Applying Dynamic Taint Propagation in Order to Enforce Domain Driven Security

FREDRIK ADOLFSSON

Master in Computer Science
Date: March 6, 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

Chapter 1

Introduction

One of the greatest strengths with deploying applications on the World Wide 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 whishes 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. [18, 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 application and propagates onto the other variables it comes in contact with. It is possible to detaint 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. [20, 25] 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 objects are correct. [26, 13]

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* in the field and

a *Delimitations* section. Lastly, is there a section about the *Methodology* behind the thesis.

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 defined in this section.

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

Definition 1.1.2. Web Application is a application deployed with accessability from the web.

Definition 1.1.3. Taint is denoting a flag that is attached to values indicating that the value might be of possible harm to the application.

Definition 1.1.4. Detaint denotes the process of removing the taint flag from a value and therefore marking the value as safe to the application.

Definition 1.1.5. Domain is explained in Secure by Design [16] as part of the real world where something happens.

Definition 1.1.6. 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 trough 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 implementation 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 Haldar, Chandra, and Franz [9] and Zhao et al. [27]. Both reports have implemented a Dynamic Taint Propagation tool for web deployed Java applications where the manipulation of the Java bytecode is done trough 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 [23].

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 Analyser.

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 chapter starts with a general description about *Web Application* structure and is followed by a section with the two most common *Security Vulnerabilities* to a web application. After those sections follows two sections describing *Dynamic Taint Propagation* and *Domain Driven Security*. The last section is a chapter about *Java Instrumentation*.

2.1 Web Application

To make applications available for a large set off people and also make the 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 a three tier architecture. 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] An illustration of the three layer architecture can be seen in figure 2.1.

As can be seen in figure 2.1 dose the tiers only communicate with the tire closest to itself. This causes the second tier to become a safe guard for tier three where the valuable and possibly sensitive information is stored. The storage tier contains all the information the application needs to provide the wanted service. Such information might

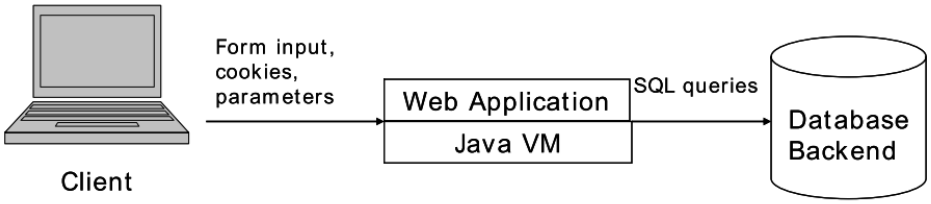

Figure 2.1: Web application structure [9].

for example be name, email, personal number and credit card information. [3]

2.1.1 Structured Query Language

The communication between tier two and tier three is done trough a standardize language called Structured Query Language, mostly known as SQL. SQL is created to programmatically manipulate and access databases. The vast majority of todays database uses SQL. The language works by building queries specifying the wanted information or task. The query will be evaluated and handled up upon by the SQL engine. [5]

2.2 Security Vulnerabilities

The organization Open Web Applications Security Project, mostly known for its shortening (OWASP), is a online community which aim to provide knowledge how to to secure web applications. [18] OWASP have produced reports about the top 10 security risks with a web application and the latest was published 2017. 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. This thesis will look at security risk number one and eight which is Injection Attacks and Cross-site Scripting. [19]

2.2.1 Injection

The most common security risk is Injection Attacks. [19] A Injection Attack is any attack where the attacker's input changes the intent of

the execution. Common result of Injection Attacks are file destruction, lack of accountability, denial of access and data loss. [24]

Injection Attacks can be divided into two different subgroups. These two subgroups are SQL Injection and Blind SQL Injection. [24]

SQL Injection

SQL Injection is when a SQL query is tampered which results in gaining content from the database which was not intended. Listing 2.1 displays a SQL Query which is open to SQL Injections. This is due to the fact that the variable UserId is never validated before it is propagated into the query. [3, 24]

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 valid Integer (since Integer is what we have decided that user id is in this application). 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 query that always evaluates to true. The result of this will be that the query returns the whole table of users. This problem can be prevented in a couple of different ways. The first is through validation of the input. By verifying the input as seen in listing 2.3 can we protect the query from being assessable to SQL Injection.

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. A 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 a SQL query that tells the SQL engine to sleep depending on a condition. A example of this can be seen in listing 2.5. [3, 24]

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, which is by analyzing the error messages, and depending on what they return build a image off the wanted answer. This is mostly done by testing different combination of true and false questions. [3, 24]

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 was created just after the release of JavaScript. The attack was conducted trough loading a malicious web application into a frame on the site that the attacker want to gain information of. The attacker could then through

JavaScript access any content that is visible or typed into the web application. To prevent this form of attack were the standard of Same-Origin Policy introduced. Same-Origin Policy restricts JavaScript to only access content from its own origin. [8, 22]

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 rooms where it was possible to inject malicious scripts 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. [8]

XSS can be divided into three different sub categories. Which are: reflected, Stored and DOM Based XSS.

Reflected XSS

Reflected XSS is mostly conducted trough a malicious link that an unknowing user clicks. The malicious link will exploit a vulnerable input on the targeted web application and though the input reflect back content to the user. [24]

Stored XSS

Stored XSS is when malicious scripts is stored in the targeted web applications database. This malicious script is then loaded and presented for each user which is trying to access the application. [24]

DOM Based XSS

DOM Based XSS is very similar to Reflected XSS but it dose not necessary have to be reflected from the application server. DOM Based XSS modifies the DOM tree and is able to exploit the user. [24]

2.3 Dynamic Taint Propagation

Taint Propagation, also known as taint analysis and taint checking, is a tool to analyse the flow of information in a domain. [20] It works by giving input data a tainted property which follows the data and propagates onto other data which it comes in contact with. Tainted data can be detainted. This is done by validating the data. The taint property is later checked in security sensitive sinks. [20] Taint Propagation

can be done in two different ways: Static and dynamic. The static is a evaluation tool which is done statically and before runtime. Dynamic Taint Propagation is a tool that is executing in runtime. Dynamic Taint Propagation works by actively blocking any data that is trying to enter the sink. This thesis will not be handle the Static version of Taint Propagation.

Perl and Ruby are two programming languages which have adapted to user Dynamic Taint Checking. [21, 14] And there are some tools who enables taint checking for other languages such as TaintDroid [15] and FlexTaint [25].

As named earlier is two of OWASP's top 10 application security risks of 2017 Injection Attack and Cross-Site Scripting (XSS). [19] Protection against these two attacks are best done by validating input data which Dynamic Taint Propagation helps to enforce.

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, 12] DDD is more of a thought process and methodology to follow every step of the process. [7] In *Domain-driven design reference: definitions and patterns summaries* do Evans [6] 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. [17] 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. [7]

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 develope the product is communicated back and forth between the two parties. [17] 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. [6]

Evans [6] 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. [17]

Wilander [26] and Johnsson [13] 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 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 that they are validated so they cant be populated with erroneous data. [26, 13, 1, 23]

2.5 Java Instrumentation

Java Instrumentation is a way to modify the execution of a application on the Java Virtual Machine (JVM) without having knowledge nor the need of modifying the application code itself. This makes it beneficial to implement for example monitoring agents and event loggers. Java Instrumentation works by modifying the bytecode of the JVM. [10] There exists a number of libraries that can help the developer in the task of instrumenting the JVM. The one that will be used in this thesis is called Javassist. Javassist provides a library that simplifies the task of editing and creating java class file at runtime. [11]

2.5.1 JVM

2.5.2 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] Cristian Darie. *The Programmer's Guide to SQL*. eng. 2003. ISBN: 1-4302-0800-7.

[6] Eric Evans. *Domain-driven design reference: definitions and patterns summaries*. Dog Ear Publishing, 2015.

[7] Eric Evans. *Domain-driven design : tackling complexity in the heart of software*. eng. Boston, Mass.: Addison-Wesley, 2004. ISBN: 0-321-12521-5.

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

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

[10] *Instrumentation (Java Platform SE 8)*. URL: https://docs.oracle.com/javase/8/docs/api/java/lang/instrument/Instrumentation.html (visited on 02/27/2018).

[11] *Instrumentation (Java Platform SE 8)*. URL: https://docs.oracle.com/javase/8/docs/api/java/lang/instrument/Instrumentation.html (visited on 02/27/2018).

[12] 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.

[13] Dan Bergh Johnsson. *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).

[14] *Locking Ruby in the Safe*. URL: http://ruby-doc.com/docs/ProgrammingRuby/html/taint.html (visited on 01/25/2018).

[15] 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.

[16] FIX ME!!! *Secure by Design*. FIX ME!!!, 2018.

[17] Scott Millett. *Patterns, principles, and practices of domain-driven design*. Wrox, a Wiley brand, 2015.

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

[19] 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.

[20] 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.

[21] *perlsec - perldoc.perl.org*. URL: http://perldoc.perl.org/perlsec.html (visited on 01/25/2018).

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

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

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

[25] 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.

[26] Johan Wilander. *OWASP Sweden: Domändriven säkerhet / Domain-Driven Security*. 2009. URL: http://owaspsweden.blogspot.se/2009/09/domandriven-sakerhet-domain-driven.html (visited on 01/25/2018).

[27] 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.