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

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

```
for (; M <=N; M++) {

  P = R[M] === "" ? O.length : R[M];

  O = O[P] = M < N ? O[P] || (R[M + 1] && isNaN(R[M + 1]) ? {} : []) : J

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

Design: Intuition

Adversary-controlled Inputs

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

Data flow I

Prototype Pollution Sink 1: O = O[P]

Data flow II

Data flow III

Object taint flow

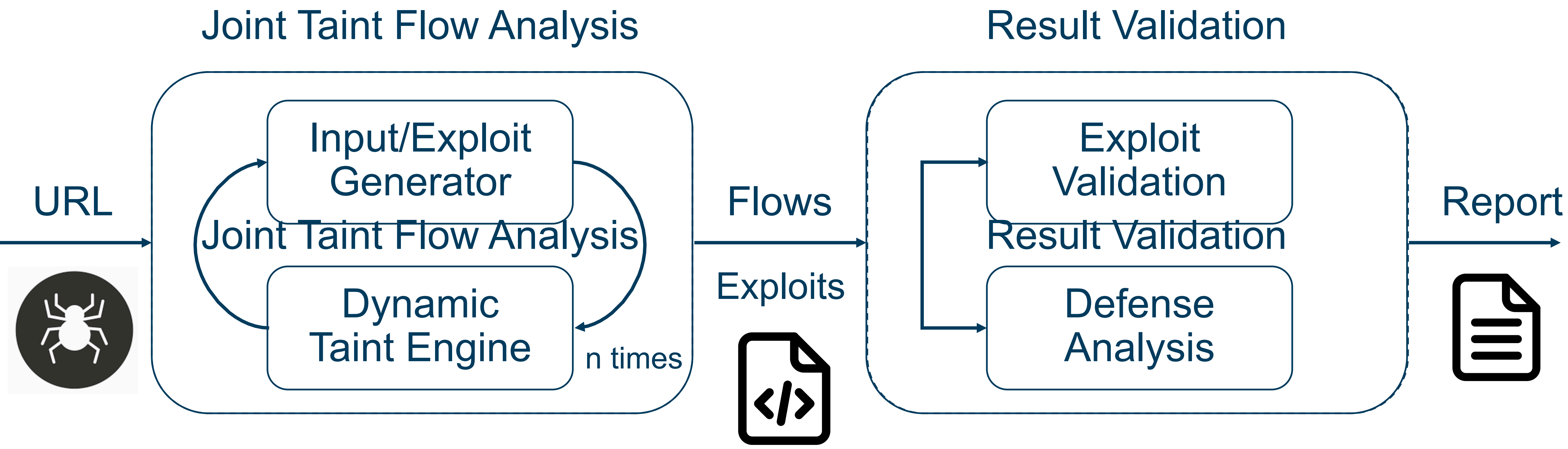
Prototype Pollution Sink 2: O[P] = J

XSS Sink: $unitSpecs.append(…+ data[field] + …);

Joint Taint Flow:

Data Flow I to III & Object Taint Flow

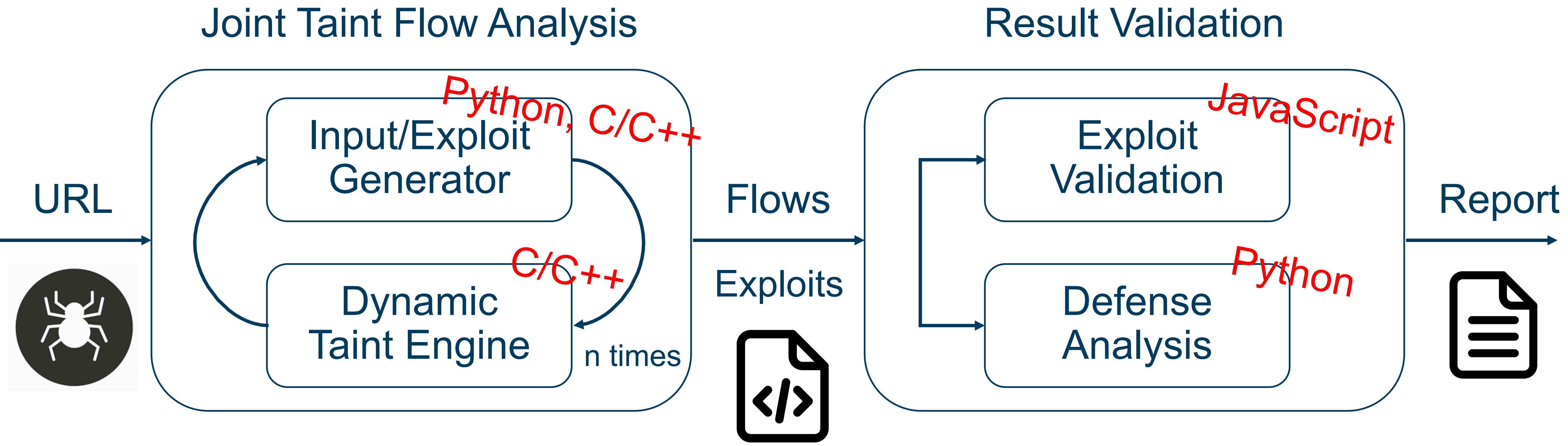
Design: System Architecture



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



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.

Roadmap

- 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

Consequences	# Vulnerabilities
XSS	48
Cookie manipulations	736
URL manipulations	830
No observable consequence	1,595
Total	2,917

Domain	Ranking	Status	Exploits
weebly.com	96	Reported	https://www.weebly.com/domains?__proto__[1]=v
cnet.com	150	Fixed	https://www.cnet.com/?constructor[prototype][1]=v
mckinsey.com	693	Fixed	https://www.mckinsey.com/?__proto__[k]=v

Breakdown by Sources/Sinks

Consequences	Sink	# Vulnerabilities
XSS	innerHTML	10
	append	4
	eval	3
	setAttribute	31
Cookie Manipulation	Arbitrary	666
	Specific	95
URL Manipulation	anchor	152
	iframe	205
	img	500
	script	192
Total of Above Three	-	1,322

Intermediate Results

Consequences	# Vulnerabilities
XSS	3
Cookie manipulations	132
URL manipulations	253
No observable consequence	313
Total	591

Vulnerabilities
48
736
830
1,595
2,917

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

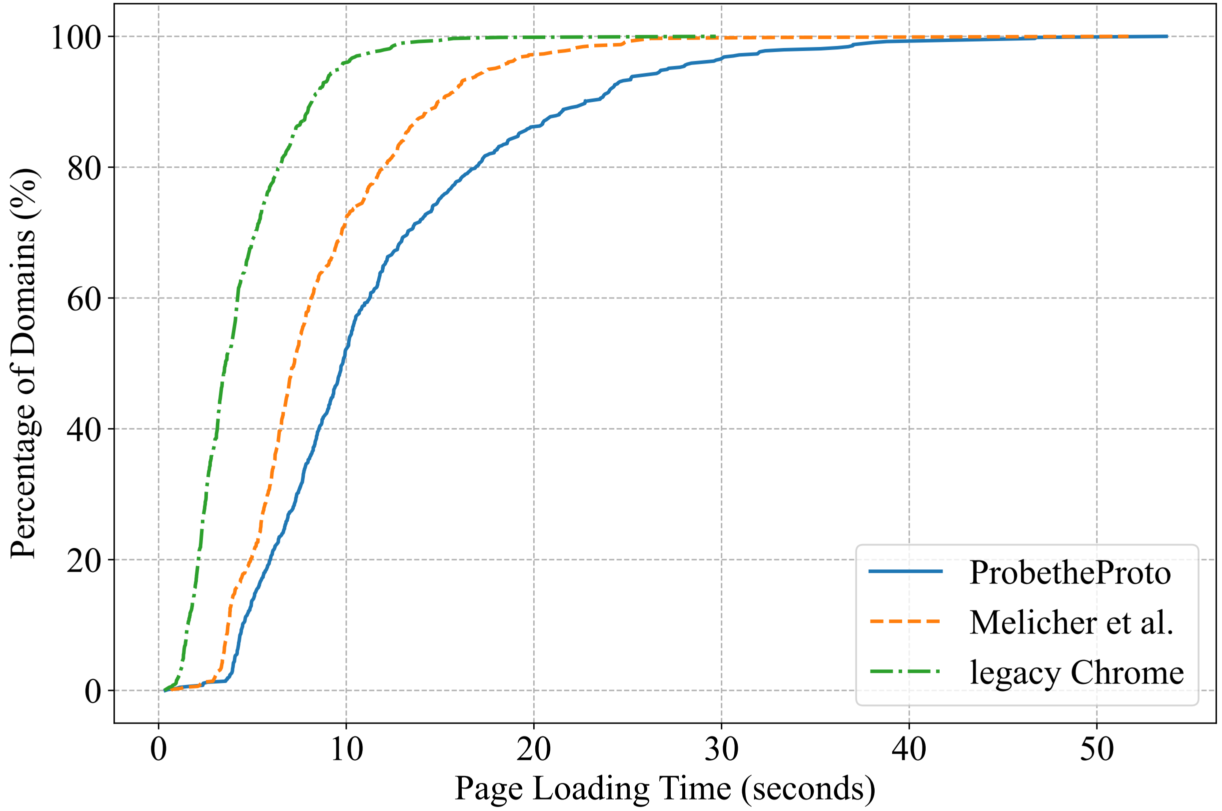
- Two experiments: (i) Top 30 thousand websites; (ii) 2,738 vulnerable websites found by our system.
- (i) ObjLupAnsys only reports one website which turns out to be a false positive.
- (ii) ObjLupAnsys only detects four websites out of 2,738.
- ProbetheProto **significantly outperforms** ObjLupAnsys.

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?



Intermediate result: >200%

38.6%

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.

Vulnerabilities	TP	FN	Total	TPR
Prototype Pollution	19	2	21	90.5%
XSS Consequences	34	9	43	79.1%

Improving False Negatives

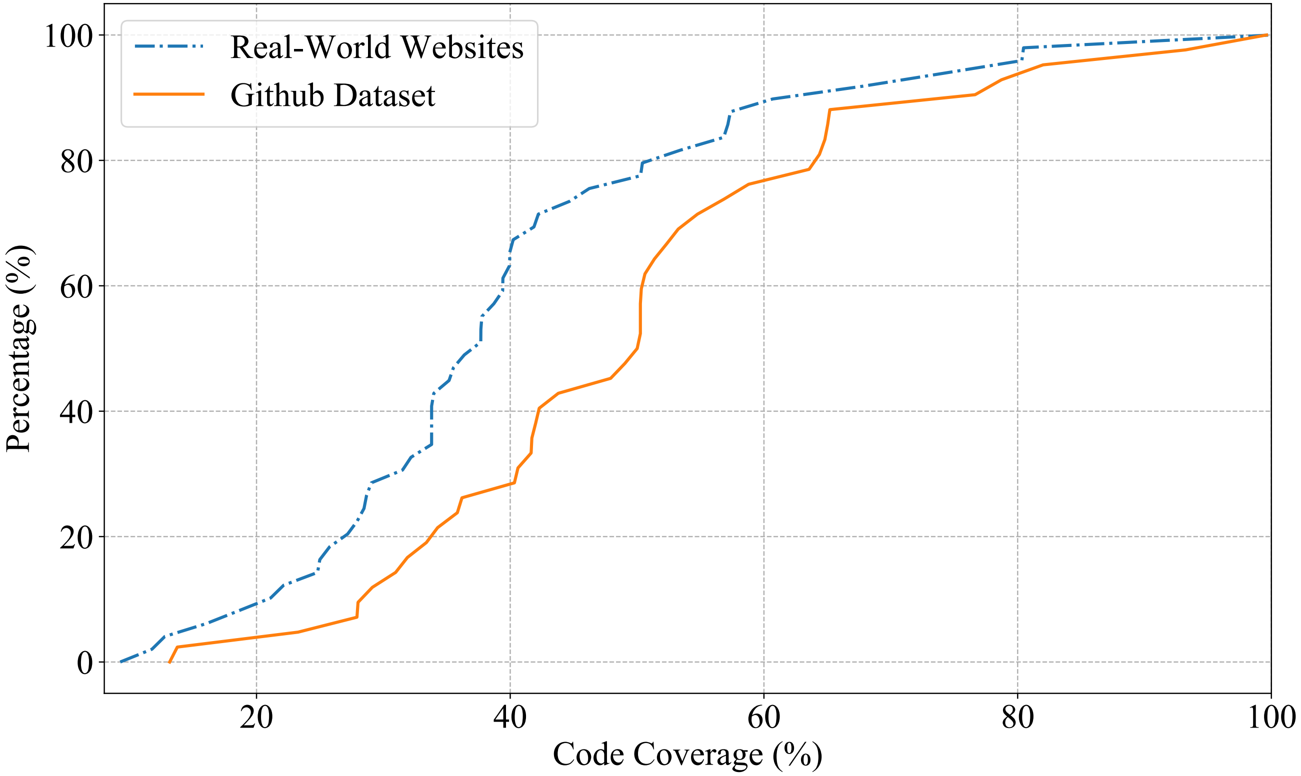
- Intermediate results: 80% FNs for XSS detection.
  - Thinking from the exploit formats …
  - __proto__[k1] [k2]=<script>alert('Exploited')</script>
  - Solution: Provide a rich list of possible XSS exploits to the Input/Exploit Generator.
  - We also run Joint Taint Flow Analysis for multiple iterations 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: Googl
- Metrics: the … target vulnerable J…
- Dataset: (i) a … in the Github dataset; (ii) 5 … ulnerable to prototype po…



CDF of code coverage

CDF of code coverage increase for input/exploit generation

Roadmap for Evaluation

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

Defense Analysis Results

Defense	Technique	# Joint Flows	# Domains
Data-flow	Property sanitization	15	6
	Object sanitization	22,235	1,489
Control-flow	Property white/blacklist	2,710	124

High-Level Idea of Defense Analysis

- Control variable experiments: two runs.
  - One with normal inputs;
  - The other with generated exploit inputs.
  - Data flow changes → 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).

```
// 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
- Consequece 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?

- ProbetheProto repo: https://github.com/client-pp/ProbetheProto.
- A list of vulnerable websites: https://github.com/clientpp/ProbetheProto/blob/main/vul_site_info.md.