An Extensible Framework for Web Application Vulnerabilities Visualization and Analysis

Tran Tri Dang and Tran Khanh Dang

Faculty of Computer Science and Engineering,
Ho Chi Minh City University of Technology, VNU-HCM, Vietnam
{tridang,khanh}@cse.hcmut.edu.vn

Abstract. The popularity of web-based applications makes them interesting targets of cyber attacks. To deal with that threat, discovering existing vulnerabilities is a proactive step. Although there are many web application scanners designed for this task, they lack visual analysis capability and do not collaborate well together. In this paper, we propose a novel visualization technique and a flexible framework to solve the two problems mentioned above. We also develop a prototype based on the proposal and use it to experiment with virtual websites. Experiment results indicate the unique benefits our work offers. But more importantly, it shows that not only improving the visualization technique from a technical viewpoint is needed, but also improving it from a human cognitive viewpoint should be placed at a higher priority.

Keywords: Web vulnerability visualization, security visualization, web application security analysis.

1 Introduction

The widespread use of web applications in different areas makes them interesting targets for attackers. In fact, the 2014 Internet Security Threat Report from Symantec highlights that web-based attacks are up 23% and 1 in 8 legitimate websites has a critical vulnerability [1]. Furthermore, the ease of web development, together with the wide availability of programming libraries, also contributes to a large degree of poor quality web applications written by amateur programmers.

To overcome the above problem, many approaches are proposed by the security researchers and practitioners. Among them, one promising approach is to develop scanning tools that are able to discover the web vulnerabilities automatically. The advantage of these tools is undeniable: they can save a lot of security analysts' time and effort in securing web applications. However, there are some limitations that these tools cannot solve alone. The first limitation is that the report interface of these scanners is not very effective in communicating scanning results to security analysts. In our survey of some selected tools, we find that the generated reports are either too detailed or too general, and it is not easy switching from one level of abstraction to another level without losing the perception of the previous view. The second limitation is that although these tools have the same goal, it is difficult to use them together to complement each other. There is a reason behind the need to make them work together seamlessly: each tool has its own strengths and weaknesses, and by combining them properly, we can amplify the strengths as well as reduce the weaknesses.

In this paper, we describe our work in solving the 2 limitations of web application vulnerability scanning tools mentioned above. For the first limitation, we propose a novel visualization that can display scanning results aggregated at many levels of detail on a same screen at a same time. In addition, the visualization is enhanced by user interaction techniques to provide more specific information. For the second limitation, we design an extensible framework to transform scanning results of different tools into a common format that can be used later by the visualization component. We also implement a prototype of this framework and use it to demonstrate the benefits our work can provide to security analysts.

The rest of this paper is structured as follow: in Section 2, we review the related works; we describe the framework architecture in Section 3; Section 4 is used to detail the visualization design and user interaction techniques; the implementation details and its test results are explained in Section 5; and finally, we use Section 6 to conclude the paper and suggest some future works.

2 Related Works

Generally speaking, there are two approaches that scanning tools used to look for vulnerabilities in web applications: white-box testing and black-box testing. In white-box testing, a web application is assessed on how well it handles user inputs, data validation, sensitive information communication, database queries, etc. based on existing source code, compiled code, or bytecode [2-5]. As a result, white-box testing tools are language-specific and platform-specific. Therefore, a web application developed in a particular environment can only be tested by compatible tools. This limitation, together with the complex nature of web application code these days, makes white-box testing tools not as popular as black-box testing tools, at least in an industrial setting. On the other hand, in black-box testing, web applications are checked against standard HTTP requests and responses for probable vulnerabilities. To do so, black-box scanners send virtual attacks to target web applications and process respective responses, looking for HTTP error codes or abnormal strings that may indicate the existence of vulnerabilities [6-9]. Because only HTTP requests and responses are used, black-box testing tools are independent of development environments and are widely used in both academic and industrial settings. Based on the differences between the two approaches mentioned above, in this work, we focus on visualizing scanning results from black-box scanners to make our work as popular as possible.

Applying information visualization techniques to solve computer and information system security problems is a rather young research field called security visualization with its own conference organized each year for practitioners and researchers in information visualization and security to exchange their works [10]. One noteworthy feature that differentiates studies in this field from other security research works is the role of human users. While other security studies treat human users as an external entity, this field considers human users as an internal component that can affect the whole system security by their good or bad decisions. To support users in making effectively and efficiently decisions, security visualization techniques provide tools for situational awareness [11], security data exploration [12], and visual analysis [13]. The level of support is partly determined by the type of data source used in visualization. Data source at a low level of abstraction allows users to do more detailed analysis while data source at a high level of abstraction often brings them overview information more easily. Some popular data source types used in security visualization include network packet [14], IDS log [15], firewall configuration [16], etc. Although information visualization is used in many security domains, it does not attract much interests from web application security researchers. To the best of our knowledge, there are only a few works on web application security visualization that are published until now [17-18].

Web applications have a complex structure with pages and links between them. To support human users in understanding this structure efficiently, visualization techniques are developed to display as much of it as possible while keeping the visible parts clear. However, before the visualization can take place, a modeling step is needed to transform web applications to structures more appropriate to be handled by computing systems. Intuitively, a directed graph is one of the best matched candidates to model a web application because of their similarities: pages are viewed as nodes, and links are viewed as edges. Then, layout algorithms are used to position nodes/edges with regard to some aesthetic criteria [19]. Some widely used algorithms to draw graphs are spring embedder layout [20], force-directed layout [21], and their variations. However, with a middle or large website, the number of pages and links can make its representative graph illegible for users to comprehend. So, more simple models are also used with some trade-offs. Among them, tree is one of the most popular structures used to visualize websites. There are two widely used approaches to draw trees: space-filling [22] and node-link diagram [23-24], each has its own advantages and disadvantages.

3 System Architecture

The system architecture of our proposed framework is depicted in Fig. 1.

In designing this framework, our expectation is that there is no limit on the number of vulnerability scanners that can be used. To realize this feature, we separate the scanning result of each tool from each other and assign an individual Result normalization component to each of the results. The task of the Result normalization components is to convert scanning results generated by their respective scanners into a common format that can be used later by the Statistics visualization component. However, because each scanner has its own unique data, types, and ranges, the normalization process is far from complete. Therefore, to make the framework support as many scanning tools as possible, we extract a minimum set of data that should exists in all scanners' results, such as: vulnerable URLs, severity levels, full requests and responses. For tools' specific data, we provide access to them by linking user interactions to respective tools' interface.

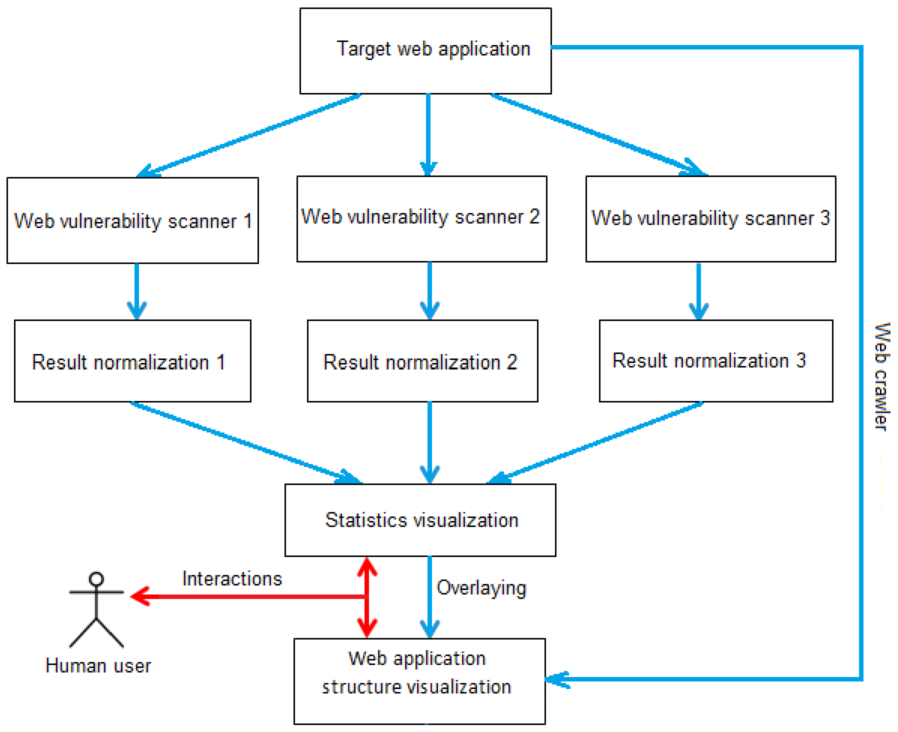


Fig. 1. Our proposed framework's system architecture

The Web application structure visualization component is responsible for presenting the target web application in an organized and easy to understand manner. In this work, we focus on the vulnerabilities discovered by scanners on pages, therefore we model target websites as rooted trees. This results to a more simple visualization and is appropriate to medium to large websites. Particularly, we use the Tree pattern of Polar plot [25] to emphasize the hierarchical nature of target web applications. The structural data this component uses for visualization is provided by a web crawler.

The task of the Statistics visualization component is to calculate basic statistics from raw result data obtained by scanning tools and overlie them on the drawing of target web application structure. More specifically, for each web page, we compute the total numbers of vulnerabilities discovered by each tool, grouped by their severity levels, and display these numbers as a stack chart on the respective web page node. Although simple, this statistics overlaying technique provides some analytical capabilities that are not easy to acquire by traditional tools: comparing the tools' effectiveness at various places; seeing the similarities and differences between nodes; and knowing the overall vulnerability distribution quickly.

Another component in this framework is the Interactions. Human users interact with this component to adjust the appearances of the Web application structure visualization and the Statistics visualization components. Usually, this component is used for more focused analysis, happen after users having an overview picture and determining an interesting area. In this implementation, the framework supports basic interactions like node collapsing/expanding, filtering, zooming, and details-on-demand.

4 Visualization and Interaction Design

4.1 Web Application Structure Visualization

We model each target web application as a rooted tree with the home page as the root node. Each page with a unique URL is modeled as a tree node. Parent-child relationships between nodes are determined based on URL values of their respective pages. For example, a node with its page URL as "homepage/something" is the parent of a different node with its page URL as "homepage/something/child", but not the parent of another node with its page URL as "homepage/somethingelse/child". Two nodes with their page URLs as "homepage/child1" and "homepage/child2" are sibling nodes, and the common parent of them is the root node. From this construction, it is easy to see that each node will have no more than one parent, and the constructed graph is indeed a tree. Furthermore, the size of each node is decided by the number of vulnerabilities found on the related page of that node. We use node size to encode the number of vulnerabilities to make pages with many security issues stand out from the rest. We then apply a Polar plot-like on the constructed tree to create the needed visualization result. Compared to top-down layout, this radial layout has more room for the nodes. Fig. 2 depicts a sample website visualized by our method.

4.2 Statistics Visualization

For each page in the target web application, we extract the details of the vulnerabilities found and use it to create the Statistics visualization. In particular, we calculate the total numbers of vulnerabilities each scanner found and further divide these numbers into groups by common severity levels of vulnerabilities in those groups. These numbers are then used to draw stack charts with each color for each scanner's result, and each transparent level for each severity level (Fig. 3, left). In this prototype version, we use three values: high, medium, and low to present severity levels. For scanning tools that output three severity levels, their results can be used directly. But for scanning tools that output a different number, a manual normalization rule is required at setup time. After all charts are built, they are overlaid on their respective tree nodes (Fig. 3, right).

4.3 User Interactions

We implement some basic user interactions to provide more focused analysis when needed. These interactions include: node collapsing/expanding, filtering, zooming, and details-on-demand. When a node collapses, its subtrees disappear and the vulnerabilities on these subtrees are totaled for the collapsed node. Vice versa, when a node expands, its subtrees become visible with stack charts on them. While node collapsing is useful for summarizing data at a higher level, node expanding is appropriate for more detailed analysis. Different from node collapsing/expanding, which affects a selected part of the visualization, filtering affects the whole of it. Filtering is used to display interesting information only, for example, statistics of some particular scanners, or nodes whose vulnerabilities exceed a threshold. Zooming provides a simple mechanism to view an interesting region more clearly. And the remaining interaction, details-on-demand, allows users to view more specific information about the vulnerabilities as well as brings them to particular tools' interfaces when required. Fig. 4 demonstrates some interactions mentioned above.

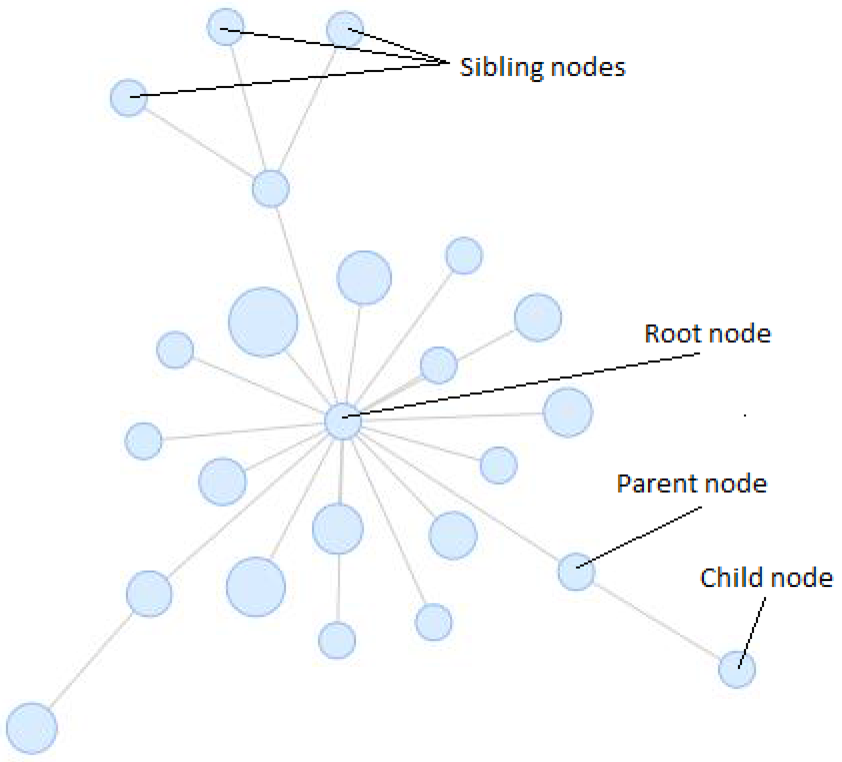


Fig. 2. Working of the Web application structure visualization component. The target web application is modeled as a rooted tree with the home page as the root node. Parent-child relationships are determined by URL values. Node sizes are decided by the number of vulnerabilities found on their respective pages.

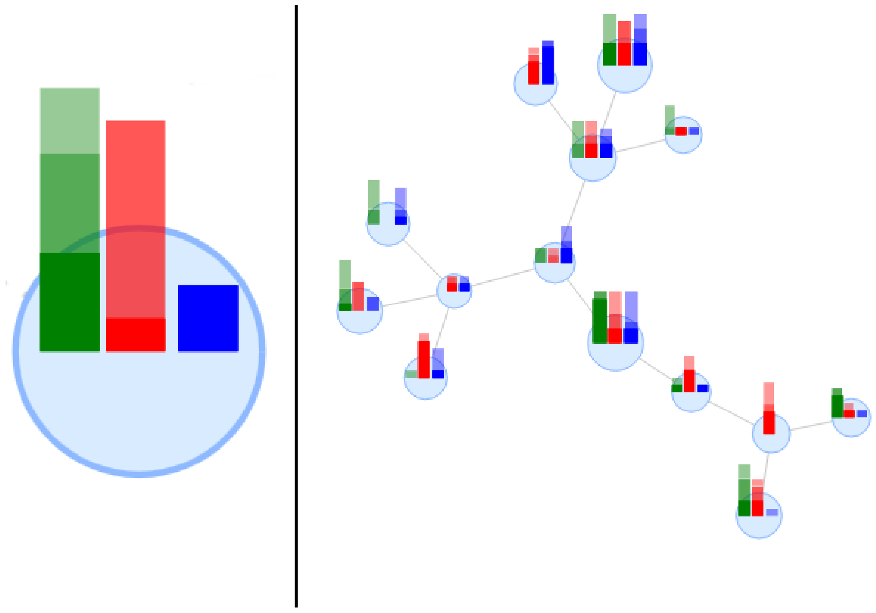
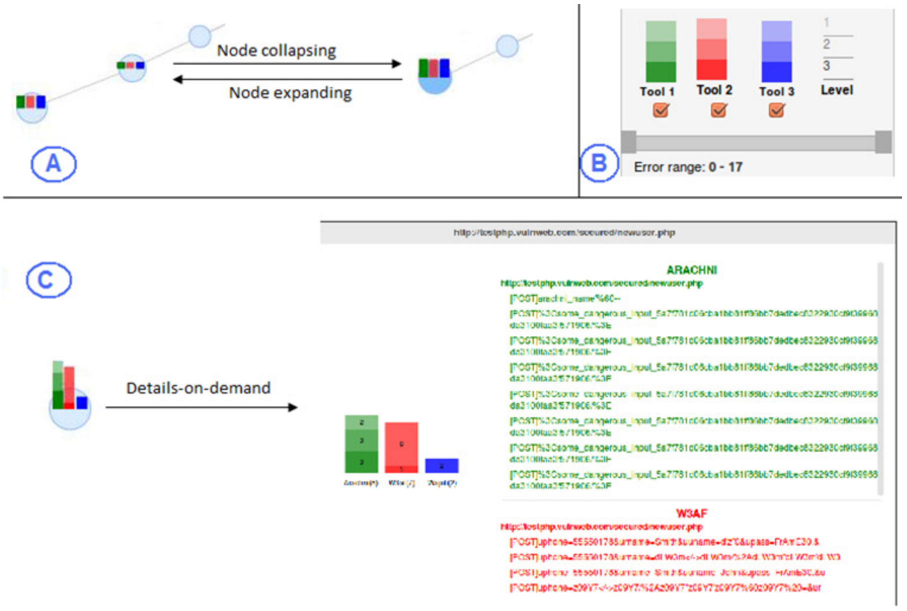


Fig. 3. The Statistics visualization component. The number of vulnerabilities are totaled for each scanner and grouped by their severity levels. On the left: stack charts are used to display the statistics; each scanner's result has a unique color; and transparent levels are assigned to severity levels. On the right: the stack charts are overlaid on their respective nodes.



```
ARACHNI
http://testphp.vulnweb.com/secured/newuser.php
[POST]arachni_name%60--
[POST]%3Csome_dangerous_input_5a7f781c06cba1bb81f95bb7dedbec8322930cf9f36660
da3100faa3f571906%3E
[POST]%3Csome_dangerous_input_5a7f781c06cba1bb81f95bb7dedbec8322930cf9f36668
da3100faa3f571906%3F
[POST]%3Csome_dangerous_input_5a7f781c06cba1bb81f86bb7dedbec8322930cf9f39968
da3100faa3f571906%3F
[POST]%3Csome_dangerous_input_5a7f781c06cba1bb81f86bb7dedbec8322930cf9f39968
da3100faa3f571906%3E
[POST]%3Csome_dangerous_input_5a7f781c06cba1bb81f95bb7dedbec8322930cf9f36660
da3100faa3f571906%3E
[POST]%3Csome_dangerous_input_5a7f781c06cba1bb81f95bb7dedbec8322930cf9f36668
da3100faa3f571906%3F
[POST]%3Csome_dangerous_input_5a7f781c06cba1bb81f86bb7dedbec8322930cf9f39968
da3100faa3f571906%3F
W3AF
http://testphp.vulnweb.com/secured/newuser.php
[POST]uphone=555501785urname=Smith&uuname=dz'0&upass=FrAm830.8
[POST]uphone=555501785urname=d! W0m<-->d! W0m%2Ad. W0m'd! W0m'd. W3
[POST]uphone=555501785urname=Smith&uuname=John&upass=FrAm830.8u
[POST]uphone=209Y7<-->209Y7%2A209Y7"209Y7'209Y7%60209Y7%20=&ur
```

Fig. 4. Some interactions our framework supports. A: node collapsing/expanding. B: filtering user interface. C: details-on-demand in action.

5 Experiment

5.1 Implementation Details

We select three open source web application scanners for this framework prototype implementation: Arachni [26], w3af [27], and Wapiti [28]. There is no particular reason for the selection, except that they are free so we can start to work with them immediately. For each scanner, we create a related parser that can process that scanner's result into a common format, and then store them in one central database.

The visualization and user interaction are implemented on the web platform. Although web-based applications cannot offer as high performance as native applications do, we hope that the flexibility and popularity that the web offer can make our work more accessible. The library we use for the implementation is D3.js [29]. This library uses only W3C-compliant formats and languages like Scalable Vector Graphics (SVG), JavaScript, HTML5, and Cascading Style Sheets (CSS3), making it a good choice for our purpose.

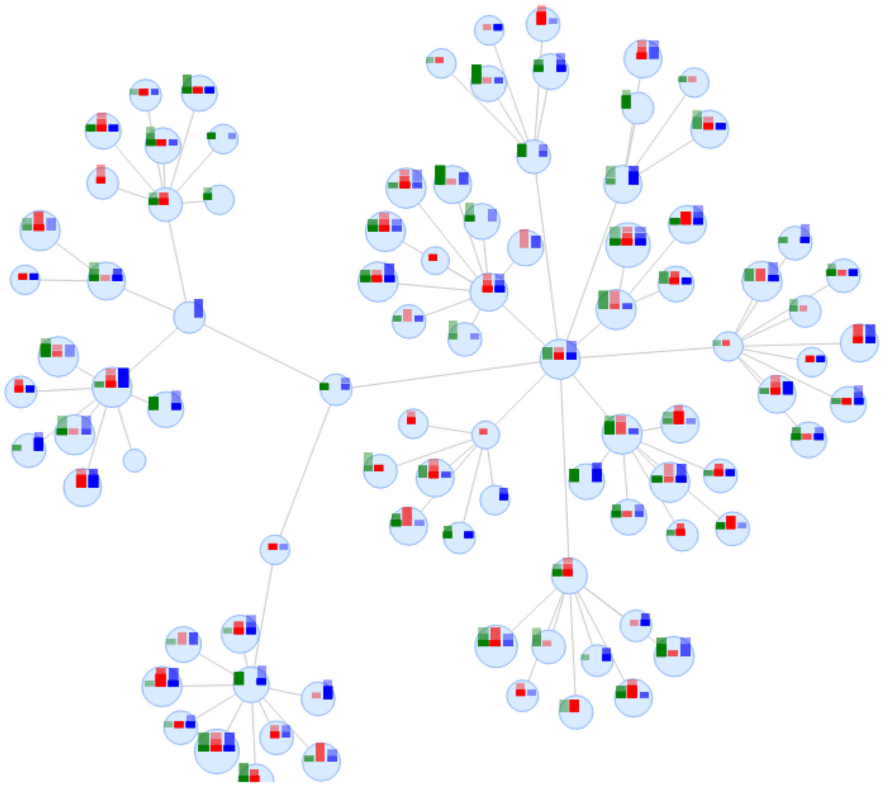


Fig. 5. When the number of nodes is about 70, we cannot instantly grasp and understand the visualization result without some interactions. In this figure, it is not easy for us to decide which node is root or which node has the largest number of vulnerabilities by viewing only.

5.2 Testing the Framework

We create some virtual websites whose sizes increase from small to large to test the framework. Furthermore, we deliberately make some security holes on random pages in these sites. Our main goal in doing so is to create an artificial environment in which we can determine the framework's limit. As our experiment results point out, there are in fact two limits which we call technical-limit and user-limit.

Technical-limit is the limit of the visualization technique itself. It is the size of the target web application at which the performance of the framework decrease significantly. It is difficult for users to do analysis tasks at this limit, because of the lack of interactivity. Our prototype reaches this limit when the target website's number of pages is about 200. User-limit is the limit caused by users' cognitive ability. It is the size of the target web application at which the visualization is difficult for users to grasp and analyze without interaction. From our own experiments, this limit is much lower than the technical-limit, only about 70 pages. Fig. 5 demonstrates a case where the user-limit is reached for us.

6 Conclusion and Future Works

In this paper, we have described our proposal to solve two limitations of traditional web application vulnerability scanners: the first one is the lack of an effective interface to do dynamic analysis; and the second one is the difficulty in integrating scanning results of many tools together. To overcome the first problem, we created a novel visualization technique that can display scanning result statistics over the whole target web application. In addition, we apply basic interactions to provide more focused analysis when needed and to deal with medium to large websites. For the second problem, we design and implement a flexible framework that can integrate scanning results of many scanners together. We argue this work can enhance the effectiveness and efficiency of human users in security analysis task. At least, the proposed framework has three features support that belief: it allows users to compare performances among tools at various places; it helps them spot the similarities and differences between pages; and it shows them the overall vulnerability distribution in a novel way.

However, we need to do more real-world experiments to confirm the advantages of this work. As the experiment results point out, not only we need to test the framework's visualization technique, but more importantly, we also need to test it with regard to human cognitive perspective. Another essential, but not easy, task is getting feedbacks from actual web security penetration testers. These feedbacks are crucial in improving the practicality of the framework.

Acknowledgements. This research is funded by Ho Chi Minh City University of Technology (HCMUT) under grant number T-KHMT-2014-37. The authors would like to thank the D-STAR Lab members for their programming supports and constructive feedbacks. The authors would also like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

References

1. Symantec: 2014 Internet Security Threat Report, vol.19, http://www.symantec.com/security_response/publications/threatreport.jsp
2. Jovanovic, N., Kruegel, C., Kirda, E.: Precise Alias Analysis for Static Detection of Web Application Vulnerabilities. In: 2006 Workshop on Programming Languages and Analysis for Security, pp. 27–36. ACM, New York (2006)
3. Wassermann, G., Su, Z.: Sound and Precise Analysis of Web Applications for Injection Vulnerabilities. In: 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 32–41. ACM, New York (2007)
4. Wassermann, G., Su, Z.: Static Detection of Cross-Site Scripting Vulnerabilities. In: 30th International Conference on Software Engineering, pp. 171–180. ACM, New York (2008)
5. Rimsa, A., D'amorim, M., Pereira, F., Bigonha, R.: Efficient Static Checker for Tainted Variable Attacks. Science of Computer Programming 80, 91–105 (2014)
6. Huang, Y.-W., Huang, S.-K., Lin, T.-P., Tsai, C.-H.: Web Application Security Assessment by Fault Injection and Behavior Monitoring. In: 12th International Conference on World Wide Web, pp. 148–159. ACM, New York (2003)
7. Kals, S., Kirda, E., Kruegel, C., Jovanovic, N.: SecuBat: a Web Vulnerability Scanner. In: 15th International Conference on World Wide Web, pp. 247–256. ACM, New York (2006)
8. Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., Vigna, G.: Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. In: 2008 IEEE Symposium on Security and Privacy, pp. 387–401. IEEE Computer Society, Washington (2008)
9. Doupé, A., Cavedon, L., Kruegel, C., Vigna, G.: Enemy of The State: a State-Aware Black-Box Web Vulnerability Scanner. In: 21st USENIX Conference on Security Symposium. USENIX Association, Berkeley (2012)
10. Visualization for Cyber Security, http://www.vizsec.org/
11. Paula, R., Ding, X., Dourish, P., Nies, K., Pillet, B., Redmiles, D., Ren, J., Rode, J., Filho, R.: In the Eye of the Beholder: a Visualization-Based Approach to Information System Security. International Journal of Human-Computer Studies 63, 5–24 (2005)
12. Leschke, T., Sherman, A.: Change-Link: a Digital Forensic Tool for Visualizing Changes to Directory Trees. In: 9th International Symposium on Visualization for Cyber Security, pp. 48–55. ACM, New York (2012)
13. Fischer, F., Mansmann, F., Keim, D.A., Pietzko, S., Waldvogel, M.: Large-Scale Network Monitoring for Visual Analysis of Attacks. In: Goodall, J.R., Conti, G., Ma, K.-L. (eds.) VizSec 2008. LNCS, vol. 5210, pp. 111–118. Springer, Heidelberg (2008)
14. Conti, G., Grizzard, J., Ahamad, M., Owen, H.: Visual Exploration of Malicious Network Objects Using Semantic Zoom, Interactive Encoding and Dynamic Queries. In: 2005 IEEE Workshops on Visualization for Computer Security, IEEE Computer Society, Washington (2005)
15. Abdullah, K., Lee, C., Conti, G., Copeland, J., Stasko, J.: IDS RainStorm: Visualizing IDS Alarms. In: 2005 IEEE Workshops on Visualization for Computer Security. IEEE Computer Society Press, Los Alamitos (2005)
16. Mansmann, F., Göbel, T., Cheswick, W.: Visual Analysis of Complex Firewall Configurations. In: 9th International Symposium on Visualization for Cyber Security, pp. 1–8. ACM, New York (2012)

17. Dang, TT., Dang, TK.: A Visual Model for Web Applications Security Monitoring. In: 2011 International Conference on Information Security and Intelligence Control, pp. 158-162. IEEE Computer Society, Washington (2011)
18. Dang, T.T., Dang, T.K.: Visualization of Web Form Submissions for Security Analysis. International Journal of Web Information Systems 9, 165–180 (2013)
19. Battista, G., Eades, P., Tamassia, R., Tollis, I.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall PTR, Upper Saddle River (1998)
20. Kamada, T., Kawai, S.: An Algorithm for Drawing General Undirected Graphs. Information Processing Letters 31, 7–15 (1989)
21. Fruchterman, T., Reingold, E.: Graph Drawing by Force-Directed Placement. Software: Practice and Experience 21, 1129–1164 (1991)
22. Shneiderman, B.: Tree Visualization with Tree-Maps: 2-D Space-Filling Approach. ACM Transactions on Graphics 11, 92–99 (1992)
23. Munzner, T., Burchard, P.: Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space. In: First Symposium on Virtual Reality Modeling Language, pp. 33–38. ACM, New York (1995)
24. Yee, K.-P., Fisher, D., Dhamija, R., Hearst, M.: Animated Exploration of Dynamic Graphs with Radial Layout. In: 2001 IEEE Symposium on Information Visualization, pp. 43–50. IEEE Computer Society, Washington (2001)
25. Draper, G., Livnat, Y., Riesenfeld, R.: A Survey of Radial Methods for Information Visu-alization. IEEE Transactions on Visualization and Computer Graphics 15, 759–776 (2009)
26. Arachni, http://www.arachni-scanner.com/
27. w3af, http://w3af.org/
28. Wapiti, http://wapiti.sourceforge.net/
29. D3.js, http://d3js.org/