How is Proto being Probed? The Experimental Aspects behind the Large-scale Measurement of Client-Side Prototype Pollution Vulnerabilities

Zifeng Kang, Song Li, and Yinzhi Cao
Department of Computer Science, Johns Hopkins University
Roadmap

• Introduction
  • What are prototype pollution and its consequences?
  • How do we detect them? What is the System design?
• Implementation
• Evaluation
• Discussion
• Wrap-up
Introduction

• What is Prototype Pollution?
  • A relatively-new JavaScript vulnerability type discovered in 2018
  • Polluting a base object’s property, e.g., Object.prototype.toString

• Related Work
  • [ESEC/FSE’21], [USENIX’22]
  • Issues: (1) consequence is unclear, and (2) server-side apps only

• What are Consequences?
  • Further vulnerability (damages) caused by Prototype Pollution
  • Examples: Cross-site Scripting (XSS) and Cookie/URL manipulation
Design: Intuition

• Idea: Joint Taint Flow Analysis

Adversary-controlled Inputs

```javascript
?__proto__[k]=<script>alert('Exploited')</script>
```

```javascript
for (; M <= N; M++) {
    P = R[M] === "" ? O.length : R[M];
```

```javascript
}
data = {'123': 'abc'};
for (var field in data) {
    unitSpecs.append("<li><span class=" + field + ">" + data[field] + "</span></li>");
```
Design: Intuition

Prototype Pollution Sink 1: O = O[P]
Prototype Pollution Sink 2: O[P] = J

Adversary-controlled Inputs

Data flow I

Object taint flow

Data flow II

Data flow III

Joint Taint Flow:

Data Flow I to III & Object Taint Flow

XSS Sink: $unitSpecs.append( ...+ data[field] + ... );
Design: System Architecture

Joint Taint Flow Analysis

Input/Exploit Generator

Dynamic Taint Engine

n times

Flows

Exploits

Result Validation

Exploit Validation

Result Validation

Defense Analysis

URL

Report
Roadmap

- Introduction
- Implementation
  - What software tools do we use to implement ProbetheProto?
  - What challenges have we met when deploying it on real-world websites?
- Evaluation
- Discussion
- Wrap-up
Implementation: Choices of Programming Languages

Joint Taint Flow Analysis

Input/Exploit Generator

Dynamic Taint Engine

Exploits

Flows

Result Validation

Exploit Validation

Defense Analysis

Report

URL

C/C++
n times

Python, C/C++

JavaScript

Melicher et al.
Chromium, V8 engine
Experience with deploying

• Getting Chromium to run
  • Got a Google link from Melicher et al. for their Chromium-based system
  • Deploying Ubuntu 14 and other dependencies for the old-version Chromium

• Modifying v8 engine
  • Using gdb to debug v8
  • Searching for lines of interest, e.g., v8/src/object.h, v8/src/runtime/runtime-object.cc, etc.
  • Compilation takes too long: Use the incremental building!
Problems with crawling

• Crawler choice: Python or Chrome extension?
  • Old version Chromium: no proper chromedriver found.
  • How to control the browser: through bash scripts.

• Crawler settings: choosing the parameters.
  • How many instances running in parallel?
  • Running multiple windows or running multiple tabs in one window?
  • What is the timeout for each page and for each website?
Runtime Incidents when crawling

• Links that download files will stop all instances.
  • Solution: filter the links.
  • Should periodically check the crawler status manually.
  • Should set checkpoints for the crawler to continue.

• Cache/Memory is full: Causes the browser to crash.
  • Periodically clear the cache/memory.
  • Also, remove the useless config files of Chromium.
• Introduction
• Implementation
• Evaluation
  • What are the experiment settings and evaluation results for each of our RQ?
  • What are the intermediate/unsuccessful results and what did we do to improve them?
• Discussion
• Wrap-up
Roadmap for Evaluation

• I. Measurement Results
• II. Comparison
• III. Performance
• IV. False Negatives
• V. Code Coverage
• VI. Defense
Measurement Settings

• Target: top one million Tranco websites.
• Server details: 192 GB memory and Intel® Xeon® E5-2690 v4 2.6GHz CPU.
• Time period: from November 12th, 2021 until December 3rd, 2021 for three weeks in total.
• Crawler parameters: 20 instances running in parallel and a 120-second timeout for each website.
Measurement Results

• Zero-Day vulnerabilities
  • Total: 2,917 out of one million
  • Fixed: 240
  • Consequence breakdown
• Vulnerable domain examples

<table>
<thead>
<tr>
<th>Consequences</th>
<th># Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSS</td>
<td>48</td>
</tr>
<tr>
<td>Cookie manipulations</td>
<td>736</td>
</tr>
<tr>
<td>URL manipulations</td>
<td>830</td>
</tr>
<tr>
<td>No observable consequence</td>
<td>1,595</td>
</tr>
<tr>
<td>Total</td>
<td>2,917</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain</th>
<th>Ranking</th>
<th>Status</th>
<th>Exploits</th>
</tr>
</thead>
<tbody>
<tr>
<td>weebly.com</td>
<td>96</td>
<td>Reported</td>
<td><a href="https://www.weebly.com/domains?__proto__%5B1%5D=v">https://www.weebly.com/domains?__proto__[1]=v</a></td>
</tr>
</tbody>
</table>
Breakdown by Sources/Sinks

<table>
<thead>
<tr>
<th>Consequences</th>
<th>Sink</th>
<th># Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSS</td>
<td>innerHTML</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>append</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>eval</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>setAttribute</td>
<td>31</td>
</tr>
<tr>
<td>Cookie Manipulation</td>
<td>Arbitrary</td>
<td>666</td>
</tr>
<tr>
<td></td>
<td>Specific</td>
<td>95</td>
</tr>
<tr>
<td>URL Manipulation</td>
<td>anchor</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td>iframe</td>
<td>205</td>
</tr>
<tr>
<td></td>
<td>img</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>script</td>
<td>192</td>
</tr>
<tr>
<td>Total of Above Three</td>
<td>-</td>
<td>1,322</td>
</tr>
</tbody>
</table>
## Intermediate Results

<table>
<thead>
<tr>
<th>Consequences</th>
<th># Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSS</td>
<td>3</td>
</tr>
<tr>
<td>Cookie manipulations</td>
<td>132</td>
</tr>
<tr>
<td>URL manipulations</td>
<td>253</td>
</tr>
<tr>
<td>No observable consequence</td>
<td>313</td>
</tr>
<tr>
<td>Total</td>
<td>591</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Vulnerabilities</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>48</td>
</tr>
<tr>
<td>1</td>
<td>736</td>
</tr>
<tr>
<td>2</td>
<td>830</td>
</tr>
<tr>
<td>12</td>
<td>1,595</td>
</tr>
<tr>
<td>Total</td>
<td>2,917</td>
</tr>
</tbody>
</table>
How did we improve the results?

• Removing false positives: Design the result validation module.
  • Validate both prototype pollution exploits and consequence exploits.
  • Follow the standard validation steps for prototype pollution, to avoid any false positives.

• Uncovering more vulnerabilities: Improve the Input/Exploit Generator.
  • Apply various input formats.
  • E.g., nested array lookup: \( k0[k1][k2]=v \)
  • And different delimiters: \( k0=v0&k1=v1&k2=v2 \)
Responsible disclosure

• Search for email addresses
  • Developed an information retrieval tool based on regular expressions
  • Search on whois record and their own websites
• Problem: half not found or invalid!
• Solution: We manually inspect over 1,000 websites to find out how to reach out to them and send the reports automatically.
• We allow 45 days as the responsible disclosure window.
Roadmap for Evaluation

• I. Measurement Results
• II. Comparison
• III. Performance
• IV. False Negatives
• V. Code Coverage
• VI. Defense
Comparison with Prior Works

- Problem: No prior works measuring client-side prototype pollution and its consequences!
- Solution: We modify a state-of-the-art server-side detection tool, called ObjLupAnsys, to support client side and then compare our system with it.
- We added client-side sources, e.g., location and document.cookie, to ObjLupAnsys to make it better fit the client-side applications.
Comparison Results

• Two experiments: (i) Top 30 thousand websites; (ii) 2,738 vulnerable websites found by our system.

• (i) ObjLupAnsys only reports one website which turns out to be a false positive.

• (ii) ObjLupAnsys only detects four websites out of 2,738.

• ProbetheProto **significantly outperforms** ObjLupAnsys.
Roadmap for Evaluation

• I. Measurement Results
• II. Comparison
• III. Performance
• IV. False Negatives
• V. Code Coverage
• VI. Defense
Performance Overhead Improvements

- Reasonable overhead now: **38.6%** compared with legacy Chromium.
- Intermediate results: over **200%** overhead compared with legacy Chromium.
- How did we improve that?

![Graph showing page loading time and percentage of domains](image-url)
Improving Performance Overhead

- Make sure our implementation is optimized.
  - The object taint bit is a previously unused one.
  - No additional memory is involved.
  - The codes for input/exploit generation is efficient.

- Remove unnecessary functionalities in Melicher et al.’s taint tracking engine.
  - Change configurations to a light-weight version.
  - E.g., set is_debug flag to false.
  - Release memory for information important to their paper but unnecessary to ours.
Roadmap for Evaluation

• I. Measurement Results
• II. Comparison
• III. Performance
• IV. False Negatives
• V. Code Coverage
• VI. Defense
False Negative Results

- Experiment settings: a manually-annotated benchmark from a Github repository.
  - (a) scripts with prototype pollution vulnerabilities
  - (b) scripts that are vulnerable to XSS if a prototype pollution is present.
- Results: 9.5% FNs for prototype pollution, 20.9% for XSS consequences.

<table>
<thead>
<tr>
<th>Vulnerabilities</th>
<th>TP</th>
<th>FN</th>
<th>Total</th>
<th>TPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype Pollution</td>
<td>19</td>
<td>2</td>
<td>21</td>
<td>90.5%</td>
</tr>
<tr>
<td>XSS Consequences</td>
<td>34</td>
<td>9</td>
<td>43</td>
<td>79.1%</td>
</tr>
</tbody>
</table>
Improving False Negatives

• Intermediate results: 80% FNs for XSS detection.
  • Thinking from the exploit formats …
  • `__proto__[k1][k2] = <script>alert('Exploited')</script>`
  • Solution: Provide a rich list of possible XSS exploits to the Input/Exploit Generator.
  • We also run Joint Taint Flow Analysis for multiple iterations to generate multiple parameters in nested object lookups, each iteration responsible for one parameter in each bracket.
Roadmap for Evaluation

• I. Measurement Results
• II. Comparison
• III. Performance
• IV. False Negatives
• V. Code Coverage
• VI. Defense
Code Coverage Results

- Tools: Google Chrome’s DevTools
- Metrics: the ratio between used and total bytes of the target vulnerable JavaScript file(s).
- Dataset: (i) all of the 43 files with XSS consequences in the Github dataset; (ii) 50 random real-world websites that are vulnerable to prototype pollution.
Roadmap for Evaluation

• I. Measurement Results
• II. Comparison
• III. Performance
• IV. False Negatives
• V. Code Coverage
• VI. Defense
## Defense Analysis Results

<table>
<thead>
<tr>
<th>Defense</th>
<th>Technique</th>
<th># Joint Flows</th>
<th># Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-flow</td>
<td>Property sanitization</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Object sanitization</td>
<td>22,235</td>
<td>1,489</td>
</tr>
<tr>
<td>Control-flow</td>
<td>Property white/blacklist</td>
<td>2,710</td>
<td>124</td>
</tr>
</tbody>
</table>
High-Level Idea of Defense Analysis

• Control variable experiments: two runs.
  • One with normal inputs;
  • The other with generated exploit inputs.
  • Data flow changes $\rightarrow$ Defense!

• Data flow unchanged but data contents differ?
  • The contents are altered by a defense.
  • Category: data-flow defense.

• Data flow changed? (Taint flow disappeared)
  • The flows are altered by a defense.
  • Category: control-flow defense.
Learning from Case Study (I)

• Case study gives us hints about defense categories in real-world websites.
  • Example: facebook.com (property sanitization, a sub-category of data-flow defense).

    // property sanitization
    // convert a from "__proto__" to "\ud83d\udf56"
    function i(a) {

        return a === "__proto__" ? "\ud83d\udf56": a

    }
Learning from Case Study (II)

• Case study gives us hints about defense categories in real-world websites.
  • Example: kiev.kupikupon.com.ua (control-flow defense).

```javascript
// a property whitelist for control-flow defense
function (i, e) {
  var n = { "utmz": {} }, s = n[i];
  if ("utmz" === i) {
    /* When i="__proto__", this code block will not be executed. */
    ... }
  }
```
Case studies are powerful!

- Different sources that trigger prototype pollution
  - holocaust.cz, for Message sources
- Consequene category collection
  - 247sports.com, for cookie manipulation
- Defense analysis category collection
  - facebook.com, for data-flow defense and control-flow defense
Roadmap

• Introduction
• Implementation
• Evaluation
• Discussion
• Wrap-up
Discussion

• Did you use experimentation artifacts borrowed from the community?
  • Yes.
  • The dynamic taint engine by Melicher et al.
  • The prior detection tool by Song et al.
  • Google Chrome DevTools.
Discussion

• Did you attempt to replicate or reproduce results of earlier research as part of your work?
  • Yes.
  • Performance overhead by Melicher et al.
  • Measurement results of ObjLupAnsys by Song et al.
Discussion

• What can be learned from your methodology and your experience using your methodology?
  • Go over each part of the system and/or the whole working process to find which ones are causing unsuccessful results.
  • Learn from the case studies when there are unexpected results.
  • Control variables during experiment to get reliable evaluation results.
Discussion

• Did you produce any intermediate results including possible unsuccessful tests or experiments?
  • Yes.
  • Unsuccessful results including unreliable measurement results, high overhead, and high false negatives.
  • Eventually, we improved all of those results.
Roadmap

• Introduction
• Implementation
• Evaluation
• Discussion
• Wrap-up
Wrap-up

- ProbetheProto is the **first** large-scale measurement of client-side prototype pollution and further consequences.
- ProbetheProto discovers 2,917 zero-day, exploitable vulnerabilities: 48 leading to XSS, 736 cookie manipulations, and 830 URL manipulations.
- We have learnt lessons when we improve the intermediate/unsuccessful results, such as conducting case studies and control-variable experiments.
Thank you. Questions?