#235 PTV: Better Version Detection of JavaScript Web Libraries Based on Unique Subtree Mining

Email notification
Select to receive email on updates to reviews and comments.

▼ PC Conflicts
Wei Yang

Submitted

Submission (1.2MB) Mar 22, 2024, 6:09:05 PM UTC · 61ad672b

▼ Abstract
Identifying the versions of libraries used by a web page is an important task for sales intelligence, website profiling, and even security analysis, enabling more fine-grained web analysis. Recent work uses tree structure to represent the property relationships of the library at runtime, leading to more accurate library identification. However, current tree-based detection methods cannot be directly used to detect specific versions of libraries. This paper proposes a novel algorithm to find the most unique structure out of each tree in a forest so that the size of the trees can be greatly minimized. We show that an implementation of our algorithm in a state-of-the-art library detection tool, not only guarantees the soundness of detection result but reduces associated costs where tree-based detection methods can be used to detect library versions. Our experiment results on over 500 real-world libraries with 30,000 unique versions show that our tool reduces space requirements by up to 99% and achieves more precise version detection compared with existing tools.

▶ Authors (anonymous)
X. Liu, L. Ziarek [details]

▶ Topics, attachments, and options

	OveMer	ArtAss
Review #235A	2	
Review #235B	3	
Review #235C	2	3

You are an **author** of this submission.

Edit submission Add Cycle-1-Rebuttal response

Reviews in plain text

Review #235A

Overall merit

2. Weak reject

Paper summary

Knowing which library versions are included in a program is a necessity for assesing the program's security. Previous work, `PTdetector` [3] used dependency tree structures to quickly determine JavaScript libraries, but would require large amounts of storage space to also determine each version. This paper introduces `PTV`, an optimized version of `PTdetector` in which only subtrees unique to a specific version are identified, stored, and compared, greatly reducing storage needs.

Strengths

- Novel identification of library versions
- Looking for unique subtrees is a unique and novel contribution
- Significant reduction in storage space

Weaknesses

- Requires re-training for every new library version
- Requires *all* library versions to be available
- Not good at detecting privately fixed libraries

Detailed comments for authors

This is a neat extension of existing work, `PTdetector`, effectively enabling the approach not only to detect which libraries are being used, but also which version of the library is being used. The idea to use unique `pTree` subtrees as version detectors is original and apparently effective.

My main gripe with this work is that *forks* of library versions would not be detected accurately – a common problem in bill-of-material approaches. If I as a vendor, for instance, notice or get notified that there is a flaw in, say, `library-1.2.3`, and I do not want to wait for the release of `library-`

`1.2.4`, then I could easily clone the library into a new "privately fixed" version, say `library-1.2.3a`. Then,

- a scanner going for version labels would detect the new version;
- a scanner `library-1.2.3a` against `library-1.2.3` would at least not falsely match it to `library-1.2.3`;
- `PTV`, however, *would* likely (falsely) match it to `library-1.2.3`.

Checking only for an abstraction such as `pTree` might miss the change already (depending on whether my fix introduces new dependencies); checking for unique subtrees, as in this paper, would likely miss my fixes (unless the fixes affect the unique subtree, or the fixed version gets included in the next scan). The authors should discuss this problem, and possibly extend their evaluation to assess the risk of private fixes.

Novelty

Using unique `pTree` subtrees for matching versions is original and efficient.

Rigor

The evaluation is sound, but does not take into account the risk of private fixes (see above), where competitors might have an edge.

Relevance

This is mostly an optimization / extension of the earlier `PTdetector` work, which will be hard to generalize beyond the concrete problem of identifying JavaScript library versions - but which, in turn, is a big problem.

Verifiability and Transparency

The tool is available for users already.

Presentation

The paper has a number of typographical problems, which also exist in [3]:

- It's `Listing 1`, not `Listing. 1`
- References: It's `JavaScript`, `NPM`, `COTS`, and (ahem) `PTdetector`, not `javascript`, `npm`, `cots`, or `ptdetector`.
- References: Who is "C. E. Store"? Is this the Chrome Extension Store?

Some parts such as the `LDC` description (§2B.1) are plagiarized verbatim from [3]. I assume this is a cse of self-plagiarism.

Questions for authors' response

1. Would `PTV` detect private fixes to libraries, in contrast to related work?

2. How would you evaluiate the above?

Review #235B

Overall merit

3. Weak accept

Paper summary

The paper presents PTdetector-Version (PTV), an extension of a dynamic JavaScript library detector known as PTDetector, where PTV makes PTDetector library version-aware. PTDetector relies on a data structure, referred to as pTree, which is built from the property tree of a running JavaScript web page. In a library collection step, a feature dataset (i.e., a set of pTrees) is curated from running the libraries' files in a "dummy client", while the library detection for a given web page then boils down to a similarity-based tree matching of the extracted pTree against the pTrees in the feature dataset. A limitation of PTDetector is that, for a given library, it only constructs the pTree for the latest library version. On the contrary, PTV constructs a forest of pTrees, where each pTree represents a dedicated version. To deal with this blowing-up of the feature datset, the paper at hand introduces a new algorithm designed to minimize the number of trees employed in library detection. The central concept involves isolating the most distinctive structure from each tree within the forest, which decreases the data stored and utilized during library detection. PTV has been tested against current library detection methods (including but not limited to PTDetector) using a substantial amount of real-world libraries and web pages. Results demonstrate that PTV effectively deals with the memory requirements of the increaset feature dataset without compromising its accuracy of detecting library versions.

Strengths

- Relevant problem that is even tackled by commerical tools
- Rigoroulsy engineered algorithm for minimizing PTDetector's feature dataset, thus making its library detection version-aware
- Broad evaluation of the new PTV approach yielding promising results

Weaknesses

- It is not entirely clear how to interprete the results of RQ3
- No discussion of threats to validity of experimental evaluation

Detailed comments for authors

Novelty

The technical novelty provided by PTV compared to PTDetector is the new algorithm to minimize pTrees used for library (version) detection, along with its implementation and experimental evaluation.

Rigor

The algorithm is motivated by an analysis of the technical challenges of PTV, with the intuitions of how to overcome these challenges being turned into an algorithm design in a rigorous manner. This includes correctness proofs, theoretical complexity analysis, and an algorithm implementation used for experimentation. The paper is sound from a technical point of view.

As for the experimental evaluation, the high-level setup for evaluating PTV is reasonable. I fully agree with the evaluation strategy assessing the minimization capabilities of PTV (RQ1), and the results are convincing. As for the detection accuracy on real-world web pages (RQ2), the authors have chosen a reasonable set of tools that constitute a proper baseline. In my view, focusing on the consistency of the all the tools' results is a reasonable strategy here.

A more debatable aspect of the evaluation is the interpretation of the results of RQ3. It is not entirely clear to me whether running through all the historic versions of a library constitutes cases that are representative scenarios that occur in practice, or whether this setup is rather artificially created for the sake of evaluating PTV's potential. Both is fine, but it needs to be discussed.

A crucial concern with the experimental evaluation in general is that it lacks a discussion of potential threats to validity.

Relevance

While the improvement over state-of-the-art seems to be moderate for real-world scenarios (RQ2), RQ3 demonstrates the full potential of PTV. Moreover, the qualitative discussion of selected cases provide interesting insights. Indeed, there is potential to improve the state-of-the-art in tackling a relevant problem. One of the tools used as baseline in the evaluation is a commercial tool, which is remarkable.

Verifiability and Transparency

While the high-level presentation in terms of the paper is already clear and concise in most parts, the paper includes a reference to an anonymized GitHub repo for all details. The given information seems to be sufficient to fully replicate the results of the paper, and to independently conduct further experiments.

Presentation

The paper is generally well written and easy to follow. The only aspect that puzzled me when first reading the strategy for RQ2 was that all the tools may report a range of versions instead of a single dedicated one. This could be made more explicit already in the technical part of the paper, i.e., when describing the background and PTV in detail.

Review Summary

In sum, though technical novelty is a bit incremental, the paper presents a well-designed algorithm including its theoretical analysis and practical implementation on top of PTDetector. The version-

awareness and the library detection itself is a relevant problem, with the paper showing some well-argued indicators and experimental results for potential impact.

Questions for authors' response

1. It is not entirely clear to me how to interprete the results of RQ3. Is this setup rather artificially created for the sake of evaluating PTV's potential?

2. More generally, do you see any potential threats affecting the validity of the experimental results and the drawn conclusions?

Review #235C

Overall merit

2. Weak reject

Paper summary

In this paper, the authors proposed a set of algorithms for detecting versions of JavaScript libraries. The main idea is to leverage pTrees generated by PTDetector to encode the object property relationships in the JavaScript source code. The authors further attempted to minimize the size of pTrees by extracting the unique subtrees, and matched subtrees against known minimum subtrees of JavaScript libraries to determine the version. Experiment results proved the proposed tool, PTV, is effective in minimizing the pTree and can pinpoint the correct library versions in most cases.

Strengths

- Targeted a relevant and critical problem
- Detailed illustration of the design and algorithms
- Effective in minimizing the memory overhead

Weaknesses

- Certain comparative experiments read unfair
- Lack of evaluation on the capability of identifying the exact version
- Noticeable overhead
- Limited discussion about the practical usage and real impact

Detailed comments for authors

In this work, the authors targeted a very important problem that is highly relevant to the SE community. I also admire that the authors provided a very clear and detailed description of the definitions, designs, and algorithms.

Regarding the current submission, however, I also found several issues. Please find my detailed comments below.

- Novelty

The library syntactic features are mainly encoded using an existing tool PTDetector. Although the authors also made certain modifications to PTDetector, e.g., encoding set MD5, the novelty mainly lies in the minimum subtree extraction. From my perspective, the novelty is a bit limited.

- Rigor

1) On P5, the authors mentioned both the unique subtrees of T1 in Figure 4 are not subtrees of any tree in {T4, T5}. I wonder why the first subtree is not a subtree of T4, also, why is not a tree containing path 1-2-4 and 1-2-5 be a unique subtree of T1.

2) On P8, the authors mentioned their extended PTDetector identified almost twice different libraries compared with existing tools. However, I suppose the reason would be the extended tool is equipped with much more known libraries and theoretically be able to detect 556 libraries. Therefore, this comparison (in Table IV) looks unfair to me. Compared with existing tools, the ratio of detected_libraries / detectable_libraries is much lower. It would be good to provide an explanation of the possible reasons.

3) According to Table V and P8, the three examined tools generated inconsistent results in 19 cases. The authors claimed that PTV produced sound results in all the inconsistent occurrences. However, it reads unclear to me how many results are returned by PTV. For instance, if in most cases PTV pinpoints a wide range of possible versions (i.e., they cannot be readily distinguished by the pTrees), the practical impact would be diminished, as substantial additional efforts are still required to figure out the version number. It would be good if the authors could add a brief description here.

4) According to the evaluation results, PTV exhibits a much higher overhead compared with the existing tools, mostly due to the need to collect the path records. The overhead would even increase when the total number of detectable libraries increases. When incorporating new libraries or versions, the unique subtrees may also need to be updated, introducing additional overhead. Therefore, the overall cost reads noticeably high to me.

5) Another issue is about the practical use. As also discussed in the paper, library version detection is critical for security analysis tasks, e.g., vulnerability detection. However, no evaluation or discussion about it was provided. Nonetheless, I would not treat it as a major limitation of this work, as the main focus is library version detection algorithms.

- Relevance

JavaScript libraries are very prevalently used in web application development, and the version detection is critical and highly relevant to the SE community.

- Verifiability & Transparency

This paper provides detailed description about the dataset and implementation. The innovation and evaluation results are presented clearly and can be verified.

- Presentation

1) P4, "Our goal is to find a minimum ... that satisfies replacing ... will not change the algorithm output" -> "a minimum ... so that replacing ... will not ..."

2) P7, "we removed those are not designed to" -> "... those that are not ..."

3) P6 mentioned the inner and outer dependency, without providing any details or definition. Although the authors refer interested readers to [3], it would be good to also provide a brief description.

4) P6, "not only have no impact on the accuracy ... provide more information to allow us ..." reads contradictory. I suppose it should be "but also provide superfluous information".

Questions for authors' response

- Does PTV frequently return multiple possible versions as the detection results? How many versions can be distinguished by the pTrees?
- In Fig 4, why the first subtree is not a subtree of T4? Why is not a tree containing path 1-2-4 and 1-2-5 be a unique subtree of T1?
- In Table IV, the ratio of detected_libraries / detectable_libraries achieved by PTV seems lower than existing tools. What could be the reasons?

Artifact assessment

3. Satisfactory, i.e., the artifacts are in line with what is declared in the submission form or the paper [OR] the authors explained why the artifacts are not provided and I find the explanation to be reasonable.

Comments on artifact assessment

The authors have released the source code and provided usage guidance in a README file.