

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Morphing Web Pages to Preclude Web Page Tampering Threats

Luís Pedro Borges Abreu



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Luís Filipe Pinto de Almeida Teixeira

Second Supervisor: Paulo Alexandre Pinheiro da Silva

July 11, 2016

Morphing Web Pages to Preclude Web Page Tampering Threats

Luís Pedro Borges Abreu

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Jorge Manuel Gomes Barbosa

External Examiner: André Ventura da Cruz Marnoto Zúquete

Supervisor: Luís Filipe Pinto de Almeida Teixeira

July 11, 2016

Abstract

The number of Internet users keeps growing every year. Moreover, the Internet is becoming a daily tool, which impacts the individual's lives, used either as a work tool or for entertainment purposes. However, by using it, people become possible targets for cyber attacks as they keep exchanging data, sometimes sensitive and private data, with remote servers.

Among all the different attacks types, MitB is the reason behind the genesis of this thesis subject. MitB attacks are performed by a computer program running on user's computer that is commonly known as Malware and has access to what happens inside a browser window. It can be a system library or even a browser extension programmed to, automatically, misrepresent the source code of the client-side server response, and other information stored in user's browsers. They rely on markup and DOM anchors to identify sections of a web page to attack. The end result of an attack will be dictated by the malware's ability to successfully identify the right location on the web page to perform the attack.

Polymorphism is a broad concept that can be applied to web pages as a tool to both neutralize and defeat such kind of attacks, as documented by Shape Security, Inc. in 2014. By applying polymorphic techniques to web pages, server responses will be textually different between requests, while the visual display to the user will always be the same. That is, the values of static attributes and the structure of HTML documents may be modified on the server immediately before responses are sent off, creating a polymorphic version of the web page, or by pre-building these new versions on the server to decrease the real time computational costs. Therefore, no two HTML documents will be textually the same, turning web pages into somehow a moving target against MitB attacks. This level of protection is necessary since all attacker's changes are made locally, client side, making their detection difficult by control and security structures implemented on the service provider's servers.

In this thesis, we aim to develop a tool to apply polymorphism to protect web pages and users from MitB attacks based on markup and DOM anchors. This tool will be evaluated by accuracy and efficiency. The first metric will be evaluated by recording and comparing the list of errors and warnings generated by original web pages and by their polymorphic versions created with our tool. The efficiency will be evaluated by running automated attempts for tampering web pages protected by our tool.

Resumo

O número de utilizadores da Internet continua a aumentar todos os anos e a Internet é cada vez mais uma ferramenta diária na vida de cada indivíduo, utilizada como instrumento de trabalho ou de entretenimento. Contudo, ao navegar na Internet, os utilizadores tornam-se possíveis alvos de ataques informáticos um vez que efetuam transações de dados, muitas vezes privados e sensíveis, com servidores remotos.

Entre os diferentes ataques informáticos existentes, destaca-se o ataque MitB que deu origem ao tema desta dissertação. Os ataques MitB são realizados com recurso a Malware instalado e em execução nos computadores dos utilizadores, que tem acesso às informações das janelas dos navegadores de Internet - por exemplo através de bibliotecas de funções do sistema operativo ou até recorrendo a extensões dos navegadores de Internet. Estes ataques utilizam âncoras do DOM para identificar as secções de uma página web onde pretendem atacar - recolhendo dados ou modificando a própria página. O resultado do ataque será diretamente influenciado pela capacidade do Malware em identificar os pontos de ataque numa determinada página web.

O Polimorfismo é um conceito geral que pode ser aplicado a páginas web como uma ferramenta para neutralizar e derrotar este tipo de ataques informáticos, tal como foi documentado pela empresa Shape Security, Inc. em 2014. Aplicando técnicas de polimorfismo a páginas web, as respostas de um servidor serão textualmente diferentes entre si, mas o resultado visual apresentado ao utilizador será sempre o mesmo. Concretamente, os valores dos atributos estáticos e a estrutura dos documentos HTML poderão ser modificados no servidor, criando assim versões polimorfas de uma página web. Estas transformações podem ser realizadas em tempo real no servidor ou pré-calculadas. Desta forma, nunca dois documentos HTML serão textualmente iguais, tornando as páginas em alvos em movimento, dificultando os ataques MitB. Este nível de proteção é necessário uma vez que todas as alterações da página realizadas pelo atacante são locais e portanto difíceis de detectar pelas estruturas de segurança e controlo implementadas nos servidores dos fornecedores dos serviços.

Nesta dissertação pretende-se criar uma ferramenta capaz de aplicar técnicas de polimorfismo a páginas web para proteger as mesmas e os seus utilizadores contra ataques MitB. A ferramenta desenvolvida será avaliada de acordo com a qualidade do código gerado com as transformações, e será ainda avaliada por eficiência contra ataques MitB que recorrem a âncoras da markup e do DOM.

Contents

1	Introduction	1
1.1	Context	1
1.2	Problem	3
1.3	Motivation	4
1.4	Objectives	4
1.5	Challenges	4
1.6	Document Structure	5
2	Polymorphism	7
2.1	Definition	7
2.1.1	Reference Polymorphism	8
2.1.2	Structure Polymorphism	8
2.2	Challenges	9
2.3	Acknowledgements	9
3	Web Page Morphing Techniques	11
3.1	ShapeShifter by Shape Security, Inc.	11
3.1.1	Polymorphism	12
3.1.2	PolyRef	13
3.1.3	PolyRef vs. Our Solution	15
3.2	Registries of Intellectual Property	15
4	Software Requirements and Architecture	17
4.1	System Requirements	17
4.2	User Requirements	19
4.3	Software Architecture	21
5	Prototype Implementation	25
5.1	Methodologies	25
5.2	HTML Attributes' List - Anchors	26
5.3	Renaming Function	27
5.3.1	Encryption Algorithm as a Name Generator	27
5.4	MorpherApp	28
5.4.1	MorpherWWW - Configurations Web Interface	29
5.4.2	Tools Configurations	30
5.5	Reference Polymorphism Application	30
5.5.1	HTML documents	31
5.5.2	CSS files	31

CONTENTS

5.5.3	JavaScript scripts	32
5.6	Structure Polymorphism Application	34
5.6.1	Node Replication	34
5.6.2	Node Wrapping	36
6	Results	39
6.1	Rendering Results	39
6.2	MorpherApp Performance	40
6.3	Web Page Loading Time	40
6.4	Tamper Attempts	41
7	Conclusions and Future Work	47
	References	51
A	Example Web Page Source Code	55
A.1	Original Source Code	55
A.2	Morphed Source Code	56
B	Testing Scripts	59

List of Figures

3.1	Example of a field alternation sequence. ¹	13
3.2	Multiple fields stacked display. ²	15
3.3	POST request with a value distributed across multiple fields (in blue), containing a hidden field (name in green) with the encrypted constant (in red). ³	16
4.1	Use case diagram. The MorpherApp represents our application that will apply polymorphism techniques to a set of files. The MorpherWWW is the web interface created for users to manage configurations.	20
4.2	MorpherApp class diagram.	21
4.3	Prototype activity diagram.	22
5.1	Renaming function example of transformation and reverse functions results.	28
5.2	MorpherApp's prototype activity diagram.	29
5.3	CSS box model. [1]	37
6.1	Rendering result of an example web page.	39
6.2	Rendering result of the web page in figure 6.1.	40
6.3	Box plot for the homepage loading time. The plot was created with a sample of 1000 accesses for each server mode: original and polymorphic. The polymorphic mode use a set of 100 different versions for the same homepage.	42
6.4	Dashboard of TamperMonkey. In the dashboard, all scripts created can be controlled to be inject into their targets or not.	43
6.5	Result for the attack performed by the script (1) on the original web page. On the right is the web page HTML source code and on the left is its visual display.	44
6.6	Result for the attack performed by the script (1) on a morphed version of the web page in 6.5 . On the right is the web page HTML source code and on the left is its visual display.	44
6.7	Result for the attack performed by the script (2) on the original web page. On the right is the web page HTML source code and on the left is its visual display.	45
6.8	Result for the attack performed by the script (2) on a morphed version of the web page in 6.7 . On the right is the web page HTML source code and on the left is its visual display.	45
6.9	Result for the attack performed by the script (3) on the original web page. On the right is the web page HTML source code and on the left is its visual display.	46
6.10	Result for the attack performed by the script (3) on a morphed version of the web page in 6.9 . On the right is the web page HTML source code and on the left is its visual display.	46

LIST OF FIGURES

List of Tables

3.1	Example of reference polymorphism defined by Shape Security, Inc. ⁴	13
3.2	Example of JavaScript consistent changes and JavaScript extended polymorphism. ⁵	14
3.3	Basic information on the relevant registries of intellectual property found. ^{[2][3][4]}	16
4.1	List of Requirements to be implemented in our solution.	23
4.2	JavaScript methods and CSS selectors that may return unexpected results after structure polymorphism to a web page.	24
5.1	Group of CSS attribute's selectors that cannot be transformed without knowing which HTML elements in a web page create a match with them.	31

LIST OF TABLES

List of Codes

1.1	Facebook web inject by Zeus. ⁶	3
5.1	Content of the Anchors file used by the built solution prototype.	26
5.2	Redefinition of the <code>document.querySelectorAll</code> method. The "lu.get(a)" is our function that processes a query to rename all attributes in it.	34
A.1	HTML source code of the example web page where the tests in B were executed.	55
A.2	HTML source code of one morphed version of the example web page where the tests in B were executed.	56
B.1	Script created to execute the first test.	59
B.2	Script created to execute the second test.	60
B.3	Script created to execute the third test.	61

LIST OF CODES

Abbreviations

AES	Advanced Encryption Standard
API	Application Programming Interface
AST	Abstract Syntax Tree
BHO	Browser Helper Object
CSS	Cascading Style Sheet
DLL	Dynamic Linked Library
DOM	Document Object Model
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
MitB	Man-in-the-Browser
SSL	Secure Sockets Layer
URL	Uniform Resource Locator

Chapter 1

Introduction

This chapter is used to expose the context of this thesis and to state the problem aimed to be solved. The main objectives, motivations, and expected results are also presented. Finally, the probable challenges to face during the development of the project are described and this chapter ends by explaining how this document is structured.

1.1 Context

Internet users are exposed to multiple threats and cyber attacks every day. Malware is one of these threats, which is classified according to their actions and targets.

Malware has three basic elements: an infection point (where the user gets initially infected), a control and command server (the point from where an attacker can send new configurations and updates for the malware and send other commands and instructions to be executed on targeted computers), and a drop server (this is where all the stolen data from the user is stored - like screenshots, passwords, etc.). To be able to take any action, first, a computer needs to be infected. Malware can be an email attachment, a program downloaded from the Internet, an infected USB key, an infected link on social networks, among many others. Once the malware is installed, the computer becomes compromised and vulnerable to be used as a bot to perform cyber attacks[4].

One of the rising attacks on the Internet using malware, especially to attack banking sessions, is called Man-in-the-Browser. *"Structurally they are a man-in-the-middle attack between the user and the security mechanisms of the browser."*¹

This attack uses a specific type of malware called Trojan, which is capable of infecting Internet browsers and recording all data stored inside them and all the data entered by users.

These trojans can also tamper transactions between browsers and web servers by modifying HTTP requests created by browsers and HTTP responses received by them. With this action, they

¹Retrieved from [5].

Introduction

are able to modify the content displayed to the user by changing, adding or removing content in web pages.

*"(...) these new attacks cannot be detected by the user at all, as they are using real services, the user is correctly logged in as normal, and there is no difference to be seen."*²

MitB attacks take place locally, in users' computers at the Application Layer. That is the reason why the security measures implemented by web services are not effective to stop MitB attacks. There is more than one method to access browser's data, which implies more than one way for malware to perform these attacks. Browser's extensions and BHOs are two methods with permissions to access browsers' data. One kind of malware can intercept HTTP requests in the browser before they are sent to the network and intercept HTTP responses after they are received in the browser. This is browsers' extensions scenario. Other types of malware like Zeus are able to intercept the same requests by DLL hooking. In this case, the HTTP requests are intercepted outside the browser and before being sent to the network and HTTP responses are intercepted after being received on the computer but before the message can arrive in the browser.

These MitB attacks are automated attacks that happen when the user performs relevant actions, such as authenticating in an online banking web page.

*"The new breed of new trojan horses can modify the transactions on-the-fly, as they are formed in browsers, and still display the user's intended transaction to her."*³

MitB is becoming a popular method because it's very easy for a user to be infected and because malware fulfills his attack after legitimate authentication of users (when an authentication is needed).[6]

Zeus is an example of a trojan used to create a giant criminal network to steal money from different bank accounts. This trojan appeared for the first time in 2007 in the United States.

*"The cyber thieves [owners of Zeus] targeted small- to medium-sized companies, municipalities, churches, and individuals, infecting their computers using a version of the Zeus Botnet. The malware captured passwords, account numbers, and other data used to log into online banking accounts. This scheme resulted in the attempted theft of \$220 million, with actual losses of \$70 million from victims' bank accounts."*⁴

Nevertheless, later in 2011, the source code of Zeus was leaked on the Internet and the original Zeus continues to evolve to new forms as Chthonic, a Zeus-based trojan detected in the fall of 2014.

To demonstrate how DOM anchors can be used to perform the web injects, the code in 1.1 is one web inject executed by Zeus on infected machines . Every time a user on an infected machine

²Retrieved from [5].

³Retrieved from [5].

⁴Retrieved from [7].

Introduction

with Zeus opens the Facebook page, Zeus will search the HTML tag defined in "data_before" and after he will inject the code between "data_inject" and "data_end". This example was carefully chosen to also expose a flaw on our polymorphism approach: HTML tags can not be modified (otherwise, browsers won't be able to understand the document!) which means it is still possible to inject code into web pages based on default HTML tags like "head" and "body".

```
1 set_url http://www.facebook.* G
2 data_before
3 </head>
4 data_end
5
6 data_inject
7 <script type="text/javascript">var isloaded = false;</script>
8 <script type="text/javascript"
9     src="http://clicktracking.biz/gwjs.php?pub=139622&gateid=MTc4MDUw">
10 </script>
11 <script type="text/javascript">
12     if (!isloaded) { window.location = 'http://clicktracking.biz/abp'; }
13 </script>
14 <noscript>
15     <meta http-equiv="refresh" content="0;url=http://clicktracking.biz/java"/>
16 </noscript>
17 data_end
```

Code 1.1: Facebook web inject by Zeus.⁵

1.2 Problem

HTML injection is one of the exploits where malware is able to modify the code of a web page by adding, removing or modifying content using markup and DOM anchors to identify the places to strike. By doing so, malware can add, for example, new forms or new fields to original forms in order to get more information from users (enabling attackers to act on user's behalf) or inject advertisement banners or other ways of changing web pages' display. Hence, it's of great importance to protect users not only against computer's infection but also to protect them against attacks performed on their own infected machines.

Due to the static nature of HTML documents, MitB attacks are effective in the way they can statically refer to a location in the document and from there, add, remove or modify all or parts of the document. By giving HTML documents' polymorphic characteristics, they become a "moving target" which increases the level of security of a web page against tampering threats.

In this thesis, we aim to preclude web page tampering threats by reducing and neutralizing some of the malware's power used in MitB attacks when using DOM and markup anchors.

⁵Code extracted from goo.gl/szOKAC.

1.3 Motivation

Malware has been changing rapidly over the years while client software is having trouble keeping up. This leads to a lack of effectiveness of Anti-Virus and Anti-Malware solutions, and thus a need for new security measures arises.[8]

The company Jscrambler S.A. proposed the subject for this thesis that is a solution they are interested in studying and adding to their product and so this was an opportunity to work in a professional environment while finishing my Master's degree.

The Internet can be a huge place to work and to entertain but is also a place with a lot of threats. Security is one of my concerns especially when it comes to unskilled or undisciplined end users. Every day I see my parents using computers and the Internet to work. I am always trying to advise and teach them with my knowledge to protect them a bit more from being exposed to online threats. But it's a hard task, and despite all my efforts their computers are many times infected with malware. It's thinking about them and in all the other users that I find my motivation for this thesis.

1.4 Objectives

The main objective of this thesis is to add a new security measure to websites, so end users are more protected against MitB attacks that take advantages from markup and DOM anchors while using infected computers to access the Internet. To accomplish this, two types of polymorphic strategies are presented: reference polymorphism and structure polymorphism. Each one is explained and detailed in chapter 2. Briefly, reference polymorphism aims to defeat DOM attacks using simple anchors like `div#myId` and structure polymorphism intends to defeat more advanced DOM attacks that locate anchors by searching their position on the document rather their specific attributes and values.

1.5 Challenges

This project is very ambitious and represents a lot of challenges to solve. The concept of polymorphism applied to web pages is easy to define and describe, yet not so simple to implement and to apply. There are a lot of questions and variables to take into account first.

Every request a server receives generates a response. When a user requests a specific web page, the response should be an HTML document representing the requested web page. Polymorphic techniques are to be applied to every response before leaving the servers. How can this technique be applied with the minimum impact on the user's experience and on the business model of a web page? How will the server store the information about the transformations made so it can understand later requests? Or will the server be stateless and leave the storage of the information to each client? For example, when modifying the value of the attribute `name` of a HTML `input` tag,

the server needs to be aware of the correspondence between the original value and the new value to be able to translate it to the correct one when the users submit that form.

Changing static values on HTML documents implies also changes on CSS and JavaScript scripts so the web page is able to keep its visuals intact and its interactions through JavaScript working. This is another challenge: the changes must be in accordance with documents and scripts.

1.6 Document Structure

This document is divided into different chapters. To better understand this thesis and its concepts is essential to detail and expose in the first place what polymorphism means to us and to our solution. In chapter 2, we present this concept and other two derived concepts (reference polymorphism and structure polymorphism) we propose to implement as a solution for this thesis' problem. At the end of the chapter, we present important notes related to expected challenges when implementing these concepts and also some acknowledgments on the flaws of this solution.

Then, in chapter 3 are presented the solutions found according to this thesis problem and the wanted solution to solve it. As this project is being developed in collaboration with a software security company, we also present some information on the registries of intellectual property found.

In order to develop a solution prototype for the stated problem, the software requirements and the architecture of the application were defined and are exposed in chapter 4. Following this, in chapter 5 are presented the details of the prototype's implementation and then in chapter 6 we describe the results obtained with the developed prototype.

Lastly, in chapter 7 are presented the conclusions and perspectives for future work.

Introduction

Chapter 2

Polymorphism

This chapter aims to introduce the concept of polymorphism applied to web pages in order to preclude web page tampering threats. We bring forward two types of techniques to morph web pages: reference polymorphism and structure polymorphism. At the end of the chapter, the challenges and the flaws of these techniques are exposed.

2.1 Definition

According to the Oxford Dictionary of English, polymorphism is a noun that stands for "*the condition of occurring in several different forms (...)*"¹. This represents a generic concept used in different fields to define a more specific condition. In this thesis, the term polymorphism is applied to web code written to create web pages. This is just a concept molded to this case. Even when applied to web code, this word can take different specific meanings, but maintaining always the core definition of the word.

In this document, the word polymorphism is used to define the ability of a web page to be transformed into different versions (textually different) while always maintaining the same visual display and functionalities expected by end users. The only thing that changes is the code needed to present a web page, which means the changes can happen not only in HTML documents but also in CSS files and JavaScript scripts. It is extremely important to highlight that, if a web page is using CSS or JavaScript, the changes must be consistent when the information to be changed stands as a reference between the different files.

As described before, there is malware that works by making use of specific elements of a web page. With polymorphism, these elements are to be changed to have different values in order to make things more expensive for attackers.

Two types of polymorphism may be defined, according to the needs of transformation. The first is presented as reference polymorphism and it's related to elements' values. The second one

¹Extracted from Oxford Dictionary of English. Ed 2010. Oxford University Press.

is defined as structure polymorphism and it's related to the code structure and elements' order. Both types are also just concepts detailed in the next sections. Along with the description of the solution developed in this thesis, the implementation of these concepts are explained and detailed in a more technical way in chapter 5.

This is not an approach to preventing the infection but an approach to securing clients on compromised environments.

2.1.1 Reference Polymorphism

All the HTML elements can be valid anchors on the document. Moreover, each element can be detailed by using different attributes such as `id` and `class`. The more detailed, the more vulnerable to be reached using a markup or DOM anchor. The malware used in DOM attacks may need these anchors in order to find the targeted elements of a web page. Reference polymorphism stands as a method to rename all of these elements' values to obfuscate them and turn them into unpredictable new ones. This way, when applying this type of polymorphism to a web page, each version of the web page will be different in attributes' values. It is important to mention that reference polymorphism by itself does not defeat any attack and this method is only useful when used by a web server to provide always different versions of the original web page to every request. Only this way, reference polymorphism shows its power to defeat DOM attacks performed by MitB malware, as they are unable to find the wanted targets.

The transformations made in HTML documents should be extended to other files that may be needed by a web page, such as CSS and JavaScript files. These files should also be submitted to transformations in order to be compliant with previous changes in the HTML document. Only this way is ensured the new version of the web page is functioning as the original.

The transformations are by its very nature a rename of values. The rename function as a big role in all the process: it should produce new random values. Random because otherwise, attackers could be able to understand the patterns used in this function and reverse it, meaning they could break this security measure.

2.1.2 Structure Polymorphism

DOM attacks can be much more sophisticated than just simple and direct anchors. Indirectly, attackers can still find their targets in a web page by searching based on their position on the document. Reference polymorphism is not capable of defeating this kind of indirect use of anchors. To stop this more advanced DOM attacks, reshaping HTML documents could be one solution. Again, like before, it is important to highlight that structure polymorphism will only be effective when used by a web server to provide always different versions of the original web page to every request.

Structure polymorphism is a method to reshape the structure of HTML documents. Reshaping may include dummy node injection and node wrapping. Specifically, what structure polymorphism

does is changing nodes position, cardinality, and nesting degree. Nevertheless, it should take into account that the final rendering of the web page should be kept the same.

Reference polymorphism and structure polymorphism may be used together to stop MitB malware from performing DOM attacks on a web page.

2.2 Challenges

The first challenge related to reference polymorphism is how to find all the elements that should be transformed and how to do it on CSS and JavaScript files accordingly. This is the first step to implementing this method and also the first challenge.

In relation to structure polymorphism, the biggest challenge is to define how the reshaping process should be done without causing rendering differences. New CSS rules may be needed to achieve this.

After the "how" the challenge becomes "when". When should the changes be made and what are the costs of both methods? Polymorphism is intended to improve the end users security when using a web page (whether the end users' machine is infected with MitB malware or not) and so it should not cause any significant impact on users experience. In the perfect scenario, polymorphism would be applied on-the-fly without any impact on a web service structure, response time and end users experience.

2.3 Acknowledgements

Although we see reference polymorphism and structure polymorphism as a solution for this thesis problem, it should be taken into account that other forms of polymorphism can be defined to refine and strengthen the whole concept of web page polymorphism to protect end users against MitB attacks.

Yet, the solution described to implemented in this thesis is not one hundred percent effective: the detected flaws are not on technique's design but in the HTML document structure. Although the elements `<html>`, `<head>` and `<body>` can be omitted from the HTML document, browsers will add them after parsing the document to create a DOM tree with the basic structure defined by W3C [9]. This means every web page will always have this elements and so, despite all the efforts done with reference and structure polymorphism, malware is still able to inject HTML code as a `<script>` tag by using this element as anchors. HTML injection is still possible at the end or the beginning of the tags `<head>` and `<body>`. All the other HTML tags can also be reached despite malware not being able to locate a specific element.

Finally, reference and structure polymorphism are introduced to defeat DOM attacks. Nevertheless, web pages are still vulnerable to HTTP attacks and GUI attacks, as MitB has a variety of methods to perform the wanted attacks.[10]

Polymorphism

Chapter 3

Web Page Morphing Techniques

In this chapter are presented and described the relevant solutions found and the registries of intellectual property that have an impact on this thesis. A solution was marked as relevant according to their objectives, techniques and actions performed and also the targets. In this thesis, we searched to find solutions to fight MitB attacks by applying transformations to a web page in order to serve the clients always with unpredictable code to represent the same web page. The majority of solutions found, were said to be able to defeat MitB attacks but they were not applying any transformations nor the word polymorphism to describe them was found. The intended solution to this thesis is also a solution to protect infected users. A lot of other solutions found were instead intended to protect users from the infection in the first place and thus were not included in this document.

There was only one company with the wanted approach - Shape Security, Inc. - and their technique is called PolyRef and is briefly described here.

3.1 ShapeShifter by Shape Security, Inc.

Shape Security, Inc. is an American cyber security company that aims to deploy software and hardware solutions to protect websites and its users against automated attacks.

In 2014, Shape Security, Inc. had his paper "Polymorphism as a Defense for Automated Attack of Websites" published in the book of the 12th International Conference of Applied Cryptography and Networks Security (ACNS). In this paper, the company presented their technique to defend websites against automated attacks. This technique is being used in their botwall solution - the ShapeShifterTM.

"With a ShapeShifterTM in-line, a website appears different to adversaries on every page view while remaining unchanged for normal human users. All forms of automation require the ability to operate a website, whether at the HTTP level, the DOM

*level, or the GUI level."*¹

PolyRef is the name given to their technique that uses polymorphism as a strategy of defence that "(...) impedes two types of attacks: HTTP attacks (...) and DOM attacks (...) "². ShapeShifterTM also uses other techniques to improve the security by deflecting GUI attacks.

It is firstly designed to be placed between a firewall and a web server so that, when a web page sent by the web server arrives, PolyRef will look for the target forms and will apply two different techniques: reference polymorphism and field polymorphism. After the changes, a new polymorphic version of the web page will be sent to the destination. When one of the modified forms is submitted, PolyRef will restore all the form field names back to their original values so the web server is able to understand the request.[11]

In a new version, the ShapeShifter Cluster is deployed adjacent to web traffic instead of being inline.

The polymorphic techniques deployed with the Shape Security, Inc. botwall are the relevant subject of study for this thesis. The following sections contain a description of PolyRef and how it works and also what "Polymorphism" means to this company. The information presented on PolyRef is based on their paper published in 2014.

3.1.1 Polymorphism

*"(...) mixed in with legitimate traffic is malicious synthetic traffic that rides on botnets to stuff credentials, hijack accounts, and scrap prices. Instead of guessing about the nature of traffic, Shape uses polymorphism to disable the mechanism by which malware, botnets, and scripts interact with your website and mobile APIs"*³

3.1.1.1 Reference Polymorphism

Shape Security, Inc. applies reference polymorphism only to HTML form elements. Form and field names and other element identifiers are replaced with random character strings preventing attackers from directly locating the targeted field. To keep the web application consistent, these changes are extended to JavaScript and CSS scripts. Similarly, polymorphism is also extended to JavaScript by replacing the symbols connected to HTML forms. (See tables 3.1 and 3.2)

This method is efficient against HTTP attacks and direct DOM attacks. However, advanced DOM attacks can indirectly find targeted fields and so defeat reference polymorphism.[11]

¹Retrieved from the homepage of Shape Security, Inc. Official Website on December 15, 2015.

²Extracted from [11].

³Retrieved from the homepage of Shape Security, Inc. Official Website on December 15, 2015.

⁴Based on an example presented on the homepage of Shape Security, Inc. Official Website on December 15, 2015.

⁵Example extracted from [11].

Table 3.1: Example of reference polymorphism defined by Shape Security, Inc.⁴

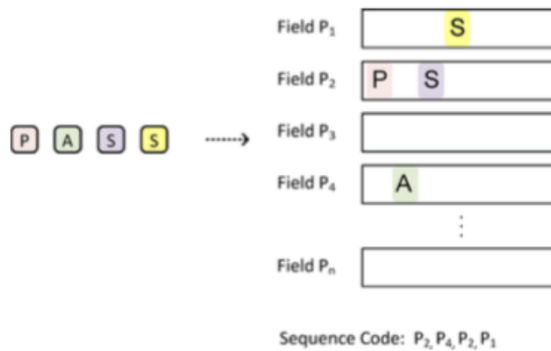
Original version
<code><input id="username" name="username"/></code>
Polymorphic version on a given load
<code><input id="bnoQtn2bH4mr" name="rWyt3L8Gj374"/></code>

3.1.1.2 Field Polymorphism

Field polymorphism is the technique by Shape Security, Inc. to impede advanced DOM attacks. In this technique, *"a field is broken into multiple fields"*⁶.

The objective is to create multiple fields to store the information of only one field in random parts. In the website users perspective, everything remains the same because *"all fields are stacked in the display and appear like one field"*⁷. This concept is illustrated with figures 3.1 and 3.2.

Behind the scenes, keystrokes are distributed between the multiple fields using a unique pattern defined for each page. As the user types, it changes the focused field. The alternation sequence is defined by a constant that is embedded into the dynamically generated JavaScript added to each page. This constant is encrypted by a shared key stored only in the PolyRef server and is required to reassemble the sequence. Finally, it is embedded in the return POST as a hidden field, as shown on figure 3.3.[11]

Figure 3.1: Example of a field alternation sequence.⁸

3.1.2 PolyRef

PolyRef was designed to be an HTTP proxy located adjacent to the web server. Every time a web page passes through this proxy, PolyRef will apply polymorphism as described before. Later,

⁶Extracted from [11].

⁷Extracted from [11].

⁸Figure extracted from [11].

⁹Figure extracted from [11].

¹⁰Figure extracted from [11].

Table 3.2: Example of JavaScript consistent changes and JavaScript extended polymorphism.⁵

Original version
<code>var username = document.getElementById("username").value;</code>
HTML reference polymorphism
<code>var username = document.getElementById("bnoQtn2bH4mr").value;</code>
JavaScript extended polymorphism
<code>var kBk5SjCT6p4t = document.getElementById("bnoQtn2bH4mr").value;</code>

when a form is submitted, the same proxy will process it and restore the key/value pairs of the form to the expected content by the server so the application is able to continue its operations.

It will transform every HTTP response containing a web page to its polymorphic version using the techniques described before. The changes to be made have to be consistent in order to keep the web page functionality intact. Hence, the transformations can be trivial if the page contains only plain HTML or can be more complex when containing also CSS and JavaScript.

It has two different phases. First, is the pre-computation phase in which PolyRef learns the relevant symbols to replace later. After, is the application phase in which reference polymorphism and field polymorphism are applied to web pages.

3.1.2.1 Pre-computation Phase

*"This phase is performed automatically the first time a new page is processed and then cached, and hence can employ heavyweight techniques."*¹¹

Before starting any automatic operation, a page profile should be manually configured (as for the first version of PolyRef) in order to identify the target forms where the transformations are desired. Only then, PolyRef is ready to start the pre-computation phase.

The first step is to parse the HTML document into a DOM tree structure used to find the desired forms. If the HTML document has JavaScript or CSS scripts embedded or linked to it, the next step is to identify all the references in these files to the target forms in order to keep consistency. In this step, scripts are parsed into ASTs to make sure each symbol is a property of a form or not. To complement this process, a static analysis is also performed to catch any variable referencing form fields. During this first step, variable consistency is kept by exploiting "(...) compiler optimization techniques such as constant folding and propagation (...)"¹².

Finally, after all references are identified, PolyRef is able to randomize all the symbols in a consistent way across HTML documents and JavaScript and CSS scripts. From this point, PolyRef can generate new polymorphic versions of web pages.

¹¹Retrieved from [11].

¹²Extracted from [11].

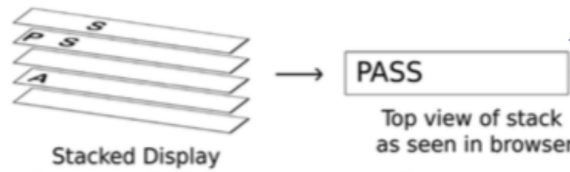


Figure 3.2: Multiple fields stacked display.⁹

3.1.2.2 Application Phase

After pre-computation, a web page is processed and cached and ready to be used by PolyRef to create a polymorphic version. This phase happens in real time for every HTTP response sent by the server. Since PolyRef already knows what and where to apply the techniques describe, this phase of the process is very straightforward. It will generate a polymorphic version of a web page and send it to the users browser to be displayed.

During the transformations, PolyRef will create a map between symbols and the randomized values to be used as a replacement. For each form, there will be a different map. Each map will be encrypted and added to the related form as a hidden field. When receiving an HTTP POST request, PolyRef decrypts the mapping and restores all names to their originals values ensuring the underlying server of the web application receives the expected values.^[11]

3.1.3 PolyRef vs. Our Solution

Despite the similarities, there are significant differences between the two approaches. PolyRef is to be applied to HTML forms only, hence not protecting the whole web page. Our solution aims to apply reference polymorphism to all the elements in a web page, being this, the first difference found. Another difference between the two is the technique structure polymorphism described as part of our solution. The PolyRef does not mention any technique to restructure the web page as a method of protection. Instead, PolyRef presents field polymorphism but only as a way to protect input fields. Finally, PolyRef mentions he's capable of deflecting DOM attacks and HTTP attacks. In our solution, HTTP attacks are out of the scope.

3.2 Registries of Intellectual Property

Since this thesis is being developed in collaboration with the Portuguese company Jscrambler, S.A. and the final results can lead to a new solution to be integrated into their product to be commercialized, the study of existing registries of intellectual property is a great matter of interest.

There are at least three patents registered to the same applicant - Shape Security, Inc. - where are described techniques to apply transformations on web pages as a strategy to defeat MitB attacks. These relevant registries found are presented on table 3.3.

Web Page Morphing Techniques

```
POST login.php HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 18

P1=S&P2=PS&P3&P4=A
KYTr29y7rhKJP6=QmFzZTY0IGVuY29aW5nIHNjaGVtZXMgYXIIIGNvbW1vbmx5IHVzZWQgd2hlbiB0aGVyZ
SBpcyBhIG5IZWQgdG8gZW5jb2RI
```

Figure 3.3: POST request with a value distributed across multiple fields (in blue), containing a hidden field (name in green) with the encrypted constant (in red).¹⁰

Shape Security Inc. has patented a technique to protect ordinary users against web exploits like MitB attacks. Their technique is said to be able to modify HTML, CSS and JavaScript code "(...) *before it is served over the Internet by a server system so as to make more difficult the exploitation of the server system by clients that receive the code (including clients that are infected without their users' knowledge).* (...) "¹³.

The main objective is to create a moving target by modifying a web page at each response served in order to deliver an unpredictable code that malware cannot use to interact with the content in a malicious way.

Table 3.3: Basic information on the relevant registries of intellectual property found.[2][3][4]

Title	International Publication Date	International Publication Number
Stateless Web Content Anti-Automation	25.09.2014	WO/2014/150659
Protecting Against the Introduction of Alien Content	25.09.2014	WO/2014/150569
Polymorphic Treatment of Data Entered at Clients	26.11.2015	WO/2015/179286

¹³Extracted from [12].

Chapter 4

Software Requirements and Architecture

Following the objectives referred in [1.4](#), we defined and detailed how we could achieve Reference Polymorphism and Structure Polymorphism in order to further develop the solution.

In this chapter are described all the requirements defined to be implemented in the final solution and also the guidelines selected for the development process, organized as system requirements and user requirements.

4.1 System Requirements

Since the beginning that the solution was expected to create a group of restrictions to potential users as they cannot rely on all JavaScript methods and CSS selectors (see [table 4.2](#)) to control their own web page because our approach will introduce changes in HTML documents structure. By this fact, the solution should be able to accept different parameters in order to allow the tuning of each functionality by its users so they can somehow control the output.

With reference polymorphism, we want to modify HTML anchors by renaming all of the elements' attributes so their values becomes dynamic as opposed to the static nature of the HTML markup. It was described in [chapter 2](#) that an attribute present in an HTML document should be transformed if and only if it's value is not pre-defined by the HTML standards. Following this, it is important to study the HTML standard elements so that we can safely distinguish what attributes are illegible to be submitted to transformations and the ones which are not.

Values' transformation is at its core a renaming process. The renaming function should strictly obey to a set of rules to ensure a proper and safe renaming. This function should be able to deliver new values compliant to HTML and CSS standard grammars in order to produce language compatible values. The output of this function should also be generated using a random password to allow the renaming to be always different for the same input when calculating a new web page morphed version. Lastly, the transformed values related to form parameters may need to be reversed on the server so that the information can be interpreted and processed.

Applying Reference Polymorphism to HTML documents and CSS files, using static analysis, is similar as in both cases all attributes found that are defined in the HTML Attributes List should be transformed by a Renaming Function. On the other hand, processing JavaScript scripts is a different scenario because at first we should search for DOM manipulation methods in order to understand what their arguments represent and only after this identification process, if an argument is related to an HTML attribute present in the HTML Attributes List, it should be submitted to the Renaming Function.

The identification process is very important as it will define what JavaScript methods will be parsed by our tool. A simple example to understand this need can be seen with two different document available methods: `getElementById("id")` and `querySelector("#id")`. The first method expects to receive an argument that is going to be used to search `id` values but the second method also allows to search for an `id` but in this case, the value should have the CSS symbol to identify ids - `#`. This example shows the importance of the identification process and also shows that each method's arguments should be handled in a specific way.

With structure polymorphism, we aim to reshape HTML documents using two approaches: Node Wrapping and Node Replication. Both approaches will increase the size of HTML documents as some nodes are going to be replicated and new nodes will be added. This technique will introduce a factor of randomness in these documents that can help deflect automated MitB attacks but can also create a challenge for developers to identify specific places in their own code. Another important fact is that not all web pages need the same factor of randomness, as others need extra protection is HTML elements like forms and input fields. For this reason, Node Replication and Node Wrapping should accept configurations from users to allow tuning the randomness factor and special cases to force randomness to be introduced.

Since polymorphism is to be applied to HTML documents, CSS files, and JavaScript scripts it was defined that one tool should be crafted to handle each type of file and also the different polymorphic techniques. This means, two different tools should be developed to handle HTML documents: one to apply reference polymorphism and the other to apply structure polymorphism.

Creating different tools also introduce a new requirement related to usability and user experience: all tools should allow to be used in a pipeline. Also, we want to define an interface to be adopted by all the tools so that all of them are used in the same way, with the same arguments and options and the same output format. This will help to create a feeling of belonging between the different tools and also to create the same user experience across all of them.

As stated in chapter 2, polymorphism is a broad concept that can be applied to web pages in many different ways. Hence, it makes sense to create a modular solution that allows adding new polymorphic techniques in the future and at some point let users select which modules they want to use.

The described requirements for architecture of the solution is one of the tenets of the Unix philosophy as stated by Mike Gancarz in its book "Linux and the Unix Philosophy":

"Small is beautiful. Small things have tremendous advantages over their larger counterparts. Among these is the ability to combine with other small things in unique and

useful ways, ways often unforeseen by the original designer."

*"Make each program do one thing well. By focusing on a single task, a program can eliminate much extraneous code that often results in excess overhead, unnecessary complexity, and a lack of flexibility."*¹

All the information regarding the requirements and the architecture of the solution were grouped and presented on the table [4.1](#).

4.2 User Requirements

Our main application, that should have integrated all the tools mentioned before, is defined to be able to provide users with a fully configurable application ready to use in order to apply polymorphism to a set of files representing a web page. Nevertheless, the application should be able to be integrated with other projects without dependencies between components so that users can choose the techniques they want to integrate into their own projects. With this two objectives in mind, a set of user requirements were defined and are listed below. The figure [4.1](#) shows the use case diagram for our solution.

1. The user should be able to apply polymorphism to a set of files in order to create a morphed version of the same set.
2. The user should be able to access the configurations panel in order to verify the current settings for the solution.
3. The user should be able to access the configurations panel in order to tune the application of polymorphism.
4. The user should be able to define the HTML attributes to transform in order to apply the renaming function where he needs.
5. The user should be able to select the number of morphed versions to create for a given set of files in order to get a set of morphed versions for one execution.
6. The user should be able to specify the extensions he uses for HTML documents, CSS files, and JavaScript scripts in order to instruct the application on how each file should be handled.
7. The user should be able to select the output directory where the application will save each morphed version created in order to control where the morphed versions are stored.
8. The user should be able to show or ignore the logs of the transformations applied to a set of files in order to visualize how the values are being transformed.
9. The user should be able to select what polymorphism techniques he wants to apply to a set of files in order to personalize the output of the application.

¹Extracted from [\[13\]](#).

10. The user should be able to apply Reference Polymorphism to JavaScript files by static analysis or dynamic treatment in order to choose the best option for each web page.
11. The user should be able to define the probability for Node Replication to occur in order to control the randomness of a given web page.
12. The user should be able to define the probability for Node Wrapping to occur in order to control the randomness of a given web page.
13. The user should be able to define the CSS class name for replicated nodes in order to apply the new style properties so that the visual display is not changed.
14. The user should be able to define the CSS class name for wrapping nodes in order to apply the new style properties so that the visual display is not changed.
15. The user should be able to force Node Replication of a group of HTML elements in order to create more confusion around more sensitive elements on a web page.

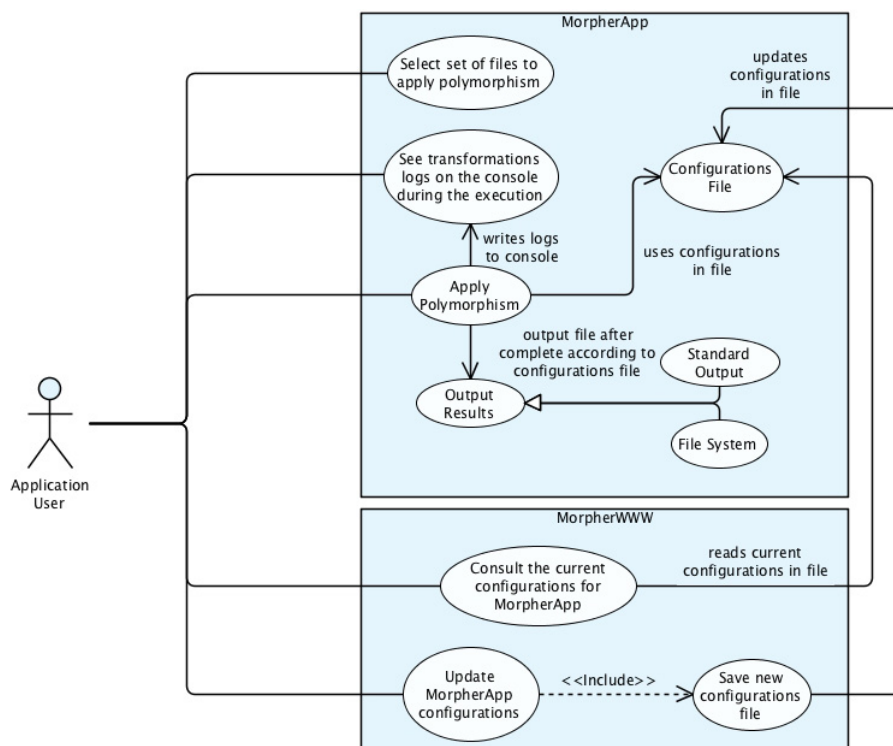


Figure 4.1: Use case diagram. The MorpherApp represents our application that will apply polymorphism techniques to a set of files. The MorpherWWW is the web interface created for users to manage configurations.

4.3 Software Architecture

As mentioned in 4.1 and 4.2 the solution should be built in a modular way to promote independence between the tools and at the same time allowing to create a fully configurable main application to orchestrate a pipeline for each type of file in a set of input files.

The main application for our solution will be called *MorpherApp* and will have the following components: (1) File Type Checker; (2) Name Generator; (3) Configurations; (4) Reference Polymorphism; (5) Structure Polymorphism; (6) Anchors. The third component is the only one that is dependent from *MorpherApp* as it is his configuration file with the information on how to operate and how to control the behavior of the other components. The renaming function and the HTML attributes' list will be (2) and (6). The component (1) will be used to distinguish each input file per type: HTML, CSS or JavaScript. Finally, the polymorphic techniques will be performed by (4) and (5) which are generalizations for the different handlers needed: HTML handler, CSS handler, JavaScript handler; HTML node wrapping; HTML node replication.

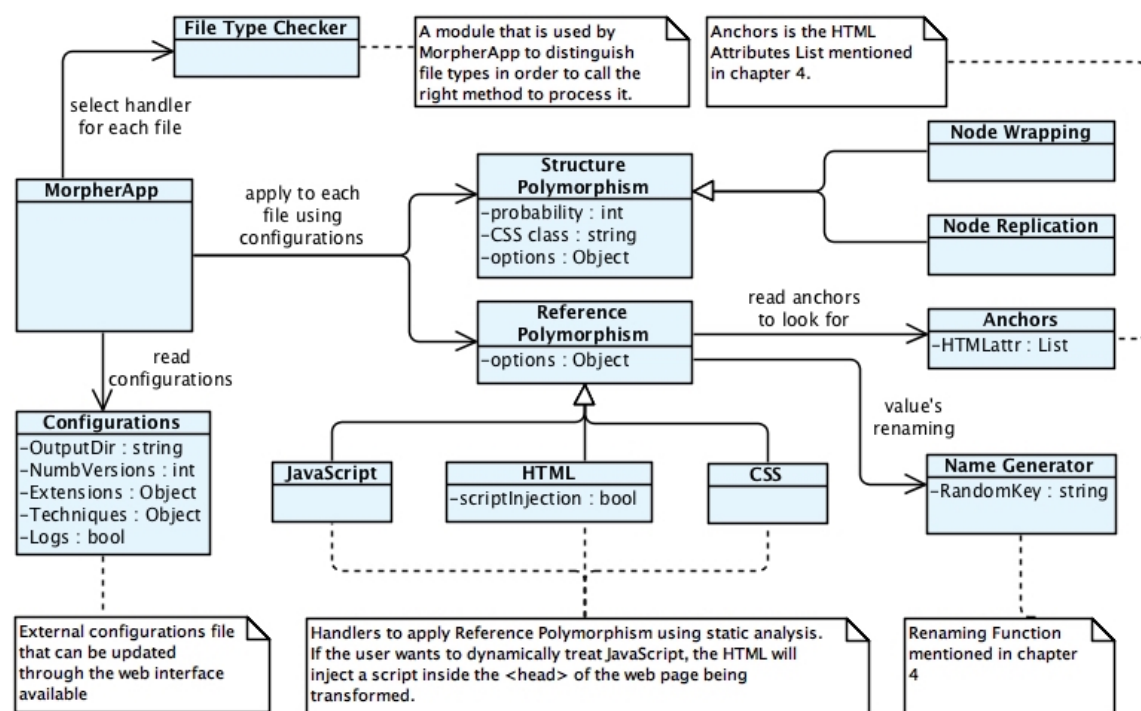


Figure 4.2: MorpherApp class diagram.

The figure 4.2 shows the solution's class diagram how the components are connected. Following the requirements defined earlier in this chapter, a user should be able to tune the application's behavior by updating the configurations file before starting the main application. To start the application the user must select all the files to be morphed. For each file, the *MorpherApp* will identify the file type and call the appropriate handler and at the end, based on the configurations, it will output the result to the standard output or to a new file. Structure polymorphism implies new CSS

rules to be added. Hence after processing a CSS file, if this polymorphic technique is being used, these new CSS rules will be appended to the end of that CSS file. Structure Polymorphism must be applied before Reference Polymorphism as the first may add new attributes in order to apply new styles that must be morphed. For the prototype to develop we decided to make Structure Polymorphism optional for the user as this technique may break a set of JavaScript methods and CSS selectors stated on table 4.2. See figure 4.3 for the activity diagram for the current MorpherApp prototype.

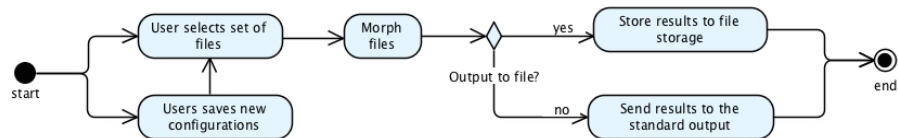


Figure 4.3: Prototype activity diagram.

Table 4.1: List of Requirements to be implemented in our solution.

HTML List of Attributes	Define which HTML attributes can see their values transformed into new ones.
	Define a renaming function to transform HTML attributes' values.
Renaming Function	This function should receive an HTML attribute's value and return a new value generated with a password. $f(input, password)$
	This function should return an output that is compliant with the HTML and CSS grammar for naming attributes.
	Given an output, this function should be able to reverse on the server to the original value.
Reference Polymorphism	This technique should be performed by three different tools so that each tool can respectively handle one of the three types of files: HTML, CSS, JavaScript.
	The tool to handle HTML documents should use the HTML Attributes List defined to identify where it should apply the Renaming Function. All HTML elements should be processed.
	The tool to handle CSS files should use the HTML Attributes List defined to identify where it should apply the Renaming function. All selectors and rules should be processed.
	The tool to handle JavaScript files should be able to identify methods related to DOM manipulation in order to apply the Renaming Function to their arguments when they refer to attributes defined in HTML Attributes List.
Structure Polymorphism	This technique should be performed by one tool that handles the HTML documents.
	The tool should be able to apply Node Replication and Node Wrapping to each input file received.
	The tool should be able to introduce randomness when applying Node Replication and Node Wrapping.
	The tool should be able to receive configurations in order to control the way Node Replication and Node Wrapping are applied.
Tools to Craft	Define an interface to be implemented by all the tools in order to concede the same usage to all of them. Each tool should receive the file content as input and should return the transformed file content.
	Each tool should return a new HTML, CSS or JavaScript file according to the handled file type.
	Each tool should return a valid file according to the W3C online standard validators.
	Each tool should be able to handle the same group of options in order to: show the changes applied to a file; output the morphed file to the standard output; create a new file on the system to store the morphed file.

Table 4.2: JavaScript methods and CSS selectors that may return unexpected results after structure polymorphism to a web page.

JavaScript	CSS
firstChild	:first-child
lastChild	:first-of-type
parentNode	:last-child
nextSibling	:last-of-type
previousSibling	:nth-child(n)
item()	:nth-last-child(n)
	:nth-last-of-type(n)
	:nth-of-type(n)
	:only-of-type
	:only-child

Chapter 5

Prototype Implementation

Based on the requirements and architecture models presented and described in chapter 4, we developed a prototype in order to study and analyze the layer of protection polymorphism can give to web pages, in particular with reference polymorphism and structure polymorphism. In this chapter are presented all the details and important aspects of the prototype implementation.

5.1 Methodologies

To keep track of all the changes and also allow the ongoing project to be shared and accessible from different computers, we used a git remote repository to store the project. The GitFlow¹ branching model was adopted in order to keep everything organized in a way every developer could easily understand. Thinking also in readability, we took the advantages of using linting tools as ESLint² so that all the code written obeys to a community standard set of syntactic rules.

The project was divided into different components as mentioned in 4.3. First, we started by creating polymorphism tools and only after, the MorpherApp was built to glue it all together. To maintain always an executable version of the current working tool and ultimately, the MorpherApp, we adopted the Scrum agile methodology: every Monday we planned a weekly sprint and every morning we had a 10 minutes meeting to state what tasks were completed the day before. This method helped to stay on track during all the development process.

All tools developed were validated by formal testing and the generated HTML and CSS code were validated using the W3C validators public APIs.

¹<http://nvie.com/posts/a-successful-git-branching-model/>

²<http://eslint.org>

5.2 HTML Attributes' List - Anchors

The Anchors file represents the HTML Attributes' List discussed in chapter 4 and has it was mentioned before, this file holds a list of HTML attributes, that can be safely morphed, organized by "global attributes" and "element specific attributes".

All HTML elements were analyzed without filtering in any way the elements by HTML version. Putting means to this, the elements analyzed include HTML5 elements and older HTML elements, even if deprecated, as long as it has support by all major browsers. We decided to do this to give users with web pages not HTML5 compliant to still be capable of using MorpherApp.

```

1 {
2   "global-attrs": ["id", "class", "contextmenu"],
3   "a": ["download", "name", "target"],
4   "applet": ["code", "name", "object"],
5   "area": ["download", "media", "target"],
6   "base": ["target"],
7   "button": ["form", "formaction", "name", "target", "value"],
8   "fieldset": ["form", "name"],
9   "form": ["name"],
10  "frame": ["name"],
11  "iframe": ["name", "srcdoc"],
12  "img": ["usemap"],
13  "input": ["form", "formaction", "formtarget", "list", "pattern", "name"],
14  "keygen": ["form", "name"],
15  "label": ["for", "form"],
16  "link": ["target"],
17  "map": ["name"],
18  "meter": ["form"],
19  "object": ["usemap", "name", "form"],
20  "option": ["value"],
21  "output": ["for", "form", "name"],
22  "param": ["name"],
23  "td": ["headers"],
24  "textarea": ["form", "name"],
25  "th": ["headers"],
26  "track": ["label"],
27  "select": ["name"]
28 }
```

Code 5.1: Content of the Anchors file used by the built solution prototype.

An attribute should be added to this list if and only if it's value is not pre-defined by HTML standards, otherwise, applying changes to that value will cause the web page to behave unexpectedly. Hence, all attributes whose values are only dependent on the developer's will, can be transformed across all types of files.

Given this, the result of this process of HTML attributes identification was a JSON file, presented in 5.1, that is used by all the three tools created to apply the reference polymorphism in order to match the attributes of HTML element that should be transformed. Since this is an external file, MorpherApp users can easily update and modify it to only apply transformations to the HTML elements they think it's appropriate given his scenario.

5.3 Renaming Function

One of the obstacles found during the implementation of reference polymorphism tools was how to generate different and random names while ensuring that the algorithm will not produce two equal outputs for two different inputs. Following this line of thinking, the solution found was to take advantage of encryption algorithms that are deterministic (so a given input results always in the same output), chaotic (the inputs "ab" and "ac" will result in very different outputs despite being alike) and finally with these algorithms we can take advantage of the low probability of producing outputs collisions. Hence, the naming function used in our solution prototype is an encryption algorithm.

5.3.1 Encryption Algorithm as a Name Generator

By using an encryption algorithm we do not want to hide the original values of attributes but rather generate new ones to substitute them. As mentioned in the requirements' chapter (4), the renaming function should allow to reverse morphed values and at the same time generate different outputs for the same input. Hence, we selected an encryption algorithm that is deterministic and symmetric. This means that, by changing a password, the same algorithm will output different values for the same input as we are looking for.

This enables us to modify HTML and CSS code independently because every time the `class="myclass"` appears, given a password, the generated name will be always the same and so there's no need of keeping track of the changes along the process, ensuring always compatibility between the files. This also reduces the storage has only one hash representing the password will be stored in favor of a map of all transformations done. Each transformation will result in a pair of the version number and encryption key.

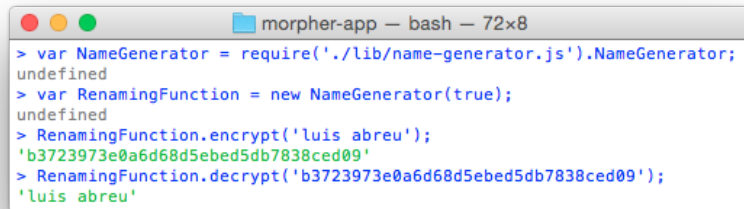
Finally to ensure a good random output, a good random password is needed. Since our application is built with Nodejs we took advantage of the "crypto" library. The "crypto" library from Nodejs has one method that *"Generates cryptographically strong pseudo-random data."*, which is enough for this prototype, and this method is `crypto.randomBytes(size[, callback])`.³

This is the method used by our Name Generator to seed the encryption algorithm with a random password each time a new version of a set of files is being created, hence ensuring randomness in the outputs for the same input.

³Nodejs "Crypto" Official Documentation page - goo.gl/fdh3sC

Prototype Implementation

The algorithm selected was the AES (acronym of Advanced Encryption Standard) because is a strong algorithm that is yet to be cracked and is the first recommended encryption algorithm by the American Federal Bureau of Investigation (FBI) as for today.[14]



```
> var NameGenerator = require('./lib/name-generator.js').NameGenerator;
undefined
> var RenamingFunction = new NameGenerator(true);
undefined
> RenamingFunction.encrypt('luis abreu');
'b3723973e0a6d68d5ebd5db7838ced09'
> RenamingFunction.decrypt('b3723973e0a6d68d5ebd5db7838ced09');
'luis abreu'
```

Figure 5.1: Renaming function example of transformation and reverse functions results.

5.3.1.1 CSS vs HTML grammar

The HTML grammar allows naming the "id" and "class" attributes starting with a number between 0 and 9. The CSS grammar does not. In CSS these values must start by an alphabetic character. Hence, since we cannot be sure the renaming function will not output values started by numbers, we are going to force this to happen by generating also a random char between "a" and "z" that will be append at the beginning of all the outputs of the naming function and removed before trying to reverse one of the outputs. This random character is unique as the password for each polymorphic version created from a set of files.[15]

5.4 MorpherApp

MorpherApp is the name given to our solution that applies reference polymorphism and structure polymorphism to a set of input files. Reference polymorphism was developed to handle HTML documents, CSS files, and JavaScript scripts. Structure polymorphism was developed only for HTML documents as these are the targets of the attacks. On the other hand, changing CSS selectors can lead to visual changes since the rules are interpreted from top to bottom. Finally, changing JavaScript structure is the scope of minification, obfuscation and other tools to protect these type of files and applications.

Every component described in 4.3 was put together in order to produce one or more morphed version of a set of files that is executing as shown in figure 5.2.

Processing CSS files or CSS code inside the HTML element `<style>` is the same thing. This is also valid for processing JavaScript files and JavaScript code inside the HTML `<script>` element, and also for HTML and CSS code that may exist in the middle of JavaScript code.

Prototype Implementation

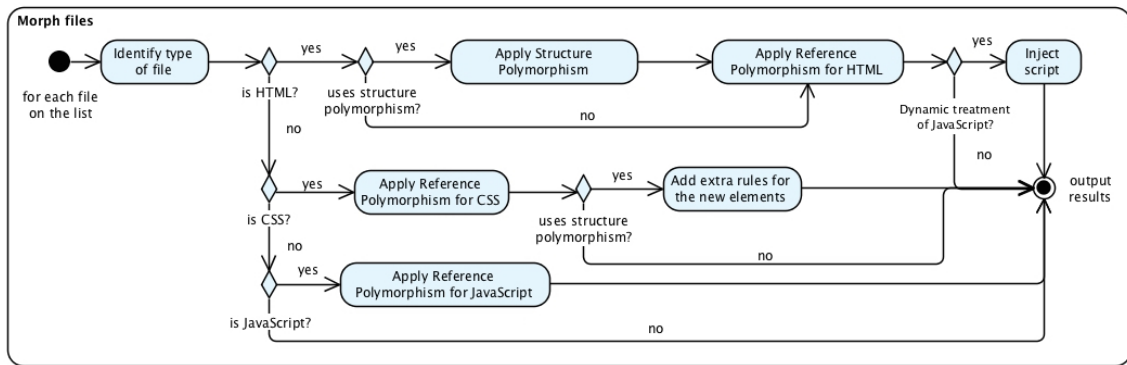


Figure 5.2: MorpherApp's prototype activity diagram.

Each time the MorpherApp starts to generate a new version of morphed files for a set of input files, a new randomly generate key will be used so that the Name Generator will produce different values for different versions of the same attribute's value.

When using the MorpherApp to create more than one version for the same file or only one version for a set of input files, it is recommended to specify the output directory otherwise the results will all be sent to the standard output which can make it difficult for distinguish between each one. Furthermore, the MorpherApp will only generate a `password.key` file with the information about the random key used by the Name Generator if an output directory is specified. This file is important when using morphed versions of a web page as some information may need to be reversed before being processed by its server.

5.4.1 MorpherWWW - Configurations Web Interface

One of the requirements was that users should be able to access the current MorpherApp configurations and by his will change them. To make this task user-friendly, we created a local application that exposes an HTML web page to interact with. In this window, users can consult the current configurations and also update them. Using a web interface makes also easier to see and safer to update the configurations file as opposed to a user directly makes changes in this file - which we allow to do but do not recommend as the user can make the file unreadable for the MorpherApp who will refuse to run if that happens.

All the configurations presented on chapter 4 are available for users. The most relevant ones are:

- Number of versions to create given a set of files - this value will define how many times the MorpherApp will order the same task for an input set of files to execute. In the output folder, the MorpherApp will create a new directory to hold the files for each version created and a key file. This way, MorpherApp can, for instance, store the output right to a server's website folder so that the server will use the morphed versions instead of the ones. Nevertheless, some changes are needed in the server. See section 6.3 for more details in this topic.

Prototype Implementation

- Structure Polymorphism - as explained on section 4.3, this technique was made optional.
- Node Replication probability and unique name that can be used to create a new CSS class.
- Node Wrapping probability and unique name that can be used to create a new CSS class.
- JavaScript static analysis - see section 5.5 to know more about the JavaScript reference polymorphism approach implemented.
- Extensions for HTML, CSS, and JavaScript files - for some reason, users may use different extensions for one of these three types of files. As an example, HTML documents has the `.html` extension but in some cases can have the `.htm` extension. Make this configurable turns MorpherApp much more personalizable and general. Nevertheless, this introduces a warning for users as they are responsible for not misleading the application, otherwise, MorpherApp will invoke an inappropriate handler for a file that will not be transformed.

5.4.2 Tools Configurations

All the tools were built using the same model has a base to produce equal structured tools with the same interface, options, and outputs.

5.5 Reference Polymorphism Application

There were created three tools, one for each type of file, yet they share the same interface and options. The input of each function is the file content of HTML documents, CSS files or JavaScript script which means none of these tools will deal with opening files for reading because it's not their responsibility. Another reason for this implementation decision is to allow to create a pipeline with all of these tools and others that may be needed.

Each tool accepts an optional object with two or four options. These options are (1) `logs`, (2) `res`, (3) `dir` and (4) `file`. The value for (1) and (2) can be either `true` or `false` and (1) will display all the attributes renamed to the standard output and (2) will send the final output also to the standard output. For the options (3) and (4), their value must be a string and when they are set, the output will be stored in a new file created with the name specified in (4) - including extension - inside the directory given in (3). If no options are provided, (1) will be `false` and (2) will be `true`. The other two will not be taken into account.

Since each tool was also designed to be integrated into other applications, as we did when creating the MorpherApp, each tool will also be able to return the output at the end of the `morph` function execution.

One important thing to mention is that all of the tools described in 5.5 perform its transformations based on static code analyses.

5.5.1 HTML documents

The application of reference polymorphism to HTML documents was very straightforward. The first step was to parse an input HTML document to a representing AST. The second step was to navigate through the generated AST while searching for the global attributes and element specific attributes defined in the Anchors file discussed in 5.2 and presented in figure 5.1. Every time there's a match, the renaming function will be called to generate a new value for substituting the original one.

If the users has set the configurations to order a dynamic analysis of JavaScript code, a new JavaScript script will be inject on all HTML documents in order to be able to modify some methods to keep everything working. See section 5.5.3 for more details on JavaScript analysis.

Lastly, when there are no more nodes to analyze in the AST, a new HTML document is created and outputted for the standard output or to a new file on disk.

5.5.2 CSS files

In order to apply reference polymorphism to CSS files, at first we need to parse all the code to create an AST representation and from there, analyze all of the existing rules. Despite the representation of each attribute being different from HTML documents, the wanted tags are the ones defined in the Anchors file. When parsing a selector, we look for special symbols such as "#" and "." that are used to represent ids and class names. At-rules and special operators like ">" are also supported in order to find all of the selectors within a rule.

However, there are a group of CSS selectors that cannot be independently transformed has their application to HTML elements is not straightforward. These selectors are presented and explained on table 5.1. Depending on the HTML document using the CSS code with these selectors, the matching element may vary. Hence is not safe to apply our renaming function directly on the value specified as it represents only a part of potential complete values. For example `div[class^="col-"]` will match `<div>` nodes having the attribute `class` with values such as "col-medium" and "col-large".

Table 5.1: Group of CSS attribute's selectors that cannot be transformed without knowing which HTML elements in a web page create a match with them.

CSS Selectors	Description
<code>[attr ~ v]</code>	Selects all elements with an attr attribute containing the word v
<code>[attr = v]</code>	Selects all elements with an attr attribute value starting with v
<code>elem[attr ^= v]</code>	Selects every <elem>element whose attr attribute value begins with v
<code>elem[attr \$= v]</code>	Selects every <elem>element whose attr attribute value ends with v
<code>elem[attr *= v]</code>	Selects every <elem>element whose attr attribute value contains v

Using encryption as our renaming function algorithm introduces a problem here because "col-medium" and "col-large" will originate completely different names even though they have the first four characters in common which in turn will be completely different from the output of "col-".

This happens because we are using a chaotic algorithm in order to introduce more randomness and unpredictability. This means that, if we transform the value it will not match the wanted HTML elements, but if we do nothing the style will also not be applied to them as well.

To overcome this issue, the independence must be broken between the current CSS file under transformations and the HTML documents that will be using this CSS code to style its elements. This is the only way we can predict elements matches and replace the original selector for a new CSS rule contains as many selectors as the matches found using the operator "=" instead of any of the others in 5.1. Following the last example, the selector `div[class^="col-"]` would be replaced by `div[class="col-medium"]`, `div[class="col-large"]`, creating one selector for each HTML element that would be matched. After computing these new selectors, we can safely apply the renaming function and move forward with the analysis. When we reach the end of the CSS file, the tool will generate new CSS code with all the transformations from the AST representation.

5.5.3 JavaScript scripts

JavaScript is a dynamic interpreted language where the code is not always running synchronously. JavaScript can also be used for event-based programming (asynchronous code). The versatility of this language creates a great obstacle to statically analyze its code as we did before because we cannot safely infer the value of some variable that can be used as an argument to a DOM manipulation methods at a specific moment of the execution. Another example of an unpredictable value is a variable used in one script that is declared in a different script also loaded on the same web page. A third example is the keyword `var` that can be omitted when creating a new variable. Statically declaring a variable as `var variable = "value"` is interpreted as a variable declaration. Without the keyword, statically that line of code would be interpreted as an assignment expression. Hence, statically analyzing JavaScript can only be carried successfully when Literals are used as arguments of functions in favor of variables' names, functions' calls or expressions.

Another obstacle is that JavaScript enables developers to redefine and reference objects and functions, which, for example, makes possible to create a new variable that is, in fact, a reference to the `document` object and so allowing to call all `document` methods through the new variable. Finally, there are multiple ways of manipulating the DOM such as by using the DOM standard calls, frameworks like jQuery or even by creating your own methods to encapsulate a group of instructions related with the DOM.

All of this to conclude that the only way to statically analyze JavaScript code is by looking for a specific group of methods and their arguments to understand what type of DOM manipulation will occur. However, any value can only be transformed if it is a Literal because we cannot safely infer the value of a generic variable. Hence, we parsed a JavaScript script to a corresponding AST and then we process each line of code looking for a group of methods known for DOM manipulation. We decided to look only for standard DOM calls in this prototype.

In this line of code we expect to find the value of an id and since the argument of the function is a Literal it can be safely transformed: `document.getElementById("value")`. On the other hand, if the argument was the variable `myid` declared as `var myid = "value"` we couldn't apply the

transformations. We can only know statically the value of a variable if it was initialized when declared but, we can only assume the first value if there are not new assignments to that variable somewhere in the code because if it does we can no longer be certain as we do not know in which order the instructions will be executed.

5.5.3.1 Dynamic Approach

To enhance the support for JavaScript files so that the web pages are still dynamic and controlled by these type of scripts, we came up with another way of dynamically treat JavaScript DOM manipulation methods. This approach involves the injection of a script on web pages to be executed first so that the `document` object methods are modified to be able to interact with morphed HTML elements.

This means the code needs to run on the client's browser and so some code needs to be sent there as (1) the content of the Anchors file, (2) the password used for creating the current web page version, (3) the renaming function, and (4) the functions used by 5.5.1 and 5.5.2 to transform HTML and CSS code respectively. All of this is needed because the transformations done with the MorpherApp have to be reproduced in real time on the client's browser. This approach creates the disadvantage of exposing (3) which is the core method for all the transformations.

To protect all of this information sent to the client, all the code is minified, obfuscated and is written inside of an immediately invoked function. The minification and obfuscation make the code more difficult for human attackers to understand it. Being an immediately invoked function means all the code is only available inside that function scope. However, if we type `document.getElementById` on the console, the output will be our code and the attacker will instantly know what we have done. To mislead attackers, each function that is redefined by us has also the "toString()" method redefined, which will return the original function `toString()` and so the output will always be the same as if it was the native code:

```
function getElementById () { [native code] }.
```

The methods redefined were `getElementById`, `getElementsByClassName`, `setAttribute`, `id`, `className`, `querySelector`, `querySelectorAll`. One of them is presented in 5.2. In each of these functions, the argument is processed and transformed by the renaming function and then they are passed to the native functions. Doing this, we do not need to worry if we are given a variable or a function call as the argument because it will be evaluated before the redefined methods are executed by the JavaScript engine, removing the static analysis restriction: only Literals as function's arguments can be transformed.

This enables developers to use also some methods in frameworks like jQuery, but not all of them. For example, jQuery's method `removeClass` is one of the not supported methods. Its normal execution would be: (1) get the current classes for that element; (2) search if the class the user wants to delete is present; (3) if (2) is true, remove the class from the string of classes and execute `document.setAttribute()` with the new updated string of classes. What makes this unsupported is the first step because the current classes of an HTML element are morphed and so, if jQuery

searches for the original value it will never find a match, hence not removing any class from the element.

```
1 var documentOriginals = {  
2     /* (...) */  
3     querySelectorAll: document.querySelectorAll  
4 };  
5  
6 document.querySelectorAll = function (a) {  
7     return documentOriginals.querySelectorAll.call(document, lu.get(a));  
8 };  
9 document.querySelectorAll.toString = function () {  
10     return 'querySelectorAll() { [native code] }';  
11 }
```

Code 5.2: Redefinition of the `document.querySelectorAll` method. The "lu.get(a)" is our function that processes a query to rename all attributes in it.

Nevertheless, the same JavaScript methods restrictions applies (table 4.2) and both JavaScript analyzes approaches cannot be used together.

5.6 Structure Polymorphism Application

Changing the structure of an HTML document is a great challenge as it may create differences in the final rendering result of a web page, but also because it may break a group of JavaScript methods and a group of CSS selectors that rely on element's hierarchy level and element's position (table 4.2). By breaking, we mean "returning unexpected and unwanted" results. All of the JavaScript methods and CSS selectors will work but since the HTML document structure will be randomly modified, they are not recommended to be used. XPath expressions should also be avoided for the same reason.

To achieve structure polymorphism, two approaches were defined: (1) node replication and (2) node wrapping.

For the solution prototype, only one tool was developed that is able to apply both of the approaches mentioned in an HTML document, based on a given probability that can also be used as a selection method: if an approach's probability is zero, our tool will not apply it to the input HTML document. Sections 5.6.1 and 5.6.2 have more details on node replication and node wrapping approaches.

5.6.1 Node Replication

As the name suggests, this approach is based on replicating existing nodes in an HTML document. Replicating nodes help not only changing the structure of an HTML document by adding more

nodes but also promotes confusion to an attacker that may find two HTML elements `<input type="password">` when it should only exist one, as an example. Adding copies of existing nodes or injecting other dummy nodes are one way of changing the structure of an HTML document while causing zero impact on the rendering result of a web page because all of these nodes can have applied the CSS property `display: none`.

CSS takes a source document, organized as a tree of elements, and renders it onto a canvas. To do this, it generates an intermediary structure called the box tree, which represents the formatting structure of the rendered document. In this intermediary structure, each box represents its corresponding HTML element (or pseudo-element) in space and/or time on the canvas. To create the box tree, CSS first uses cascading and inheritance to assign a value for each CSS property to each element in the source tree. Then, for each element, it generates zero or more boxes as specified by that element's display property. Setting the property `display: none` to an element will cause that element and/or its descendants to not generate any boxes at all, which means these elements will not be added to the rendering tree and so it is like they did not exist. [16]

Hence, we can safely state that all added nodes to an HTML document will not cause any impact on its visual display if all of them have the CSS property `display` set to `none`.

*"(...) [`display: none`] Turns off the display of an element (it has no effect on layout); all descendant elements also have their display turned off. The document is rendered as though the element did not exist."*⁴

All HTML nodes will be processed to be replicated with an exception for all elements not defined inside the `<body>` element.

To introduce randomness in this process, each node will only be replicated given a random byte and a user-defined probability of replication. Given that one byte can represent a positive integer in the range $[0, 255]$, we decide that the replication occurs if `randomByte < P(replication) * 255`.

When a node is replicated, the new node will be a sibling of the original node but the position where it will be injected is again defined by a second random byte.

The computation of the position for the replicated node is given by the equation 5.1 that allows us to project one value x_i into a new range $[a, b]$, resulting in the position y_i . This equation has two steps: (1) normalizing x_i so that $x_i \in [0, 1]$; (2) shift the normalized value to the new range so that $y_i = x_i \in [a, b]$.

$$y_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \times (b - a) + a \quad (5.1)$$

Since we are using a random byte, our positive integer x will have $\min(x) = 0$ and $\max(x) = 255$. The new range will represent the possible positions, thus $a = 0$ and $b = s + 1$, where s is

⁴Extracted from Mozilla Developer Network - goo.gl/5Lrldf.

the number of siblings that the current node being replicated has plus himself. Hence, 5.1 can be simplified to equation 5.2

$$y_i = \frac{x_i \times (s + 1)}{255} \quad (5.2)$$

Any web page is composed of different sections of information and interactions. Hence some of them may have more sensitive information than others and so they may need more protection than the others. For this reason, our implementation gives the possibility to force the application of Node Replication to a set of HTML element with a second replication probability.

With Node Replication we are able to expand the DOM tree and to modify the positions of the original elements while maintaining the same hierarchy levels.

Lastly, each replicated element will have a new class with a name specified in the configurations file for users to modify in order to prevent using a value that is already being used in a web page. When the replicated element has at least two classes after adding the new one, they will all be shuffled.

5.6.2 Node Wrapping

The second technique to achieve structure polymorphism is node wrapping and this technique is intended to change the hierarchy of HTML elements. Just as node replication, node wrapping will also be applied to all elements declared inside the `<body>` element and, for each element, the same principle applies: the decision of wrapping an element will be given by a random byte and a user-defined probability.

The CSS property used before with node replication cannot be used here because if the new element is set to `display: none` all of its descendants will also be off the rendering tree. Since we are adding a new element to wrap an original element, doing it would hide content from a web page. The only possible way of hiding an element while keeping its descendants visible is to use the CSS property `visibility: hidden` for the new wrapping element and `visibility: visible` to all of its descendants. This property will allow elements to generate boxes to the rendering tree, as opposed to `display`.

"The visibility property can be used to hide an element while leaving the space where it would have been. It can also hide rows or columns of a table." ⁵

As mentioned in the quote above, using the visibility property may affect the web page layout. Since we are creating an element to serve as a container of another element, we selected the generic block-level HTML element `<div>` because "(...) a block-level element occupies the entire space of its parent element" ⁶

By occupying the entire space of the parent, we can say our container will be in direct contact with its parent and so the only gap that could be introduced between them can be eliminated by

⁵Extracted from Mozilla Developer Network - <https://goo.gl/Wt4eSB>.

⁶Extracted from Mozilla Developer Network - <https://goo.gl/bRNED6>.

Prototype Implementation

setting `margin: 0`. Internally, the new element should also eliminate any gap between itself and its children by setting `padding: 0`. Finally, to ensure there's no border separating the outside from the inside, and so creating a gap, the property `border: 0` should be also applied. (See 5.3) The last CSS property applied was `display: inline`.

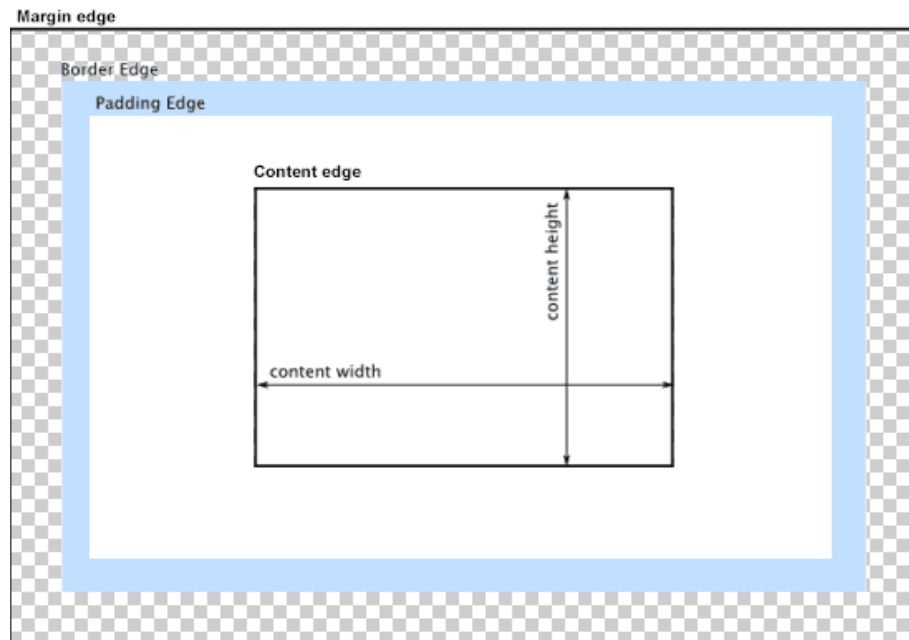


Figure 5.3: CSS box model. [1]

Prototype Implementation

Chapter 6

Results

6.1 Rendering Results

When applying polymorphism to a web page, one key point is the impact on layouts that the transformations may introduce. The MorpherApp was reconfigured to apply both reference polymorphism and structure polymorphism to an example web page. On figures 6.1 and 6.2, you can see the visual rendering of the original web page and one given morphed version created with our prototype. Figure 6.2 has a morphed version of the web page in figure 6.1. The source code for each web page is exposed on the right side of each figure.

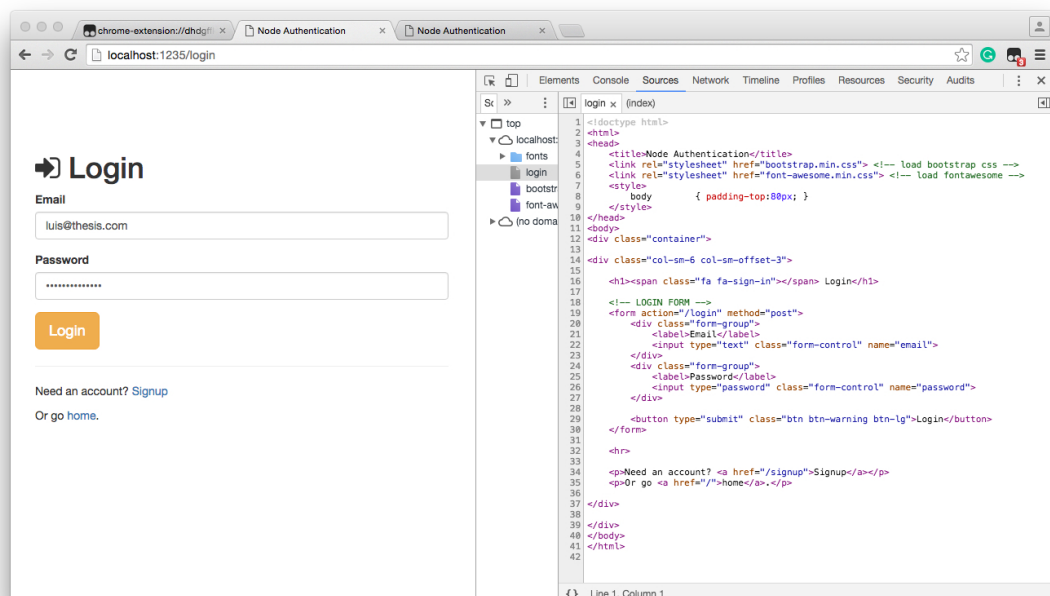


Figure 6.1: Rendering result of an example web page.

Results

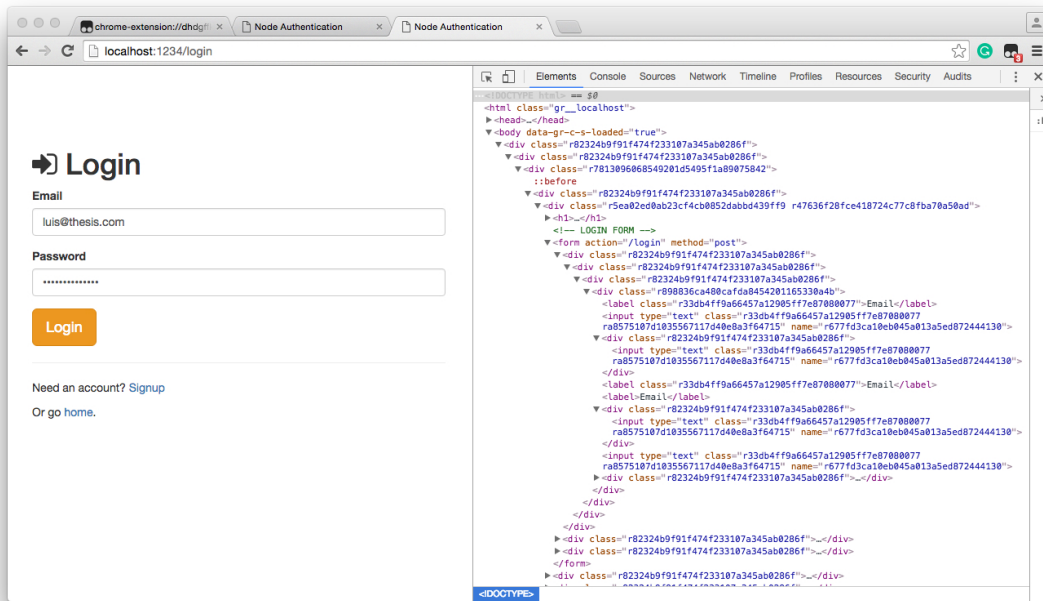


Figure 6.2: Rendering result of the web page in figure 6.1.

6.2 MorpherApp Performance

To test our prototype with maybe real examples, we used an example website composed by four HTML documents and two style sheets from different frameworks: Bootstrap and Font-Awesome.

The stylesheets had 150KB in size: Bootstrap was 121KB and Font-Awesome 29KB. The HTML documents were 6KB in total being the homepage the larger with 3KB. In total, we had 156KB to process.

MorpherApp took around 30 seconds to produce only one version and around 10 minutes to produce the 100 versions used for further testing. For this reason, we did not tested our prototype to be producing one morphed version per request on real-time and rather we created a poll of versions for a web serve to use.

6.3 Web Page Loading Time

The objective of applying polymorphism to a web page is to serve the clients with different versions created for the same web page. For testing, we set up a web server to deliver one example website. To this setup we call MorpherServer¹.

¹MorpherServer was an adaptation of the example project created by Chris Sevilleja, published in 2013 in goo.gl/0PVy1M

Results

With MorpherServer we wanted to be able to deliver the original website and also the morphed versions created by MorpherApp. Two instances of the same server application were executed using different ports.

Given this, with MorpherApp we created 100 different versions. Each of them created with 50% of probability to apply Node Wrapping and Node Replication. The reference polymorphism was applied using the Anchors file in 5.1.

We executed a web page load test for the homepage of both server's instances - one for to serve the original web page and the other to serve its polymorphic version.

With the obtained results we built the plot presented on figure 6.3. The homepage of the original website took on average 0.942 seconds to load, while the polymorphic version took, on average, 0.976 seconds to load the homepage. Hence, the polymorphic version was 3,6% slower. The original homepage file had 3KB in size. The polymorphic versions were, on average, 26.7KB in size - almost 9 times bigger than the original. These tests were performed 1000 times in the same machine with the same web server. When the server was in polymorphic mode, it has served 100 different versions of the same web page, one per request, repeating versions after 100 requests. This means that each morphed version was delivered 100 times.

6.4 Tamper Attempts

In this thesis, we presented polymorphism as means to defeat MitB automated attacks using DOM and markup anchors. Hence, to test our solution we took advantage of a third-party browser's extension that allowed us to create a script to be injected into a web page defined by a URL, simulating, locally, a MitB attack.

You can see on figure 6.4 the dashboard of TamperMonkey - the browser's extension mentioned before. We were able to create a different script for each test we wanted to perform and work with one at a time by turning on and off the scripts. Each script created on TamperMonkey has a `@match <url>` annotation where the URL for the target web page is specified.

We created three different scripts to be injected into the original version of our example web page and also on the polymorphic versions of the same web page: (1) B.1, (2) B.2 and (3) B.3. The first script (1) attempts to alert the password entered in a login form. The second script (2) attempts to alert the username entered in the same login form. The third script (3) attempts to alert again the password entered on the login form. Each script will try these attacks using different anchors. Each script will add an event listener to all `<form>` elements found on the web page under attack. All images related to the results of these attacks, both on the original web page and on one of its morphed versions, show on the left side the rendering result of the page and on the right side the source code so that we can demonstrate the code differences between the targeted pages.

Script (1) will try to match the `<input>` element with `name="password"`. This attack represents the usage of a direct anchor. We use the query `document.querySelector('input[name="password"]')` that we know would work with the original web page and that is proved in 6.5. When performing the attack on a morphed version of the original web page, the alert box showed a

Results

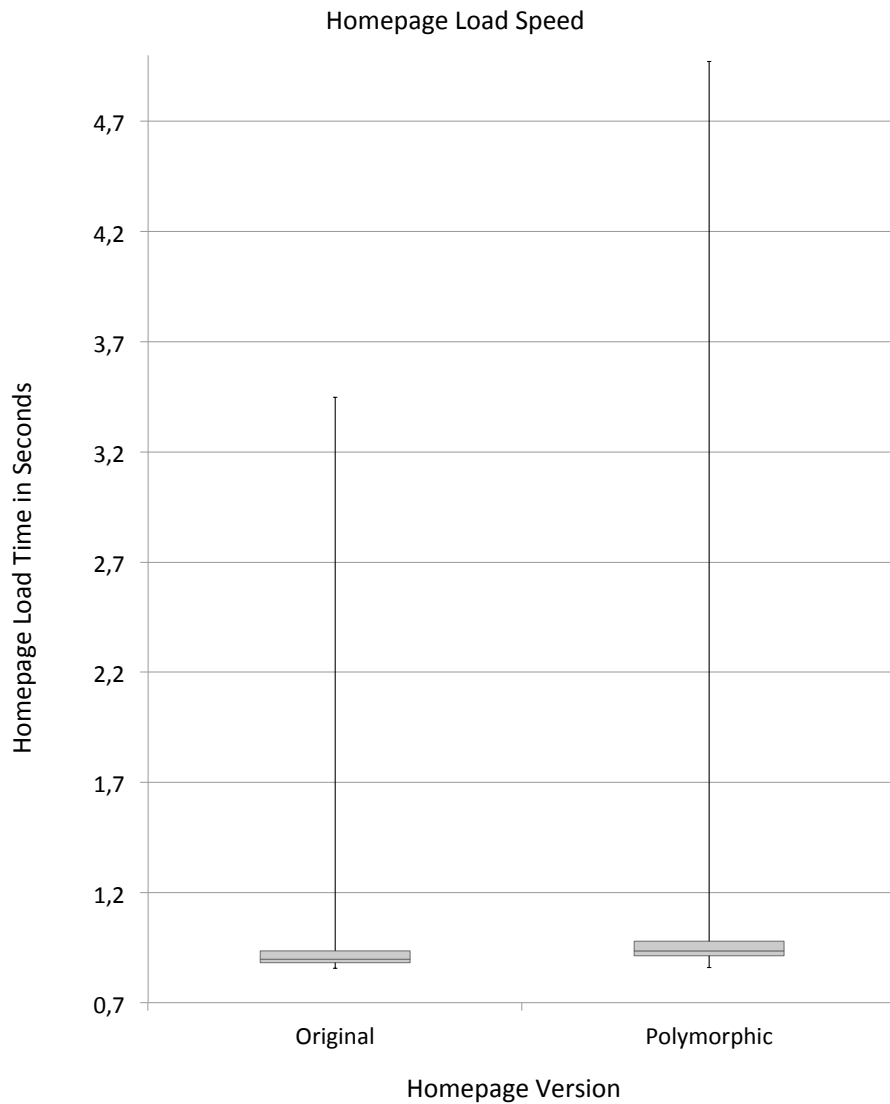


Figure 6.3: Box plot for the homepage loading time. The plot was created with a sample of 1000 accesses for each server mode: original and polymorphic. The polymorphic mode use a set of 100 different versions for the same homepage.

different result. Since the attribute used in the query was transformed by our MorpherApp, the script could not found the element and, as we scripted, the attack should pop up the alert window saying "where is your password?". Figure 6.6 shows the result of the first attack on a morphed version of the web page.

With script (2) we wanted to perform an attack using a similar anchor by using the query `document.querySelector('input[type="text"]')`. The `type` attribute used this time is one that cannot be transformed because of its meaning in HTML. On the original web page, there is only one input element with `type="text"` and so the match should result in the element where users are going to type their username to log in. Figure 6.7 shows the result of the attack on the original web page and figure 6.8 shows the result of the same attack but on the web page's morphed version.

Results

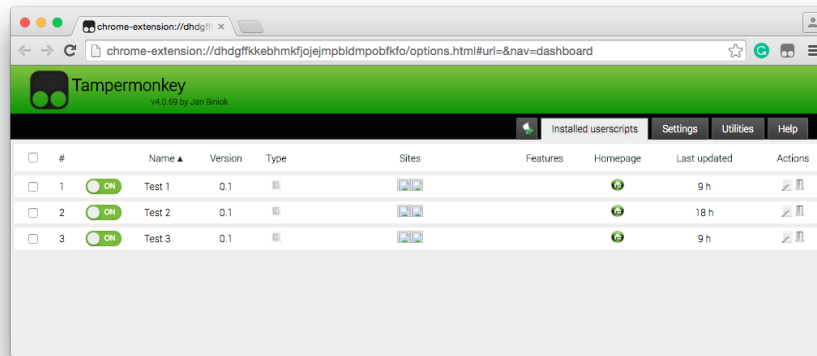


Figure 6.4: Dashboard of TamperMonkey. In the dashboard, all scripts created can be controlled to be inject into their targets or not.

As in (1), the script (2) was instructed to alert "Where is your email?" if the attack could not find the element or if its content is empty. If the content is empty means there was a match and this can happen because of node replication. However, even replicated only one will have the username as only one of them is visible and interactive to users.

Script (3) will execute its attack using XPath to locate the `<input>` password. This time, we want to attempt an attack using a different type of query by referring to the HTML element by a path. The results can be seen on figures 6.9 and 6.10. For this script, the alert box should only pop up if the attack could find an element match and its message should be the value found. In this case we can see in figure 6.10 the XPath expression matched an HTML element but since the message was empty we are certain that element was not the one where the user typed his username.

The scripts (1), (2) and (3) can be found at the end of this document on appendix B and the source code of both the original web page and the its morphed version can be found in A.

Results

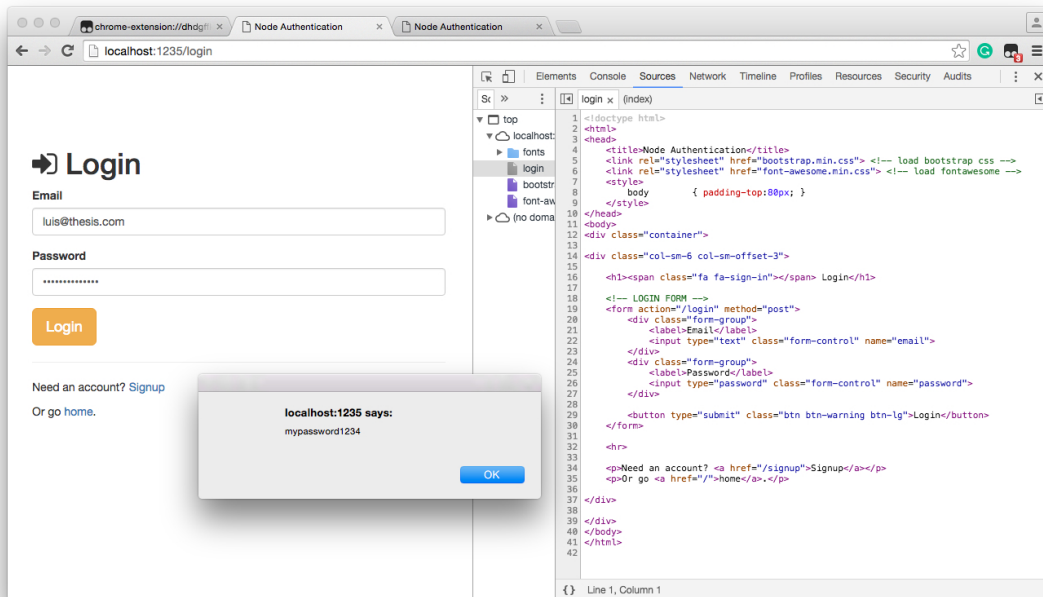


Figure 6.5: Result for the attack performed by the script (1) on the original web page. On the right is the web page HTML source code and on the left is its visual display.

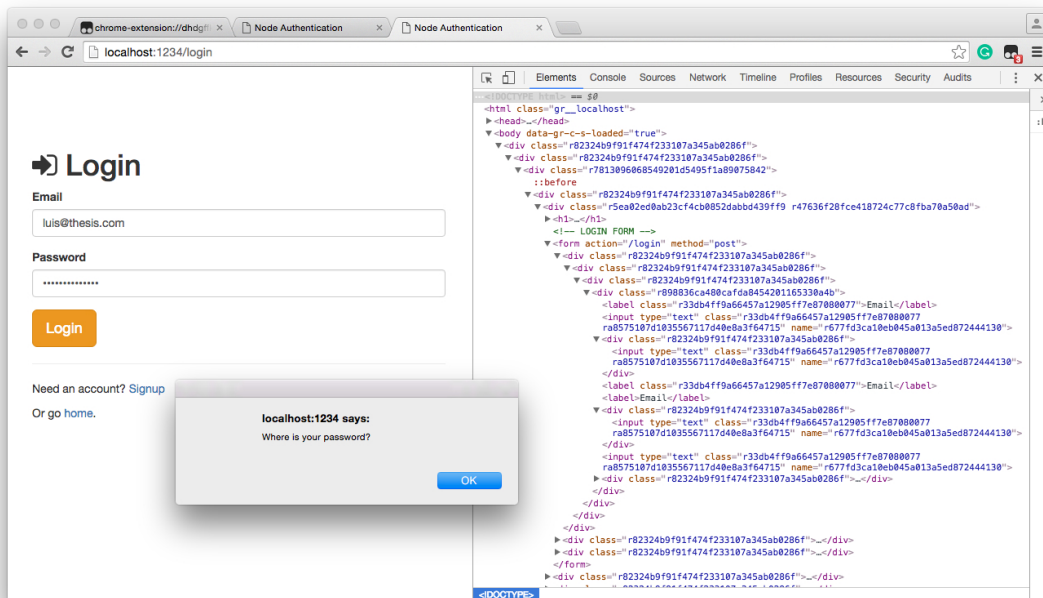


Figure 6.6: Result for the attack performed by the script (1) on a morphed version of the web page in 6.5 . On the right is the web page HTML source code and on the left is its visual display.

Results

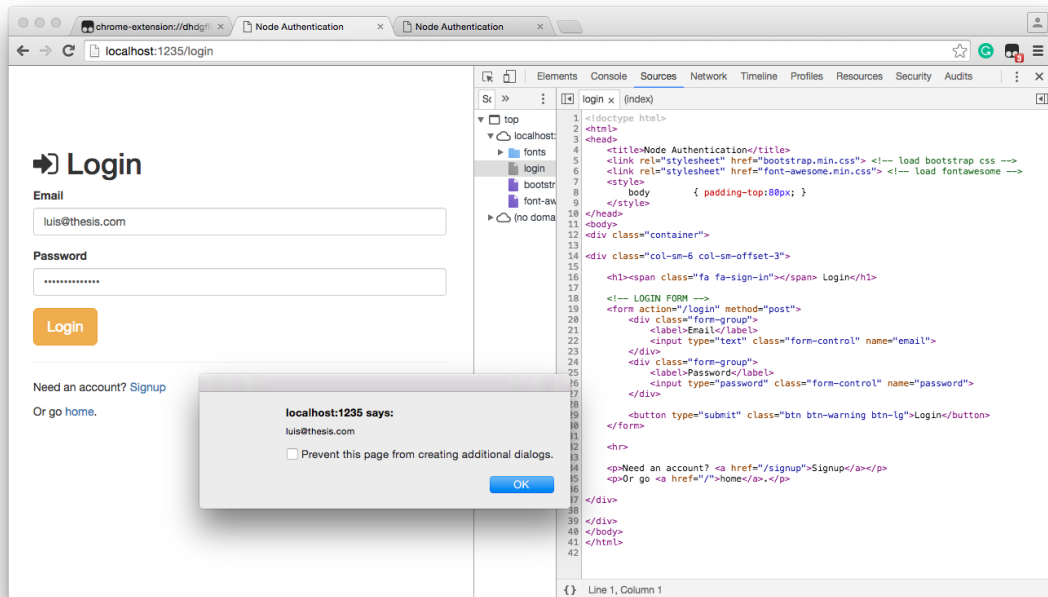


Figure 6.7: Result for the attack performed by the script (2) on the original web page. On the right is the web page HTML source code and on the left is its visual display.

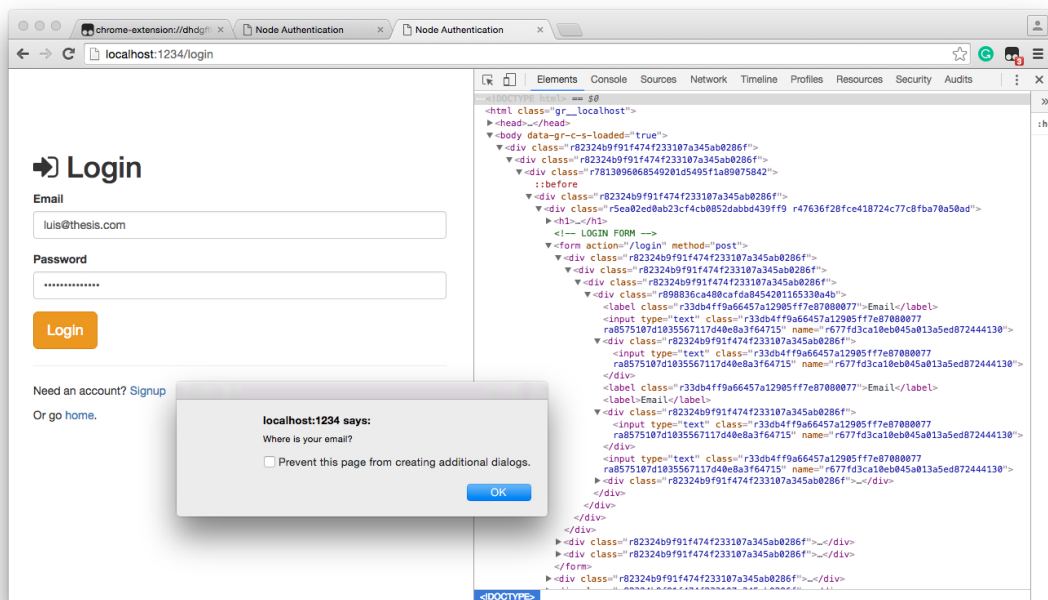


Figure 6.8: Result for the attack performed by the script (2) on a morphed version of the web page in 6.7 . On the right is the web page HTML source code and on the left is its visual display.

Results

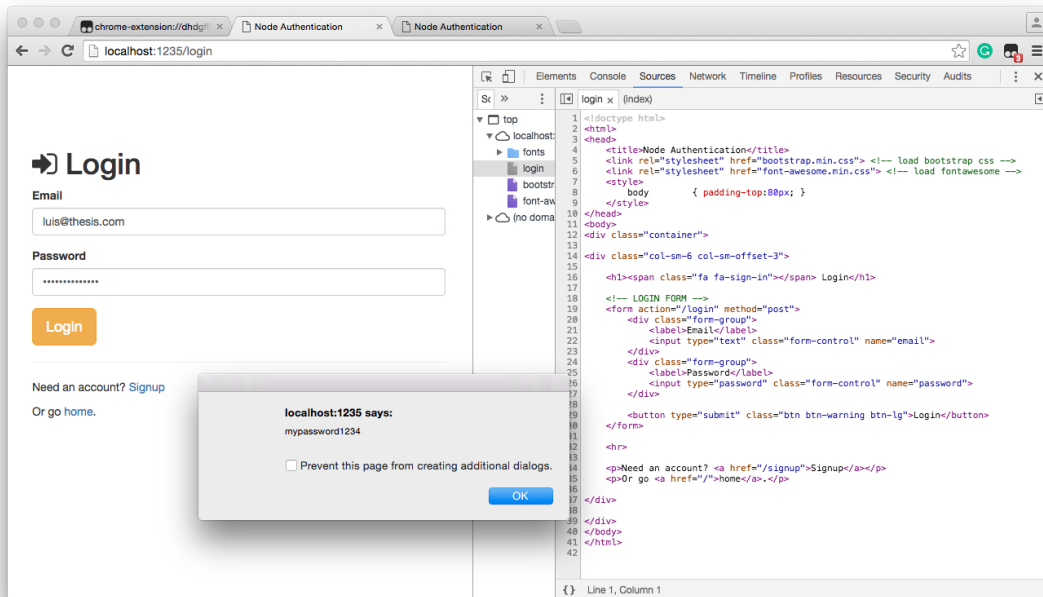


Figure 6.9: Result for the attack performed by the script (3) on the original web page. On the right is the web page HTML source code and on the left is its visual display.

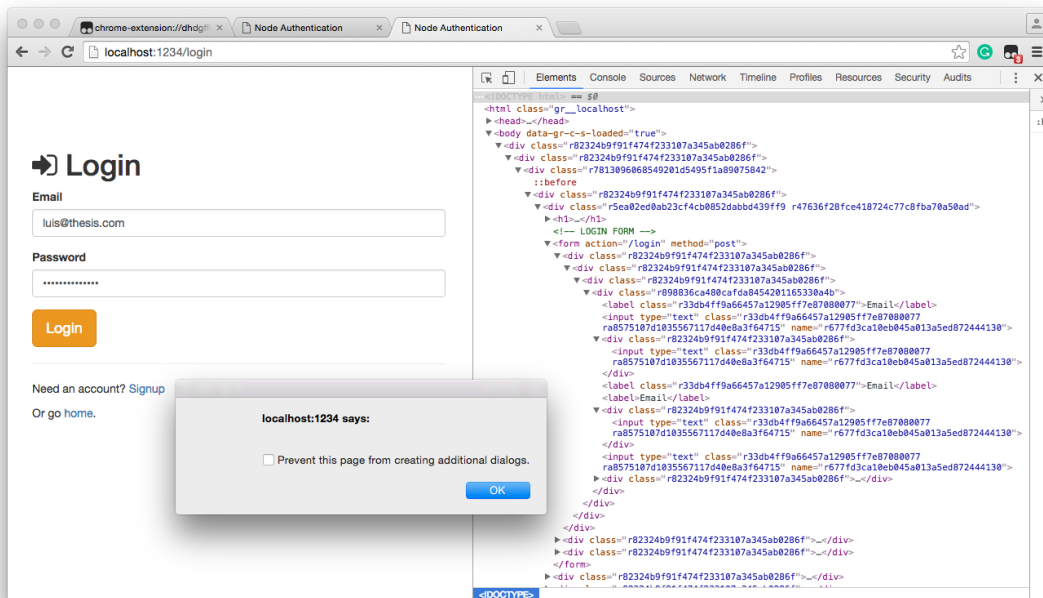


Figure 6.10: Result for the attack performed by the script (3) on a morphed version of the web page in 6.9 . On the right is the web page HTML source code and on the left is its visual display.

Chapter 7

Conclusions and Future Work

Internet users are exposed to many different cyber attacks. Malware is spread over the Internet and can be installed on users' computers by their action even though they have no knowledge that it is happening. MitB malware is able to add, remove or modify the content of web pages loaded on browsers to deceive users to give private information or trick the users with the information they see on their screens. DOM and markup anchors are used by MitB malware to drive these types of attacks.

We believe polymorphism is one way to defeat this specific type of MitB attacks, as the company Shape Security, Inc. has shown possible, even though our proposal shows it has some flaws on its own. In chapter 2 we defined the concept of polymorphism applied to web pages and we brought forward two morphing techniques to be developed as the solution to our problem: reference polymorphism and structure polymorphism.

The concept of polymorphism presented can be extended by defining new techniques to strengthen web pages local security against MitB attacks and others, but can also be applied to different problems in order to create new solutions.

The research made to the time of this document shows us that there is an American company developing a similar solution to the one we propose. Furthermore, the description in chapter 3 is based on their paper date of 2014 and so it is possible to be out-dated. The searches showed us also that there are few solutions based on polymorphism to protect infected end users. The majority of solutions found against MitB attacks are intended to prevent the infection and not to protect the user in his compromised computer.

It is important to mention that, even before we start developing our solution, we had already found some points of failure that could maybe be overcome in the future by means of new polymorphic techniques.

With the developed solution we are able to apply reference polymorphism and structure polymorphism to a web page. The application of these techniques is configurable and so, it can be tuned to introduce more or less randomness into a web page.

Conclusions and Future Work

Reference polymorphism revealed to be effective as it can break all queries that are relying on the values of polymorphic attributes. It is a technique fairly straightforward to apply based only on static analysis of HTML and CSS code. HTML code can be transformed independently from the other files of a web page. Processing CSS code revealed the existence of a group of selectors (presented in 5.1) that can only be transformed by depending on the HTML documents using them in order to obtain an accurate application of this technique. Static analysis of JavaScript code has revealed to be the biggest obstacle for applying reference polymorphism to scripts as we cannot deduce variable's values in a specific line of code due to the possible asynchronous code that might exist. Nevertheless, we were able to apply transformations to keep scripts working with their web pages. This approach on JavaScript code analysis introduces a big restriction to developers: they can only use the standard DOM API and all the arguments must be Literals.

To keep web pages working with JavaScript we developed a script to be injected into web pages in order to redefine DOM standard methods that can be used to change attributes of HTML elements or that can be used to query the web page DOM tree to access an HTML element. With this script we were able to also support frameworks like jQuery. There's only one operation that can only be achieve if we provide a method for: removing values from HTML element's class attribute. This approach as the disadvantage of exposing part of our solution's source code and the password used to transform all attributes.

Structure polymorphism was implemented by using two different approaches: node replication and node wrapping. This technique introduces a measure of randomness to a web page in order to try deflecting MitB attacks using DOM anchors, but even though it was possible to apply, this measure of randomness will also affect the work of the web page developer. This is because the same anchors used by attackers to spoof information or add some content can also be used legitimately by the web page developer to add dynamic behavior to it. The JavaScript methods and CSS selectors in 4.2 are not recommended to be used in web pages morphed with structure polymorphism as they can lead to unexpected results.

Node replication and node wrapping will increase the number of HTML elements on a web page. This increase is random and dependent on the user-defined probabilities for each one to occur. Node replication will not break any query but it will cause its results to be different. Nevertheless, since its application process is random, there's still the possibility for attacker's queries to return the expected results.

The tests performed to validate structure polymorphism revealed that it is possible to reshape an HTML document while keeping its layout intact. Nevertheless, more testing should be carried to evaluate the solution with more web pages.

Given the randomness of the results produce by MorpherApp we were able to deflect a set of automated tests created to evaluate the efficiency of our solution.

The MorpherServer was created to test how a web server could deliver different web page versions without the clients to be aware since the rendering of each version is the same as the original web page. To enable a web server to use different web page versions only two changes were needed: (1) a middleware function to parse the parameters of submitted forms in order to

Conclusions and Future Work

reverse their name to its original ones, which are the ones expected by the server; (2) a metric to decide when should a new version be delivered to clients.

Our tests showed that applying both polymorphic techniques to web pages will result in files almost 9 times bigger than the originals when replicating and wrapping nodes both with 50% of probability. Structure polymorphism is the main responsible for this size changes. Moreover, we did not test to apply polymorphism in real-time on a server because structure polymorphism will consume a few seconds which were considered not acceptable for real-time scenarios. However, serving morphed versions of the same web page only results in an increase of 3,6% on load time.

Since the prototype was built with available open-source tools, some of them introduced some limitations and thus, by improving them we can extend our solution. Another way to improve this prototype is to turn the generated names shorter.

In summary, polymorphism techniques may be a new way of protecting web pages and clients against automated MitB attacks. Reference polymorphism is an effective approach that on its own is capable of deflecting attacks. Structure polymorphism introduces a measure of randomness to a web page and despite being effective in reshaping an HTML document to deflect automated attacks, we think that it should only be applied to web pages without dynamic controls as it can introduce obstacles to programmers while developing a web page.

Finally, other polymorphic techniques should be implemented and tested in order to leverage the security of web pages running on clients' computers. Polymorphism seems to be a path for client's security, when interaction with web pages on compromised environments, that needs to be investigated and explored.

Conclusions and Future Work

References

- [1] Mozilla Developer Network. Introduction to the CSS box model. Available at https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model, last time accessed in June 10, 2016.
- [2] Justin D Call, Xiaohan Huang, Xiaoming Zhou, Subramanian Varadarajan, and Marc R Hansen. Protecting Against the Introduction of Alien Content, 2014.
- [3] Xinran WANG and Zhao YAO. Polymorphic Treatment of Data Entered at Clients, 2015.
- [4] IBM Security. Basics of Man-in-the-Browser (MitB) Malware Targets You. Available at <https://www.youtube.com/watch?v=4bPlAFZaju0>, last time accessed in October 28, 2015.
- [5] P Gühring. Concepts against Man-in-the-Browser Attacks. Available at <http://www2.futureware.at/svn/sourcerer/CAcert/SecureClient.pdf>, last time accessed in November 20, 2015.
- [6] C Cain. Analyzing Man-in-the-Browser (MITB) Attacks. Available at <https://www.sans.org/reading-room/whitepapers/forensics/analyzing-man-in-the-browser-mitb-attacks-35687>, last time accessed in November 21, 2015.
- [7] FBI. International Cooperation Disrupts Multi-Country Cyber Theft Ring. Available at <https://www.fbi.gov/news/pressrel/press-releases/international-cooperation-disrupts-multi-country-cyber-theft-ring>, last time accessed in February 2, 2016.
- [8] Entrust Inc. Defeating Man-in-the-Browser Malware.
- [9] W3C. A quick introduction to HTML. Available at <https://www.w3.org/TR/html5/introduction.html#a-quick-introduction-to-html>, last time accessed in February 5, 2016.
- [10] Shape Security Inc. Defeating Account Hijacking. Protection Against Automated Web Attacks., 2015.
- [11] Xinran Wang, Tadayoshi Kohno, and Bob Blakley. Polymorphism as a Defense for Automated Attack of Websites. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security: 12th International Conference, ACNS 2014, Lausanne, Switzerland*, pages 513–530. Springer International Publishing, Cham, 2014.

REFERENCES

- [12] R. Hansen, Marc, Sumit Agarwal, Subramanian Varadarajan, and D. Call, Justin. Stateless Web Content Anti-Automation, 2014.
- [13] Mike Gancarz. *Linux and the Unix philosophy*. 2003.
- [14] FBI. Cryptography. Available at https://www.nsa.gov/ia/programs/suiteb_cryptography/, last time accessed in March 15, 2016.
- [15] W3C. Grammar of CSS 2.1. Available at <http://www.w3.org/TR/CSS21/grammar.html>, last time accessed in April 3, 2016.
- [16] W3C. CSS Display Module Level 3. Available at <https://www.w3.org/TR/css-display-3/>, last time accessed in May 27, 2016.
- [17] Arcot. Protecting Online Customers from Man-in-the-Browser and Man-in-the-Middle Attacks. Available at <http://www.ca.com/%7E/media/Files/whitepapers/protection-from-mitm-mitb-attacks-wp.pdf>, last time accessed in December 4, 2015.
- [18] IBM Trusteer. Man-in-the-Browser (MitB). Available at <http://www.trusteer.com/glossary/man-in-the-browser-mitb>, last time accessed in December 14, 2015.
- [19] Kaspersky Lab. What is Malware, and How to Protect Against It? Available at <http://usa.kaspersky.com/internet-security-center/internet-safety/what-is-malware-and-how-to-protect-against-it>, last time accessed in October 28, 2015.
- [20] Nicolas Falliere and Eric Chien. Zeus: King of Bots. Technical report, Symantec Corporation, Cupertino, CA, USA, 2009.
- [21] Kaspersky Lab. What is a Trojan Virus? Available at <http://usa.kaspersky.com/internet-security-center/threats/trojans>, last time accessed in October 29, 2015.
- [22] Kaspersky Lab. What is a Botnet? Available at <https://blog.kaspersky.com/botnet/1742/>, last time accessed in December 12, 2015.
- [23] Norton. Bots and Botnets - A Growing Threat. Available at <http://us.norton.com/botnet/>, last time accessed in November 11, 2015.
- [24] Shape Security Inc. Shape Botwall Service. Shape Security Case Study with Fortune 500 retailer., 2015.
- [25] Shape Security Inc. At-a-glance. Available at <https://static.shapesecurity.com/Shape-at-a-glance.pdf>, last time accessed in December 6, 2015.
- [26] Sourcefire. Man-in-the-Browser Attacks (Part 1): Overview | Sourcefire Chalk Talks. Available at <https://www.youtube.com/watch?v=tLaH75LrHYI>, last time accessed in November 15, 2015.
- [27] Sourcefire. Man-in-the-Browser Attacks (Part 2): Under the Hood - API Hooking. Available at <https://www.youtube.com/watch?v=tmCYSuEbQ6M>, last time accessed in November 5, 2015.

REFERENCES

- [28] Sourcefire. Man-in-the-Browser Attacks (Part 3): Countermeasures | Sourcefire Chalk Talks. Available at <https://www.youtube.com/watch?v=SOCxM8zXfgU>, last time accessed in November 5, 2015.
- [29] BBC Click. BBC Click - The Man In The Browser. Available at <https://www.youtube.com/watch?v=DUnZMwXCkyw>, last time accessed in December 28, 2015.
- [30] Google. Declare Permissions. Available at https://developer.chrome.com/extensions/declare_permissions, last time accessed in January 13, 2016.
- [31] Google. Permission Warnings. Available at https://developer.chrome.com/extensions/permission_warnings, last time accessed in January 13, 2016.
- [32] Microsoft Corporation. What is a DLL? Available at <https://support.microsoft.com/en-us/kb/815065>, last time accessed in January 13, 2016.
- [33] Microsoft Corporation. Browser Helper Objects: The Browser the Way You Want It. Available at [https://msdn.microsoft.com/en-us/library/bb250436\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/bb250436(v=vs.85).aspx), last time accessed in January 13, 2016.
- [34] Mozilla. Introduction to the CSS box model. Available at https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model, last time accessed in June 10, 2016.
- [35] W3C. CSS Display. Available at <https://www.w3.org/TR/css-display-3/>, last time accessed in November 27, 2016.

REFERENCES

Appendix A

Example Web Page Source Code

A.1 Original Source Code

```
1 <!doctype html>
2 <html>
3   <head>
4     <title>Node Authentication</title>
5     <link rel="stylesheet" href="bootstrap.min.css">
6     <link rel="stylesheet" href="font-awesome.min.css">
7     <style> body { padding-top:80px; } </style>
8   </head>
9   <body>
10    <div class="container">
11      <div class="col-sm-6 col-sm-offset-3">
12        <h1><span class="fa fa-sign-in"></span> Login</h1>
13        <form action="/login" method="post">
14          <div class="form-group"> <label>Email</label>
15            <input type="text" class="form-control" name="email">
16          </div>
17          <div class="form-group"> <label>Password</label>
18            <input type="password" class="form-control" name="password">
19          </div>
20          <button type="submit" class="btn btn-warning btn-lg">Login</button>
21        </form>
22        <hr>
23        <p>Need an account? <a href="/signup">Signup</a></p>
24        <p>Or go <a href="/">home</a>.</p>
25      </div>
26    </div>
27  </body>
28 </html>
```

Code A.1: HTML source code of the example web page where the tests in B were executed.

A.2 Morphed Source Code

```

1  <!doctype html>
2  <html>
3  <head>
4      <title>Node Authentication</title>
5      <link rel="stylesheet" href="bootstrap.min.css">
6      <link rel="stylesheet" href="font-awesome.min.css">
7      <style> body { padding-top:80px; } </style>
8  </head>
9  <body>
10 <div class="r82324b9f91f474f233107a345ab0286f"><div class="
    r82324b9f91f474f233107a345ab0286f"><div class="
    r7813096068549201d5495f1a89075842">
11 <div class="r82324b9f91f474f233107a345ab0286f"><div class="
    r5ea02ed0ab23cf4cb0852dabbd439ff9 r47636f28fce418724c77c8fba70a50ad">
12 <h1><div class="r82324b9f91f474f233107a345ab0286f"><span class="
    r6ffa44482d93f1906f6698000de64e1d r82efa54987ab1d9b1a4c6c10c41e244e"></span
></div><span class="r6ffa44482d93f1906f6698000de64e1d
    r82efa54987ab1d9b1a4c6c10c41e244e r33db4ff9a66457a12905ff7e87080077"></span
> Login</h1>
13 <form action="/login" method="post">
14 <div class="r82324b9f91f474f233107a345ab0286f"><div class="
    r82324b9f91f474f233107a345ab0286f"><div class="
    r82324b9f91f474f233107a345ab0286f"><div class="
    r898836ca480cafda8454201165330a4b"><label class="
    r33db4ff9a66457a12905ff7e87080077">Email</label><input type="text"
    class="r33db4ff9a66457a12905ff7e87080077
    ra8575107d1035567117d40e8a3f64715" name="
    r677fd3ca10eb045a013a5ed872444130"><div class="
    r82324b9f91f474f233107a345ab0286f"><input type="text" class="
    r33db4ff9a66457a12905ff7e87080077 ra8575107d1035567117d40e8a3f64715"
    name="r677fd3ca10eb045a013a5ed872444130"></div><label class="
    r33db4ff9a66457a12905ff7e87080077">Email</label>
15 <label>Email</label><div class="r82324b9f91f474f233107a345ab0286f"><
    input type="text" class="r33db4ff9a66457a12905ff7e87080077
    ra8575107d1035567117d40e8a3f64715" name="
    r677fd3ca10eb045a013a5ed872444130"></div>
16 <input type="text" class="r33db4ff9a66457a12905ff7e87080077
    ra8575107d1035567117d40e8a3f64715" name="
    r677fd3ca10eb045a013a5ed872444130"><div class="
    r82324b9f91f474f233107a345ab0286f"><input type="text" class="
    ra8575107d1035567117d40e8a3f64715" name="
    r677fd3ca10eb045a013a5ed872444130"></div>
17 </div></div></div></div>
18 <div class="r82324b9f91f474f233107a345ab0286f"><div class="
    r898836ca480cafda8454201165330a4b">

```

Example Web Page Source Code

```
19     <div class="r82324b9f91f474f233107a345ab0286f"><div class="
      r82324b9f91f474f233107a345ab0286f"><div class="
      r82324b9f91f474f233107a345ab0286f"><div class="
      r82324b9f91f474f233107a345ab0286f"><label>Password</label></div></
      div></div></div>
20     <div class="r82324b9f91f474f233107a345ab0286f"><label class="
      r33db4ff9a66457a12905ff7e87080077">Password</label></div><input
      type="password" class="ra8575107d1035567117d40e8a3f64715" name="
      rddfd74c439be7c57caa9cb0b72cb1823">
21   </div></div>
22   <div class="r82324b9f91f474f233107a345ab0286f"><button type="submit" class="
      "rd2414407aa93bf19e36318a00176f760 rbab3a9f512a864dfe8e55cd79783cd18
      rd46d2b6e4e9e2e2abaa07b3b9db1e918">Login</button></div>
23 </form>
24 <div class="r82324b9f91f474f233107a345ab0286f"><h1 class="
      r33db4ff9a66457a12905ff7e87080077"><span class="
      r6ffa44482d93f1906f6698000de64e1d r82efa54987ab1d9b1a4c6c10c41e244e"></span
      > Login</h1></div><div class="r82324b9f91f474f233107a345ab0286f"><p class="
      r33db4ff9a66457a12905ff7e87080077">Or go <a href="/">home</a>.</p></div><
      div class="r82324b9f91f474f233107a345ab0286f"><hr></div>
25 <p class="r33db4ff9a66457a12905ff7e87080077">Need an account? <a href="/signup"
      >Signup</a></p><div class="r82324b9f91f474f233107a345ab0286f"><div class="
      r82324b9f91f474f233107a345ab0286f"><p>Need an account? <a href="/signup">
      Signup</a></p></div></div>
26 <div class="r82324b9f91f474f233107a345ab0286f"><form action="/login" method="
      post" class="r33db4ff9a66457a12905ff7e87080077">
27   <div class="r898836ca480cafda8454201165330a4b">
28     <div class="r82324b9f91f474f233107a345ab0286f"><label>Email</label></
      div>
29     <div class="r82324b9f91f474f233107a345ab0286f"><input type="text" class
      ="ra8575107d1035567117d40e8a3f64715" name="
      r677fd3ca10eb045a013a5ed872444130"></div>
30   </div>
31   <div class="r82324b9f91f474f233107a345ab0286f"><div class="
      r82324b9f91f474f233107a345ab0286f"><div class="
      r898836ca480cafda8454201165330a4b">
32     <label>Password</label>
33     <div class="r82324b9f91f474f233107a345ab0286f"><input type="password"
      class="ra8575107d1035567117d40e8a3f64715" name="
      rddfd74c439be7c57caa9cb0b72cb1823"></div>
34   </div></div></div>
35   <button type="submit" class="rd2414407aa93bf19e36318a00176f760
      rbab3a9f512a864dfe8e55cd79783cd18 rd46d2b6e4e9e2e2abaa07b3b9db1e918">
      Login</button>
36 </form></div><div class="r82324b9f91f474f233107a345ab0286f"><div class="
      r82324b9f91f474f233107a345ab0286f"><p><div class="
      r82324b9f91f474f233107a345ab0286f"><div class="
      r82324b9f91f474f233107a345ab0286f"><a href="/" class="
      r33db4ff9a66457a12905ff7e87080077">home</a></div></div><a href="/" class="
```

Example Web Page Source Code

```

    r33db4ff9a66457a12905ff7e87080077">home</a>Or go <div class="
    r82324b9f91f474f233107a345ab0286f"><a href="/">home</a></div>.</p></div></
    div><div class="r82324b9f91f474f233107a345ab0286f"><div class="
    r82324b9f91f474f233107a345ab0286f"><div class="
    r82324b9f91f474f233107a345ab0286f"><div class="
    r82324b9f91f474f233107a345ab0286f"><hr class="
    r33db4ff9a66457a12905ff7e87080077"></div></div></div></div>
37 </div></div>
38 </div></div></div>
39 </body>
40 </html>
```

Code A.2: HTML source code of one morphed version of the example web page where the tests in [B](#) were executed.

Appendix B

Testing Scripts

```
1 // ==UserScript==
2 // @name      Test 1
3 // @namespace  http://tampermonkey.net/
4 // @version    0.1
5 // @description try to take over the world!
6 // @author     Luis Abreu
7 // @match      http://localhost:1235/login
8 // @match      http://localhost:1234/login
9 // @grant      none
10 // ==/UserScript==
11
12 (function() {
13     'use strict';
14     var password = document.querySelector('input[name="password"]');
15     var forms = document.forms;
16     for (var i = 0; i < forms.length; i++) {
17         forms[i].addEventListener('submit', function () {
18             if (password !== null && password.value !== '') {
19                 alert(password.value);
20             } else {
21                 alert('Where is your password?');
22             }
23         });
24     }
25 })();
```

Code B.1: Script created to execute the first test.

Testing Scripts

```
1 // ==UserScript==
2 // @name      Test 2
3 // @namespace  http://tampermonkey.net/
4 // @version    0.1
5 // @description try to take over the world!
6 // @author     Luis Abreu
7 // @match      http://localhost:1235/login
8 // @match      http://localhost:1234/login
9 // @grant      none
10 // ==/UserScript==
11
12 (function() {
13     'use strict';
14     var email = document.querySelector('input[type="text"]');
15     // We need to apply the event listener to all forms because
16     // our tool may have replicated the original one
17     var forms = document.forms;
18     for (var i = 0; i < forms.length; i++) {
19         forms[i].addEventListener('submit', function () {
20             if (email !== null && email.value !== '') {
21                 alert(email.value);
22             } else {
23                 alert('Where is your email?');
24             }
25         });
26     }
27 })();
```

Code B.2: Script created to execute the second test.

Testing Scripts

```
1 // ==UserScript==
2 // @name      Test 3
3 // @namespace  http://tampermonkey.net/
4 // @version    0.1
5 // @description try to take over the world!
6 // @author     Luis Abreu
7 // @match      http://localhost:1235/login
8 // @match      http://localhost:1234/login
9 // @grant      none
10 // ==/UserScript==
11
12 (function() {
13     'use strict';
14     var forms = document.forms;
15     for (var i = 0; i < forms.length; i++) {
16         forms[i].addEventListener('submit', function () {
17             var xpath = '/html/body/div/div/form/div[2]/input';
18             var inputPassword = document.evaluate(xpath, document, null,
19                 XPathResult.ANY_TYPE, null);
20             var thisPassword = inputPassword.iterateNext();
21             var alertText = '';
22             while (thisPassword) {
23                 alertText += thisPassword.value + '\n';
24                 thisPassword = inputPassword.iterateNext();
25             }
26             alert(alertText);
27         });
28     }
29 })();
30 \end{minted}
31 \label{script:t3}
```

Code B.3: Script created to execute the third test.

Testing Scripts

Glossary

Advanced DOM attacks DOM attacks that manipulate the DOM tree by complex and indirect anchors to find specific nodes. A complex anchor is defined by an expression to locate a node based on his position on the DOM. [62](#)

Anchors Any XPath expression or CSS selector that can be used to return information from the DOM tree or markup language. [62](#)

Application Layer "Layer 7" in the OSI model and "application layer" in the TCP/IP model. [62](#)

Bot Any compromised PCs controlled by an attacker remotely. [62](#)

Botwall The term "botwall" was defined by Shape Security, Inc. in 2014 in [11] and stands for *"A website security layer intended to mitigate programmatic or automated use of a website user interface that is intended exclusively for use by a human"*. [62](#)

Browser Helper Object Is a Dynamic Linked Library used as a plugin for Internet Explorer that can be used to modify web pages content and to record information. [62](#)

Cryptography/Encryption Techniques employed in protecting integrity or secrecy of electronic messages by converting them into unreadable cipher text. Only the use of a secret key can convert the cipher text back into human readable form. [62](#)

DOM Document Object Model is a programming interface for HTML, XML and SVG documents. It provides a structured representation of the document as a tree and it defines a way that the structure can be accessed from programs so that they can change the document structure, style and content. [62](#)

DOM attack An attack that relies on the DOM that is rendered when the browser loads a page with the objective of manipulating the DOM representation. [62](#)

Dynamic Linked Library Is a library that contains code and data that can be used by more than one program at the same time, hence promoting code reuse and efficient memory usage. DLLs are created by Microsoft and available on Windows machines. [62](#)

HTML injection Is a type of injection issue that occurs when a user is able to control an input point and is able to inject arbitrary HTML code into a vulnerable web page.. [62](#)

Glossary

Malware Refers to a variety of forms of hostile or intrusive software, including computer viruses, worms, trojan horses, ransomware, spyware, adware, etc. [62](#)

Man-in-the-Browser Cyber attack that controls the user's web browser, so that information being transferred can be observed, intercepted and manipulated. [62](#)

Markup A system for marking or tagging a document that indicates its logical structure and gives instructions for its layout and style. [62](#)

Polymorphism When applied to web pages refers to the condition of a web page to be transformed into new different versions in code while keeping the same visual display. [62](#)

POST/POST request/HTTP POST request Is one of many requests methods supported by the HTTP protocol used by World Wide Web. By design, this request requests that a web server accepts and stores the data enclosed in the body of the request. [62](#)

Proxy Is a computer that offers a computer network service to allow clients to make indirect network connections to other network services. [62](#)

Request/HTTP request Request sent by a computer to retrieve some file using the HTTP protocol. [62](#)

Response/HTTP response Response sent by a web server back to a request received from a computer using the HTTP protocol. [62](#)

Simple DOM attacks DOM attacks that manipulate the DOM tree by using simple and direct anchors to find specific nodes. An id or class defines a simple anchor for example. [62](#)

Trojan Is a type of malware that is often disguised as legitimate software. [62](#)

Web Application An application delivered over the Internet. [62](#)

Web Page Is a document commonly written in HTML that is accessible through the Internet or other network using an Internet browser. A web page is accessed by entering a URL address and may contain text, graphics, and hyperlinks to other web pages and files. [62](#)