Applying dynamic taint propagation in order to enforce domain driven security

FREDRIK ADOLFSSON

Master in Computer Science
Date: February 5, 2018
Supervisor: Musard Balliu
Examiner: Mads Dam
Swedish title: Tillämpa dynamic taint propagation för att genomdriva domändriven säkerhet
School of Computer Science and Communication

Abstract

Sammanfattning

Contents

Chapter 1

Introduction

1.1 Problem

1.2 Aim

1.3 Definitions

Definition 1.3.1. Domain

Definition 1.3.2. Domain Model

Definition 1.3.3. Value Object

1.4 Delimitations

1.5 Methodology

Chapter 2

Background

One of the greatest problems, but in the same time greatest strength, with deploying applications on the web is that they are accessible from everywhere where there exists internet access. This means that they are also easily accessible for people who whishes to harm or abuse the application. Two of the more common security risks for a web application is Injection Attack and Cross-Site Scripting. [12, 4]

2.1 Web Application

The most common architectural design for a web application is based on three tires. 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]

LÄGG TILL BILD PÅ STRUKTUREN

As can be seen in picture **BILD** dose the tires 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]

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. [12] 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. [13] This report will look at the number one and eight security risks of 2017 which is injection attacks and cross-site scripting. [13]

2.2.1 Injection

The most common security risk is Injection Attacks. [13] 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. [17]

There is two kinds of different Injection Attacks. These two are SQL Injection and Blind SQL Injection. [17] 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, 17]

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 sins the second or parameter is always true. This problem can be prevented in a couple of different ways. The first is trough 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 trough 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 trough 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 thid can be seen in listing 2.5. [3, 17]

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, 17]

2.2.2 Cross-site Scripting

2.3 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. [14] 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. [14]

Perl and Ruby are two programming languages which have adapted to user dynamic taint checking. [15, 9] And there are some tools who enables taint checking for other languages such as TaintDroid [10] and FlexTaint [18].

Two of OWASP top 10 application security risks of 2017 is Injection and Cross-Site Scripting (XSS). [13] 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 Design

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, 7] 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 trough 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. [11] 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 develope the product is communicated back and forth between the two parties. [11] 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. [11]

2.4.1 Domain Driven Security

Wilander [19] and Johnsson [8] 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 Deign (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. [19, 8, 1, 16]

Chapter 3

Implementation

3.1 Plain (Bad name)

3.2 Taint Propagation?

3.3 Domain Driven Security

Chapter 4

Result

Chapter 5

Discussion

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] 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.

[8] 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).

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

[10] 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.

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

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

[13] 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.

[14] 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.

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

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

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

[18] 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.

[19] 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).