Dynamic Taint Tracking for Domain Driven Security

Specification and Time Schedule

FREDRIK ADOLFSSON - FREADO@KTH.SE

Master in Computer Science
Date: March 21, 2018
Supervisor: Musard Balliu
Examiner: Mads Dam
Principal: Jonatan Landsberg & Simon Tardell
School of Computer Science and Communication

Contents

Chapter 1

Introduction

The creation of the World Wide Web (web) has caused a huge impact on today's society [21]. The web is a source for information and it connects the world through a unanimous platform. Many businesses have decided to take advantage of the web platform to share information and communicate with customers. However, this does not come without drawbacks. The information sharing is a weakness in the same manner as it is a strength. The web application is not only accessible for the targeted user groups but for anyone with access to the web. This entails that malicious users who wish to abuse and/or cause harm to other users have the accessibility to possibly do so.

There are several possible attacks that a web application is vulnerable to. The attack most frequently conducted today will probably not be the same as the most conducted in the future. The Open Web Applications Security Project, known as OWASP, is an online community which aims to provide knowledge about how to secure web applications [10]. OWASP have produced reports about the top 10 security risks for a web application and the latest was published 2017. In this report was Injection Attacks number one and Cross-Site Scripting number seven [11, 10, 4].

This thesis will look at the two named security risks and perform evaluations and benchmarks of a possible solution to prevent these kinds of unwanted information disclosure.

Chapter 2

Background

Discussions about application security often relies on the CIA Triad which represent the three primary concepts in information security. These three are confidentiality, integrity and availability. Confidentiality are rules that specifies the access restrictions to the application. Integrity specifies that application data should be accurate and not altered. Availability is about ability to access the application and application data [3]. This thesis focuses on confidentiality and integrity vulnerabilities and how we can prevent them.

Injection Attacks are the number one security vulnerability for web applications. Injection Attack is a collection name for any attack where the attacker's input changes the intent of the execution. Some possible versions of Injection Attack are injection of queries that manipulates SQL, NoSQL, OS and LDAP executions [11]. The most common goals and result of Injection Attack is file destruction, lack of accountability, denial of access and data loss [18].

Cross-Site Scripting on the other hand was listed as number seven in OWASP top 10 security vulnerabilities published 2017 [11]. The origin of the attack goes back to the beginning of the web and one of the first Cross-Site Scripting attack was conducted just after the release of JavaScript. The attack was conducted through loading a malicious web application into a frame on the site that the attacker wanted to gain control of. The attacker could then leverage malicious scripts from the malicious frame to access any content that was entered into the web application. The first prevention for Cross-Site Scripting was

then introduced through the standard of Same-Origin Policy. Same-Origin Policy restricts JavaScript to only access content from its own origin [6, 15].

To prevent these types of security vulnerabilities in web applications, a variety of tools and methodologies have been created. Dynamic Taint Tracking is one of these tools which goal is to prevent possible Injection and Cross-Site Scripting attacks in runtime. This is done by marking untrusted input from sources, which is a marking point where malicious data might enter the system, as tainted. This is done through a taint flag attached to the input. This taint flag follows the input throughout the application and propagates onto any other data it encounters. It is possible to detaint (remove the taint flag) tainted data but this is only done after the data have been sanitized through validation. The taint flags are checked in areas called sinks which are markings for entry points to sensitive code [13, 19]. The decision of what to do when a tainted variable try to pass through a sink might vary depending on the application. However, the common reaction is to stop the execution of the tainted code. Other actions such as logging or raising an alarm are also common.

An example of Taint Tracking can be seen in listing 2.1. In this example *getAttribute* is a source, *executeQuery* a sink and *validate* a sanitizer. On line one the input from the source is flagged tainted and the taint propagates onto *userId*. The sanitizer on line two validates *userId* and removes the taint flag. Lastly, the sink on line three execute since the argument is not tainted. If a user sends in a malicious userId containing "101 OR 1 = 1" the validator would either halt the execution or sanitize the String. However, removing line two would result in tainted data entering the sink. This would without a Dynamic Taint Tracking tool result in giving the malicious user the entire list of Users. With a Dynamic Taint Tracking tool however, would result in the sink halting the execution therefore preventing unwanted information disclosure.

Listing 2.1: Taint Tracking

```
1  userId = getAttribute("userId");
2  validate(userId)
3  executeQuery("SELECT␣*␣FROM␣Users␣WHERE␣
       userId␣=␣" + userId);
```

The above described Dynamic Taint Tracking tool focuses on preventing malicious code to enter the application. There are security policies restricting input from sources to pass through sinks without first being sanitized through validation. The same application could however be used to enforce policies restricting sensitive data from sinks to pass through sources without being sanitized to not contain sensitive data.

This thesis will implement and evaluate a Dynamic Taint Tracking tool to prevent confidentiality and integrity vulnerabilities in web applications. The thesis will also evaluate the security benefits of Domain Driven Security, a programming paradigm which has been proposed to combat confidentiality and integrity vulnerabilities. Concretely, we will benchmark our Dynamic Taint Tracking tool against injection, cross-site scripting and information disclosure vulnerabilities.

2.1 Related Work

Stendahl [17] wrote a thesis in 2016 where he evaluated if a Domain Driven Security can prevent Injection Attacks and Cross-Site Scripting. He concluded that there is a security gain towards Injection Attacks and Cross-Site Scripting by following the Domain Driven Security methodology. The gained security comes from proper validation of variables before propagating the data into the value objects.

Haldar, Chandra, and Franz [7] have written a report about Dynamic Taint Tracking in Java where they try to solve the problem of not properly validating user input. They managed to construct a tool that is independent from the web applications source code and the results from using the tool is a gain in security. Haldar, Chandra, and Franz [7] ran their benchmarks on OWASP's project WebGoat [2] but acknowledged in their report that benchmarks of real-world web applications need to be tested.

There do exist two Dynamic Taint Tracking tools where Phosphor [14] is one and Security Taint Propagation [5] is another. Both are open source projects and developed for Java applications.

2.2 Goal & Objective

The goal of this thesis is to implement and benchmark a Dynamic Taint Tracking tool which aims to enforce the security gains of Domain Driven Security. The Dynamic Taint Tracking tools objective is to prevent unwanted information disclosure by prevention of Injection and Cross-Site Scripting attacks in runtime. Possible malicious code shall be prevented from executing and logged.

The principal, Omegapoint, is interested in everything that might validate, invalidate, evolve or bring a further value to the programming paradigm Domain Driven Security. The reason for this is because the concept of Domain Driven Security was born and is in development by Omegapoint consultants. Omegapoint also like to see a prototype of a Dynamic Taint Tracking tool which can block attacks in runtime.

Chapter 3

Research Question & Method

How can an implementation of a Dynamic Taint Tracking tool enforce the security gains of Domain Driven Security.

The assignment would be to benchmark and evaluate the implementation of a Dynamic Taint Tracking tool and discuss if it helps to enforce the security gains of Domain Driven Security. The process of this thesis would be to conduct, in order:

Literature Study The literature study is where information relevant to the thesis need to be gathered and presented. Details of what the focus will lie upon is presented in chapter 5.

Tainting & Detainting This step is the part where tainting and detainting rules are decided. These needs to be decided since the next step is the implementation of the Dynamic Taint Tracking tool. I will focus on defining rules that enforces security policies to protect for integrity. The question of how these can be reused or slightly modified to enforce policies of confidentiality protection should also be mentioned.

Implementation The implementation step is where the Dynamic Taint Tracking tool is implemented. Omegapoint have developed a proof of concept product which I will continue my work upon. This tool is developed in and for Java with help of the Javassist [9] which makes the manipulation of bytecode easier. The proof

of concept is developed to check taint on HTTP query strings through a Spring server. I will continue the work and expand its coverage.

Benchmarking This step is where the Dynamic Taint Tracking tool will be benchmarked. The benchmarking will be a comparison to see how it relates to other similar applications. The two application that will be the reference points are Phosphor [14] and Security Taint Propagation [5]. The applications to benchmark will be a mixture of some real-world applications and benchmarking applications. Omegapoint have two internal applications that can be used and Standford SecuriBench [8] and Stanford SecuriBench Micro [16] are sets of open source programs that are developed for evaluating security tools, which therefore are optimal to use for benchmarking.

The values that is in focus during the benchmark are the values in the table below.

- **Performance Overhead** - Measure of time added to the execution by using the Dynamic Taint Tracking tool.
- **Prevention Rate** - Percentage measure of ratio between total attacks and prevented attacks.
- **False Positive Rate** - Percentage measure of ratio between spurious attacks and real attacks.

The tools to use to test the previously named applications will be all or some of; OWASP Zed [12], w3af [20] and Loader [1].

Analysis The analysis step is where the benchmarking results is reflected upon and written into the report. The discussion will contain thought of how it helps to secure applications and what have been successful and unsuccessful.

Report Writing & Presentation The last steps are to finalize the report and present the thesis.

Chapter 4

Evaluation & News Value

The thesis is considered to reach its objective if a Dynamic Taint Tracking tool have been developed and is in working state where integrity and confidentiality policies are enforced. The tool must have been benchmarked with comparisons to current state of art applications where the parameters performance overhead, prevention rate and false positive rate are the values for the reasoning. There should also be an evaluation containing well thought comments and observations about the benchmarking results and how the results might or might not enforce the programming paradigm Domain Driven Security.

The evaluation is conducted using Stanford SecuriBench [8] and Stanford SecuriBench Micro [16] as benchmarkings. Both are sets of applications where the number of security vulnerabilities is known. The difference between the two is that Stanford SecuriBench Micro contains smaller test cases while Stanford SecuriBench consists of faked "real-life" applications.

These benchmarking applications will then be used together with the security scanners OWASP Zed and w3af. Tests without any of the Dynamic Taint Tracking tools is then to be conducted. This is to acquire a baseline with information about what vulnerabilities that can be detected with the scanners. The same tests will then be run with each of the Dynamic Taint Tracking tool. The parameters performance overhead, prevention rate and false positive will then be found by comparing the Dynamic Taint Tracking tool with the baseline execution.

The relevance in the thesis lies in the problem with software security. Since we are going towards an age where digitalization only grows larger is the question about how we can secure our software extremely important and relevant. The hypothesis is that we can help in the process of enforcing more secure software. The question is with how much and if there are negative side effects. The work should be of interest for anyone wanting to see a gain in their systems security. This application, or an application of similar sort, could protect their system without any, or minor, modifications to the source code. This is especially of interest for individuals or business who have applications where the work needed to secure them would result in too much rework.

Chapter 5

Pre-study

The literature study will focus on gathering the relevant information needed for the report. These areas are listed in the table below:

- Java Applications
- Injection Attacks
- Cross-Site Scripting
- Dynamic Taint Tracking
- Domain Driven Security
- Javassist

Research into JVM modifications must also be included since it is needed for the implementation of the Dynamic Taint Tracking tool. The information will be obtained by researching for relevant books, reports and other possible material. Two of the founders of the concept of Domain Driven Security work at Omegapoint and are accessible for questions. Conduction interviews with the founders might be of interest.

Chapter 6

Conditions & Schedule

6.1 Resources

To save some time the development of the Dynamic Taint Tracking tool will continue the work that Simon Tardell have started. Which is a tool developed in and for Java with help of the Java library Javassist [9]. Applications to use to evaluate the implementation is also of need. Examples of these have been mentioned in chapter 3.

6.2 Limitations

- The Dynamic Taint Tracking tool does not have to be a production ready. The goal is to develop a prototype.
- Web applications is the targeted applications.
- The scope of the thesis will not contain Static Taint Tracking.
- The tool is developed in Java with Javassist.

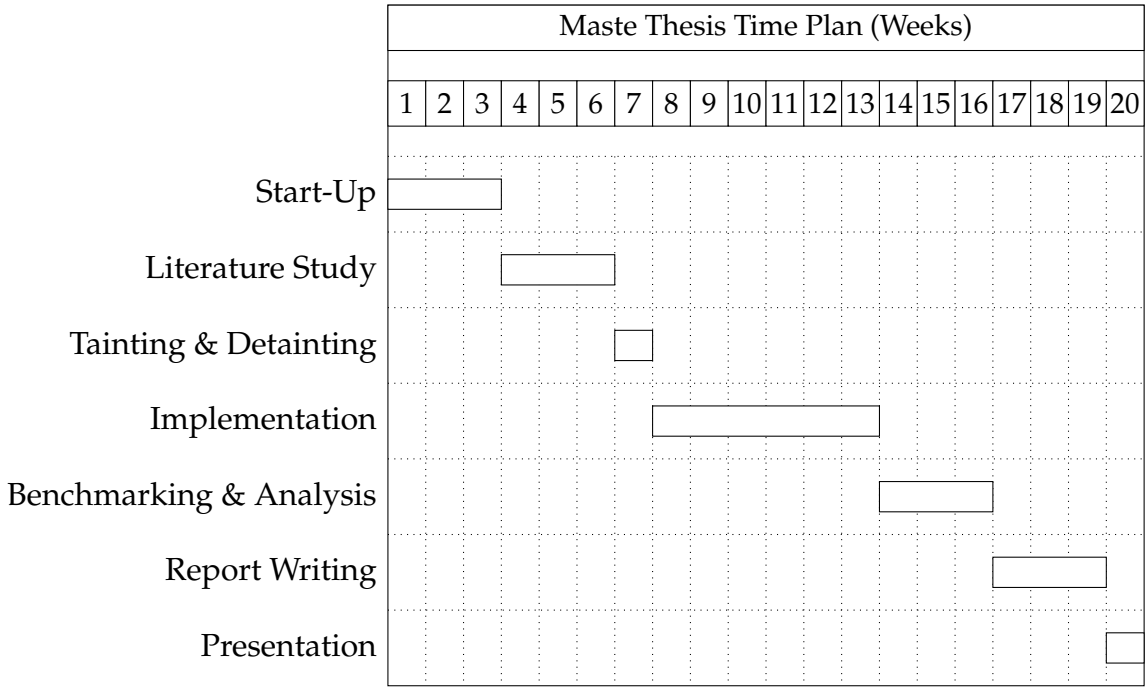
6.3 Company Supervisor

- **Jonatan Landsberg:** Will assist with supervision on the academic part if the thesis.

- **Simon Tardell:** Supervisor in the technical parts of the thesis. He is also the author of the first draft of the Dynamic Taint Tracking tool which this thesis will continue its work upon.

6.4 Time Plan

Below is my time plan for the Master's Thesis. The goal is to continuously, throughout all phases, add to the report. I have also reserved a couple of weeks in the end for writing the report. I believe that this time can be used to add to or rewrite sections if needed.



Bibliography

[1] *Application Load Testing Tools for API Endpoints with loader.io*. URL: https://loader.io/ (visited on 03/05/2018).

[2] *Category:OWASP WebGoat Project - OWASP*. URL: https://www.owasp.org/index.php/Category:OWASP%7B%5C_%7DWebGoat%7B%5C_%7DProject (visited on 03/06/2018).

[3] "Chapter 1 - What is Information Security?" eng. In: *The Basics of Information Security*. 2014, pp. 1–22. ISBN: 978-0-12-800744-0.

[4] Michael Cross. *Developer's guide to web application security*. eng. Rockland, MA: Syngress Publishing, 2007. ISBN: 1-281-06021-6.

[5] *Dynamic Security Taint Propagation in Java via Java Aspects*. URL: https://github.com/cdaller/security_taint_propagation (visited on 03/06/2018).

[6] Seth Fogie. *XSS attacks cross-site scripting exploits and defense*. eng. Burlington, MA: Syngress, 2007. ISBN: 1-281-06024-0.

[7] Vivek Haldar, Deepak Chandra, and Michael Franz. "Dynamic Taint Propagation for Java". In: (). URL: https://pdfs.semanticscholar.org/bf4a/9c25889069bb17e44332a87dc6e2651dce86.pdf.

[8] *Introduction to Stanford SecuriBench*. URL: https://suif.stanford.edu/%7B~%7Dlivshits/securibench/ (visited on 03/07/2018).

[9] *Javassist by jboss-javassist*. URL: http://jboss-javassist.github.io/javassist/ (visited on 02/27/2018).

[10] Open Web Application Security Project. *OWASP*. URL: https://www.owasp.org/index.php/Main%7B%5C_%7DPage (visited on 02/01/2018).

[11] OWASP. "OWASP Top 10 - The Ten Most Critical Web Application Security Risks". In: *Owasp* (2017), p. 22. URL: https://www.owasp.org/images/7/72/OWASP%7B%5C_%7DTop%7B%5C_%7D10-2017%7B%5C_%7D%7B%5C%%7D28en%7B%

5C%%7D29.pdf.pdf%7B%5C%%7D0Ahttp://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:OWASP+Top+10+-2010%7B%5C#%7D1.

[12] *OWASP Zed Attack Proxy Project - OWASP*. URL: https://www.owasp.org/index.php/OWASP%7B%5C_%7DZed%7B%5C_%7DAttack%7B%5C_%7DProxy%7B%5C_%7DProject (visited on 03/05/2018).

[13] Jinkun Pan, Xiaoguang Mao, and Weishi Li. "Analyst-oriented taint analysis by taint path slicing and aggregation". In: *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS* 2015-November (2015), pp. 145–148. ISSN: 23270594. DOI: 10.1109/ICSESS.2015.7339024.

[14] *Phosphor: Dynamic Taint Tracking for the JVM*. URL: https://github.com/gmu-swe/phosphor (visited on 03/06/2018).

[15] *Same Origin Policy - Web Security*. URL: https://www.w3.org/Security/wiki/Same%7B%5C_%7DOrigin%7B%5C_%7DPolicy (visited on 02/07/2018).

[16] *Stanford SecuriBench Micro*. URL: https://suif.stanford.edu/%7B~%7Dlivshits/work/securibench-micro/ (visited on 03/15/2018).

[17] Jonas Stendahl. "Domain-Driven Security". In: (2016), p. 39. URL: http://kth.diva-portal.org/smash/get/diva2:945707/FULLTEXT01.pdf.

[18] Praveenkumat H Subbulakshmi T. *Secure Web Application Deployment Using Owasp Standards: An Expert Way of Secure Web Application Deployment*. Createspace Independent Publishing Platform, 2017.

[19] Guru Venkataramani et al. "FlexiTaint: A programmable accelerator for dynamic taint propagation". In: *Proceedings - International Symposium on High-Performance Computer Architecture* (2008), pp. 173–184. ISSN: 15300897. DOI: 10.1109/HPCA.2008.4658637.

[20] *w3af - Open Source Web Application Security Scanner*. URL: http://w3af.org/ (visited on 03/05/2018).

[21] *World wide web skapas – nu kan internet bli en publiksuccé | Internetmuseum*. URL: https://www.internetmuseum.se/tidslinjen/www/ (visited on 03/06/2018).