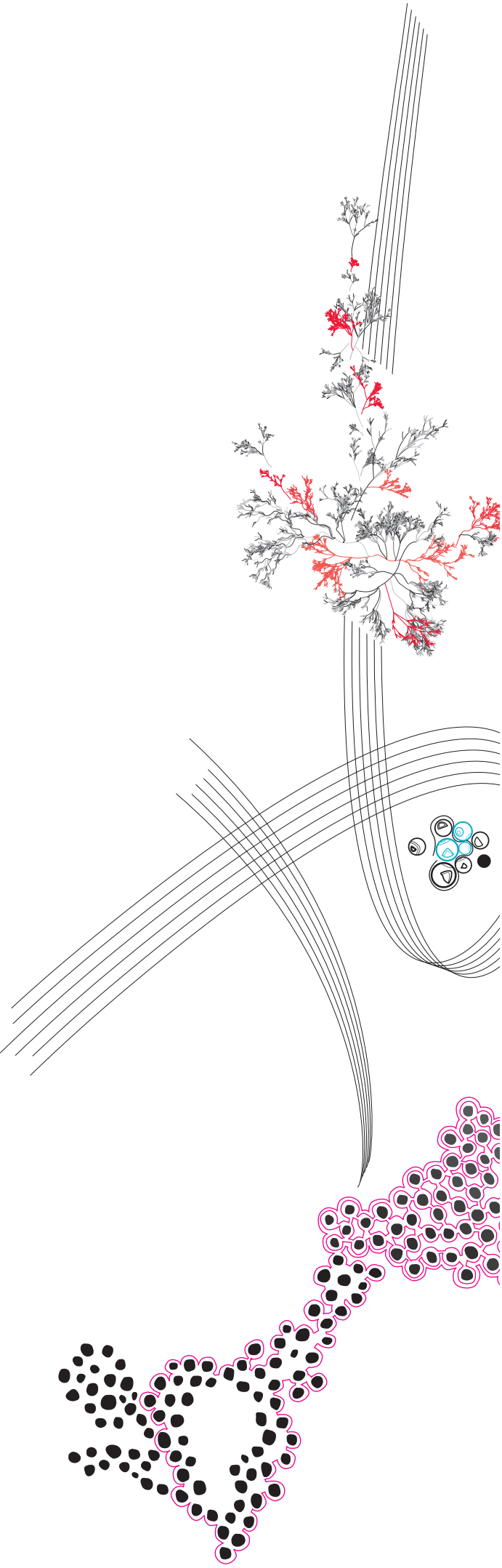


The Generation of Booter (black)lists

Master thesis

Joey de Vries
j.devries-1@student.utwente.nl

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE (EWI)
CHAIR: DACS (DESIGN AND ANALYSIS OF COMMUNICATION SYSTEMS)

EXAMINATION COMMITTEE
Prof. dr. ir. Aiko Pras,
Dr. Mena B. Habib,
Jose Jair C. Santanna, M.Sc.
Justyna J. Chromik, M.Sc.
Khelghati, S.M, M.Sc.

UNIVERSITY OF TWENTE.

Abstract

Distributed Denial-of-Service (DDoS) attacks have been a very effective tool for skilled attackers to disrupt online services. In recent years a new phenomenon appeared that offers similar attack capabilities to unskilled users at the *press of a button*. These so called **Booter** websites that offer *DDoS-as-a-service* allow anyone to execute attacks for less than USD 5. Previous research tried to mitigate these growing concerns by dynamically generating a blacklist to prevent (or warn) the average unskilled user from getting access. Our research aims to improve the blacklist generation. A more extensive web crawler is developed to find potential online Booters from Google search engine and other sources not indexed by Google (such as black-market forums and online video-platforms). We additionally develop a more accurate Booter classifier that is improved upon with a machine learning algorithm. Finally, we update the Booter (black)list on a daily basis to give the most accurate information about Booters to security specialists and interested researchers.

Contents

1 Introduction

A large array of Distributed Denial-of-Service (DDoS) attacks is still difficult to overcome [1] [2]. Usually these type of attacks are performed by skilled attackers that have extensive knowledge in the fields of network programming and security [3]. However, nowadays DDoS attacks are offered as an online service to users with close to zero knowledge in the aforementioned fields, allowing anyone to perform a DDoS attack for less than USD 5 [4]. These online services called *Booters* pose a serious threat to small and medium sized networks and services [5] [6] [7] [4] [8]. The number of DDoS attacks has increased significantly at the end of 2013 [9] with recent numbers again indicating a significant growth in DDoS attacks at the end of 2014 [10]. Not surprisingly, these numbers are influenced by the growing presence of Booters [11].

Earlier research [12] proposed a mitigation scheme that prevents or warns visitors at the access level in the form of a Booter blacklist. Research shows more than 53% of Booter users being unskilled users [4] making this mitigation scheme effective against the Booter phenomenon. The generated blacklist shows promise but is as of today not effective enough for practical considerations; a better, more reliable scheme has to be envisioned.

1.1 Objectives

The goal of the research is to **improve the dynamic generation of Booter (black)lists** introduced by [12]. The aimed result of our research is to block or prevent amateur users from obtaining access to Booters in the form of a blacklist. The generated blacklist will furthermore prove useful for further research into the Booter phenomenon.

To accomplish our goals we have defined the following research questions (RQ) as the basis of the research:

- **RQ1:** Which Booter website characteristics were not addressed by previous work [12]?
- **RQ2:** How to effectively generate a list of potential Booters?
- **RQ3:** What classification approaches can be used to improve the Booter classifier proposed by [12]?

1.2 Contribution

There has been little research in the area of Booters and only a single report [12] discussed the possibilities of an automated blacklist. By focusing on the access level of Booters (discussed in chapter 2) we aim to greatly reduce the clientele of Booters which, as a result, would result in large revenue losses for Booter owners, causing them to disband their operations. The research is split into two parts: first we extensively aim to characterize a Booter website to develop and improve a web crawler (discussed in chapter 3) and second, as the web crawler will inevitably also find non-Booter websites we build a web classification system to accurately differentiate benign websites from Booter websites (discussed in chapter 5). By improving the proposed blacklist generation we aim to develop a system with acceptable accuracy levels for practical considerations (described

in section 2.4). This blacklist and its accompanying system can and have been used by network corporations and researchers to help mitigate the phenomenon that is Booter.

Our research provides the following contributions:

- **Part 1**: a web crawler that obtains potential Booters found in search engines, video platforms, and underground hacker forums.
- **Part 2**: a classifier is developed that filters the obtained list of potential Booters giving valuable insight in the possibilities of automatic Booter verification.
- **Overall**: a comprehensive list of Booters to be used for blacklisting purposes or future research.

The resulting list of Booters is primarily intended as a blacklist, but also serves as a valuable resource for future research into the Booter phenomenon. For that reason the title of this research parenthesized blacklist as (black)list, as similarly done by [12].

This page intentionally left blank.

Part I
The Crawler

This page intentionally left blank.

2 Understanding Booters

To mitigate the Booter phenomenon we first have to understand Booters themselves, their operations, their market and how they apply themselves globally. This chapter presents an overview of Booters together with an extensive analysis of the blacklist's effectiveness.

Distributed Denial-of-Service (DDoS) attacks are characterized by an explicit attempt to prevent the legitimate use of a service [13]. DDoS attacks disrupt online services by over-flooding network or system resources either by brute-force or by system or protocol vulnerabilities. DDoS attacks require sophisticated networking knowledge combined with vast amounts of network resources to successfully execute [14].

Since a vast amount of knowledge is required to execute DDoS attacks of significant scale the attacks were reserved to knowledgeable attackers and only occurred on high-value targets [15]. However, starting in 2010 [7] DDoS attacks were offered to the average unskilled users in the form of an easy-to-use web interface. These websites allowed anyone to execute a DDoS attack for a relatively modest price (less than USD 5 [4]) simply at the *press of a button*. These *Booter* websites became part of what is often referred to as *Crimeware-as-a-service* [3] where common customers can easily access and acquire criminal tools and methods for personal use.

Booters consist of a public website and multiple (infected) systems at the core of their technical infrastructure. Together with its clients and attack targets we can model the Booter attack infrastructure as presented in Figure 1.

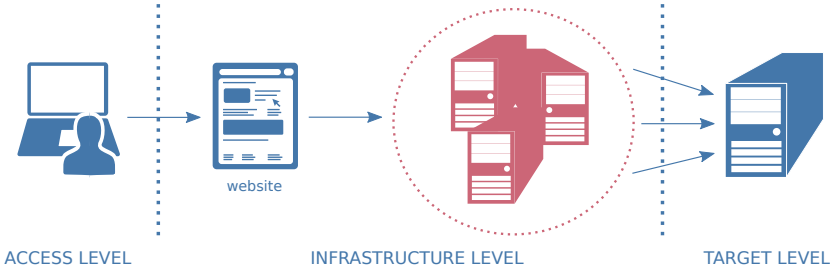


Figure 1: Booter attack infrastructure.

The Booter attack infrastructure can effectively be modeled as a large chain where the chain is only as strong as its weakest link [16]. There are three distinct layers of a Booter's infrastructure; we briefly discuss each one in reverse order.

- **Target level** - portrays the target system(s) of DDoS attack(s) launched by a Booter service. The target level includes high-bandwidth servers, website hosts and home networks. DDoS attacks at the target level are widely researched [13] [17] [18] [19] as attacks are easily identified due to their large volumetric scale. Most DDoS defense mechanisms discussed at the target level are however not as effective against large scale attacks [12] [14].
- **Infrastructure level** - describes the Booter website and the rented or

infected machines operated by the Booter owner. There is little known about the exact infrastructure of Booters, but analysis on leaked databases [4] [20] showed a combination of distinct servers and web-shells being the primary resources for Booter services [5]. Web-shells [4] are DDoS scripts hosted on (infected) machines accessed via HTTP/GET or HTTP/POST.

- **Access level** - describes the entry point of Booter services. Booters offer a user-friendly online front-end for launching DDoS attacks at a small price attracting amateur users. Research by Chromik [12] aimed to mitigate the Booter phenomenon by preventing users access to the Booter websites using a blacklist that would effectively sit in the link between the user and the Booter website. The approach by Chromik showed promise, but the concluded accuracy of 88% is not sufficient enough for practical considerations.

Mitigating the Booter phenomenon at any of the three architecture links will completely mitigate the Booter phenomenon. As the blacklist approach by [12] showed promise and effectively disrupts the first chain of the Booter attack infrastructure we will primarily focus on the access level. To get a solid grasp of the access level we first discuss the opening of the infrastructure chain: its users.

2.1 Booter users

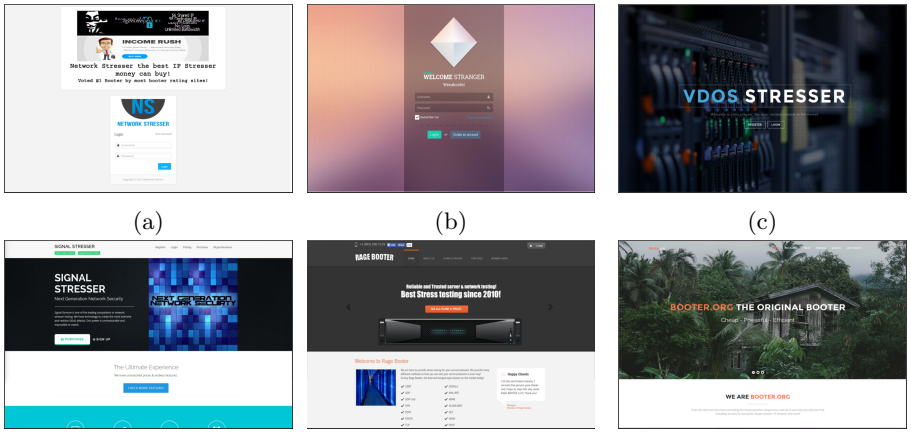
Booters target users that want to perform DDoS attacks. A large demographic of its users either lack the required knowledge or the required resources to execute these attacks. Literature shows that the target audience primarily consists of online gamers [20], willing to invest financial resources to disrupt their opponent's network [21]. Because most online video-games instantly declare defeat to a player losing its network connection, disrupting their opponents' service is very advantageous. By crawling black-market hacking forms [22] we confirmed that Booter services are frequently used for disrupting online games. Also, a large majority of DDoS attacks are small in scale showing these attacks do not target high-valued organizations [23]. Common examples are disrupting opponents' networks in a player-versus-player game e.g. League of Legends, DotA and Counter-Strike or disrupting a service to settle scores or grudges in server-based games like Minecraft. These forums are frequently visited by so called script-kiddies, a term coined by hackers to define unskilled want-to-be hackers that use tools without explicitly understanding them. A further confirmation of unskilled users being the majority of the Booter's audience is the frequent advertisement or crimevertisement [3] of Skype and website resolvers that resolve the target's IP-address based on their account name and domain name respectively; something a skilled attacker would have no difficulty with. Additional target services include small to medium sized websites or individual home networks [24].

Literature [4] confirms, by analyzing leaked databases, that the customer-base consists over 53% of unskilled users. A significant portion of the customers utilize Booters without taking any significant precautions like accessing Booters using a VPN [4] making their activities more easily detectable. This also highlights the probable effectiveness of our research. Due to the fact that a

large portion of a Booter's customer-base consists of unskilled users a blacklist approach will prove effective against mitigating the Booter phenomenon. To generate such a blacklist we need to understand how Booters target or attract their users and take a closer look at how Booters present themselves.

2.2 Presentation

Large-scale DDoS attacks used to be offered and advertised in the underground scene, commonly visited by malicious users [3]. Booters however, target average users without any experience in the underground hacker scene. Therefore Booters wish to be found as easy and publicly as possible. For this reason, Booters make a best-effort to associate themselves with public search engines like Google [12], openly advertise themselves in black market forums e.g. `hackforums.net`, social media like Twitter [25] and video platforms similar to Youtube [26], and aim to create a visually pleasing front-end for their services. A Booter's aim is not to preserve secrecy, but is in contrast the public advertisement of its services. Booters have a distinct visual appearance with slight deviations among them. In Figure 2 we list 6 different Booter websites:



(a) (b) (c)

(d) (e) (f)

Figure 2: Booter appearance

The images were taken from the Booters' landing page i.e. the first visible page presented after visiting its domain. From these 6 Booters three distinct layouts can be identified, which naturally extend to other Booter websites.

1. (2a, 2b) The landing page shows only a login form with minimal information about the Booter.
2. (2c) The landing page shows even less information and acts merely as an eye-pleaser before continuing to the actual Booter content. Contains the Booter's title and button(s) to continue (often login and register buttons).
3. (2d, 2e, 2f) The landing page contains detailed information about the Booter described in a visually pleasing way. Information typically entails what they represent, their features, subscription plans and contact or support details.

Booters operate under a collection of different names. In the literature they are often coined the term *DDoS-as-a-service* [20] [4] or *DDoS-for-hire* [7]. Similarily, online researchers and security journalists adapt the same terminology for describing Booters [27] [28]. Booters advertise themselves as either a Booter or a stress-testing service, commonly abbreviated to *Stresser* [29] [30] [20].

The term Stresser is often used by Booters due to its more legitimate appeal [12]. Stress testing is characterized as deliberately placing a system under heavy stress to test a system's endurance or durability. Stress testing can have a legitimate purpose if used on one's own networks or supported by contractual agreements, but since Stressers do not validate the attacker nor the target there is no practical difference between a Booter and a Stresser [12] [31]. Furthermore, a DDoS attack launched by a Booter will, regardless of the target, have a negative impact on unaware networks in between. Given the nature of Booters that advertise themselves as legitimate stress-testing services, this will hold no ground in a court of law [32]. The illegitimate appeal becomes even more apparent in the next section where we delve into the offered attacks.

2.3 Attack types

Booters offer a wide array of DDoS attacks that can be categorized according to a relatively simple, but effective taxonomy. We categorize Booter attacks by their nature, their target and whether they apply reflection techniques.

The nature of an attack can be categorized by the OSI-model's layer being abused as either the *transport layer* or the *application layer*. Transport layer attacks typically include sending large numbers of UDP or TCP packets by exploiting vulnerabilities and features in transport layer protocols. Application layer attacks involve exhausting significant amounts of system or network resources by exploiting application-specific protocols.

The target of an attack can be categorized as *network* attacks or *system* attacks. Network attacks are congestion based methods in that they utilize a large number of (compromised) machines to send a massive amount of junk IP traffic to the victim, preventing further communication [1]. System attacks involve directly exhausting a system's resources, often paired with application layer attacks.

Finally, a further categorization is made by listing whether an attack is a *direct* attack or a *reflected* attack. Direct attacks target a system directly while reflected attacks abuse protocols to re-direct attacks from multiple networks. A specific flavor of reflected attacks are Distributed Reflection Denial-of-Service (DRDoS) [19] attacks. DRDoS attacks abuse the ability of network protocols to spoof the source IP-address of the packets making it difficult to track where the packets originated from. By spoofing the source IP-address to the victim's IP-address the servers will reply towards the victim's network. This type of attack is combined with amplification properties such that a small request packet returns a much larger *amplified* response. The simultaneous abuse of multiple amplifiers permits a highly-distributed DoS attack to be conducted from a single internet uplink.

Chromik did extensive research on the attacks offered by Booters of which we list their results and supplement the list with more recent attack types offered by Booters [12] [22]. Each of the attack types are briefly categorized and grouped

by their nature. It should be noted that Booters use different names for similar attacks [4] so we will group these attacks together by their underlying technical properties.

2.3.1 Transport layer attacks

A total of 5 transport layer attacks were identified.

- **SYN flood attack** - targets a network via direct attacks and is coined under different names e.g. TCP attack, SSYN (Spoofed SYN), ESSYN (Enhanced SSYN). The attacker exploits a weakness of the three-way TCP handshake by sending TCP packets to a server with the SYN-flag set to initiate a connection. With a randomized source IP-address of the TCP packet the server will reserve network resources for the connection and request a TCP package with the ACK flag set to complete the request. Because the source IP-address is configured to a random IP-address the server will not receive a reply and as a result allocates resources that are never consumed. A large number of these TCP SYN flood requests will exhaust the server's network resources and no new TCP connections will be processed.

- **UDP flood attack** - targets a network via direct attacks. The victim receives a large number of UDP packets, which it has to respond to with the same amount of ICMP packets [30]. As a result the victim will be overloaded and no longer able to receive further requests. The source IP can be spoofed as well which is why Booters often use the name SUDP (Spoofed UDP).

- **DRDoS attack** - targets a network using reflective properties. Booters extensively make use of the amplification properties of these type of attacks to generate massive amounts of bandwidth with a relatively low number of sources [33]. Protocols used for DRDoS attacks vary but common protocols are: DNS (Domain Name Server), CharGen (Character Generator) [30], SSDP (Simple Service Discovery Protocol), NTP (Network Time Protocol) and Gaming Server protocols e.g. Quake [34] or Steam [19]. Using DNS, specifically DNSSEC, amplification rates of 179 times are possible [35], while amplification using NTP servers allow for amplification rates up to 4670 times [19].

- **UDP lag** - similar to the UDP flood attack, but instead of sending traffic at a constant rate it is sent in specific intervals, degrading the network [36]. The victim's network operations remain operational, but with slow and unreliable network traffic [30].

- **Dominate** - a modified version of the SSYN attack with a different rotation of TCP flags. The aim of this attack is to bypass certain DDoS protection services e.g. OVH, Voxility and Blacklotus [37].

2.3.2 Application layer attacks

A total of 7 application layer attacks were identified.

- **GET attack** - targets a system directly by sending a large amount of HTTP GET requests, aiming to exhaust the web-servers resources.
- **POST attack** - similar to the GET attack, but sends partial HTTP POST requests with a body of specified size following. The resulting POST body is then sent at an extremely slow rate to slow down the webserver.
- **Slowloris** - an HTTP application layer attack directly targeting a system comparable to the SYN flood attack [20]. Several connections are opened and partial requests are sent, never completing the full request [36]. The aim is to exhaust a web-server's resources by initiating a large number of uncompleted requests. Frequently a new HTTP header is sent, keeping the connections alive while adding up to the request.
- **RUDY (R-U-Dead-Yet?) attack** - similar to the Slowloris attack, but halts sessions by using never ending POST requests with large header values [30].
- **ARME attack** - a directed application layer attack that exhausts the SWAP memory of Apache web-servers using a vulnerable Apache exploit [30]. Once an Apache web-server is out of SWAP memory it will kill all processes.
- **XML-RPC** - a network-targeted DRDoS attack using Wordpress's pingback feature which is a feature enabled by default in Wordpress installations and difficult to disable [38]. Attackers are able to reflect large amounts of traffic to target networks using Wordpress's pingback feature.
- **Joomla** - similar to the XML-RPC DRDoS attack. The Joomla attack uses a vulnerability in Joomla's Google Maps plugin [39] to reflect large amounts of traffic.

Of particular interest are DRDoS attacks which have the highest impact of all listed attacks [40]. It should come as no surprise that these type of attacks have recently had a very high impact, specifically using the NTP protocol [41] [42]. Crawling black-market hacker forums [22] show that these attacks are the most-advertised attacks (primarily due to their high impact). Furthermore, Booters might offer different attacks than executed. For example, one Booter advertised an attack as 'UDP' but in reality performed a DRDoS attack based on DNS and CharGen [20].

Extensive database analysis showed that UDP and TCP flood attacks are the most frequently used attacks [4] which is surprising since one would expect DRDoS attacks to be more frequent. This could (partly) be explained by Booters mostly using web-shells to execute their attacks which does not lend itself well for DRDoS attacks [4]. However, visiting black-market forums [22] do show the most popular Booters specifically using dedicated servers for executing DRDoS attacks. This again strongly highlights the relative impact of Booters. Booters are already able to disrupt most online services while still not yet utilizing their complete attack potential. With the upcoming rise of DNSSec and the, as of today, not extensively used NTP DRDoS attacks Booters have a strong potential to take down large networks with relative ease [43]. These

upcoming dangers of Booters further states the relevance of a high-accuracy blacklist where we discuss its effectiveness in the next section.

2.4 Blacklist considerations

With a more complete understanding of the Booter phenomenon we can discuss the probable effectiveness of a blacklist. The main idea behind the blacklist is to prevent the target users of Booters in getting access to Booter websites. This can either be accomplished by entirely blocking the website or by displaying a warning message e.g. "*This website is blacklisted for questionable behavior, any further traffic will be extensively monitored by your local federation agencies. Proceed at own risk or contact ... for further details.*" which will scare a large portion of the users in proceeding. Due to Booters losing a significant amount of their users, the Booter phenomenon should reduce significantly as well. However effective such a blacklist may be, there are several considerations to take into account when using blacklists [12].

First, a blacklist is only as effective as it is correct and up-to-date [44]. If a blacklist is not up-to-date it will fail to recognize the more recent Booters, still allowing users to freely use Booter systems for a limited period of time. In addition, it might have blacklisted a URL which is no longer a Booter website, but a legitimate website [45]. Also, if a blacklist is not correct it might wrongly classify a legitimate website as an illegitimate Booter website disallowing anyone access. It is therefore of utmost importance to create a correct blacklist with as much accuracy as possible while also frequently updating the list. Because this research aims to dynamically generate such a blacklist it will be relatively easy to keep the list up-to-date since no manual labor is required. However, the downside of this approach is that it will significantly reduce the correctness or accuracy of the blacklist if incorrect heuristics are assumed. This directly states the importance of our research as we aim to significantly improve the blacklist accuracy for use in a practical setting.

Second, a blacklist will not prevent users with sufficient knowledge from accessing Booters since they can still use a Virtual Private Network (VPN) to mitigate the blacklist in use. However, due to the majority of users being relatively unskilled a blacklist will still block most users.

Third, blacklisting URLs effectively filters the web which is considered unlawful [46]. Web filtering is however still frequently applied in practice such as in child pornography filtering and spam filtering. Given the illegitimate practice of Booters this can apply as well. This does require a general assurance that such a blacklist will not filter legitimate URLs as that will be met with heavy objections.

Blacklisting will not be a single golden key to Booter mitigation since it will not be able to completely prevent access to Booter websites. However, it is a promising approach diminishing the Booter phenomenon by a significant extent. Furthermore, such a blacklist is not only effective for denying access, but also proves useful for further research into the Booter phenomenon. Automated blacklist generation does require a strong set of heuristics which is what we will focus on in the next chapter.

This page intentionally left blank.

3 Booter website characteristics

Automated blacklist generation requires an extensive set of heuristics to differentiate Booter websites from non-Booter websites. This chapter discusses characteristics specific to Booter websites used for Booter classification.

In our research we develop a web crawler to obtain a list of potential Booters. This list will inevitably contain non-Booter websites thus there is a need for differentiating Booter websites from non-Booter websites. In order to differentiate a Booter website from their legitimate counterparts it is important to carefully describe their characteristics.
Having a complete set of attributes that uniquely characterize a Booter significantly improves the accuracy of dynamically generated blacklists. The characteristics described will be part of a feature vector set used to classify websites as Booters; this will be discussed in more detail in chapter 5. We discuss Booter characteristics as found in the current literature and from our own research, discuss their practical implications, their effectiveness and where relevant their strength against attacker mitigations. A significant portion of the Booter characteristics is based on the characteristics used by [12]. Our aim is to update the characteristics used with more accurate and relevant characteristics; all supplemented characteristics are denoted with an asterisk. Due to the large amount of characteristics they are grouped in three categories: structure-based characteristics, host-based characteristics and content-based characteristics.

3.1 Structure characteristics

Structure-based characteristics revolve around the general website structure of Booters. We identify a total of 4 structure characteristics: number of pages, URL type, average depth level and the average URL length.

a. Number of pages is the number of web pages a Booter website consists of. As identified by [12] Booters have a relative small number of pages, often no more than 10 to display general information and account controls. The number of pages identified did not exceed over 50 making this a strong characteristic for differentiating Booters from most other websites. However, a weakness of this characteristic is that Booters can mitigate this fairly easy by adding a large number of *fake* pages. This weakness can be solved by crawling all visible URLs of a Booter to determine its number of *referenced* pages as a Booter will most likely not link to their *fake* pages.

b. URL type is the type of URL required to reach the landing page of a Booter which can be categorized in 1 of 3 different URL types [12]. Table 1 shows a summary of the URL types witnessed from Booters.

URL Type	URL
1	booter-example.tld or booter-example.tld/login.php
2	domain.tld/booter-example/
3	booter-example.domain.tld

Table 1: Booter URL Types

URLs of type 1 (hostname with or without filename) as [12] pointed out is the preferred URL of most Booters, whereas type 2 (part of other website) and type 3 (subdomain) are much less common and often indicate webpages that are not Booters, but provide information about Booters.
Due to a large portion of Booters having a URL type of 1 this characteristic might prove useful for differentiating between actual Booters and articles or blog-posts discussing Booters. Booter owners could try to mitigate this characteristic by restricting themselves to URLs of type 2 and 3. However, URLs of type 1 are more distinct and attractive to Booter owners so Booters will likely remain available via type 1 URLs.

c. Average depth level - The depth level of a website indicates the maximum amount of inbound hyperlinks to follow to reach any webpage. This can be obtained by crawling all inbound hyperlinks and tracking the crawl attempts required to reach a page. We limit the amount of crawl attempts to a total of 50 distinct inbound hyperlinks as Booter websites do not have more than 50 pages (see 3.1.a). Initial observations show Booters having a low average depth level making this a strong characteristic. Mitigating this characteristic is difficult, since forcing larger page levels complicates the Booter website's structure which Booter owners prefer not to.

d. Average URL length* - larger websites like online news or popular blog websites have a relatively complicated back-end structure which results in large and more complicated URLs. For instance, the URL to a news article might include in its path the domain, news category, author and the date of publishing. Compared to Booters with short and relatively simple URLs this URL length contrast proves an interesting characteristic. The average URL length of a website's content is used to build a score as linearly interpolated between the smallest average URL length and the maximum URL length as found in a training dataset [47].

The structure of a website conveys valuable information about the type of website. In the next section we additionally take a look at the business and personal aspects of Booters and Booter owners for further identification.

3.2 Host characteristics

Host-based characteristics revolve around the organization or managerial aspects of Booters. Booters have a total of 5 interesting host-based characteristics which makes them stand out among other websites: domain age, domain reservation duration, WHOIS private, DPS and page rank.

a. Domain age - Most Booters have a short life span [12] and first started appearing in 2010 [7]. As a result, websites with a domain registered before 2010 can almost certainly be excluded as a Booter. Similarly, due to their short life span the age of a domain acts as an important characteristic of a Booter [48]. It is not uncommon for a Booter to go offline within a year of service.

b. Domain reservation duration* - Due to a Booter's short life span Booter owners tend to register their domains for a duration not more than 1 year, the

minimal registration duration, while legitimate website owners often register their domain for much longer periods. By querying the expiration date of a domain and subtracting this from the current date we get the remaining reservation duration of a domain. It is expected that most Booters have a reservation duration of less than 1 year.

c. WHOIS private - WHOIS is a query/response protocol that provides information about domain names and their nameservers at request. This information includes register information like name, address, e-mail and information about the domain including registration date and used nameservers [48]. It is not difficult to visualize that Booter owners prefer to keep their contact details anonymous. Domain registrars i.e. services that sell domains, offer to anonymize contact details for an added price which results in the register information becoming private [49]. Querying WHOIS data of a Booter domain will likely result in private contact details which could serve as a strong characteristic for the domain belonging to a Booter.

d. DPS - A DDoS Protection Service (DPS) offers a popular mitigation method against DDoS attacks [7]. A DPS offers protection by rerouting large volumes of traffic to multiple filtering centers [50] to filter all DDoS traffic before rerouting it to the original target. Around 2011 Booters realized they can use their services not only to sell, but also to clear out their competition by attacking other Booters on the market [7] [27]. Due to Booters becoming a target of their own product, almost all Booters are registered with a DPS [31]. This thus gives another clear characteristic of Booters, one being registered with a DPS.

e. Page rank* - Online services like Alexa [51] build statistics determining the popularity of a website. A website with a page rank of 1 depicts it the most popular website. As Booters are visited by a very specific group of users (see section 2.1) the page rank of a Booter is likely to be relatively low. Websites discussing Booters like security blogs, corporation websites and news sites will generally have a higher page rank as they target a more general audience. For instance, at the time of writing the popular security blog `www.krebsonsecurity.com` has an Alexa rank of 26805 while one of the more popular Booters has an Alexa rank of 616446; a significant difference. As a Booter characteristic the page rank helps us differentiate between more popular websites mentioning Booters and the less popular Booter websites themselves [52] [53]. As a metric, we take the highest Alexa rank listed among Booters, add a small offset and measure a binary value whether a website has a higher or lower rank; a higher rank would indicate the website being benign.

Host-based characteristics describe a lot about the organizational aspects of Booter websites. Booter websites also consist of a significant amount of identifying information in the content of their web pages which is what we focus on in the next section.

3.3 Content characteristics

Booters are largely defined by their structural and host-based characteristics, but their content gives valuable hints as well. We identified a total of 6 con-

tent based characteristics: average content size, outbound hyperlinks, category-specific dictionary, resolver indication, Terms of Services page and login-form depth level.

a. Average content size* - In contrast to other websites, Booters either have close to zero content or brief commercial-like statements regarding their services. As a result the average amount of content on a web page is likely to be small, specifically pages other than the landing page (e.g. pricing, FAQ) [47]. As Booters prefer to keep their content simple and minimal this will be an interesting characteristic that is not easily mitigated. It should be noted that when discussing average content size we specifically talk about textual content, neglecting the underlying code (Booters do have large code footprints per page as they make extensive use of JavaScript for their visuals). Another consideration is that some Booters paste large descriptive sections of text on their landing page mostly aimed at boosting their search results, but hide the actual text for its viewers. This characteristic only takes the visible content into account, ignoring all hidden textual content.

b. Outbound hyperlinks* - Outbound hyperlinks are characterized as hyperlinks that link to domains other than the current domain [47] [54]. Booters offer services in a highly competitive market and as such rarely mention services other than their own. As a result Booter websites hardly ever include hyperlinks to external domains. This is a strong characteristic as this strongly differentiates Booter websites from blogs and news websites that include a vast array of outbound hyperlinks [44].

c. Category-specific dictionary* - Booter websites generally advertise their services with words specific to their category e.g. Booter, pricing, powerful, methods. These category-specific *keywords* introduce an interesting concept of identifying the likelihood of a website being a Booter by the occurrence of these keywords [54] [55] [56]. Further research is required to determine the effectiveness of this characteristic as it is difficult to determine the proper set of keywords, not all Booters display textual content and it becomes difficult to differentiate between articles about Booters. By introducing relative occurrence of a keyword compared to the page's total content size and by further analyzing keywords with machine-learning text classification approaches [57] this characteristic could prove relevant.

d. Resolver indication* - As discussed in section 2.1 Booters recurrently offer service resolvers that *resolve* the underlying IP-address of several services like domain-names, Skype accounts and video-game accounts. As Booters blatantly advertise these services this serves as a feasible characteristic [6]. Care should be taken to differentiate between articles discussing resolvers and actual Booter websites. As we already analyze category-specific keywords it is trivial to simultaneously detect resolver keywords as well.

e. Terms of Services page - Booter websites try to hide their illegal activities by dedicating one of their website pages to a Terms of Services (ToS) page. It is a legal agreement composed by a set of rules that users need to follow to use a Booter's services [12]. As the presence of a ToS page indicates a

website selling a service this proves an interesting characteristic.

f. Login-form depth level* - As briefly indicated in section 2.2 Booter websites repeatedly start with a login form or have it available within one click. Therefore, the depth level (see 3.1.c) of a page containing a login form potentially indicates a Booter website as they are often found at a lower depth level compared to other websites [44].

In this chapter we identified a total of 15 Booter website characteristics in contrast to a total of 9 Booter website characteristics as determined by previous research [12]. The Booter website characteristics will aid us in verifying whether a website identified by our crawler is indeed a Booter website.

Given these Booter website characteristics and an arbitrary website we can calculate a score for each characteristic. The total of 15 individual scores returns a feature vector set that aims to contain noticeable differences for distinction between Booter feature vectors and non-Booter feature vectors. In the next section we discuss the web crawler developed to find potential Booters where we use the Booter website characteristics to generate a feature vector for each potential Booter website.

This page intentionally left blank.

4 The crawler

For the generation of a Booter (black)list a list of potential Booter URLs has to be collected first. This chapter discusses the web crawler system developed to retrieve potential Booters from multiple online sources and how their respective characteristic scores are generated.

As discussed in section 2.2 Booters publicize themselves as openly as possible. As an effect, Booters are publicly found on video platforms, black-market forums, and indexed by major search engines like the Google search engine. We take these properties to develop a web crawler that crawls and indexes the aforementioned online web sources to obtain a list of Potential Booter Domain names (PBDs). As this will inevitably result in not only a list of Booters, but also a considerate amount of non-Booters we have to differentiate these with a Booter classifier as discussed in chapter 5. In this section we discuss the implemented crawler system, focus on its results from scraping individual websites and finally discuss the normalization transformation of Booter website characteristics.

4.1 The crawler system

The crawler system takes as input a list of search keywords and a list of 15 Booter website characteristics as determined in chapter 3. The crawler system incorporates two smaller sub-systems, each having a specific command set for its specified purpose. A basic overview of the crawler system is displayed in Figure 3.

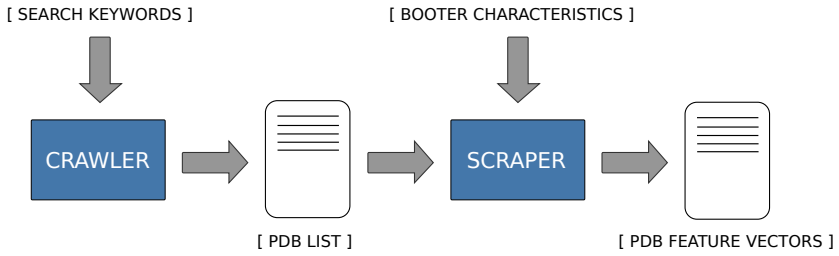


Figure 3: The crawler system

- The *crawler* sub-system takes as input a list of keywords and crawls multiple online resources for PBDs. A PBD is uniquely characterized by its subdomain(s) and domain name only like *booter-example.webhost.tld* or *booter-example.tld*. A list of all the obtained PBDs is sent to the *scraper* sub-system.
- The *scraper* sub-system processes each individual PBD as obtained from the *crawler* and generates relevant statistics based on the given list of 15 Booter website characteristics. The data generated from each of the PBDs is used by the verifier as discussed in chapter 5.

4.1.1 Requirements

We identify 5 main requirements necessary to develop a crawler system for the generation of Booter (black)lists:

1. The crawler sub-system should return a large and comprehensive list of PBDs. No crawler will ever be able to return a complete list of its targets, but we require a best-effort approach to finding PBDs from the web.

2. The crawler sub-system should promote extensibility such that other online resources can easily contribute.

3. The scraper sub-system should generate a feature score for each of the 15 Booter website characteristics given a PBD.

4. The crawler system should be undetectable as an automated process and aim to reproduce a legitimate browser as best as possible.

5. The crawler system should be efficient as the Booter (black)list is required to be up to date.

These requirements list the minimum set of features a crawler system should be composed of. In the next section we discuss the implementation and design of the crawler system based on the aforementioned requirements.

4.1.2 Implementation and design

Both the crawler sub-system and the scraper sub-system are developed in the Python 3.5 programming language. For HTTP web requests we made extensive use of the *Requests* [58] and *Cloudflare scrape* [59] package of the Python library repository. Furthermore, we also used the *Pythonwhois* [60] package for WHOIS data retrieval and the *etree* [61] and *urllib.urlparse* [62] library for parsing HTML and URLs respectively.

To satisfy the first requirement the crawler sub-system is divided into several smaller crawlers where each operates under a common feature set. Each of the individual crawlers crawls a specific online website resource: public search engine, video platform or black-market forum. Below we list a complete reference of the resources crawled together with relevant details specific to each crawler.

- **Google search engine** - Crawls all search results returned with the following keywords: *Booter*, *Stresser* and *DDoSer* that were carefully selected by analyzing current literature and observing the amount of relevant results. Each keyword supplies the crawler with a maximum total of around 800 PBDs [1]. It should be noted that Google hosts an API [63] specifically designed for returning a list of URLs given a search keyword. Their official API however restricts any keyword search up to a total of 64 results which was deemed incomplete for satisfying the first requirement. For this reason a custom crawler was developed for the Google search engine bypassing the restrictions.

[1] Google restricts its search results up to around 800 URLs per keyword

- **Youtube** - Crawls the Youtube video-platform [26] for Booter-related advertisements with the following keywords: *Booter*, *Online Booter* and *Stresser*. Each of the returned video items are crawled by their description and titles to obtain potential Booter URLs. A total maximum of around 500[2] PBDs are obtained per keyword.

- **hackforums.net** - Crawls the *Marketplace* section of this popular black-market forum to obtain PBDs. Each individual forum post with relevant tags is crawled to find any PBD hidden in the top post's content.

Resources like the Google search engine return the largest number of results and are most plausible for the assumption of our research: amateur users use public search engines to find Booter websites [12]. However, as websites first need to be indexed by search engines to become publicly available it can take a considerable amount of time for them to appear on the public search engine list. For this reason we similarly crawl video-platforms and black-market forums as these additionally find recently publicized Booter websites not yet indexed by public search engines. As the individual crawler sub-systems operate under a common feature set it is trivial to extend the crawler system with additional input sources befitting the system's second requirement. As building a complete web crawler that features all relevant sources is not the focus of our research we conclude the aforementioned three sub crawlers sufficient for our purposes.

The third requirement involves scraping a PBD to generate feature scores for each of the 15 Booter website characteristics. Each of the 15 Booter website characteristics is individually processed as extensively described below.

1. **Number of pages** - We obtain the number of pages by an automatic scrape process for each PBD. This is accomplished by recursively retrieving all inbound hyperlinks of a web page and traversing each of them until there are no new pages left or a maximum crawl limit is reached. We configure a maximum crawl limit as large websites have tens of thousands of web pages and crawling each of them is disproportional to our research. Previous research [12] demonstrated that the number of pages of Booter websites does not exceed 50, which we therefore set as the maximum crawl limit. All unique inbound hyperlinks found count as the PBD's number of pages characteristic.

2. **URL type** - In section 3 we discussed a total of three different URL types. As a PBD is only classified by its subdomain(s) and domain name we discard all URL type 2 classifications (domain.tld/booter-example), but significantly boost performance due to a numerous amount of domains being scraped only once. We make the assumption that Booter URLs do not appear as type 2 URLs as it is non-trivial to programmatically determine whether a URL is of type 2. This does render the Booter URL type characteristic with less precision, but by manually verifying the amount of type 2 URLs we concluded this was negligible. Because of this we denote all URLs of type 2 as URLs of type 1 (booter-example.tld).

[2]Similar to Google's search engine, Youtube restricts the maximum number of results up to around 500 results per keyword.

3. **Average depth level** - The average depth level is obtained by maintaining a minimum depth index while scraping each web-page. Each web-page followed from an inbound hyperlink gets its depth level incremented, unless a smaller depth level was previously traversed. For instance, if a web page named *pricing.php* can minimally be traversed by following a path of 2 inbound hyperlinks from the PBD's landing page, the depth level of the page is scored a 2.0.

4. **Average URL length** - The average URL length is obtained by averaging the length of all found inbound hyperlinks [3] from the scrape process of a single PBD.

5. **Domain age** - The domain age characteristic is scored as the number of days that have passed since the PBD's registration. The domain age is calculated by first querying the *creation date* key from the PBD's WHOIS records and then subtracting this date from the current date.

6. **Domain reservation duration** - The domain reservation duration is calculated as the difference in days between the current date and the PBD's *expiration date* as obtained from its WHOIS records.

7. **WHOIS private** - The WHOIS private characteristic is a binary value of 1.0 when a PBD's WHOIS records are private and 0.0 when its records are publicly available. To determine whether a PBD's WHOIS records are private we apply a heuristic. A list of keywords retrieved from all private Booter WHOIS records is used to validate whether a WHOIS record's registrant name, e-mail and organization fields contain any of the private registration keywords; think of keywords like *private*, or privacy-protection company names like *whoisguard* and *protecteddomainservices*.

8. **DPS** - The DDoS Protection Service (DPS) website characteristic is a binary value that is 1.0 when a website is under DPS protection and 0.0 without DPS protection. Similar to the WHOIS private characteristic its score is determined from parsing the WHOIS nameservers for DPS related keywords. For instance, if a nameserver of *Cloudflare* is found, the largest occurring DPS provider, it is clear the PBD is under DPS protection. If no DPS nameservers are found, we do a further analysis to see if visiting the PBD results in a re-direct page often accompanied with detecting web crawlers and other automatic traffic. As these re-direct pages are often associated with DPS providers we check if there is any present and apply a keyword search on its content to determine if it is a DPS page and if so, denote the PBD a DPS score of 1.0.

9. **Page rank** - The Page rank website characteristic is determined by querying the Alexa [51] page rank servers. As some PBD websites are relatively new and not yet publicly known to search or ranking engines the Alexa page rank of such a website is deemed undefined.

10. **Average content size** - The average content size characteristic is determined by the content size of each of the scraped pages. The content of a

[3] All inbound hyperlinks include the website's domain name as to not bias the results

page consists of all textual content found in HTML paragraph *p* tags and between any child-element found in HTML's *div* tags.

11. **Outbound hyperlinks** - Similar to how we determine all inbound hyperlinks for further scraping, the outbound hyperlinks are obtained by retrieving all scraped hyperlinks that direct to different domains. The outbound hyperlinks website characteristic is the average number of outbound hyperlinks found from all scraped pages of a PBD. We also included a list of exception domains that we did not take into account as an outbound hyperlink since they have no economic relevance to a Booter website. These include social-media domains like `facebook.com` and `twitter.com` and other hyperlinks irrelevant to the characteristic like external image files, Skype protocol URLs or e-mail URLs.

12. **Category-specific dictionary** - The category-specific dictionary characteristic takes a list of keywords specific to Booter websites e.g. *stress*, *booter* or *ddos* and calculates the average percentage of occurrence on a scraped page's content. The category-specific dictionary only operates on the landing page as that page best reflects a PBD's intentions.

13. **Resolver indication** - The resolver indication characteristic checks from a list of resolver keywords like *skype* or *xbox* and denotes a score of 1.0 wherever any of the keywords occurs on the PBD's landing page or 0.0 otherwise.

14. **Terms of services page** - The presence of a terms of services page is determined in two steps. First, we validate whether any of the URLs clearly contains a terms of services phrase like *http://.../tos*. Second, we compare each of the scraped pages' content for the presence of terms-of-service-specific key phrases like *terms and conditions* or *we have the right*. Any confirmed presence of terms of services results in a score of 1.0, otherwise a score of 0.0 is appointed.

15. **Login-form depth level** - The login-form depth level characteristic is determined by retrieving the smallest depth level of any login (or register) form. The presence of a login-form (or register-form) is determined by the existence of an HTML password input field.

To satisfy the 4th requirement we have to evaluate the design choices we made with regards to sending and retrieving web requests, specifically when scraping individual websites to determine their characteristics for further classification. Scraping individual websites is a difficult feat as websites take numerous measures to mitigate automatic crawlers (often referred to as *robots* or *spiders*) in favor of performance and confidentiality. For instance, as most Booter websites are protected by DPS companies like Cloudflare, they include an intermediate launching page that checks potential crawlers by validating JavaScript operations. A large portion of online websites use similar measures to prevent automatic scripts from accessing their website content.

A solution to these crawler mitigations frequently found in the current literature is the use of Selenium [64]: a browser emulator that forwards web requests to a genuine web browser on the client system. As an actual web browser is used, all content, images, JavaScript and other browser-related functionality is

executed, making it difficult for websites to detect it as an automatic process. Chromik (2015) [12] used the Selenium web browser for automating a series of web requests to detect and scrape Booter websites; this proved a practical solution. However, as the focus of our research is purely on availability, textual content and hyperlinks there is no need to use a fully-fledged web browser for forwarding requests as this brings a considerable overhead. For instance, loading a website with Selenium will force the browser to visually process and display all results: load all referenced image, audio or video content and execute JavaScript commands; all irrelevant for our purposes. As we prefer to prevent as much overhead as possible as denoted by the 5th requirement we built our own approach to mitigating crawler detection systems. This includes header flags presenting ourselves as legitimate browsers, a JavaScript emulator and the option to re-direct to different pages where required.

The source code of the entire crawler can be found online at `https://github.com/joeydevries/Booter-black-list` and several high-level snippets of the source code can be found in Appendix A. The interested reader is invited to read the source code's documentation for further insight into each specific crawler process. In the next section we present the first preliminary results obtained from running the crawler processes.

4.2 Preliminary results

A preliminary training run of the crawler system was executed a total of 4 times between 25th and 30th of June 2015 running 2 hours by average and returning a total of 928 PBDs. Each of the PBDs was scraped by the scraper sub-system and manually verified for use in a training dataset for Booter classification (as discussed in chapter 5). The training dataset of 928 PBDs consists of 113 Booter websites and 815 non-Booter websites. In Table 2 the intermediate unprocessed values of each of the 15 Booter website characteristics (as discussed in chapter 3) is presented. As the class of each PBD was manually verified we can isolate the results between Booter and non-Booter. This gives a clear overview of the website characteristic discrepancies between Booters and non-Booters.

Characteristic	Booter			Non-Booter		
	avg.	**min**	**max**	**avg.**	**min**	**max**
1. Number of pages	7.88	1.0	71.0	981.75	1.0	21620.0
2. URL type	1.04	1.0	2.0	1.20	1.0	2.0
3. Average depth level	0.92	0.0	3.88	1.75	0.0	5.09
4. Average URL length	24.93	8.0	44.38	53.65	6.5	151.21
5. Domain age	395.96	4.0	5304.0	3564.29	7.0	10360.0
6. Domain res. duration	310.93	0.0	2000.0	812.22	7.0	3631.0
7. WHOIS private	0.73	0.0	1.0	0.28	0.0	1.0
8. DPS	0.73	0.0	1.0	0.21	0.0	1.0
9. Page rank	1.1×10^7	1.8×10^5	2.6×10^7	3.2×10^6	1.0	2.6×10^7
10. Average content size	127.00	0.0	1394.57	679.08	0.0	4801.42
11. Outbound hyperlinks	0.41	0.0	5.5	14.10	0.0	996.04
12. Category-specific dict.	0.039	0.0	0.33	0.014	0.0	0.15
13. Resolver indication	0.22	0.0	1.0	0.19	0.0	1.0
14. Terms of services page	0.47	0.0	1.0	0.44	0.0	1.0
15. Login-form depth level	1.38	0.0	3.0	2.06	0.0	3.0

Table 2: Grouped preliminary results of the Booter crawler system

As expected, after manual analysis there are clear distinctions between the results of Booters versus the results of non-Booters. For instance, we highlight the Number of pages characteristic from which it is clear it can significantly assist us in differentiating between Booter (7.88) and non-Booter (981.75) websites. In contrast, the results show almost no significant distinction for the highlighted resolver indication and terms of services page characteristic. In chapter 5 we explore different combinations of these Booter website characteristics and based on these results weaken or strengthen several of the characteristics.

The data presented is representative of the scraped websites by a maximum of 50 crawled pages per PBD. While this does fully crawl (most) Booter websites, it fails to completely gather statistics for larger non-Booter websites as a significant portion is left out. The crawl limit especially has effect on characteristics like average depth level and login-form depth level as the 50 crawl results will be biased towards lower depth levels. Furthermore, not all characteristics return valid results for each PBD. For instance, the WHOIS protocol gives no result guarantee when querying the relevant nameservers. This sometimes leads to undefined results, in which case we omit the results from the computations in Table 2.

The results displayed so far are non-normalized i.e. they are all displayed in dissimilar intervals. As the current feature vector results will produce biased results when directed to a Booter classifier we first normalize the results i.e. transform all results to a comparable equal range.

4.3 Score normalization

For classification of Booters the classifier system takes as input a feature vector of 15 Booter website characteristic scores per PBD. Given this feature vector it returns whether that feature vector represents a Booter or a non-Booter website. The Booter classifier discussed in chapter 5 assumes the feature vector to

be normalized as otherwise its result would be biased towards the larger-valued Booter website characteristics. Each of the Booter website characteristics is transformed or normalized into a binary value or a decimal value in the unit interval [0,1] before being sent to the classifier system. Not every characteristic will perfectly map to the unit interval in which case a range is defined to transform the characteristic to the unit interval. The value ranges are depicted from a combination of the dataset's scores and the assumption of certain heuristics. This does mean the heuristics assumed inevitably *closes* the score values within a certain configuration. This requires careful thought on the heuristics assumed which we carefully describe. We believe the selected configuration best represents the Booter characteristics at the date of this thesis' publishing.

We take a total of three different approaches to score normalization as some procedures are better suited than others to a specific type of characteristic. The approaches when it comes to normalizing the crawled scores are binary, linear and quadratic interpolation.

4.3.1 Binary interpolation

Six of the 15 website characteristic are binary i.e. they are either scored a 0 or 1. The normalization procedure for binary characteristics involves a decision boundary for when its preliminary values result in a score of 1. The website characteristics that require binary interpolation are URL type, WHOIS private, DPS, Page rank, Resolver indication and Terms of services page.

- **URL type** - Almost all Booters have a URL type of 1 thus a score of 1.0 is appointed for a URL type of 1. In contrast, a score of 0.0 is appointed to URLs of type 3.
- **Page rank** - Any PBD website that has a lower[4] page rank than the determined Alexa page rank of 200000 receives a normalized score of 1.0 while websites with an equal or higher page rank of 200000 are denoted a score of 0.0. We selected a page rank of 200000 as the highest Alexa rank found among Booters was 180830 and 213599 making it an effective threshold. An undefined page rank is scored similar to low ranking pages as 1.0.
- **WHOIS private, DPS, Resolver indication, Terms of services page** - These Booter website characteristics require no further normalization and are a direct translation of their original scores.

4.3.2 Linear interpolation

The remaining 9 Booter website characteristics generate scores in relatively large intervals, which we wish to normalize to the unit interval. This involves selecting a specific range of values within the large interval to map to the unit interval using a linear interpolation equation to normalize the scores. The linear interpolation equation used for score normalization is denoted in equation 4.1.

$$S_n = 1.0 - \frac{x - min}{max - min} \tag{4.1}$$

[4]Note that a lower page rank actually represents a higher number as page rank is a descending metric.

The value of *min* and *max* denote the interval range we map to the unit interval. These values are carefully selected for each individual linear website characteristic based on the training dataset and the assumption of several heuristics. Note that values outside the selected interval are clamped to 0.0 and 1.0 respectively e.g. a number of pages score of 274 which is outside the max interval is scored 1.0.

- **Average depth level** - The normalization transformation of the average depth level is a linear transformation between 1.0 and 3.0. These values of *min* and *max* were observed from the dataset as the minimum and maximum depth level of Booter websites.
- **Average URL Length** - As observed from the training dataset Booter websites have URLs of lengths between 15 and 30 with the occasional outlier, compared to an average of 53.65 for non-Booter websites. Given these observations we normalize the average URL length with *min* and *max* values of 15 and 30 respectively.
- **Domain age** - The normalized domain age is calculated as the linear interpolation in days between the current date *min* and the oldest creation date *max* found of a Booter in the training dataset which is the 28th of October 2011. For instance, a domain age of 200 days, given a *max* value of 1450 and equation 4.1 returns a normalized score of 0.862.
- **Domain reservation duration** - We observed from the training dataset that most Booters have a reservation duration less than 365 days. This confirms our hypothesis that Booters, due to their fragile nature, do not reserve or extend a domain for more than one year. Furthermore, as Booters do not instantly launch as soon as a domain is bought their observed domain reservation duration is rarely close to 0. Due to these observations we select a *min* and *max* value of 183 days and 365 days respectively.
- **Average content size** - Booter websites have a low average content size with an average of 127.00 compared to an average of 679.08 of non-Booter websites. We linearly normalize the results with a selected *min* and *max* value of 50 and 250 words respectively; these numbers were deemed most representative of Booters as obtained from the training dataset as Booters rarely have less than 50 and more than 250 words of text.
- **Outbound hyperlinks** - We normalize the outbound hyperlinks characteristic between a *min* and *max* value of 0.0 and 2.0 as we deem a website having more than 2 outbound hyperlinks on average a non-Booter website. From the Booter training set it is quite apparent that Booters rarely feature outbound hyperlinks as hypothesized in chapter 3.
- **Category-specific dictionary** - The category-specific dictionary score represents the percentage of Booter-specific keywords occurring on the landing page's content. Given the observed score values we find a *max* value of 5 percent or 0.05 and a *min* value of 1 percent or 0.01.
- **Login-form depth level** - Booter websites frequently have a login-form at a depth level of 0 and 1 compared to non-Booter websites most often

having a login-form at depth level 2. Based on these observations we linearly interpolate between a min value of 0 and a max value of 2.

4.3.3 Quadratic interpolation

Most of the Booter website characteristics are binary or follow a general linear pattern. The remaining number of pages characteristic is however better suited for quadratic interpolation, starting at a relative slow pace before increasingly stronger score reductions apply. A quadratic interpolation function is a function of the form shown in equation 4.2.

$$S_n = ax^2 + bx + c \tag{4.2}$$

The quadratic interpolation function produces a curve that places progressively greater weight on the input value x, but starts relatively slow. The motivation for using quadratic interpolation quickly becomes apparent as we describe the number of pages website characteristic normalization.

- **Number of pages** - As we deem 50 the maximum number of pages of Booter websites (excluding outliers) we have a metric to transform any amount of pages to a score between 0.0 and 1.0 by interpolating between 0 and 50 with 0 pages being a score of 1.0. However, as most Booter websites do not have 0 pages, but roughly around 5 to 10 pages we use a quadratic interpolation function that decreases the score in a slower pace at the start and in a stronger pace when it is closer to the maximum crawl limit. For normalizing the number of pages characteristic we take the quadratic interpolation equation 4.2 and select $a = -2$, $b = 0$ and $c = 1$ to get equation 4.3.

  $$y = -2x^2 + 1 \tag{4.3}$$

  Equation 4.3 produces a preferable curve, but is not yet suited for normalizing the number of pages as an x value of 1.0 returns a value below the y-axis. We first solve for $y = 0$ to retrieve $x = 0.707106781$ to transform the number of pages between 0 and 50 to $x = 0$ and $x = 0.707106781$ respectively; this produces the final equation 4.4 for normalizing the number of pages with n being the number of pages. Note that all normalized values that end up outside the normalized range are clamped between 0.0 and 1.0.

  $$y = -2 * (\frac{n}{50/0.707106781})^2 + 1 \tag{4.4}$$

  Equation 4.4 transforms any number of pages between 0 and 50 to a score between 1.0 and 0.0 respectively. The graph of the equation is shown in Figure 4.

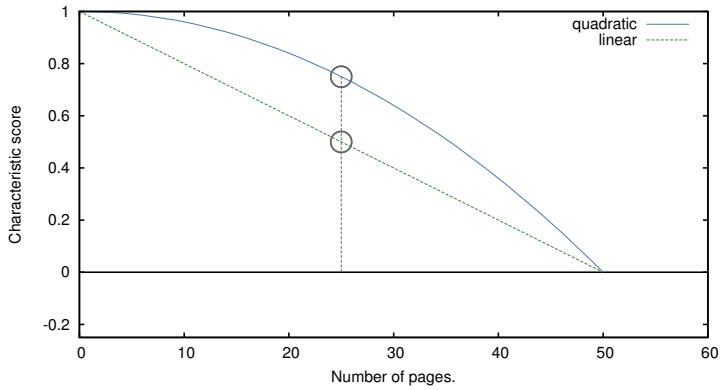


Figure 4: Normalization graph of number of pages from Equation 4.4.

From Figure 4 we can see that a website hosting a total of 25 pages is deemed a score of 0.75 compared to a score of 0.5 using default linear interpolation. As this metric favors smaller websites and largely dismisses larger websites we believe this metric better reflects the number of pages characteristic.

Each of the normalization transformations are summarized in Table 3.

Characteristic	**Transformation**	**Value range**
1. Number of pages	Quadratic	[0,50]
2. URL type	Binary	-
3. Average depth level	Linear	[0,2]
4. Average URL length	Linear	[15-30]
5. Domain age	Linear	[-, 1180[5]]
6. Domain res. duration	Linear	[183,365]
7. WHOIS private	Binary	-
8. DPS	Binary	-
9. Page rank	Binary	[200.000,-]
10. Average content size	Linear	[50,250]
11. Outbound hyperlinks	Linear	[0,2]
12. Category-specific dict.	Linear	[0.01,0.05]
13. Resolver indication	Binary	-
14. Terms of services page	Binary	-
15. Login-form depth level	Linear	[0,2]

Table 3: Normalization transformations

By applying the normalization transformations atop of the crawler dataset we get a new dataset, this time with feature vector scores that can directly be used as input to the Booter classifier. In Table 4 the normalized crawler dataset is shown, again isolated by Booter and non-Booter results.

[5]Subject to date of verification.

Characteristic	Booter avg.	Non-Booter avg.
1. Number of pages	0.93	0.23
2. URL type	0.96	0.80
3. Average depth level	0.87	0.57
4. Average URL length	0.36	0.07
5. Domain age	0.78	0.14
6. Domain res. duration	0.90	0.61
7. WHOIS private	0.71	0.29
8. DPS	0.71	0.21
9. Page rank	0.90	0.30
10. Average content size	0.70	0.16
11. Outbound hyperlinks	0.84	0.19
12. Category-specific dict.	0.49	0.24
13. Resolver indication	0.24	0.19
14. Terms of services page	0.47	0.44
15. Login-form depth level	0.52	0.27

Table 4: Normalized grouped results of the Booter crawler system

With normalized results we can still clearly see a significant distinction between Booter websites and non-Booter websites. For instance, the highlighted number of pages and resolver indication characteristic still show similar discrepancies compared to the preliminary results. The clear distinction shows promise for Booter classification as there apparently are noticeable measurable differences to differentiate Booter websites from non-Booter websites. In the next section we will use these feature vector inputs to classify a website as either a Booter or a non-Booter website where we use a large array of classification algorithms to achieve this.

Part II
The Classifier

This page intentionally left blank.

5 Booter classification

As the crawler system will inevitably produce non-Booter websites as potential Booters, a classifier system is developed to filter the non-Booter websites from the list. In this section we extensively look at this classifier system and carefully describe the classification metrics used in the process.

In order to filter the complete list of Potential Booter Domain names (PBDs) to a list of only Booter domain names a classifier system is developed. Given a 15 dimensional PBD feature vector as input, the Booter classifier system states whether this score vector is that of a Booter website or a non-Booter website. In Figure 5 a basic overview of the classifier system is presented.

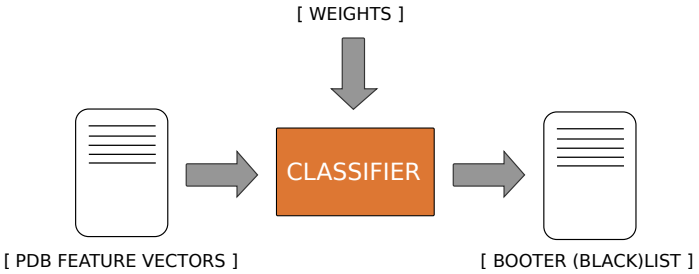


Figure 5: The Classifier system

The classifier system takes as input a list of PBD feature vectors from the crawler system together with a set of weights (we discuss weights in section 5.3.3) and returns a list of only Booter domain names. Similar to the crawler system, the classifier system is completely written in Python and the interested reader is invited to read the full source code at the research's public GitHub Repository `https://github.com/joeydevries/Booter-black-list`.

To differentiate Booter websites from non-Booter websites a total of 7 different classification metrics are implemented and extensively verified to accurately test the optimal classification method for Booter classification. We describe the classification metrics used and their motivation in detail in section 5.2. Before we discuss the classification metrics in the context of a Booter (black)list we first briefly introduce classification accuracy and the confusion matrix.

5.1 Classification accuracy

To test the effectiveness or accuracy of the proposed classifier system we need to develop a notion of classification accuracy. The classification accuracy is characterized as the probability a random PBD feature vector is correctly classified as either a Booter or non-Booter. Figure 6 shows a *confusion* matrix denoting a total of 4 classification possibilities. From the confusion matrix one can derive not only the number of times the classifier misclassifies a website, but also the type of misclassification.

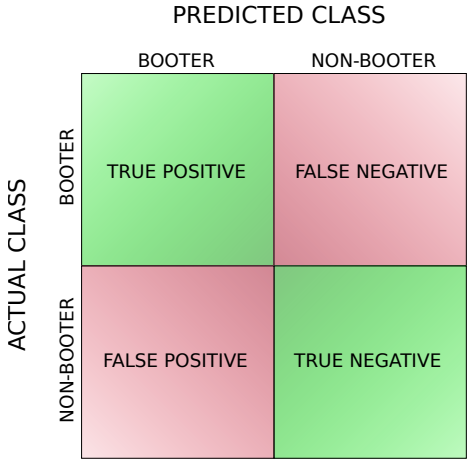


Figure 6: Confusion matrix

- **True positive** (T_P) - when a website is correctly classified as a Booter.
- **True negative** (T_N) - when a website is correctly classified as a non-Booter website.
- **False positive** (F_P) - when a non-Booter website is incorrectly classified as a Booter. This is called a Type I error.
- **False negative** (F_N) - when a Booter website is incorrectly classified as a non-Booter website. This is called a Type II error.

Ideally we would only have true positives and true negatives e.g. a perfect web filtering system would produce a diagonal confusion matrix. Given the confusion matrix we can calculate the Classification Accuracy Rate (CAR) of the classifier as the number of true positives and true negatives divided by the total sum $T(M)$ of test samples M as seen in equation 5.1.

$$CAR = (T_P + T_N)/T(M) \tag{5.1}$$

Furthermore, we also identify a total of 3 global indicators [54] derived from the confusion matrix that estimate the likelihood of error:

- **Global error rate** (G_{er}) - The number of samples incorrectly classified: $(F_P + F_N)/T(M)$. The global error rate is the complement of CAR.
- **Type I error rate** (TI_{er}) - The number of samples incorrectly classified as a Booter: $(F_P)/T(M)$.
- **Type II error rate** (TII_{er}) - The number of samples incorrectly classified as a Non-Booter: $(F_N)/T(M)$.

Several of the described classification schemes introduced in section 5.2 require configuring a threshold value at which the system decides a specific score indicates either a Booter or a non-Booter. Balancing this threshold value inclines the error samples more towards one end of the confusion matrix than the other. For instance, a low threshold value results in a higher *Type 1 error rate* as

non-Booter websites are more often classified as Booter websites. Additionally, a high threshold value results in a higher *Type II error rate* as Booter websites are less often classified as Booter websites. Given the nature of blacklists we favor a reduction of false positives over false negatives as a large number of false positives renders a blacklist inoperable (see section 2.4). In contrast, a higher number of false negatives makes the blacklist less efficient, but still practical.

The classification metrics introduced in section 5.2 are trained by the training dataset from section 4.3. Testing the accuracy of the system given the same training data produces biased results. Therefore, the accuracy of the system is tested against a total of three newly obtained datasets from the crawler system. Each of the test datasets is obtained and manually verified between the 7th and 21th of August 2015. The test dataset totals a number of 465 websites denoted in Table 5.

Dataset	**Date**	**Booters**	**non-Booters**	**Total**
Test set 1	07-08-2015	48	62	110
Test set 2	18-08-2015	57	157	214
Test set 3	21-08-2015	35	106	141
		140	325	465

Table 5: Manually classified test dataset

Against these datasets we test a total of 7 different classification metrics. The classification metrics are tested against each individual dataset and the accuracy rates are averaged between the datasets. We describe each classification metric in detail in the next sections.

5.2 Classification metrics

Given a set of 15 identified Booter website characteristics and any arbitrary PBD feature vector set of individual characteristic scores we want to evaluate whether this PBD feature vector belongs to either a Booter or a non-Booter website. Each characteristic of a Booter as described in section 3 is represented as a score in the unit interval [0,1] that depicts how much it satisfies the respective characteristic; this can either be binary or a decimal value in between. Given such a set of feature scores obtained from a single website we want to evaluate a binary result indicating whether the website is a Booter website. Several classification methods exist that given a set of identifying features can evaluate their similarity in a binary decision. By focusing on a multitude of classification algorithms we can test each individual metric and determine which classification metric is best suited for Booter classification.

As research in classifying Booters is relatively young we identify several classification methods from the research literature of phishing, (child-)pornography detection and big-data analysis. Table 6 presents an overview of classification metrics found in the literature coupled with their respective accuracies in relevant website classification literature.

Metric	Paper	Accuracy
Euclidean distance	[65] [12] [66]	79-94%
Squared Euclidean[6]	[67]	-
Manhattan distance[7]	[68] [65]	-
Fractional distance	[65] [69]	-
Cosine distance	[12]	85%
K-Nearest Neighbors	[57] [70] [71]	70-98%
Naive Bayes	[47] [55] [72] [70] [53]	74-99%

Table 6: Proposed Booter classification schemes

The Euclidean, Cosine distance, K-Nearest Neighbors and the Naive Bayes classification are frequently found in literature of phishing, Booter and child-pornography website classification. These respective classification metrics returned interesting results that could prove effective in Booter classification. The squared Euclidean, Manhattan and Fractional distance were not mentioned in any of the phishing, Booter and child-pornography literature, but proved effective when working with high dimensional feature vectors [68] [65]. As the website feature vectors are 15 dimensional, which we deem as high dimensional, these classification metrics can prove particularly effective in Booter classification.

There exists a multitude of alternative classification metrics that we did not include in the Booter classification analysis (like Hamming distance, Logistic regression and Support Vector Machines). As implementing and testing each of the available classification metrics is infeasible for the proposed research duration we selected the algorithms based on their apparent and accurate results in a low and high dimensional datasets and their relevant literature occurrence. Further tests using a different set of classification metrics is reserved for future research.

Table 7 presents a brief overview of the practical characteristics of each classification metric together with their high-dimensional (n) strength as derived from the literature.

Metric	$n > 3$	Complexity	Efficiency
Euclidean distance	weak	low	high
Squared Euclidean	weak	low	high
Manhattan distance	medium	low	high
Fractional distance	strong	medium	high
Cosine distance	medium	low	high
K-Nearest Neighbors	strong	low	medium
Naive Bayes	strong	medium	high

Table 7: Classification metric characteristics

Each of the proposed classification metrics will be extensively discussed in the next sections, including their implementation and their respective accuracy on the test dataset.

[6] Derived from the Euclidean distance metric.
[7] See footnote 6

5.3 Distance metrics

A distance metric defines the distance between two sets of features i.e. the amount two sets of features geometrically vary. The general idea of distance metrics is to represent a set of feature characteristics as an n-dimensional feature vector. Given two vectors $\bar{A}$ and $\bar{B}$, where $\bar{A} = [a_1, a_2, ..., a_n]$ and $\bar{B} = [b_1, b_2, ..., b_n]$ their distance can be calculated by several distance metrics: Euclidean, squared Euclidean, Manhattan, Fractional and Cosine distance. Distance metrics can be evaluated on binary values of either 0 or 1 or decimal values within the unit interval [0,1].

Care should be taken when using geometrical distance to carefully standardize the feature values [73]. For instance, if one of the features is measured at a scale of 100 units while the other features are measured at a scale of 1 unit, the distance would be heavily biased towards the larger-scaled feature. We take extra care to restrain the individual feature values to the same scale as it would otherwise leave the metric ineffective. This standardization or normalization was extensively discussed in section 4.3.

Additionally, the classification scheme of our research in contrast solely operates on similarity between two feature vectors $\bar{A}$ and $\bar{B}$. We therefore derive the similarity S between two vectors $\bar{A}$ and $\bar{B}$ given a distance metric D as expressed in equation 5.2. All negative values are clamped to 0.0.

$$S(\bar{A}, \bar{B}) = 1 - D(\bar{A}, \bar{B}) \tag{5.2}$$

The distance metrics involves two vectors of which feature vector $\bar{A}$ is denoted as the PBD feature vector to classify. The other vector $\bar{B}$ we refer to as the *perfect Booter vector* defined as $\bar{B}_p = [1.0, 1.0, ..., 1.0]$. By measuring the similarity between any website feature vector and the perfect Booter vector we effectively calculate the Booter likelihood score of the input PBD feature vector. We tried experimenting with this perfect Booter vector by not assuming a perfect score of only 1s, but by updating its values to take the averages of all Booter scores as found on the training dataset, and by manually selecting score values based on their Booter occurrence. Updating the perfect Booter vector did however not prove effective for significantly and consistently improving the metrics' accuracy.

5.3.1 Decision boundary

Each of the distance metrics generates a distance and equivalently a similarity score that have yet to be interpreted for classification. For each of the distance metrics a final binary decision is made based on a given threshold at which any similarity score above the threshold is deemed a Booter website. As each distance metric applies different geometrical characteristics in its equations the optimal threshold varies for each metric. The optimal threshold is characterized as the threshold T that returns the optimal CAR on a training dataset favoring a reduction of Type I errors over Type II errors where relevant. This can be translated into an *objective function* F_O denoted in equation 5.3 that given a distance metric D returns the optimal threshold T based on the previous

characterization.

$$F_O(D) = \begin{cases} \sim Max(CAR) \\ TI_{er} < TII_{er} \end{cases} \tag{5.3}$$

Given equation 5.3 we test each distance metric on a large number of different thresholds to find the optimal threshold per metric. This optimal threshold value is calculated from the training dataset and is fed to the classifier system. Each distance metric calculates their own optimal threshold which is carefully described in the upcoming sections.

5.3.2 Weights

Each of the PBD feature vectors currently have an equal significance distribution among their vector elements. As pointed out by [12] some website characteristics are more relevant than others and this should be reflected in the metric calculations. By introducing an n-dimensional weight vector $\bar{W}$ we significantly increase the accuracy of the classification metrics. Given a weight vector $\bar{W} = [w_0, w_1, ..., w_n]$ we scale a PBD feature vector $\bar{A}$ by the weight vector $\bar{W}$ and normalize the weighted results by dividing the results by the largest weight element $Max(\bar{W})$. This weight scaling process to get a new vector $\bar{A}'$ is illustrated in equation 5.4.

$$\bar{A}' = \frac{\bar{A} * \bar{W}}{Max(\bar{W})} = [(a_0 * w_0)/Max(\bar{W}), ..., (a_n * w_n)/Max(\bar{W})] \tag{5.4}$$

Given that these weights should favor stronger characteristics over weaker characteristics it would significantly boost the classification accuracy. However, we need an initial set of weights that properly portray the Booter feature characteristics in order for this to work. To find the ideal set of weights we make use of what is called the *odds ratio* [74] [12] where we calculate the odds of a characteristic occurring in the positive class compared to the odds of the characteristic occurring in the negative class. For instance, take a sample set of 100 PBDs with 40 Booters and 60 Non-Booters and the DDoS Protection Service (DPS) characteristic. Among the Booters we find 35 Booters being protected by a DPS with only 12 Non-Booters being protected by a DPS. The odds of a Booter being protected by a DPS is thus 35 to its remainder of 5 which becomes 7:1, with the odds of a Non-Booter being protected being 12 to its remainder of 48 which then becomes 0.25:1. This gives us an odds ratio of 7/0.25 or 28 for the DPS characteristic.

Given the training dataset obtained in section 4.3 we calculate the odds ratio for each of the 15 characteristics shown in Table 8. As calculating the odds ratio for each characteristic requires a binary partition, we converted all decimal-valued scores $>= 0.5$ to indicate the score being that of a Booter.

Characteristic	Odds ratio	Weight
1. Number of pages	40.97	1.00
2. URL type	6.00	0.15
3. Average depth level	5.03	0.12
4. Average URL length	7.00	0.17
5. Domain age	22.19	0.54
6. Domain res. duration	5.77	0.14
7. WHOIS private	5.98	0.15
8. DPS	9.07	0.22
9. Page rank	20.93	0.51
10. Average content size	12.26	0.30
11. Outbound hyperlinks	22.83	0.56
12. Category-specific dict.	3.00	0.07
13. Resolver indication	1.39	0.03
14. Terms of services page	1.13	0.03
15. Login-form depth level	2.92	0.07

Table 8: Booter odds ratio and weights

From the odds ratios it becomes apparent which Booter website characteristics are more relevant than others and vica versa as respectively highlighted in Table 8. By scaling the odds ratios by the maximum odds ratio occurrence to get the final weight vector $\bar{W}$ we effectively transform Equation 5.4 to equation 5.5. This also effectively diminishes the effect of Booter characteristics like *Resolver indication* that were deemed ineffective for differentiating between Booter websites and non-Booter websites.

$$\bar{A}' = \bar{A} * \bar{W} = [\bar{A}_0 * \bar{W}_0, \bar{A}_1 * \bar{W}_1, ..., \bar{A}_n * \bar{W}_n] \tag{5.5}$$

Introducing a weight vector $\bar{W}$ into the classification calculations should significantly boost the accuracy and precision of the classifier system. In the upcoming sections we describe both the weighted and unweighted process of each classification metric.

5.3.3 Euclidean distance

Euclidean distance is a distance metric based on geometrical properties and one of the most often used metrics in data classification literature [65]. Euclidean distance defines the distance between two points in Cartesian space as characterized by their geometrical interpretation. The distance D between two vectors $\bar{A}$ and $\bar{B}$ is equal to the length of the difference vector between $\bar{A}$ and $\bar{B}$ [73] as expressed in equation 5.6.

$$D(\bar{A}, \bar{B}) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 ... (a_n - b_n)^2} = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{5.6}$$

For standardization purposes we prefer normalizing all score results, transforming each resulting distance score to the unit interval [0,1]. To normalize the Euclidean distance scores we divide the total score by the maximum possible

distance. We can obtain the maximum possible Euclidean distance by calculating the maximum point-wise distance in each dimension as shown in equation 5.7 given the perfect Booter vector $\bar{B}_p$.

$$D_{max} = \sqrt{\sum_{i=1}^{n} (\bar{B}_{p_i} - 0.0)^2} \tag{5.7}$$

Important to note is that certain Booter website characteristics do not consistently return relevant results. For instance, in the case of WHOIS data like the *WHOIS private* and *Domain age* characteristic the WHOIS server is not always available. In the case of incomplete data, we omit these inapplicable results from the feature vector ensuring the feature vector's completeness. This effectively fluctuates the n parameter, but keeps D_{max} applicable for score normalization.

As mentioned in section 5.3.1 each distance metric requires an optimal threshold for its binary decision. Given the Euclidean distance metric and the training dataset obtained from section 4.3 we generate the graph in Figure 7 given a total of 100 varying threshold parameters. For each individual threshold value we calculate the Classification Accuracy Rate (CAR) and the Type I and Type II error accuracy rates.

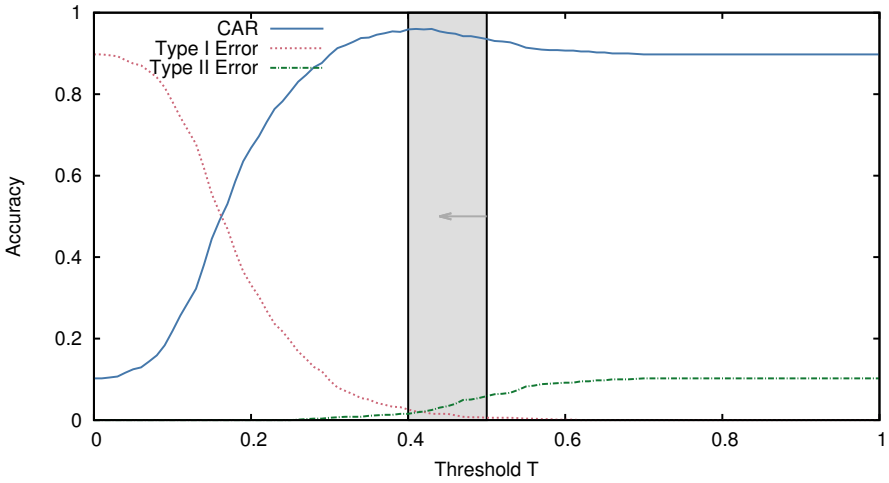


Figure 7: Euclidean distance accuracy rates on varying thresholds.

As discussed in section 5.3.1 an optimal threshold should be selected from Figure 7. We favor a reduction of Type I errors over Type II errors which we marked gray as the threshold area of interest. Given the objective function F_O from equation 5.3 we select an optimal threshold value T of 0.41.

Classification accuracy

Testing the accuracy of the classifier system involves measuring each of the 4 classification possibilities as derived from the confusion matrix discussed in section 5.1. For each classification metric we obtain the true positives T_p, the true negatives T_n, the false positives F_p and the false negatives F_n. With these 4

classification possibilities we generate the total classification accuracy and global error rates of the Euclidean distance metric.

Given the three test datasets described in section 5.1 featuring a total of 465 PBD feature vectors and the optimal threshold value T of 0.41 we obtain the accuracy results presented in Table 9. The results are the averaged results of individual accuracy rates of each test dataset.

Metric	CAR	TI_{er}	TII_{er}
Euclidean distance	0.920	0.030	0.050

Table 9: Euclidean distance classification accuracy

The Euclidean distance metric returns a higher accuracy rate than originally expected, mainly because of its simplicity and weaknesses in high dimensional datasets. As described in section 5.3.2 we can still improve the distance metric by introducing a weight vector $\bar{W}$.

Weights

By scaling both the input feature vector and the perfect Booter vector by the weight vector $\bar{W}$ we update the Euclidean distance metric with the weighted approach. Due to the weight vector $\bar{W}$ shifting the distance values and the normalization procedure we again have to measure the optimal threshold. In Figure 8 a total of 100 different threshold values were tested against the training dataset using weighted Euclidean distance.

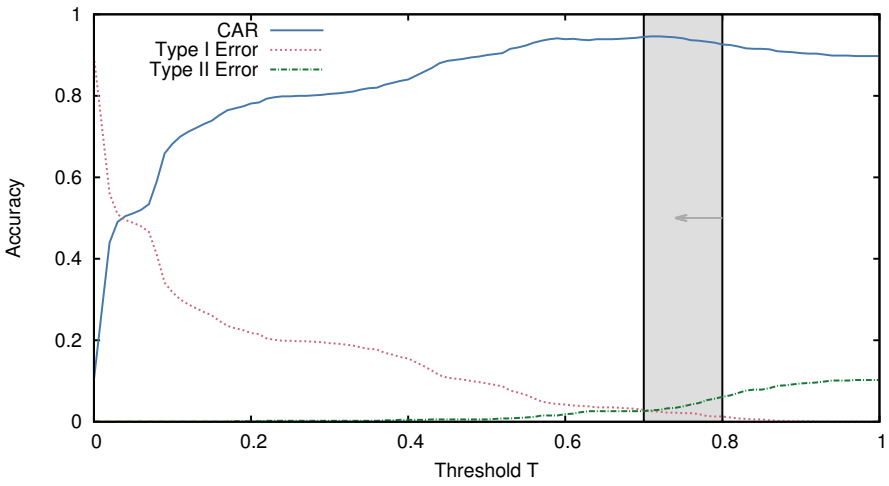


Figure 8: Weighted Euclidean distance accuracy rates on varying thresholds.

Given Figure 8 and the objective function F_O we find the optimal threshold value T of 0.71 that together with the test dataset give us the accuracy rates of Table 10.

Metric	CAR	TI_{er}	TII_{er}
Weighted Euclidean distance	0.933	0.017	0.049

Table 10: Euclidean distance classification accuracy

The weighted adaptation of the Euclidean distance metric performs marginally more accurate compared to its unweighted sibling. Specifically interesting to note here is the significant reduction of the Type I error rate which is highly favorable in the context of a Booter (black)list.

5.3.4 Squared Euclidean distance

Squared Euclidean distance is characterized by squaring the Euclidean distance equation 5.6, effectively removing the square root. The squared Euclidean distance is defined as D_s between vectors $\bar{A}$ and $\bar{B}$ [75] expressed in equation 5.8.

$$D_s(\bar{A}, \bar{B}) = \left(\sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \right)^2 = \sum_{i=1}^{n} (a_i - b_i)^2 \tag{5.8}$$

The squared Euclidean distance places progressively greater weight on features that are further apart. For normalization purposes we also normalize the results of the squared Euclidean distance equation. To normalize the results of equation 5.8 we divide its results by the maximum obtainable distance value as similarly done in equation 5.7. Equation 5.9 denotes the maximum squared Euclidean distance value given the perfect Booter vector $\bar{B_p}$.

$$D_{max} = \sum_{i=1}^{n} (\bar{B_{p_i}} - 0.0)^2 \tag{5.9}$$

This is effectively the same equation as equation 5.7 with the omission of the square root.

Classification accuracy

The Squared Euclidean distance metric is also a geometric classification metric and thus requires a threshold for its binary decision. In Figure 9 a total of 100 different thresholds T were measured for the squared Euclidean distance.

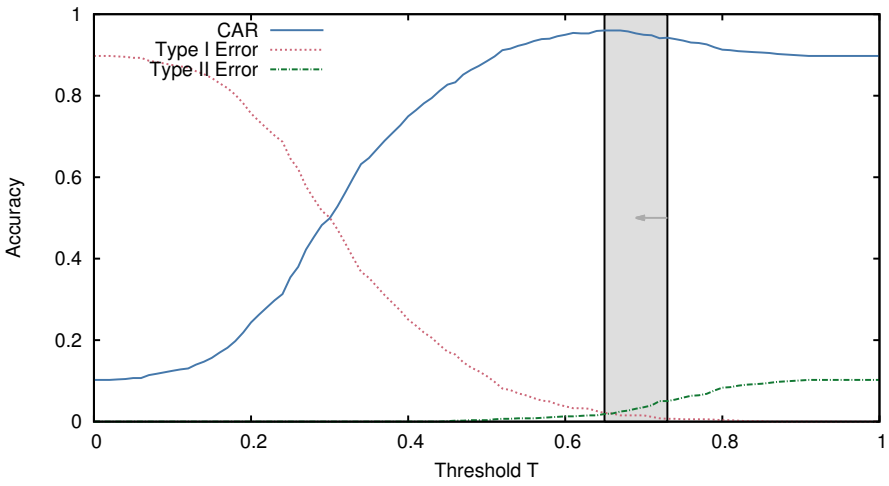


Figure 9: Squared Euclidean distance accuracy rates on varying thresholds.

Given Figure 9 and the objective function F_O we select an optimal threshold value T of 0.66. Selecting this threshold on the test dataset returns the accuracy results presented in Table 11.

Metric	CAR	TI_{er}	TII_{er}
Squared Euclidean distance	0.908	0.030	0.062

Table 11: Squared Euclidean distance classification accuracy

The results are slightly less accurate compared to the results of the Euclidean distance metric. As the squared Euclidean distance gives higher significance to vector elements further apart we can describe the lesser accuracy rates by Booter vectors and non-Booter vectors being mostly similar to some extent. Furthermore, as the TI_{er} remains similar and only the TII_{er} has reduced, the stronger distance focus only seems to affect the detection of Booter websites.

Weights

As proposed in section 5.3.3 we equivalently introduce a weight vector $\bar{W}$ into the squared Euclidean distance calculations. In Figure 10 we measured the accuracy results of the weighted squared Euclidean distance metric on the training dataset with varying threshold values.

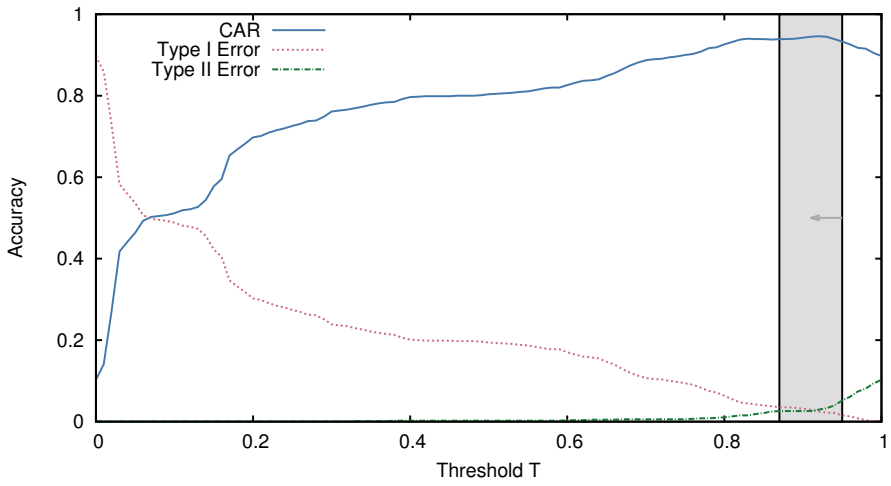


Figure 10: Weighted squared Euclidean distance accuracy rates on varying thresholds.

Given Figure 10 and the objective function F_O we find the optimal threshold value T of 0.89 that together with the test dataset give us the accuracy rates of Table 12.

Metric	CAR	TI_{er}	TII_{er}
Weighted squared Euclidean distance	0.940	0.022	0.039

Table 12: Weighted squared Euclidean distance classification accuracy

The weighted version of the squared Euclidean distance metric significantly boosts its accuracy rates. Similar to the Euclidean distance metric, the weighted adaptations seem to favor better Type I over Type II error accuracy rates.

5.3.5 Manhattan distance

The Manhattan distance defines the distance D_m between vectors $\bar{A}$ and $\bar{B}$ as the point-wise difference across dimensions [75].

$$D_m(\bar{A}, \bar{B}) = \sum_{i=1}^{n} |a_i - b_i| \tag{5.10}$$

As described by [65] the Manhattan distance metric is much more effective in higher dimensions (n larger than 3) compared to Euclidean distance. Normalizing the Manhattan distance results is done by dividing its results by D_{max} as calculated in equation 5.11 given the perfect Booter vector $\bar{B}_p$.

$$D_{max} = \sum_{i=1}^{n} (\bar{B}_{p_i} - 0.0) \tag{5.11}$$

Classification accuracy

In Figure 11 a total of 100 different thresholds were measured for the Manhattan distance metric.

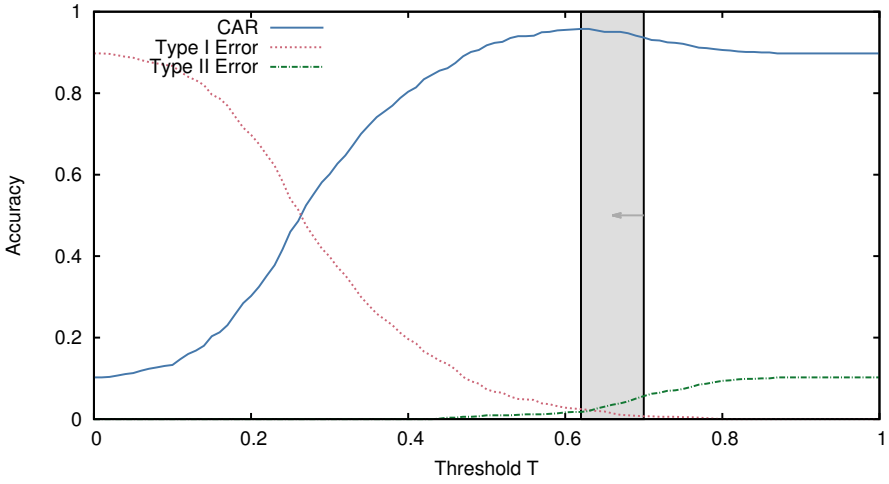


Figure 11: Manhattan distance accuracy rates on varying thresholds.

Given Figure 11 and the objective function F_O we select an optimal threshold value T of 0.62 that returns the accuracy results presented in Table 13.

Metric	CAR	TI_{er}	TII_{er}
Manhattan distance	0.927	0.024	0.049

Table 13: Manhattan distance classification accuracy

The Manhattan distance returns more accurate results compared to the default Euclidean distance (albeit marginally). This is in accordance with our original hypothesis that the Manhattan distance metric is more accurate in high dimensional datasets.

Weights

We equivalently introduce a weight vector $\bar{W}$ into the Manhattan distance calculations. In Figure 12 we measured the accuracy results of the weighted Manhattan distance metric with varying threshold values.

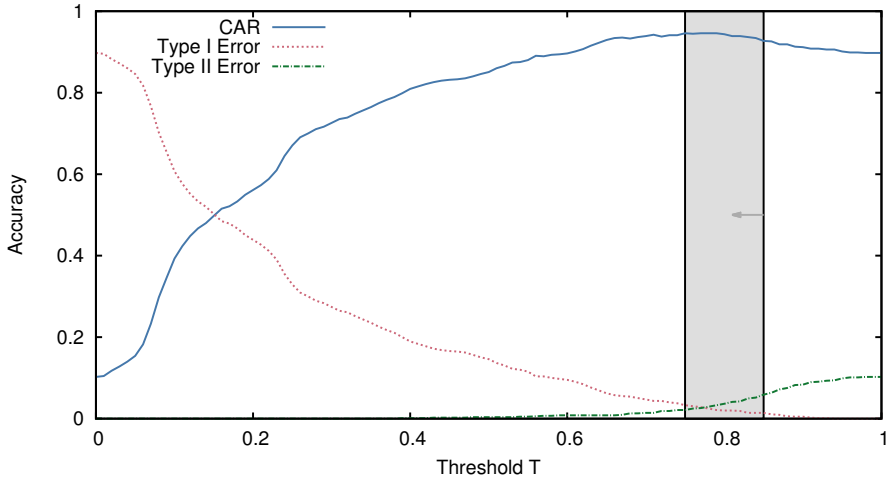


Figure 12: Weighted Manhattan distance accuracy rates on varying thresholds.

Given Figure 12 and the objective function F_O we find the optimal threshold value T of 0.75 that together with the test dataset give us the accuracy rates of Table 14.

Metric	CAR	TI_{er}	TII_{er}
Weighted Manhattan distance	0.931	0.019	0.049

Table 14: Weighted Manhattan distance classification accuracy

A moderate improvement, but similar to the squared Euclidean's weighted metric there is a significant improvement in the Type I error accuracy.

5.3.6 Fractional distance

The Fractional distance describes a distance metric that returns more accurate results compared to Euclidean and Manhattan distance when working specifically with high dimensional feature vectors. The fractional distance D_f [65] between feature vectors $\bar{A}$ and $\bar{B}$ is expressed in equation 5.12.

$$D_f(\bar{A}, \bar{B}, f) = \sum_{i=1}^{n} \left((a_i - b_i)^f\right)^{1/f} \tag{5.12}$$

Where f is some integer < 1. Note that when f equals 2 we obtain the Euclidean distance equation 5.6. The value of f is best chosen within the interval $f \in (0.25, 0.75)$, but a value of $f = 0.5$ will improve classification performance in nearly all circumstances [69].

To find the maximum distance for normalization of equation 5.12 we use equation 5.13 given the perfect Booter vector $\bar{B}_p$.

$$D_{max} = \left(\sum_{i=1}^{n} (\bar{B}_{p_i} - 0.0)^f\right)^{1/f} \tag{5.13}$$

Classification accuracy

In Figure 13 we measure a total of 100 different thresholds for the Fractional distance metric.

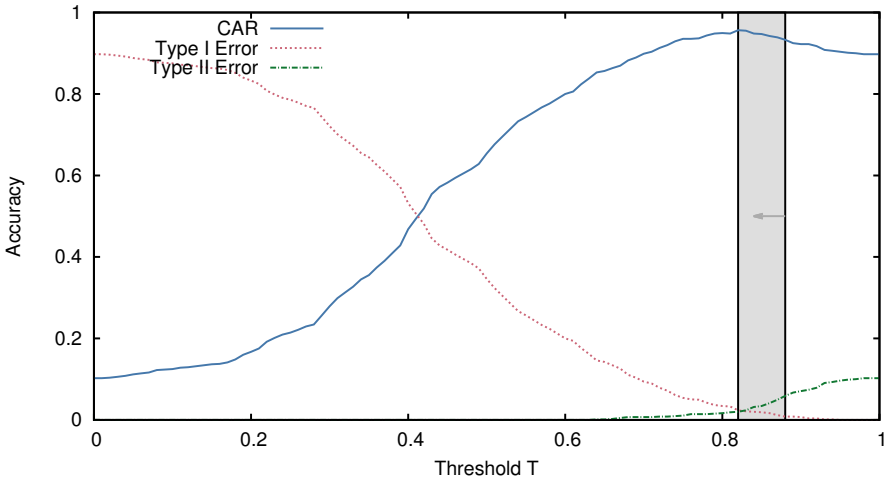


Figure 13: Fractional distance accuracy rates on varying thresholds.

Given Figure 13 and the objective function F_O we select an optimal threshold value T of 0.82 that returns the accuracy results presented in Table 15.

Metric	CAR	TI_{er}	TII_{er}
Fractional distance	0.914	0.024	0.062

Table 15: Fractional distance classification accuracy

Surprisingly, the Fractional distance metric performs relatively poor when it comes to the other distance metrics. Furthermore, the lower accuracy rates neglects our hypothesis of the Fractional distance metric measuring more accurate in high dimensional datasets.

Weights

As proposed in section 5.3.3 we equivalently introduce a weight vector $\bar{W}$ into the Fractional distance calculations. In Figure 14 we measured the accuracy rates of the weighted Fractional distance equation on varying threshold values.

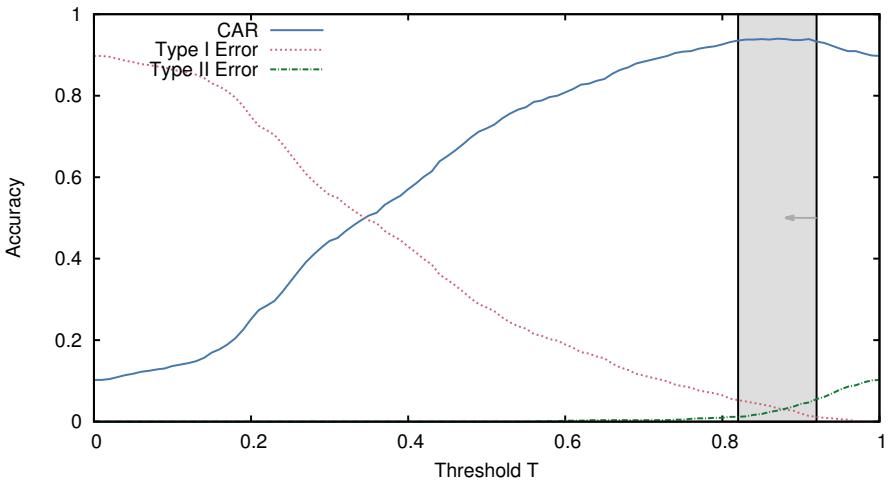


Figure 14: Weighted Fractional distance accuracy rates on varying thresholds.

Given Figure 14 and the objective function F_O we find the optimal threshold value T of 0.86 that together with the test dataset give us the accuracy rates of Table 16.

Metric	CAR	TI_{er}	TII_{er}
Weighted Fractional distance	0.923	0.022	0.056

Table 16: Weighted Fractional distance classification accuracy

The weighted adaptation returns a moderate improvement over the unweighted version, strongly favoring the Type I error rate over the Type II error rates. Even with the added weight vector $\bar{W}$ the fractional distance still performs relatively weak compared to the other distance metrics again neglecting the original hypothesis.

5.3.7 Cosine distance

The Cosine distance S_C defines the similarity as the scalar product between vectors $\bar{A}$ and $\bar{B}$ divided by their respective lengths as described by [12].

$$S_C = \cos(\theta) = \frac{\bar{A} \cdot \bar{B}}{||\bar{A}||||\bar{B}||} = \frac{\sum_{i=1}^{n} a_i \times b_i}{\sqrt{\sum_{i=1}^{n} a_i^2 \times \sum_{i=1}^{n} b_i^2}} \tag{5.14}$$

The Cosine distance metric can have denominator values equal to 0 at which point we replace the result of the equation with a similarity score of 0. Chromik (2015) [12] did an extensive comparison between the Euclidean distance metric and the Cosine distance metric and concluded that the Cosine distance metric returned better results with a Booter feature set. Since the Booter feature set of [12] is a subset of our Booter feature set the results are likely to be similar. As the Cosine distance metrics returns the cosine of the angle between two vectors in the range -1 to 1 there is no need to normalize the results. Negative Cosine distance scores get clamped to 0.0.

Classification accuracy

In Figure 15 we measure a total of 100 different thresholds for the Cosine distance metric on the training dataset.

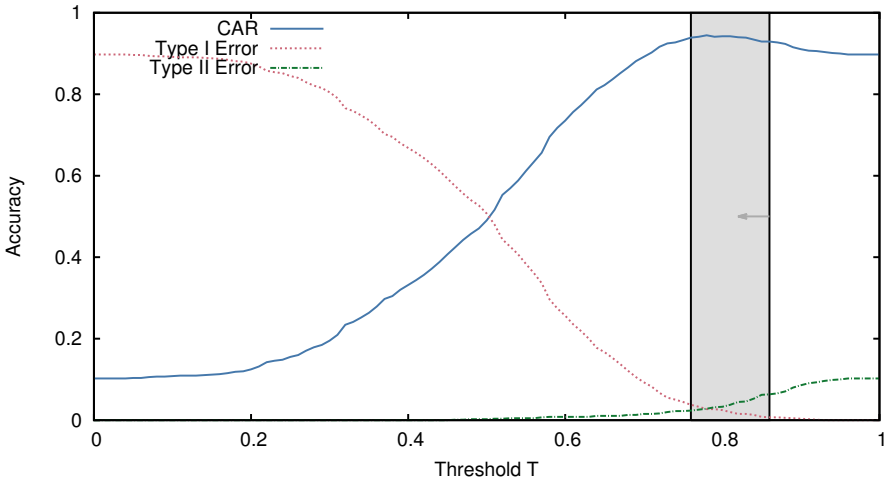


Figure 15: Cosine distance accuracy rates on varying thresholds.

Given Figure 15 and the objective function F_O we select an optimal threshold value T of 0.78 that returns the accuracy results presented in Table 17.

Metric	CAR	TI_{er}	TII_{er}
Cosine distance	0.914	0.049	0.037

Table 17: Cosine distance classification accuracy

The Cosine distance metric performs relatively inaccurate compared to the Euclidean distance metrics. This is a surprising result as earlier research [12] observed a significant improvement over using the Cosine distance metric over the Euclidean distance metric.

Weights

As proposed in section 5.3.3 we equivalently introduce a weight vector $\bar{W}$ into the Cosine distance calculations. In Figure 16 we measured the accuracy results of the weighted Cosine distance metric with varying threshold values.

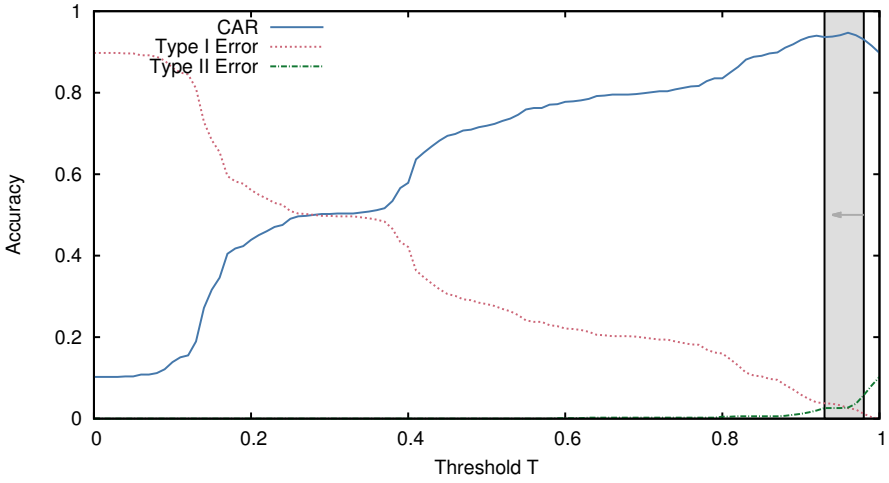


Figure 16: Weighted Cosine distance accuracy rates on varying thresholds.

Given Figure 16 and the objective function F_O we find the optimal threshold value T of 0.94 that together with the test dataset give us the accuracy rates of Table 18.

Metric	CAR	TI_{er}	TII_{er}
Weighted Cosine distance	0.944	0.022	0.034

Table 18: Weighted Cosine distance classification accuracy

Surprisingly enough adding weights to the Cosine distance metric significantly boosts its accuracy rates, even so much that the Cosine distance metric performs best compared to the other distance metrics on Booter feature vectors. This also confirms earlier measurements by Chromik (2015) [12] where the Cosine distance metric performed most optimal with the addition of weights.

5.4 Naive Bayes

With distance metrics the similarity between two feature vectors is modeled by their geometric or angular distance. In contrast, Naive Bayes classification makes use of probability to classify feature vectors [76]. Given a Booter characteristic n being the *evidence* whether a Booter has a small number of pages and the *outcome* $\bar{B}$ as the website being a Booter we can compute $P(B|n)$, the probability that B occurs given the *evidence* n [47]. Conditional probability is defined as the probability that something will happen, given that something else has already happened. This is characterized by equation 5.15 with O being some outcome and E being some evidence.

$$P(O \ \& \ E) = P(O) \times P(E|O) \tag{5.15}$$

This gives us a basic formula for the Bayes Rule.

$$P(O \ given \ E) = \frac{P(E|O) * P(O)}{P(E)} \tag{5.16}$$

Equation 5.16 only describes a single piece of evidence and for our research we have a large array of evidence in the form of a feature vector. The math for describing the Bayes Rule given multiple evidence is relatively complicated. To get around the complications we treat each piece of evidence as independent which is called (for this reason) the Naive Bayes approach.

$$P(O|(E_1, E_2, ..., E_n)) = \frac{P(E_1|O) \times P(E_2|O) \times ... \times P(E_n|O) \times P(O)}{P(E_1) \times P(E_2) \times ... \times P(E_n)} \quad (5.17)$$

Given equation 5.17 and the probabilities of each piece of evidence occurring we can calculate the probability of a specific outcome [77]. Calculating the probabilities of all possible outcomes returns a list of probabilities of which the highest probability is selected as the most plausible outcome. Then, from the Naive Bayes equation, assuming that Booter and non-Booter websites occur with equal probability (with $O = 1$ being a Booter), we compute the probability as a feature vector $\bar{A}$ belonging to a Booter as:

$$P(O = 1|\bar{A}) = \frac{P(\bar{A}|O = 1) \times P(O = 1)}{P(a_1) \times P(a_2) \times ... \times P(a_n)} \quad (5.18)$$

The Naive Bayes classification scheme allows us to pre-compute all probabilities given a training set and as a result Naive Bayes only comes down to basic arithmetic. This makes it an easy, quick and efficient classification scheme that is extensively used in text classification and spam filters [54] [70]. The scheme does require a representative training set to pre-compute all probabilities and as a result it is highly dependent on the completeness and accuracy of a training dataset.

Given the training dataset as discussed in section 4.3 we can compute the individual probabilities of Booter website characteristics occurring. The Naive Bayes classification metric assumes all individual characteristics to be binary: a characteristic is either true or false. As some of the Booter website characteristics are decimal values in the unit interval we first have to transform these to their binary equivalents before calculating the individual probabilities. This is accomplished by converting all decimal valued metrics of value higher or equal to 0.5 to 1.0 and similarly the other way around. For instance, if we take a normalized Number of pages feature score of 0.72 which is equal or higher than 0.5 we convert this score to 1.0. Given the total training dataset of binary characteristic values we can calculate all individual characteristic probabilities as resulted in Table 19.

Characteristic	X = Booter	X = Non-Booter
P(Number of pages / **X**)	0.97	0.23
P(URL type / **X**)	0.93	0.80
P(Average depth level / **X**)	0.94	0.67
P(Average URL length / **X**)	0.37	0.06
P(Domain age / **X**)	0.89	0.14
P(Domain res. duration / **X**)	0.89	0.62
P(WHOIS private / **X**)	0.38	0.28
P(DPS / **X**)	0.72	0.18
P(Page rank / **X**)	0.85	0.35
P(Average content size / **X**)	0.74	0.16
P(Outbound hyperlinks / **X**)	0.92	0.20
P(Category-specific dict. / **X**)	0.52	0.19
P(Resolver indication / **X**)	0.22	0.19
P(Terms of services page / **X**)	0.44	0.36
P(Login-form depth level / **X**)	0.82	0.66

Table 19: Probability of likelihood of each characteristic given outcome X.

Furthermore, as Booters and non-Booters are not evenly distributed in the training dataset we also calculate the probabilities of any random feature vector being either a Booter or a non-Booter website. These prior probabilities are calculated by dividing the samples of a given outcome by the total number of samples as shown in Table 20.

Outcome	Base rate
P(Booter)	0.1001
P(Non-Booter)	0.8999

Table 20: Prior probabilities or *base rates*.

Given the individual characteristic probabilities and the prior probabilities from Tables 19 and 20 respectively we calculate the total probabilities of any random PBD feature vector being either a Booter or a non-Booter. The result of both outcomes is compared and the outcome with the highest probability is selected as the most plausible outcome which depicts the classifier system's binary decision.

5.4.1 Classification accuracy

The Naive Bayes classification scheme has no further variables to collect and/or configure and its decision is completely binary; no threshold parameter is required. As a result, the accuracy of the Naive Bayes classification metric is completely dependent on the calculated probabilities based of the training dataset. Given the training dataset as obtained in section 4.3 we calculate the classification accuracy rate together with the error rates TI_{er} and TII_{er} as shown in Table 21.

Metric	CAR	TI_{er}	TII_{er}
Naive Bayes classification	0.912	0.056	0.032

Table 21: Naive Bayes classification accuracy

Surprisingly enough, the Naive Bayes classification metric performs relatively poor compared to accuracy measurements in similar literature. As the current Naive Bayes classification metric uses an equal significance distribution we improve the Naive Bayes classification accuracy by altering the distribution of the feature vectors by introducing weights into the equation

5.4.2 Weights

We introduce weight vector $\bar{W}$ in the Naive Bayes classification to improve the accuracy rates of the classification metric. Introducing weight vector $\bar{W}$ into the Naive Bayes equation returns the accuracy results as shown in Table 22.

Metric	CAR	TI_{er}	TII_{er}
Weighted Naive Bayes classification	0.918	0.049	0.032

Table 22: Weighted Naive Bayes classification accuracy

The introduction of weights only marginally improved the accuracy scores of the Naive Bayes classification metric and can hardly be measured as a significant improvement. A relative poor score of the Naive Bayes classification metric can be accounted to several practicalities. First, the metric's performance is closely related to the representative quality of the training set. Even though the training set performed relatively accurate with regards to the other metrics, perhaps its probability calculations were not suited for Booter measurements. Second, the Naive Bayes classification is primarily intended for probabilities of binary partitions instead of unit intermediate values e.g. the Number of pages characteristic not being simply 1 or 0. Because the training dataset did not consist of binary partitions only, a conversion process involved converting unit-valued characteristics to their binary equivalents. As the training dataset did not entirely consist of binary partitions this could negatively influence the accuracy scores of the Naive Bayes classification metric.

5.5 K-Nearest neighbors

K-Nearest Neighbors is a classification metric used for classification or regression and relatively simple to understand. The general idea behind k-NN is that given a plot of n-dimensional space, denote each feature of the feature vector on a single axis in n-dimensional space [78] [57]. For instance, a 2 dimensional feature vector will have its first feature denoted along the x-axis and its second feature denoted along the y-axis. The feature vector is then compared to a trained set of 2 dimensional points where each point is either classified as a Booter or a non-Booter website. Figure 17 shows a 2 dimensional graph characterized by two features on the x and y axis respectively with the blue squares representing non-Booter website feature vectors and the red circles representing the Booter

feature vectors. The green triangle denotes the feature vector to be classified with a generated feature score on both axes.

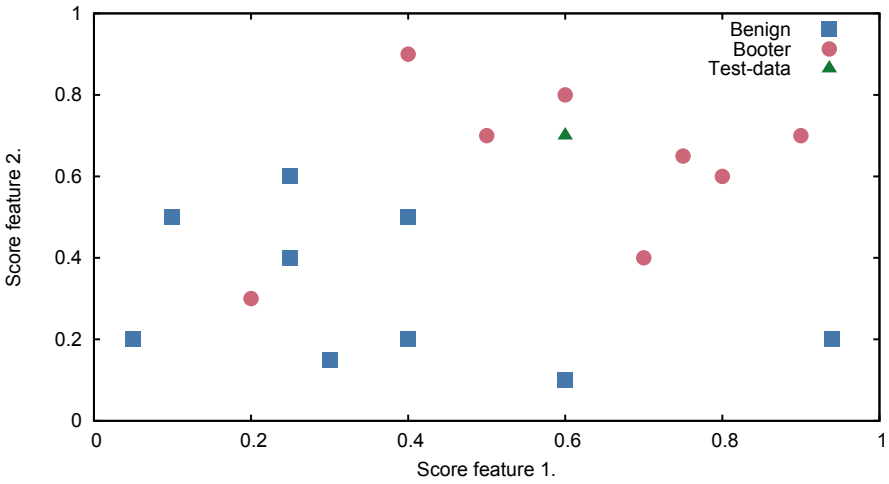


Figure 17: k-NN on 2 dimensional features.

The notion behind k-NN is that from a geometrical and visual perspective it is often quite apparent as to which class the feature vector belongs to. For instance, in Figure 17 it should come as no surprise that the feature vector depicted by the green triangle belongs to the Booter class as it is closest to those trained sets of points. The classification scheme finds the k closest neighbors between the green triangle and all nearby feature points and classifies the green triangle by the group in its more immediate vicinity.
Formally, k-NN calculates the Euclidean distance[8] between n features in an n-dimensional space and retrieves the k nearest neighbors for classification [79]. When k equals 1 we take the closest neighbor, this divides the space into an n-dimensional set of cells. With k $>$ 1 we sample multiple points with the likelihood L of the point P belonging to Booters B expressed as in equation 5.19.

$$L_B(P) = \frac{k\_red\_points\_in\_vicinity}{\#red\_points} > \frac{k\_blue\_points\_in\_vicinity}{\#blue\_points} \tag{5.19}$$

Care should be taken in specifying k as a larger k gives better balance (outliers become less significant), but makes boundaries between different classes less distinct.

5.5.1 Classification accuracy

The accuracy of the k-NN classification metric is tested against the test dataset. However, the k-NN classification metric can use any arbitrary distance equation for calculating its closest neighbors. It is not yet sure which distance metric

[8]Or any other distance metric

performs best on a Booter dataset so we first identify which of the earlier introduced distance metrics returns the most accurate results on the training dataset. In Figure 19 the classification accuracy rate of all 5 distance metric is measured with varying K between 1 and 15.

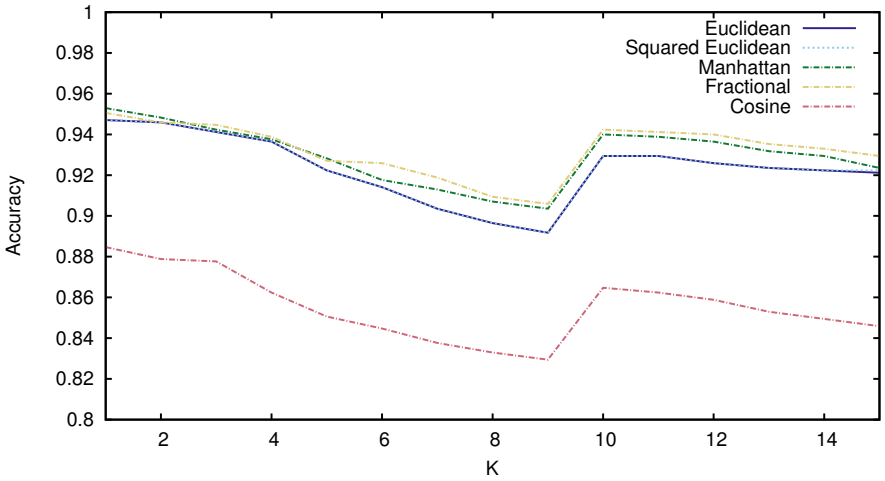


Figure 18: K-NN CAR on multiple distance metrics

Regardless of the distance metric in use, the k-NN accuracy scores follow a general trend. At small values of k we find high accuracy rates that gradually decrease over k. At around a k of 10 we find accuracy rates close to their original peak. Furthermore, all distance metric with the exception of the Cosine distance metric perform relatively the same. The inaccuracies from the Cosine distance metric are likely due to distance sorting which does not seem effective on angular differences. Based on Figure 18 we select the Fractional distance metric as the best performing metric in k-NN. Selecting k requires an extra comment as the ideal k value based on the classification accuracy rates is 1. A small k makes the boundary between the two classes more distinct, but reduces the general balance. As a higher balance reduces the significance of outliers (which we deem to have) we preferably select a higher value of k. Based on Figure 18 we select a k of 10 that returns similar high classification accuracy rates while providing a strong balance; in the accuracy analysis we test both probable optimal versions of k.

Given the fractional distance metric and a k parameter of both 1 and 10 we find the k-NN accuracy rates in Table 23 as applied on the test dataset.

Metric	k	CAR	TI_{er}	TII_{er}
K-Nearest Neighbors	1	0.869	0.039	0.092
K-Nearest Neighbors	10	0.910	0.073	0.017

Table 23: K-Nearest neighbors classification accuracy

As originally expected the lower k value scores significantly worse than the higher k value. However, the classification accuracy rates are considerably less

accurate than any of the other classification metrics. By introducing weights we aim to improve the classification accuracy rates of the k-NN classification metric.

5.5.2 Weights

In accordance with the previous sections we introduce a weight vector $\bar{W}$ into the k-NN calculations. As k-NN makes extensive use of distance metrics to calculate the relative neighbor scores it makes sense to embed the weight vector into the distance metric used. As the distance metric in use was earlier introduced with weights we refer to sections 5.3.3 to 5.3.7 for implementational details. Similarly, we measure the effectiveness of each weighted distance metric on the k-NN accuracy scores. In Figure 19 we measure the same five distance metric with incorporated weights on varying k.

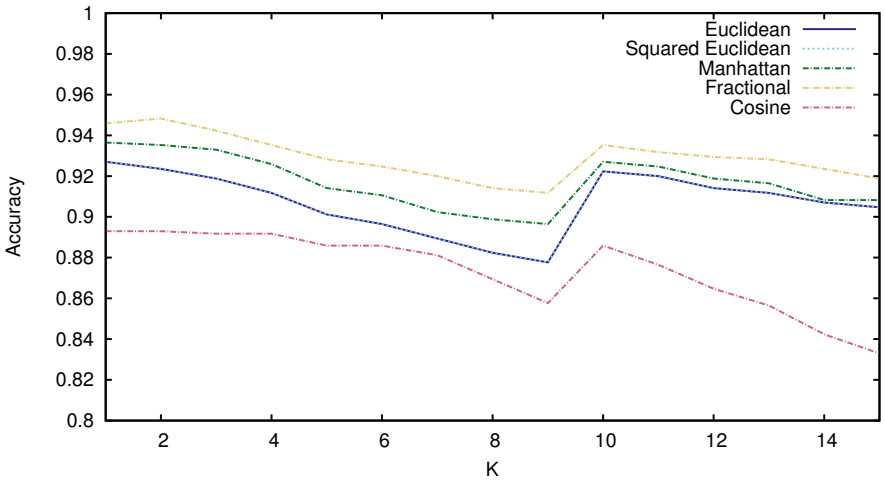


Figure 19: K-NN CAR on multiple weighted distance metrics

Introducing the weight vector $\bar{W}$ generates largely the same result and we again pick the Fractional distance metric as the more optimal distance metric. Similarly, we select a small k of 2 and an optimal k of 10 which are both tested against the test dataset. Running the k-NN algorithm with weighted Fractional distance on the test dataset returns the accuracy results as shown in Table 17.

Metric	k	CAR	TI_{er}	TII_{er}
Weighted K-Nearest neighbors	2	0.925	0.037	0.039
Weighted K-Nearest neighbors	10	0.914	0.060	0.026

Table 24: Weighted K-Nearest neighbors classification accuracy

The classification accuracy rates of the weighted k-NN implementation have marginally improved, but are still insufficient compared to the other classification metrics. The unexpected inaccuracy could perhaps be influenced by the unequal distribution of Booters among the dataset, which might negatively affect the k-NN calculations. Moreover, Figure 18 and Figure 19 both follow an

interesting pattern which is likely an artifact specifically accounted to the training dataset. Basing the value of k on this specific training set could negatively influence the accuracy rates. However, the training dataset is currently the best representation of a Booter dataset available and thus serves best at obtaining the mentioned k parameters. Furthermore, the k-NN classification metric generally has a higher Type I error rate compared to the other metrics which makes it less favorable on a Booter dataset.

We summarize the accuracy results of each of the classification metrics in Table 25 including their Type I TI_{er} and Type II TII_{er} error rates.

	Unweighted			Weighted		
Metric	CAR	TI_{er}	TII_{er}	CAR	TI_{er}	TII_{er}
Euclidean distance	0.920	0.030	0.049	0.933	0.017	0.049
Squared Euclidean distance	0.908	0.030	0.062	0.940	0.022	0.039
Manhattan distance	0.927	0.024	0.049	0.931	0.019	0.049
Fractional distance	0.914	0.024	0.062	0.923	0.022	0.056
Cosine distance	0.914	0.049	0.037	0.944	0.022	0.034
Naive Bayes	0.912	0.056	0.032	0.918	0.049	0.032
K-Nearest Neighbors	0.910	0.073	0.017	0.914	0.060	0.026

Table 25: Grouped accuracy results of the Booter classifier system

To our surprise the distance metrics perform significantly better than the Naive Bayes classification and the k-NN classification metric. As the Naive Bayes classification metric and the k-NN metric performed relatively accurate in similar literature we would have expected them to outperform the distance metrics. The obvious accuracy differences can be accounted to both techniques not being particularly suited to a Booter dataset with its previously mentioned technicalities. Furthermore, the distance metrics require an additional threshold variable which the Naive Bayes classification and the k-NN metrics lack. The threshold value allows for a higher degree of freedom and potentially plays a large role in the higher classification accuracy rates of the distance metrics. From the results we denote the weighted Cosine distance metric highlighted in green as the best metric for classification on a Booter dataset. Additionally, the Euclidean distance metric highlighted in yellow is also a strong metric for Booter classification due to its low Type I error rate. However, the difference in CAR between the Cosine and Euclidean distance metric is too large to favor the Euclidean distance metric. It should be noted that even though some perform less favorable than others, the accuracy rates of all classification metrics are plausible and a significant improvement over previous Booter literature. We are however not yet satisfied with the current accuracy rates of the classification metrics. Using weights give the metrics a sense of adaptability that allow us to vary the individual weight elements to boost the relevance of certain characteristics. By capturing this property of weights we can build a system to find the optimal set of weights on the training dataset which is part of our focus in the next section.

This page intentionally left blank.

6 Adjustments and considerations

In this chapter we discuss how to adjust the weight vectors to improve the system's accuracy. We discuss a supervised machine learning algorithm and determine whether such a machine learning approach is a viable solution for increasing the system's accuracy. Furthermore, we analyze the results of the classification system and discuss the origin of its errors.

In the previous chapter we introduced weight vectors into the classification metric calculations which improved their classification accuracy rates. By varying the individual weight elements we could find a weight vector $\bar{W}'$ that better represents a Booter dataset. We hypothesize that this improves the system's accuracy on Booter feature vectors; this approach also gives us a considerate amount of flexibility. By alternating the relative contributions of each individual weight we can tweak the algorithm such that different characteristic combinations may contribute differently to Booter classification. The default weights as determined in section 5.3.3 were derived from the odds ratio obtained from the training dataset. While this ideally represents a strong set of weights for a Booter dataset, it is probable a more optimal set of weights exist. To find the *optimal weight vector* $\bar{W}'$ we can manually tune the weight elements, but a better and more long-term solution would be to develop a supervised machine learning algorithm.

6.1 Weight adjustments

Given a weight vector $\bar{W}$ and a testing set T, define a system M that varies the weight elements to generate a new set of weights $\bar{W}'$. The new set of weights is then tested against the training set and given an error score $S(\bar{W}) = s$, favoring the weight distributions producing smaller error scores. The system M runs a continuous process on the testing set T and returns at any given time a set of weights that best describe the set of Booters i.e. the weights that return the smallest error score s.

6.1.1 Supervised machine learning

The system M is best described by pseudo code as demonstrated in Algorithm 1.

```
1  count ← 0;
2  while True do
3  |  W̄, s_err ← select_db()  /* returns current best weight/error pair */ ;
4  |  W̄′ ← ∅;
5  |  for i ← 0 to len(W̄) do
6  |  |  W̄′[i] ← W̄[i] ∗ rand_gauss();
7  |  end
8  |  count ← count + 1;
9  |  if count % 5 == 0 then
10 |  |  index ← rand_int(0, len(W̄));
11 |  |  W̄′[index] ← W̄′[index] ∗ rand_float(1.0, V_e);
12 |  end
13 |  s_err_new ← error_function(W̄′);
14 |  store_in_db(s_err_new, W̄′);
15 end
```

Algorithm 1: Weight adaptability learning

At line 3, Algorithm 1 queries the current optimal weight vector $\bar{W}$ with the lowest error rate s_err as found in the database. Given weight vector $\bar{W}$ the algorithm varies each individual weight element (between lines 5 to 7) using a random multiplier returned by $rand\_gauss$. We sample the random multipliers from a bell-curved Gaussian (normal) distribution that describes a curve with most of its values close to a peak defined by A. By randomly sampling the Gaussian distribution with an A of 1.0 we ensure the weight multipliers remain relatively close to 1.0 as to not distort the original weight vector too much. The Gaussian function that describes a bell-curve shape is shown in Equation 6.1 [80].

$$y = Ae^{-(x-\mu)^2/2\sigma^2} \tag{6.1}$$

The parameter A defines the height of the curve's peak, μ describes the center or mean of the peak and σ (the standard deviation) controls the width of the curve. As we want to keep most weights close to 1.0 and vary them slightly each run we set A to 1.0 and select a relatively large σ to keep most values close to A. Selecting a σ and μ of 0.5 each we selectively describes a cropped curve that we sample within the unit interval. Equation 6.2 then centers the Gaussian distribution around a μ of 0.5 where a random x between 0.0 and 1.0 frequently returns a y close to 1.0.

$$y = e^{-(x-0.5)^2/(2*0.5^2)} \tag{6.2}$$

Equation 6.2 is plotted in Figure 20.

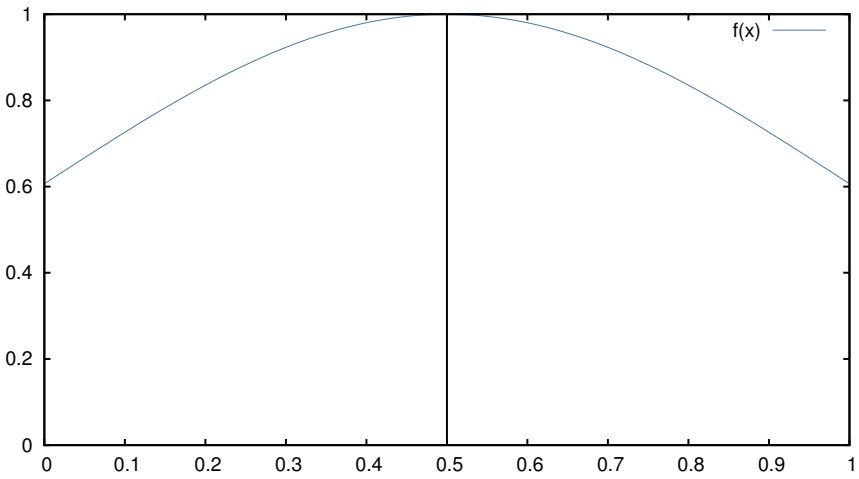


Figure 20: Gaussian distribution for weight adaptation

The function *rand_gauss* from Algorithm 1 thus selects a random x between 0.0 and 1.0 and samples the Gaussian distribution from Figure 20.

To steer the algorithm in completely different directions as to promote evolution we multiply a random element of weight vector $\bar{W}'$ by a large multiplier between 1.0 and V_e. We selected a positive value of 2.5 for V_e as this evenly balances out the average Gaussian weight reductions of any weight element over 5 iterations. As shown in Algorithm 1 (between lines 9 to 12) this operation repeats itself every 5 iterations. This prevents the algorithm from consistently heading in a single direction while trying to find other interesting combinations that could prove effective.

Once the temporary new weight vector $\bar{W}'$ is generated it is passed to the error function $error\_function$ (in line 13) that calculates the precision or global error rate G_{er} of the current set of weights. As multiple classification metrics were discussed in section 5.1 we would ideally have to tune the supervised machine learning system for each individual metric. This is a necessary step as each classification metric uses a different error function to determine its accuracy; something we cannot generalize for all classification metrics. For that reason we specifically focus the system on the strongest classification metric as determined at the end of section 5.1 which was the Cosine distance metric.

Given the Cosine distance metric, the error function consists of a total of three steps. First, the Cosine distance scores are re-calculated with the newly defined weights given the training dataset. Second, a total of 100 threshold parameters are checked against the training dataset to find the most suitable threshold T for the weight vector $\bar{W}'$; this is determined by the Classification Accuracy Rate (CAR). Last, given T the Cosine distance score is re-calculated on the test dataset(s) and the obtained global error rate G_{ger} is returned.

Finally, the new error rate and corresponding weight vector $\bar{W}'$ are stored in the database (at line 14) and the algorithm repeats itself until manually terminated. In the next section we discuss the results and effects the algorithm had on the classifier system.

6.1.2 Results

The machine learning system from Algorithm 1 was periodically run between the 7th of September to the 24th of September producing and testing a total of 824 different weight vectors. The updated weight vector list was tested against the objective function F_O defined in Equation 5.3 to find the optimal weight vector $\bar{W}$ that has both a high Classification Accuracy Rate (CAR) and preferably a low Type I error rate TI_{er}. In Table 26 the most optimal (normalized) weight vector $\bar{W}'$ returned from Algorithm 1 is shown.

Characteristic	$\bar{W}$	$\bar{W}'$
1. Number of pages	1.00	1.00
2. URL type	0.15	0.16
3. Average depth level	0.12	0.13
4. Average URL length	0.17	0.17
5. Domain age	0.54	0.30
6. Domain res. duration	0.14	0.18
7. WHOIS private	0.15	0.19
8. DPS	0.22	0.32
9. Page rank	0.51	0.47
10. Average content size	0.30	0.21
11. Outbound hyperlinks	0.56	0.40
12. Category-specific dictionary	0.07	0.10
13. Resolver indication	0.03	0.04
14. Terms of services page	0.03	0.03
15. Login-form depth level	0.07	0.04

Table 26: Updated weight vector $\bar{W}'$

When normalized there are slight variations noticeable, but no significant distortions in the original weight distribution. This shows that the original weight vector was a relative good match, but not yet optimized for Booter classification. Given the new weight vector $\bar{W}'$ and the Cosine distance metric we sample the training dataset to find an optimal threshold T of 0.95 given the objective function F_O in equation 5.3. The resulting threshold T and weight vector $\bar{W}'$ are then tested against the test dataset as shown in Table 27.

	CAR	TI_{er}	TII_{er}
Weighted Cosine distance w/ $\bar{W}$	0.944	0.022	0.034
Weighted Cosine distance w/ $\bar{W}'$	0.955	0.015	0.030

Table 27: Updated weight vector $\bar{W}'$

Given the CAR and the error rates it is clear the machine learning algorithm determined a more effective set of weights for Booter classification. The new weight vector $\bar{W}'$ returned a CAR that outperformed the original CAR over a percentage point on the Cosine distance metric which we deem a significant improvement. Furthermore, it also managed to significantly boost the Type I error rate of the Cosine distance metric.

The observation we make here is that weight-based classification metrics and machine learning prove an interesting combination for significantly improving a Booter classification system's accuracy rates. As the approach shows definitive promise, future research on alternative machine learning algorithms like neural networks or linear regression may aid in Booter classification.

6.2 Considerations of our top results

The weighted Cosine distance metric significantly boosted the classification accuracy over previous literature. The classification system is however not able to classify any arbitrary website as a Booter or a non-Booter with 100% confidence. To determine the origin of the misclassification errors and determine their impact we manually analyze the individual error websites.

6.2.1 Type II error analysis

The Cosine distance metric and the classification accuracy rates of Table 27 returned a Type II error rate of 0.030. Given the total amount of 465 websites in the Booter test dataset this means that a total of 14 websites were incorrectly classified as non-Booter websites. For these websites the 15 Booter website characteristics as discussed in chapter 3 were insufficient in classifying them as Booter websites. By manually analyzing each individual website the origin of the errors are sufficiently clear. The basic premise of the Booter classification research was that Booter websites follow a general pattern of organization and presentation which most of the 14 Booter websites did not follow. Two of the 14 Booter websites however did follow a general pattern common to Booters that were still not correctly classified. This can largely be accounted to these two Booters performing relatively inaccurate on the 15 website feature characteristics. For instance, both of them are relatively popular and longstanding and thus score poor on the page rank, domain age and domain reservation duration. Also, one of the Booters contained the number of pages outlier of 71 which due to its weighted importance played a significant role in its incorrect classification.

From the Type II error analysis it is clear not all Booter websites can be correctly classified by the system, largely due to outliers or the lack of a general pattern. Further improving the Type II error accuracy would require expanding or tuning the Booter website characteristics and testing different normalization range intervals to accommodate the outliers without sacrificing the Type I error accuracy rate.

6.2.2 Type I error analysis

Most relevant to a Booter (black)list system are the Type I error rates i.e. the number of times a non-Booter website is classified as a Booter website. Given the Cosine distance metric and the classification accuracy rates of Table 27 we find a Type I error rate of 0.015 which corresponds to a total of 7 non-Booter websites out of 465 websites. Given these 7 Booter websites we find that all of them are not Booter websites themselves, but indirectly associated with Booter websites making them by definition morally questionable. The listed websites sell Booter attack scripts, tools and scripts for creating your own Booter or are hosts for GUI Booters which is a term to coin Booter desktop applications. This is a

significant observation when it comes to the applicability of a Booter (black)list as false positives are highly regarded as a strong objection against blacklists as discussed in section 2.4. Given that the number of false positives are all websites that are likely to be illegitimate a blacklist approach would remain effective as the blacklist has the added side-effects of blocking questionable Booter related websites as well.

From the Type I error analysis we can conclude that the classification system does not block legitimate non-Booter websites. The blocked non-Booter websites are indirectly related to Booters themselves, which we ethically deem as acceptable errors.

From the Type II error observation we observed two Booter websites not being included in the (black)list as they performed poorly on relatively important Booter website characteristics. We deemed these as insignificant errors as the errors were the cause of noticeable outliers. However, one can derive a possible mitigation tactic from Booter owners to specifically focus on generating website characteristics outside the defined ranges. These mitigations could in the future compromise the described methodology. Nonetheless, we described for most of the Booter website characteristics in chapter 3 that mitigating these characteristics will be difficult or unlikely. Furthermore, given the adaptable properties of weight vectors and the machine learning system M we can adapt to these changing circumstances.

The previous Type I and Type II observations open the discussion of a final note of the Booter classification accuracy. If the true purpose of the Booter (black)list is to find Booters with the highest possible accuracy, the Type II error rate is irrelevant to some regard. A higher Type II error rate would result in less Booters getting detected, but a sufficiently low Type I error rate renders the (black)list effective to its purpose; this in according to the objective function F_O. Given these considerations we can effectively disregard the relatively small amount of Type II errors and denote the accuracy rate in the context of a Booter (black)list to **98.50%**. Furthermore, the inaccuracies of the accuracy rate are accounted to morally questionable websites indirectly associated with Booters. Given previous research and similar literature this is a clear contribution to Booter classification.

Part III
Conclusion

This page intentionally left blank.

7 Concluding remarks

This thesis served as describing the complete process of generating a Booter (black)list in an automatic fashion. First, we extensively analyzed Booters in chapter 2 to understand the Booters as a phenomenon. From the analysis we determined the proposed methodology and retrieved a list of 15 Booter website characteristics. Next, we developed a web crawler in chapter 4 to crawl the web for an up-to-date list of Potential Booter Domain names. Using the 15 Booter website characteristics from chapter 3 we further scraped all individual websites to generate both a (normalized) test and training dataset for further classification as described in section 4.2 to 4.3. Next a classifier was developed in chapter 5 to filter the list of Potential Booter Domain names to a list of only Booter domain names. Given the web crawler and the classifier we did an extensive classification study in sections 5.2 to 5.5 on multiple classification metrics to determine the most accurate metric for Booter classification. Most of the classification metrics showed promise, but mainly due to a threshold boundary the distance metrics and specifically the Cosine distance metric performed most accurate. Additionally, to further improve the accuracy rates of the classification metrics we used the adaptability property of the weight vectors introduced in the classification metrics. This property allowed us to develop a supervised machine learning algorithm described in section 6.1 that aims to find a new weight vector more suited to Booter classification. Together with weights and the machine learning algorithm the Classification Accuracy Rate of the system reached 95.49% with an accuracy rate in the context of a Booter (black)list of 98.50%. We believe the resulting accuracy rates to be a significant improvement over previous research and an important contribution to Booter classification. Furthermore, the final analysis on the classification error rates in section 6.2 concluded all Type I errors to be indirectly related to Booters making them morally questionable. These observations together with the final blacklist accuracy rate of 98.50% give us the confidence to conclude that the system generates Booter (black)lists that are accurate enough for practical considerations.

7.1 Contribution

Overall, the research provided the following contributions:

- A Booter (black)list that is up-to-date and the most accurate form of a Booter (black)list at the time of this writing. The Booter (black)list is available per request at `http://booterblacklist.com`.
- A crawler and classifier system that generates up-to-date Booter (black)lists with practical accuracy rates.
- A listing of all the research's source code found at the research's public GitHub repository `https://github.com/joeydevries/booter-black-list/`.
- A significant improvement over previous Booter classification literature, including the analysis of testing multiple classification metrics for their effectiveness.

Additionally, the methodology described operates solely on Booters, but can easily be generalized to different classification areas by updating the feature characteristics to the respective area's domain.

7.2 How to use the Booter (black)list

The Booter (black)list can be used in mostly two ways. First and foremost the Booter (black)list serves as a list of Booter websites for future research. Given the high Booter accuracy rate we are confident that the list and future iterations of the list will prove a valuable resource for future Booter research. Second, the Booter (black)list can be used as a blacklist in operational networks. Given the small Type I error rate the Booter (black)list and future iterations are considered practical for use as a blacklist. Furthermore, based on the requirements of the system the threshold parameter T can be tweaked to further enhance the precision of the system more towards one end of the confusion matrix. Given the nature of blacklists however an ethical discussion is required as the chance of blocking a legitimate website still resides, albeit marginally. This becomes especially relevant when a blacklist is involved in large-scale operations as commonly executed by ISPs. However, even though non-Booters could end up blacklisted, the analysis in chapter 6.2 concluded that the blocked non-Booter websites can be regarded morally questionable.

7.3 Limitations

The research focused strongly on a set of heuristics that differentiate Booter websites from non-Booter websites, primarily the similarities between all hosted online Booter websites. While this is unlikely to change drastically in the near future, future trends could significantly change the visual and textual content of Booter websites. This does reduce the accuracy of the system, but the general assumptions remain that together with machine learning could still remain practical. Another limitation of the classifier system is that all classification parameters are based on the training dataset and end up biased towards the dataset. However, the training dataset is currently the best representation of a Potential Booter Domain name list available. The resulting parameters should thus best reflect a Booter dataset. As the Booter phenomenon will continue to grow, a better dataset can be envisioned. Furthermore, the proposed blacklist can be bypassed using a VPN or by directly storing IP-addresses as a blacklist would be based on domain names only. However, this does still require an above-average level of technical knowledge from its users which we believe is still a smaller portion of the Booter's user-base.

7.4 Future research

The accuracy rates are still not optimal and specifically the Type II error rates would benefit from a higher accuracy. The research showed the results of Booter classification using multiple classification schemes giving valuable insight into classification in the Booter context. We did leave out other classification metrics that could prove interesting in Booter classification like Logistic Regression and Support Vector Machines. Future research into different classification metrics and machine learning algorithms like neural networks could significantly boost the accuracy and flexibility of a Booter classification system.

7.5 Acknowledgements

I would like to thank my supervisors Jair J. Santanna, and Justyna S. Chromik for guiding me in the process of academic research; specifically Jair J. Santanna who continuously provided extensive feedback and helped me stay motived in the process. I would also like to thank Mena B. Habib for valuable insight and feedback into the Booter classification and Khelghati, S.M for constructive feedback on the crawler system. Furthermore, I want to thank the DACS group for having me work in their offices during my research while providing a healthy environment for academic discussions.

References

[1] Holl, P.: Exploring DDoS defense mechanisms. Future Internet (FI) and Innovative Internet Technologies and Mobile Communications(IITM) (2015) 25–32

[2] Anstee, D., Cockburn, A., Sockrider, G., Morales, C.: Worldwide infrastructure security report. http://pages.arbornetworks.com/rs/arbor/images/WISR2014.pdf (2014) Accessed on 31 March 2015.

[3] Sood, A., Enbody, R.: Crimeware-as-a-service: A survey of commoditized crimeware in the underground market. International Journal of Critical Infrastructure Protection **6**(1) (2013) 28–38

[4] Santanna, J., Durban, R., Sperotto, A., Pras, A.: Inside Booters: an analysis on operational databases. In: 14th IFIP/IEEE International Symposium on Integrated Network Management (IM 2015). IFIP/IEE (2015)

[5] Prolexic: Threat: DDoS Booter shell scripts. http://www.prolexic.com/knowledge-center-ddos-threat-advisories-booter-shell-scripts.html (2013) Accessed on 31 March 2015.

[6] Krebs, B.: The world has no room for cowards. http://krebsonsecurity.com/2013/03/the-world-has-no-room-for-cowards (2013) Accessed on 24 April 2015.

[7] Santanna, J., v. Rijswijk, R., Hofdstede, R., Sperotto, A., Wierbosch, M., Zambenedetti Granville, L., Pras, A.: Booters - An Analysis of DDoS-as-a-Service Attacks. In: International Symposium on Integrated Network Management. IFIP/IEE (2015)

[8] Prolexic: Multiplayer video gaming attacks. http://www.prolexic.com/knowledge-center-white-paper-gaming-reflection-attacks-drdos-ddos/infographic.html (2013) Accessed on 24 March 2015.

[9] Prolexic: Quarterly global DDoS attack report Q3. http://www.prolexic.com/knowledge-center-ddos-attack-report-2013-q3.html (2013) Accessed on 26 March 2015.

[10] Verisign: Distributed Denial of Service trends report. http://www.verisigninc.com/en_US/cyber-security/ddos-protection/ddos-report/index.xhtml?loc=en_US&dmn=ddostrendsinfographic?cmp=SO-DDOS-ABLOG (2014) Accessed on 26 March 2015.

[11] Technologies, A.: Akamai's [state of the internet] / security (q4 2014). http://www.stateoftheinternet.com/downloads/pdfs/2014-internet-security-report-q4.pdf (2014) Accessed on 31 March 2015.

[12] Chromik, J.J.: Booters (black)list. Master's thesis, Twente University (2015)

[13] Mirkovic, J., Reiher, P.: A Taxonomy of DDoS attack and DDoS defense mechanisms. SIGCOMM Computer Communication Review **34**(2) (2004) 39–53

[14] NSFOCUS: Analysis of ddos attacks on spamhaus and recommended solution. http://www.nsfocus.com/2013/SecurityView_0514/129.html (2013) Accessed on 7 April 2015.

[15] Networks, A.: The business value of ddos protection. https://www.arbornetworks.com/ddos/The%20Business%20Value%20of%20DDoS%20Protection.pdf (2011) Accessed on 14 April 2015.

[16] Levchenko, K., Pitsillidis, A., Chachra, N., Enright, B., Félegyházi, M., Grier, C., Halvorson, T., Kanich, C., Kreibich, C., Liu, H., et al.: Click trajectories: End-to-end analysis of the spam value chain. In: Security and Privacy (SP), 2011 IEEE Symposium on, IEEE (2011) 431–446

[17] Garg, D.: DDoS mitigation techniques - a survey. Conference on Advanced Computing, Communication and Networks (2011) 1302–1309

[18] Walfish, M.and Vutukuru, M., Balakrishnan, H., Karger, D., Shenker, S.: DDoS defense by offense. In: ACM SIGCOMM proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, Pisa, Italy, ACM (September 2006) 303–314

[19] Rossow, C.: Amplification hell: revisiting network protocols for DDoS abuse. In: Proceedings of the 2014 Network and Distributed System Security (NDSS) Symposium. (February 2014)

[20] Karami, M., McCoy, D.: Understanding the emerging threat of DDoS-as-a-Service. In: Proceedings of the 6th UNSENIX Workshop on Large-Scale Exploits and Emergent Threats. LEET'13 (2013)

[21] BBC: Hacktivists step up web attack volumes. http://www.bbc.com/news/technology-31000908 (2015) Accessed on 2 April 2015.

[22] hackforums: Hack forums. http://hackforums.net/forumdisplay.php?fid=232 Accessed on 14 April 2015.

[23] Centarra: 5 facts about ddos. https://centarra.com/2014/03/14/5-facts-about-ddos-attacks.html (2014) Accessed on 14 April 2015.

[24] Krebs, B.: The internet of dangerous things. http://krebsonsecurity.com/2015/01/the-internet-of-dangerous-things/ (2015) Accessed on 31 March 2015.

[25] Costolo, D.: Twitter. twitter.com (2006) Accessed on 2 June 2015.

[26] Google: Youtube. youtube.com (2006) Accessed on 2 June 2015.

[27] Krebs, B.: Spreading the disease and selling the cure. http://krebsonsecurity.com/2015/01/spreading-the-disease-and-selling-the-cure/ (2015) Accessed on 31 March 2015.

[28] Lackey, J.: A new twist on Denial of Service: DDoS as a Service. http://blogs.cisco.com/security/a_new_twist_on_denial_of_service_ddos_as_a_service/ (2010) Accessed on 21 April 2015.

[29] Santanna, J., Sperotto, A.: Characterizing and mitigating the ddos-as-a-service phenomenon. Monitoring and Securing Virtualized Networks and Services **8508** (2014) 74–78

[30] Die, M.M.: Ddos'er as service - a camouflage of legit stresser. http://blog.malwaremustdie.org/2014/06/ddoser-as-service-camouflation-of-legit.html (2014) Accessed on 7 April 2015.

[31] Krebs, B.: DDoS services advertise openly, take PayPal. http://krebsonsecurity.com/2013/05/ddos-services-advertise-openly-take-paypal/ (2013) Accessed on 21 April 2015.

[32] of Europe, C.: Convention on cybercrime, art. 5, 6.1ai, 6.1b, 6.2, 8b, 11. http://conventions.coe.int/Treaty/en/Treaties/Html/185.htm (2001) Accessed on 14 April 2015.

[33] US-CERT: UDP-based amplification attacks. http://www.us-cert.gov/ncas/alerts/TA14-017A (2014) Accessed on 31 March 2015.

[34] Nolla, A.: Amplification ddos attack with quake3 servers: An analysis. http://blog.alejandronolla.com/2013/06/24/amplification-ddos-attack-with-quake3-servers-an-analysis-1-slash-2/ (2013) Accessed on 16 April 2015.

[35] v. Rijswijk-Deij, R., Sperotto, A., Pras, A.: DNSSsec and Its Potential for DDoS Attacks. Proceedings of the Fourteenth ACM Internet Measurement Conference **14** (2014) 449–460

[36] Hoque, N., Monowar, B., Baishya, R., Bhattacharyya, D., Kalita, J.: Network attacks: taxonomy, tools and systems. Journal of Network and Computer Applications **40** (2013) 307–324

[37] CStress: The dominate method. http://cstress.net/blog/tag/dominate-ddos/ (2015) Accessed on 2 June 2015.

[38] SpiderLabs, T.: Wordpress xml-rpc pingback vulnerability analysis. https://www.trustwave.com/Resources/SpiderLabs-Blog/WordPress-XML-RPC-PingBack-Vulnerability-Analysis/ (2014) Accessed on 16 April 2015.

[39] of the internet, S.: Joomla reflection ddos-for-hire [high risk]. http://www.stateoftheinternet.com/resources-web-security-threat-advisories-2015-joomla-reflection-attack-ddos-for-hire.html (2015) Accessed on 16 April 2015.

[40] Prince, M.: The DDoS that almost broke the internet. http://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet (2013) Accessed on 31 March 2015.

[41] Prince, M.: Technical details behind a 400Gbps NTP amplification DDoS attack. http://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack (2014) Accessed on 31 March 2015.

[42] Krebs, B.: The new normal: 200-400 Gbps DDoS attacks. http://krebsonsecurity.com/2014/02/the-new-normal-200-400-gbps-ddos-attacks/ (2014) Accessed on 27 April 2015.

[43] Brian Prince, S.W.: Ddos attacks prove costly, could top 800 gbps in 2015: Research. http://www.securityweek.com/ddos-attacks-prove-costly-could-top-800-gbps-2015-research (2014) Accessed on 14 April 2015.

[44] Ludl, C., McAllister, S., Kirda, E., Kruegel, C.: On the effectiveness of techniques to detect phishing sites. In: Detection of Intrusions and Malware, and Vulnerability Assessment, Springer (2007) 20–39

[45] Sheng, S., Wardman, B., Warner, G., Cranor, L., Hong, J., Zhang, C.: An empirical analysis of phishing blacklists. In: Sixth Conference on Email and Anti-Spam (CEAS), California, USA (2009)

[46] Stol, W., Kaspersen, H., Kerstens, J., Leukfeldt, E., Lodder, A.: Governmental filtering of websites: the Dutch case. Computer Law & Security Review **25**(3) (2009) 251–262

[47] Lindemann, C., Littig, L.: Coarse-grained Classification of web sites by their structural properties. Proceedings on Web information and data management **8** (2006)

[48] Felegyhazi, M., Kreibich, C., Paxson, V.: On the potential of proactive domain blacklisting. LEET'10 Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms and more **3** (2010) 6

[49] Brennan, C.: Private domain name registration (2006) US Patent App. 11/008,610.

[50] Integralis: Ddos protection bypass techniques. https://media.blackhat.com/us-13/US-13-Nixon-Denying-Service-to-DDOS-Protection-Services-WP.pdf (2013) Accessed on 2 April 2015.

[51] Alexa: Alexa website. http://www.alexa.com (2015) Accessed on 23 April 2015.

[52] Chu, W., Zhu, B., Xue, F., Guan, X., Cai, Z.: Protect sensitive sites from phishing attacks using features extractable from inaccessible phishing urls. In: Communications (ICC), 2013 IEEE International Conference on, IEEE (2013) 1990–1994

[53] Kausar, F., Al-Otaibi, B., Al-Qadi, A., Al-Dossari, N.: Hybrid client side phishing websites detection approach. International Journal of Advanced Computer Science and Applications (IJACSA) **5**(7) (2014)

[54] Hammami, M., Chahir, Y., Chen, L.: WebGuard: A web filtering engine combining Textual, Structural, and Visual content-based analysis. IEEE Transactions on Knowledge & Data Engineering **18** (2006)

[55] Lindemann, C., Littig, L.: Classifying web sites. International conference on World Wide Web **16** (2007)

[56] Insoon Jo, E., Heon Y, Y.: Interactive website filter for safe web browsing. Journal of Information Science and Engineering **29**(1) (2013) 115–131

[57] Kriegel, H., Schubert, M.: Classification of Websites as Sets of Feature Vectors. In: Proceedings of the IASTED International Conference Databases and Applications. (2004) 127–132

[58] Reitz, K.: Requests: Http for humans (2015)

[59] Vorona, A.: Cloudflare scrape (2015)

[60] Slootweg, S.: Python whois (2013)

[61] Foundation, P.S.: The elementtree xml api (2015)

[62] Dower, S.: Cpython: urllib.urlparse (2015)

[63] Developers, G.: Google web search api (2010)

[64] Huggins, J., Gross, P., Wang J, T.: Selenium, browser automation (2004)

[65] Aggarwal, C., Hinneburg, A., Keim, D.: On the surprising behavior of Distance Metrics in high dimensional space. Database Theory – ICDT. Lecture Notes in Computer Science **1973**(1) (2001) 420–434

[66] Chang, T., Kuo, C.C.J.: Texture analysis and classification with tree-structured wavelet transform. Image Processing, IEEE Transactions on **2**(4) (1993) 429–441

[67] Kaplantzis, S., Mani, N.: A study on classification techniques for network intrusion detection. In: IASTED Conference on Networks and Communication Systems (NCS 2006), Thailand. (2006)

[68] Hinneburg, A., Aggarwal, C., Keim, D.: What is the nearest neighbor in high dimensional spaces? (2000)

[69] Howarth, P., Rger, S.: Fractional distance measures for content-based image retrieval. Advances in Information Retrieval Lecture Notes in Computer Science, 27th European Conference on IR Research, ECIR 2005 **3408**(1) (2005) 447–456

[70] Sun, C., Rampalli, N., Yang, F., Doan, A.: Chimera: large-scale classification using Machine Learning, Rules and Crowdsourcing. Proceedings of the VLDB Endowment **7**(13) (2014) 1529–1540

[71] Weinberger, K., Blitzer, J., Saul, L.: Distance metric learning for large margin nearest neighbor classification. In: Advances in neural information processing systems. (2005) 1473–1480

[72] Ma, J., K. Saul, L., Savage, S., M. Voelker, G.: Beyond Blacklists: learning to detect malicious web sites from suspicious URLs. Proceedings of the 15th ACM SIGKDD International conference on Knowledge **15** (2009) 1245–1254

[73] Greenacre, M.: Measures of distance between samples: Euclidean. (2008)

[74] Garera, S., Provos, N., Chew, M., Rubin, A.: A framework for detection and measurement of phising attacks. In: Proceedings of the 2007 ACM workshop on Recurring malcode, ACM (2007) 1–8

[75] Statsoft: Cluster analysis. `http://www.statsoft.com/Textbook/Cluster-Analysis` (2011) Accessed on 20 April 2015.

[76] Keogh, E.: Naive bayes classifier. `http://www.cs.ucr.edu/~eamonn/CE/Bayesian%20Classification%20withInsect_examples.pdf` Accessed on 9 April 2015.

[77] Rish, I.: An empirical study of the naive Bayes classifier. In: IJCAI 2001 workshop on empirical methods in artificial intelligence. Volume 3., IBM New York (2001) 41–46

[78] Cover, T., Hart, P.: Nearest neighbor pattern classification. Information Theory, IEEE Transactions on **13**(1) (1967) 21–27

[79] Peterson, L.E.: K-nearest neighbor. Scholarpedia **4**(2) (2009) 1883 revision #136646.

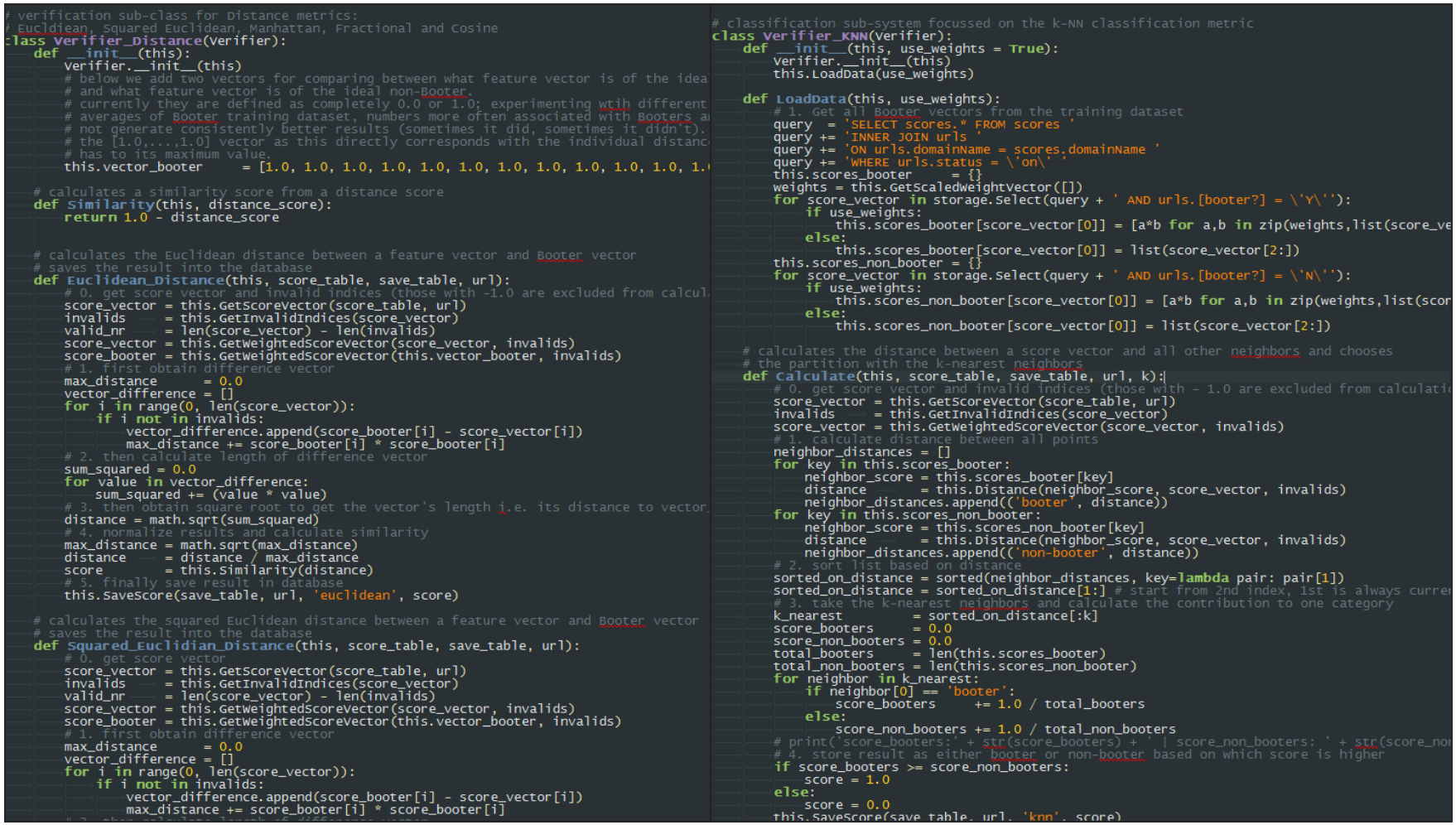
[80] H, G.: A simple algorithm for fitting a gaussian function. Signal Processing Magazine **28**(5) (2011) 134–137

Part IV
Appendix A

```python
class Crawler_Google2(Crawler):
    'Crawler of Google via default web requests'
    def __init__(this, sleep_level=1):
        domain = 'https://www.google.com/search?ie=UTF-8&num=100'
        Crawler.__init__(this, domain, sleep_level)
        this.PrintNote('CRAWLING GOOGLE2')
        this.PrintDivider()
        this.Initialize()

    # overrides Crawler's crawl function
    def Crawl(this, max_results=100):
        keywords = ['Booter', 'DDOSer', 'Stresser']

        nr_pages = int(max_results / 100)
        this.PrintUpdate('initiating crawling procedures: Google')

        for keyword in keywords:
            this.PrintDivider()
            this.PrintNote('KEYWORD: ' + keyword)
            this.PrintDivider()
            for i in range(0, nr_pages):
                counter = 0
                # dynamically generate search query
                query =  "&q=" + keyword  + '&start=' + str(i * 100) + '&filter=0'
                url = this.Target + query

                this.PrintDebug('crawling: ' + query)
                # read html and parse JSON
                response = this.JSCrawl(url)
                tree    = html.fromstring(response.text)
                urls = tree.xpath('(//div)[@class="g"]//h3[@class="r"]/a/@href')

                split   = 10
                for url in urls:
                    try:
                        # parse url
                        if '/url?q=' in url:
                            url = url[7:].split('&sa')[0]
                        this.AddToList(BooterURL(url), 'Google')

                        if counter % split == 0:
                            this.PrintDivider()
                        counter = counter + 1
                    except Exception as ex:
                        this.PrintError('EXCEPTION: ' + str(ex))
                this.Sleep()

        this.PrintUpdate('DONE; found ' + str(len(this.URLs)) + ' potential Booters')
        this.PrintDivider()
```

```python
        average_depth_level = 1.0
    else:
        average_depth_level = max(1.0 - ((average_depth_level_raw - 1.0) / 2.0), 0.0)
    this.PrintUpdate('average depth level: ' + str(average_depth_level_raw))

    # - 1.4. Average URL length
    average_url_length     = -1.0
    average_url_length_raw = -1.0
    for page in inbounds: # use inbounds, not pages crawled as they give much more results
        average_url_length_raw = average_url_length_raw + len(page)
    # calculate score: interpolate linearly from lowest occurence to highest Booter occurence
    average_url_length_raw = average_url_length_raw / len(inbounds)
    if average_url_length_raw <= 15:
        average_url_length = 1.0
    else:
        average_url_length = max(1.0 - ((average_url_length_raw - 15) / 15), 0.0)
    this.PrintUpdate('average url length: ' + str(average_url_length_raw))


    ### 2. content-based characteristics
    this.PrintDivider()
    this.PrintUpdate('obtaining content-based characteristics')
    this.PrintDivider()

    # get whois information
    # "Each part represents the response from a specific WHOIS server. Because the WHOIS da
    # servers to follow a unique response layout, each server needs its own dedicated parser."
    domain_age                  = -1.0
    domain_age_raw              = -1.0
    domain_reservation_duration     = -1.0
    domain_reservation_duration_raw = -1.0
    try:
        with timeout(seconds=10):
            whois = pythonwhois.get_whois(landing_page.GetTopDomain(), False) # http://cryto.
    except Exception as ex:
        this.PrintError('EXCEPTION: get WHOIS data: ' + str(ex))
    try:
        # - 2.1. Domain age
        current_date    = datetime.datetime.today()
        date_registered = whois['creation_date'][0]
        domain_age_raw    = (current_date - date_registered).days
        # calculate score: linear interpolation between current_date and first occurence of
        # booter in data: 2011
        days_since_first = (current_date - datetime.datetime(2011, 10, 28)).days
        domain_age = max(1.0 - (domain_age_raw / days_since_first), 0.0)
        this.PrintUpdate('domain age: ' + str(domain_age_raw))
    except Exception as ex:
        this.PrintError('EXCEPTION: whois keywords, likely registrar: ' + str(ex))

    try:
        # - 2.2 Domain reservation duration
        current_date                = datetime.datetime.today()
        expire_date                 = whois['expiration_date'][0]
        domain_reservation_duration_raw = (expire_date - current_date).days
        # calculate score: between 1 - 2 years; < 1 year = 1.0
        if domain_reservation_duration_raw < 183:
            domain_reservation_duration = 1.0
        else:
```

Figure 21: Snippets of the crawler system's source code.



```
# verification sub-class for Distance metrics:
# Euclidean, Squared Euclidean, Manhattan, Fractional and Cosine
class Verifier_Distance(Verifier):
    def __init__(this):
        Verifier.__init__(this)
        # below we add two vectors for comparing between what feature vector is of the idea
        # and what feature vector is of the ideal non-Booter.
        # currently they are defined as completely 0.0 or 1.0; experimenting wtih different
        # averages of Booter training dataset, numbers more often associated with Booters a
        # not generate consistently better results (sometimes it did, sometimes it didn't).
        # the [1.0,...,1.0] vector as this directly corresponds with the individual distanc
        # has to its maximum value.
        this.vector_booter     = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.

    # calculates a similarity score from a distance score
    def Similarity(this, distance_score):
        return 1.0 - distance_score


    # calculates the Euclidean distance between a feature vector and Booter vector
    # saves the result into the database
    def Euclidean_Distance(this, score_table, save_table, url):
        # 0. get score vector and invalid indices (those with -1.0 are excluded from calcul
        score_vector = this.GetScoreVector(score_table, url)
        invalids     = this.GetInvalidIndices(score_vector)
        valid_nr     = len(score_vector) - len(invalids)
        score_vector = this.GetWeightedScoreVector(score_vector, invalids)
        score_booter = this.GetWeightedScoreVector(this.vector_booter, invalids)
        # 1. first obtain difference vector
        max_distance      = 0.0
        vector_difference = []
        for i in range(0, len(score_vector)):
            if i not in invalids:
                vector_difference.append(score_booter[i] - score_vector[i])
                max_distance += score_booter[i] * score_booter[i]
        # 2. then calculate length of difference vector
        sum_squared = 0.0
        for value in vector_difference:
            sum_squared += (value * value)
        # 3. then obtain square root to get the vector's length i.e. its distance to vector
        distance = math.sqrt(sum_squared)
        # 4. normalize results and calculate similarity
        max_distance = math.sqrt(max_distance)
        distance     = distance / max_distance
        score        = this.Similarity(distance)
        # 5. finally save result in database
        this.SaveScore(save_table, url, 'euclidean', score)

    # calculates the squared Euclidean distance between a feature vector and Booter vector
    # saves the result into the database
    def Squared_Euclidian_Distance(this, score_table, save_table, url):
        # 0. get score vector
        score_vector = this.GetScoreVector(score_table, url)
        invalids     = this.GetInvalidIndices(score_vector)
        valid_nr     = len(score_vector) - len(invalids)
        score_vector = this.GetWeightedScoreVector(score_vector, invalids)
        score_booter = this.GetWeightedScoreVector(this.vector_booter, invalids)
        # 1. first obtain difference vector
        max_distance      = 0.0
        vector_difference = []
        for i in range(0, len(score_vector)):
            if i not in invalids:
                vector_difference.append(score_booter[i] - score_vector[i])
                max_distance += score_booter[i] * score_booter[i]
```

```
# classification sub-system focussed on the k-NN classification metric
class Verifier_KNN(Verifier):
    def __init__(this, use_weights = True):
        Verifier.__init__(this)
        this.LoadData(use_weights)

    def LoadData(this, use_weights):
        # 1. Get all Booter vectors from the training dataset
        query  = 'SELECT scores.* FROM scores '
        query += 'INNER JOIN urls '
        query += 'ON urls.domainName = scores.domainName '
        query += 'WHERE urls.status = \'on\' '
        this.scores_booter     = {}
        weights = this.GetScaledWeightVector([])
        for score_vector in storage.Select(query + ' AND urls.[booter?] = \'Y\''):
            if use_weights:
                this.scores_booter[score_vector[0]] = [a*b for a,b in zip(weights,list(score_ve
            else:
                this.scores_booter[score_vector[0]] = list(score_vector[2:])
        this.scores_non_booter = {}
        for score_vector in storage.Select(query + ' AND urls.[booter?] = \'N\''):
            if use_weights:
                this.scores_non_booter[score_vector[0]] = [a*b for a,b in zip(weights,list(scor
            else:
                this.scores_non_booter[score_vector[0]] = list(score_vector[2:])

    # calculates the distance between a score vector and all other neighbors and chooses
    # the partition with the k-nearest neighbors
    def Calculate(this, score_table, save_table, url, k):
        # 0. get score vector and invalid indices (those with - 1.0 are excluded from calculati
        score_vector = this.GetScoreVector(score_table, url)
        invalids     = this.GetInvalidIndices(score_vector)
        score_vector = this.GetWeightedScoreVector(score_vector, invalids)
        # 1. calculate distance between all points
        neighbor_distances = []
        for key in this.scores_booter:
            neighbor_score = this.scores_booter[key]
            distance       = this.Distance(neighbor_score, score_vector, invalids)
            neighbor_distances.append(('booter', distance))
        for key in this.scores_non_booter:
            neighbor_score = this.scores_non_booter[key]
            distance       = this.Distance(neighbor_score, score_vector, invalids)
            neighbor_distances.append(('non-booter', distance))
        # 2. sort list based on distance
        sorted_on_distance = sorted(neighbor_distances, key=lambda pair: pair[1])
        sorted_on_distance = sorted_on_distance[1:] # start from 2nd index, 1st is always curre
        # 3. take the k-nearest neighbors and calculate the contribution to one category
        k_nearest          = sorted_on_distance[:k]
        score_booters      = 0.0
        score_non_booters = 0.0
        total_booters      = len(this.scores_booter)
        total_non_booters = len(this.scores_non_booter)
        for neighbor in k_nearest:
            if neighbor[0] == 'booter':
                score_booters     += 1.0 / total_booters
            else:
                score_non_booters += 1.0 / total_non_booters
        # print('score_booters: ' + str(score_booters) + ' | score_non_booters: ' + str(score_no
        # 4. store result as either booter or non-booter based on which score is higher
        if score_booters >= score_non_booters:
            score = 1.0
        else:
            score = 0.0
        this.SaveScore(save_table, url, 'knn', score)
```

Figure 22: Snippets of the classifier system's source code.