

Applying Dynamic Taint Propagation in Order to Enforce Domain Driven Security

FREDRIK ADOLFSSON

Master in Computer Science

Date: February 23, 2018

Supervisor: Musard Balliu

Examiner: Mads Dam

Swedish title: Tillämpa dynamisk taint propagering för att
genomdriva domändriven säkerhet

School of Computer Science and Communication

Abstract

Sammanfattning

Contents

1	Introduction	1
1.1	Definitions	2
1.2	Problem	2
1.3	Aim	2
1.4	Related Work	3
1.5	Delimitations	3
1.6	Methodology	3
2	Background	4
2.1	Web Application	4
2.2	Security Vulnerabilities	5
2.2.1	Injection	5
2.2.2	Cross-site Scripting	7
2.3	Dynamic Taint Propagation	8
2.4	Domain Driven Security	8
2.5	JVM Bytecode Manipulation	9
2.5.1	Javassist	9
3	Implementation	10
3.1	Tainting Rules	10
3.2	Detainting Rules	10
3.3	Benchmarking	10
4	Result	11
5	Discussion	12
5.1	Domain Driven Security	12
6	Future Work	13

7 Conclusion	14
Bibliography	15

Chapter 1

Introduction

One of the greatest strengths with deploying applications on the World Wide Web (web) is that they are accessible from everywhere where there exists a internet access. This is sadly one of its greatest weaknesses as well. The applications are easily accessible for people who wishes to abuse or cause them harm. Among the number of security risks that a web application is vulnerable to is two of the more common Injection Attack and Cross-Site Scripting. [15, 4]

To minimize the risk of accidentally introducing security flaws in to the application have a variety of tools and methodologies been created. One of these is Dynamic Taint Propagation which marks input from the user as tainted through a taint variable attached to the input. This taint variable follows the input throughout the system and propagates onto the other variables it comes in contact with. It is possible to detain the input but this is only done after the input have been validated. The taint value is later checked in sensitive areas through something called sinks. Execution is halted if a tainted variable is detected trying to enter the sensitive area through the sink. [17, 22] One of the methodologies that have been coined is the programming paradigm Domain Driven Security. Domain Driven Security aim to secure applications by focusing on the core domain models and making certain that validation of the value object is correct. [23, 10]

The following sections of the chapter will aim to specify the why and how behind the conduction of the thesis. It starts with a section of *Definitions* followed by *Problem* description and explanation of the thesis *Aim*. These sections is then followed by *Related Work* and *Delim-*

itations section and then, lastly, is there a section about the *Methodology* behind the thesis execution.

1.1 Definitions

Throughout this thesis will a consistent terminology be used. Some terms might be new or might just be usable in different ways. To remove confusion will a list of definitions be introduced in this section.

Definition 1.1.1. Application is used to denote a computer process which is constructed to solve one or more tasks for the user.

Definition 1.1.2. Web Application is used to denote applications deployed with accessibility from the web.

Definition 1.1.3. Domain is explained in Secure by Design [13] as part of the real world where something happens.

Definition 1.1.4. Domain Model is a fraction of the domain where each model have a specific meaning.

1.2 Problem

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

Is it possible to achieve the security gains of Domain Driven Security through applying Dynamic Taint Propagation to web applications. What would the possible drawbacks and advantages be.

1.3 Aim

The aim of this thesis is to implement and evaluate a Dynamic Taint Propagation tool and discuss if it helps to enforce the security gains of Domain Driven Security. The benchmark will check the values; injection prevention rate, false positive rate and added time overhead.

1.4 Related Work

Previous work that have been conducted to Dynamic Taint Propagation is for example the work of Halдар, Chandra, and Franz [8] and Zhao et al. [24]. Both reports have implemented a Dynamic Taint Propagation tool for web deployed Java applications where the manipulation of the Java bytecode is done through Javassist.

Domain Driven Security is somewhat more unknown, compared to Dynamic Taint Propagation, and there are not many reports that discusses and analyses the domain. But one thesis that is noticeable, is the thesis of Stendahl [20].

1.5 Delimitations

The thesis will focus on security risks of web applications. Even though other application areas might be vulnerable to the same kind of vulnerabilities have this delimitation been drawn to keep the scope small and precise. The thesis dose nether create nor benchmark any form of Static Taint Analysis.

The Dynamic Taint Propagation tool is created in and for Java applications with the help of the Java library Javassist. No comparison of different bytecode manipulators will be conducted.

1.6 Methodology

The methodology of this thesis is a combination of quantitative and qualitative research. The first part of the thesis, which consists of the literature study followed by reasoning about tainting and detainting rules, is based on quantitative research. The Second part of the thesis is to evaluate the implemented Dynamic Taint Propagation tool. This is done in a quantitative manner where benchmarks evaluating performance and security gain functionality will be conducted.

Chapter 2

Background

This Chapter will present some background knowledge with needed information to comprehend the chapters to follow. The chapters starts with a generally description about how a *web application* work which then is followed by a section with two of the most common *security vulnerabilities* to a web application. After that is there a chapter describing *Dynamic Taint Propagation* and one describing *Domain Driven Security*. The last section in the chapter is about *JVM bytecode manipulation*.

2.1 Web Application

To make applications available for a large set off people and make te application accessible from now days almost everywhere do businesses deploy their applications to the web. They deployment of an application can vary a lot but the most common architectural structure for a web application is based on three tiers. The first is the presentation tire which is the visual components rendered by a browser. The second is the logic tire which can be seen as the brain of the application. The last and third is the storage tier, where the second tier can store data as needed. [3] Figure 2.1 illustrates this structure.

As can be seen in picture 2.1 dose the tiers only communicate with the tire closest to itself. This causes the second tire to become a safe guard for the tire three where the valuable information for a attacker lies. The storage tier contains all the information the application needs to provide the wanted service. Such information might for example be name, email, personal number and credit card information. [3]

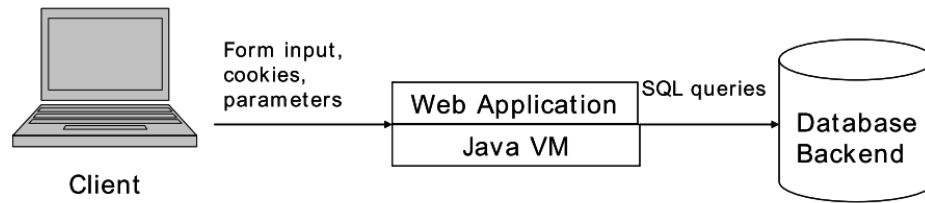


Figure 2.1: Web application structure [8].

2.2 Security Vulnerabilities

The organization Open Web Applications Security Project, mostly known for its shortening (OWASP), is a online community which aim to help to secure web applications. [15] OWASP produces a report about the top 10 security risks with a web application. The report contains information about the ten most common application security risks that for the current year. Information such as how the security risk is exploited and possible prevention method is also presented. [16] This report will look at the number one and eight security risks of 2017 which is injection attacks and cross-site scripting. [16]

2.2.1 Injection

The most common security risk is Injection Attacks. [16] A Injection Attack is any attack where the attacker's input changes the intent of the execution and executes malicious code. Common result of Injection Attacks are file destruction, lack of accountability, denial of access and data loss. [21]

There is two kinds of different Injection Attacks. These two are SQL Injection and Blind SQL Injection. [21] Both will be described below.

SQL Injection

SQL Injection is when a SQL query is tampered with which then results in gaining data from the database which were not intended. Listing 2.1 displays a possible SQL Query which is open to SQL Injections. This is due to the fact that the variable `UserId` is never validated before it is being used to query the database. [3, 21]

Listing 2.1: Code Acceptable to SQL Injection

```
userId = userInput
"SELECT_*_FROM_Users_WHERE_userId=_ " + userId
```

The query will work as intended as long as the user input, notated with *userInput*, only is a user id. But what happens if the user input is *10 or 1 = 1*? This user input would result in the query seen in listing 2.2.

Listing 2.2: SQL Injection

```
SELECT * FROM Users WHERE userId = 10 or 1 = 1
```

This query would result in that the database returns the whole table of users since the second or parameter is always true. This problem can be prevented in a couple of different ways. The first is through validation of the input. In our example do we expect the input to be a Integer. By verifying that the input as seen in listing 2.3 can we protect the query from being injected with unwanted commands.

Listing 2.3: Preventing SQL Injection through Verification

```
userId = userInput
isInteger(userId)
"SELECT_*_FROM_Users_WHERE_userId=_ " + userId
```

A second more common alternative is to use SQL Parameters which handles the verification for the user. This leaves the verification and validation of input up to the SQL engine. Our example written with SQL Parameters can be seen in listing 2.4.

Listing 2.4: Preventing SQL Injection through SQL Parameters

```
userId = userInput
sqlQuery = "SELECT_*_FROM_Users_WHERE_userId=_@0"
db.Execute(sqlQuery, userId)
```

Blind SQL Injection

Blind SQL Injection is very similar to SQL Injection. The only difference is that that attacker dose not receive the wanted information from the database. The information is instead received by monitoring variables such as how long time the response took or what kind of error messages it returns. A example of the first is to create a Blind SQL Injection where the query tells the SQL engine to sleep depending on a condition a example of this can be seen in listing 2.5. [3, 21]

Listing 2.5: Time Based Blind SQL Injection

```
SELECT * FROM Users WHERE userId = 1 WAITFOR DELAY '0:0:5 '
```

The second variant of Blind SQL Injection is by analyzing the error messages and depending on what they return build a image off the wanted answer. [3, 21]

2.2.2 Cross-site Scripting

Cross-Site Scripting (XSS) have been a vulnerability since the beginning of the internet. One of the first XSS attacks created just after the release of JavaScript. The attack was passible by loading another website into a frame on the site that the attacker controls. The attacker could then through JavaScript access any content that is visible or typed into the loaded frame. To prevent this from being possible were the standard of Same-Origin Policy which restricted JavaScript to only access content on its own origin. [7, 19]

But the introduction of the Same-Origin Policy did not stop the attackers from preforming XSS attacks. The next wave of attacks were mostly towards chat roms where it was possible to inject JavaScript tags into the input of the message. Which would then later be reflected by the server itself, when displaying the message for other users, and thereby bypassing the Same-Origin Policy. [7]

Reflected vs Persistent

2.3 Dynamic Taint Propagation

Taint propagation, also known as taint analysis and taint checking [SOURCE NEEDED?], is a tool to analyse the flow of information in a domain. [17] It works by giving input data a tainted property which follows the data and propagate onto other data which it is in contact with. The taint property is later checked in security sensitive sinks. [17]

Perl and Ruby are two programming languages which have adapted to user dynamic taint checking. [18, 11] And there are some tools who enables taint checking for other languages such as TaintDroid [12] and FlexTaint [22].

Two of OWASP top 10 application security risks of 2017 is Injection and Cross-Site Scripting (XSS). [16] Protection against these two attacks are best done by validating input data which taint propagation reminds and forces the developer to do.

2.4 Domain Driven Security

There exists a plethora of tools who aim to help in the process of developing complex domain models, but Domain Driven Design (DDD) is not one if them. [2, 9] DDD is more of a thought process and methodology to follow every step of the process. [6] In *Domain-driven design reference: definitions and patterns summaries* do Evans [5] describe DDD through three core ideas:

- Focus on the core domain.
- Explore models in a creative collaboration of domain practitioners and software practitioners.
- Speak a ubiquitous language within an explicitly bounded context.

The core domain is the part of your product that is most important and often is your main selling point compared to other similar products. [14] A discussion and even possible a documentation describing the core domain is something that will help the development of the product. The idea is to keep everybody on the same track heading in the same direction. [6]

The second idea is to explore and develop every model in collaboration between domain practitioners, who are experts in the given domain, and software developers. This ensures that important knowledge needed to successfully develop the product is communicated back and forth between the two parties. [14] The third idea is important to enable and streamline the second. By using a ubiquitous language will miscommunication between domain and software practitioners be minimized and the collaboration between the two parties can instead focus on the important parts which is to develop the product. [5]

Evans [5] do as well argue about the weight of clearly defining the bounded contexts for each defined model, and this needs to be done in the ubiquitous language created for the specific product. The need of this exists because of the otherwise great risk of misunderstandings and erroneous assumptions in the collaborations between the different models. [14]

Wilander [23] and Johnsson [10] created 2009 a blog post each in a synchronous manner where they together introduces the concept of Domain Driven Security (DDS) to the public. They describe DDS as the intersection between Domain Driven Design (DDD) and application security. DDD is about developing complex domain models and one of the most basic rule of application security is to always validate input data. DDS in other hand, is about the importance of creating and maintaining domain models who are reflecting the product correctly and they are validated so they cant be populated with erroneous data. [23, 10, 1, 20]

2.5 JVM Bytecode Manipulation

2.5.1 Javassist

Chapter 3

Implementation

3.1 Tainting Rules

3.2 Detainting Rules

3.3 Benchmarking

Chapter 4

Result

Chapter 5

Discussion

5.1 Domain Driven Security

Chapter 6

Future Work

Chapter 7

Conclusion

Bibliography

- [1] Johan Arnör. “Domain-Driven Security’s take on Denial-of-Service (DoS) Attacks”. In: (2016), p. 54. URL: <http://kth.diva-portal.org/smash/get/diva2:945831/FULLTEXT01.pdf>.
- [2] Steven C Bankes. “Tools and techniques for developing policies for complex and uncertain systems Introduction: The Need for New Tools”. In: (). URL: http://www.pnas.org/content/99/suppl%7B%5C_%7D3/7263.full.pdf.
- [3] Justin Clarke-Salt. *SQL Injection Attacks and Defense, 2nd Edition*. eng. Syngress, June 2009. ISBN: 9781597499736.
- [4] Michael Cross. *Developer’s guide to web application security*. eng. Rockland, MA: Syngress Publishing, 2007. ISBN: 1-281-06021-6.
- [5] Eric Evans. *Domain-driven design reference: definitions and patterns summaries*. Dog Ear Publishing, 2015.
- [6] Eric Evans. *Domain-driven design : tackling complexity in the heart of software*. eng. Boston, Mass.: Addison-Wesley, 2004. ISBN: 0-321-12521-5.
- [7] Seth Fogie. *XSS attacks cross-site scripting exploits and defense*. eng. Burlington, MA: Syngress, 2007. ISBN: 1-281-06024-0.
- [8] Vivek Haldar, Deepak Chandra, and Michael Franz. “Dynamic Taint Propagation for Java”. In: (). URL: <https://pdfs.semanticscholar.org/bf4a/9c25889069bb17e44332a87dc6e2651dce86.pdf>.
- [9] Rabia Jilani et al. “ASCoL: A Tool for Improving Automatic Planning Domain Model Acquisition”. In: *AI*IA 2015 Advances in Artificial Intelligence*. Ed. by Marco Gavanelli, Evelina Lamma, and Fabrizio Riguzzi. Cham: Springer International Publishing, 2015, pp. 438–451. ISBN: 978-3-319-24309-2.

- [10] Dan Bergh Johnson. *Dear Junior - Letters to a Junior Programmer: Introducing Domain Driven Security*. 2009. URL: <http://dearjunior.blogspot.se/2009/09/introducing-domain-driven-security.html> (visited on 01/25/2018).
- [11] *Locking Ruby in the Safe*. URL: <http://ruby-doc.com/docs/ProgrammingRuby/html/taint.html> (visited on 01/25/2018).
- [12] Jianan Ma. "TaintDroid : An Information- - Flow Tracking System for Realtime Privacy Monitoring on Smartphones Jianan Ma Problem Contribution / Implementation Details Questions / Suggestions". In: (2010), p. 3.
- [13] FIX ME!!! *Secure by Design*. FIX ME!!!, 2018.
- [14] Scott Millett. *Patterns, principles, and practices of domain-driven design*. Wrox, a Wiley brand, 2015.
- [15] Open Web Application Security Project. OWASP. URL: https://www.owasp.org/index.php/Main%7B%5C_%7DPage (visited on 02/01/2018).
- [16] 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.
- [17] 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.
- [18] *perlsec - perldoc.perl.org*. URL: <http://perldoc.perl.org/perlsec.html> (visited on 01/25/2018).
- [19] *Same Origin Policy - Web Security*. URL: https://www.w3.org/Security/wiki/Same%7B%5C_%7DOrigin%7B%5C_%7DPolicy (visited on 02/07/2018).

- [20] Jonas Stendahl. "Domain-Driven Security". In: (2016), p. 39. URL: <http://kth.diva-portal.org/smash/get/diva2:945707/FULLTEXT01.pdf>.
- [21] Praveenkumat H Subbulakshmi T. *Secure Web Application Deployment Using Owasp Standards: An Expert Way of Secure Web Application Deployment*. Createspace Independent Publishing Platform, 2017.
- [22] 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.
- [23] Johan Wilander. *OWASP Sweden: Domändriven säkerhet / Domain-Driven Security*. 2009. URL: <http://owaspsweden.blogspot.se/2009/09/domandriiven-sakerhet-domain-driven.html> (visited on 01/25/2018).
- [24] Jingling Zhao et al. "Dynamic taint tracking of web application based on static code analysis". In: *Proceedings - 2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2016* (2016), pp. 96–101. DOI: 10.1109/IMIS.2016.46.