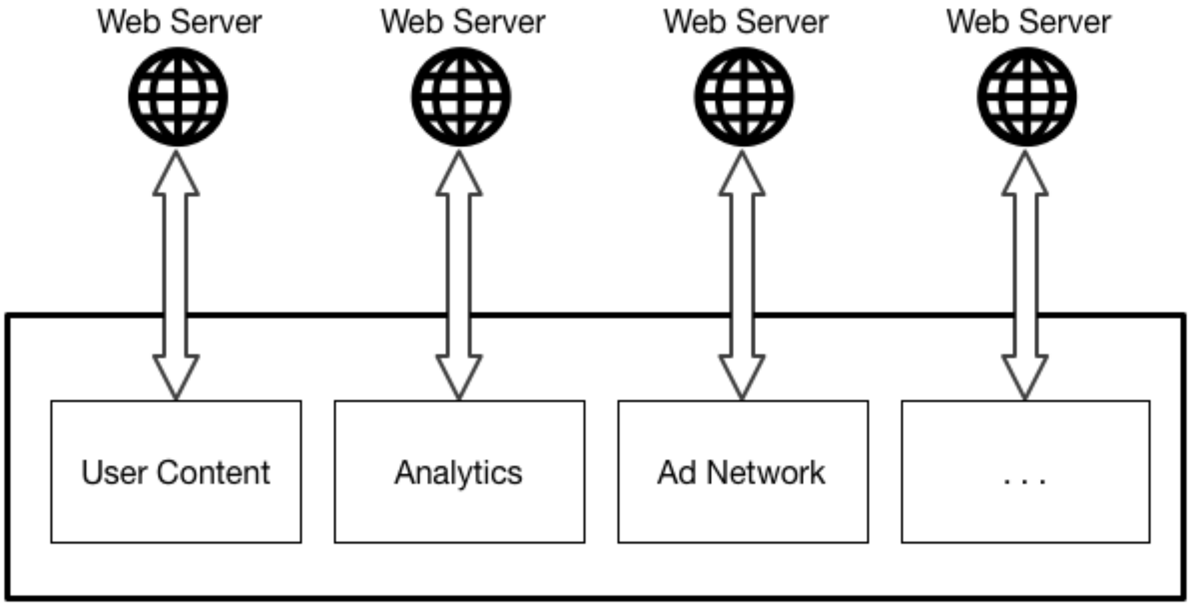
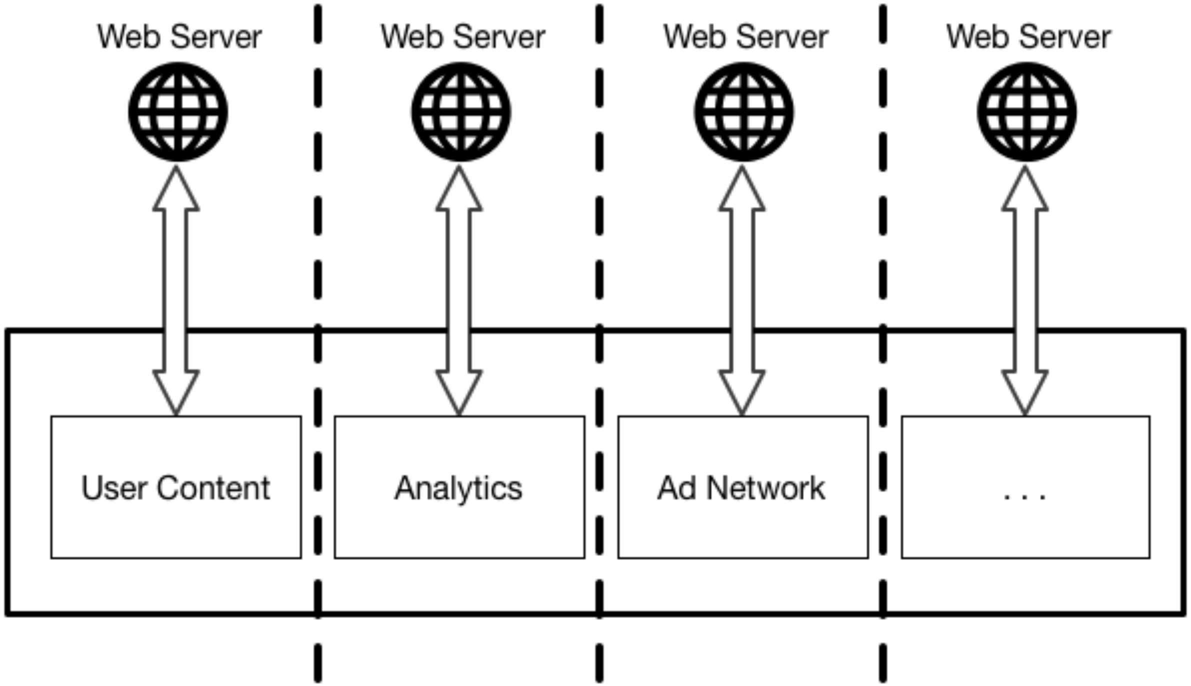
ZigZag - Automatically Hardening Web Applications Against Client-side Validation Vulnerabilities

Michael Weissbacher, William Robertson, Engin Kirda, Christopher Kruegel, Giovanni Vigna

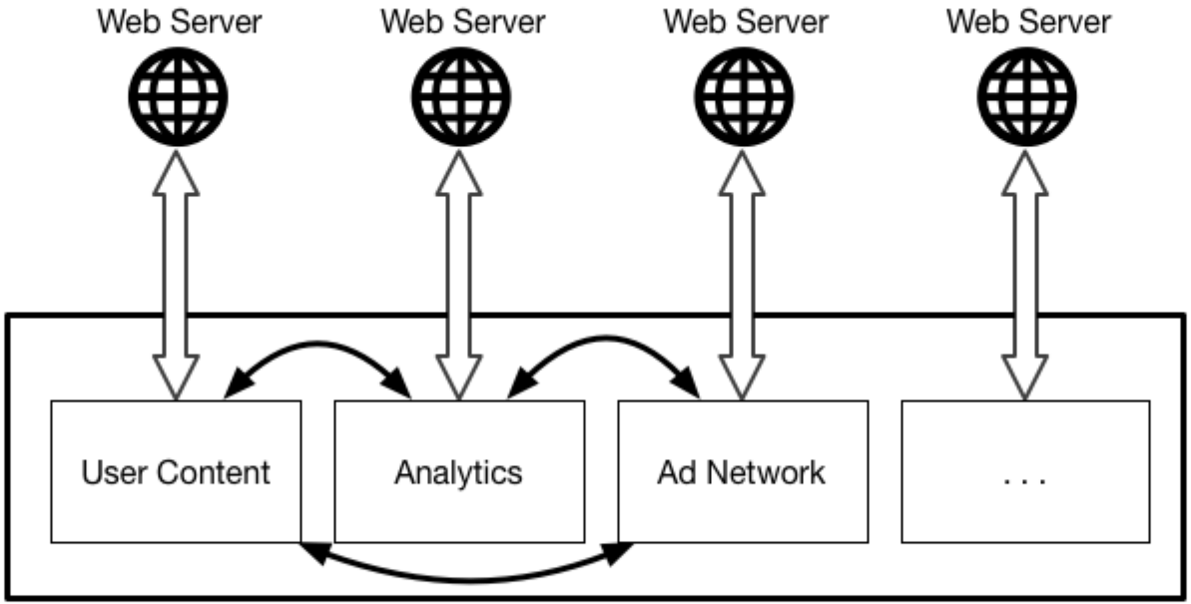
XMLHTTPRequest (XHR) and postMessage



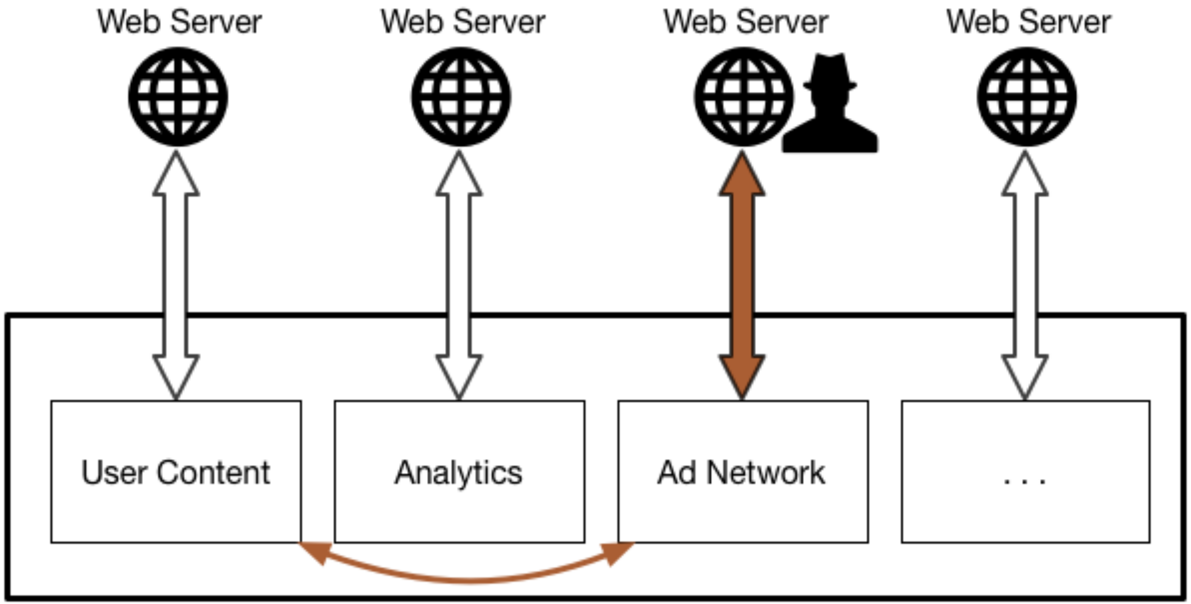
XMLHTTPRequest (XHR) and postMessage



XMLHTTPRequest (XHR) and postMessage



XMLHTTPRequest (XHR) and postMessage



Client-Side Validation Vulnerabilities

- Bugs in client-side code of web application (JavaScript)
- Unsafe usage of untrusted data
- Validation
  - missing
  - broken

Client-Side Validation Vulnerabilities

- Attacks
  - Origin mis-attribution
  - Command injection
  - Cookie leakage
- Defense / Detection
  - Server-side detection not possible
  - ⇨ Client-side

Example of CSV Vulnerability

```
function listener(event){
    if (event.origin.indexOf("domain-a.ru") != -1){
        eval(event.data);
    }
}

if (window.addEventListener){
    window.addEventListener("message", listener,false);
} else {
    window.attachEvent("onmessage", listener);
}
```

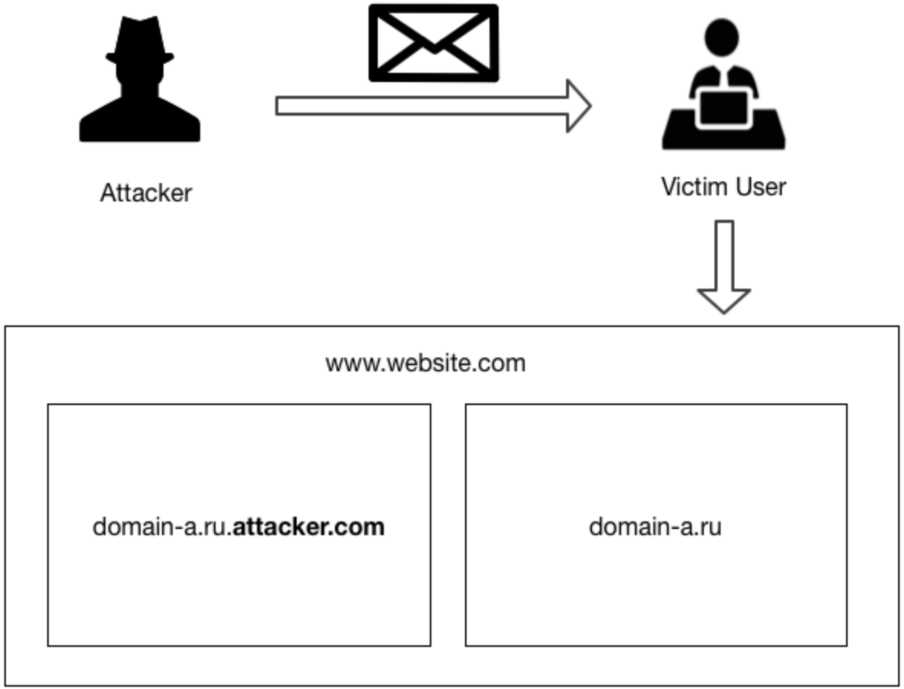
Example of CSV Vulnerability

```
function listener(event){
    if (event.origin.indexOf("domain-a.ru") != -1){
        eval(event.data);
    }
}
```

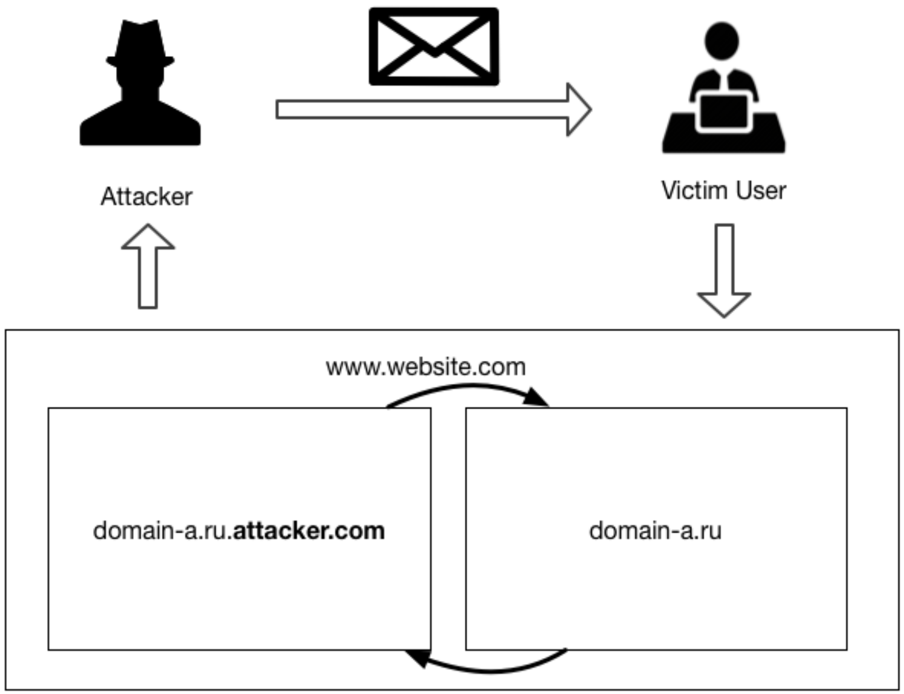
- Developer must verify origin correctly
- Counterexample: `"domain-a.ru.attacker.com"`

Example Attack

Example Attack



Example Attack



Goal: Secure JavaScript Apps

- Hardening benign-but-buggy applications
- Fully automatic: no developer interaction
- Detection / defense in browser alone
    - No browser modifications or extensions
- Handle unknown vulnerabilities

Related Work

- CSV has been shown to be prevalent [1,2]
- 84 out of top 10,000 websites vulnerable [1]
- Bug finding systems
  - FLAX [2], Kudzu [3]
- Defense system
  - Changes to sites [1], standards [1]

[1] The Postman Always Rings Twice: Attacking and Defending postMessage in HTML5 Websites. Son et al., 2013

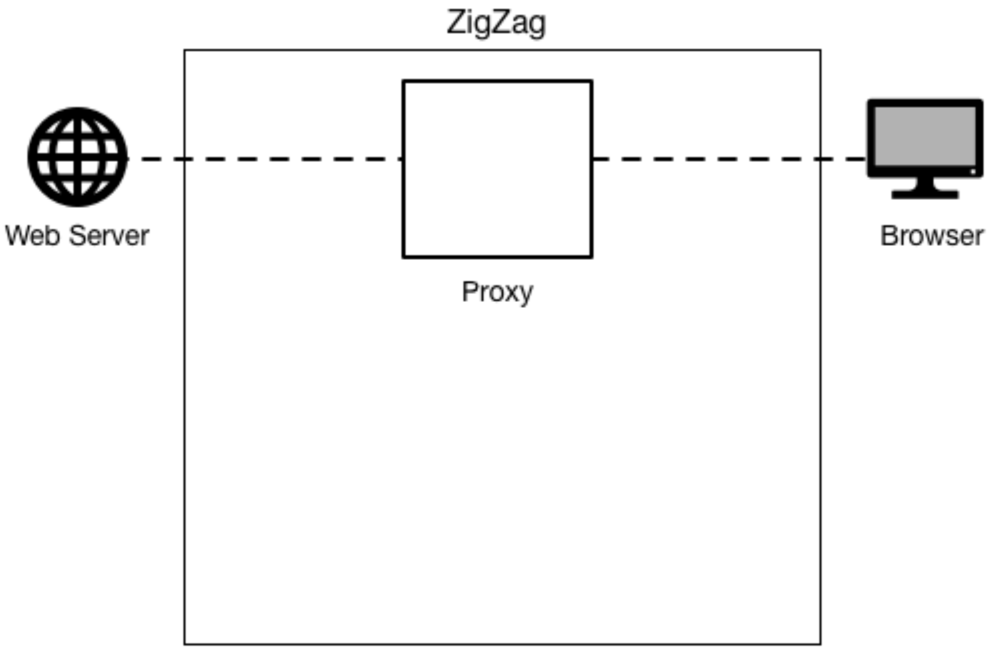
[2] FLAX: Systematic Discovery of Client-side Validation Vulnerabilities in Rich Web Applications. Saxena et al., 2010

[3] A Symbolic Execution Framework for JavaScript. Saxena et al., 2010

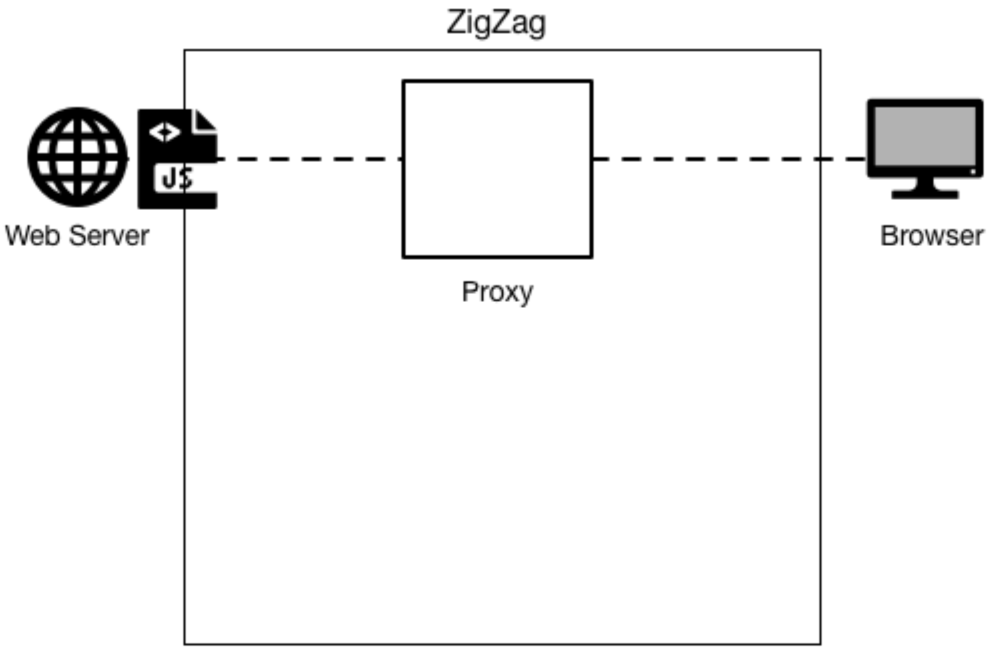
ZigZag Overview

- Anomaly detection system preventing CSV attacks
- Instrumentation of client-side JavaScript
- Generates model of benign behavior
- Two phases
  - Learning of benign behavior
  - Prevent attacks

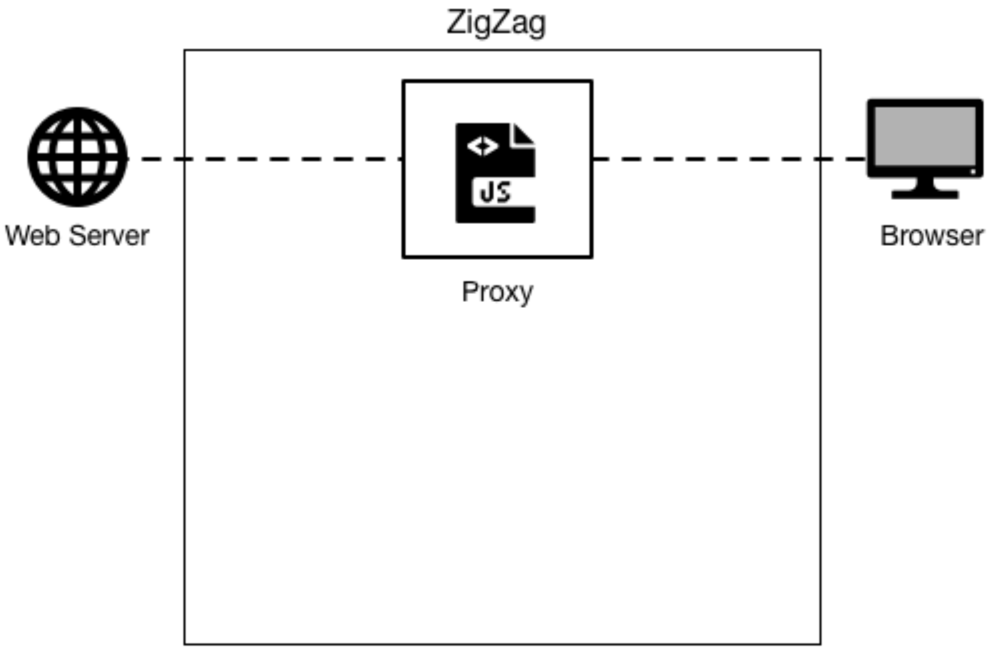
Learning Phase



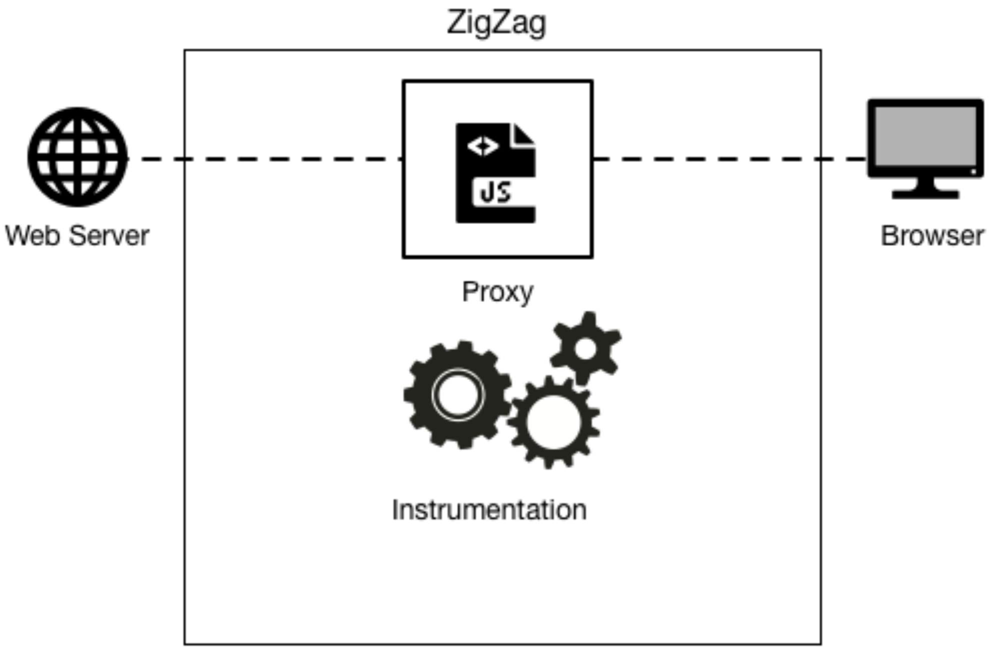
Learning Phase



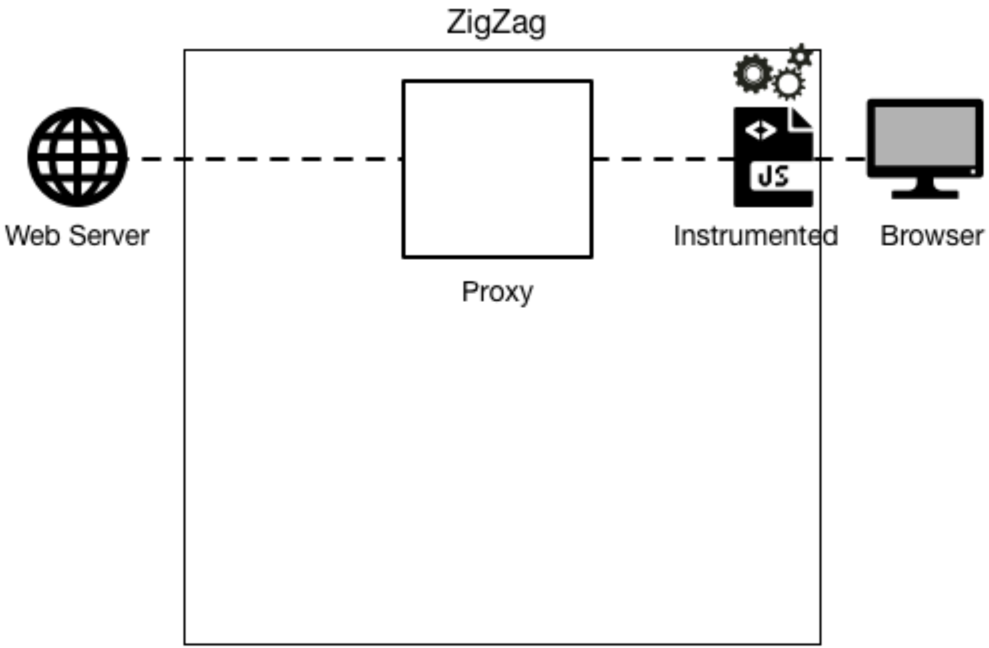
Learning Phase



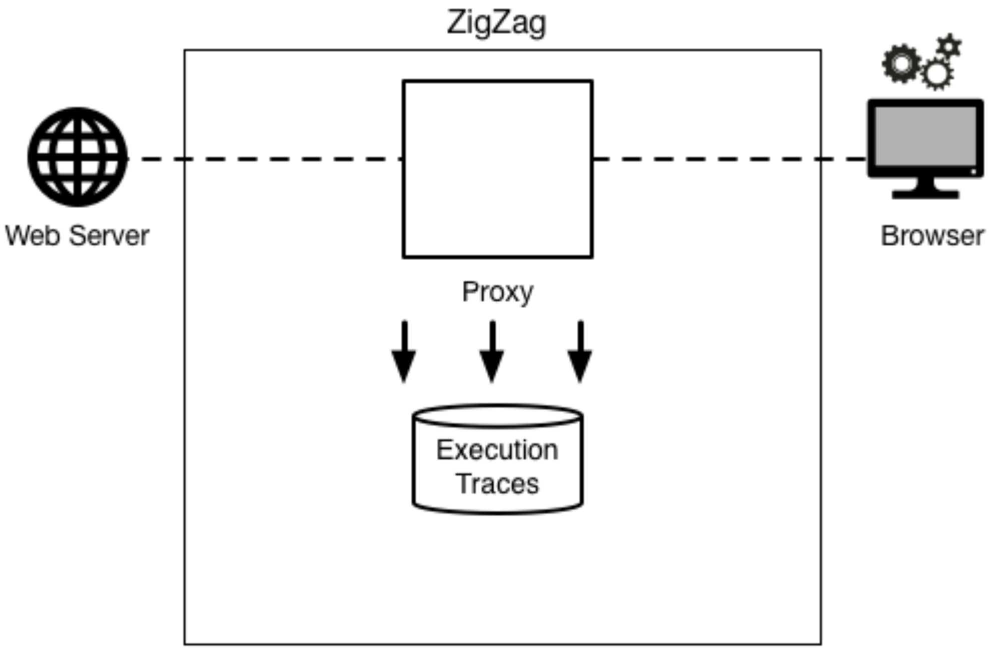
Learning Phase



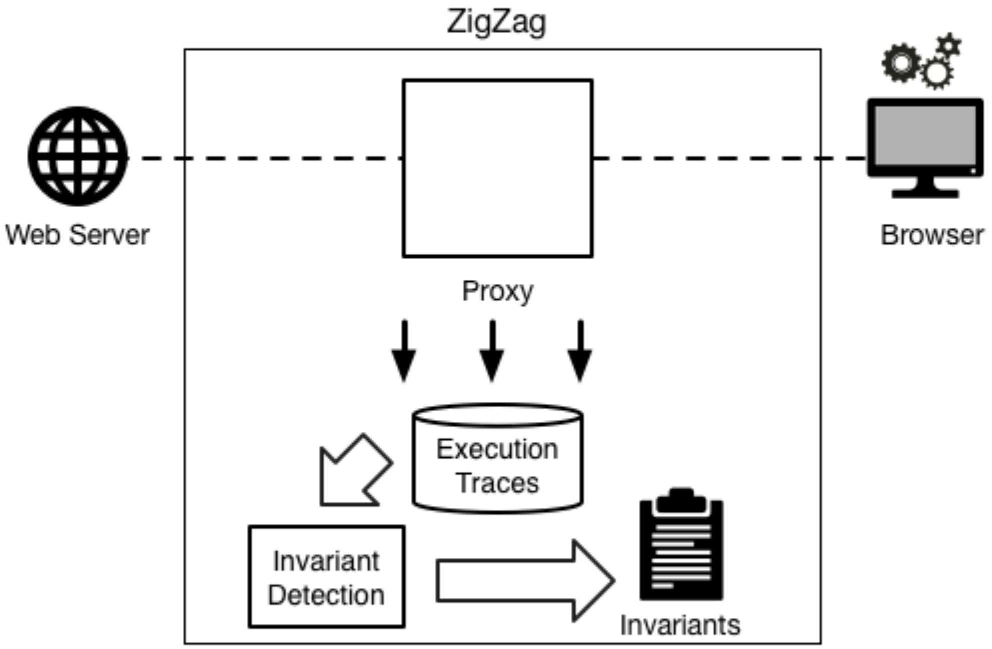
Learning Phase



Learning Phase



Learning Phase



Background: Program Invariants

- Assertions over variables at program points
  - `x == y`
  - `x + 5 == y`
  - `x < y`
- Dynamic detection
  - Statistical likelihood
  - DAIKON [1]

[1] The Daikon System for Dynamic Detection of Likely Invariants. Ernst et al.

Supported Invariants

Data Types	Invariants
All	Types
Numbers	Equality, inequality, oneOf
String	Length, equality, oneOf, isPrintable, **isJSON, isEmail, isURL, isNumber**
Boolean	Equality
Objects	All of the above for object properties
Functions	**Calling function**, return value

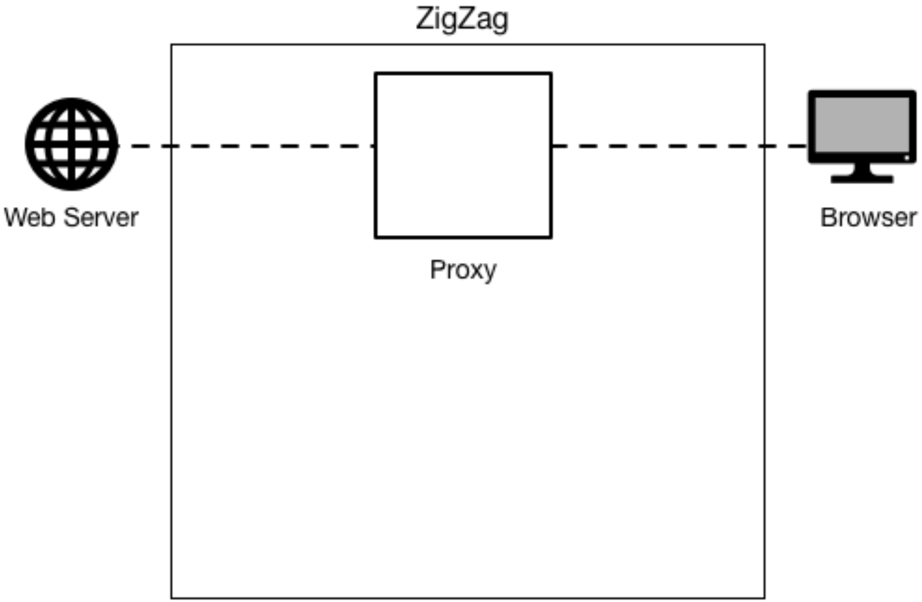
Generated Invariants

- Function is only invoked from the global scope

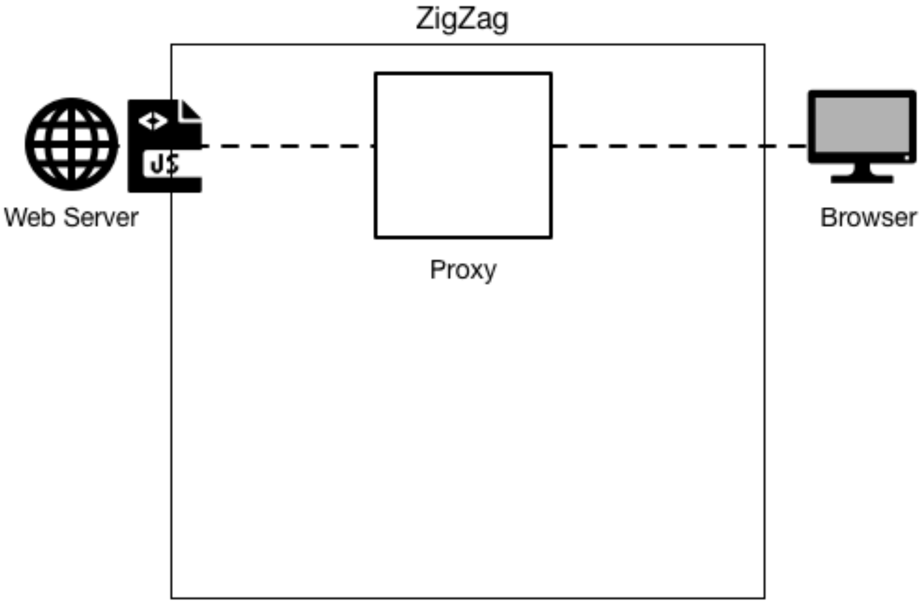
- `typeof(v0) === 'object'`
  `&& v0.origin === 'http://domain-a.ru'`

- `v0.data === "$('#right_buttons').hide();"`
  `|| v0.data === 'calculator()'`

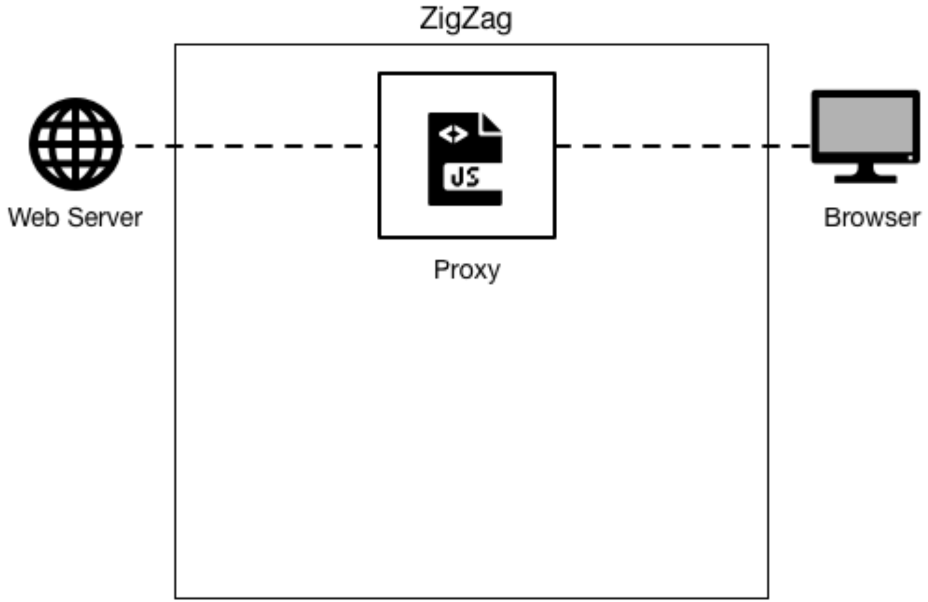
Enforcement Phase



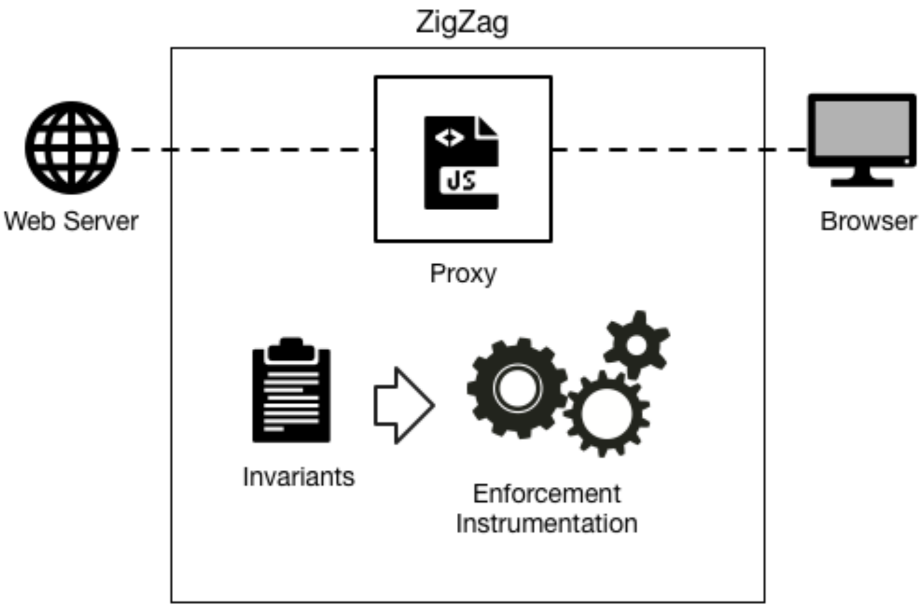
Enforcement Phase



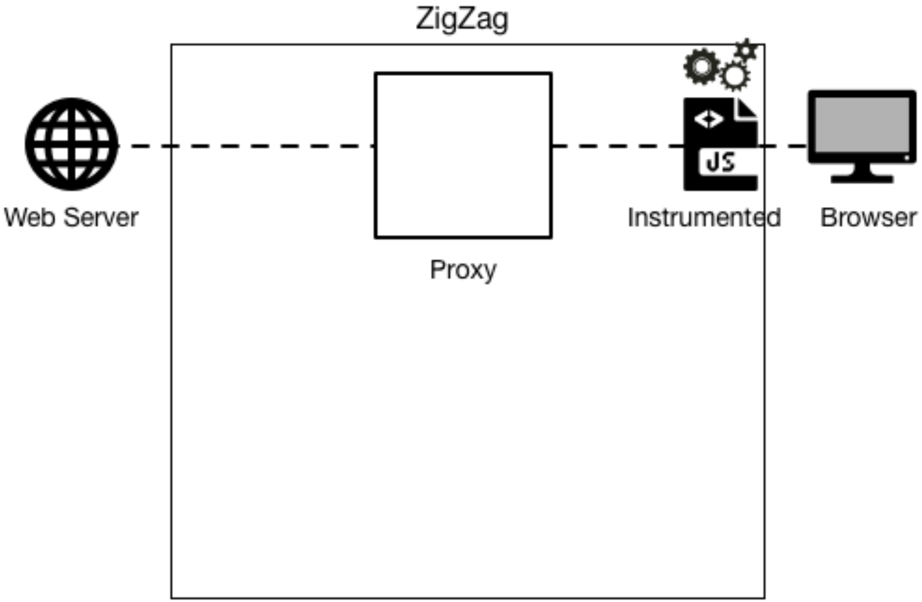
Enforcement Phase



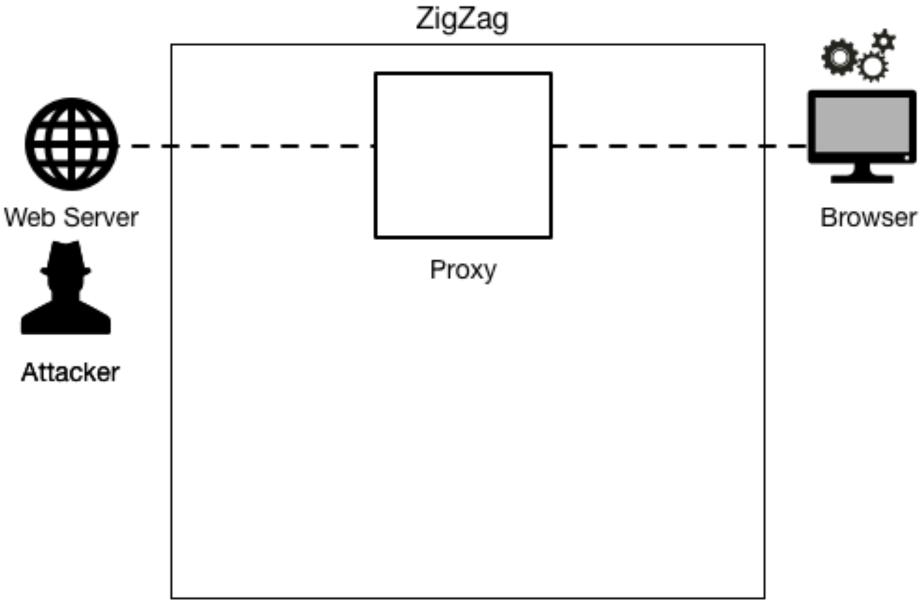
Enforcement Phase



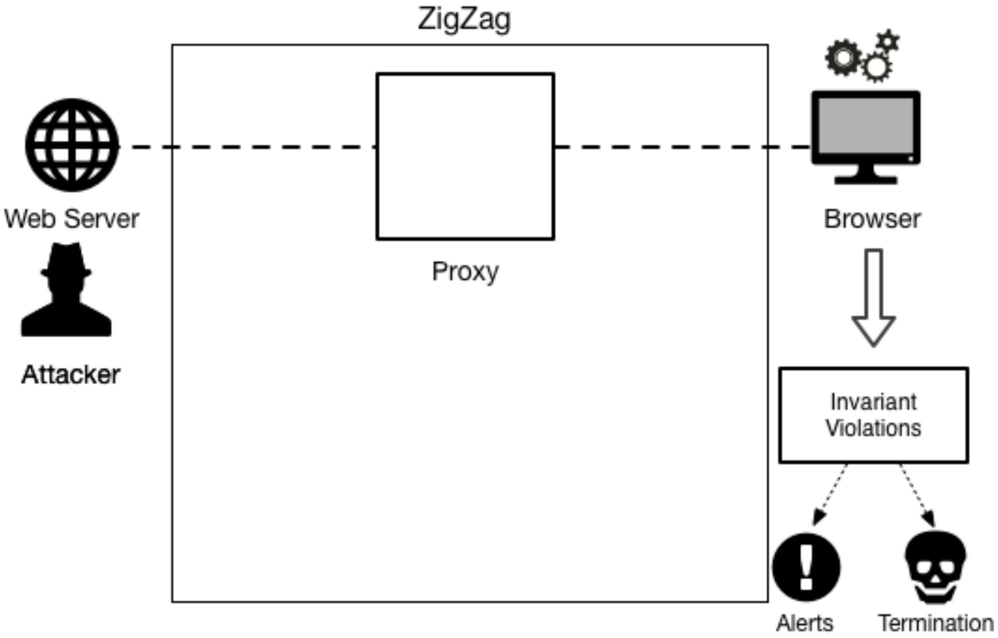
Enforcement Phase



Enforcement Phase



Enforcement Phase



Enforcement Details

Original

```
function x(a, b) {
  c = a + b;
  return c;
}
```

Instrumented

```
function x(a, b) {
  var callcounter = __calltrace(functionid, codeid, sessionid);
  c = a + b;
  return __exittrace(functionid, callcounter, subexitid, codeid, sessionid, c);
}
```

Enforcement Details

```
__calltrace = function(functionid, codeid, sessionid) {
    var v0 = arguments.callee.caller.caller.arguments[0];
    var v1 = ...
    if ( functionid === 0 ) {
        __assert(typeof(v0) === 'object' && v0.origin === 'http://domain-a.ru' );
        __assert(v0.data === "$('#right_buttons').hide();" ||
                 v0.data === 'calculator()' );
        ...
    } else if ( functionid === 1 ) {
        ...
    }
    ...
    return __incCallCounter();
}
```

Deployment Options

- Transparent proxy
  - Deploy at gateway
  - Some latency
- On-site rewriting
  - One-time instrumentation
  - Protects all users
- Invariant sharing
  - Proxy setup

Challenges

- In-Browser web app runtime performance
  - Instrumentation of functions that are not security relevant
- Server-side code templates
  - New code can be generated for individual page visits

Targeted Instrumentation

- What functions are relevant to be instrumented?

- Lightweight static analysis to refine instrumentation
  - document, window, ...
  - XHR, eval, postMessage, …
  - ⇨ Output: list of functions to be instrumented

- Benefits: increased performance at runtime

Server-side JavaScript Templates

- Parameterize code with:
  - Configuration
  - Username
  - Timestamps, etc.

- Programs differ, but <u>AST is similar</u>

- Singleton training sets: Anomaly detection not possible

Server-side JavaScript Templates

- Parameterize code with:
  - Configuration
  - U
  - Ti

Q: How to generate invariants over classes of programs?

- Programs differ, but AST is similar
- Singleton training sets: Anomaly detection not possible

Server-side JavaScript Templates

```
// Server-side JavaScript template
var state = {
    user: {{username}},
    session: {{sessionid}}
};


// Client-side JavaScript code after template instantiation
```

```
var state = {
    user: "UserX",
    session: 0
};
```

```
var state = {
    user: "UserY",
    session: 1
};
```

```
var state = {
    session: 2,
    user: "UserZ",
};
```

Server-side JavaScript Templates

```
// Server-side JavaScript template
var state = {
    user: {{username}},
    session: {{sessionid}}
};


// Client-side JavaScript code after template instantiation
```

```
var state = {
    user: "UserX",
    session: 0
};
```

```
var state = {
    user: "UserY",
    session: 1
};
```

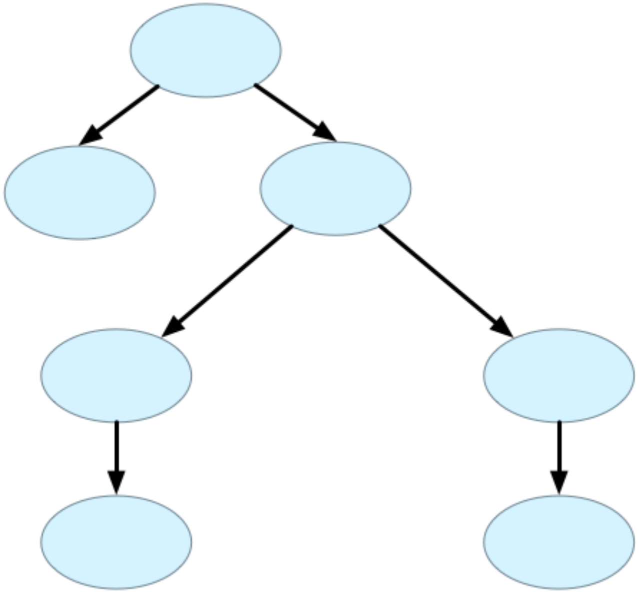
```
var state = {
    session: 2,
    user: "UserZ"
};
```

Program Generalization

- When is code similar?
  - Values of primitives differ
  - Object properties differ
- AST can differ due to:
  - Object properties omitted
  - Order of properties
- Goal:
  - Structural comparison
  - Fast instrumentation

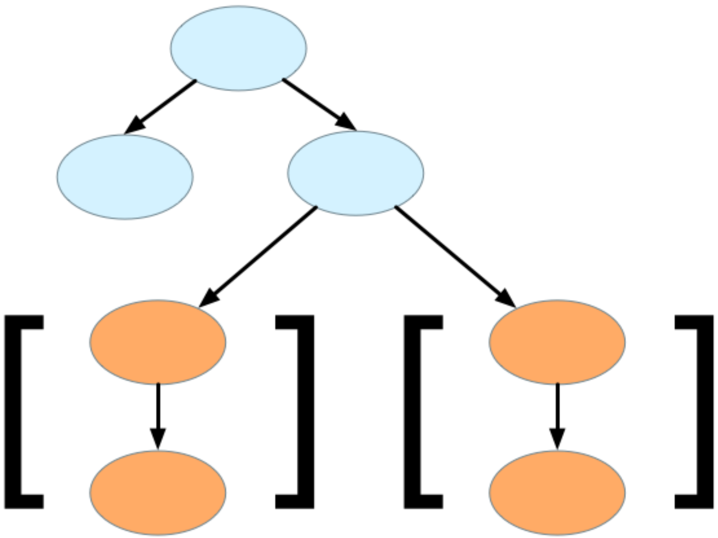
Structural Comparison

Uninstrumented program - extract abstract syntax tree



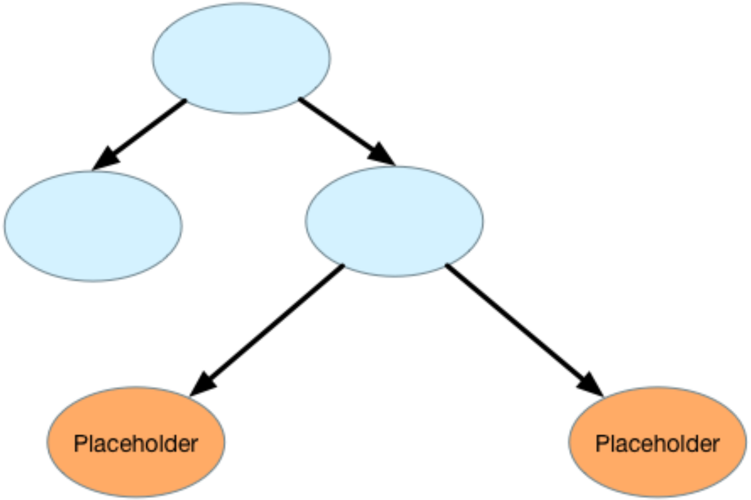
Structural Comparison

Detect program points for generalization



Structural Comparison

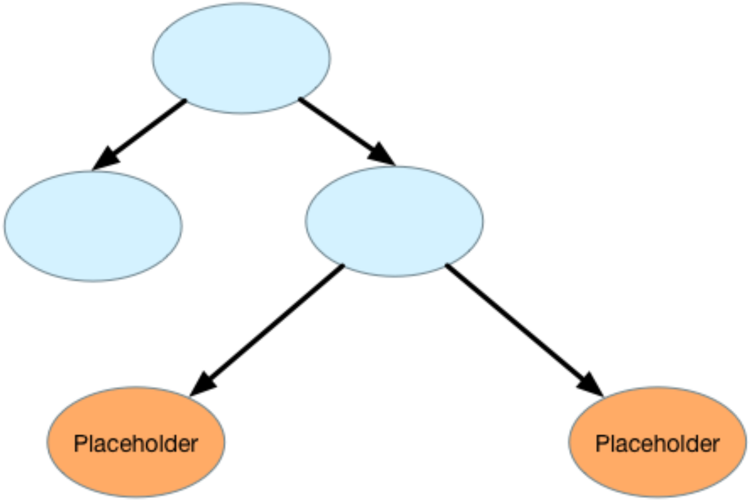
Generalize: add placeholders



Structural Comparison

Generalize: add placeholders

Extract canonical string representation of AST for matching



Program Generalization

- Generalize program by:
  - Removing primitive values
  - Remove primitive object properties
  - Order object properties

- Instrument generalized version

Program Generalization (ctd.)

- When new program is detected as specialized version:
  - Differences from the template are detected
  - Generated patch-set is applied to instrumented version

- Result:
  - Anomaly detection extended from same to similar programs
  - Performance close to cached instrumentation

Evaluation: Security Benefits

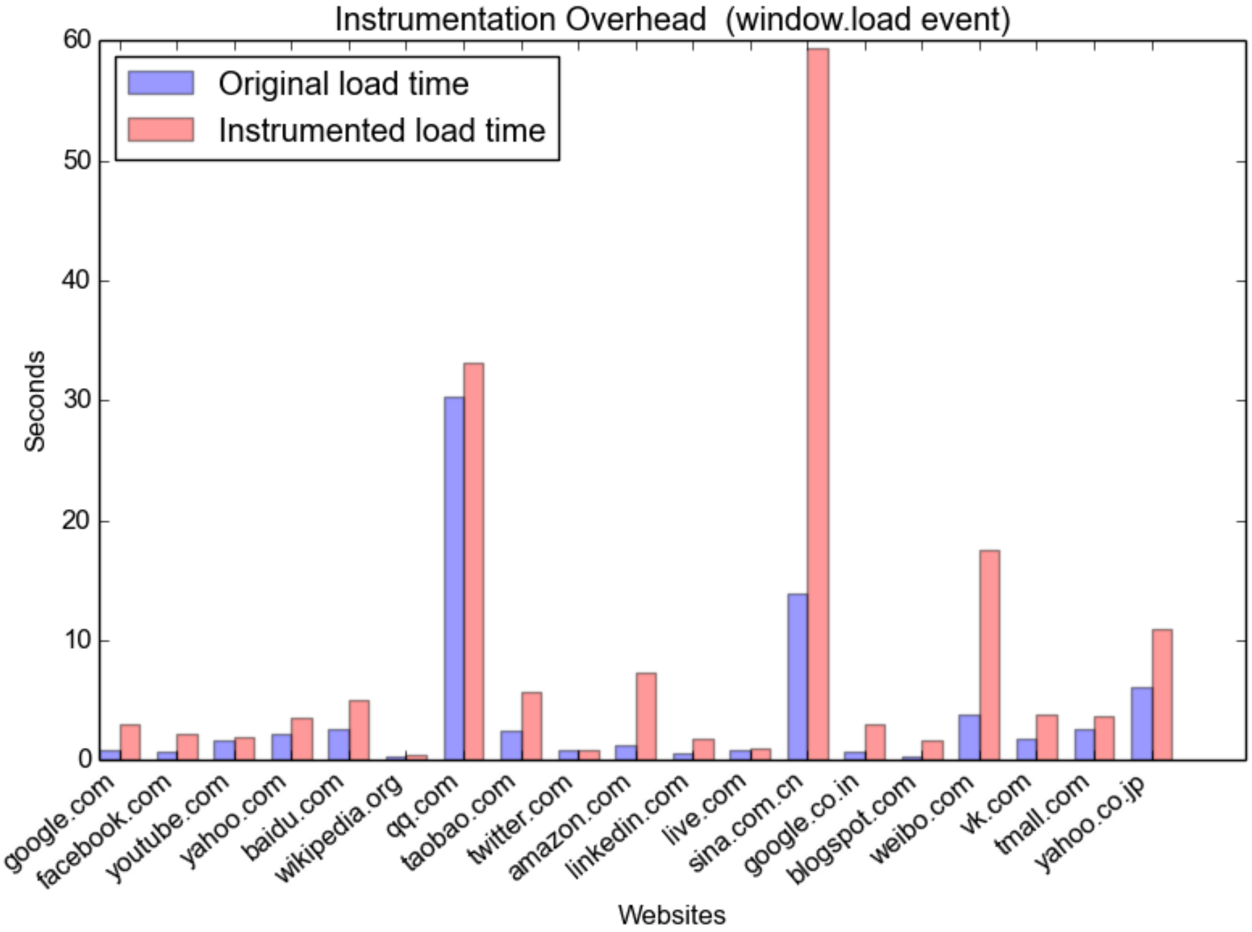
- Four real-world vulnerable sites
- Synthetic webmail application

- Attacks caught
- Functionality retained

Evaluation: Performance

- Alexa Top 20
  - Median overhead: 2.01s (112%)
  - No false positives

- Microbenchmark: 0.66ms overhead


Instrumentation Overhead (window.load event)
Original load time
Instrumented load time
Seconds
0
10
20
30
40
50
60
google.com
facebook.com
youtube.com
yahoo.com
baidu.com
wikipedia.org
qq.com
taobao.com
twitter.com
amazon.com
linkedin.com
live.com
sina.com.cn
google.co.in
blogspot.com
weibo.com
vk.com
tmall.com
yahoo.co.jp
Websites

Contributions

- In-browser anomaly detection system for hardening against previously unknown CSV vulnerabilities

- Invariant patching: extending invariant detection to server-side templates

Conclusions

- CSV attack prevention through invariants
- Real-world obstacles
  - Server-side template
  - Function whitelisting
- ZigZag
  - Effective defense
  - Enforcement in browser only
  - Moderate performance overhead

Thank you for your attention

@mweissbacher