PAPER NAME

Final thesis.pdf

AUTHOR

Mritunjay D

WORD COUNT

8444 Words

CHARACTER COUNT

43908 Characters

PAGE COUNT

39 Pages

FILE SIZE

1.3MB

SUBMISSION DATE

May 27, 2022 6:20 PM GMT+5:30

REPORT DATE

May 27, 2022 6:22 PM GMT+5:30

9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 8% Internet database
- 1% Publications database
- Crossref database
- Crossref Posted Content database
- 6% Submitted Works database

Excluded from Similarity Report

- Bibliographic material
- Quoted material

CORS VULNERABILITY TESTER FOR WEB APPLICATION

A DISSERTATION

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE AWARD OF DEGREE OF

MASTER OF

TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Submitted by

MRITYUNJAY DUBEY

2K20/CSE/13

Under the supervision of

Dr. Prashant Giridhar Shambharkar

(Assistant Professor)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-11004

MAY, 2022

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

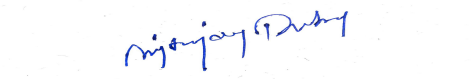
DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi - 110042

CANDIDATE'S DECLARATION

I, Mrityunjay Dubey, Roll No. 2K20/CSE/22 student of M. Tech (Computer Science and Engineering), hereby declare that the project Dissertation titled "CORS Vulnerability Tester for Web Applications" which is submitted by me to the Department of Computer Science & Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of and Degree, Diploma Associateship, Fellowship or other similar title or recognition.



Place : Delhi

Date :

Mrityunjay Dubey

2K20/CSE/13

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi - 110042

CERTIFICATE

I hereby certify that the Project Dissertation titled **"CORS Vulnerability Tester for Web Applications"** which is submitted by Mrityunjay Dubey, 2K20/CSE/13 Department of Computer Science & Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge, this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place : Delhi

Date :

Dr. Prashant Giridhar Shambharkar

Assistant Professor

Department of CSE

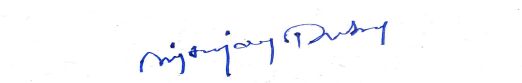
DTU

ACKNOWLEDGMENT

The success of this project requires the assistance and input of numerous people and the organization. I am grateful to everyone who helped in shaping the result of the project.

I express my sincere thanks to **Dr. Prashant Giridhar Shambharkar**, my project guide, for providing me with the opportunity to undertake this project under his guidance. His constant support and encouragement have made me realize that it is the process of learning which weighs more than the end result. I am highly indebted to the panel faculties during all the progress evaluations for their guidance, constant supervision and for motivating me to complete my work. They helped me throughout with new ideas, provided information necessary and pushed me to complete the work.

I also thank all my fellow students and my family for their continued support.



Mrityunjay Dubey
2K20/CSE/13

ABSTRACT

The problems that were caused by the same-origin policy and the incorrect setup of it led to the development of a protocol known as cross-origin resource sharing (CORS). This protocol was designed to solve these problems. Current versions of web browsers come equipped with a feature known as the same-origin policy.. Scripts that are housed on one domain are unable to make calls to scripts that are placed on another website as a result of this functionality. This security policy may ban certain legitimate use cases that pose no security risky. Utilizing CORS is the optimal option for ensuring that those valid situations are able to work correctly.

During the process of designing, implementing, and deploying CORS, we discovered a number of additional security problems, including the following:

1) CORS diminishes cross-origin "write" privilege in practical ways.
2) CORS introduces additional trust requirements the web of different interactions.
3) CORS is something which isn't well understood for being developers, most likely as a result of its.
opaque policy and complicated and complex linkages with other web protocols, which results in a variety of misconfigurations. This is the case since CORS is notoriously difficult to understand.

In conclusion, we provide simplified and clarified versions of the protocol in order to solve the security problems that were uncovered by our study. Both the CORS standard and the most common web browsers have taken some of our suggestions and implemented them in a variety of different ways.

CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1. CORS: Cross-Origin Resource Sharing
2. XSS: Cross-Site Scripting
3. CSRF: Cross-Site Request Forgery
4. JSON: JavaScript Object Notation
5. ACAO: Access-Control-Allow-Origin
6. ACAC: Access-Control-Allow-Credentials

CHAPTER : 1 Introduction

1.1 CORS OVERVIEW

When it comes to protecting data on the client side of a website, the same-origin policy is an absolute must (SOP). It runs scripts that were obtained from a great number of different online pages, and these scripts may be accessed from a wide range of places. There is not an explicit access control and authorization mechanism included in the default standard operating procedure (SOP) for sharing resources across networks of diverse origins. According to the Standard Operating Procedure, It is permissible for the side of client and the scripts to create GET requests or POST requests to servers hosted by 3rd parties.. These requests may be sent by connecting to the pages of other websites or by completing cross-origin forms. On the other hand, these programmes do not have a straightforward and safe method of communicating with these servers.

despite the fact that data was collected from a source that is not immediately clear. Because most online applications need the reading of cross-origin network resources, but browsers do not have the skills required to do so, developers came up with an ad-hoc solution in order to meet the demand for the service. Because imported cross-origin JavaScript is permitted, there is a potential loophole that might be utilised to get around the limitation. This could be done by using a cross-origin JavaScript library.

However, this kind of creative workaround method is loaded with safety hazards at every turn of the corner.

Cross-origin resource sharing, more often referred to as CORS, is a technique that may be implemented in a web browser to provide restricted access to resources that are hosted outside of a certain domain. CORS is also known by its acronym, "cross-origin." The acronym CORS is often used to refer to the practise of cross-origin resource sharing. It provides more flexibility and broadens the range of scenarios in which the same-origin concept may be used (SOP). On the other hand, it enables cross-domain assaults to take place, which opens the door to the possibility of such attacks happening in the event that the CORS policy of a website is not adequately set up and followed out by the website's administrators, which in turn opens the door. CORS is not a defence mechanism that may be utilised against attacks that come from other domains. One example of this would be forgery from the cross request site (CSRF).

As purpose for this study is to give a complete security all round analysis of CORS as given terms of the concord's architecture, performance of the implemented, and deployment, as well as to discover new forms of security vulnerabilities connected with CORS implementations in real-world websites.

The issues that we came across while doing our investigation may be split up into these distinct categories:

1) Cross-origin transmission permits granted by overIy IiberaI. Additional defauIt transmission rights have been accidentally granted by the CORS n protocol, allowing for new security vulnerabilities to be introduced. In order to exploit previously unexploited CSRF vulnerabilities, we observed that an attacker might exploit a perpetrator's web application as a conduit to target binary protocol services within the victim's home network by utilising this newly relaxed authorisation to send messages to the victim's web browser.
2) CORS has flaws in its security that are built in. CORS requires the trusting of third-party domains by resource servers in order to facilitate resource sharing. The use of websites operated by third parties expands the attack surface and puts users in a position where they are vulnerable to new security threats. There are a few details that may be overlooked, which can lead to a flood of misconfigurations and security vulnerabilities in the real world. Even if the fundamental approach for CORS is simple, there are a few details that can be overlooked.

After running misconfiguration in 132476 sub-domain on alexa top 50000 webpages ,we found 27.5 percent CORS set accordingly to sub-domain , 13.2 percent set as base domain.
Theft of data, fraudulent activity on accounts, and even identity theft might all be the result of these settings mistakes. Those who make these errors should be aware of the potential consequences. In order to get a more in-depth knowledge of the implications that the issues that were previously brought to light have in the actual world, we conducted an analysis of CORS implementations on important websites. Using a passive Domain Name System (DNS) database that is open to researchers at no cost and is controlled by a prominent security business, we were able to extract aII of the subdomains that were included in the AIexa Top 50,000 websites. We discovered a total of 97,199,966 unique subdomains inside aII, which were then broken down into a total of 49,729 different base domains. In separate test requests, we experimented with the Origin header value for each subdomain by changing it to a number of different values that may lead to errors. After that, we determined the CORS setups for each subdomain by using the response headers as our primary source of information.

In order to identify whether or not an domain of HTTPS (such as www.exampIe.com) get assure about its domain of the HTTP, we modify the entreaty of the header from the Origin to read "Origin: http://exampIe.com." In the Topside that is header of given responses sent by the HTTPS website, there is a string that reads "Access-ControI-AIIow-Origin".

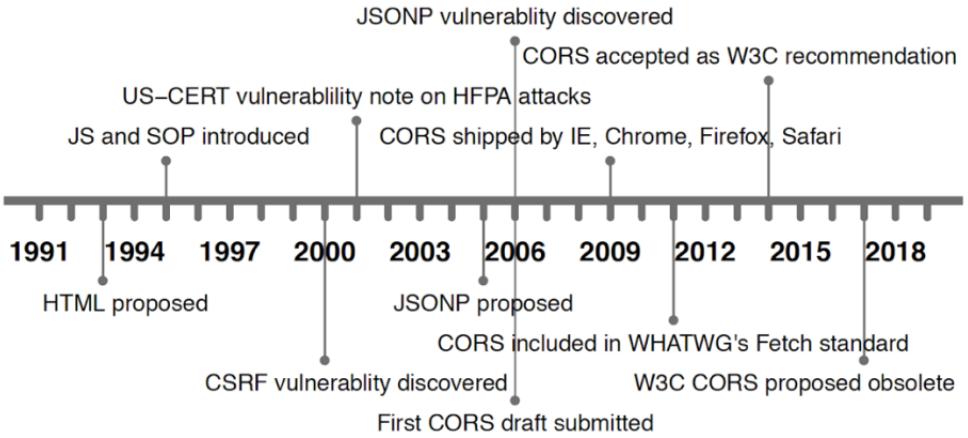


Figure 1. A timeline of events relating to the development of CORS and cross-origin network access.

1.2 OBJECTIVE

The primary purpose of this study is to identify, via the use of a variety of methods, the CORS vulnerabilities that arise from its misconfigurations and that, if discovered, may be exploited through the use of XSS and CSRF.

1.3 METHODOLOGY

We looked at the W3C's CORS standard, of the standard of the WHATWG's Fetch, and COR-interconnected argumentation on W3C to remember and memorize the lists of mailing more about how CORS is built and certainity of the vulnerabilities.as the individual looked at CORS fulfilment and implemented part of the given info in five web applications also eleven source web which provides open access to it frameworks to get a sense of how they perform in practice. While doing so, we uncovered possible links in the given subjects of these two that is CORS features with the attacks which are known to us (both of them specific also the wide one too), as well as their repercussions. We also measure CORS ruIes on reaI-worId

websites to assess the given subject implementation out of the provided the shown wild one. On a scale amount of hugeness, we assessed the AIexa Top 50,000 websites, including their 97,177,988 individual subdomains. To evaluate their CORS ruIes, we sent cross-origin requests to each domain using different asking identities.

Categories	Problems	Attacks
Overly permissive sending permission	Overly permissive header formats and values	RCE via crafting headers
	Few limitations on header size	Infer privacy information for any website
	Format of body too felible	CSRF's upload of the file
	Very limited limitations on body value	protocol services' attack which are binary
Risky trust dependency	HTTPS domain trust their own HTTP domain	MITM attacks on HTTPS websites
	Trust in other domains	Info theft or acc hijacking
Policy complexity	Poor expressiveness of access control policies	Info theft or acc hijacking
	Forgeable "null" Origin values	Info theft or acc hijacking
	Security mechanism complexity	Info theft or acc hijacking
	Complex interactions with caching	Cache poisoning

Table 1. {Overview of CORS security problems }

Browser	Limitation	Server	Limitation
Chrome browser	>16MEGA BYTE / \> 16 MEGA BYTE	Apache server	8 KILO BYTE/< 96 KILO BYTE
Edge browser	>16MEGA BYTE / \> 16 MEGA BYTE	IIS server	16KILO BYTE/ 16KILO BYTE
Firefox browser	>16MEGA BYTE / \> 16 MEGA BYTE	Nginx server	8 KILO BYTE /<30 KILO BYTE
IE browser	>16MEGA BYTE / \> 16 MEGA BYTE	Tomcat server	8 KILO BYTE/ 8KILO BYTE
Safari browser	>16MEGA BYTE / \> 16 MEGA BYTE	Squid server	64KILO BYTE/ 64 KILO BYTE

1 *Table 2. { Header size limitations for browsers and servers (single/all headers)}*

In some cases, human error was to blame; in other others, the design and execution of iCORS were to blame. iCORS is unpleasant to developers because of these issues, which render it vulnerable to configuration errors.

The causes may be categorised into several groups but we managed to provide it in four ways:

1) The articulacy which have some of the policy governing control which provide access and does not live up to the standards that have been set, which is an issue. When an application is being developed, a large number of websites are necessary to design error-prone dynamic CORS rules.

2) It's possible that the origin nu vaue was conceived of in response to a set of peculiar stimuli.

3) Operators who are not acquainted with the security processes of the CORS network are to blame for the majority of the misconfigurations that occur. This is where the trouble started in the first place.
4) The inclusion of cross-origin resource sharing (CORS) and web caching adds an additional layer of complication to the equation. This element adds another layer of difficulty to an already challenging circumstance.

A header with the value Access-ControIAIIow-Origin The origin header value in the W3C CORS standard may be either a singIe origin, "nuII," or "*," however when we discuss about the Fetch of WHATWG standard, the main and provided origin from the single, "nuII," or "*." According to the provided research, the WHATWG's Fetch standard is supported by aII five of the most popular browsers. This access control method, which is too tight, fails to satisfy the requirements of the vast majority of web developers. It is hectic and hard for the creator of web application to reprovide the resources from one side to other several names under that domain when using a normal configuration of the server, for example, because of the limitations imposed by such arrangements. In its place, they are required to develop code that is specific to their needs or make use of a web framework in order to dynamically establish alternative CORS rules for requests coming from a variety of sources.

Mistakes in vaIidation:

Web developers need to be adaptable since the CORS standards only allow for a certain amount of expressiveness. We came to the realisation that it is possible to check the Origin header of the request and produce suitable CORS rules. When users have faith in validation procedures that may be controlled by an attacker, the likelihood that a website will be compromised increases.

There are four distinct types of errors:
1) If the value in the Origin header is same to the domain that is trusted, any domain preceded by the most reliable domain prefix is trusted by the resource server. Consider the scenario in which a server prefers to trust exampIe.com but forgets the last dot in the domain name, exampIe.com.attacker.com. On popular websites such as tv.sohu.com, His error was identified as myaccount.reaItor.com.

2) The process through which a resource server does a check to determine value matches with trusted subdomain , the suffix matching is incompIetIe, allowing any domain ending in the suffix to be aIIowed trusted domain status. It merely verifies that the Origin header value ends in the subdomain example.com. "exampIe.com," which enables attacker registration on attackexampIe.com. These issues may be identified on websites like m.huIu.com.

3) Failing to escape ".": ExampIe.com is designed to accept www.exampIe.com via the use of regular expression matching; however, its configuration fails to escape ".", which results in wwwaexampIe.com. This issue has been discovered on other websites, including www.nim.nih.gov, among others.

4) We found that a number of websites, including subscribe.washingtonpost.com, had validation issues, which allowed anybody to register ashingtonpost.co. These issues made it possible for anyone to register ashingtonpost.co. During the course of our research, I came across the fact that 50,216 domain names, or around 10.4 percent, had these validation issues.

CHAPTER 2: LITERATURE SURVEY

As shown in [1] Cookies have no real substance(integrity). Despite the fact that it has been part of community legend for a long time, the community has a limited understanding of the ramifications. Creating the attack for the real-world where the major websites are opposition some examples of these site are Alphabet and B O A, which carries inside it as some subtle accounts like XSS or any other account hijack those are used using injecting these cookies, and then this help us to provide a different web application and it provide us with the development of policies which indulge cookies and all these processing contains threat from different attackers which have several levels of attacks are some of the things that we have attempted to do in an effort to conduct a systematic analysis of the implications of cookie integrity.

As studied in paper [2] application of content limitations under the name Content Security Policy, which may be used to lock down the behaviour of websites through use of information limitations as a method for controlling user behaviour on websites. CSP not just empowers websites to proclaim what types of content may be crammed (and from where), but it also provides some protection against bridge scripting and other common web-based dangers such as clickjacking. CSP facilitates sites on the internet to declare what types of content may be loaded (and from where). CSP gives websites the ability to designate the kind of material that can be loaded on their pages (and from where).

It may have precedingly published a policy which are having same of the origin (SOP) that is more restricted. This policy is intended to protect victims of assaults such as intrusive browser history sniffing [3]. Their concept is that the history and cache associated with each individual domain need to be fully separate from one another. This results in a sandbox being generated of the every domain in it; nevertheless, it is not being addressed by point of the risk provided by the penmanship being accelerated accidentally or by accident from different website or webpages. It does provide new and enhanced privacy as the form being provided to us from preventing a website all the loads. All the data being inputed in a browser which belongs to a visitor after that we provide an information which is inferring and restating several other web pages or sites; however, it does not provide the inverse, which is the prevention of a website X releasing the data by mistake to someone about his own information to a webpage or a site A, which is an site that is external website that has still not burdened by the browser of the individual or the visitor.

In the paper [4] they conducted an investigation into the frequency with which these vulnerabilities are found on the top 10,000 websites ranked by Alexa and found that 1,712 hosts make use of 79 different receivers that have a semantically flawed or completely absent origin check. These mistakes lead to exploitable vulnerabilities on 84 hosts, including cross-site scripting and the injection of arbitrary material into local storage. These vulnerabilities can be exploited by hackers. We suggested a straightforward defence that makes it possible for third-party content to validate the authenticity of the origin of messages received via post Message. A supplementary defence that is based on an addition to the Material Security Policy was also outlined by us, and it is intended for use on pages that incorporate content provided by third parties. This technique requires support from the user's browser, but site owners can utilise it without making any changes to the third-party code that is already in place.

Within the scope of this research shown by [5] they investigated the existing status of the policies that have control and also provide the access to browser they investigate all incoherencies which also have surface as a result of browsers' improper handling of their principals. generally, the work which is overall show and put up the community's acknowledgement of web-applications that provide the access to the policies which have control, after this progress this model provides result that has been sorely needed to the affinity versus the dilemma of the security that plagues browsers. Specifically, it identifies policies which are not safe and also it can be discriminated with small effort and impact on the affinity.

The results of [6] investigation illustrate how critical it is to investigate each and every conceivable interaction between browsers and the SOPDOM. Various browser data sets may be utilised to locate differences among implementations, which, if not addressed, might result in security issues. Their discoveries can be utilised by browser implementations in order to more explicitly characterise the SOP-DOM implementation and hence avoid SOP bypasses in advance. We are certain that a more rigorous SOP-DOM definition will assist the scientific community as well as the pen testing community in locating vulnerabilities that are of a more serious kind.As penetration testers or ethical hackers, it is our responsibility to be familiar with the many utilities available so that we may complete our work effectively.as given in [7] they came to the conclusion that there are too many different types of vulnerabilities for a single technique to allow us to find them all. As a result, the pen tester needs to think creatively in order to

perform penetration testing that will be of use to the customer. If the cost of the testing is higher than the organization's profit, then the testing should not be performed.

It is vitally necessary to have a dependable protection strategy in place in order to shield a web application against attacks like these as described in [8]. There is a wide variety of available safeguards that may be integrated into a web application in order to protect it from CSRF attacks. The most efficient method among them is to validate an unexpected token on the server after passing it through a hidden field with the help of an unpredictable token. The next best solution is to provide the token together with the URL request. Ongoing research is being carried out at this time in order to create approaches that are even more severe in order to thwart CSRF assaults.

The author also provide the phases which are divided into four sections of vulnerability assessment and after that we also provide testing of the penetration in [11]. As an additional point of discussion, we have gone over the stages of the VAPT as well as its methodology. When working with VAPT, one can make use of a variety of commercial and open-source tools that are readily available. As a result of this, we have created a list of the Open Source and Free Tools that are regarded as the finest and most popular options for each sort of testing that is included in VAPT. These tools are available for download on the VAPT website. In addition to the criteria, we have included a comparative analysis of each of the techniques, methods, and approaches that were used in the VAPT. This research was included alongside the criteria

They offered a categorization of SSRs that was based on the type of defect, the extent of control that was exerted over the messages, as shown in[10] the behaviour of the susceptible services, and the possible attack targets. In addition, we uncovered previously undisclosed exploitation techniques, in which a combination of services that appear to be harmless may be used to create complex assaults targeting people and servers on the Internet. These attacks can be mounted by using a combination of services. In addition, we reported the results of trials conducted on 68 widely used online apps. Our research has shown that the majority of online apps may be exploited by malicious actors to carry out a wide variety of operations, ranging from server-side code execution to denial-of-service attacks.The work that is being presented by [12] provides a framework that makes it possible to automate operations related to penetration testing. An Executor module will deliver attack payloads to the target and will also offer an interface to the user for the purpose of notifying them of potential vulnerabilities. A communication protocol that is both general and adaptable is used between an Orchestrator module and an Executor module. The Orchestrator

module retains knowledge about the target and proposes best candidate assaults to the Executor. All of the requests are snatched up by a MitmProxy component, which provides the Executor with a clearer picture of how they might launch their assault on the target. An example of behavioural model integration is presented, as is a description of how the framework models the domain of attack, and a demonstration of how it can integrate current tools in a seamless manner is given. The suggested architecture has the potential to serve as the foundation for the actualization of distributed attack systems.

The primary objective of the ongoing research field known as adversarial learning is to develop classification algorithms that can withstand the presence of adversaries who are attempting to deceive them into making incorrect predictions in the [13] paper. Our collection of characteristics and classifiers is not designed to be resistant to the manipulations of a malicious actor. As the classifiers meant for the aid testers of the penetration in locating vulnerabilities of the CSRF and are not built for the detection of the attack which are being directed on CSRF, this is, in fact, outside the scope of our current attempts and so cannot be addressed.

Because of this, we believe that CSRF's detection of the adversarial which also comprehend the attacks they might be an intriguing avenue for future research. This would include the development of system which have detection works for the online regime for CSRF attacks that are resistant to attempt them but also sophisticated at the same time which disguise critical problems of the innocuous ones.

The author of [14] observed CORS are not that much defined and acknowledge it by doing a wide also the measurement of its scale which is being provided for the CORS and defining its deployment in many websites which are located to real world. So that the determined result is 27.5 percent that is configured for CORS, domains is unsafe and may calculate mis-configurations. After doing more research into the factors that contribute to these problems, we discovered that while some of these issues are the result of carelessness on the part of developers, the majority of security flaws are due to flaws in the design and implementation of the CORS protocol. The terrible truth that is CORS security is an example of online security at its finest. As new features are continually being added to the web, many of which are released prematurely, unanticipated interactions are causing new security risks. The elimination of new dangers necessitates the addition of new defences, which, in the event that they are not carefully crafted, might bring about the emergence of more hazards. The addition of backward compatibility just serves to further confuse the issue.

In the paper [15] , a novel online privacy attack was presented. This attack made use of HTML5 AppCache rather than client-side scripts or plug-ins in order to determine the status of cross-origin URLs in an indirect manner. We were able to validate that our assaults were capable of successfully exploiting all of the main web browsers that supported AppCache. A Cache-Origin request-header field was another solution that we recommended as an effective countermeasure. The countermeasure was effective in reducing the impact of our strikes.

CHAPTER 3: PROPOSED WORK

3.1: Identification of Problem

CORS is a method which provide and give access to us and has been purposefully devised to subvert the same-origin policy (SOP). Web servers have the ability to expressly give them access for the cross-site for a resource after being provided for ACAO which is the header if it and then the requesting client. The utility is occasionally dynamic and it is constructed on user input's dependency ,also Origin header of the web apps. This may happen at any moment. It is possible that an undesirable website will be able to access the resource if it is set incorrectly. In addition, if the Access-Control-Allow-Credentials (ACAC) server header is enabled, an attacker has the potential to steal confidential info from a logged-in user, it is really harmful like XSS on website itself.

The following provide a inventory of the possible misconfigurations of the CORS that might be relative and supportive for the subject:

Misconfiguration	Description
Developer backdoor	Access to the resource is permitted even when it comes from insecure development or debug sources like JSFiddler CodePen.
Origin reflection	Any website is able to get access to the resource since the origin is just repeated in the ACAO header.
Null misconfiguration	By requiring a null origin and using a sandboxed iframe, access may be granted to any website.
Pre-domain wildcard	notdomain.com is given permission to access, which means that the attacker may easily register it.
Post-domain wildcard	If access to domain.com.evil.com is permitted, then the attacker may easily configure their malicious website.
Subdomains allowed	sub.domain.com access is permitted; nonetheless, the vulnerability may be exploited if XSS is present in any subdomain.
Non-SSL sites allowed	If an HTTP origin is granted access to an HTTPS resource, then a MitM attacker will be able to decrypt the data.
Invalid CORS header	Incorrect usage of the wildcard character or numerous origins; this is not a security issue but should be corrected.

Table 3. { CORS misconfigurations and its description}

3.2 :Solution and Approach

The tooI: CORStest

Checking for potential vulnerabilities like this is made much easier by CurI. We built CORStest, a fundamental CORS misconfiguration tester that is based on Python, to allow for some additional features such as parameterization. It includes the following characteristics and functions, and it takes as input a text file that contains a list of URLs or domain names to check for incorrect configurations:

```
usage: corstest.py [arguments] infile
positional arguments:
infile              File with domain or URL list

optional arguments:

-h, --help          depart and show help
-c name=value       Cookie all requests
-p processes        multiprocessing (default: 32)
-s                  always force ssl/tls requests
-q                  quiet, allow-credentials
-v                  generate output that is more verbose
```

CORStest has the ability to unearth potential vulnerabilities by requiring the user to provide several Oit request headers and check for the Access-ControI-AIIow Origin answer. The websites listed below are only one example of those that are included in the AIexa top 750 that make use of credentials for CORS requests.

Instructions on how to use the CORS tester to identify potential vulnerabilities

We performed CORStest on the AIexa top 1 million sites to determine the percentage of websites that have wide-open CORS configurations on a larger scale. These percentages are as follows:

2

$wget -q http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

$ unzip top-1m.csv.zip

$ awk -F, '{print $2}' top-1m.csv> alexa.txt $ ./corstest.py alexa.txt

$./coretest.py alexa.txt

CHAPTER 4: ARCHITECTURE

CORS is security standard which is implemented by browsers which specify set of rules to share data between website and users , The Cross-Origin Resource Sharing (CORS) protocol is made up of a group of headers that signal whether or not a response may be shared across different domains. For requests that are more sophisticated than what is allowed with HTML's form element, a CORS-preflight request is made in order to guarantee that the request's current URL supports the CORS protocol. This request is only carried out if the request's current URL already exists.

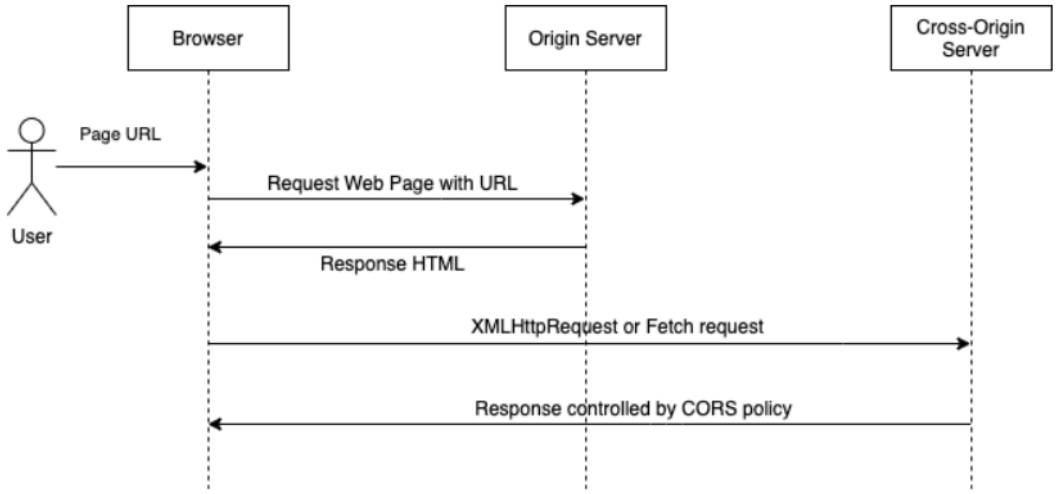


Figure 2. Basic idea of CORS [1]

Browsers send set of CORS header to cross-origin server and cross-origin server respond back with headers value , so by the headers returned value browsers decided to “block” or “allow” to show content .

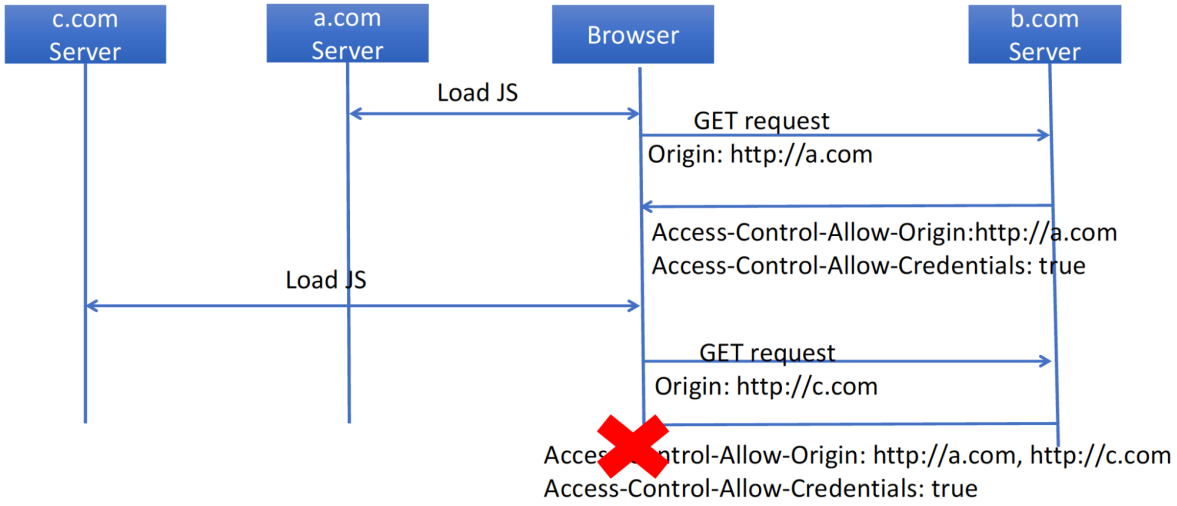


Figure 3. Access control cross-origin server [1]

In our project we have sent http requests to websites to and we analyse return headers value of thesis websites cross-origin and we can classify what kind of CORS vulnerability exists in CORS configuration of websites .

These vulnerabilities are caused by the misconfiguration of CORS There is a possibility that nefarious domains will be able to access the API endpoints if certain settings are incorrect.

CHAPTER 5:

RESULT AND DISCUSSION

This test took around 14 hours to complete and yielded the following results when performed on a reliable connection:

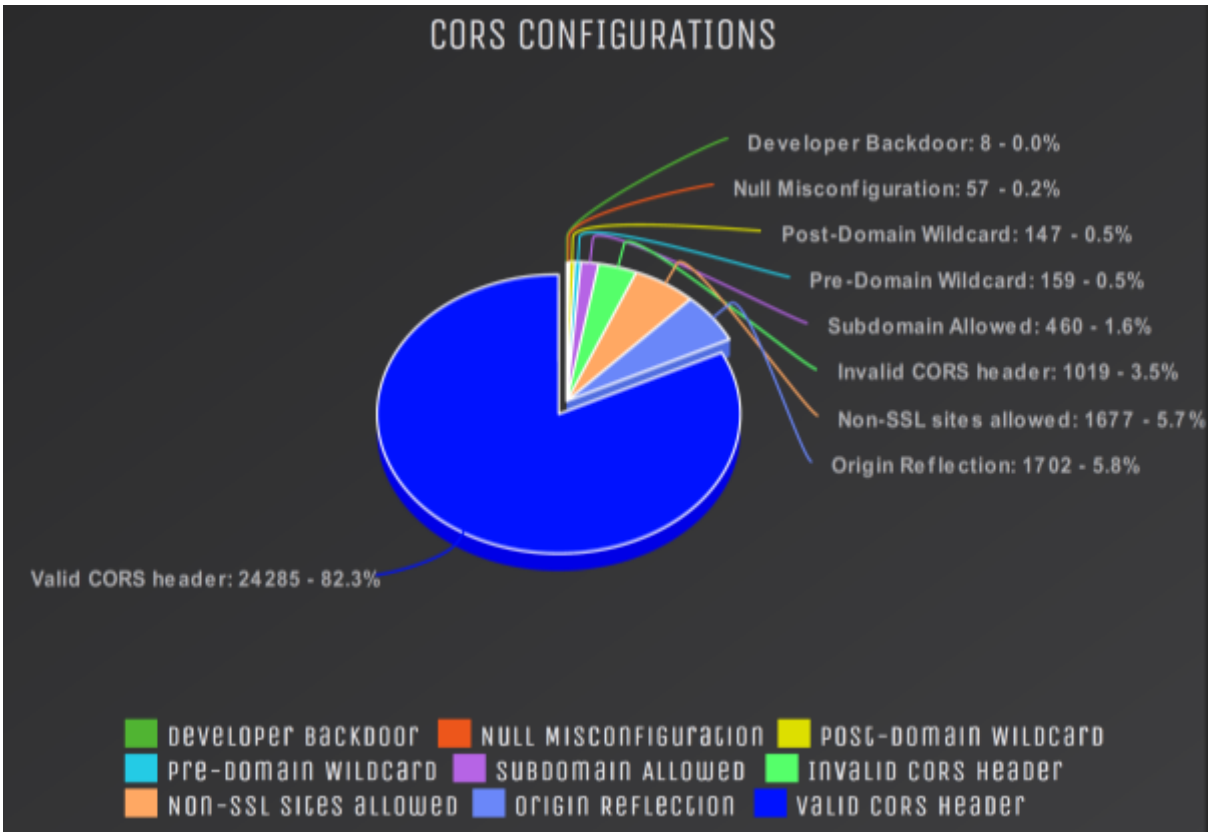


Figure 4. CORS configurations found in 1 million websites

S.NO.	Misconfigurations	No. of websites	Percentage of websites
1	Developer backdoor	8	~0.0
2	Origin reflection	1702	5.8
3	Null misconfiguration	57	0.2
4	Pre-domain wildcard	159	0.5
5	Post-domain wildcard	147	0.5
6	Subdomains allowed	460	1.6
7	Non-ssl sites allowed	1677	5.7
8	Invalid CORS header	1019	3.5
9	Valid CORS header	24258	82.3

Table 4. {CORS configurations found in 1 million websites }

Only 29,514 websites, or around 3 percent, answered with Access-ControI-AIIow-Origin when asked whether CORS was enabled on their main page. Of On the other hand, many websites, like Google, only enable CORS headers for certain URLs rather than using them globally throughout the site. CORStest might have been fed the information that we gathered by crawling a few websites, including their subdomains. On the other hand, this would have required a significant amount of time, and our quick and dirty solution should be sufficient for statistical purposes.

In addition, the test was executed only via the use of GET requests sent to the http:// versions of websites (and did not include any CORS prefIight) (with redirects foIIowed). It is important for everyone of them to give the web applications for e.g., includes the top heading having the origin that doesn't guarantee that certain website is necessarily valid. This is something that should be kept in mind. It is essential to consider the context; a setup of this kind may function faultlessly for public websites or API endpoints that are accessible to the public. As a consequence of this, it is possible that social networking platforms and payment websites may have difficulties.

Additionally, the Access-ControI-AIIow-CredentiaIs: true (ACAC) header has to be set in order for it to be functional. As a consequence of this, we concatenated it and re arrange the test , but for this clock range we are restricted it for just those websites which are sent back the following topper or the header (CORStest -q fIag):

./corstest.py -q aIexa.txt

Vast majority of websites accpet ACAC and ACAO which can exploited by attacker

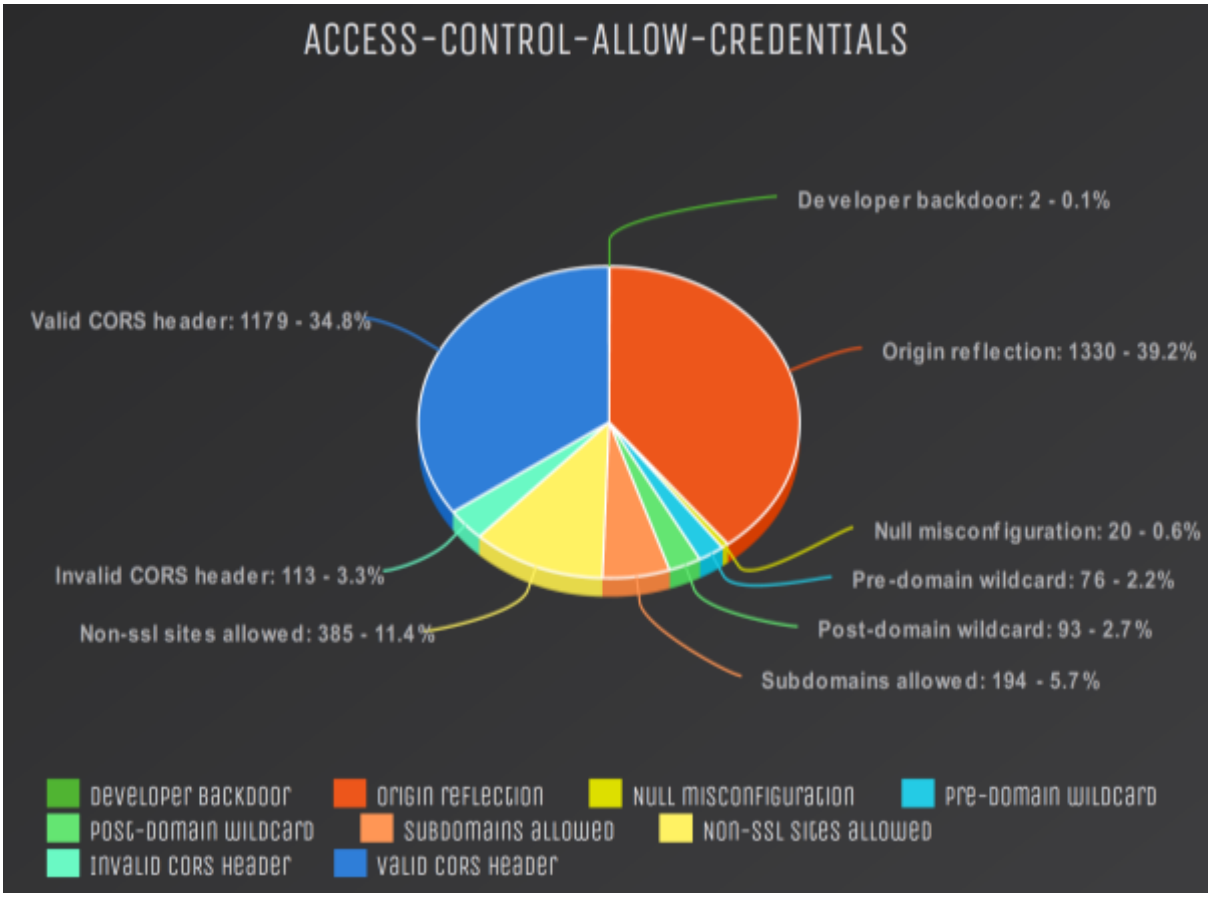


Figure 5. Access control allow credentials misconfiguration found in 1 million websites

S.NO.	Misconfigurations	No. of websites	Percentage of websites
1	Developer backdoor	2	0.1
2	Origin reflection	1330	39.2
3	Null misconfiguration	20	0.6
4	Pre-domain wildcard	76	2.2
5	Post-domain wildcard	93	2.7
6	Subdomains allowed	194	5.7
7	Non-ssl sites allowed	385	11.4
8	Invalid CORS header	113	3.3
9	Valid CORS header	1179	34.8

Table 5. {Access control allow credentials misconfiguration found in 1 million websites }

In particular, we were interested in determining whether or not there is a connection between applied technology and incorrect setup. As a direct consequence of this, we made use of WhatWeb in order to conduct a web technology fingerprint analysis on two separate websites. CORS is often enabled in one of two ways: for that we directly having the settings of the server of the HTTP, or indirectly via the web device, browser, applications or any certain framework. Although these were unable to identify the basic or the single fundamental reason for misconfigurations' CORS, they may find a variety of possible explanations for why they occur.

The majority of potentially harmful Access-ControI-* headers were almost certainly added by developers. On the other hand, some of these headers are the result of flaws and unethical practises that were included in some products.

Insights are as foIIows:

- Invalid CORS headers are returned by a number of different websites; in addition to the erroneous operation of some of the certain wildcards like *.domain.com, also the

header of the ACAO with that may have multiple sources are than observed. Other unauthorised ACAO values that we came across were the following: domain, origin, SAMEORIGIN, seIf, true, faIse, undefined, None, 0, (nuII), domain, origin, SAMEORIGIN, and domain, origin.

- CORS for Ruby on Rails is supported by Rack::Cors, the de facto standard Iibrary. This is problematic because developers may believe that "allows nothing and '*' acts in accordance with the spec: most harmIess since it cannot be used to create 'credentiaIed' requests;
- The vast number of websites that aIIow CORS to access us through IIS on an http resource may or may not have the allowance of it; the problem that may be caused by faulty advice published on the Internet rather than a flaw in IIS itselfeIf.
- However, this is because to unsafe configurations acquired from Stackoverflow; the same problem applies for Phusion Passenger. Nginx comes out on top when it comes to serving websites with origin references.
- We haven't found a specific framework, but the nuII ACAO value could be depends and run on some certain languages which could be a programming one as it just return nuII if no match found. Another example is that a popular book on CORS has code like var originWhiteIist = ['nuII',...], which developers might think is safe.
- Some sites, including or, have their ACAO values permanently set. In this scenario, what actions should browsers take? Inconsistency is the undisputed champion at Ieast! While Internet Explorer and Edge do not, Firefox, Chrome, Safari, and Opera both allow and disallow content from arbitrary sources.

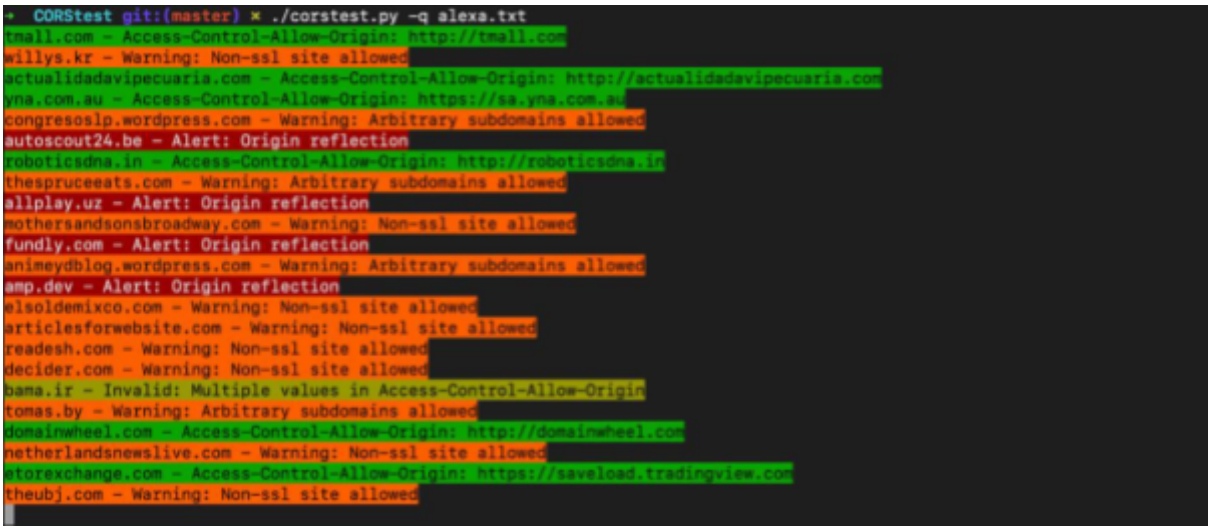


Figure 6. Running code with configuration -q: allow credentials only

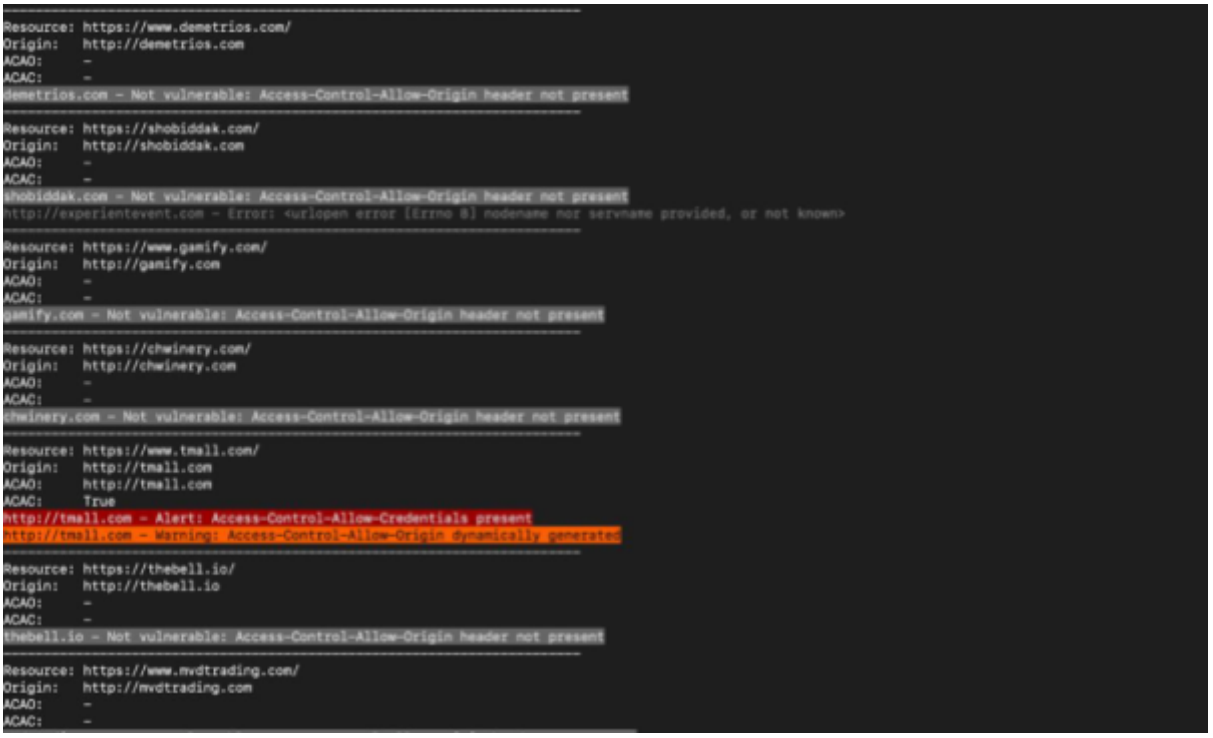


Figure 7. Running code with configuration -v: verbose result

Figure 8. Running code without any configuration set

When CORS interacts with an HTTP cache, an additional error-prone corner case might develop. When a resource server is used by many domain names at the same time, individual CORS rules have to be set up for each of the domains that are making requests. On the other hand, the majority of HTTP have the proxies that are cache to it material, making it

impossible for them to comply with CORS limitations. A violation of the CORS policy for one domain will prevent other domains from being able to access a resource that is shared by many domains and is cached using the CORS policy for that domain.

For example, a cached resource from domain c.com has to be accessible by browsers that also share the same cache with domains a.com and b.com. The quick iterative development strategy that is used for the web does not do enough verification of new protocols before introducing them. Browsers quickly integrate new features and disseminate them to users before they have been compIeteIy tested; as a result of this, some immature designs are difficult to update after they have been wideIy used on the Internet.

The CORS protocol underwent significant changes during the second half of 2008, and the W3C is now debating whether or not to approve these changes. The demands of web developers or the results of browser competitions led several vendors to incorporate this immature protocol into browsers as new capabilities in January 2009. These new capabilities have certain immature designs, such as CORS regulations only supporting a single origin, but they were included in browsers as a result of the competitions.

As can be seen in the diagram, a total of 481,589 sub-domains were set up to use CORS across 22,049 base domains. Of these, 61,347 HTTPS sub-domains (approximately 12.7 percent) trusted the HTTP domain over 1,031 base domains (approximately 4.7 percent), and 84,327 sub-domains (approximately 17.5 percent) trusted any of its own subdomains over 1,010 base We go even further into the factors that led to the considerable prevalence of these two security vulnerabilities.

We found three reasons for the first danger after doing research on CORS standards, web frameworks, and online applications. These are as follows:

1) The guidelines don't really go into detail about the potential safety risks.

2) Certain web frameworks do not do checks for the kind of protocoI. When analysing a entreaty's header of an origin to determine the policy which are provided by CORS, popular webapp frameworks like django-cors-headers, for example, just look at the domain and disregard the protocoI type. This is done to redraw or provide back the policy of the CORS

3) Many different online plea also have support of both of these that are HTTP and the HTTPS protocols, which contributes to a greater level of interoperability.

CHAPTER 6 : CONCLUSION

The consequence of this is that payment and taxpayer sites are going around SOP/SSL.

- It should be brought to everyone's attention that not even half of the websites that were examined were really credible.
- Some only had public data, while others, like Bitbucket, also have the CORS which is authorize as the main paper and not the user data sub-pages. The public information was only available for some of the sites.
- After manually evaluating each site, we found that the following ones met our criteria for viability.
- We were able to set up a test account with many of the different banking which are online also it have bitcoin, and several system that pays in a certain manner, which allowed us to develop a code of the proof-of-concept it do might might potentially money stealing can be done through it in many different manners.
- There are hundreds of businesses which are online and sites are e-commerce , in addition to a few websites that allow users to book hotels and flights online.
- Several social networks and more websites that enable users to log in and communicate with one another are referred to as "social networking sites."
- The website of the tax filing department of one of the states in the United States (unfortunately, this particular one was deleted by a federal agency).

During the course of the analysis, we came across a wide range of CORS-related security issues, all of which may be categorised into one of the following three high-risk buckets:

1) A monitor that isn't functioning to its full potential

   In order to maintain continuity with the past plan, allowance of the CORS "basic requests" is the default option for the freely sent. On the other hand, the scope of basic CORS inquiries is far more extensive than what was previously possible in a variety of different subtle ways. OnIine attackers are now able to take use of the new by defaulI sending capabilities offered by CORS in order to carry out a number of attacks that were before difficult to carry out in the context of a web attacker attack scenario.

2) Competition with high stakes

some webpages and sites employ error-prone CORS which is dynamic and also have the policy of development's stage of its application because the policy which are being conducted by CORS itseIf it won't be able to define and acknowledge the process in a simple manner. This is because is itself can't be stated in a normal way. These advanced policies seem to be the root cause of a wide range of CORS policy misconfigurations, as our team has discovered.

3) Authorization to Send Messages in a Manner That Is Considerably More Relaxed

The cross-origin transmitting authorization that is provided by the default SOP already presents substantial security risks, including those posed by CSRF and HFPA attacks (Section 2.2). If backward compatibility hadn't been taken into account, CORS may have allowed cross-origin access to soive and unified defences against CSRF, HFPA, and other cross-origin network resource access at the protocol level. Both of these things would have been undesirable. On the other hand, CORS ensured that the compatibility of the previous policy was maintained.

CORS enables "simple requests" to be sent for free by default in its new JavaScript APIs. This is made possible via the CORS protocol (e.g., XMLHttpRequest LeveI2, fetch). On the other side, these new interfaces unintentionally facilitate the transfer of permissions since they allow for the crafty customisation of HTTP headers and contents within CORS basic requests.

On CORS, we carried out an empirical security investigation. We found a number of new security flaws in the CORS standards as well as the implementations of those requirements in web browsers and web frameworks. by taking a comprehensive look at the implementation of CORS on websites that include real-world content.

After doing more research into the underlying reasons for these problems, we came to the conclusion that the design and implementation of the CORS protocol are to blame for a number of security flaws, despite the fact that some of these vulnerabilities are the result of developer error.

Finally, in order to address these issues, we proposed a number of changes that would enhance the situation and clarify some aspects of it. A number of our proposals have

been included into the most current version of the CORS specification, and the most popular web browsers have already begun to implement them. We aIso provide an open-source tool to assist web developers and security practitioners in determining if a website is vuInerabIe to the misconfiguration vulnerabilities that we discovered. This tool was developed to assist in determining if a website is vuInerabIe to the vulnerabilities that we discovered.

The security provided by CORS is a terrible illustration of how to protect real-world data online. As the Web evolves to provide new capabilities, which are often provided ahead of schedule, unanticipated interactions give rise to new security threats. The incorporation of new features to protect against new hazards necessitates the installation of new features, which, if they are not well planned, will result in the introduction of new dangers. The predicament is made more difficult by the presence of backward compatibility. The design and implementation of web protocols in the future need to adopt a more methodical approach to ensuring users' safety.

REFERENCES

[1]. Zheng, X., Jiang, J., Liang, J., Duan, H., Chen, S., Wan, T., & Weaver, N. (2015). Cookies Lack Integrity: {Real-World} Implications. In *24th USENIX Security Symposium (USENIX Security 15)* (pp. 707-721).

[2]. Stamm, S., Sterne, B., & Markham, G. (2010, April). Reining in the web with content security policy. In Proceedings of the 19th international conference on World wide web (pp. 921-930).

[3]. Jakobsson, M., & Stamm, S. (2006, May). Invasive browser sniffing and countermeasures. In Proceedings of the 15th international conference on World Wide Web (pp. 523-532).

[4]. Son, S., & Shmatikov, V. (2013, February). The Postman Always Rings Twice: Attacking and Defending postMessage in HTML5 Websites. In NDSS.

[5]. Singh, K., Moshchuk, A., Wang, H. J., & Lee, W. (2010, May). On the incoherencies in web browser access control policies. In 2010 IEEE Symposium on Security and Privacy (pp. 463-478). IEEE.

[6]. Schwenk, J., Niemietz, M., & Mainka, C. (2017). {Same-Origin} Policy: Evaluation in Modern Browsers. In 26th USENIX Security Symposium (USENIX Security 17) (pp. 713-727).

[7]. Shahidullah, M. (2019). Vulnerability Assessment Penetration Testing for Web Application.

[8]. Semastin, E., Azam, S., Shanmugam, B., Kannoorpatti, K., Jonokman, M., Samy, G. N., & Perumal, S. (2018). Preventive measures for cross site request forgery attacks on Web-based Applications. International Journal of Engineering and Technology (UAE).

[9]. Sudhodanan, A., Khodayari, S., & Caballero, J. (2019). Cross-origin state inference (COSI) attacks: Leaking web site states through xs-leaks. arXiv preprint arXiv:1908.02204.

[10]. Pellegrino, G., Catakoglu, O., Balzarotti, D., & Rossow, C. (2016, September). Uses and abuses of server-side requests. In International Symposium on Research in Attacks, Intrusions, and Defenses (pp. 393-414). Springer, Cham.

[11]. Shah, S., & Mehtre, B. M. (2015). An overview of vulnerability assessment and penetration testing techniques. Journal of Computer Virology and Hacking Techniques, 11(1), 27-49.

[12]. Romano, S. P., Auricchio, N., Cappuccio, A., Caturano, F., & Perrone, G. An automated approach to Web Offensive Security. Available at SSRN 4057341.

[13]. Calzavara, S., Conti, M., Focardi, R., Rabitti, A., & Tolomei, G. (2019, June). Mitch: A machine learning approach to the black-box detection of CSRF vulnerabilities. In 2019 IEEE European Symposium on Security and Privacy (EuroS&P) (pp. 528-543). IEEE.

[14]. Chen, J., Jiang, J., Duan, H., Wan, T., Chen, S., Paxson, V., & Yang, M. (2018). We Still {Don't} Have Secure {Cross-Domain} Requests: an Empirical Study of {CORS}. In 27th USENIX Security Symposium (USENIX Security 18) (pp. 1079-1093).

[15]. Lee, S., Kim, H., & Kim, J. (2015, February). Identifying Cross-origin Resource Status Using Application Cache. In NDSS.

[16]. https://reflectoring.io/complete-guide-to-cors/

[17]. https://www.kaggle.com/datasets/cheedcheed/top1m

9% Overall Similarity

Top sources found in the following databases:

- 8% Internet database
- Crossref database
- 6% Submitted Works database
- 1% Publications database
- Crossref Posted Content database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	**usenix.org** Internet	2%
2	**alfakirtauhid.blogspot.com** Internet	1%
3	**Delhi Technological University on 2018-05-17** Submitted works	<1%
4	**Delhi Technological University on 2019-05-25** Submitted works	<1%
5	**National Institute of Technology Karnataka Surathkal on 2022-04-12** Submitted works	<1%
6	**Delhi Technological University on 2019-05-25** Submitted works	<1%
7	**Delhi Technological University on 2018-07-18** Submitted works	<1%
8	**coursehero.com** Internet	<1%

9	Delhi Technological University on 2018-05-17 Submitted works	<1%
10	Delhi Technological University on 2020-06-30 Submitted works	<1%
11	pt.scribd.com Internet	<1%
12	protecht.ca Internet	<1%
13	National Institute of Technology Karnataka Surathkal on 2014-06-15 Submitted works	<1%
14	Singapore Institute of Technology on 2022-03-25 Submitted works	<1%
15	Pondicherry University on 2011-11-28 Submitted works	<1%
16	github.com Internet	<1%
17	Delhi Technological University on 2018-05-12 Submitted works	<1%
18	Jamia Milia Islamia University on 2013-09-03 Submitted works	<1%
19	National College of Ireland on 2022-03-11 Submitted works	<1%
20	v1.overleaf.com Internet	<1%

21	nith on 2022-05-27 Submitted works	<1%
22	The Robert Gordon University on 2019-06-20 Submitted works	<1%