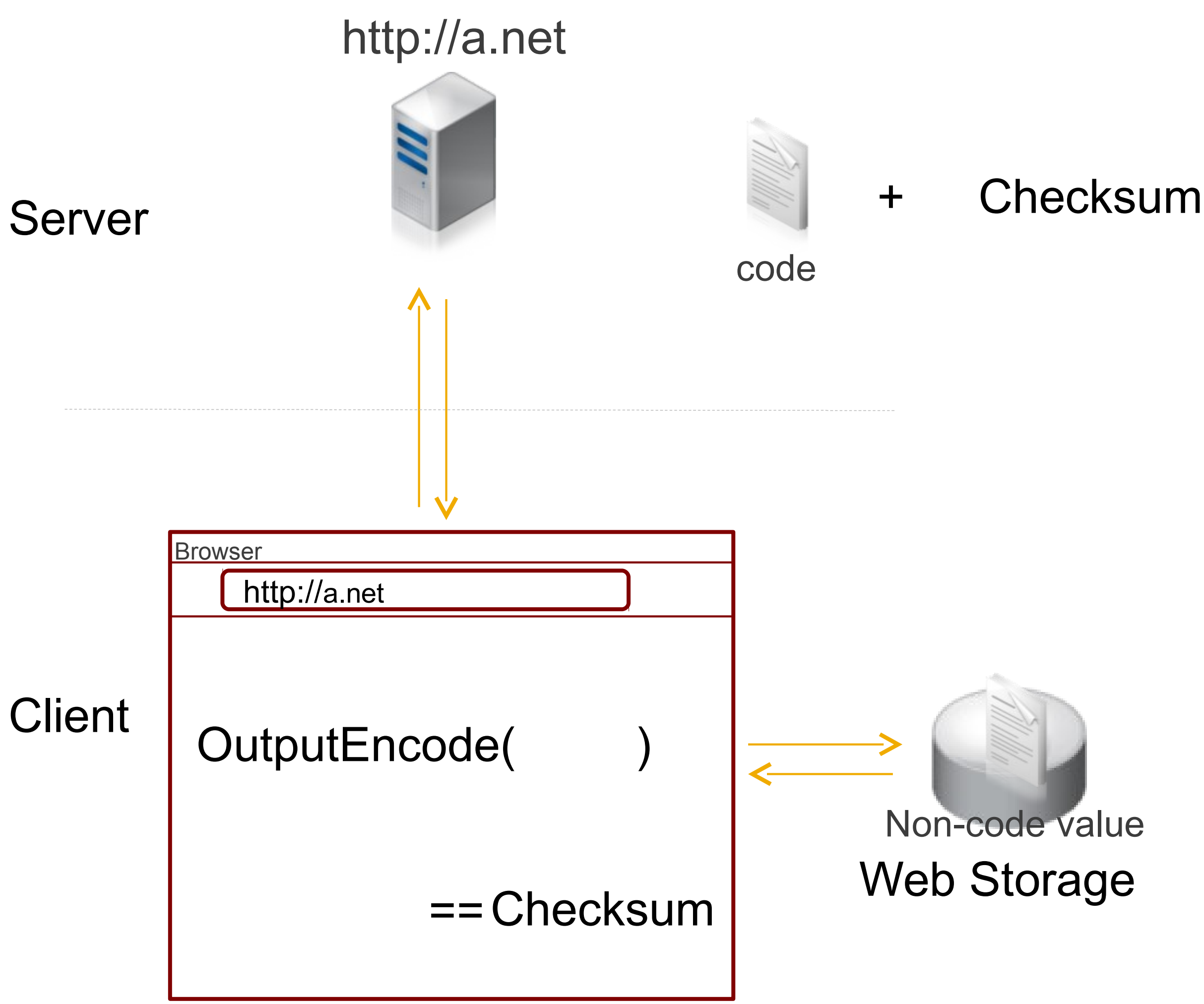
Problem

- Anti-XSS techniques such as output encoding do not work

- This would make cached code unusable
- This would not help against the URL attacks
-

- **Alternative**
- Verify that values from Web Storage originate from your application and that integrity is guaranteed

Countermeasure
Implementation



Countermeasure

Implementation:

- JavaScript Library:

```
<script type="text/javascript" src="./webStorageWrapper.js">
```

- Transparent to the applications by utilizing function wrapping techniques:

```
var wrapper = new StorageWrapper();

Object.defineProperty(window, "localStorage", {value: wrapper});
```

-

```
//Get Item
var testVar = localStorage.getItem("foo");
```

Countermeasure
Evaluation

Performance

- Two performance critical steps:
  - Transfer of our library to the client
  - Calculation of checksums on the client-side
- **Transfer of our Library**
- Size of the library 563 bytes (packed) + 1731 for SHA256 = 2,294 bytes in total
- Average size of the 2,084 code fragments: ~76,000 bytes
- Hashing library not necessary in future (with the JS Crypto API available)

Calculation of checksums

- For collision free checksums we chose SHA256 as a hashing algorithm
- To evaluate hashin de values from our survey

Browser	Total time in ms	Average
Firefox	55,790	0,026 s
Safari	51,284	0,024 s
Chrome	55,087	0,026 s
Opera	180,372	0,086,s

Conclusion

Web Storage is a client-side storage mechanism that is used for

- Client-side state keeping
- Content caching

- **Caching code within Web Storage is a dangerous practice**
- Enables second order attacks
- Used in practice
- Usage is likely to increase

Traditional Anti-XSS mechanisms are not applicable for cached code

- Would make cached code unusable

We proposed a lightweight integrity preserving mechanism for Web Storage

- Enables secure usage of Web Storage
- Preserves benefits
- Only implies very small overhead

Thank you

Contact information:

Martin Johns
martin.johns@sap.com
http://martinjohns.com
@datenkeller