Masters in Informatics Engineering
Thesis
Final report

noPhish - Anti-phishing system using browser fingerprinting

João Pedro Figueiredo Correia Rijo Mendes

jprmendes@student.dei.uc.pt

Advisor:
Mário Zenha Rela
Date: July 12, 2011

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Abstract

In a world defined by technology and massive use of the internet, security issues must be a concern. There are many techniques used to overtake online security barriers and phishing is definitely one of them. The present work focuses on this technique, studying and implementing a way to detect this kind of attacks using a new approach called “browser fingerprinting”. The final solution aims to gather user’s browser information and, with it, create a unique signature that will identify that specific user and from then on determine if he/she is a legitimate user or an impersonated attacker. One of the reached conclusions was that this method might not be adequate if the attacker simulates the target user’s browser fingerprint and enters the system being considered trusted. However, it was possible to conclude that, in certain conditions, this can be a useful method applied to different matters.

Keywords

Anti-Phishing, Browser Fingerprint, Browser details, Device fingerprinting

Contents

List of Figures

Chapter 1 Introduction

1.1. Phishing

With the growth of the internet and its users, the security threats emerged and attacks became more frequent, hidden under several possible forms. If, in one hand, we register simple pranks or network accesses using sniffing methods, in the other hand there are attackers who want to obtain some financial gain or to harm people and/or systems. As an example of this particular behavior we have "phishing": a social engineering technique used to take advantage of human ignorance.

Phishing comes from "fishing", considering that the attacker throws the bait, hoping that, in a group where most will ignore it, some will feel tempted to bite it.

These attacks can be traced back to the mid-1990s when the American company *AOL* was attacked with credit card fraud. To prevent future attacks they created an algorithm that made it extremely difficult for hackers to set up fake accounts. In response, the hackers started to pose as *AOL* representatives in emails and instant messages sent to *AOL* members. The emails requested the members' usernames and passwords. Many people in the mid-1990s didn't have a full understanding of cybercrimes, so they unwittingly handed out sensitive information that was used by the hackers to enter their accounts and perform illegal activities [1].

The most known form of these attacks consists in sending emails to users falsely claiming to be a legitimate enterprise, bank or government institution, in an attempt to scam the user through identity theft. Usually this email contains links to websites that asks users to update or change their personal information, such as passwords or private social codes. Generally, all this process is transparent to the victim, so in many cases they don't find out they have been tricked until they check their account details, or being informed that something is wrong by the legitimate enterprise or organization [2].

1.2. Browser fingerprinting

Browser fingerprinting is a method that pulls together innocuous browser data, such as system fonts, language, and operating system. All this information, together, composes a browser fingerprint. Giving a real example, it is like identifying a person: just having long nose, won't identify anyone, but adding other information like the name, even if anybody else has the same one, id number, height, and so on, we soon have enough information to uniquely identify this person [3].

The use of this method can be considered privacy violation, as it can be used to collect user's navigation information and use it in behavioral advertising. Currently there is no standard policy that prohibits websites to collect this kind of available information. In any case, in this work, this method will be used to complement traditional user authentication mechanisms as an additional parameter to identify the user.

1.3. Motivations

As mentioned before, phishing is not a technical attack, since it explores the human element. The user's weakness doesn't allow him to realize that he/she is being tricked. It is the user who commits the biggest mistake (trust the bait) and therefore this hack is not easily preventable with technology. But it can be detected as we intent to demonstrate with this study.

There are some interesting questions that this study will help answer: is browser fingerprinting an effective method to detect phishing attacks? Can we manage the evolving of signatures over time? What techniques must be used to compare signatures? Integrating this project in a real environment will be the perfect case study in order to answer these questions.

1.4. Scope

This is an investigation Thesis with a strong practical component which is part of the *Masters of Faculdade de Ciências e Tecnologia da Universidade de Coimbra.*

Besides the investigation and software development, this Thesis includes all the project management activities, along with planning, scheduling and work tracking. All these tasks are made with the monitoring of an advisor teacher.

The integration part of the project was developed in direct interaction with NONIO, which is a group of *Universidade de Coimbra* that administrates a web application used by students, teachers and employees called "inforestudante". This application is used to manage several processes like grading, scheduling and payments.

1.5. Objectives

The main objectives of this work are to study the viability of using browser fingerprinting technique to uniquely identify users that are accessing a certain system and to identify the potentials and weaknesses of this approach to help detect phishing attacks or minimize its impact.

To achieve this, this study will initially focus on investigating the effectiveness of browser fingerprinting as a complementary method of user authentication. Thus, it is necessary to verify if it is possible to identify users, being able to tell, accurately, if either they are trusted or impersonating attackers.

The final solution must collect browser attributes creating a signature (fingerprint) which will then be compared with the fingerprints already stored in the database, confirming the user's identity. If the system considers that the user is not trusted it can either limit his/her access or ask for personal questions to prove its identity.

In the end we must have enough data in order to make it possible to assess the effectiveness and efficiency of this approach in the identification of phishing attackers.

1.6. Results

The detailed results can be seen in chapter 5.

In general we can consider that the results are positive. They show that, if applied in the right conditions, it is possible to integrate this technique in an application in order to guarantee a better security level, being possible to identify situations where the user is a phishing victim.

We successfully implemented this project on a real application, what allowed us to understand the pros and cons of this approach, being able to identify "untrusted" signatures and understand the reason of those results.

1.7. Work distribution

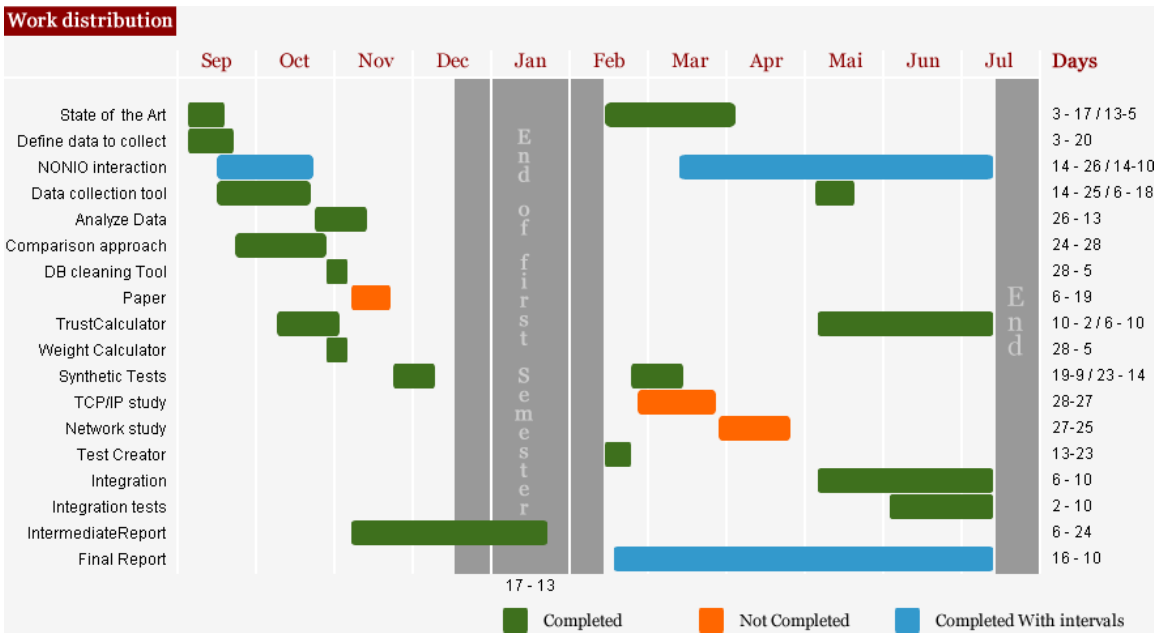


Figure 1 - Work distribution

As it can be seen, some of the work activities were done in parallel. For instance, when the interaction with NONIO team began by asking permission to collect their user's data, we started developing the data collector tool. The interaction only ended after concluding and deploying this tool in NONIO's page.

The tasks represented with the label "completed with intervals" were developed in a phased manner as it was regarding the interaction with NONIO. In this case, the interaction took place between relatively wide "intervals". The Second "NONIO interaction" activity represents, among others, the initial meetings and talks that took place in an attempt to integrate noPhish with NONIO, also represented in the figure by "integration".

There are tasks that were not completed, which are represented by the label "Not completed". The ones that took place after the end of the first semester were not completed due to the alterations to the management plan that had to be made in order to make the integration with NONIO possible.

The detailed aspects of each activity can be found later in this report.

1.8. Outline

In chapter 2, the state of the art is presented, where it is possible to read about the work that is being done in this area and the theoretical possibilities to solve some of the problems found during this project.

Chapter 3 contains the adopted approaches. In this chapter we point out the final system's workflow and its desirable behavior. This chapter contains the decisions we had to make throughout the project and their relation with what is described in the State of The Art.

In chapter 4, we describe the work that was in fact done following the planned approaches. All the components developed are explained in detail. The tests are also presented in the end of this chapter.

Finally, in chapters 5 and 6 are the project's results and conclusions.

Chapter 2
State of the art

This work began based on the *EEF's* (Electronic Frontier Foundation - privacy-advocacy group) studies "*How unique is your web browser*" by Peter Eckersley, which states the risks associated to the capability of identifying each browser by its fingerprint. The same study also focuses on the privacy problems that browser fingerprinting may pose and what countermeasures may be appropriate to prevent it and its use as a security mechanism [4].

2.1. Device Fingerprinting

This approach emerged back in the early 1990s when inventor Ric Richardson was helping musicians to use new software for playing their electronic keyboards. There was no way to let people test the software before buying it, so Richardson designed a "demonstration" version that would let people test it, but not copy it. The idea was to configure the software to work only after it was linked to a unique computer. So, he developed a way to catalog each computer's individual properties [5].

Browser fingerprinting is a device fingerprinting method based on transparent device profiling. It basically uses client-side languages and other methods available in the browser, in order to profile a device. Device fingerprinting also uses other methods like tagging, operating system fingerprinting and *TCP* fingerprinting. Besides transparent device profiling, device fingerprinting may also use client-based methods, which require installing software on an end computer [6].

Device fingerprinting is considered to be a replacement to the use of cookies in the sense that, while cookies can be deleted or blocked, it is almost impossible to delete a fingerprint after it has been collected or even prevent fingerprinting at all. One study, surveying 70 million website visits found that a fingerprint of an applicable device could be generated 89% of the time whereas cookies could only be used 78% of the time [5].

Client based fingerprinting and remote fingerprinting

In the case of a client-base approach there is the need to install executable software in the end computer. The advantage of using this method is that it has access to information that is highly unique, persistent and harder to tamper with, like hard drive serial number or *MAC* address of the network card. This approach has a major disadvantage: aside from the fact that most corporate computers won't allow anything to be installed from an external website, this method is not practical in most ecommerce transactions because the process requires some action or permission on behalf of the user.

On the other hand, remote device profiling is a new device fingerprint method that relies on information that can be measured remotely via a profiling server. This information is based on anonymous attributes that can be measured or derived from the user's browser characteristics, operating system and connection. This is a practical approach to ecommerce, online media and retail financial businesses, given that it has zero impact to the user's customer experience and their privacy and does not require registration. Despite the fact that, with this transparent profiling method, protected device attributes such as *MAC* address or hard-drive serial number are not available, recent advances in *TCP/IP* protocol and Operating System fingerprinting now enable a device to be profiled beyond more obvious browser characteristics such as browser type and version [6].

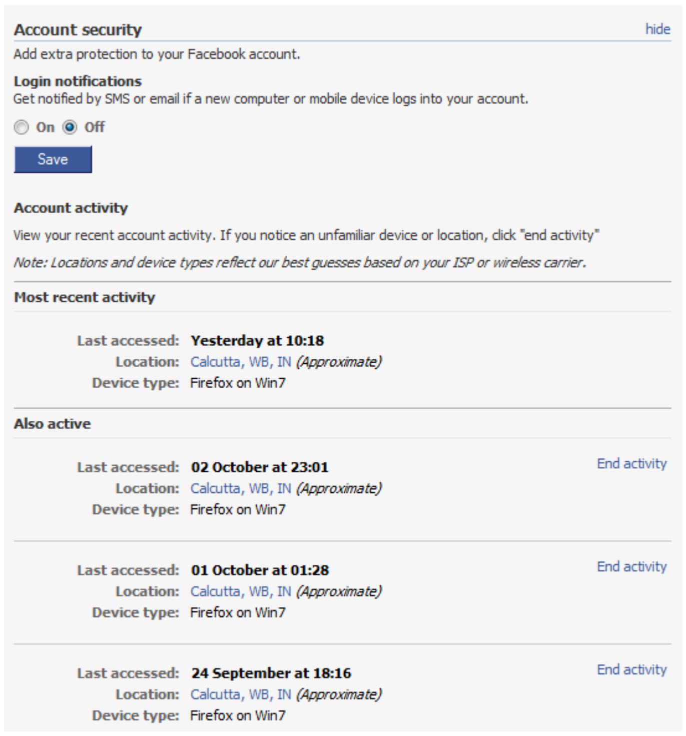
Security

Tracking companies can use this data to uniquely identify computers, cell-phones and other devices, and then build profiles of the people who use them. Hence, the controversy grows over this intrusive online tracking, since it's hard, even for sophisticated web surfers, to tell if their machine is being fingerprinted. There is not a way for people to delete fingerprints after they have been already collected.

When it comes to device fingerprinting, "*we have no convenient options for privacy*" said Peter Eckersley. "*All the things we can do are inconvenient to the point of being really impractical*". Recently Mr. Eckersley found out that about 91% of nearly 1 million computer users surveyed could be fingerprinted simply by visiting a website.

But this technique is also used to help fraud prevention, because the same techniques used to profile the users, can be applied, taking advantage of that knowledge, to identify them, enabling a fraudster's device to be recognized even when he changes the identity through the use of *proxies* and stolen credit card or account password information. Depending on the business, different strategies are used to leverage device fingerprinting to combat fraud. Detecting anomalies related to the device fingerprint is a powerful way of providing first time fraud detection. An example of anomaly detection would be determining that a device was connecting through a *proxy* to hide its real location, or determining that a device is currently under the control of a *botnet*. When fraudsters find a hole in defenses they will try to extract the maximum value as fast as they can. Creating velocity filters based on a Device Fingerprint will enable the minimization of fraud costs even when names, credit card details and *IP* Addresses are changed. A device fingerprint is a powerful tool for finding related transactions either as an identifier in itself or as a mean of finding transactions with related characteristics (e.g. finding related transactions performed from the same *ISP* and *location*). This technique is also a valuable tool to be able to detect when accounts or subscriptions are being accessed or shared illegally. If a device has been involved with fraud, adding that device to a blacklist will enable the protection of customers that share the same device reputation network [6].

Last year, *Facebook* implemented a new security functionality that consists in providing users with recent activity on their account, including the last time the account was accessed, the device used, what approximate city it was located in, and the browser and operating system on the device. It will also provide the same details for other sessions if they are active on other devices and offer the user the ability to click "end activity" to log that device off remotely (Figure 2).


Account security
hide
Add extra protection to your Facebook account.
Login notifications
Get notified by SMS or email if a new computer or mobile device logs into your account.
On [✓ Off]
Save
Account activity
View your recent account activity. If you notice an unfamiliar device or location, click "end activity"
Note: Locations and device types reflect our best guesses based on your ISP or wireless carrier.
Most recent activity

Last accessed:	**Yesterday at 10:18**
Location:	Calcutta, WB, IN *(Approximate)*
Device type:	Firefox on Win7

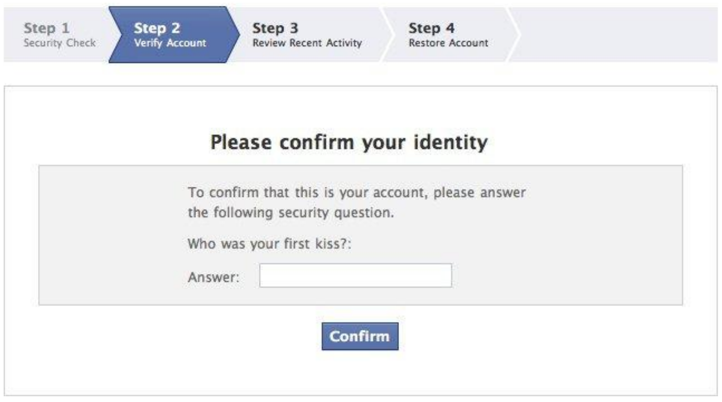
Also active

Last accessed:	**02 October at 23:01**	End activity
Location:	Calcutta, WB, IN *(Approximate)*	
Device type:	Firefox on Win7	
Last accessed:	**01 October at 01:28**	End activity
Location:	Calcutta, WB, IN *(Approximate)*	
Device type:	Firefox on Win7	
Last accessed:	**24 September at 18:16**	End activity
Location:	Calcutta, WB, IN *(Approximate)*	
Device type:	Firefox on Win7	

Figure 2 – Facebook account security

This company also built a new system to block suspicious logins even before they happen. When an unusual device is trying to get access to an account, the system asks the accessing person to answer an additional verification question to prove the user's identity as the real account owner (Figure 3). These private and personal questions are designed to help the legitimate user and to preclude the attacker to get access to the account information. The system may ask the user to enter a birth date, to identify a friend in a photo or answer a previously provided security question.

The *Facebook* security team claims that when a user is asked to go through this process, it is just them saying "Hi, we are here to help you protect your account". Although the company state that they had great results with the implementation of these functionalities, they reiterate that the first line of defense is the user, hence they encourage the user to practice safe behavior on *Facebook* applications [7].



Please confirm your identity
To confirm that this is your account, please answer the following security question.
Who was your first kiss?:
Answer:

Figure 3 - Facebook security

Advertising and privacy

Nowadays advertisers don't buy *ads*, instead they want to target and get access to specific people. Device fingerprinting can satisfy that need by recording user's online behavior, shopping habits and even demographic information. Some say that it's considered to be the next generation of online advertising.

The American company *BlueCava*, has recorded over 200 million devices and built a "Reputation" database. They plan to sell this information to advertisers willing to pay for it. *BlueCava's Device Reputation Exchange* allows businesses to share their experiences with the various devices that visit their respective websites. This allows businesses that experience fraud to tell other businesses about the "bad" devices, so they can be blocked from doing further fraudulent transactions. Just like credit card companies have been doing for years, only with devices. *BlueCava's* exchange also allows businesses to share useful information about the category of products that the devices are purchasing. This enables other websites to provide more relevant content for each device, improving the overall experience for users.

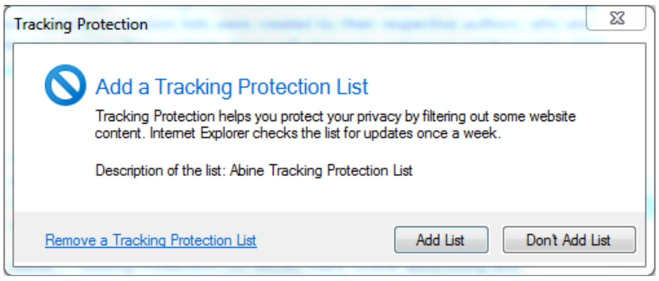
This matter brings evident questions – are the people willing to give this kind of information? Do they want to get tracked out and receive advertising? – The truth is that these companies using device profiling don't explicitly notify the people whose devices they fingerprint. These companies claim they are not using any personal information, and in fact they have no idea who is the actual user of the device, so the privacy of the individual is protected [6] [8].

Digital tracking and Fingerprinting methods are currently legal but, in the United States of America, there is an ever-increasing pressure over the online-advertising industry, from both federal regulators and some Congress members, warning them that the government will intervene if they don´t start doing more to protect consumer privacy. There are some recommendations from the *Federal Trade Commission* on that matter. *FTC* proposes that a "Do not track" system should be implemented directly in every browser if the industry doesn't start coming up with its own solutions shortly. This system would enable end users to block web service providers, marketers and advertisers from monitoring their online behavior and *FTC* would then police companies that implement tracking technologies ensuring that they comply with the user's requests. While some marketing firms say that they will create opt-out functionalities if they adopt fingerprint technology, others already have it implemented. It basically consists in having an opt-out cookie or add-on on the user's browser.

In the end of last year, the *Better Advertising Project, Inc.* announced an unprecedented partnership toward launching a program that will allow consumers to edit their interests, demographics and other profile information collected about them. It will also allow people to choose not to be tracked at all. This program follows the advices stated by government and *FCT* [9].

BlueCava's CEO David Norris, states that his company "supports the *FTC's* efforts to push the industry to self-regulate". *BlueCava's* privacy system allows consumers to specify their preferences and opt-in/out of being tracked, separately from being targeted. "We believe that there is an important distinction between the two" – tracking is all about "watching" which sites you visit on the web, while targeting is about serving relevant information for consumers. "We believe that consumers should have the choice, so we provide both options". [10]

Microsoft implemented a tool to block tracking in its newest version of *Internet Explorer*. The tool, allows users to stop certain websites and tracking companies from gathering information about them. Users are able to subscribe to something called "tracking protection lists" (Figure 4), which represents several lists with web addresses used by tracking companies. *Internet Explorer* then automatically blocks those companies from accessing the user's computer [11].


Tracking Protection

Add a Tracking Protection List

Tracking Protection helps you protect your privacy by filtering out some website content. Internet Explorer checks the list for updates once a week.

Description of the list: Abine Tracking Protection List

Figure 4 - Microsoft tracking protection list

Customized content

It's a common practice to analyze computer and browser details in order to display the website correctly to each user. When a page is accessed by a mobile device, some web applications gather some information about the device's resolution and perform mobile detection techniques to display the content accordantly to the right screen resolution and size.

2.2. TCP/IP Stack fingerprinting

This type of fingerprint consists in collecting a group of attributes from a remote device available during the communications in the *layer 4* of this protocol. Since certain parameters within the *TCP* protocol definition depend on the implementation of each operative system meaning that different operating systems set different defaults for these values, it is expected that, by combining all the collected parameters, one can infer the remote machine's operating system or use them to help creating a device fingerprint.

The TCP/IP fields that may vary include the following:

- Initial packet size (16 bits)
- Initial TTL (8 bits)
- Window size (16 bits)
- Max segment size (16 bits)
- Window scaling value (8 bits)
- "don't fragment" flag (1 bit)
- "sackOK" flag (1 bit)
- "nop" flag (1 bit)

As said before, by analyzing these factors from a packet, one may be able to determine the remote operating system. This works better for some operative systems then others, not being 100% accurate. However by looking at several signatures and combining the information, the accuracy of identifying the remote host increases [12].

Active fingerprinting

Usually, stack fingerprinting has been done using active tools, which operate on the principle that each operating system responds differently to a variety of malformed packets.

In order to determine an operative system running in a given host, all one has to do is build a database on how different operating systems respond to different packets, then send that *host* a variety of malformed packets, check how it responds and compare these responses to a database.

Passive fingerprinting

With this approach, instead of actively querying the remote system, the packets are passively captured. By analyzing *sniffer* traces and identifying these differences, one may be able to determine the operating system of the remote host [12] [13].

2.3. Browser information

To this subject, Browser information refers to the user's system information that can be freely accessed by the browser through the use of client-side languages, as *JavaScript* or *Flash*. *Browserspy.dk* is a website containing several methods on how to obtain different types of information.

To collect such information, the user must access the website where the collection scripts are running. All the information that is gathered has to be converted in a unique fingerprint. To do so, there are several possible approaches.

There are numerous distinguishable attributes provided by browsers that allow us to tell them apart. A good example is *User-Agent*, which is a string containing the name, operative system and browser's version number. This particular attribute is sent every time a user visits a web server.

```
E.g.: Mozilla/5.0(Windows; U; Windows NT 5.1; en-GB; rv:1.8.1.6) Gecko/20070725
Firefox/2.0.0.6
```

As can be seen, apparently, there is a lot of information in this given user-agent string. All of it is useful to distinguish a browser from another. In fact, user-agent string contains about 10.5 bits of identifying information (string usually carries 5-10 bits), which means that, by picking a random person's browser, only one in 1500 ($2^{10.5}$) other users will share the same string [14].

Entropy

Having separate facts about a person is not enough to identify it. But if we manage to have a group of attributes, it turns out that there is a possibility to deduce the person's identity. Consequently, each one of the facts is partially identifying. There is a mathematical key measure of information that allows us to measure how unique a person's identity is considering a given fact: entropy [14].

The called information theory is considered to have been founded in 1948 by Claude Shannon in his seminal work, "A Mathematical Theory of Communication". According to Shannon's information theory, entropy is a measure of the uncertainty associated with a random variable which quantifies the expected value of the information contained in a message, usually in units such as bits [15].

Thus, the entropy H of a discrete random variable X with possible values {x1,…,xn} is represented by

$$H(X) = \mathrm{E}(I(X)). \tag{2.1}$$

E represents the expected value, and I is the information content of X. So, I(X) is a random variable. Having p as the probability mass function of X then the entropy can explicitly be written as

$$H(X) = \sum_{i=1}^{n} p(x_i)\, I(x_i) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i), \tag{2.2}$$

If the unit for the entropy is bit, so the base of the logarithm is 2.

2.4. Fingerprint comparison

Regarding methods for comparing signatures, we can say little about the specific algorithms used by entities applying this methodology. One of the most immediate approaches is to hash the concatenated browser's features and compare signatures by performing a string match between them.

There are various ways of comparing strings and consequently signatures. Notwithstanding, there is the need to choose an approach that meets the requirements of each particular situation.

Approximate string comparison

In computing, approximate string comparison is the technique of finding strings that match a pattern and that can be expressed in percentage rather than exactly. The problem of approximate string comparison is typically divided into two sub-problems: finding approximate substring matches inside a given string and finding dictionary strings that match the pattern approximately.

In approximate string comparison, the objective is to find matches for short strings, like words from a dictionary in longer texts, in situations where a small number of differences is to be expected.

Levenshtein distance

The *Levenshtein* distance measures the amount of difference between two sequences. It is defined as the minimum number of edits needed to transform one string into the other, with the acceptable edit operations being deletion, insertion or substitution of a character.

The *Levenshtein* distance is generally used when comparing short strings but it can also be used to compute the distance between two longer strings. However, the cost to compute it, which is roughly proportional to the product of the two string lengths, makes it non viable.

Damerau- Levenshtein distance

Damerau-Levenshtein is an extension of the *Levenshtein. Damerau-Levenshtein* distance algorithm also considers the number of transpositions of two adjacent characters. For example, comparing "joao" with "ojao" would give a *Levenshtein* distance of 2 and a *Damerau-*

Levenshtein of 1. This numbers can be used to calculate a ratio based on the maximum length among the strings. In this case "joao" and "ojao" would have a *Damerau-Levenshtein* similarity ratio of 75% (1 out of 4 characters is different) [16].

Case-base reasoning

In 1980, at Yale University, Roger Schank began the first studies on the techniques based on case-based reasoning (*CBR*). In 1983, Janet Kolodner developed the first *CBR* system based on Schank's dynamic memory model, serving as a base for other *CBR* systems

CBR is the process of solving problems based on solutions of past and similar problems. When a doctor uses the symptoms of a recent case to assign a cause and a cure, a lawyer who resorts to jurisprudence to solve current cases or a judge who creates case law, they all are using case-based reasoning.

For computer reasoning purposes, case-based reasoning has been formalized as a four steps process:

1. Retrieve: Given a target problem, retrieve cases from memory which are relevant to achieve a solution for it.
2. Reuse: Map the solution from the previous case to the target problem. This may involve adapting the solution as needed to fit the new situation.
3. Revise: Having mapped the previous solution to the target situation, test the new solution in the real world (or a simulation) and, if necessary, revise.
4. Retain: After the solution has been successfully adapted to the target problem, store the resulting experience as a new case in memory.

The Retrieval step is itself a huge area of research in *CBR*. Usually, there are two alternatives for retrieval process available in the commercial *CBR* tools: the *k-Nearest Neighbor (k-NN)* and *Decision Trees*.

K-NN involves establishing a similarity metric by which the closeness of two cases can be measured. Then the target case is compared to all the cases in the case-base and the k nearest are retrieved. It has the disadvantage that retrieval time increases directly with the size of the case-base.

The alternative is to use *Decision Trees (k-D-Trees)* where case retrieval is proportional to the depth of the *D-Tree* (i.e. k); the maximum depth is the number of attributes used in retrieval. *K-D-Trees* do have the disadvantage that they need to be rebuilt from scratch when new cases are added to the case-base.

There are *CBR* systems involving approaches to representation and retrieval that are radically different from this one, however, the majority of *CBR* systems have specifications represented as feature vectors and use either *k-NN* or *k-D Trees* for retrieval [17] [18].

Chapter 3
Approaches

3.1. Browser information

To begin with, there is the need to collect and select the browser attributes to use in the final composition of the fingerprint. To do so, an initial analysis of the existing browser attributes and the means to retrieve them is mandatory.

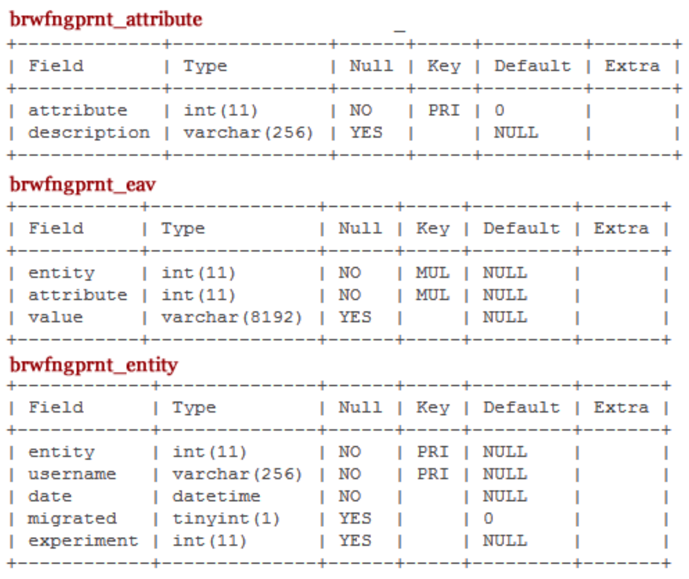
Data to collect

The first step is to identify all the possible collectable browser attributes pointing out the technologies needed to extract them. And, subsequently, create a database to store real values from those attributes.

Afterward, the ideal scenario is to retrieve the browser information in a real environment, to which the solution is to develop and integrate an information collection module. This module consists in a hidden frame, pointing to our own web server, that can be placed anywhere on a page to collect and store the browser information. That page contains several scripts that gather all the attributes we've pointed out initially.

Multiple source attributes

The browser attributes originate from multiple sources like *Flash*, *Java*, *Javascript* and *PHP*. To collect all the attributes from the different sources, the approach, in this phase, is to store them as the respective script finishes executing. Thus, there is the need to create an entity with a unique *ID*. As the attributes are gathered, they are stored in an *EAV* structured table, identified by that entity.



brwfngprnt_attribute

```
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| attribute   | int(11)      | NO   | PRI | 0       |       |
| description | varchar(256) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

brwfngprnt_eav

```
+-----------+---------------+------+-----+---------+-------+
| Field     | Type          | Null | Key | Default | Extra |
+-----------+---------------+------+-----+---------+-------+
| entity    | int(11)       | NO   | MUL | NULL    |       |
| attribute | int(11)       | NO   | MUL | NULL    |       |
| value     | varchar(8192) | YES  |     | NULL    |       |
+-----------+---------------+------+-----+---------+-------+
```

brwfngprnt_entity

```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| entity     | int(11)      | NO   | PRI | NULL    |       |
| username   | varchar(256) | NO   | PRI | NULL    |       |
| date       | datetime     | NO   |     | NULL    |       |
| migrated   | tinyint(1)   | YES  |     | 0       |       |
| experiment | int(11)      | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
```

Figure 5 - EAV structured database

As can be seen (Figure 5), in an *EAV* structured database, we manage to save the different users first and then associate all the collected data with that users.

3.2. Browser fingerprinting system

Proposed Browser Fingerprinting system approach

Assuming that the noPhish system is integrated with a real entity, when login is performed with regular credentials, our system checks the provided fingerprint against the database, by sending the browser information to a web service that calculates the trust of the fingerprint. If the result is trusted, the user is allowed to access his account. In the other hand, if the result is not trusted the user might be a *phisher* and a second validation should be triggered. This validation can be executed in the form of a security question, request of a token-card number, or validation of text message (cell phone). This second validation should keep *phishers* out while letting legitimate users in.

Concerns

There are some factors that need to be taken into consideration when building a system of this kind, including impact, flexibility, matching accuracy and technique and fingerprinting depth.

Preferably the system should have zero impact on user's experience as well as the system server and infrastructure. Thus, all the process shall be transparent to the user.

The system can be implemented as a web service enabling easy and cost effective integration in the end architecture.

When choosing the approach to generate, maintain and compare fingerprints, we must aim for implementing a system that can change over-time and is self-configurable, since some of the browser characteristics do not remain constant. Hence, when generating the fingerprints, the approach cannot be to perform a simple hash of the collected attributes. In addition, the system must perform all the fingerprint comparison as fast as possible.

There must be an initial analysis of the different characteristics that can be gathered from the browser, especially when considering that today's browsers support technologies like Flash, Java and JavaScript that are capable of obtaining extensive information. However, such technologies can be disabled, jeopardizing the fingerprint collection. Nevertheless, this concern should be taken into account when studying the scope of integration.

When composing the fingerprint, there is information that is very easy to manipulate by a knowledgeable fraudster and is also blind to warning signals that lie beneath what the browser shows. So, the approach is to analyze and study technologies that are able to recognize a fraudster even when the browser's attributes change or when cookies are deleted, and can more accurately alert to when a high risk proxy is being used.

Fingerprint composition

Selection of final attributes

After collecting enough information from the browsers into the *EAV* database, the intent is to select the final attributes to use among all the ones that were collected. Since the idea is to guarantee that the most differentiating attributes are part of the fingerprint, we need to perform an analysis of the collected data focusing on checking the values of the attributes and their frequency, fill rate and entropy (See Appendix A). Ordering the attributes by entropy and checking the distinct values numbers, we select the most differentiating ones to compose the final fingerprints (See Appendix B).

As can be seen (See Appendix A), although *IP* Address has high entropy, it is a fast changing attribute and therefore we did not consider it for the selected attributes.

Subsequently to the selection of the fingerprint composition, there is the need to copy the data retrieved in the *EAV* database to the final System's database. In order to process that transfer, one have to use a technique called pivot tables. The use of this method is very helpful because, it allows results to be displayed in an independent table, showing reorganized data.

Two of the most important steps for implementing a *CBR* approach are choosing which attributes will compose the case and calculating the weight of each attribute. Since the composition is chosen, it remains calculating the weight. A more traditional solution would be to hand pick attributes and manually fine tune the weights until a good result is obtained. We selected a better approach that goes back to what it is intended to be implemented in the system - a genetic programming and information theory based methodology. For that reason, the idea is to use entropy as the main drive to select and confer weights to the attributes. We split the 100 point total through the attributes according to their percentage of total entropy. For example, for two attributes with entropy 4 and 6 we would assign a weight of 40 to the first attribute and 60 to the second. This guarantees that the most differentiating attributes have the highest weights.

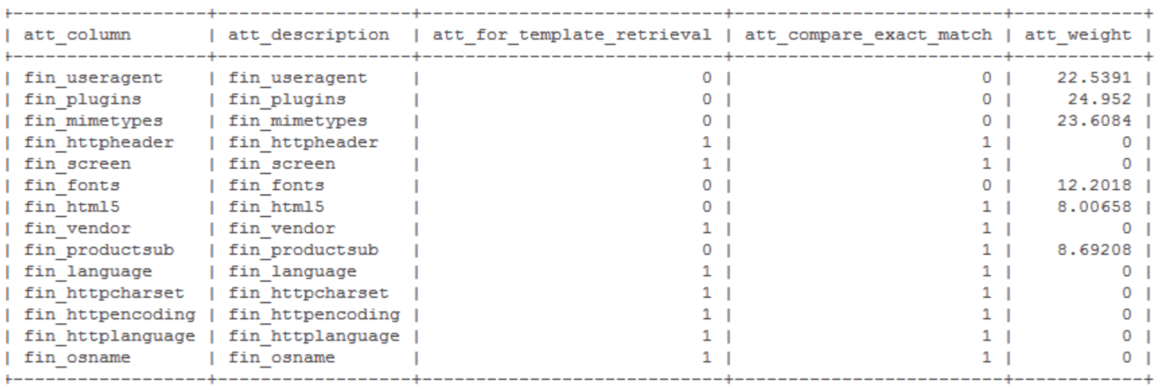
Fingerprint comparison

The ambition regarding fingerprint comparison is to develop an algorithm that performs it as fast and efficient as possible. In our system, the objective is not to concatenate the gathered information into a static fingerprint. Fingerprints are expected to change frequently, as the browser is updated or the user installs new fonts or plug-ins. Having these aspects into consideration, traditional *Bayesian networks* are left out, because constantly recalculating the network would have a severe performance impact. For the same reason, custom rules approach was discarded, since we would have to manually update the rules over time. *Adaptive Bayesian networks* would fix the need to constantly recalculate the whole network by dynamically incorporating new data into the model. However, comparing with some other analyzed approaches, this approach is not as simple as we intend to implement the algorithm, and it would take us more time than planned to spend on this task.

Concerning our system intent, case base reasoning appears to be the most appropriate solution. After discussing and studying some possible approaches, the solution is to adopt a template retrieval approach in which the system returns all cases that fit within certain parameters. This method limits the search space to a relevant section of the case-base. After that, we implement a *nearest neighbor* approach that involves the assessment of similarity between stored cases and the new input case, based on matching a weighted sum of features. To complement this approach, we will use a configurable attribute, used to collect the top score cases. We also follow a *Null Adaptation* approach, which is a simple technique that consists in applying whatever solution is retrieved to the current problem without adapting it.

Retrieving the nearest neighbors

In the process of getting the nearest neighbors of a given case, we use several approaches when comparing browser's features. The comparison is done by exactly matching and by approximate comparison.



```
+-----------------+-----------------+--------------------------+-------------------------+------------+
| att_column      | att_description | att_for_template_retrieval | att_compare_exact_match | att_weight |
+-----------------+-----------------+--------------------------+-------------------------+------------+
| fin_useragent   | fin_useragent   |                        0 |                       0 |    22.5391 |
| fin_plugins     | fin_plugins     |                        0 |                       0 |     24.952 |
| fin_mimetypes   | fin_mimetypes   |                        0 |                       0 |    23.6084 |
| fin_httpheader  | fin_httpheader  |                        1 |                       1 |          0 |
| fin_screen      | fin_screen      |                        1 |                       1 |          0 |
| fin_fonts       | fin_fonts       |                        0 |                       0 |    12.2018 |
| fin_html5       | fin_html5       |                        0 |                       1 |    8.00658 |
| fin_vendor      | fin_vendor      |                        1 |                       1 |          0 |
| fin_productsub  | fin_productsub  |                        0 |                       1 |    8.69208 |
| fin_language    | fin_language    |                        1 |                       1 |          0 |
| fin_httpcharset | fin_httpcharset |                        1 |                       1 |          0 |
| fin_httpencoding | fin_httpencoding |                      1 |                       1 |          0 |
| fin_httplanguage | fin_httplanguage |                      1 |                       1 |          0 |
| fin_osname      | fin_osname      |                        1 |                       1 |          0 |
+-----------------+-----------------+--------------------------+-------------------------+------------+
```

Figure 6 - Attributes table

Regarding the template retrieving method, the process is done by making an exact comparison of the values of the attributes previously selected for this purpose (Figure 2), between the stored cases and the new case.

For those values limited to a fix set of options, the comparison is made by exact string comparison. If the values are unbound to a set of options, like fonts or plug-ins, the comparison is made using approximate string comparison algorithms that expresses proximity as a percentage (Figure 6).

Approximate string comparison

As exposed before, we use approximate string comparison as an essential step of the signature comparison process. The problem is that *MySQL* does not provide any functions for it. This way, the approach is to initially test out different techniques and choose the one that solves the problem in a more efficient and effective way.

The two most used algorithms for approximate string comparison are the *Levenshtein* and the *Damerau-Levenshtein* distance algorithms. The tests started by using the *Damerau-Levenshtein* distance using a *MySQL* stored procedure. However the performance is not acceptable since it takes one second to calculate the distance between two strings. The performance improved when we implemented a *UDF* written in *C++*. Nevertheless, a real comparison could be of one fingerprint to several thousands, and frequently the total time summed up to 600 seconds.

Analyzing the results, the conclusion is that the algorithm choice is not the best. Both *Levenshtein* and *Damerau-Levenshtein* consider individual characters as the base unit, but although this is valid for spell checkers and other user inserted strings (because of the mistakes inserting characters), it does not make sense when comparing data collected from the browser. If we compare, considering the browsers data, "*Firefox*" and "*Firebird*" have nothing in common, but *Damerau-Levenshtein* would assign a ratio of 50%. Taking this in consideration, we modified the algorithm to consider the similarity based on the number of words in common between strings. The approach is to consider the words as tokens, no matter their order. For example "*Arial, Verdana, DIN*" and "*Arial, Verdana*" would give a similarity of 66% (1 out of 3 tokens is different). The performance was greatly improved, speeding from the 600 seconds to at most 60 seconds.

Tools to manage signatures

Database Cleaner

For performance issues, it is necessary to maintain the number of signatures relatively small. We decided to implement a database cleaning tool that would only keep the N newest fingerprints for each user, where N is the maximum number of fingerprints per user, allowing to keep several fingerprints for each user but not so much as to slow down the system.

Using *Oracle*, to implement this approach, it is only necessary to group the fingerprint by username, rank them by date and delete the ones with rank higher than N. These analytical functions can be done in *Oracle* in just one query. However, we are using *MySQL* which does not have analytical functions and does not allow deletion with multiple sub-queries. *MySQL* would need 3 different queries with minor performance to perform the same algorithm emulating analytical functions.

To circumvent this performance issue, we developed a new algorithm. This approach uses a threshold M, and if a user has more than M fingerprints, only the L newest are kept (where M is the threshold of deletion, and L is the optimum number of fingerprints).

Weight Calculator Tool

Because browser's reality is constantly changing as well as the entropy of each attribute, we built a tool that executes the weight calculation and updates the algorithm every time needed. This way, even if, in the future, emerges the need to select and introduce new browser characteristics, with this tool, the entire weight calculation process is simpler.

3.3. Trust calculator algorithm approach

System algorithm

1. The web-page triggers a call to the system;
2. The system collects the browser fingerprint and the user's identification in the original system;
3. The system makes a template retrieval based on a configurable set of attributes;
4. The system uses the weights to calculate the distance between the cases retrieved in (3) and the new case:
   - 4.1. When comparing two cases there is a high possibility of some attributes not being defined. In that case the following rule applies: if the attribute in both fingerprints is undefined, it is a match, if the attribute is defined in one fingerprint but not the other, then it is a mismatch;
   - 4.2. For attributes whose values are limited to a fixed set of options (e.g. a *Boolean*) the comparison should use exact comparison algorithms;
   - 4.3. For attributes whose values are unbound to a set of options (for example font list), the comparison should use approximate comparison algorithms that express proximity as a percentage;
5. The system collects the top score groups based on a configurable number;
6. If the user's user name is represented in the highest score group, then the system returns "Trusted";

7. If the user name is represented elsewhere in the top score groups, then the system returns Suspicious;
8. If neither 6 nor 7, then the system returns "Untrusted";
9. If the browser is "Trusted" or "Suspicious" there are two scenarios:
   9.1. If the new fingerprint is not an exact match with an existing fingerprint, this one is added to the fingerprint database;
   9.2. If the new fingerprint is an exact match with an existing fingerprint, it is updated the record date of the existing fingerprint.

3.4. Integration Approach

In the login process of the "inforestudante" application, managed by NONIO, a browser fingerprint is collected after the user logs in successfully and accesses to the initial page for the first time.

This signature is sent along with a hash of the user´s identification through an *iframe* to the noPhish *web service*, letting the user continue to utilize the application.

After the web service receives the fingerprint, the trusting mechanism is called and the fingerprint is processed. If the reply is "untrusted" or "suspicious", the signature should be stored in a temporary table and its identifier should be sent to the NONIO's *web service* along with the result.

If NONIO system receives a response, it should check if the user was notified by email about a suspect misuse of his/her credentials for that signature in concrete. If it has not been notified that day (between 23:59 and 0h) an email should be sent to the user's official email account. The email should contain a link where the user can change the password if he/she thinks that there is suspicious misuse of the credentials. The link also must contain the user hash and the id of the signature stored in the noPhish system so that the application can recognize which user is changing the password and the respective signature. If the user submits the password modification, the received signature id, if it already exists in the signature list, shall be removed.

When the 0 hours of each day are exceeded, the NONIO system shall send a list of trusted signature *IDs* to the noPhish web service. The signature temporary table should be cleaned and the trusted signatures must be stored in the fingerprint table.

3.5. TCP/IP and Network analysis

With the intention of considering other techniques for device fingerprinting, a study on *TCP/IP* fingerprinting was carried out. The objective was to analyze the feasibility of using this type of data to complement the already collected signatures to ensure greater effectiveness in the algorithm. Testing the means to retrieve this new type of data and analyze the performance and efficiency when combining the already implemented mechanisms with this approach is necessary.

TCP/IP fingerprinting

Knowing that each Operative System implements this layer a in a slightly different way, the different operating system's *TCP/IP* stack will respond differently given the same circumstances in a *TCP/IP* communication.

This way, the approach is to test out tools and fingerprinting techniques, before implementing in the final system.

NMAP

NMAP ("Network Mapper") is a free and open source utility security auditing that also can be used to explore the network. It can also be useful for tasks like network inventory, managing service upgrade schedules, and monitoring host or service uptime.

Since *NMAP* is an active tool, it interrogates the target machine's *TCP/IP* stack by sending it different packets and observing the response. Knowing how a given operating system's *TCP/IP* stack would respond in advance to each of the eight tests allows *NMAP* to determine with a high degree of accuracy not only which operating system the target is running, but also what version it is running as well.

NMAP holds all of its known operating system fingerprints in a text file called "nmap-os-fingerprints". There are a few hundred fingerprints documented that include at least one entry for all the popular operating systems [19].

For testing *NMAP* and the possible integration with the signature Collector module, *PHP-NMAP* was used, which is a *PHP* Web frontend. The approach is, when the user's *IP* is retrieved, the system calls and runs *NMAP* having that *IP* as parameter. The results can either be inserted in the fingerprint as new attributes, or can be managed aside from the current trust calculator system.

P0f

This is a passive *OS* fingerprinting tool. It can be used for gathering profiling information about accessing users, customers or attackers, restricting access to certain systems or otherwise handling them differently or detecting users with illegal network hookups using masquerade detection, content optimization, pen-testing and thru-firewall fingerprinting [20].

The TCP/IP details that we are analyzing are:

- TTL - Time To Live,
- Window Size,
- DF - Don't Fragment bit,
- TOS - Type of Service.

By analyzing these details from a packet, one may be able to determine the remote operating system. This approach is not 100% accurate and some operative system analysis gives better results than others. Nonetheless, by looking at numerous signatures and combining the information, the accuracy of identifying the remote host increases.

Hence, the approach, using this tool, is to have a server running p0f and saving the logs from user's accesses to the servers. Then, when the system requires that information, it calls for a comparison between the retrieved fingerprint and a database of *TCP/IP* signatures.

Network data

Detailed Network Information is another fingerprint component that can be studied in order to be integrated in the final noPhish system. The approach is to use the *RIPE*[1] Database, which contains registration details of *IP* addresses and AS numbers originally allocated by the *RIPE NCC*. This information contains the organizations that hold the resources, where the allocations were made, and contact details for the networks.

The approach is to run a *whois* client when the user's *IP* is retrieved in the signature Collector module of the noPhish system, and save the returning results as new fingerprint attributes.

3.6. Testing approaches

Test Creator tool

Prior to the integration process, it is necessary to test the developed components of the system, particularly, the trust calculator module. Therefore, an application to generate signatures and test the implemented features must be created. This application shall generate new fingerprints, allowing the verification of the behavior of the algorithm. The generated fingerprints can simulate browser updates or the complete change of browser by a user.

The goal is to determine if the comparison between the signatures is being made correctly and if what changes in each attribute can compromise the expected results. We want to check for possible adjustments to the data collection and data analysis tools that we are using in order to get better results.

Integration tests

After implementing the entire system, it is necessary to test the integration. The approach for this procedure is very simple. The system is tested at every point of communication with NONIO. In the case of communication between *web services*, it should be created a test method that allows assessing the success of communication between both systems.

Before deploying the final systems, we must simulate all the steps represented in the system architecture, by checking the behavior of the database and create log files.

[1] http://www.ripe.net/

Chapter 4
Work done

4.1. Final Browser fingerprinting system

NoPhish System Architecture

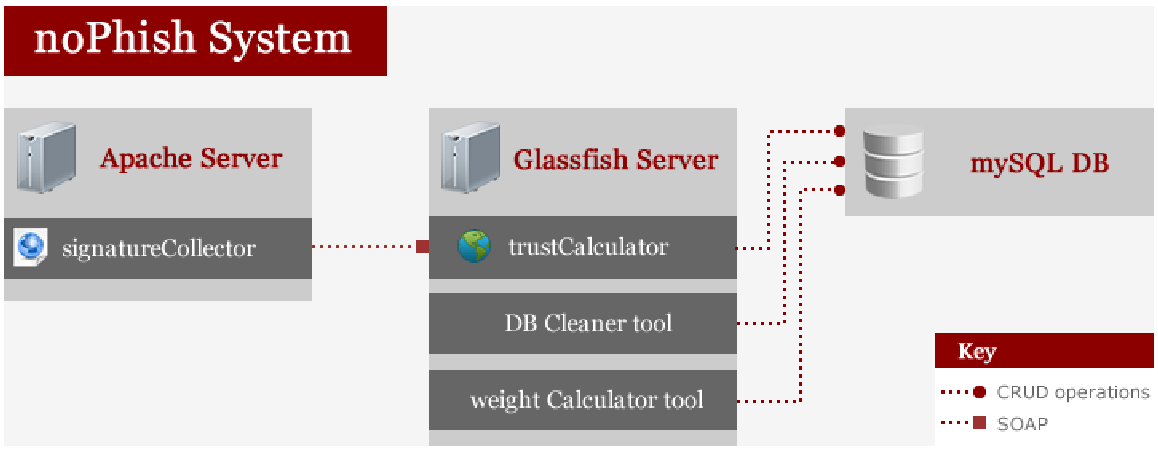


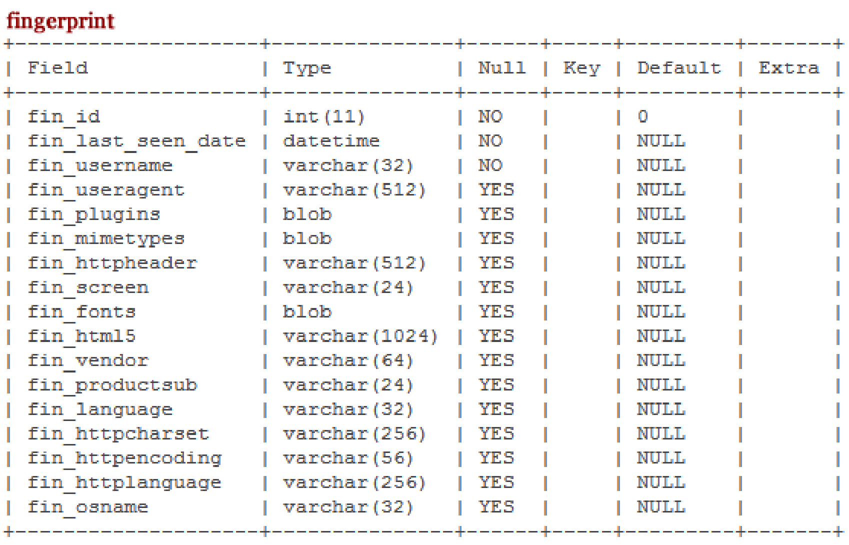
Figure 7 – noPhish Architecture

As can be seen (Figure 7), the noPhish system is composed by two servers and a *mySQL* database. The signature collection tool runs on an *Apache* server and it is basically a group of *PHP* pages containing the scripts for collecting the browser attributes.

The trust calculator *web services* runs on a *Glassfish* server and it is capable to perform *CRUD* operations on the *mySQL* database. The "DB Cleaner Tool" and the "Weight Calculator Tool" are running outside the web service context allowing the user to perform maintenance of the database directly.

The signatures are sent to the trust calculator from the web page through *SOAP* under *HTTP*.

DB structure



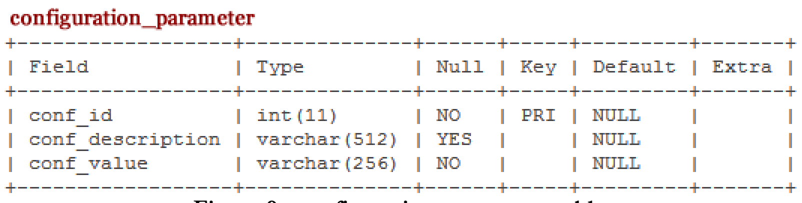
fingerprint

```
+--------------------+---------------+------+-----+---------+-------+
| Field              | Type          | Null | Key | Default | Extra |
+--------------------+---------------+------+-----+---------+-------+
| fin_id             | int(11)       | NO   |     | 0       |       |
| fin_last_seen_date | datetime      | NO   |     | NULL    |       |
| fin_username       | varchar(32)   | NO   |     | NULL    |       |
| fin_useragent      | varchar(512)  | YES  |     | NULL    |       |
| fin_plugins        | blob          | YES  |     | NULL    |       |
| fin_mimetypes      | blob          | YES  |     | NULL    |       |
| fin_httpheader     | varchar(512)  | YES  |     | NULL    |       |
| fin_screen         | varchar(24)   | YES  |     | NULL    |       |
| fin_fonts          | blob          | YES  |     | NULL    |       |
| fin_html5          | varchar(1024) | YES  |     | NULL    |       |
| fin_vendor         | varchar(64)   | YES  |     | NULL    |       |
| fin_productsub     | varchar(24)   | YES  |     | NULL    |       |
| fin_language       | varchar(32)   | YES  |     | NULL    |       |
| fin_httpcharset    | varchar(256)  | YES  |     | NULL    |       |
| fin_httpencoding   | varchar(56)   | YES  |     | NULL    |       |
| fin_httplanguage   | varchar(256)  | YES  |     | NULL    |       |
| fin_osname         | varchar(32)   | YES  |     | NULL    |       |
+--------------------+---------------+------+-----+---------+-------+
```

Figure 8 - fingerprint table

The “fingerprint” table (Figure 8) contains one line per browser signature. Each signature is characterized by the username associated with it, the last seen date, and all the fingerprinting attributes.

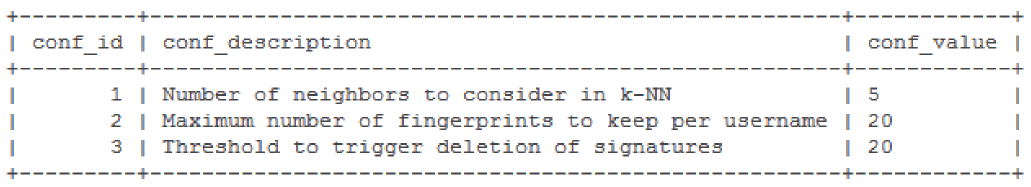
As can be seen (Appendix B) in the signatures we used “city” attribute that was obtained using the given *IP* address, however we had to remove it from the signatures latter in the project because we noticed the *API* we were using to retrieve that value stopped working.



```
configuration_parameter
+------------------+--------------+------+-----+---------+-------+
| Field            | Type         | Null | Key | Default | Extra |
+------------------+--------------+------+-----+---------+-------+
| conf_id          | int(11)      | NO   | PRI | NULL    |       |
| conf_description | varchar(512) | YES  |     | NULL    |       |
| conf_value       | varchar(256) | NO   |     | NULL    |       |
+------------------+--------------+------+-----+---------+-------+
```

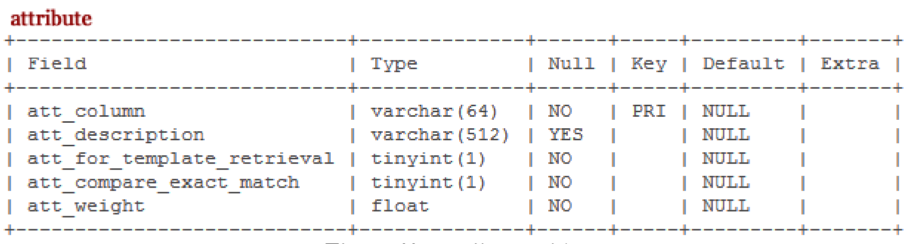
Figure 9 - configuration_parameter table

The “Configuration_parameter” table (Figure 9) contains multiple configuration parameters for the application. The parameters can be seen in the following figure (Figure 10).



```
+---------+-----------------------------------------------------+------------+
| conf_id | conf_description                                    | conf_value |
+---------+-----------------------------------------------------+------------+
|       1 | Number of neighbors to consider in k-NN             | 5          |
|       2 | Maximum number of fingerprints to keep per username | 20         |
|       3 | Threshold to trigger deletion of signatures         | 20         |
+---------+-----------------------------------------------------+------------+
```

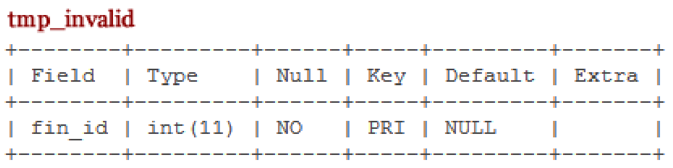
Figure 10 - configuration parameters



```
attribute
+---------------------------+--------------+------+-----+---------+-------+
| Field                     | Type         | Null | Key | Default | Extra |
+---------------------------+--------------+------+-----+---------+-------+
| att_column                | varchar(64)  | NO   | PRI | NULL    |       |
| att_description           | varchar(512) | YES  |     | NULL    |       |
| att_for_template_retrieval| tinyint(1)   | NO   |     | NULL    |       |
| att_compare_exact_match   | tinyint(1)   | NO   |     | NULL    |       |
| att_weight                | float        | NO   |     | NULL    |       |
+---------------------------+--------------+------+-----+---------+-------+
```

Figure 11 - attribute table

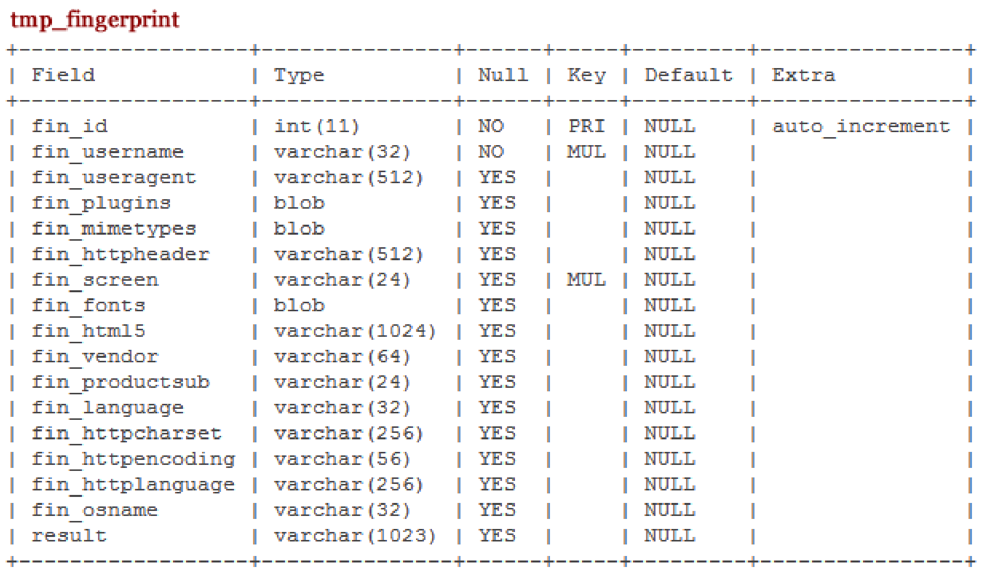
The “attribute” table (Figure 11) contains the list of available attributes. There is a *Boolean* that indicates if the attribute should be used for template retrieval, and a *Boolean* that indicates if the attribute should be compared using exact match or approximate comparison. There is a value which indicates the weight of the field when calculating the distance between signatures. The value of the “att_column” attribute is the same as the name of the attributes in the “fingerprint” table.



```
tmp_invalid
+--------+---------+------+-----+---------+-------+
| Field  | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| fin_id | int(11) | NO   | PRI | NULL    |       |
+--------+---------+------+-----+---------+-------+
```

Figure 12 - tmp_invalid table

The “tmp_invalid” table (Figure 12) contains an integer that represents fingerprints *IDs*. This table is used by the “Database Cleaning Tool”, allowing it to save the signatures *IDs* that are later removed from the system in the Database cleaning process.



tmp_fingerprint

```
+------------------+----------------+------+-----+---------+----------------+
| Field            | Type           | Null | Key | Default | Extra          |
+------------------+----------------+------+-----+---------+----------------+
| fin_id           | int(11)        | NO   | PRI | NULL    | auto_increment |
| fin_username     | varchar(32)    | NO   | MUL | NULL    |                |
| fin_useragent    | varchar(512)   | YES  |     | NULL    |                |
| fin_plugins      | blob           | YES  |     | NULL    |                |
| fin_mimetypes    | blob           | YES  |     | NULL    |                |
| fin_httpheader   | varchar(512)   | YES  |     | NULL    |                |
| fin_screen       | varchar(24)    | YES  | MUL | NULL    |                |
| fin_fonts        | blob           | YES  |     | NULL    |                |
| fin_html5        | varchar(1024)  | YES  |     | NULL    |                |
| fin_vendor       | varchar(64)    | YES  |     | NULL    |                |
| fin_productsub   | varchar(24)    | YES  |     | NULL    |                |
| fin_language     | varchar(32)    | YES  |     | NULL    |                |
| fin_httpcharset  | varchar(256)   | YES  |     | NULL    |                |
| fin_httpencoding | varchar(56)    | YES  |     | NULL    |                |
| fin_httplanguage | varchar(256)   | YES  |     | NULL    |                |
| fin_osname       | varchar(32)    | YES  |     | NULL    |                |
| result           | varchar(1023)  | YES  |     | NULL    |                |
+------------------+----------------+------+-----+---------+----------------+
```

Figure 13 - tmp_fingerprint table

The “tmp_fingerprint” table (Figure 13) contains the attributes retrieved from the browser and the result of the trust calculation process. This table is used so that the system is able to store the fingerprints not considered “trusted”. The system uses the stored fingerprint id on this table, to send it to NONIO along with the username, in order to notify NONIO about “untrusted” or “suspicious” accesses to the system.

Integration details

The integration architectural details can be seen in the following figure (Figure 14).

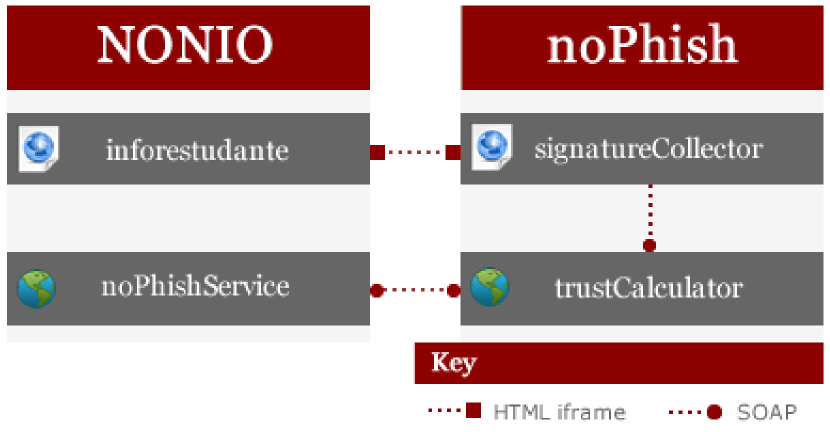


Figure 14 - Integration

Figure 14 only represents the modules where the communication between NONIO and noPhish system is direct.

In order to run the “signatureCollector” tool inside the “inforestudante” page, it was necessary to configure an *SSL* certificate signed by a valid certification authority. Only in that way was possible to eliminate the unwanted security warning that appeared on the page.

The "signatureCollector" module, included inside "inforestudante", has the user *ID* as a *GET* parameter provided by NONIO (Figure 15). Therefore, it is possible to relate the collected fingerprint with its user.

```
<div id="toggleText" style="display: none">
    <iframe
        width="100px"
        height="100px"
        src="https://nophish.dei.uc.pt/index.php?user=1D806974EADE68C9A3FB6F66C2795324">
    </iframe>
</div>
```

Figure 15 - Iframe calling noPhish

Signature collector

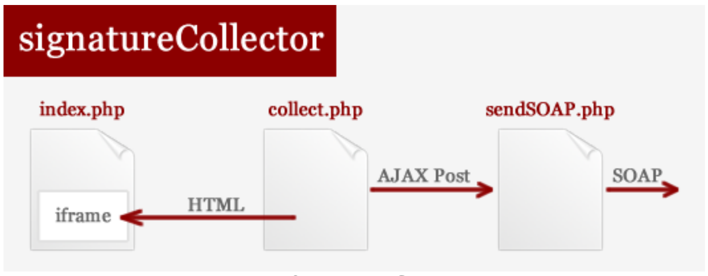


Figure 16 - Signature Collector tool

The signature collector module is composed by simple *PHP* pages. The "collect" page contains *PHP*, *Java* and *JavaScript* scripts. On that same page, the browser's attributes are retrieved and then sent to "sendSOAP" page through a *POST* request.

To obtain the data from the java applet, it was necessary to build a *JavaScript* function that grabs the results from the applet.

On the "sendSOAP" page, a *XML* message is constructed containing the collected attributes and the user ID retrieved from NONIO. With the help of a *PHP* library called *nuSOAP*, the *SOAP* client is constructed based on a given *WSDL* and the message is sent through *SOAP* to the *Web Service* described in the definition language file.

Trust Calculator

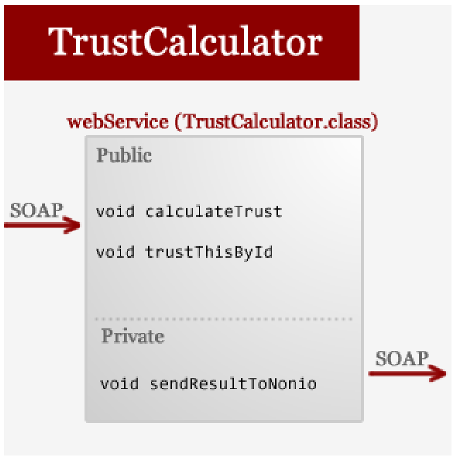


Figure 17 - Trust Calculator module (generic)

CalculateTrust

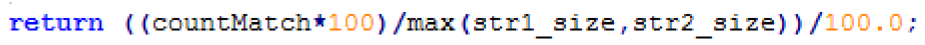
The “calculateTrust” Web Method is called by the “signatureCollector” module after it retrieves the browser’s attributes. That method receives the username, a list of attribute names, which must be the same as the “att_column” in the “attribute” table, and a list of corresponding attribute values.

The system stores those values in a hash table, composing the new case that will be used to get its nearest neighbors.

The method to get a list of the nearest neighbors consists in building a custom *mySQL* query. That query is composed by:

- A SELECT clause, where we compare the attributes used for weight comparison (“att_for_template_retrieval”=0 AND “att_weight”>0). The comparison is done by exact match or approximate string comparison, depending on the value of “att_compare_exact_match” in the “attribute” table of each attribute. The result of each comparison is multiplied by the weight of the compared attribute.
- WHERE clause. Here only the attributes used for template retrieval are compared, by using an exact match comparison. This allows us to retrieve only the cases that match the template parameters.
- GROUP by comparison score.
- ODER by descending order.
- LIMIT. Here we insert the “conf_value” existent in the “configuration_parameter” table, which represents the maximum number of neighbors (Figure 9).

The approximate string comparison is done by calling “ratio” which is a *mySQL User Defined Function*. This function returns the similarity between two strings, by counting the number of matched tokens between them, dividing that value by the max number of tokens of the two strings and multiplying all by 100 (Figure 18).



```
return ((countMatch*100)/max(str1_size,str2_size))/100.0;
```

Figure 18 - ratio.cpp return

After getting the list with the nearest neighbors, if the size of that list is equal to zero then the result of the trust calculation process is “untrusted”. If the username of the given case matches with the most common username in the neighbors the result is “trusted”. If the username appears in the neighbors, but is not in the most common group, the result is “suspicious”. By default the result is “untrusted”.

If the result is not “trusted”, the system stores the given case as a fingerprint in the “tmp_fingerprint” table. Then it calls the NONIO’s web service, sending the temporary fingerprint Id, the username and the result of the trust calculation process.

The NONIO’s web service runs on *HTTPS*. That being, in order to establish the communication channel it was necessary to implement a function that trusts the *SSL* certificates.

TrustThisById

This method receives an array of integers that contains the *IDs* of the temporary fingerprints that are considered trusted by NONIO (See 3.4. integration approach).

The system converts the array of integers into a coma separated string and builds a query to obtain the corresponding fingerprints. Then it stores them in the final "fingerprint" table. After this process, the "tmp_fingerprint" is truncated.

Tools

DB Cleaner Tool

As mentioned, this tool is used to clean the database for performance improvement. And the best way we found to keep a reasonable number of fingerprints per user, is to keep the latest ones stored in the database and delete the oldest ones.

The configuration attributes used to perform the cleaning operations are stored in the "configuration_parameter" table (Figure 8).

For each user, all her/his fingerprints' *IDs* that exceed the maximum number defined in the "configuration_parameter" table are inserted into a temporary table – "tmp_invalid" (Figure 12). Then, the system eliminates all the signatures from the "fingerprint" table (Figure 8), whose identifiers were stored in the "tmp_invalid" table.

Notice that, before selecting the exceeded fingerprints, they are sorted in a descending order by date. In this way, the more recent ones remain in the database.

Weight Calculator Tool

With this tool, the process initiates with the selection of the attributes that are not used for template retrieval. Then it calculates the entropy of each one using a *mySQL* query, replacing every "?" by the names of the selected attributes (Figure 19).

```
+ "SELECT round(sum(p_log_p),2) AS entropy \n"
+ "FROM \n"
+ "    (SELECT ?, -(count(1) / total)*log2(count(1) / total) as p_log_p\n"
+ "    FROM fingerprint, (SELECT count(1) AS total FROM fingerprint) total\n"
+ "GROUP BY ?) p_log_p;";
```

Figure 19 - mySQL query for entropy

These values are stored in a *Hashtable*. The next step is to balance the weights according to each attribute's entropy values. To do so, the system divides the attribute's entropy by the total entropy and multiplies it by 100.

The weight values are then updated in the "attribute" table for each attribute.

System's behavior

Consolidating, the overall system's behavior is the following (Figure 20):

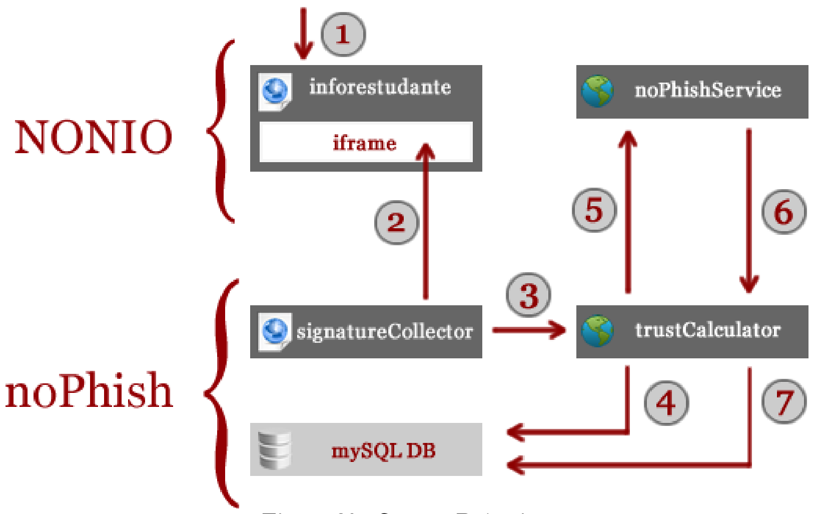


Figure 20 - System Behavior

1. The user accesses the inforestudante page. This page retrieves a hash of the user's username which is passed by *GET* parameter when including an *iframe* pointing to noPhish system.
2. The "signatureCollector" module is included inside the *iframe* running scripts to collect the Browser's information.
3. The collected information is sent through *SOAP* to the "trustCalculator" module. Here the system calculates the trust level of the given fingerprint.
4. If the result is not "trusted" the system stores the fingerprint in a temporary table.
5. The "trustCalculator" module sends the temporary fingerprint *ID* to NONIO's "noPhishService" through *SOAP*.
6. The "noPhishService" *Web Service* sends the trusted fingerprints' *IDs* back to "trustCalculator".
7. The "trustCalculator" stores the trusted fingerprints and cleans the temporary table, including the signatures that were considered to be not "trusted".

4.2. *TCP/IP* and Network study

NMAP

We tested out *NMAP* by calling it from a *PHP* page simulating the signature collecting tool. Once the user's *IP* is retrieved, the *PHP* script calls *NMAP* sending the *IP* as a parameter.

Being an active tool, the time it took to run and give results was not in accordance with the plan regarding performance. For that reason, the use of this option was discarded.

P0f

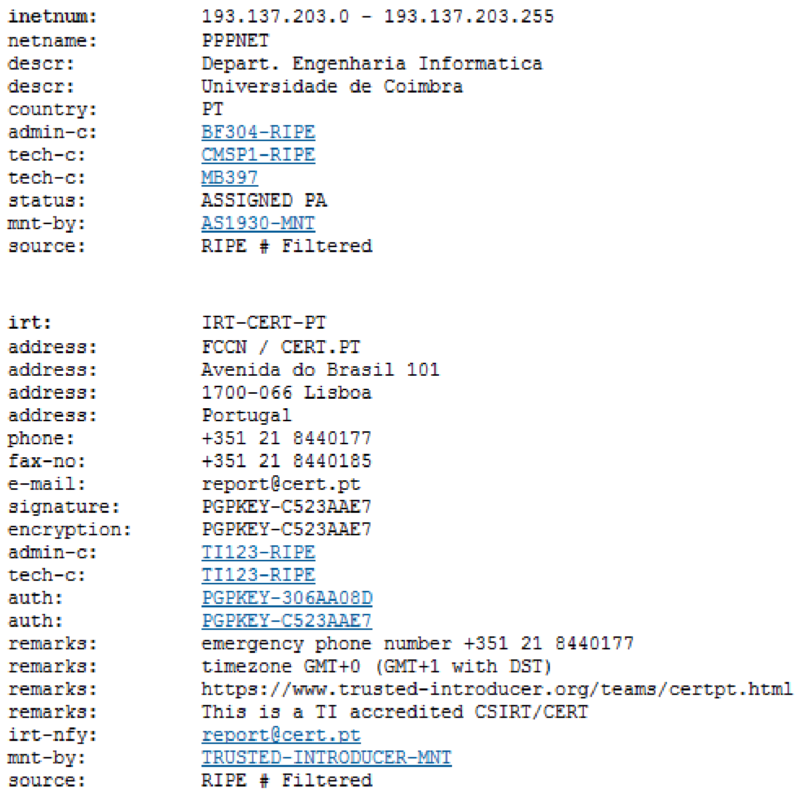
Once we discarded the active tool, we tested out *p0f*. We installed *p0f* in the noPhish server and saved the accesses registry to a log file. Since we cannot run p0f on demand, we had to find a way of accessing the log with the fingerprints when needed. This mechanism would only be called in some situations, not being part of the defined signature collector module.

The idea was to create a method that would retrieve the log file and compare the signature of certain *IP* with a database of *TCP/IP* fingerprints.

This module was not included in the final architecture, since we altered the work plan in order to proceed with the integration part of the project with NONIO. The fact is that the integration was supposed to start in the beginning of the second semester. However the final agreement regarding the integration process was only reached in Mai, in the middle of the semester (Figure 1), in a phase where the plan excluding integration was already defined.

Network data

A script that calls the *whois* client installed on the server and retrieves the information was developed. That information is then parsed to different attributes.



```
inetnum:        193.137.203.0 - 193.137.203.255
netname:        PPPNET
descr:          Depart. Engenharia Informatica
descr:          Universidade de Coimbra
country:        PT
admin-c:        BF304-RIPE
tech-c:         CMSP1-RIPE
tech-c:         MB397
status:         ASSIGNED PA
mnt-by:         AS1930-MNT
source:         RIPE # Filtered


irt:            IRT-CERT-PT
address:        FCCN / CERT.PT
address:        Avenida do Brasil 101
address:        1700-066 Lisboa
address:        Portugal
phone:          +351 21 8440177
fax-no:         +351 21 8440185
e-mail:         report@cert.pt
signature:      PGPKEY-C523AAE7
encryption:     PGPKEY-C523AAE7
admin-c:        TI123-RIPE
tech-c:         TI123-RIPE
auth:           PGPKEY-306AA08D
auth:           PGPKEY-C523AAE7
remarks:        emergency phone number +351 21 8440177
remarks:        timezone GMT+0 (GMT+1 with DST)
remarks:        https://www.trusted-introducer.org/teams/certpt.html
remarks:        This is a TI accredited CSIRT/CERT
irt-nfy:        report@cert.pt
mnt-by:         TRUSTED-INTRODUCER-MNT
source:         RIPE # Filtered
```

Figure 21 - whois data

The better approach would be to add these attributes (Figure 21) to the fingerprint. But before doing so we would have to go through all the process of retrieving new data and performing a new data analysis to understand the relevance of those attributes as part of fingerprint. However, this approach was also left aside in the final architecture since the integration with NONIO proceeded.

4.3. Testing

Synthetic Tests

TestCreator

In the initial phase of the project, it was necessary to test the calculation process. Since we had several signatures and we still didn't integrate the application, we had to make synthetic tests so we could verify the effectiveness and efficiency of the "trustCalculator" module.

We developed a tool which consists in a web page, that lists and modifies existing signatures, and that let us create new ones.

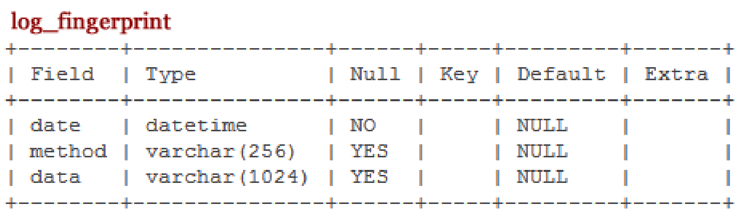
We then created fingerprints that simulated upgraded browsers, browsers with new installed fonts and plugins, and also completely different browsers.

Once the created fingerprints were stored in a new table, we ran a *java* script that calls the trust calculator tool and saves the results in a log file.

Integration

Logs

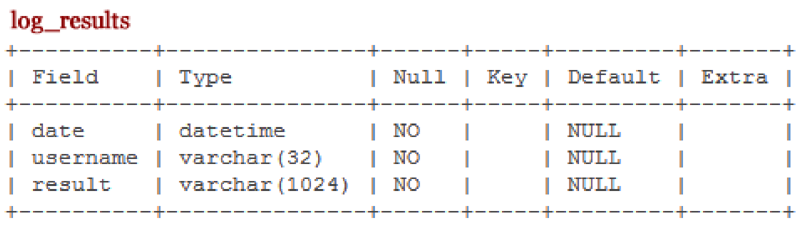
In order to analyze the results obtained after the integration, we created a series of log tables. These tables not only keep records of problems, but also store logs and results of each system's module.


log_fingerprint

```
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| date   | datetime      | NO   |     | NULL    |       |
| method | varchar(256)  | YES  |     | NULL    |       |
| data   | varchar(1024) | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
```

Figure 22 - log fingerprint table

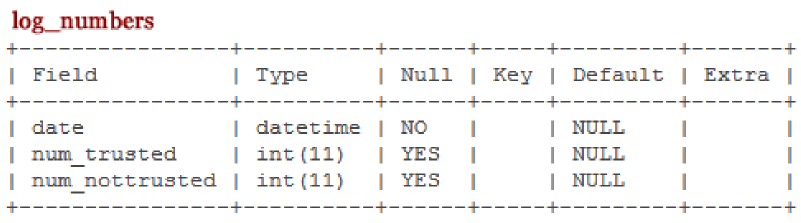
The "log_fingerprint" table is used to store *SOAP* accesses and problems occurred in each one of the system's module (Figure 22). This table helped solving problems when we tested the system.


log_results

```
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| date     | datetime      | NO   |     | NULL    |       |
| username | varchar(32)   | NO   |     | NULL    |       |
| result   | varchar(1024) | NO   |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
```

Figure 23 - log results table

In the "log_results" table (Figure 23) are stored the results of the trust calculation process, indicating whether the signature of a particular user was considered "trusted", "suspicious" or "untrusted".


log_numbers

```
+----------------+----------+------+-----+---------+-------+
| Field          | Type     | Null | Key | Default | Extra |
+----------------+----------+------+-----+---------+-------+
| date           | datetime | NO   |     | NULL    |       |
| num_trusted    | int(11)  | YES  |     | NULL    |       |
| num_nottrusted | int(11)  | YES  |     | NULL    |       |
+----------------+----------+------+-----+---------+-------+
```

Figure 24 - log numbers table

The "log_numbers" table (Figure 24) is for storing the number of "trusted" and "not trusted" fingerprints after the NONIO's Web service calls the "trustCalculator" web service.

In this way, each day, we can make an accounting on the results, telling exactly how many users accessed the system and how many of those were considered "trusted" or not.

Testing phases

The integration tests were phased. In this way, it was possible to test each module individually in its several and successive phases, to check the communication between each module and finally ending the process testing all the architecture components.

Initially the "signatureCollector" was tested by sending the data collected from the browser to the "trustCalculator". Then a log was generated to register the *SOAP* requests, and in that way it was possible to check the correction of the data (Figure 22).

The next step was to invoke the "calculateTrust" method that logged the result obtained from the trust calculation of each signature (Figure 23). The signatures' *IDs* that should later be sent to the "nophishservice" were also saved.

Then, the all process was repeated, adding the invocation of "nophishservice", sending the "untrusted" signatures *ID* and storing them in the "tmp_fingerprint" table (Figure 13). NONIO's system generated an email that was sent to us and NONIO, with the information that was later meant to be sent to the users as defined in the architecture (See 3.4. integration approach).

In the next stage, the *IDs* of "trusted" signatures were sent every 10 minutes by "nophishervice" to the "trustthisbyid" method. A log was generated in "trustcalculator" that recorded those IDs, calculating the number of trusted and "untrusted" signatures.

In the end, the production version was deployed. NONIO then started to send the trusted signatures at 00 hours of each day, and emails were sent to the user´s about the possible suspicious misuse of their credentials (See 3.4. integration approach).

Chapter 5
Results and Validation

5.1. Synthetic tests

In the initial phase of the project, a sample of 4874 signatures was collected. Using the tool "testCreator" we generated several synthetic signatures in order to test the "trustCalculator" and obtain their level of trust.

Thus, the synthetic tests ran showed that we can successfully identify a false user accessing the system (e.g. one who is accessing with phished credentials) 100% of the times.

Description	Example	Rate
True Positive	Phisher with valid credentials is blocked	100%
False Positive	Legitimate user with valid credentials is blocked	>0%
True Negative	Legitimate user with valid credentials is allowed	<100%
False negative	Phisher with valid credentials is allowed	0%

Table 1 - Synthetic tests

In some cases a legitimate user was also blocked. This happened in cases where the fingerprint has changed in such a way that no longer matched with any other on the database. This is an expected behavior. If the user's fingerprint changes in such way, he/she must prove his/her identity (See 3.4. integration approach).

In order to determine the actual rate of false positives and true negatives we needed to check the log files resulted from the integration with NONIO.

5.2. Integration

Before proceeding with the integrated systems' deploy, we collected and stored new signatures of the user´s who accessed NONIO between the end of Mai and June. 147387 signatures were collected during this period.

This data was stored so that the users already had registered signatures and wouldn't be considered "untrusted" in the first accesses following deployment. In this way, there already was a data base to signature comparison.

After deployment, from 1192 accesses to the "inforestudante", 1014 signatures were considered "trusted", 15 were "suspicious" and 163 were identified as "untrusted".

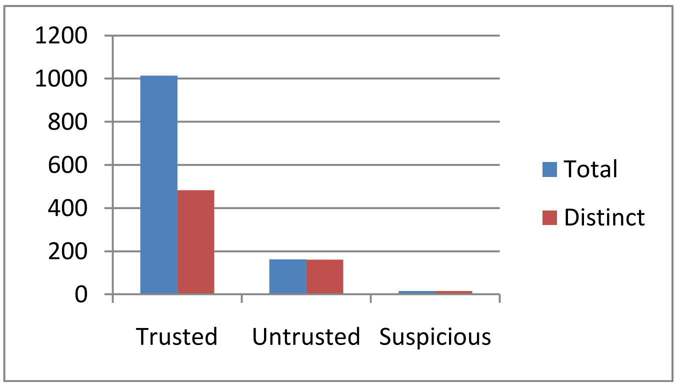


Figure 25 - Number of results

From that, 483 were distinct "trusted" signatures, 161 "untrusted" and 15 were "suspicious", summing up a total of 659 distinct users (Figure 25).

About 12% of the "untrusted" signatures represents users that didn't have any signature registered in the system, what can be explained by the fact that during the period in which the signatures were being collected and stored, those users didn´t access "inforestudante". The remaining percentage represents users who are using a different browser than the ones registered in the data base.

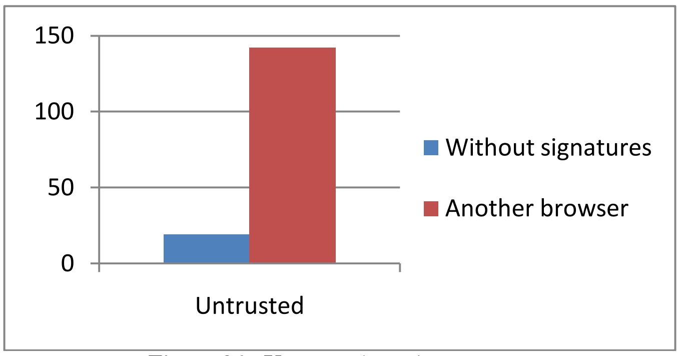


Figure 26 - Untrusted results

It was now necessary to verify if between the "untrusted" signatures there was any attacker or malicious user. To answer this question it would be necessary to analyze the user´s responses to the warning emails they received from NONIO (See 3.4. integration approach).

Unfortunately, it was not possible to collect this data since the integration could only be ended when it was already impossible for NONIO to obtain and send this data in a time that would allow meeting the deadline gave to the delivery of the Thesis. Thus, regarding this matter, the results are, therefore, presented in the assumption that there was no attackers access.

So being, with that presumption one must consider that there might be a number of users who reported that their credentials were used by someone else, by clicking the link that was presented to them in the email sent by NONIO, for mere precaution, or just because they didn´t recall if it was them or not who accessed the "inforestudante" web page at the given time.

Nevertheless, based on common knowledge, we can say that the results that we would obtain would in fact meet the assumptions made, since even if 30 students responded by saying they didn´t accessed their accounts, it would be unlike that those 30 accesses were made by attackers.

Chapter 6
Conclusions

In this report, we presented a methodology to help detecting phishing attacks using a new approach called "browser fingerprinting". Recalling the objectives described in section 1.5, it is possible to outlook that each one of them was addressed.

We can conclude that this kind of technique is more suitable when it is used as a complement of an authentication system, helping to detect situations where a user is being victim of a phishing attack. It is true that our system is able to detect when a different browser is accessing certain account, but if we used this technique as a single authentication mechanism in an application, the attacker would start collecting and using target fingerprints allowing him to enter the system. Our system doesn't have the capability to distinguish a forged fingerprint from an original one, so if it matches, the attacker is considered to be a legitimate user.

We also think that it might be more efficient in an intranet corporation environment, where the fingerprint of the device doesn't change. So, the fingerprint evolution and management were two of the more delicate concerns in this project.

For this reason, although being a little aside from what is "browser fingerprinting", if we managed to integrate the *TCP/IP* fingerprinting component into our system, it would be harder for the attacker to simulate the *TCP/IP stack* data, since it would be collected in the server. Therefore, implementing this type of systems seems to be a good solution. Unfortunately, we were unable to implement and test this approach with more detail, regarding the changes in the time management due to the fact that the architectural approach for the integration changed several times as to what was originally planned.

The integration with a real entity, with its own active processes, can be complex, mainly because an active system cannot be broken by the integration procedure and traditionally the interactions between teams are slow. This question becomes especially relevant when considering that the integration process requires constant communication and understanding.

Taking into account the work done and the obtained results, we consider that this was a productive project, not only due to the acquired knowledge from all of the investigation and development processes, but also because it was a subject that motivated us throughout the entire project.

6.1. Future Work

Similar to what happens in the *Facebook* application (Figure 1), an interesting approach would be to include a "security" section in the inforestudante page so that the users could see the browsers that accessed to their account in the last days/weeks. In that way, after the user receives the warning e-mail, he/she could check the details of the accesses to his/her account.

The "signatureCollector" module should be implemented directly in the inforestudante page. Not only for performance issues but also because the data access and retrieval must not fail. Being running inside an *iframe* and calling an external server through *SOAP* requests, the process is slow and susceptible to errors. Thus, it would be easier to guarantee an effective and efficient data retrieving process.

The obtained results shows that a considering large amount of “untrusted” signatures comes from users who don´t have a registered signature in the system. In order to overcome this detected problem in the implementation of techniques like browser fingerprinting, it is possible to approach some solutions such as:

- To create a system in which the user accepts to participate by activating a security functionality ;
- The system itself detects the first users´ application login access. This would create a star point so that the user would be, from start, associated to a valid signature that would evolve from there on.

Acknowledgments

Thanks to Marco Jorge (*marcobjorge@gmail.com*), my partner in this project in the first semester, that helped in the implementation and discussion of the project.

For their help, related to the integration with *NONIO*, I would like to thank Prof. Marco Vieira (*mvieira@dei.uc.pt*), Pedro Pinto (*plpinto@dei.uc.pt*) and André Capitão (acapitao@student.dei.uc.pt).

Thanks to Professor Mário Zenha Rela (*mzrela@dei.uc.pt*) for being my project advisor, providing solutions and guidance throughout the project.

References

[1] Robson, Daniel. 2010. "A Brief History of phishing". (*http://www.brighthub.com/internet/security-privacy/articles/82116.aspx*).

[2] Reid E.Carl . 2009. "History of phishing". (*http://www.allspammedup.com/2009/02/history-of-phishing/*).

[3] Larkin,Erik . 2010. "Browser Fingerprints: A Big Privacy Threat", PC Wold.

[4] Eckersley, P. 2009. "How Unique Is Your Web Browser?". Electronic Frontier Foundation.

[5] Angwin, J., Valentino-Devries, J. 2010. "Race is On to 'Fingerprint' Phones, PCs" (*http://online.wsj.com/article/SB10001424052748704679204575646704100959546.html?mod=WSJ_hp_MIDDLENexttoWhatsNewsThird*). The Wall Street Journal

[6] ThreatMetrix. "Device fingerprinting and fraud protection white paper". (*http://www.scribd.com/doc/5342718/Device-Fingerprinting-and-Online-Fraud-Protection-Whitepaper*)

[7] Popov, Lev. 2010 "Staying in Control of Your Facebook logins". *(https://blog.facebook.com/blog.php?post=389991097130*). Facebook blog

[8] Valentino-Devries, J. 2010. "'Evercookies' and 'Fingerprinting': Are Anti-Fraud Tools Good for Ads?". (*http://blogs.wsj.com/digits/2010/12/01/evercookies-and-fingerprinting-finding-fraudsters-tracking-consumers/*). The Wall Street Journal

[9] Business Wire. 2010. "Better Advertising announces open data parternship". *(http://www.businesswire.com/news/home/20101203005590/en/Advertising-Announces-Open-Data-Partnership-Designed-Give)*

[10]Dunaway, Gavin. 2010. "Fingerprint and privacy". *(http://www.adotas.com/2010/12/answers-served-bluecavas-norris-talks-device-fingerprinting-and-privacy/)*

[11]Microsoft. "Tracking Protection". *(http://ie.microsoft.com/testdrive/Browser/TrackingProtection/Default.html)*

[12]Project Honeynet. 2002. "Know your enemy: passive fingerprinting". *(http://old.honeynet.org/papers/finger/)*

[13]TCP/IP stack fingerprinting. *(http://en.wikipedia.org/wiki/TCP/IP_stack_fingerprinting)*

[14]Eckersley, P. 27 January 2010. "A primer on information theory and privacy". (*https://www.eff.org/deeplinks/2010/01/primer-information-theory-and-privacy*). Electronic Frontier Foundation.

[15]Schneider, T.D. 14 April 2007. "Information theory primer with an appendix on logarithms". (*http://www.lecb.ncifcrf.gov/~toms/paper/primer/primer.pdf*). National Cancer Institute.

[16] Approximate String Matching. *(http://en.wikipedia.org/wiki/Approximate_string_matching)*

[17]Cunningham, Pádraig, "CBR: Strengths and Weaknesses", Department of Computer Science. Trinity College Dublin, Ireland.

[18]Case based Reasoning. (*http://en.wikipedia.org/wiki/Case-based_reasoning)*

[19]Glasser, Thomas. 2000. "Intrusion Detection FAQ: TCP/IP Stack Fingerprinting principles". (*http://www.sans.org/security-resources/idfaq/tcp_fingerprinting.php*)

[20]Zalewski, Michal. 2006. "The new p0f". *(http://lcamtuf.coredump.cx/p0f.shtml)*

Appendix A
Data analysis

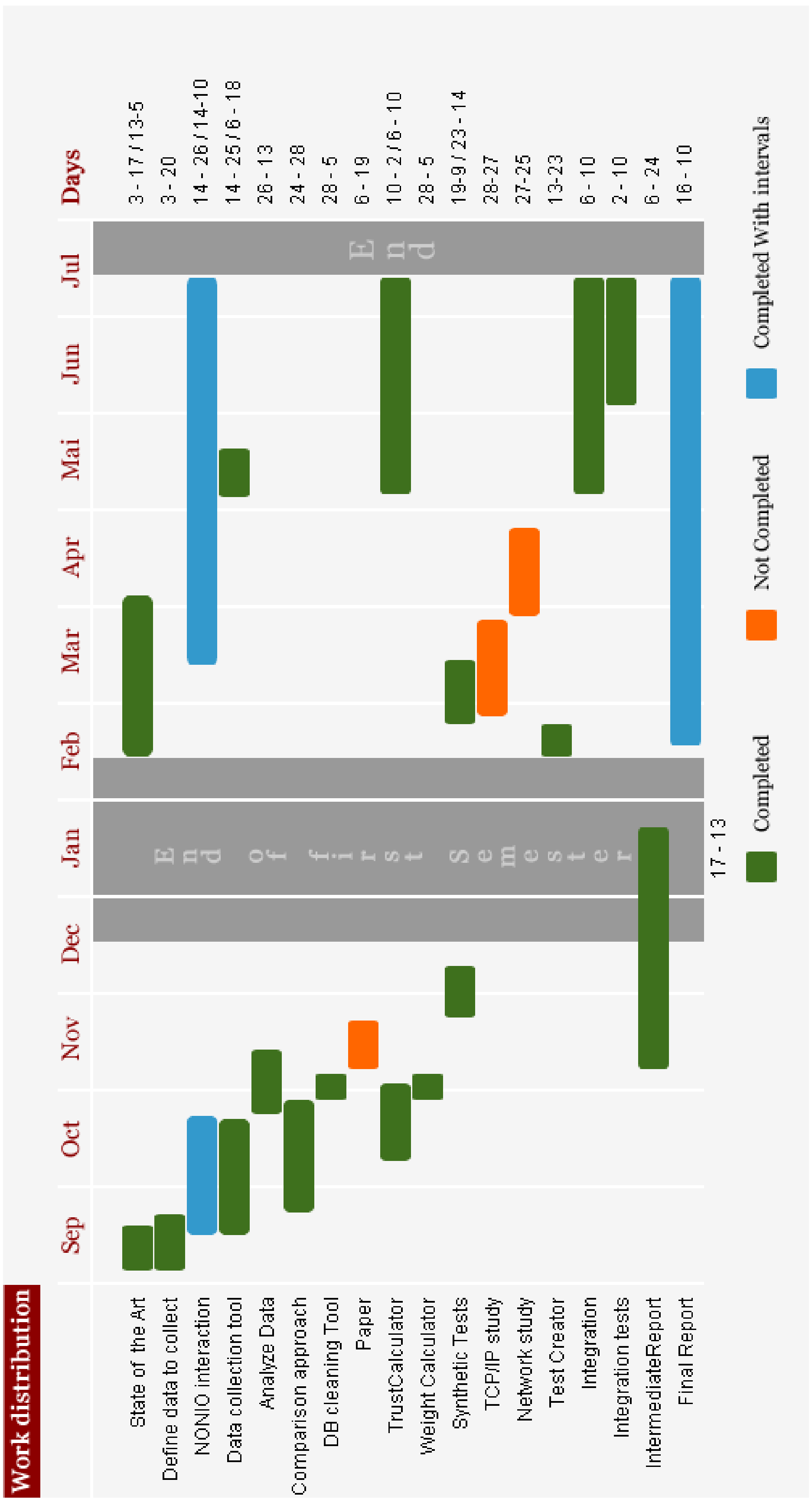
Description	Entropy (w/out Nulls)	Entropy (w/ Nulls)	Fill Rate	# Values Filled	# Distinct Values
User Agent	7,80	12,28	0,99	4808	1026
Application Code Name	0,00	0,29	0,77	3757	1
Application Minor Version	1,10	1,14	0,77	3752	11
Application Name	1,01	1,07	0,77	3757	3
Application Version	6,03	4,96	0,77	3757	620
Browser Language	1,22	1,23	0,77	3752	12
Cookie Enabled	0,84	0,85	0,98	4783	2
CPU Class	0,94	1,01	0,77	3752	4
Platform	0,58	0,73	0,77	3757	13
System Language	1,11	1,14	0,77	3752	12
User Language	1,15	1,18	0,77	3752	14
Plugins	8,82	6,35	0,77	3757	1509
Taint Enabled	0,89	0,97	0,77	3756	5
Mime Types	7,80	6,33	0,77	3755	1492
HTTP Accept Headers	3,14	3,11	0,98	4785	132
Screen	4,58	4,44	0,95	4630	135
Timezone	0,48	0,52	0,83	4022	20
System Fonts	8,86	3,67	0,31	1502	741
LSO Cookies Enabled	0,00	0,45	0,18	875	1
HTML 5 Support	2,52	2,23	0,77	3755	46
IP Address	9,68	9,55	0,98	4783	2095
Country	0,16	0,18	0,98	4783	22
Mobile access	0,06	0,08	0,98	4783	6
Samba User Info	6,46	0,62	0,06	286	121
Vendor	1,77	1,66	0,77	3752	10
Product	0,92	1,00	0,77	3752	2
Vendor Sub	1,07	1,11	0,77	3752	9
Product Sub	2,62	2,31	0,77	3752	50
Language	1,96	1,80	0,77	3752	19
Java Enabled	0,52	0,57	0,79	3849	2
System Panel Location 1	0,39	0,59	0,77	3749	3
System Panel Location 2	1,31	1,30	0,77	3747	5
Javascript Enabled	0,51	0,53	0,98	4783	2
HTTP Accept Charset	1,65	1,59	0,98	4783	12
HTTP Accept Encoding	1,65	1,64	0,98	4783	8
HTTP Accept Language	3,00	2,97	0,98	4785	77

Description	Entropy (w/out Nulls)	Entropy (w/ Nulls)	Fill Rate	# Values Filled	# Distinct Values
Flash Enabled	0,86	0,89	0,80	3900	2
Audio/Video HW Disabled	0,03	0,45	0,18	875	2
Has Accessibility	0,36	0,51	0,18	875	2
Has Audio	0,00	0,45	0,18	875	1
Has Audio Encoder	0,03	0,45	0,18	875	2
Has Embedded Video	0,00	0,45	0,18	875	1
Has IME	0,26	0,50	0,18	875	3
Has MP3	0,00	0,45	0,18	875	1
Has Printing	0,03	0,45	0,18	875	2
Has Screen Broadcast	0,00	0,45	0,18	875	1
Has Screen Playback	0,01	0,45	0,18	875	2
Has Streaming Audio	0,00	0,45	0,18	875	1
Has Streaming Video	0,00	0,45	0,18	875	1
Has Video Encoder	0,03	0,45	0,18	875	2
Flash Language	0,92	0,62	0,18	875	6
Local File Read Disable	0,03	0,45	0,18	875	2
Manufacturer	0,92	0,62	0,18	875	7
Operating System	1,88	0,79	0,18	875	17
Pixel Aspect Ratio	0,38	0,52	0,18	875	18
Player Type	0,03	0,45	0,18	875	2
Screen Color	0,03	0,45	0,18	875	2
Screen DPI	0,05	0,46	0,18	875	3
System Version	0,82	0,60	0,18	875	17
Region	2,44	2,42	0,98	4783	44
City	3,74	3,71	0,98	4783	166
Available Processors	1,28	0,65	0,13	624	5
Java Total Memory	1,33	0,66	0,13	626	17
Java Maximum Memory	1,33	0,66	0,13	624	15
Java Version	2,73	0,90	0,13	624	11
Java Vendor	0,14	0,46	0,13	624	2
Java Vendor URL	0,14	0,46	0,13	624	2
Java Class Version	0,00	0,44	0,13	624	1
OS Version	1,80	0,74	0,13	624	8
OS Name	1,79	0,74	0,13	624	6
OS Architecture	0,27	0,48	0,13	624	4
IE plugins	10,27	8,24	0,77	3755	1771

Appendix B
Selected Attributes

Description	**Used for Template Retrieval**	**CBR Weight**
User Agent		20,34
Plugins		22,78
Mime Types		21,12
HTTP Accept Headers	x	-
Screen	x	-
System Fonts		10,51
HTML5 Support		7,15
Vendor	x	-
Product Sub		7,69
Language	x	-
HTTP Accept Charset	x	-
HTTP Accept Encoding	x	-
HTTP Accept Language	x	-
City		10,41
OS Name	x	-

Appendix C
Work Distribution


Work distribution
Sep
Oct
Nov
Dec
Jan
Feb
Mar
Apr
Mai
Jun
Jul
Days
State of the Art
3 - 17 / 13-5
Define data to collect
3 - 20
NONIO interaction
14 - 26 / 14-10
Data collection tool
14 - 25 / 6 - 18
Analyze Data
26 - 13
Comparison approach
24 - 28
DB cleaning Tool
28 - 5
Paper
6 - 19
TrustCalculator
10 - 2 / 6 - 10
Weight Calculator
28 - 5
Synthetic Tests
19-9 / 23 - 14
TCP/IP study
28-27
Network study
27-25
Test Creator
13-23
Integration
6 - 10
Integration tests
2 - 10
IntermediateReport
6 - 24
Final Report
16 - 10
17 - 13
End of first Semester
End
Completed
Not Completed
Completed With intervals