Characterisation and detections of Third-party content loading in the Web

By

Hasina Rahman

A thesis submitted to Macquarie University
for the degree of Master of Research
Department of Computing
March 2019

MACQUARIE
University
SYDNEY · AUSTRALIA

Examiner's Copy

Declaration

I certify that the work in this thesis entitled CHARACTERISATION AND DETECTIONS OF THIRD-PARTY CONTENT LOADING IN THE WEB has not previously been submitted for a degree nor has it been submitted as part of the requirements for a degree to any other university or institution other than Macquarie University. I also certify that the thesis is an original piece of research and it has been written by me. Any help and assistance that I have received in my research work and the preparation of the thesis itself have been appropriately acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Hasina Rahman Digitally signed by Hasina Rahman
Date: 2019.03.07 12:43:36 +11'00'

Hasina Rahman

Acknowledgements

I would like to thank the Almighty Allah (God) for his guidance and tranquility in my heart throughout. I would also thank my supervisor Prof. Mohamed Ali Kaafar for being supportive during the research project and Dr. Muhammad Ikram for his guidance in the project. Further, I would thank my husband Nazim-uddin-Sheikh for his immense support and help.

Abstract

The Web has evolved into a tangled mass of interconnected services within the last two decades, where websites import resources i.e. data or contents from third-party domains. These domains serve several purposes including analytics, tracking and advertisement. Websites trust their third-parties for resources in the process of loading contents or data to their web pages. The dependency of resources sometimes extent further from third-party domains to other domains thus fabricating a chain of dependency. In the resource dependency chain, the first party websites are indeed trusting on resources obtained by their direct third-parties through requests to other domains. The chain of dependency cannot be rigidly controlled by the first-party websites as they have very scarce or no information of where the loaded content have originated from. Since this is the case, the websites even end up trusting compromised websites for contents unknowingly and become prone to multifarious attacks. We characterize the implicit trust in the chain of dependency for Alexa's top 30k websites and estimate the level of risks that first-party websites may be venturing while loading resources from thirty-party domains. We found that 10.55% of the resources of top-1000 Alexa websites are obtained implicitly and that they constitute 4.1% of malicious resources in the overall count of external resources.

Contents

Declaration iii

Acknowledgements v

Abstract vii

List of Figures xi

List of Tables xiii

1 Introduction 1

2 Literature Review 7

2.1 Web security and privacy: 7

2.2 Resource inclusion and web-infection spread: 9

2.3 Web measurement: 11

3 Technical Background 15

3.1 Website rendering 16

3.2 Defining maliciousness 17

3.3 Malicious websites 17

3.4 Brief history of malicious third-party content 18

3.5 Malicious content spread through the web: 18

3.5.1 JavaScript-based attacks: 19

3.5.2 Image based attacks using web content: . . . 20
3.5.3 Attacks based on CSS: . . . 20
3.5.4 Malicious PHP script based attacks: . . . 21
3.5.5 Attacks based on other web resources: . . . 21
3.6 Analysis Methodologies . . . 21
3.6.1 Static Analysis . . . 21
3.6.2 Dynamic Analysis . . . 22

4 Experimental Evaluation **25**
4.1 Data-set . . . 25
4.1.1 Data-set Refinement: . . . 27
4.2 Methodology . . . 30
4.3 Evaluation . . . 32

5 Conclusion **45**

References **47**

List of Figures

1.1 The Chain of Trust for resources . . . 3

3.1 Cuckoo interface . . . 23

4.1 Crawl Screenshot . . . 26
4.2 CDF of the percentage of First-party and obtained resources . . . 33
4.3 CDF of the external resources in different levels . . . 34
4.4 Percent of the different external resource types in different levels . . . 35
4.5 External Resources from different categories in different levels . . . 36
4.6 CDF of malicious external resources in different levels . . . 37
4.7 Percent of malicious external Resources Types . . . 38
4.8 CDF of malicious external resources types . . . 39
4.9 VT scores of different resource types in level 1 . . . 39
4.10 VT scores in different categories . . . 40
4.11 VT scores in level 1 for different categories . . . 41
4.12 VT scores in level 2 for different categories . . . 41
4.13 VT scores in levels>=3 for different categories . . . 42
4.14 percent of various classes of malware . . . 42

List of Tables

4.1 Internal and External resource count of 1-30K Alexa ranked websites in different levels. 28
4.2 Internal and External resource count of 1st 10K Alexa ranked websites in different levels. 28
4.3 Internal and External resource count of 2nd 10K Alexa ranked websites in different levels. 29
4.4 Internal and External resource count of 3rd 10K Alexa ranked websites in different levels. 29
4.5 External resource count according to resource types in different levels. 34
4.6 External resource count according to resource types in different levels. 36
4.7 Percentage of resources in different VT Score ranges. 37

1 Introduction

In the modern web eco-system, websites are always in the process of introducing external resources (i.e. data or contents) such as AJAX HTTP requests, JavaScript, style sheets (created using CSS where CSS is a styling language), Shock Wave Flash files also known as Flash objects, iframes and multimedia files such as images, videos and text to their web pages through requests [1]. More than 90% of websites depend on content from external domains [2]. These websites also referred to as the first-parties, include resources by making requests to other websites, thereby trying to enhance the capability of engaging the end-users. The first-party websites load resources from their third-parties that serve several purposes such as analytics, advertisements, tracking, content distribution networks (CDNs) and external dynamic content display. The third-parties are those websites that are apart from the principal domains and may convey the resources themselves or from other websites. When a user visits a certain website and queries for some data such as image, video or audio that the website itself does not contain, it makes requests to its third-parties and the links of those third-parties connected through explicit or implicit trusts in the dependency chain are reached via the hyperlinks embedded within the web pages. The links to retrieve data are generally embedded inside script tags as they are acquired through highly dynamic codes. The codes are in various scripting forms such as JavaScript, ActionScript and ActiveX. These scripts inside script tags are parsed, compiled and executed when the web pages are rendered by the browser machine. The data is obtained pro grammatically through links to the contents written in languages such as JavaScript, HTML or CSS.

The website-developers in the course of requesting resources from the third-parties enable them with the control of executing their codes with the same entitlement as themselves that may lead them to several vulnerabilities. Moreover the third-parties may sometimes rely on other domains for resources while the first-parties often have little if any visibility of where the resources have originated from. This forms a chain of resource-dependency between first and third-parties and the domains on which the third-parties further depend on for loading contents.

The notion of explicit trust is defined as the first-party websites trusting on their direct third-parties for loading resources. While the trust posed by the websites on the unknown third-parties through the extended chain of dependency is the implicit trust since the resources and the sources derived from are unknowingly trusted. The first-parties therefore, in the course of including contents provided by their third-parties, establish an explicit trust with them. In some cases however, the third-party domains might in turn make appeals for contents further from other websites and thus an implicit trust is built between the first parties and the third-party domains in the dependency chain down the track.

The first-parties are considered to be at level 0 with their direct third-parties at level 1 and the corresponding content-lending domains of these third-parties to be at level 2 that may extend further to greater levels in the dependency chain as shown in Figure. Thus, the first-parties are said to have an explicit trust on their immediate third-parties that are at level 1 and implicit trust on the domains with levels 2 or more. The first-parties have very little knowledge if any, about the domains above level 1 along the dependency chain as well as the volume of content loaded from them. For example the British Broadcasting web page (bbc.com) bbc.com may distinctively load

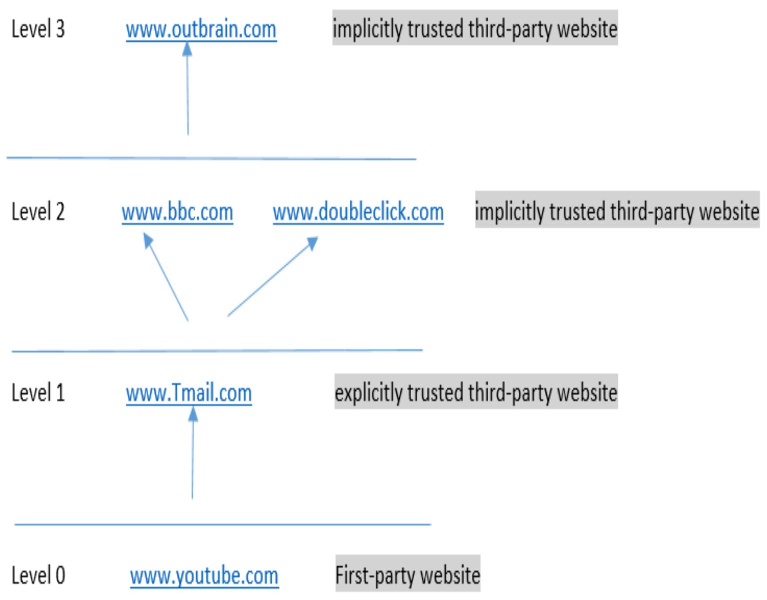


FIGURE 1.1: The Chain of Trust for resources

lines of JavaScript codes from DoubleClick.com which further loads more content from another third-party website say, Outbrain.com. Here bbc.com serves as the first-party website that explicitly trusts DoubleClick.com and implicitly trusts Outbrain.com. In this case, bbc.com might encounter a compromised (malicious) domain serving malicious scripts, down the dependency chain while importing active content (e.g. JavaScript) which would result in serious security vulnerabilities. This dependency of resources and implicit trust has been introduced by Kumar et.al [2] that drew our attention towards the increase in the number of externally loaded resources with the content loaded being one-third of the resources imported.This nature of trust driven by the first-party websites on contents from unknown domains as sources that may be compromised or malicious, makes them vulnerable to several threats such as phishing or drive-by-download attacks or even attacks by polymorphic malware.

Web-based malware is a prevalent threat to the Internet security that is growing with time. Some millions of malicious URLs are used as sources to spread the infection throughout the web community [3]. Large number of attacks are framed through third-party domains and the trust made by the first-party websites on non-visible third parties is thus exploited as the implicit trust may even be made on unauthenticated parked domains which has risen to be a noticeable threat [4]. According to the Australian Cyber Security Centre (ACSC) 2017 Threat Report, the "frequency, scale, sophistication, and severity" of cybercrimes have increased, and "more diverse and innovative attempts" are made by attackers to target the public and private sector networks and the scale of DDoS incidents is now vast [5]. According to the report, the currently challenging malware programs are credential-harvesting malware (e.g. Gozi) and Ransomware (most popular variants being Cryptolocker, Torrentlocker and Cryptowall). In the U.S., a new record high of 1,579 data breach incidents were perceived, according to the 2017 Data Breach Year-End Review released by the Identity Theft Resource Centre (ITRC) and CyberScout [6]. This review indicated a drastic upturn of 44.7% increase over the record high figures reported for 2016. It has transpired that the brisk spreading of these security over a couple of years is due to the anonymous loading of data by implicit trust between first-party websites and extended (level 2 and above) third-parties.

The web has become a platform for cyber-criminals to carry out various online criminal activities such as Phishing (fraudulent access to user's credentials), advertisement of counterfeiting products such as drugs, medicines from illegitimate sellers and porn videos. According to the 2018 reports by an U.S. news channel "The Guardian", Facebook and Google tracks their users information even beyond what a user can realise. In essence of that, there is a privacy breach in cases when one such first-party websites data is illegitimately used by their third-parties.

Third-party tracking have become a common issue with the user behaviour being tracked through elements including cookies and advertisements leading to privacy breach [7]. Gomer et al. [8] provides a comprehensive image of the characteristics of third-party tracking networks that bespoke the fact that search rankings are not given for the references to third-parties resulted from search but on the basis of tracked searching nature of users.

Since some years now, JavaScript/DOM based attacks are delineated as the top Internet security threats [9, 10]. JavaScript codes are active contents that are included by third-parties through both explicit and implicit trust and are considered as the most precarious of all resources that serve malicious contents.

The crux of the problem is that first-party websites have little if any visibility of the resources loaded from their domain's chain of dependency. The dynamic nature of the loaded content loaded and the wide adoption of traffic alterations in the path between hosts (through endpoints/ middleboxes [11] or ISPs [12]) further complicates the issue as tracking back the loaded resources becomes even more difficult. This particularly happens in case of loading active content such as JavaScript codes, on execution of which paves way for a series of exploits for the requesting websites, sometimes leading to layer-7 DDoS attacks [13] or massive ransomware campaigns [14] thus grasping the control from the developer completely. Interestingly, Ad networks and CDNs have incremented their level of protection against those exploits by keeping sensitive libraries like Java or Flash up-to-date or blocking unsafe plugins [15]. However, attackers have also switched to alternate and more advanced strategies like malvertising [16] and social engineering popups [17]. Malvertising is a technique of assimilation of malware within advertisements. Thus the web is no longer secure and the level of risk can no longer be ignored.

The post-infection period sees the victim systems under the control of the attackers, who make use of these systems to perform various forms of cybercrimes such as spamming, distributed denial-of-service (DDoS) attacks and credential thefts.

The contributions of the thesis are as follows: (i) We characterize the implicit trust in the chain of dependency for Alexa's top 30k websites and estimate the level of risks that first-party websites may be venturing while loading resources from thirty-party domains. We also define the notion of explicit trust and implicit trust involving the direct and indirect loading of resources from third-parties respectively.

(ii) We classify the third-party domains based on their categories, resources they deliver and malicious or suspicious domains at different levels for each of the first-party websites. We then analyse the percentage of malicious contents uploaded or acquired from its third-parties and decide on the level of maliciousness according to type of malware detected. It further devises the chain of dependency by tracking the external resources obtained from the malicious or suspicious domains, for each of the top 30k websites, further measuring the volume of resources imported from malicious domains at each level of dependency for each first-party website.

(iii) Moreover, we set the level of maliciousness with the help of obtained scores from VirusTotal [18], and Cuckoo Sandbox [19]. We utilize the scores to judge whether a domain is malicious or not along with the level of maliciousness in case the domain is found malicious. VirusTotal and

Cuckoo delves into the content to pose certain signatures along with the maliciousness scores which we further analyzed to reach into a comprehensive conclusion on the level of maliciousness of the third-party domain. Cuckoo helped us in finding new suspicious contents, the code-signatures of those that were not discovered earlier and then decide from the features of cuckoo signatures whether they were malicious or just suspicious and further decide on the level of severity or maliciousness of that domain if it is malicious. We thus further interpret on the reasons for the acquired scores with further study. We also make use of the PhishTank database is used as a source of classifying phishing sites as well with not just judging on the Phishing scores obtained from the online application of VirusTotal.

(iv) Furthermore, we analyze the suspicious URLs of websites as well for obfuscated malicious codes.

2 Literature Review

In this section, we review some previous works that characterise the activities on the web. Moreover, we would also illustrate the quantity of attacks that are undermined by some of the most popular websites in the Alexa's ranking. Some previous works on areas such as web security and privacy, web resource loading and spread of malicious activity as well as large scale web measurement constitute the basis of following related works.

2.1 Web security and privacy:

Tracking of one's personal data by third-party web trackers has influenced the presence of third-party advertisements and statistical computation services on the internet as well on technologies in the mobile web. The web trackers are dispersed across a network and they track users as soon as the users come across such networks while performing search operations through search engines. The exposure of users to trackers through the third-party networks referred from search queries are such that more than 99.5% of users are tracked by 10 most popular trackers within 30 clicks. This is the case, when popular search engines such as Google, Bing and Baidu are considered [8].

The third-party tracking ecosystems across various countries gather data based on an individual's nature of browsing, demographical data and certain patterns of activities. The ad-tracker cookies are used to identify individuals throughout the web ecosystem. Some thousands of third-party websites are found to be involved in the measurements, for only top-500 Alexa ranked sites. Some of the most popular websites such as Google, AOL and Yahoo are revealed to be the owners of the maximum number of third-party web trackers [7]. Earlier, traces of Google were found as a leading enterprise with United States being pervasive in third-party web tracking trade [20, 21].

The dangling DNS records are highly neglected though they are a notable hazard caused to the web ecosystem since these records could be simply exploited by adversaries and the unknown extended third-parties might be utilising such domains. There is a noticeable number of such records among top 10,000 Alexa sites and edu zones [4]. These dangling records can form the cause of various attack vectors such as phishing, SQLinjection and DDoS.

In today's web eco-system, the revenue constraints of websites are met through privacy breach of users. Bashir et al. [22] studied the resource inclusion trees of websites and analysed retargeted ads using crowdsourcing that categorized the information shared between ad markets thus making us aware of the underlying abstraction of the information flow between these markets. Retargeted ads are those ads that tag online users with the help of a pixel within the target web page or email, which sets a cookie in the user's browser. Once the cookie is set, the advertiser is able to show display ads to that user elsewhere on the internet via an ad exchange and the ads are served to these targeted users based on their previous Internet actions. Reputed Companies like Google have been verified to be using the clause of sharing data in its privacy policy for retargeted ads. Thus, user's privacy breach occurs to meet the revenue obligations of first-party websites. The ad cookies may even serve malicious contents or private credentials collected through cookies may further lead to fraudulent access.

2.2 Resource inclusion and web-infection spread:

In the age of web 2.0 technologies that accentuates user-generated content, usability, participatory and interoperability, the web has increased vulnerabilities that are exploited through various attack vectors implanted by inclusion of malicious resources. The web 2.0 technologies is referred to as the newer version of World Wide Web version 1.0, that enables users with various interactive activities in a virtual community or social media.

The programming language, Java was specifically designed for web applications with the growth of the world-wide-web. Though it was designed keeping security in mind, it has several security vulnerabilities. A significant number of working, unique, easily reproducible Java exploits exist [23]. The exploits can be classified into three main attack vectors as: single-step for those that target a single vulnerability, restricted-class being those that utilize restricted class combining them with multiple archaic building block for vectors, to perform attacks and information hiding as those that target a series of vulnerabilities to reveal the hidden information. JavaScript is an interpreted language, also known as the language of the web on the client side as it is ubiquitously used throughout the web. It is one of the three essential elements of the web content along with HTML and CSS. JavaScript code expands the functionality of the web applications and refines the user experience making it interactive. JavaScript has access to browser cookies so it can obtain information collected by cookies. It can send HTTP requests with any content to any destination using XMLHTTPRequest and other technologies. [24] It has four sort of security vulnerabilities as follows: cross site scripting, open redirect, information leakage and code injection. Firstly, client XSS (There are two types of XSS: client and server XSS while JavaScript is used as the client side scripting language and it's vulnerabilities are exploited to a devastating extent throughout the web) is a security vulnerability for the web application where an attacker can inject malicious HTML code into the values of parameters rendered by JavaScript code. For example if there is path through an URL as 'https://www.x.com?val= <script> (malicious code/snippet) </script>', the malicious JavaScript snippet will alter the Document Object Model (DOM), where the DOM is a tree-like data structure of a web page. Secondly, the vulnerability called open redirect in JavaScript code, can be used to redirect the user to a malicious website. For example, an URL as ‘https://www.x.com?val= www.malwaresite.com’ can redirect the user to a website that is either malicious or compromised. Thirdly, information leakage is another type of vulnerability where JavaScript snippet can be used to cause leakage of sensitive information obtained from browser cookies about a user, through XMLHTTPRequest to outside over a network. Lastly, eval() function in JavaScript can be used to inject malicious code that can be executed dynamically during run-time.

Nikiforakis et al. [14] demonstrated in 2012 that large proportions of websites rely on JavaScript libraries hosted on ill-maintained external web servers making JavaScript exploits trivial while that was not the case. The web developers rely on JavaScript for alluring users and the trust relationships of the top 10,000 Alexa domains and their third-party JavaScript providers is worrisome. There are possibilities of different attacks through remote script inclusions, stale domain and IP-address inclusions and redirection to typo squatting domains through wrong inclusions by web developers. There has been gradual increase in the new JavaScript inclusions throughout the years since 2001 till 2010 and these inclusions have increased more in the recent years thus increasing the risks further.

The inclusion of most sensitive libraries that have high vulnerability is common among popular websites. Since web developers generally include JavaScript libraries from third-parties to elevate their functionalities, they become more vulnerable to attacks as considerable number of these websites do not use the updated or revised version of the libraries [25].

Malicious JavaScript codes can be obfuscated and injected to compromised websites such that the encoded code can easily bypass the signature based anti-virus programs or scanners [26]. The technique of obfuscation converts the bytes of strings or code into another form without changing it's original meaning. The transformed code contains the original code divided into parts such that these parts are allocated to different variables and are later recombined for execution. There are roughly three types of obfuscation techniques: randomization, data and encode where randomization substitutes some elements of codes with random strings, data breaks a string into numerous sub-parts and allocates them to variables and encode converts JavaScript to ASCII or Unicode. These codes can be used to carry out all sorts of malicious activities such as website redirects, drive-by-downloads and exploitation of vulnerabilities. Drive-by-download is the automatic download of malicious program without the user being deceived into giving consent for. Recent studies reveal that obfuscation is employed in a considerable amount of malicious JavaScript codes [27]. In 2009, drive-by-download attacks were implanted using a virus named Gumblar that was spread using obfuscated JavaScript code. The virus caused a great deal of damage by exploiting the vulnerabilities of Internet Explorer and Adobe Applications.

The various other web content resources that spread the infection are files such as CSS[28], PHP [29] and images [30]. These resources seem to be absolutely harmless while they are pernicious and can be used to cause heavy damage.

2.3 Web measurement:

The immensity of web based attacks can be measured through various techniques such as Firewalls, Intrusion Detection, Signature based detection, Heuristic Detection, Honeypot Detection and Machine learning based detection[31]. Nowadays, these techniques are sometimes used interchangeably as well as interdependently, since they are combined in various ways to detect the malicious pieces or codes. The above mentioned technologies can be broadly classified under two categories, namely static and dynamic, depending on the type of features (whether static features of web scripts or dynamic run-time features) they employ for detection. Signature technology is specifically based on static features while honeypots deploy dynamic feature based detection. Machine learning techniques generally deploy static features for detection and measurement. There are several works on static [27, 32–36] or dynamic or both [37].These techniques can be used to detect malicious elements such as codes, hidden links and interference on the web content embedded by adversaries. The distinct features such as textual and visual [38], structural tags, third-party inclusion patterns on web [39] and lexical and string based features of URLs [40] form the basis of classification. Works by Simeonovoski et al. [41] were done on a macroscopic scale that focused on attack simulations and its impacts on prominent countries, companies and Autonomous Systems (ASes) on large-scale Distributed Denial of Service (DDoS) attacks and mass surveillance.

The earliest detection of malicious files across the connections in the web was a signature based approach [42]. The malicious contents on the web can be identified on the basis of certain regular expressions derived from previous studies on malicious codes and hidden links [31]. The various regular expressions such as the ones for Iframe class, CLSID, Shellcode, JavaScript.encode, Long strings, Partial black strings, Document.write frequency, eval() frequency, Unescape (a function that decodes a string encoded by escape() function) frequency, .exe frequency can be used to detect the malicious web pages by employing machine learning. Iframe is an HTML element that embeds a content from third-party domains. CLSIDs are identifiers for alpha-numeric characters that are used for Component Object Model (COM) where COM is an object used for communication between clients and servers. Shellcodes are snippets used as payloads to exploit the vulnerability of an application using stack and heap based buffer overflow. JavaScript.encode is a JavaScript function to encode a Uniform Resource Indicator. Document.write is a function used to directly write data to the browser. Malicious obfuscated JavaScript that are embedded within iframes, eval() or unescape() using multimedia files and scripts can be even detected using static analysis method. Some of the attributes of Iframes that can be used to detect the presence of obfuscated malicious code are malicious src domain, visibility attribute set to false and dimensions of text within it set to less than one [43]. The malicious eval() and unescape() obfuscation attributes are obfuscated snippet, plain-text code injection with invisible or hidden iframe and existence of shellcode [43].

Ma et al. [40] analysed URL(s) (Uniform Resource Locators) of Web sites to detect the malicious ones out of the benign, based on the lexical and host-based features of the URLs using machine learning. Thus the host based features such as WHOIS information, Internet Protocol (IP) properties, AS number, Blacklist, geographic location or information, connection speed, host misc., whereas the lexical features such as host name, primary domain, path tokens, last path tokens, Top Level Domains (TLDs) that are the rightmost tokens present in a domain for e.g. .edu, .uk, .cn and lexical misc. can be used for detection of malicious resources. WHOIS is a protocol used for querying databases that store registration information of domains, IP address block or AS. Any registrar that has multiple malicious domains registered under it is treated as a malicious ownership with the registration date being another feature for classification such that newly registered domains have greater chances of being malicious. The IP properties such as A, MX and NS records are verified to see if they belong to legitimate owners such that their ASes are checked. The IP prefixes (the networking sub-net mask of an IP address) and AS prefixes are checked for malicious sites in the surrounding. The geographical locations such as continent, country and city of the IP address are checked with the geological locations having hotbeds of malicious activities. The lexical feature based classification involve the real-valued features such as the number of dots "." in the URL, length of the host name and length of the complete URL along with the binary features as delimited host name using "." and delimiters as "/", "?", ".", "=", "-" and "_" in the path i.e. URL strings. This method of judging malicious resources based on URLs can be applied for all web elements such as web pages, email, calendar, chat and games without the risks of downloading dangerous contents. Machine learning approach can also be used to withstand the malware obfuscation techniques [44].

Honeypot is a powerful tool [45] to collect extensive information on various behavioural patterns of malware that are evolving more and more with every passing day. This technology helps to identify attacks and detect malware efficiently thus serving as a base for foundation of a model to defend against future attacks. Detection of malicious scripts can be done based on shellcode, overflow behaviour and hidden links using dynamic analysis [46]. The overflow behaviour indicates the behaviour of transferring the control of execution from father process (browser.exe) to Operating System. The overflow attack through overflow behaviour using shellcode can be detected dynamically by decryption of the web page, division of the whole script into a number of pieces so as to extract the shell code and then observing the script flow for larger loops that can cause heap spraying [46]. Recent web-monitoring systems can scrutinize the threats in the web content by collecting malware that accesses the target through both drive-by-downloads and on-click downloads, analyse the activities of the adversaries while they compromised accounts and tried to get fraudulent access [47].

3

Technical Background

This chapter serves the purpose of providing a background for our investigation and analysis. At first, we introduce the concept behind Web rendering that gave us the background for the analysis such that how it works dynamically using some languages that can be used to insert malicious bytes of code. Then we define certain important conceptual terms and move on to the history of some of the malicious attacks by third-parties. Next, we provide the overview of attacks through elements such as JavaScript, Image and CSS. Finally, we provide the various platforms that allows us to accomplish the two methods of analysis.

The websites ranked by Alexa within the top-10,000 are the popular ones [48] as they are frequently browsed by the users. As mentioned in Chapter1, these websites depend on certain third-parties, both explicitly and implicitly and acquire contents from them. The internal and external links of a website are embedded in the 'a' tags [49] and these are visible as hyperlinks in the page. The contents that are obtained from third-parties are originally in the form of hyperlinks of files such as JavaScripts, Cascading Style Sheets (CSS) and multimedia in the original HTML code within the script tag [1]. These contents are embedded as links inside the dynamic HTML tag holders as 'script' tags or 'iframes' while they are rendered by the browser. The rendered code comprises of the accessible content that are embedded as links in the script tags using scripting languages and these contents may have the end-users engage with contents from compromised or malicious sites.

3.1 Website rendering

The process of 'website rendering' involves numerous operations and all these operations revolve around the browser and the browser engine. A browser is simply a piece of software responsible for loading files from a remote server, whereas a browser engine is the key element that assists in displaying the contents based on the files received.

The raw HTML files are converted from bytes to characters, then parsed to tokens (i.e. small units) which are converted to nodes (i.e. separate tag entities) that are linked to form a DOM tree. The classic HTML files are generally CSS linked and the browser requests for the CSS file as soon as it finds the link in the HTML file. The raw bytes in the CSS file are similarly converted to a tree structure called CSSOM. The other important element is the JavaScript that can modify the content and styling of a web-page thus it can modify the DOM and CSSOM tree. Therefore the construction of DOM is halted whenever a script tag is encountered, until the script is completely executed. The JavaScript files are also converted from bytes to characters, tokens and nodes successively. The DOM tree construction gets completed after the execution of the JavaScript in the script tags and the DOM tree together with the CSSOM forms the render tree. The HTML, CSS and JavaScript bytes are turned into rendered pixels on screen as the web page contents are executed [50].

The web content is dynamically rendered using scripting languages such as JavaScript, AJAX and ActionScript. The process of execution involves the execution of content of the external files such as JavaScript, style sheets and multimedia present as hyperlinks inside the script tags [1].

3.2 Defining maliciousness

Any software that violates a system's security policy to cause harm is considered malicious. The extent of damage can range from affecting a single computer to affecting an entire network. These malicious software can be an assortment [51] ranging from plain nuances to harmful violations. A malicious code might seem non-malicious or just suspicious with obfuscation as in the case of JavaScript [26].

3.3 Malicious websites

A malicious website is one that endeavours to install malicious programs that may interrupt the normal functioning of computer, gather personal information, steal credentials or even gain complete control over a network [31]. Installing of a malicious program may require some activity by user such as clicking on a pop up box or hijacked or hijacking advertisement or accepting certain conditions as permission to install a software or codecs (encoder-decoder program for digital data stream or signal). Hijacked ads are the ones that are illegally seized to inject malicious content whereas hijacking ads are the ones that by themselves are malicious. The snippets of malicious codes may be hidden deep within the website code with the help of obfuscation [26].

A website can be compromised through URL injections, malicious code, or link inserted without its knowledge and seem as if it acknowledges a malicious site through direct infection spread or malicious redirects [1, 34]. Thus a malware author might breach the website and alter the code so as to redirect it to a malicious site or spread malwares through the compromised website. These malicious code injections can be identified as cross-site scripting (XSS) [52] or cross-site request forgery (CSRF) [53], and can even lead to browser hijacking where the browser settings are altered without the user's permission. Furthermore, CSRF snippets can exploit the cookies in a user's browser and security permissions for execution of actions on a different website, thus persuading the user carry to out unwanted attacks.

According to the data collected by Kumar et al. [2] in 2016, more than 90% of websites are dependent on contents from their third-parties and two-third of the resources were obtained from their dependencies. Here, crawl refers to the systematic browsing of the web for web data or information, generally used for analyses. The crawled data set by Kumar et al. [2] was analysed to emphasize the security risks due to the dependencies.

3.4 Brief history of malicious third-party content

Third-party contents such as JavaScript, style sheets and multimedia files are intensively used by websites for various functionalities including mashups (web applications that are developed using data and functionality from two or more external sources), advertisements and web widgets. However these functionalities introduce security challenges as they bypass the web server's security measures with the first-party website's permission to host contents thus making the web vulnerable to attacks [3]. For instance, the discovery of RIG Exploit Kit was suspected of data-stealing, was injected on a major trafficked website [54]. An Exploit Kit is a software that is designed to discover the software vulnerabilities in a client machine in order to exploit the vulnerabilities through malicious injections or upload. The Exploit Kit in this case was a fake URL that was added to the script tag of the website on a call to one of it's third-parties. On top of that, RIG and Angular Exploits were found on more than 30 domains in the list of Alexa ranked websites within top 10k thus indicating that the most popular websites were targeted for such attacks. RIG and Angular are two of the several variants Of Exploit Kit.

Ads from third-parties pose a serious security threat that affect a wide range of users through malvertising [1]. According to the study by Google, advertising network makes profit from ad injections along the chain of dependency of websites with the case of over 3000 advertisers being affected by ad injection [55]. These advertisers included some of the important brands such as Sears, Walmart, Target and Ebay.

3.5 Malicious content spread through the web:

Web-based malware have been growing immensely with the ubiquitous nature of the Internet and mobile devices [24]. The dynamic nature of the web content has made it a popular media to escalate infection through social engineering, phishing and infected legitimate sites [56]. Furthermore, the web malware attacks benefit from the vulnerabilities of the browsers or webpages that include the scripting languages and its libraries (for e.g. jQuery, Angular, Handlebars and YUI) included from third-parties [25]. The malicious codes embedded in the script tags spread the infection, with the worst cases being drive-by-downloads initiated by the infected webpages [34].

3.5.1 JavaScript-based attacks:

According to a study in [57], 94.9% of all the websites in the world use JavaScript thus nearly all websites are JavaScript enabled without plugins as they use JavaScript libraries. JavaScript is included in the web page in one of it's dependencies where the dependencies being the included JavaScript files in the form of HTML script tags, and even outside it. Around 87% of these websites rely on JavaScript provided by third-party domains [2]. Thus the first-party websites often use JavaScript from third-parties through both explicit and implicit trust chains [1]. However, the source of JavaScript code/snippet should be analysed or revised well before allowing it on one's website since JavaScript has emerged as a significant threat for the websites [43].

When a third-party JavaScript is loaded to the first-party website's web page it is enabled access to the DOM. There may be snippets of JavaScript code, embedded in the script tags as hyperlinks or simply as the code itself that may be used for various illegitimate activities. Some of these illegal activities include tracing a user's activity on a web page through web scrolling, keystrokes and mouse movements or tracing a user's browsing habits from the browser cookies. This leads to privacy breach and this personal data might be misused.

For some years, an attack tool called Exploit Kit that makes use of JavaScript payloads, has been used immensely to infect web pages and breach user privacy [58]. Further, JavaScript code can be injected for execution of various types of attacks through XSS [52], CSRF [53], or cross-site scripting inclusion (XSSI) [38] thus making it the most preferred language to launch an attack through exploitation of its vulnerabilities. In the case of XSSI, the malicious code is included within the third-party content.

The third-party malicious JavaScript code can also be obfuscated to make it obscure and look like a legitimate piece of code that can evade the normal signature based security checks [43]. Thus JavaScript can be used to perform surreptitious attacks and spread malware across the web through page contents.

3.5.2 Image based attacks using web content:

Image files on the web can be used to spread malware through an exploit called Stegosploit where malicious JavaScript code can be hidden within the image code. The technique hides the malicious snippet within a digital image pixel which is automatically activated when the image is loaded. It is a subtle way of causing damage through seemingly non-harmful objects and can cause any kind of damage through the combination of image and JavaScript called IMAJS. The first-parties that trust on image contents at different levels in the trust chain of dependency can be easily tricked into attacks through image files.

3.5.3 Attacks based on CSS:

Nowadays, almost every modern website relies massively on CSS in order to develop an attractive interface for the users and a large number of websites use CSS resources from third-parties to further decorate their content display and looks. However, CSS files can be used for XSS injections and load malicious external scripts [59].

In 2012, Mario Heiderich et al. revealed that CSS along with other web technologies, such as inactive SVG images, font files, HTTP requests and simple inactive HTML, can enable an attacker to perform malicious activities by achieving a partial JavaScript like behaviour [28]. More recently, Mike Gualtieri's 2018 analyses on CSS have accentuated certain vulnerabilities that can be exploited to steal sensitive data or credentials, from one's machine when connected through Internet [60]. The CSS based attack is named CSS Exfil thus indicating CSS Exfiltration.

CSS can also be used to steal CSRF tokens (they help prevent CSRF attacks) within ten seconds and persuade a user to carry out unwanted activities [28]. Thus CSS can be used to cause information leakage through a brute-force attack for illegitimate use and other malicious activities.

3.5.4 Malicious PHP script based attacks:

PHP is a dynamic programming language used for both server and client side scripting on websites. Thus it is another dynamic platform that can be used for injections malicious code into web content and cause attacks such as redirects, stealing of personal data, revealing confidential information and Denial-of-Service (DOS)[29, 61, 62].

3.5.5 Attacks based on other web resources:

There are various other forms of attacks through Flash ActionScripts, audios, videos and software executables (as .exe files). ActionScript is a programming language primarily used for web software development, specifically Adobe Flash Player [63]. Flash ActionScript attacks can be carried out through heap spraying (an exploit technique that facilitates code execution), JIT spraying (code that circumvents the protection of address space layout randomization and data execution prevention) and type confusion exploitation (code that prevents from memory object verification) [63]. On the other hand, malicious snippets can be embedded to video or audio files and used as exploits [64, 65]. Sometimes, users may be tricked into clicking on a video or audio file that might be a malicious software executable with the file's original extensions being hidden.

3.6 Analysis Methodologies

Two types of analyses can be employed for inspecting malicious contents on the web: static and behavioural [59]. Static analysis reports a malware without executing the malware whereas dynamic analysis executes the malware in a controlled and monitored environment to examine its behaviour. Another method for analysis is 'Blacklisting' where the domains are checked against blacklisted malicious domains, however this is not a feasible one [66].

3.6.1 Static Analysis

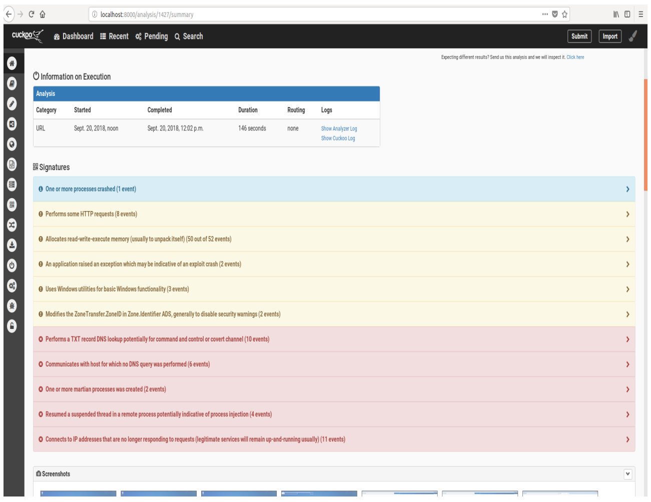
Basic static analyses examines malware without executing its actual code or instructions that are embedded in a file or a URL. It is a signature based method where signatures of the malicious HTML and JavaScript are matched against the web content codes. It uses certain features such as file type, file size, MD5 checksums or hashes and antiviruses or scanners to determine whether a snippet or a code would perform malicious activity. Such analyses can be done with multiple scanning engines or even online tools such as VirusTotal [18], NoDistribute [67], Jotti [68], MetaDefender [69], MASTIFF [70] and Crytam [71] that scans a file or URL.

3.6.2 Dynamic Analysis

On the other hand, dynamic analysis executes the malware program to examine its behaviour, interprets its functionality and creates run-time traces. The features of a malware program may consist of domain names, IP addresses, registry keys and additional files located on the system or network. Registry key is an internal database within a computer that is used to store information regarding its own hardware or software configuration. Dynamic analysis can even establish whether there are external servers that are controlled by attackers for directions, control operations or downloading of malware files [59]. Previously such analyses were manually performed by malware analysts using various tools. Today, these approaches have been replaced with commercial and open-source sandboxing environments [72]. Sandboxing malwares is an empirical approach towards applying the dynamic or behavioural analysis instead of just analysing the binary files statically. Some of the dynamic analysing tools are Cuckoo Sandbox [19], Joe Sandbox [73], Detuxsandbox [74] and SEE (Sandbox Execution Environment) by F-Secure [75].

Online tools can give away to the malware authors that someone is examining their malware so hashing can be performed on the file or URL body-content that serves as the impression of the file or URL to be scanned. Hashing in terms of cryptography as meant here, is a method of representing data in an irreversible-string format that seem to be randomly put together, with the idea of storing data in a secured manner [76].

However, in the context of malware analysis or sharing reports on some content (i.e. files, URLs or domains), hashing is performed by scanning tools to uniquely identify the content. It can be done using a couple of hashing algorithms such as s MD5, SHA-1 and SHA-256, SHA-512 and Skein. The effectiveness of the most popular hashing algorithm, MD5 has been increased with fuzzy hashing where fuzzy hashing is an efficient way to identify hashes of files with high percentage of



Expecting different results? Send us this analysis and we will inspect it. Click here

Information on Execution

Analysis					
Category	Started	Completed	Duration	Routing	Logs
URL	Sept. 20, 2018, noon	Sept. 20, 2018, 12:02 p.m.	146 seconds	none	Show Analyzer Log Show Cuckoo Log

Signatures

- One or more processes crashed (1 event)
- Performs some HTTP requests (8 events)
- Allocates read-write-execute memory (usually to unpack itself) (50 out of 52 events)
- An application raised an exception which may be indicative of an exploit crash (2 events)
- Uses Windows utilities for basic Windows functionality (3 events)
- Modifies the ZoneTransfer.ZoneID in Zone.Identifier ADS, generally to disable security warnings (2 events)
- Performs a TXT record DNS lookup potentially for command and control or covert channel (10 events)
- Communicates with host for which no DNS query was performed (6 events)
- One or more martian processes was created (2 events)
- Resumed a suspended thread in a remote process potentially indicative of process injection (4 events)
- Connects to IP addresses that are no longer responding to requests (legitimate services will remain up-and-running usually) (11 events)

Screenshots

FIGURE 3.1: Cuckoo interface

similarities [77].

VirusTotal is one such scanner that uses MD5, SHA-1 and SHA-256 among the hashing algorithms by creating a unique identity for each file, URL or domain scanned. Currently, it even uses ssdeep, a program for computing triggered piecewise hashes also known as fuzzy hashes [77].

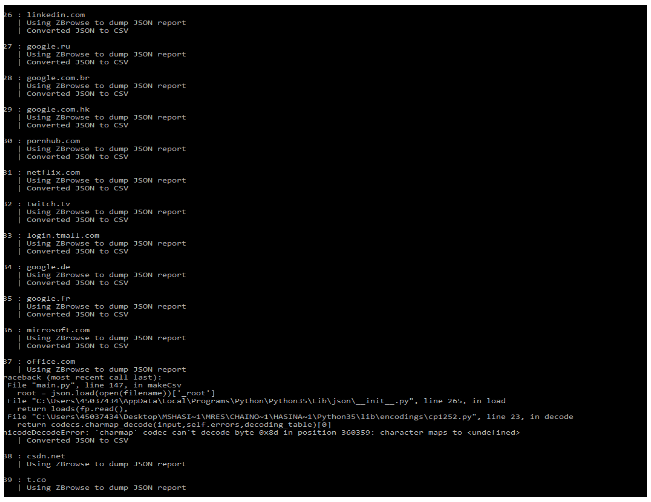
Cuckoo Sandbox [19] is another analysing environment that uses MD5 with ssdeep being used for performing hashing and fuzzy hashing respectively on URLs and files. It is an open-source Sandboxing tool that performs analysis by execution of files and URLs in an isolated-environment. Cuckoo uses various software tools such as tcpdump [78], Volatility [79] and Mycrypto [80] . Tcpdump is used to display TCP/IP and other packets transmitted across the network thus serves traffic analysis [78]. Volatility is a forensic framework used for analysis of malware programs. Mycrypto is for OpenSSL features such as RSA, DA, DH, EC and HMACs where OpenSSL itself is a software library for secure communications over a network using TLS and SSL protocols [80]. Cuckoo consists of a software that centrally operates the execution and analysis of files and URLs. Cuckoo's principal architecture is based on a host-guest environmental set up. The host is the sink (one that collects all data obtained after analysis) that starts the analysis tasks and manages them whereas the guest (a virtual machine configured for malware analysis) performs the analysis of contents, dumps the traffic, generates reports and transfers them to the host. The guest has a clean environment to run the analysis of samples and reports back to the host [19].

4 Experimental Evaluation

This chapter reports our data-set which were used for the experimental evaluations. Moreover, we discuss the methodology for analyzing the attacks mentioned in the previous chapter. Finally, we present the experimental settings and our experimental results. Furthermore, we have provided the background of our data (i.e. how we generated and the technologies involved in). We provide the statistical analysis of our experimental results.

4.1 Data-set

We had started with crawling the Alexa ranked websites in order to create our own data-set. Crawling means visting the websites in an automated manner. The notion behind creating our own data-set was that the web is a dynamic platform and it keeps on evolving with time as new dependencies (i.e. third-parties in the explicit and implicit chain) are introduced and at times older



```
26 : linkedin.com
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

27 : google.ru
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

28 : google.com.br
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

29 : google.com.hk
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

30 : pornhub.com
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

31 : netflix.com
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

32 : twitch.tv
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

33 : login.tmall.com
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

34 : google.de
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

35 : google.fr
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

36 : microsoft.com
   | Using ZBrowse to dump JSON report
   | Converted JSON to CSV

37 : office.com
   | Using ZBrowse to dump JSON report
raceback (most recent call last):
 File "main.py", line 147, in makeCsv
   root = json.load(open(filename))['_root']
 File "C:\Users\45037434\AppData\Local\Programs\Python\Python35\Lib\json\__init__.py", line 265, in load
   return loads(fp.read(),
 File "C:\Users\45037434\Desktop\MSHASI~1\MRES\CHAINO~1\HASINA~1\Python35\lib\encodings\cp1252.py", line 23, in decode
   return codecs.charmap_decode(input,self.errors,decoding_table)[0]
nicodeDecodeError: 'charmap' codec can't decode byte 0x8d in position 360359: character maps to <undefined>
   | Converted JSON to CSV

38 : csdn.net
   | Using ZBrowse to dump JSON report

39 : t.co
   | Using ZBrowse to dump JSON report
```

FIGURE 4.1: Crawl Screenshot

ones being removed. We had started the crawl on August 16, 2018 and could crawl 30K websites. Thus, our data-set comprises of the resource dependency tree of the top 30K Alexa ranked websites' main pages where the dependency tree is abstracted from the relationship between the first and third-party domains. Each resource dependency tree has a first-party website as its root node and the rest of the nodes represent the third-party websites that involve the external resources obtained both explicitly and implicitly. The concept of resource dependencies of websites is acquired from the work by Kumar et al. [2]. These dependencies are obtained in the form of URLs extracted from the network data of websites between August 16 and October 9, 2018 as Figure 5.1.

The data-set is the result of crawled data using a crawler called Zbrowse [81] with the help of headless Google Chrome browser [82] and Nodejs [83], in the form of DOM Trees. Z-browse [81] crawler released by Kumar et al [2] collects network data from the rendered main web pages of the websites. We use Google Chrome in headless mode where the headless mode allows it to render a given website and track the resource dependencies by recording the network requests sent to (resp. received from) third-party domains and Nodejs that provides the run-time environment for JavaScript outside a web browser. These requests are made through URL calls which are used to construct the dependency tree between the first party website and the third-party providers. We had further written a python script to automate the process of the crawling 30K websites with more efficiency to retrieve the resource dependencies.

A file related to a specific website for each of the top 30k websites is obtained. These files contain resource dependencies in the form of third-party URLS with their domain names, levels, protocols, IP addresses, URL types such as image, JavaScript, TEXT, HTML, GIF, Application, CSS or x-JavaScript. The levels so obtained in these files are as 1, 2, 3 or greater and are indicative of the resources being direct (obtained explicitly) or nested (obtained implicitly). Each of these files have dependency tree URLs ranging from 2 to greater than 1000 with the average standing at 500 URLs per file.

We store these information on relationships between nested resource dependencies for our study on malicious web contents by third-parties. We do so by forming a tree representation, unlike prior web resource studies [14, 84, 85] which used a bipartite graph model of web pages.

4.1.1 Data-set Refinement:

We perform the refinement of the data-set of resource dependencies for the top-30K websites. Here the top - 30K refers to those websites from which a resource dependency tree could be extracted. The decisive data-set obtained, involves the Alexa ranking from 1 (google.com) to 30,000 (naointendo.com.br). We eliminated those websites from our data-set of top - 30K that were unresponsive and resulted into empty files when trying to connect using the crawler and our scripts, thus precluding them from further analysis. We found that there are 7335 unresponsive first-party websites in our data-set of top-30K, leading to a number of 22665 websites. We applied the same approach for the third-party websites obtained as dependencies of these first-party websites in our data-set. We exclude the redundant URLs in each dependency tree from the data-set with the aim of having a clearer estimation of the threats caused by the external resources and thus enhancing the data-set such that there would not be any repetitive analysis.

Table 5.2 shows that the first-party websites rely on a large number of external resources along. The table also gives the percentage of external resources in the total resource count of 3,389,093 (combined internal and external resource count). It further provides a fine-grained analysis of the resource count and the percentage of resources served to these websites by their direct and indirect

Level	1-30K	1-10K	(10-20)K	(20-30)K
0	907453 (21.12%)	236008 (22.06%)	334431 (28.55%)	337014(29.35%)
1	2213144 (51.51%)	739590 (69.16%)	754210 (64.38%)	719344 (62.65%)
>= 2	268496 (6.25%)	93851 (8.78%)	82803(7.07%)	91842 (8.00%)

TABLE 4.1: Internal and External resource count of 1-30K Alexa ranked websites in different levels.

dependencies. We divide the external resource count and their percentage according to levels and ranks. These resources are categorized into levels (1 and >= 2) and ranks into three parts as first-parties within 1-10k, (10-20)k and (20-30)k such that each part comprises of 10,000 first party websites. It is done so that we can have a clearer understanding of the external resources for each segment of the data-set. We observe that the first-parties in each of the data segments heavily depend on their third-parties for resources. There are 24,81,640 external resources, out which 22,13,144 are obtained explicitly and 2,68,496 implicitly. Thus 51.51% of the total resources are obtained from their direct third-parties while 6.25% of the total resources are from the implicit dependencies (i.e. indirect third-parties). The table also provides us with the visibility of the top first 10k depending more on external resources than the second (i.e. (10-20)k) and third (i.e. (20-30)k) set with 77.94%, 71.45% and 70.65% external resources respectively.

Level	Tottal Resource	JavaScript	Images	css	html/xml
1	739590 (69.16%)	155233 (18.63%)	327712(39.32%)	31924(3.83%)	36351 (4.36%)
2	70787 (6.62%)	5724 (0.69%)	26054(3.13%)	258(0.03%)	11572(1.39%
3	21294 (1.99%)	853 (0.10%)	3734 (0.45%)	34 (0.00%)	3712 (0.45%)
>= 4	1770 (0.17%)	114 (0.01%)	710(0.09%)	4 (0.00%)	231(0.06%)

TABLE 4.2: Internal and External resource count of 1st 10K Alexa ranked websites in different levels.

Table 5.3, 5.4 and 5.5 illustrate various external resource types in different levels (i.e. explicit

Level	Total Resource	JavaScript	Images	css	html/xml
1	754210 (64.38%)	153595 (18.35%)	341271 (40.77%)	34831(4.16%)	37585 (4.49%)
2	58228 (4.97%)	4191 (0.50%)	23392 (2.79%)	187 (0.02%)	6577 (0.79%)
3	22550 (1.92%)	628 (0.08%)	4299 (0.51%)	45 (0.0%)	3392(0.41%)
>= 4	2025 (0.17%)	53 (0.01%)	415 (0.05%)	0 (0.0%)	226 (0.14%)

TABLE 4.3: Internal and External resource count of 2nd 10K Alexa ranked websites in different levels.

Level	Total resource	JavaScript	Images	css	html/xml
1	719344 (62.65%)	147044 (18.13%)	316376 (39.00%)	36006 (4.44%)	36325 (4.48%)
2	58055(5.06%)	4603 (0.57%)	22553 (2.78%)	174 (0.02%)	6584 (0.81%)
3	30351 (2.64%)	828 (0.10%)	4661 (0.57%)	19 (0.00%)	4579 (0.56%)
>= 4	3436(0.30%)	70 (0.01%)	1057 (0.13%)	0 (0.00%)	281 (0.03%)

TABLE 4.4: Internal and External resource count of 3rd 10K Alexa ranked websites in different levels.

and implicit) of the chain of trust for the top first 10k, second 10k and third 10k respectively. The types of resources that are commonly obtained from the third-parties in large numbers which are JavaScript, image and CSS files. Thus we have categorised the external resource types as JavaScript, image and CSS and the rest of the resource types such as Flash Object, PHP and executable (as exe) into another category. The above mentioned tables show the external resource type count in each of the levels (i.e. 1,2,3 and >=4) along with their respective percentages. It is transparent from the table of top first 10k, second 10k and third 10k that the first-party websites rely mostly on third-parties for image followed by JavaScripts further followed by html/xml and CSS files. The external JavaScript type resource count in the top first 10k, second 10k and third 10k are considerable in level 1 with image being the highest number of resources. Both the type of resource decrease in count as higher levels of dependency are reached.

Though we crawled the top-30K Alexa websites for their corresponding dependencies, we could not utilize the complete data-set because of the lack of time. Therefore we extracted a part of the data-set within the top-1000 websites. The extracted data-set comprises of 872 websites, while the rest of the websites among the top-1000 remained unresponsive.

4.2 Methodology

We performed static and dynamic analyses of malicious contents provided by the third-parties. We aimed for both the methods of analysis since static analysis would readily provide us with the known malicious signatures that would confirm the existence of malicious contents while the dynamic analysis allows us to discover new contents with malicious or suspicious behaviour. Moreover dynamic analysis would let us execute the dynamic web content i.e. JavaScripts extensively. Since we were interested in the analytic, we had chosen VirusTotal and Cuckoo as the tools for analysing domains and URLs that were obtained from the crawl results.

VirusTotal is an online scanning and threat analyzing intelligent service platform for malicious programs or codes. It uses a collection of more than 70 anti-virus engines (as of August 16, 2018) and provides maliciousness score based on how many engines have flagged an URL or file as malicious.

Though VirusTotal can be used for static analysis on URLs and domains, it is not completely a static platform since it uses some dynamic engines as well for behavioural analysis and that gives us a reason for choosing VirusTotal among other online scanners. Upon submitting an URL or domain, it provides a list of scans from different anti-virus software. This functionality makes VirusTotal popular among victims of targeted attacks as they generally have a low detection rate by an individual anti-virus software. Behind the scenes, Anti-virus (AV) vendors who cannot not detect a file that is detected by other AV that are given its payload, providing them with the visibility into attacks that may otherwise go unnoticed by that specific vendor. To facilitate sharing with AV vendors and other members of the security community, VirusTotal implements a rich API that provides users to access to the uploaded content and its meta-data.

We used the VirusTotal API for the domain and URL reports separately for each website. The requests for reports on the domains submitted for scanning, returns the scan reports that consists of a number of Anti-virus tools which flagged the third-party websites as malicious. Thus the number of positives obtained from VirusTotal can be defined as the number of anti-virus engines marking it as malicious which is also known as the VT score. The Anti-virus reports also contain meta information such as the first scan date, scan history, domain name resolution, categories as bitdef, drweb and Websense, verdict on a domain as safe or unsafe, whether it has adult contents, number of positives and the WHOIS records. The information obtained on each of the submitted domains, helped us decide its malicious factor.

We further identify the types of malware at each level for a particular first-party website. Thus to whether a malicious content of an URL obtained through explicit or implicit trust from a third-party website contains malware code of an adware, spyware or rootkit or does the URL inserted belong to a phishing site. We used the VirusTotal API for each URL of our data-set. The VirusTotal API provided the scanned reports having the various categories of malware programs identified by various anti-virus tools and phishing site scanning engines.

We also made queries to the PhishTank API for checking if the URLs belong to any phishing site with the aim of verifying the VirusTotal response on phishing sites obtained from tools such as OpenPhish, PhisTank and PhishLab. This would confirm any site marked as a phishing site as the threat to credentials that has been increasing tremendously.

Furthermore, we analysed the dependencies of each first-party website by feeding them to the sandboxing environment of Cuckoo. We chose a sandbox for the further analysis since a sandbox is a security mechanism for executing untested codes from unverified third-parties, websites or users.

Moreover, Cuckoo is an open-source automated malware analysis system that generates a report based on the behaviour of an URL or file by executing it inside an isolated but realistic environment. It uses an ‘isolated operating system’ to analyse files or URLs and gather comprehensive results, where ‘isolated operating system’ indicates a virtual machine set up on an isolated or host-only environment.

The reason for choosing a sandbox separately is to perform behavioural analysis of the URLs extensively since with such an analysis, the code would be executed and observed, thus an obfuscated malicious code or lines of JavaScript can be appropriately detected. Moreover, we chose Cuckoo for the purpose of finding the malicious URLs since it gives a reason for marking an URL as malicious after dynamic analysis on an extensive scale. We interpreted the results based on the scores and signatures obtained after analysis of each URL. The scores obtained are on the scale of 10 and the signatures such as 'performs some HTTP requests', 'allocates read-write-execute memory', 'uses Windows utilities for basic functionality', 'communication of host for which no DNS query was performed' and 'connects to IP addresses that no longer respond to requests' are returned. The signatures help us to interpret the URLs for e.g. HTTP requests made to URLs are looked into from blacklists to make sure if they do belong to such a list or not, the signature that declares dead host IPs are looked into, to understand if they are active or not since these URLs have high chance of becoming dangling records in the DNS where dangling records are those records in the DNS which are no longer in use by their registered owners.

4.3 Evaluation

As we crawled the top-30k Alexa websites, we acquired various information such as their IP address, their own content URLs (i.e. level 0 resources), content URLs obtained from their third-parties (i.e. explicit and dependencies), the corresponding trust levels (i.e. 1, 2, 3 or more) of these third-parties and the type of the content (i.e. JavaScript, images, CSS, text and HTML etc.) obtained. We thus plot the resource count (i.e. resources at level 0 and >=1), resource count at various levels and count of the various resource types of the external resources using the information in our crawled data-set. We further plot various categories (such as Business, IT, Ads, Education and Social Networks) from where the resources are obtained by these first-party websites. In this section, we plot our findings that are based on the top-1000 websites and not the top-30k since we could analyze only the top-1000 websites. We carry out an in-depth aggregate statistics over the data to establish comprehensive results. Figure 5.2 shows a CDF of the percentage of first-party websites and their total, internal and external resource count. It illustrates statistical summary of the external resources acquired by the first-party websites from their third-parties. The CDF shows how with the rise in the percentage of first-party websites, their external resource count

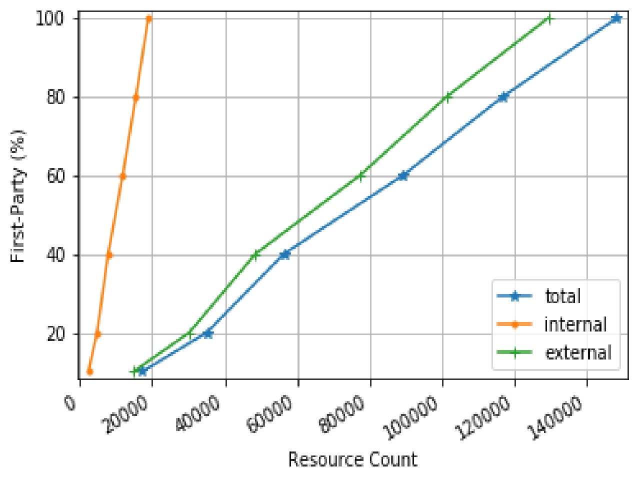


FIGURE 4.2: CDF of the percentage of First-party and obtained resources

increases such that the resources (explicit and implicit) acquired from its third-parties is massive. The total resource count of the top-1000 Alexa ranked first-party websites is 1,48,200 out of which 1,29,249 are external resources thus constituting of 87.21% of the total resources. This leads us to emphasize on the fact how the first-parties depend on their third-parties. Thus it makes the tangled state of the web, apparent to us.

We then obtained the resource count of the first-party websites at various levels of the chain of trust with the help of the information obtained from our crawled data-set. Figure 5.3 is a CDF based on the resource count of the first-parties in different levels. It is visible that the first-party websites acquire external resources at levels > 1 (i.e. 2,3,>=4). Thus a certain percentage of external resources is acquired implicitly apart from the explicit ones (i.e. level 1 resources). It thus establishes the fact that apart from trusting on direct third-parties, the first-parties also trust on some third-parties unknowingly as well since all third-parties do not reveal their dependencies and the resources acquired from these dependencies to the first-parties (i.e. whom they provide these resources).The implicit resources form 10.55% of the total resources which is a considerable amount, with the number of website resources standing to 15,636.

Moreover, we obtained the count of the different external resource types (i.e. JavaScript, image, CSS, HTML/XML and others such as flash objects, multimedia files and executable as .exe). We

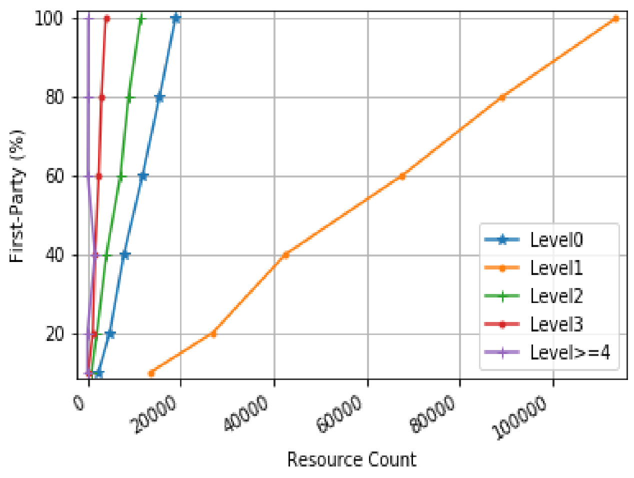


FIGURE 4.3: CDF of the external resources in different levels

Level	JavaScript	Images	css	html/xml
1	23065 (17.85%)	54599 (42.24%)	4297 (3.32%)	5210 (4.03%)
2	862 (0.67%)	3665 (2.84%)	28 (0.02%)	2105 (1.63%)
3	118 (0.09%)	697 (0.54%)	8 (0.01%)	696 (0.54%)
>= 4	8 (0.01%)	71 (0.05%)	2 (0.00%)	60 (0.05%)

TABLE 4.5: External resource count according to resource types in different levels.

can clearly derive from Table 5.5 and Figure 5.4 that the first-party websites obtain a considerable number of JavaScript files that form the third-party of external resources in level 1 (i.e. resources obtained explicitly) since it constitutes of about 17.85% of the total external resources (i.e. all the types of resources considered across all levels) but hardly occurs in dependency levels >=2 with 0.77%. On the other hand, images form a major part of external resources with it alone constituting 42.24% and 3.34% in level 1 and levels >=2 respectively. The other resources such as CSS, HTML/XML, PHP, flash objects and multimedia files constitute lower percentage in the external resource count. These mostly occur in the higher levels of dependency (i.e. >2).

We also obtained the various categories (i.e. IT, Business, search engines, Ads, Education, Social Network) of the third-parties from where the resources have been acquired by the first-parties at different levels using the VirusTotal API for domains. Among the various categories provided

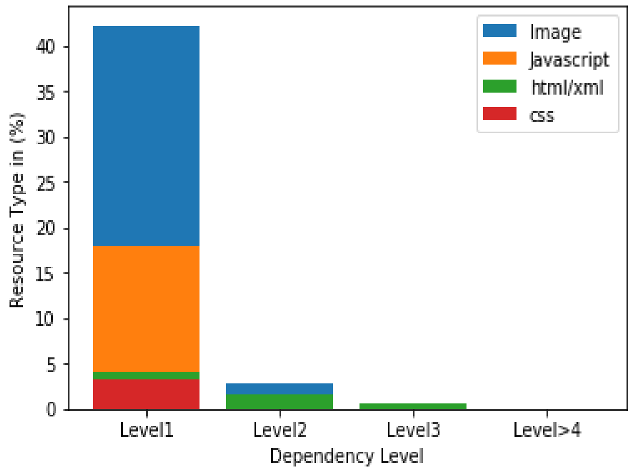


FIGURE 4.4: Percent of the different external resource types in different levels

by VirusTotal, we rely on the Websense category to decide the category of resources. VirusTotal's Websense helps to derive the category information for all resource's domains thus we treat them as unresponsive category. We tried to visualize the percentage of external resources obtained from various categories at different dependency levels and Table 5.6 helps us view our evaluations.

Figure 5.5 shows that fist-parties obtain the highest number of resources from search engines at level 1. A considerable amount of external resources are obtained from business category both explicitly and implicitly (i.e.levels $>$ 1 resources). The other categories such as search engines, social networks, IT and Ads serve resources explicitly with social networks and IT serving resources implicitly as well, as visible from the graph. We also derive that the first-party websites rely for resources on the ads category from their direct third-parties only and thus Ad resources are not obtained implicitly.

We now discuss our experimental results on malicious contents acquired from third-parties at different levels. In order to do so we used the VirusTotal API for domains. Initially, we started with the VirusTotal Public API that had a request rate of 4 posts per minute and later moved on to acquiring the Academic API that had a request rate of 20,000 requests per day since use of Public API consumed a lot of time for our analysis as each of the Alexa ranked websites had a list of 500 URLs on an average that makes an average of nearly 148 domains.

Level	Search Engines	Ads	Business	IT	Education	Social	Porn
1	30.16%	10.00%	15.92%	1.02%	0.13%	2.10%	0.00%
2	2.13%	0.24%	3.41%	0.00%	0.00%	0.54%	0.00%
3	0.00%	0.08%	0.65%	0.19%	0.00%	0.00%	0.00%
>= 4	0.01%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

TABLE 4.6: External resource count according to resource types in different levels.

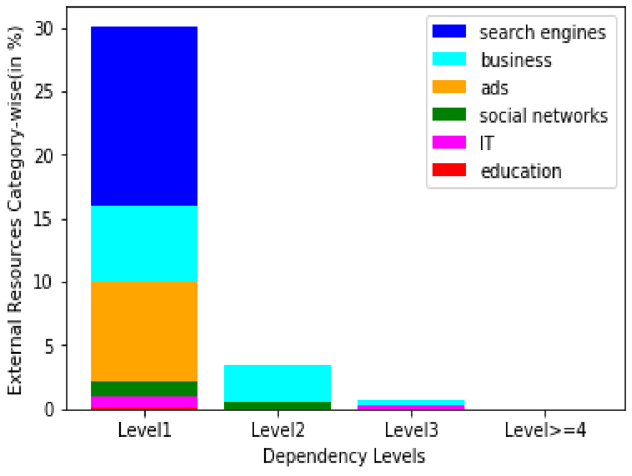


FIGURE 4.5: External Resources from different categories in different levels

We obtained the malicious scores on the explicit and implicit third-party resources using the API. The VirusTotal (VT) Score ranges between 1 to 65 for the malicious resources. We found that out of 129,249 overall external resources, 62,420 are malicious that constitutes 47.99% of external resources with 43.89% constituting the level 1 resources and 4.1% being from levels >=2. The graphs in the Figures 5.6, 5.7, 5.8 and 5.9 are the result of evaluation of the percentage of different VT Score ranges based on the total number of malicious resources (i.e. 62420).

The CDF in Figure 5.6 and the data in Table 5.7 shows the percentage of malicious external resources with VT Scores in different ranges for the different levels (i.e. 1, >=2). We define malicious resources with VT Score within the range of 1 to 9 as simply malicious, the ones with VT Score within 10 to 49 as potentially malicious and those with VT Score >=50 as dangerously

VT Score	level 1	level >= 2
<= 5	82.11%	8.14%
> 5 and <= 10	2.20%	0.31%
> 10 and <= 20	2.66%	0.46%
> 20 and <= 30	1.54%	0.13%
> 30 and <= 40	1.73%	0.05%
> 40 and <= 50	0.60%	0.00%
> 50 and <= 65	0.07%	0.00%

TABLE 4.7: Percentage of resources in different VT Score ranges.

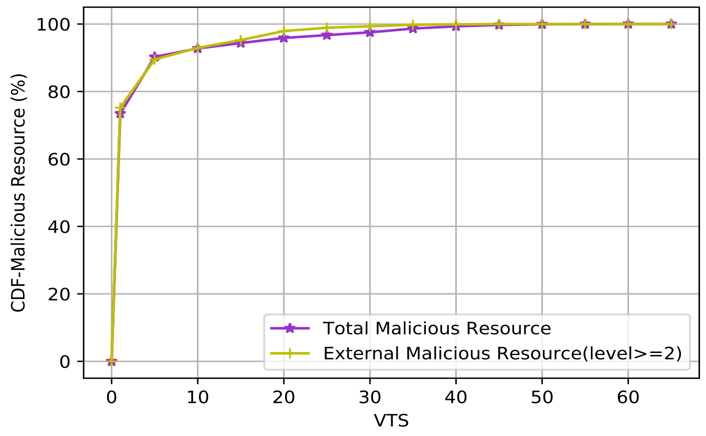


FIGURE 4.6: CDF of malicious external resources in different levels

malicious. It is visible from the the table as well as the figure that level 1 resources has the maximum percentage of VT score for the range of >= 5. We evaluated that 90.91% of malicious resources are acquired explicitly and 9.09% implicitly. These 90.91% resources are loaded to 83.70% first-party websites explicitly and 9.09% being loaded to 1.86% first-parties implicitly. These malicious resources however constitute the simply, potentially as well as the dangerously malicious ones. We further analyse through our measurement that 33.73% websites load potentially malicious resources from their direct third-parties.

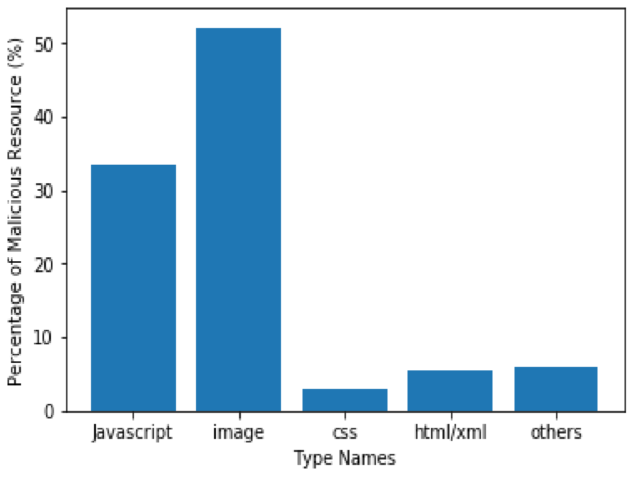


FIGURE 4.7: Percent of malicious external Resources Types

In order to visualize the malicious resource types so as to understand the nature of malicious resources, we calculated the percentage of malicious resources belonging to the four major types of resources (i.e. JavaScript, images, CSS and html/xml). Figure 5.7 clearly illustrates the fact that images and JavaScript files constitute the major percentage of malicious resources. According to our findings, the image files constitute a major part of external resources in the dependency chain and the Figure 5.6 emphasizes on how the once seemingly benign resource type can be highly precarious with it alone constituting 52.72% of the total malicious resources. It also brings into notice that the CSS files can be malicious too according to the recent emphasized revelations by researches [60] with our analysis clearly proving it. Since javaScript files are the most dangerously malicious form [60] of resource and they have considerable amount (i.e. 18.87%) of occurrence in the malicious resource count according to our analysis, it is a matter of concern.

We then moved on to the observation of the VT Scores for each of the four resource types (i.e. JavaScript, images, CSS and html/xml). We observed that JavaScript, image and HTML/XML files occur in almost all ranges of VT Scores (i.e. within 1 to 40) while the resource types that have very high VT Scores are JavaScript and image as plotted in Figure 5.8. 96.40% of the total malicious resources have VT Score <=5 while 6.03% are potentially malicious with high VT Score of >=10. We found that JavaScript and image files constitute the resources having very high VT Scores with the maximum number of resources being JavaScript files. We further observe that

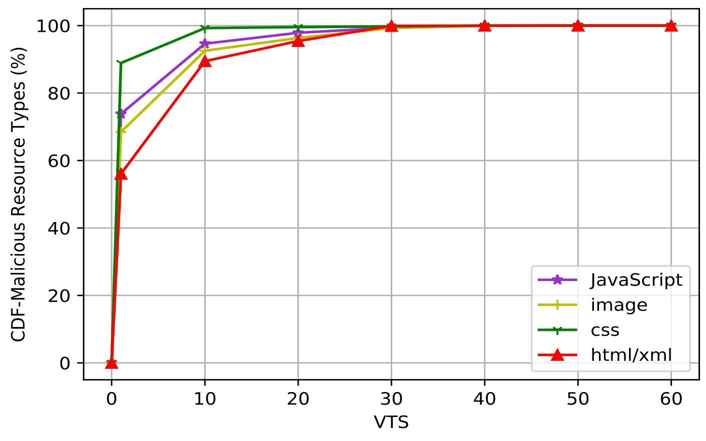


FIGURE 4.8: CDF of malicious external resources types

0.02% JavaScript and 0.01% image files have VT Scores more then 50 indicating that they are dangerously malicious. Thus we can establish that the image files from external sources is an emerging threat.

Furthermore, as we evaluated the resource types in various levels, we observed that JavaScript,

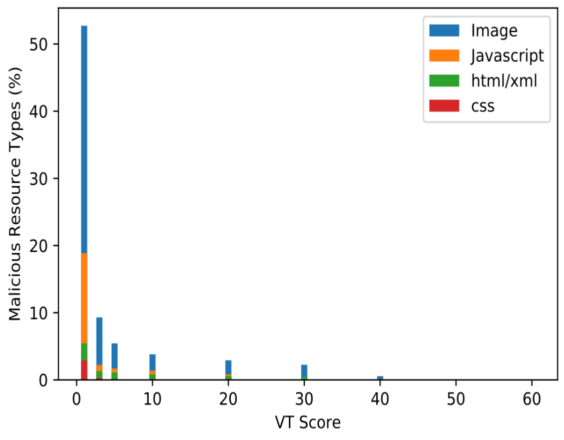


FIGURE 4.9: VT scores of different resource types in level 1

image, CSS and html/xml do not form the major malicious resource types since they occur only in level 1 as shown in Figure 5.9. These four resource types do not form the majority of malicious resources in level 2, thus we can derive that these resource types are not the ones noticeably responsible for the implicit malicious resources. The resources that are thus responsible are among resource types such as flash objects, multimedia files (as video, audio), PHP and executable (as .exe).

We found that some first-party websites namely springer.com, usbank.com and okta.com load some

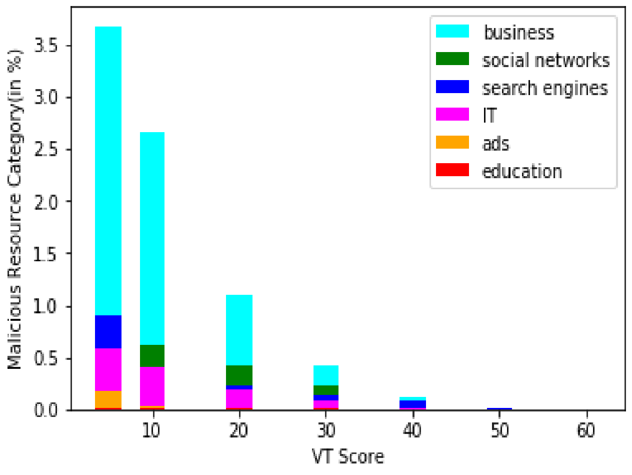


FIGURE 4.10: VT scores in different categories

dangerously malicious JavaScript files from their third-party website, s3.amazonaws.com while the first-party website namely avito.ru loads dangerously malicious image files from their third-party website, ysa-static.passport.yandex.ru. The first-parties load content from their direct third-parties thus they acquire malicious contents from level 1 resources. Since these malicious third-party websites have very high VT Scores of >50 we reckon that they are dangerously malicious.

We also analysed the categories (i.e. business, ads, IT, social networks and Education) of websites that load malicious content to the first-party websites. In order to acquire visibility on the category that loads maximum number of malicious resources out of all resources, we carry out the evaluation of malicious external resources according to their categories on the overall resources (internal as well as external). The plots in Figure 5.10, 5.11, 5.12 and 5.13 are based on the overall resources with the VT Scores within the range of 5 to 60. In Figure 5.10, we had plotted the malicious resource category based on the percentage of total resource count so as to evaluate the percentage standing of malicious categories. We observed that though each category form a part of one or

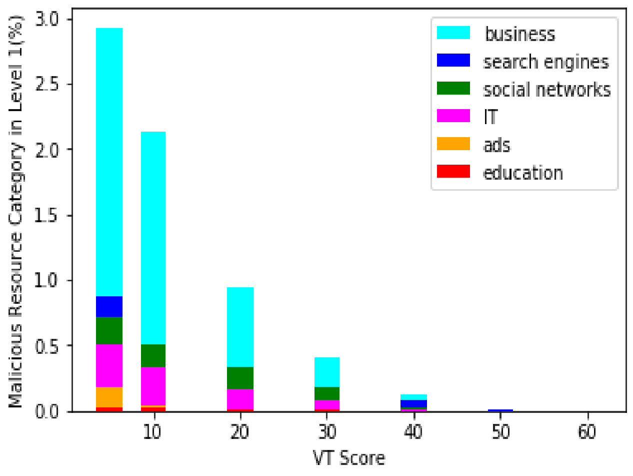


FIGURE 4.11: VT scores in level 1 for different categories

more VT Scores, a considerable amount of resources imported from the business category are responsible for malware distribution such that the malicious content obtained from this category occurs for all the above mentioned range of VT Scores. Thus it indicates that the maximum percent of malicious resources are obtained from the business category and they are the source of simply malicious to potentially malicious resources.

Then we analysed the categories of malicious resources based on levels so as to understand which

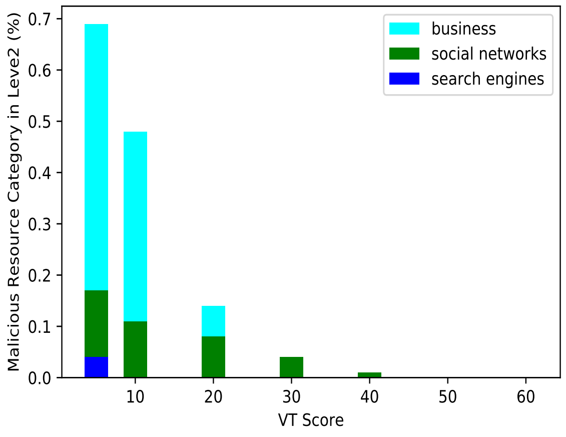


FIGURE 4.12: VT scores in level 2 for different categories

categories have the adverse affect on the first-party websites and whether they are explicit or implicit. Fig 5.11 represents the malicious resource categories in level 1. The resources obtained from this level has all the categories for VT Score <=5, while only search engines are there for very high VT Scores of > 50.

Figure 5.12 is plotted based on the level 2 resources. The graph displays how the malicious

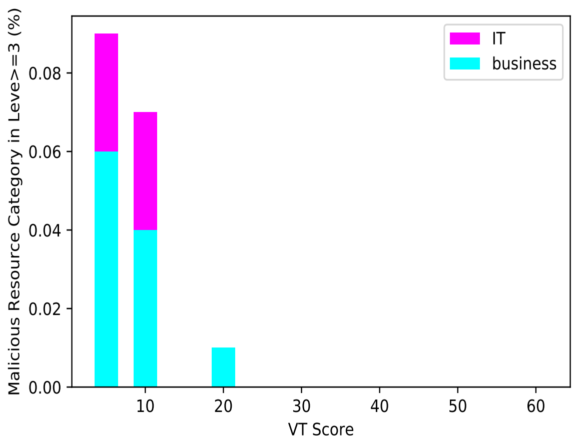


FIGURE 4.13: VT scores in levels>=3 for different categories

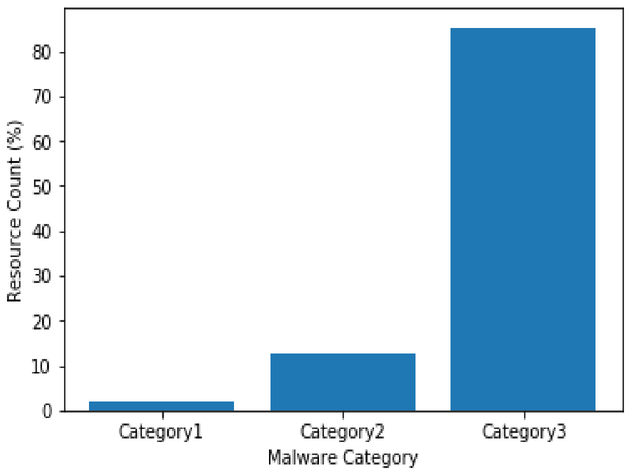


FIGURE 4.14: percent of various classes of malware

resources from categories of business and social networks occurs in VT Scores >=20 and >=40 respectively in level 2. This indicates that level 2 (implicit resources) has potentially and dangerously malicious resources from business and social networks respectively. However, business category has the maximum percentage of malicious resources in level 2 thus they appear to be in the potentially malicious category that form wide-spread threats.

Figure 5.13 is based on the malicious resource percentage according to various categories and their VT Scores in levels >=3. In these levels, resources from business and IT are potentially malicious with the maximum percent of malicious resources belonging to the IT category. Thus we establish that all levels have the business category as the common potentially malicious category.

Finally, we have included the analysis on the various classes of malware that can be categorised into three, with the first category belonging to fraudulent websites (for e.g. Phishing, Scamming and Spamming sites), the second category being the ones causing damage to property (for e.g. Rootkits, Trojans) while the third one being responsible for stealing information or credential through spying on one's machine (for e.g. Adwares, Spywares or Ransomwares). We obtained the classes of malware from the various Anti-virus engines in VirusTotal using the VirusTotal API for URLs. This allows us to decide on the type of harm that can be caused through these third-party malicious contents as well as the extent of the harm. Fig 5.14 illustrates on the various categories of malware spread through third-parties and provides the percentage of each category of malware in the overall count of malicious resources. Here, category 1 is the first category, category 2 is the second category and category 3 as mentioned above. The bar graph shows that Category 1 constitute the minimal percent of the overall malicious resources while categories 2 and 3 having 14.44% and 85.35% of the malicious resources respectively. Thus we conclude that the third-party content serve mostly such malicious programs that would lead to the theft of one's personal information or credentials. Therefore, these malicious contents hamper the privacy of users by making illegitimate attempts.

5 Conclusion

The state of the tangled web where websites load myriad of anonymous as well as acknowledged external resources from their third-parties to leverage multifarious services has critical implications for both websites and users. We have delivered a measurement of malicious external resources loaded to the top-1000 Alexa ranked first-party websites.

We revealed through our experimental analyses that over 84.56% websites trust third-party websites for importing external resources that are malicious with 33.73% and 3.17% websites loading potentially and dangerously malicious resources respectively. Our analysis also reveals that 6.95% first-parties are at the risk of loading content from domains they do not explicitly trust.

We further found 42.24% of the total amount of trusted resources are explicitly obtained image files with 3.34% obtained implicitly and 17.85% are JavaScript files from explicit resources while 0.77% being from implicit sources. Though the proportion of JavaScript resources loaded from third-party domains are supposed to be the party of maximum malicious resources, our analysis shows that image files constitute the most number of malicious resources.The image files constitute of 52.72% of the malicious resources that can cause range of active attacks.

By analyzing the dependency trees of first-party websites from different categories, we revealed some ubiquitous third party domains that are dangerously malicious. Alarmingly, we found major CDNs to serve over one third of malicious resources implicitly trusted by websites, with s3.amazonaws.com and ysa-static.passport.yandex.ru serving dangerously malicious resources to 0.6% of the top-200K website's landing pages, making use of JavaScript and image files.

We believe that our work will raise awareness on the need to tighten the loose control over indirect resource loading and implicit trust in the chain of web resources dependency by providing insights to web developers as well as content providers. It is crucial to enforce transparency and traceability mechanisms in the modern web eco-system, while loading content from external sources.

Future Work We intend to identify the nature of various threats through in-depth and extensive analysis of each of the type of attack that is prevalent in the dependency chain as well as in the overall web through further behavioural analysis. We tend to analyse the already obtained features during our analysis and make use of it to further analyse the web for newer threats so as to have an understanding to develop a prevention and security mechanisms that can be applied in between the dependency chains.

References

[1] K. A. B, S. A. and W. Robertson. *Financial cryptography and data security,*. (2017). 1, 16, 17, 18, 19

[2] D. Kumar, Z. Ma, Z. Durumeric, A. Mirian, J. Mason, J. A. Halderman, and M. Bailey. *Security challenges in an increasingly tangled web*. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 677–684 (International World Wide Web Conferences Steering Committee, 2017). 1, 3, 17, 18, 19, 26

[3] J. Chang, K. K. Venkatasubramanian, A. G. West, and I. Lee. *Analyzing and defending against web-based malware*. ACM Computing Surveys (CSUR) **45**(4), 49 (2013). 3, 18

[4] D. Liu, S. Hao, and H. Wang. *All your dns records point to us: Understanding the security threats of dangling dns records*. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1414–1425 (ACM, 2016). 4, 8

[5] *https://www.acsc.gov.au/publications/acsc*$_t hreat_r eport_2 017.pdf.(ACSC).$4

[6] *https://www.idtheftcenter.org/2017-data-breaches/*. 4

[7] M. Falahrastegar, H. Haddadi, S. Uhlig, and R. Mortier. *Anatomy of the third-party web tracking ecosystem*. arXiv preprint arXiv:1409.1066 (2014). 4, 8

[8] R. Gomer, E. M. Rodrigues, N. Milic-Frayling, and M. Schraefel. *Network analysis of third party tracking: User exposure to tracking cookies through search*. In *Proceedings of the 2013*

IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 01, pp. 549–556 (IEEE Computer Society, 2013). 4, 8

[9] J. E. Fossil, M. and T. Mack. *Symantec global internet security threat report. tech report 2009; symantec.* (Symantec, 2009). 4

[10] N. J. Percoco and T. SpiderLabs. *Global security report 2010 analysis of investigations and penetration tests*. Tech. rep., Tech. rep., SpiderLabs (2010). 4

[11] M. Ikram, N. Vallina-Rodriguez, S. Seneviratne, M. A. Kaafar, and V. Paxson. *An analysis of the privacy and security risks of android vpn permission-enabled apps*. In *Proceedings of the 2016 Internet Measurement Conference*, pp. 349–364 (ACM, 2016). 5

[12] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. *Detecting in-flight page changes with web tripwires.* In *NSDI*, vol. 8, pp. 31–44 (2008). 5

[13] *http://www.maxmind.com/en/city.*. (MaxMind). 5

[14] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. *You are what you include: large-scale evaluation of remote javascript inclusions*. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 736–747 (ACM, 2012). 5, 10, 27

[15] R. P. E. Konigsburg and E. Kvochko. *Embracing https.http://open.blogs.nytimes.com/2014/11/13/embracin https.* (MaxMind). 5

[16] J. R. Mayer and J. C. Mitchell. *Third-party web tracking: Policy and technology*. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pp. 413–427 (IEEE, 2012). 5

[17] *Mozilla foundation. public suffix list. https://publicsuffix.org/.* (Mozilla Foundation, 2018). 5

[18] V. Inc. *Virustotal public api. https://www.virustotal.com/en/ documentation/public-api/2018* (VirusTotal, 2018). 5, 22

[19] *https://cuckoosandbox.org//* (Cuckoo). 5, 22, 23, 24

[20] C. Castelluccia, S. Grumbach, and L. Olejnik. *Data harvesting 2.0: from the visible to the invisible web*. In *The Twelfth Workshop on the Economics of Information Security* (2013). 8

[21] B. Krishnamurthy and C. Wills. *Privacy diffusion on the web: a longitudinal perspective*. In *Proceedings of the 18th international conference on World wide web*, pp. 541–550 (ACM, 2009). 8

[22] M. A. Bashir, S. Arshad, W. K. Robertson, and C. Wilson. *Tracing information flows between ad exchanges using retargeted ads*. In *USENIX Security Symposium*, pp. 481–496 (2016). 8

[23] P. Holzinger, S. Triller, A. Bartel, and E. Bodden. *An in-depth study of more than ten years of java exploitation*. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 779–790 (ACM, 2016). 9

[24] K. Sun and S. Ryu. *Analysis of javascript programs: Challenges and research trends*. ACM Computing Surveys (CSUR) **50**(4), 59 (2017). 9, 18

[25] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda. *Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web*. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium (NDSS 2017). The Internet Society* (2017). 10, 19

[26] S. Morishige, S. Haruta, H. Asahina, and I. Sasase. *Obfuscated malicious javascript detection scheme using the feature based on divided url*. In *Communications (APCC), 2017 23rd Asia-Pacific Conference on*, pp. 1–6 (IEEE, 2017). 10, 17

[27] Y. Choi, T. Kim, S. Choi, and C. Lee. *Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis*. In *International Conference on Future Generation Information Technology*, pp. 160–172 (Springer, 2009). 11

[28] *https://github.com/dxa4481/cssinjection.*. (Github). 11, 20

[29] E. Rudd, A. Rozsa, M. Gunther, and T. Boult. *A survey of stealth malware: Attacks, mitigation measures, and steps toward autonomous open world solutions*. IEEE Communications Surveys and Tutorials **19**(2), 1145 (2017). 11, 21

[30] *https://www.blackhat.com/eu-15/speakers/saumil-shah.html* (Blackhat). 11

[31] S. Zhang, W. Wang, Z. Chen, H. Gu, J. Liu, and C. Wang. *A web page malicious script detection system*. In *Cloud Computing and Intelligence Systems (CCIS), 2014 IEEE 3rd International Conference on*, pp. 394–399 (IEEE, 2014). 11, 12, 17

[32] P. Likarish, E. Jung, and I. Jo. *Obfuscated malicious javascript detection using classification techniques*. In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, pp. 47–54 (IEEE, 2009). 11

[33] S. Kaplan, B. Livshits, B. Zorn, C. Siefert, and C. Curtsinger. *" nofus: Automatically detecting"+ string. fromcharcode (32)+" obfuscated". tolowercase ()+" javascript code*. Technical report, Technical Report MSR-TR 2011–57, Microsoft Research (2011).

[34] M. Cova, C. Kruegel, and G. Vigna. *Detection and analysis of drive-by-download attacks and malicious javascript code*. In *Proceedings of the 19th international conference on World wide web*, pp. 281–290 (ACM, 2010). 17, 19

[35] P. Ratanaworabhan, V. B. Livshits, and B. G. Zorn. *Nozzle: A defense against heap-spraying code injection attacks*. In *USENIX Security Symposium*, pp. 169–186 (2009).

[36] D. Canali, M. Cova, G. Vigna, and C. Kruegel. *Prophiler: a fast filter for the large-scale detection of malicious web pages*. In *Proceedings of the 20th international conference on World wide web*, pp. 197–206 (ACM, 2011). 11

[37] W. Xu, F. Zhang, and S. Zhu. *Jstill: mostly static detection of obfuscated malicious javascript code*. In *Proceedings of the third ACM conference on Data and application security and privacy*, pp. 117–128 (ACM, 2013). 11

[38] S. N. Bannur, L. K. Saul, and S. Savage. *Judging a site by its content: learning the textual, structural, and visual features of malicious web pages*. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, pp. 1–10 (ACM, 2011). 11, 19

[39] S. Arshad, A. Kharraz, and W. Robertson. *Include me out: In-browser detection of malicious third-party content inclusions*. In *International Conference on Financial Cryptography and Data Security*, pp. 441–459 (Springer, 2016). 11

[40] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. *Learning to detect malicious urls*. ACM Transactions on Intelligent Systems and Technology (TIST) **2**(3), 30 (2011). 11, 12

[41] M. Simeonovski, G. Pellegrino, C. Rossow, and M. Backes. *Who controls the internet?: Analyzing global threats using property graph traversals*. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 647–656 (International World Wide Web Conferences Steering Committee, 2017). 11

[42] I. Ghafir and V. Prenosil. *Malicious file hash detection and drive-by download attacks*. In *Proceedings of the Second International Conference on Computer and Communication Technologies*, pp. 661–669 (Springer, 2016). 12

[43] P. Seshagiri, A. Vazhayil, and P. Sriram. *Ama: Static code analysis of web page for the detection of malicious scripts*. Procedia Computer Science **93**, 768 (2016). 12, 19, 20

[44] Y.-T. Hou, Y. Chang, T. Chen, C.-S. Laih, and C.-M. Chen. *Malicious web content detection by machine learning*. Expert Systems with Applications **37**(1), 55 (2010). 13

[45] N. Weiler. *Honeypots for distributed denial-of-service attacks*. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2002. WET ICE 2002. Proceedings. Eleventh IEEE International Workshops on*, pp. 109–114 (IEEE, 2002). 13

[46] Z.-Y. Li, R. Tao, Z.-H. Cai, and H. Zhang. *A web page malicious code detect approach based on script execution*. In *Natural Computation, 2009. ICNC'09. Fifth International Conference on*, vol. 6, pp. 308–312 (IEEE, 2009). 13

[47] M. Akiyama, T. Yagi, T. Hariu, and Y. Kadobayashi. *Honeycirculator: distributing credential honeytoken for introspection of web-based attack cycle*. International Journal of Information Security **17**(2), 135 (2018). 13

[48] *https://www.alexa.com/* (Alexa). 15

[49] *https://www.w3schools.com/tags/tag$_a$.asp*. 15

[50] *https://www.pathinteractive.com/blog/design-development/rendering-a-webpage-with-google-webmaster-tools/*. 16

[51] J. Chang, K. K. Venkatasubramanian, A. G. West, and I. Lee. *Analyzing and defending against web-based malware*. ACM Computing Surveys (CSUR) **45**(4), 49 (2013). 17

[52] J. Hajian Nezhad, M. Vafaei Jahan, M. Tayarani-N, and Z. Sadrnezhad. *Analyzing new features of infected web content in detection of malicious web pages*. The ISC International Journal of Information Security **9**(2), 63 (2017). 17, 19

[53] M. Johns. *On javascript malware and related threats*. Journal in Computer Virology **4**(3), 161 (2008). 17, 19

[54] *https://www.darkreading.com/application-security/third-party-code-fertile-ground-for-malware/a/d-id/1316656.* 18

[55] *https://marketingland.com/ad-injector-study-google-127738.*. 18

[56] J. Chen and C. Guo. *Online detection and prevention of phishing attacks*. In *Communications and Networking in China, 2006. ChinaCom'06. First International Conference on*, pp. 1–7 (IEEE, 2006). 19

[57] *https://w3techs.com/technologies/details/cp-javascript/all/all.* (w3techs). 19

[58] *https://www.cybersprint.nl/blog/making-exploit-kit-analysis-great-again-part-1/.* 19

[59] Y. Wang, W. Cai, P. Lyu, and W. Shao. *A combined static and dynamic analysis approach to detect malicious browser extensions*. Security and Communication Networks **2018** (2018). 20, 21, 22

[60] *https://www.mike-gualtieri.com/posts/stealing-data-with-css-attack-and-defense.*. (mike). 20, 38

[61] *https://www.webroot.com/blog/2011/02/22/malicious-php-scripts-on-the-rise/* (Webroot, 2011). 21

[62] *https://threatpost.com/malicious-php-script-infects-2400-websites-in-the-past-week/132161/* (Threatpost). 21

[63] T. Van Overveldt, C. Kruegel, and G. Vigna. *Flashdetect: Actionscript 3 malware detection*. In *International workshop on recent advances in intrusion detection*, pp. 274–293 (Springer, 2012). 21

[64] *https://www.opswat.com/blog/can-video-file-contain-virus*. (Opswat). 21

[65] R. C. Shockley, P. J. Gendron, and J. M. Stevenson. *Detection of an undersea acoustic communications network by an energy detector*. The Journal of the Acoustical Society of America **141**(6), 4136 (2017). 21

[66] P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta. *Phishnet: predictive blacklisting to detect phishing attacks*. In *INFOCOM, 2010 Proceedings IEEE*, pp. 1–5 (Citeseer, 2010). 21

[67] *https://nodistribute.com//*. 22

[68] *https://virusscan.jotti.org/*. 22

[69] *https://metadefender.opswat.com/!/*. 22

[70] *https://git.korelogic.com/mastiff.git/*. 22

[71] *https://www.cryptam.com/*. 22

[72] Q. Chen and R. A. Bridges. *Automated behavioral analysis of malware a case study of wannacry ransomware*. arXiv preprint arXiv:1709.08753 (2017). 22

[73] *https://www.joesecurity.org/*. 22

[74] *https://github.com/detuxsandbox/detux*. 22

[75] *https://github.com/f-secure/see*. 22

[76] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. *Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web*. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 654–663 (ACM, 1997). 22

[77] *https://ssdeep-project.github.io/ssdeep/index.html*. 23

[78] *http://www.tcpdump.org/*. 23

[79] *https://www.volatilityfoundation.org//*. 23

[80] *https://mycrypto.com/account*. 23

[81] *https://github.com/zmap/zbrowse/* (Github). 26

[82] *https://developers.google.com/web/updates/2017/04/headless-chrome*. 26

[83] *https://nodejs.org/en/*. 26

[84] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. *Understanding website complexity: measurements, metrics, and implications*. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pp. 313–328 (ACM, 2011). 27

[85] S. Ihm and V. S. Pai. *Towards understanding modern web traffic*. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pp. 295–312 (ACM, 2011). 27