Detection of Web Application Vulnerabilities Accelerated by GPU

Shaotao Li
College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China

ABSTRACT: The number and importance of web application increases fast. At the same time, the influence of vulnerabilities in web application grows as well. Automated tools are urgently needed because manual code reviews are inefficient and fallible. However, the time complexity of previous static code detection tools are O(n), which is not acceptable when processing a large quantity of web applications. We propose a novel method based on GPU to accelerate the detection of Web application vulnerabilities. More specifically, we decrease the time complexity of detecting XSS vulnerabilities based on dependence tree from O(n) to O(log(n)) and decrease the time complexity of detecting SQLI vulnerabilities based on automaton from O(n) to O(1). Experiment results show that the method based on GPU is much faster than traditional method based on CPU.
KEYWORD: XSS, SQLI, Dependence Tree, Automaton, GPU

1 INTRODUCTION

Cross-site Scripting [1] and SQL injections [2] are two most dangerous kinds of attacks in Web application [3]. PHP has been the most popular programming language used by developer [4]. A lot of work has been done to find vulnerabilities in PHP Web applications automatically [5, 6]. Pixy [7] is one the most outstanding and famous tool among them. When studying the source code of Pixy, I found that it generate .dot files for both XSS and SQLI sensitive sinks which could be used to construct dependence trees and automaton respectively. Furthermore, I found that the time complexity of the method in Pixy could be decreased dramatically by using multi-thread programming based on CUDA [8].

Specifically, to determine whether a sensitive sink of XSS is vulnerability can be done by a traversal of its corresponding dependence tree. As to SQLI vulnerability, it can be done by a traversal of its corresponding automaton [9].

2 DETECTION OF XSS VULNERABILITIES BASED ON DEPENDENCE TREE

Both methods on CPU and GPU to process dependence trees are demonstrated in figure 1. In the left part of the figure 1, the dashed arrows labeled by number represent the visiting of single-thread method on CPU. Obviously, the single-thread method on CPU is a depth-first search implemented by recursion. It visits each node of the dependence once to determine whether the root node of the dependence tree is tainted. As to the right part of figure 1, it demonstrates the multi-thread method on GPU. Each of the leaf nodes of the dependence tree is assigned a thread, and then all threads backtrack to the root node and combine information from different successors of the root node concurrently. Since single-thread method on CPU visits each node once, it's time complexity is O (n). Multi-thread method on GPU backtracks from leaf to root, it's time complexity is the height of the tree, which is O (log (n)).

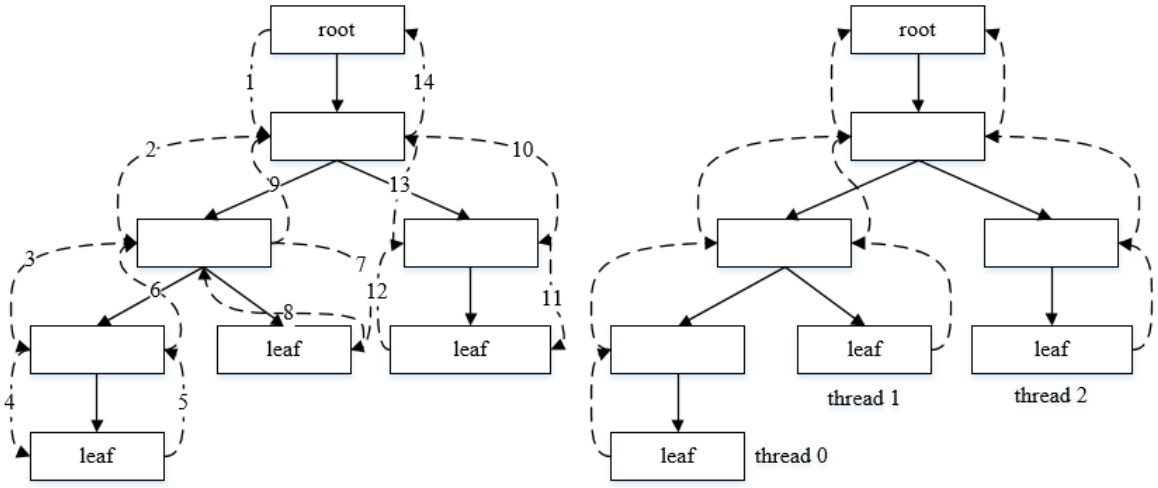

Fig. 1 CPU and GPU methods to process dependence trees

3 DETECTION OF SQLI VULNERABILITIES BASED ON AUTOMATON

Both methods on CPU and GPU to process automatons are demonstrated in figure 2. In the left part of the figure 2, the dashed arrows represent the visiting of single-thread method on CPU. Obviously, it visits each state of the automaton once, the time complexity of is O (n). In the right side of the figure 2, we assign a thread to each state of the automaton, and then all thread visits its corresponding state concurrently. Obviously, the multi-thread method of GPU has a constant time complexity which is O (1).

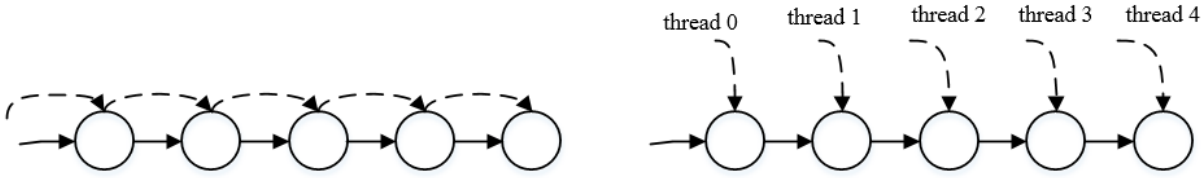

Fig. 2 CPU and GPU methods to process automatons

4 EXPERIMENTS

To evaluate the efficiency of the novel method proposed in this paper. We implement the both the traditional method on CPU and the novel method on GPU. We generate a series of PHP files as input to Pixy, then using Pixy to convert them into .dot files which can be used to construct dependence trees and automatons. Table 1 shows the detail of PC on which the method of CPU runs. Table 2 shows the detail of server on which the method of GPU runs.

Table 1 Detail of Server for CPU Experiment

Item	Detail
CPU	Intel(R) Xeon(R) CPU E5-2609 0 @ 2.40GHz
RAM	4GB
Hard Disk	500GB
OS	CentOS release 6.2 (Final)
JDK	1.7.0

Table 2 Detail of Server for GPU Experiment

Item	Detail
GPU	NVIDIA Tesla K20m
Board Memory	6GB
OS	Ubuntu 12.04
Number of Processor Cores	2046

Fig 3 shows the result of processing 10,000 dependence trees on CPU and GPU. It is worthy of mentioning that the Y-axis is logarithmic not linear. The height of bar is not proportional to the time needed to process the input files. It is obvious that time needed for GPU to process the same input files with CPU is much shorter. We can see that CPU needs about 1,144ms to process a php file that has 500 lines of codes for 10,000 times and about 10,179ms to process a php file that has 5,000 lines of codes for 10,000 times. The number of lines of codes increases 10 times and time increased 9 times. The time complexity of CPU method accords with O (n) as analyzed in the former section. As to GPU, time needed to process a php file that has 5,000 lines of codes is also roughly 11 times than a php file that has 500 lines of code. It seems that it does not ac-

cord with the time complexity of GPU that is O (log (n)). This is because that the time needed for GPU include the time to transmit data from CPU to GPU. Unfortunately, the time needed for transmitting is much longer than time need to process dependence tree in GPU, so the time needed for GPU also roughly accords with O (n).But it grows slower than time needed for CPU, because the time need to process dependence tree is much shorter than CPU.

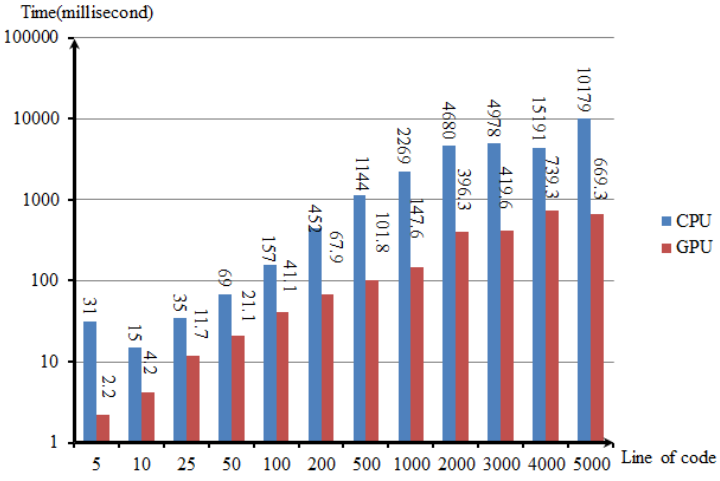

Fig. 3 Result of processing dependence trees on CPU and GPU

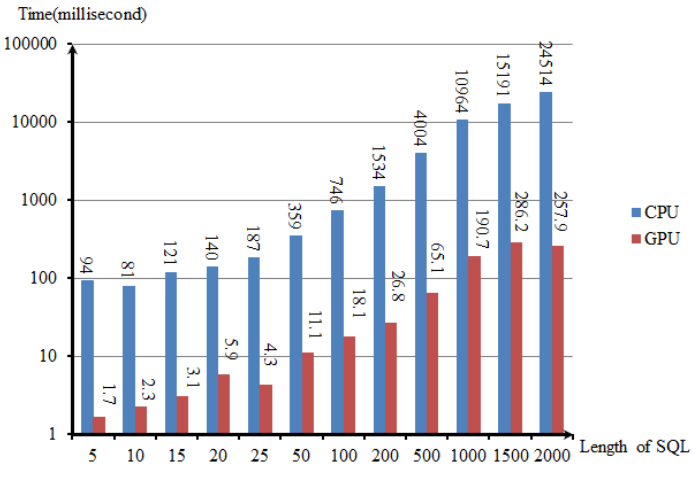

Fig. 4 Result of processing automatons on CPU and GPU

Fig 4 shows the result of processing 100,000 automations on CPU and GPU. It is worthy of mentioning that the Y-axis is also logarithmic not linear. The height of bar is not proportional to the time needed to process the input files. It is obvious that time needed for GPU to process the same input files with CPU is much shorter. We can see that CPU needs about 1,534ms to process a php file that has 200 lines of codes for 100,000 times and about 24,514ms to process a php file that has 2,000 lines of codes for 100,000 times. The number of lines of codes increases 10 times and time increased 16 times. The time complexity of CPU method roughly accords with O (n) as analyzed in the former section. As to GPU, time needed to process a php file that has 2,000 lines of codes is also roughly 10 times than a php file that has 200 lines of code. It seems that it does not accord with the time complexity of GPU that is O (1). This is because that the time needed for GPU include the time to transmit data from CPU to GPU. Unfortunately, the time needed for transmitting is much longer than time need to process automations in GPU, so the time needed for GPU also roughly accords with O (n).But it grows slower than time needed for CPU, because the time need to process dependence tree is much shorter than CPU.

5 SUMMARY

We propose novel methods based on GPU to process dependence trees and automatons that represent the sensitive sinks of XSS and SQLI, respectively. In theory, we decreased the time complexity of processing dependence trees from O (n) to O (log (n)), and decreased the time complexity of processing automatons from O (n) to O (1). Experiment results on CPU show that the time complexity processing dependence trees and automatons both accords with O (n), due to the transmitting delay which is far more longer that processing dependence trees and automatons in GPU, the time needed for GPU also roughly accords with O(n). But the experiment results demonstrate that time needed for GPU grows slower than CPU which shows that the novel methods decrease the time complexity compared to CPU.

REFERENCES

[1] Vogt P, Nentwich F, Jovanovic N, et al. Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis[C]//NDSS. 2007, 2007: 12.

[2] Merlo E, Letarte D, Antoniol G. Automated protection of php applications against SQL-injection attacks[C]//Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on. IEEE, 2007: 191-202.

[3] Williams J, Wichers D. OWASP top 10–2010[J]. OWASP Foundation, April, 2010, 1(5): 8.

[4] Jochen Weiland, Michael Schams. Usage of server-side programming languages for websites. http://w3techs.com/blog/entry/web_technologies_of_the_year_2013.

[5] Fortify 360.http://www.fortify.com.

[6] Sandcat.4PHP. http://www.syhunt.com/?section=sandcat4php.

[7] Jovanovic N, Kruegel C, Kirda E. Pixy: A static analysis tool for detecting web application vulnerabilities[C]//Security and Privacy, 2006 IEEE Symposium on. IEEE, 2006: 6 pp.-263.

[8] Cuda C. Programming guide[J]. 2012.

[9] Balzarotti D, Cova M, Felmetsger V, et al. Saner: Composing static and dynamic analysis to validate sanitization in web applications[C]//Security and Privacy, 2008. SP 2008. IEEE Symposium on. IEEE, 2008: 387-401.