

# ZigZag

against Client-side Validation Attacks

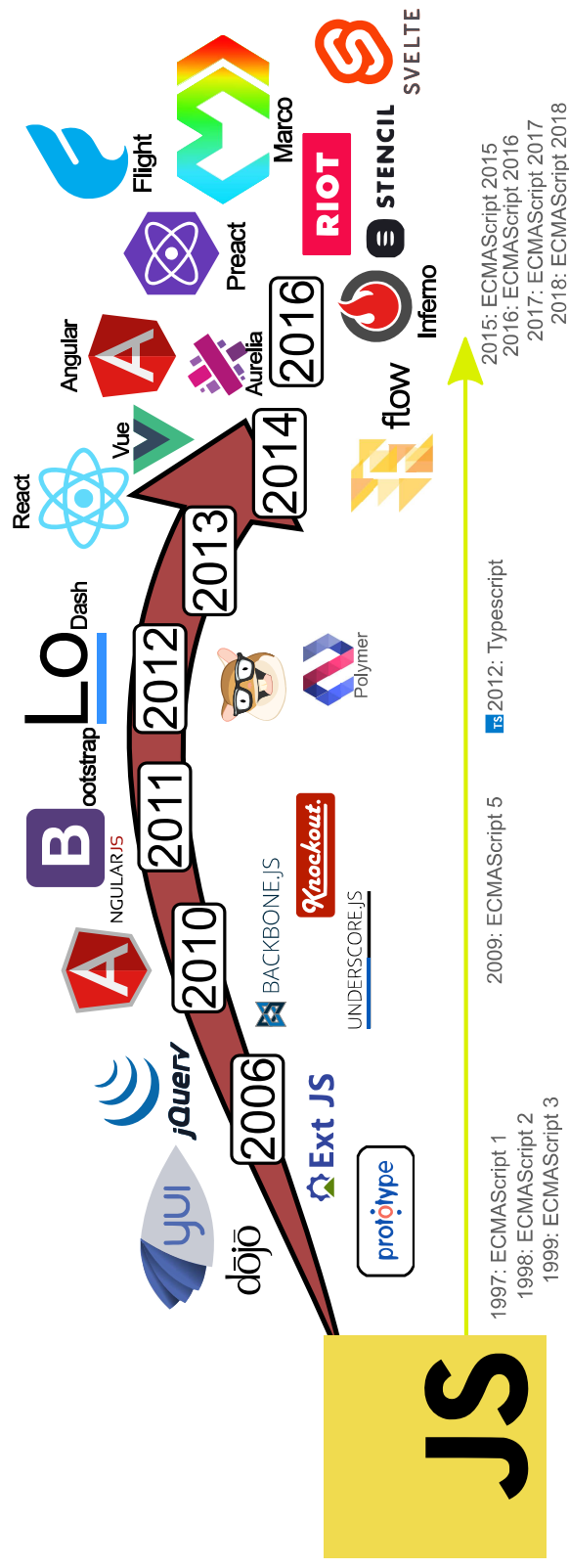
What is Client-side Validation  
(CSV)?

# Validation of Untrusted Input

- HTTP Referrer
- Browser Information
- Cookie values
- URL parameters & fragment
- Form data
- Cross-document communication
- ...

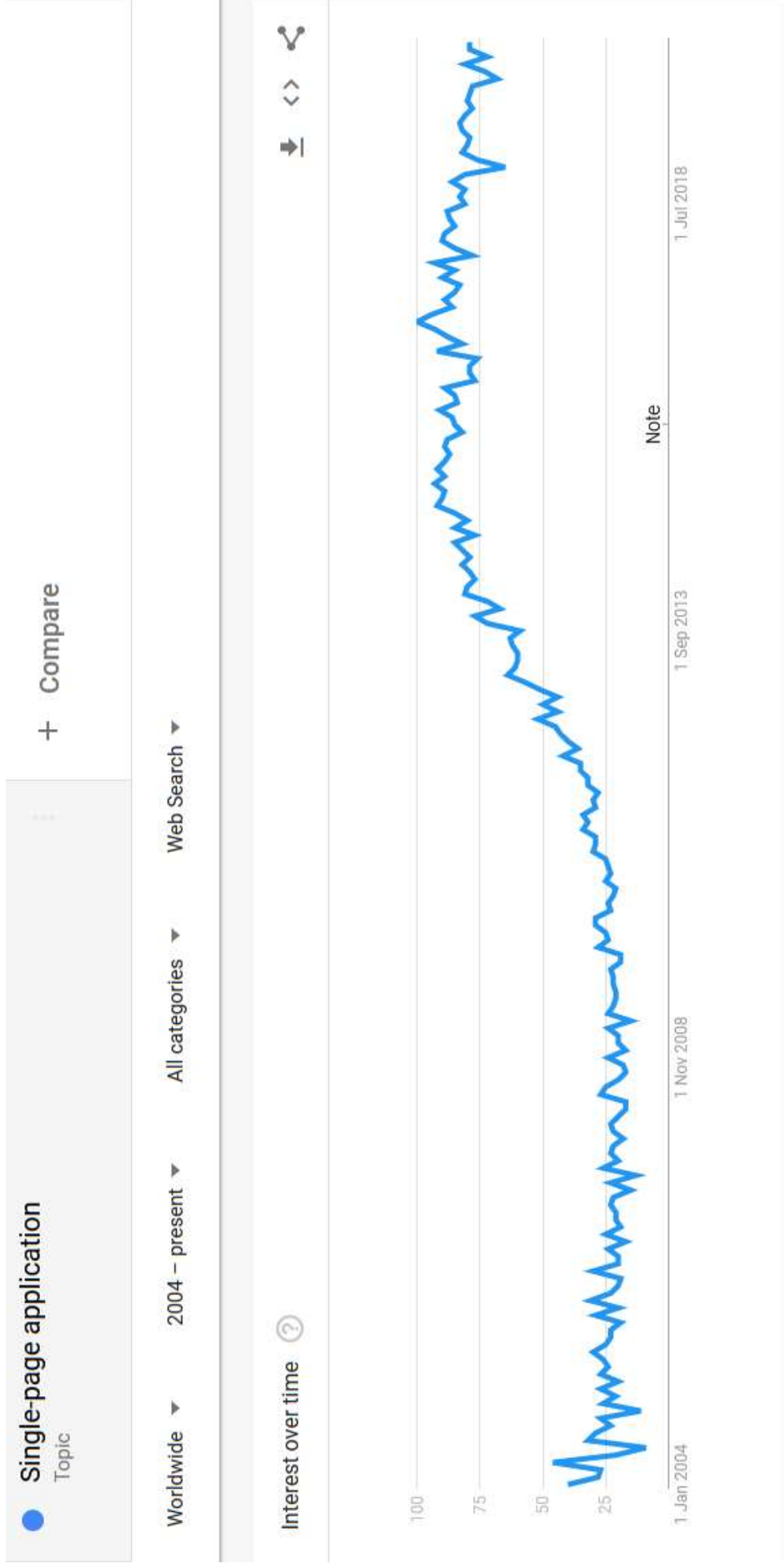
**BUT:** Not all data is processed by the server!

# The Rise of JavaScript





# The Rise of JavaScript



How do CSV Attacks work?

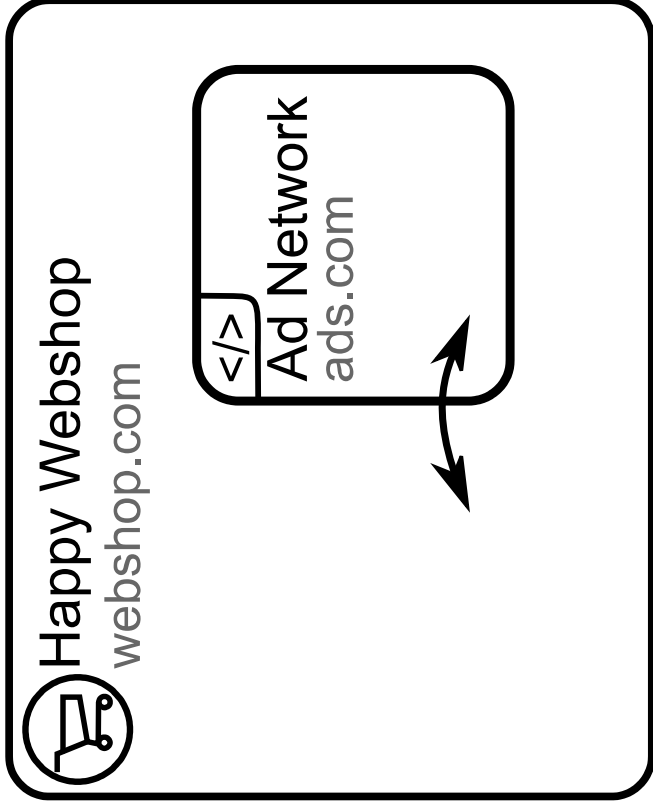
# Vulnerable example

```
1 // Handle a received message
2 var receiveMessage = function(e) {
3   // Missing check on e.origin!
4   // ...
5 }
6
7 var sendMessage = function(e) {
8   // Send data to window
9   window.postMessage(data, "*");
10 }
11
12 // Register for messages
13 window.addEventListener("message", receiveMessage, false);
```

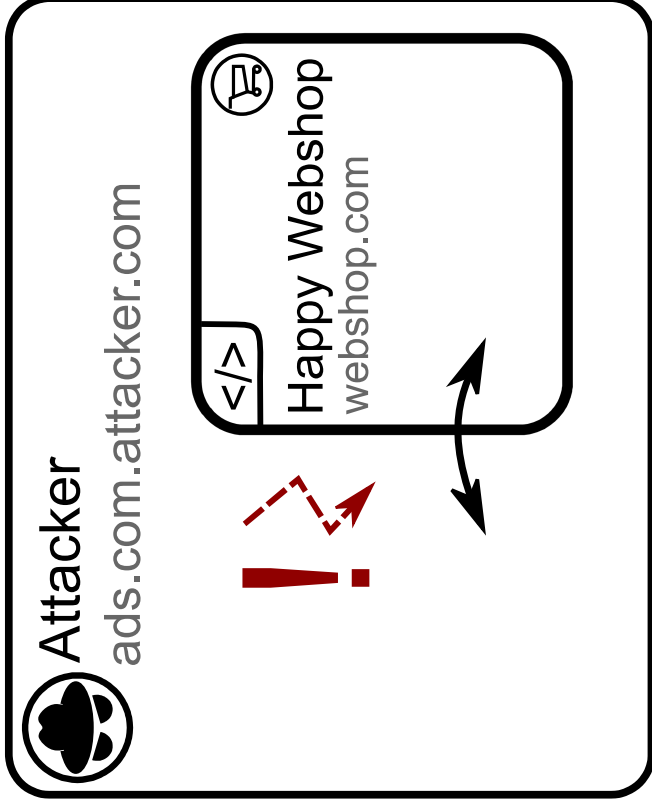


# Vulnerable example

```
1 // Handle a received message
2 var receiveMessage = function(e) {
3   // Missing check on e.origin!
4   // ...
5 }
6
7 var sendMessage = function(e) {
8   // Send data to window
9   window.postMessage(data, "*");
10 }
11
12 // Register for messages
13 window.addEventListener("message", receiveMessage, false);
```













# Critical Effects on the Client

- Origin mis-attribution
- XML-HTTP-Requests (Command injection)
- Accessing private data (Session hijacking)
- Document manipulation (XSS)





# How to prevent CSV attacks?

# Previous works






-  Program analysis for finding vulnerabilities
-  Language-based protection
-  Sandboxing potentially vulnerable code
-  Signature and anomaly checks



# Problems

-  No same origin policy
-  Difficulty of training developers
-  Rapidly evolving web platform
-  Highly dynamic interaction across documents





# ZigZag's approach

-  Fully automatic
-  No modifications to source code
-  No browser modifications needed
-  Dynamic reactions to changed conditions
-  Deployable by website operators or third parties




How does that work?

# Core Idea

## Anomaly Patching

-  Find security-relevant application parts
-  Trace normal execution flow
-  Deduce likely restrictions for variables
-  Enforce restrictions on application calls

# Find relevant functions

-  Determine checkpoints for tracing
-  Use entry and exit points of callback functions
-  Use static analysis to detect relevant API sinks

# Trace execution flow

## Store values of...

- Function parameters
- Caller / callee pairs
- Return values

# Deduce restrictions of...

## ...single values

- `type (typeof origin === "string")`
- `equality (origin === "ads.com")`
- `length (origin.length < 8)`
- `isJSON, isPrintable, isEmail, isURL, ...`

# Deduce restrictions of...

...multiple values

- equality ( $x === y$ )
- inequality ( $x < y$ )
- also for same type, isJSON, ...



# Enforce restrictions

- Repeat tracing step
- Terminate application if invalid
- Warn about violations

Any issues with this?

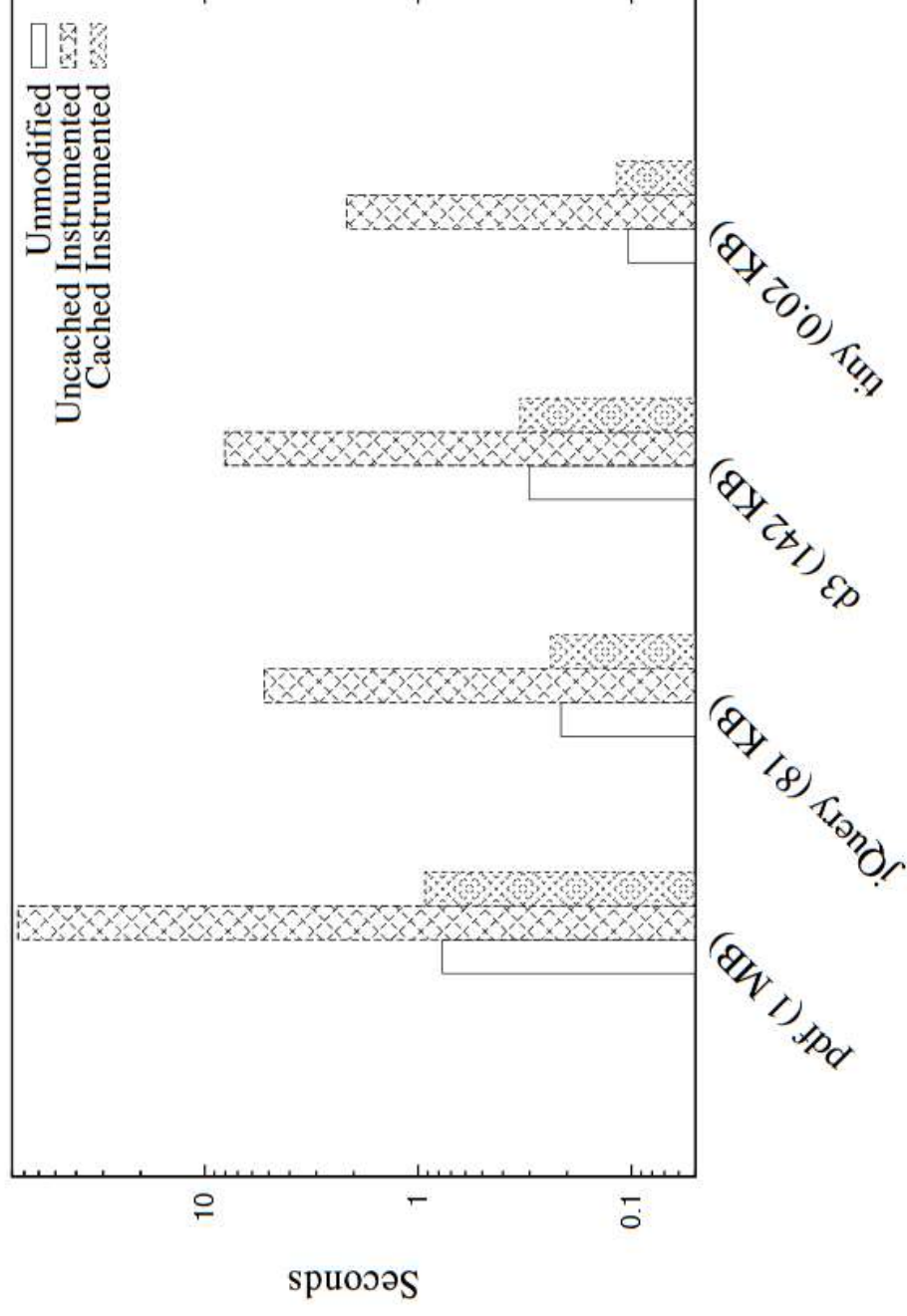
# Training Process

- Requires training in order to function
- Too few test runs  $\Rightarrow$  false-positives
- Learning phase requires attack-free usage
- End-to-end tests as possible base

# Generated Code

- Detection of dynamic code rewriting
- Detection of templated code generation
- Abstracting of restrictions for code classes

# Performance



# Security

- Protects against anomalies
- Does not hide itself
- No protection against other XSS attacks

# Does it actually work?

- Tested on Alexa Top 50
- No false-positives
- Working on all but one page

Further questions?