# Don't Trust The Locals: Exploiting Persistent Client-Side Cross-Site Scripting in the Wild

**Marius Steffens**
German OWASP Day 2018

joint work with Christian Rossow, Martin Johns and Ben Stock

OWASP
German Chapter

UNIVERSITÄT
DES
SAARLANDES

CISPA
HELMHOLTZ-ZENTRUM i. G.

# Dimensions of Cross-Site Scripting

| | Server | Client |
|---|---|---|
| Reflected | `echo "Welcome ".`<br>`  $_GET["name"];` | `document.write("Welcome " +`<br>`  location.hash.slice(1));` |
| Persistent | `mysql_query("INSERT INTO posts ...");`<br>`// ..`<br>`$res = mysql_query("SELECT * FROM`<br>`posts");`<br>`while ($row = mysql_fetch_array($res)) {`<br>`  print $res[0];`<br>`}` | `localStorage.setItem("name",`<br>`  location.hash.slice(1));`<br>`// ..`<br>`document.write("Welcome " +`<br>`  localStorage.getItem("name"));` |

# Persistent Client-Side XSS?

“With the advent of HTML5, and other browser technologies, we can **envision** the attack payload being permanently stored in the victim’s browser, such as an HTML5 database, and never being sent to the server at all.”

- OWASP Wiki

# Research Questions

- How many sites of the top 5k make use of data from storages in their client-side code?

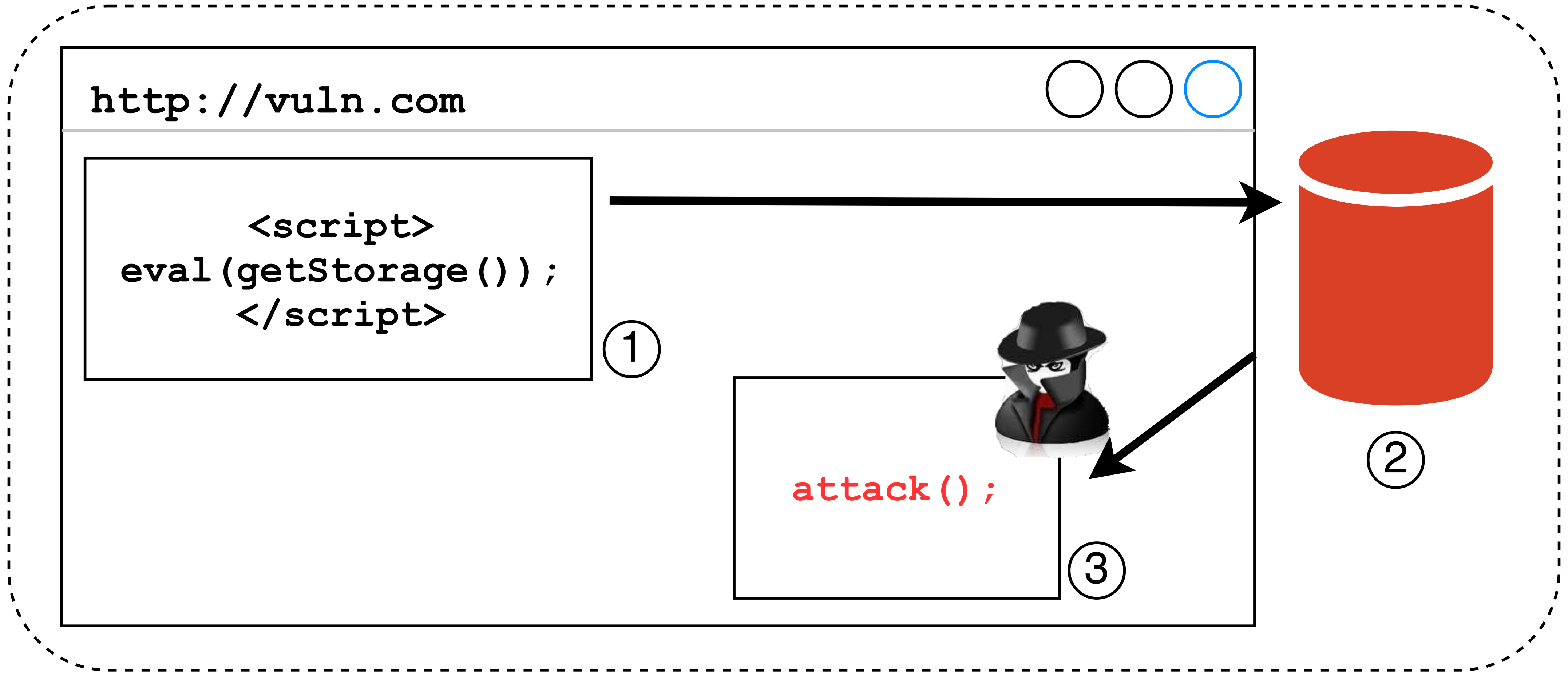
- On how many sites can such a data flow be abused if an adversary can gain control over the storage?

- Out of these, how many sites can be successfully attacked by a network and Web adversary?

- To answer: combine taint tracking with automated exploit generation
  - Our previous work (Lekies et al. CCS 2013) + a number of improvements for their shortcomings

# Persistent Client-Side Cross-Site Scripting

- Client-side technology allows for storing of data and code
  - Cookies: typically used for preferences and configuration (e.g., language)
    - bound to eTLD+1 or hostname only
    - limited storage (typically 4096 bytes), limited charset (e.g., cannot contain semicolon)
  - Web Storage: used to persist larger pieces of data
    - bound to origin of the site
    - Session Storage: persisted only within the same browser window
    - Local Storage: shared across all windows



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

```
attack();
```

# Interlude: HTTP Strict Transport Security

- HTTP header (Strict-Transport-Security) sent by server
  - only valid if sent via HTTPS
  - Strict-Transport-Security: max-age=<expiry in seconds>
    - includeSubDomains: header is valid for all subdomains
    - preload: allows for inclusion in preload list
  - ensures that site cannot be loaded via HTTP until expiry is reached

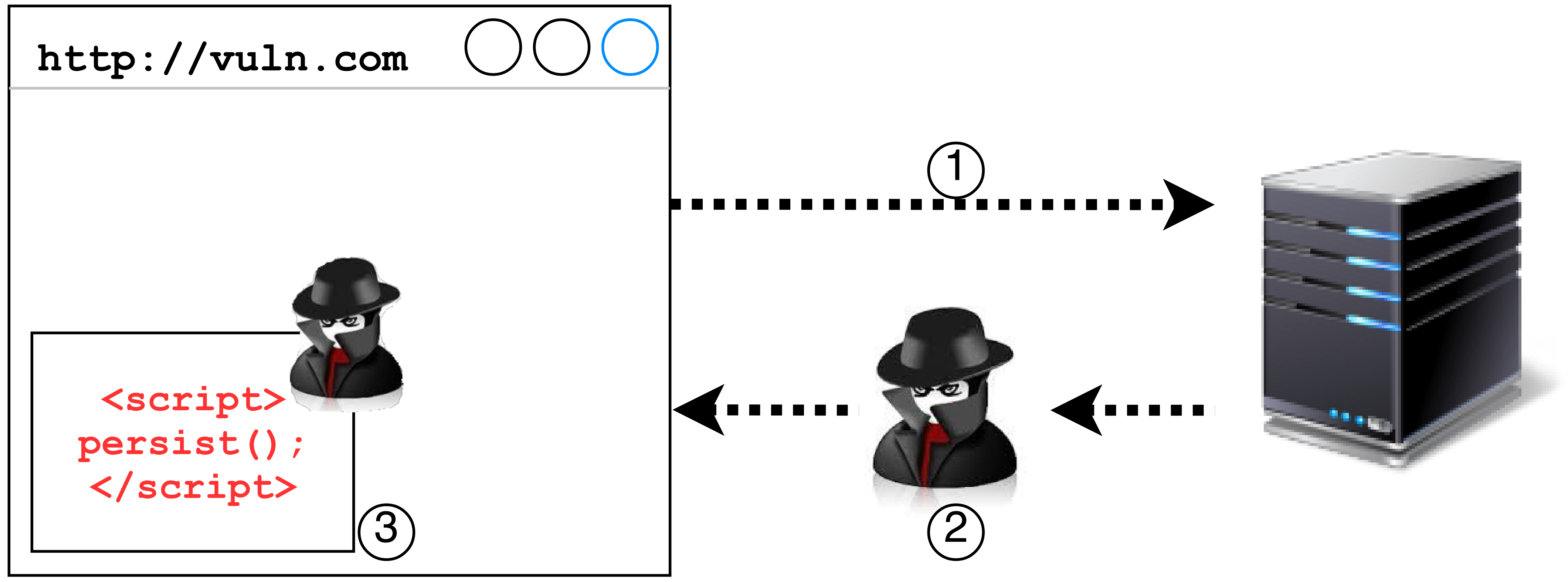
- Domains can be preloaded in browsers
  - HSTS preload list (https://hstspreload.org/)
  - only possible with at least 18 weeks max-age, includeSubDomains and automatic redirect from HTTP

# Persistent Client-Side Cross-Site Scripting: Attacker Models

- Requirement for successful attack: persisted malicious payload
  - extracted on every page load; single "infection" is sufficient

- Attacker Model #1: Network Attacker
  - can modify unencrypted connections
  - cannot get arbitrary TLS certificates

- Capabilities
  - Cookies: set cookies for any domain without HSTS
    - or with HSTS but without includeSubDomains
  - Local Storage: inject items on HTTP sites only



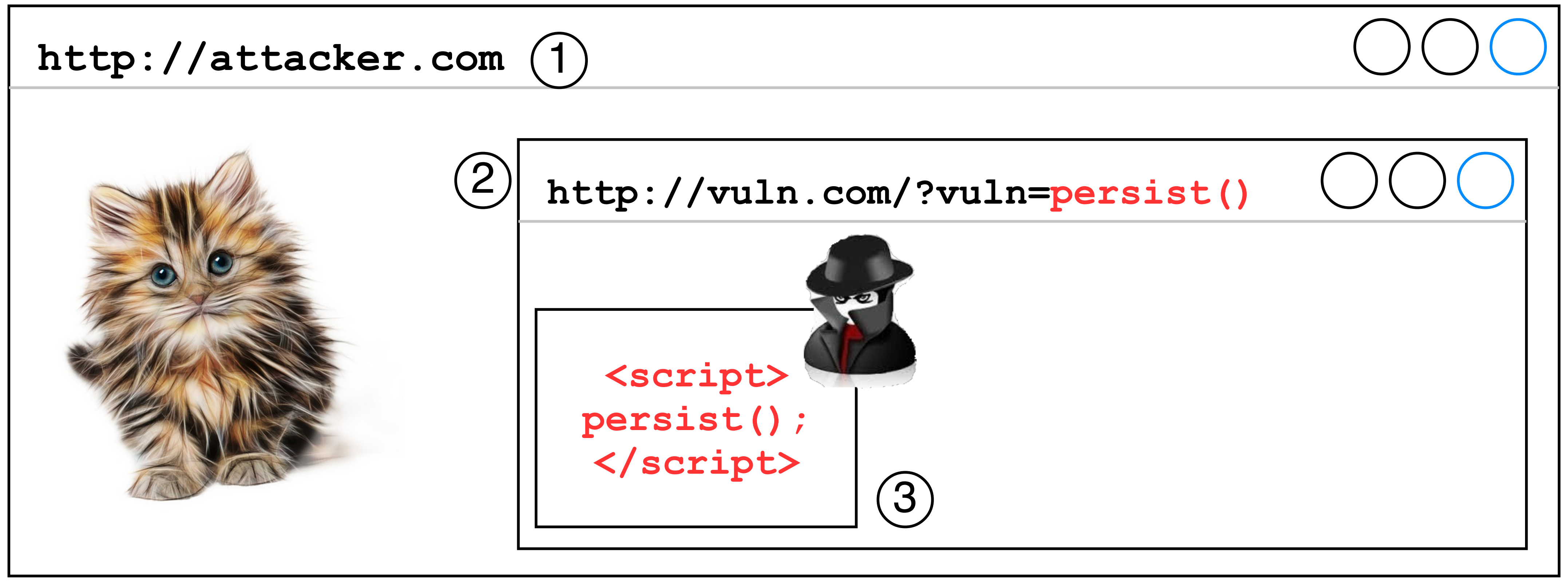
```
<script>
persist();
</script>
```

# Persistent Client-Side Cross-Site Scripting: Attacker Models

- Attacker Model #2: Web Attacker
  - can force victim's browser to visit any URL

- Attack Vector #1: Abuse existing XSS flaw
  - allows to inject data into origin (Storage) or domain (cookies)
  - HTTPS does not help at all

- Attack Vector #2: Abuse flows into storage
  - requires a flow into storage item
  - important: same storage item must be later on used
  - hard to find in practice



# Persistent Client-Side Cross-Site Scripting: Potential Attacks

- Question may arise: why bother with per-user persistent XSS if we need an XSS or an active network adversary in the first place?

- Potential answers
  - Infect storage with keylogger - wait for next login
  - Security-aware user might not login in untrusted Wifi
    - but will in his home network
  - Cryptojacking, there is always Cryptojacking

# Sites with flows from cookies/Local Storage

|  | Cookies |  |  | Local Storage |  |  |
|---|---|---|---|---|---|---|
| Sink | Total | Plain | Exploits | Total | Plain | Exploits |
| HTML | 496 | 319 | **132 (27%)** | 234 | 226 | **105 (45%)** |
| JavaScript | 547 | 470 | **72 (13%)** | 392 | 385 | **108 (27%)** |
| script.src | 1385 | 533 | **17 (1%)** | 626 | 297 | **11 (2%)** |
| Total | 1645 | 906 | **213(13%)** | 941 | 654 | **222 (24%)** |

# Exploitability under attacker models

- 293 sites can be exploited by a network attacker
  - no HTTPS at all or
  - due to lack of HSTS or lack of includeSubdomains

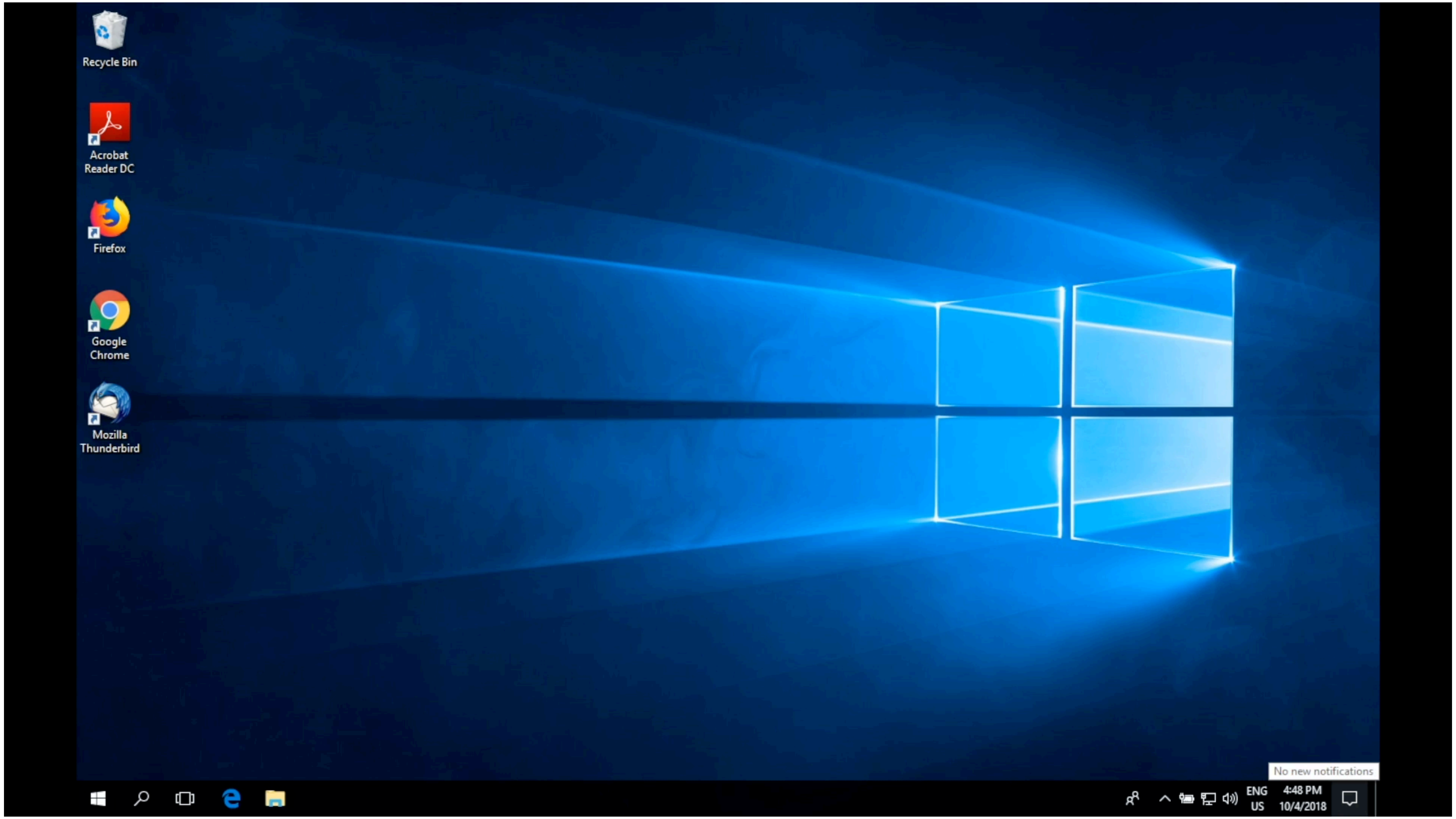
- 65 sites have reflected client-side XSS in the same origin

- Lower bound on Web attacker
  - no code coverage, no login
  - not considered any other form of XSS
  - not considered "trust relations" (domain relaxation, postMessages, ...)
    - 15% of all page loads seem to set document.domain
  - not investigated inter- and intra-storage flows (around 100M data flows in our data)

# Types of exploitable stored content

- Unstructured Data (214 domains)
  - Can be addressed via proper encoding
- Structured Data (such as JSON, 108 domains)
  - Guess what, don't use eval!
- Client-Side Code Caching (HTML / JavaScript, 101 domains)
  - Service Workers for JavaScript
  - Integrity measures
- Configuration Information (such as Hostnames, 28 domains)
  - solution depends: mostly whitelisting actually works

# PoC Two Stage Exploit



# Summing up

- Conducted large-scale study on Alexa Top 5,000
- 1,946 domains make use of storage data in their application
    - 1,324 domains do so without encoding at least once
- 418 domains have exploitable flow from storage
    - 213 from cookie, 222 from Local Storage
- Real-world exploitability by attacker models
    - 293/418 domains vulnerable to network attacker
    - 65/418 domains vulnerable to Web attacker
- Persistent Client-Side is a more widespread issue than might have been assumed