

SECURING WEB APPLICATIONS ON NODE.JS PLATFORM

Project Report submitted in partial fulfillment of the requirement for the degree
of Bachelor of Technology

In

Information Technology

By

Pranjal Srivastava (171468)

Under the supervision of

Mr. Rizwan Ur Rehman

To



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Wagnaghat, Solan-173234

Himachal Pradesh

TABLE OF CONTENT

1)	Declaration by Candidate.....	(I)
2)	Declaration by Supervisor.....	(II)
3)	Acknowledgement.....	(III)
4)	Abstract.....	(IV)
5)	Chapter 1-Introduction.....	(1)
	1.1 What is Node.js?.....	(1)
	1.2 Clients using Node.js	(1)
	1.3 Advantages of using Node.js	(5)
	1.4 Security Issues in Node.js	(6)
6)	Chapter 2- Literature Survey	(9)
	2.1 Understanding and Automatically Preventing	
	Injection Attacks on Node.js.....	(9)
	2.2 Security Assessment of Node.js Platform	(10)
	2.3 Identification of Dependency-based Attacks on Node.js.....	(11)
7)	Chapter 3- System Development	(14)
	3.1 Attack Scenario.....	(14)
	3.2 Proposed Model.....	(15)
	3.3 Techniques used.....	(18)
8)	Chapter 4- Performance Analysis.....	(28)
9)	Chapter 5-Conclusions.....	(29)
	5.1 Conclusion.....	(29)
	5.2 Future Scope.....	(29)
	5.3 Applications Contributions.....	(30)
10)	References.....	(31)

LIST OF FIGURES

Fig No.	Page No.
Fig 1: Code snippet to print history of changes of a file.....	14
Fig 2: Architectural diagram for mitigating injection and dependency based attacks...	16
Fig 3 : Architectural diagram for Synode.....	18
Fig 4: Examples of Template trees.....	19
Fig 5: Comparison of attacks.....	22
Fig 6: Assessment of the code analysis.....	27

Declaration by Candidate

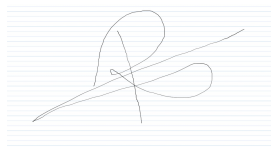
I hereby declare that the work presented in this report entitled “**Securing Web Applications on Node.js Platform**” in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat in an authentic record of my own work carried out over a period from January 2021 to May 2021 under the supervision of **Mr. Rizwan Ur Rehman** (Assistant Professor(Grade-II),Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Pranjal.S

Pranjal Srivastava (171468)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.



Mr. Rizwan Ur Rehman

Assistant Professor (Grade-II)

Computer Science & Engineering and Information Technology

Dated: 17.06.2021

Certification by Supervisor

This is to certify that the work which is being presented in this project report title “**Securing Web Applications on Node.js Platform**” for partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Information Technology** and submitted to the department of **Computer Science & Engineering and Information Technology**, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by **Pranjal Srivastava (171468)** during a period of January 2021 – May 2021 under the supervision of **Mr. Rizwan Ur Rehman** (Assistant Professor (Grade-II) , Department of Computer Science Engineering & IT) , Jaypee University of Information Technology, Waknaghat .

The above statement is made correct to the best of our knowledge.



Date: - 17.06.2021

Mr. Rizwan Ur Rehman

Assistant Professor (Grade-II)

Department of Computer Science Engineering & IT

JUIT, Waknaghat

Acknowledgement

I would like to express my deep gratitude to **Professor Rizwan Ur Rehman**, my project supervisor for his patient guidance, enthusiastic encouragement and useful critiques of this project. I would also like to thank my Professor for his advice and assistance in keeping my progress on schedule.

I would also like to extend my thanks to the technicians of the laboratory of the Computer Science & Engineering department for their help in offering me the resources in running the project.

Abstract

Node.js is a novel event-based network application platform which forces developers to use asynchronous programming interfaces for I/O operations. The native language for developing applications on this platform is JavaScript. But the platform's design decisions affect the security of its applications. This project will outline several possible security pitfalls to be aware of when using Node.js platform and server-side JavaScript. We will also address several vulnerabilities and will contribute towards developing and configuring resilient web applications on the Node.js platform.

Chapter-1

INTRODUCTION

1.1 What is Node.js?

Node.js is an open source, cross-stage platform for improving worker frameworks and quick and marvelous correspondence frameworks. Node.js was created by Ryan Dahl in 2009 and its most recent adaptation is v14.11.0. Based on Google Chrome V8 JavaScript Engine.

Node.js applications are JavaScript empowered, and can be run inside Node.js running time on OS X, Microsoft Windows, and Linux. Node.js additionally gives a rich library of different JavaScript modules that work with the advancement of web applications utilizing Node.js for a huge scope. Node.js utilizes JavaScript to compose order line devices and worker side contents to create dynamic website page content for constant web applications. It is utilized to improve web applications, for example, video real time locales, one-page applications, and other web applications.

1.2 Clients using Node.js

Node.js is another advanced technology many businesses decide to adopt in production. It is very popular in real-time applications or when looking for a quick and scaly solution. It has many advantages over other technologies and perhaps for this reason many great players use Node.js in their programs.

Netflix and Node.js

Netflix is the world's driving supplier of famous media and video real time. It is an information driven stage that utilizes a lot of A/B testing to make a rich encounter for its 93 million supporters around the world. The great quantities of various bundles routinely in the Push cycle make the issue of contingent reliance and downsizing of the application. That is the reason the organization chose to utilize Node.js for a lightweight and quick. Quite possibly the main outcomes of this is a 70 percent decrease in the first run through.

Trello and Node.js

Trello is a venture the executives application. The Trello worker side is based on Node.js. Serverdriven, a non-hindering worker has been an incredible answer for quick streaming updates, which require holding many open associations. Node.js was likewise useful as the organization refreshed the one-page application instrument. It was a speedy method to begin and ensure everything went easily.

PayPal and Node.js

PayPal, an online installment framework, has additionally taken out its backend advancement from Java to JavaScript and Node.js. Before that, the designing groups in the organization were isolated into program coders and codecs in the application layer, and it was not working as expected. From that point onward, the designers of the full stack helped, yet that model was not ideal by the same token. Tolerating Node.js tackled their issues, as it is permitted to compose program and worker applications in a similar programming language - JavaScript. Subsequently, the joined group can comprehend the issues on the two sides and react well to the requirements of the clients.

LinkedIn and Node.js

LinkedIn, the world's largest business communications service, is also trusted by Node.js, and last year they moved their mobile app backend from Ruby on Rails to Node.js. Even though it was still in its infancy, it was a wise move for the company. The new app is two to ten times faster than the one it previously installed, and is much heavier. Moreover, progress was very rapid.

Walmart and Node.js

Walmart is the largest retailer in the world, and is now entering the forefront of the online retail market. The giant has jumped on the bandwagon of partnering with Node.js - a brand new and very sophisticated technology despite the risks involved. The company has revamped the mobile app to provide complex content on the client side. Walmart has greatly appreciated Node.js' famous I / O and its single-threaded loop models that can effectively handle similar applications.

Uber and Node.js

Uber, a stage that interfaces drivers and clients needing transportation (and now food conveyance) administrations, utilizes numerous apparatuses and programming dialects in their application designing. Uber's innovation stack is continually advancing, and from that point forward they have acquainted new advances that appear with function admirably in certain spaces. All things considered, Node.js is as yet one of the vital drivers of the organization, as it empowers them to develop in accordance with the developing interest for their administrations.

Medium and Node.js

Medium is a web based distributing stage that utilizes Node.js on their web workers. In spite of the fact that from the outset a web application may appear to be a basic HTML page, there is a ton of innovation behind it. Medium is an information driven stage that advances along with clients and their conduct. Node.js is particularly helpful with regards to doing A/B tests to improve comprehension of item changes and to attempt novel thoughts.

Groupon and Node.js

Groupon, a mainstream exchanging stage that works in numerous nations all throughout the planet, has chosen to reconstruct the whole web layer over Node.js. The principal reason was the way that the stack they had recently utilized was hard to keep up. Particularly there on account of their extraordinary obtaining they end up with many different stacks to convey in various pieces of the world. These occasions have driven the organization to coordinate improvement across the entirety of their foundation.

eBay and Node.js

eBay, an international commerce company, has always been open to new technologies. The company is based on Node.js for two main reasons: they need the app as real-time as possible to maintain a live connection to the server and a solution that can configure a large number of eBay-specific services that display information on the page. Node.js seems to fit well.

NASA and Node.js

NASA utilizes Node.js also. Innovation is a higher priority than different applications since it saves lives, guarding space explorers during their risky space travel. In the repercussions of a mishap where one of the space travelers nearly kicked the bucket because of ineffective information being held at various areas, NASA confronted the test of sending EVA-related information to a solitary cloud site to decrease access times. Another program dependent on Node.js has decreased the quantity of steps in this cycle from 28 to 7.

Home Made and Node.js

Home Made is a crossover letting organization that conveys great customary lodging office administration for minimal price. The organization centers around £ 500,000 + convenience in London. The framework contains enormous, complex information with a critical number of records. As the engineers put application execution as perhaps the main, Node.js advancement group suggested Node.js as a backend arrangement. The stage was adjusted to serve more traffic, adequately dealing with different applications shipped off the worker while keeping up elite and a smooth client experience.

Yahoo and Node.js

The Yahoo frontend group has been utilizing Node.js underway to serve site pages for seemingly forever. A considerable lot of their new items are Single-Page-Applications or substance locales that utilization Node.js.

They siphon around 25,000 applications each second through Node.js workers, which guarantees that this JavaScript structure is likewise proficient. Eric Ferraiuolo, Principal Software Engineer at Yahoo, uncovered that every one of the resources they have moved to the Node.js stack have seen an expansion in execution.

1.3 Advantages of using Node.js

- **It requires a low learning curve**

Amongst the developers, JavaScript is a widely used programming languages for building front end web applications. It requires less time to learn, write efficient codes and adapt working with Node.js, even if it is just a beginner or a junior developer.

- **Scalability**

Node.js provides highly scalable environment and helps the server to process requests seamlessly by employing a non-blocking event loop mechanism. It can work with microservices which lets you to segregate your application into smaller parts. This way, tasks are allocated efficiently among different teams for fast-track development and maintenance of each division of application.

- **High Performance**

Web applications built on node benefit massively from its ability to multitask. It has a lone threaded, event driven architecture which methods numerous simultaneous requests efficiently without blocking RAM. Moreover, it is built on V8 engine and written in C++.The V8 engine breaks JavaScript functions into machine codes with high efficiency.

- **Boosted Web App Development Speed**

Being lightweight, Node.js enables developers to accelerate the app development speed. The Node Package Management registry provides a ton of libraries, ready-to-use codes, and other resources from GitHub, which helps in saving a lot of coding time and effort. It also reduces the size of the application and shortens the development cycle, thus improves the time-to-market of app.

- **Reduces loading time by quick caching**

Node.js provides a caching module which helps the developers to reduce task workload and reexecution of code. So basically, every time when first module of a web application gets a request,

the request gets cached in the application memory. This way users get to quickly access the web pages in no time.

- **Improved app response time**

Node.js is capable to handle multiple requests at the same time as it has a single-threaded eventloop model which offers a non-blocking asynchronous architecture without creating more threads by employing fewer resources. This helps to boost the responsiveness of an application.

- **Community support**

Node.js has a large and active community of software developers across the globe, contributing continuously to the development and improvement of the technology. This community is backed up by many leading tech giants like Amazon, Google, Facebook and Netflix, contributing massively to Node.js.

1.4 Security Issues in Node.js

The potential security weakness of the Node.js platform which are identified are as follows:

▪ Fragility of Node.js Applications

A single event-based construction made with both threads is an important feature and a major weakness as well. This is because if any system error discards an unexpected variant, it breaks the event loop, thus terminating the entire system. Similarly, when setting big data unexpectedly or a loop that is not controlled in the application concept, it results in blocking all other communications.

▪ Server-Side JavaScript

Client side customer projects to zero in on JavaScript learning assets. On the Client side, JavaScript applications run in a sandbox-confined space of an internet browser, where contents can't open area documents or undetected organization associations. JavaScript customer side cycles are brief and any blunders that happen in applications found on a specific website page are opened in the program window. Interestingly, outsider worker applications work without sandboxing and serve an enormous number of clients simultaneously. Worker side cycles for the most part handle the heap without interference for quite a while frame. Any defilement that isn't planned or brought about by the world by the intruder, can be cataclysmic.

▪ Handling Computation Intensive Requests

Since Node.js has a solitary occasion line design, tedious assignments, for example, picture estimation or complex computations will obstruct a huge string.

While a huge link is engaged with the mix, no new associations can be acknowledged and fundamentally the whole worker is trapped in computation time. To tackle this issue, numerous bundles are intended to make and oversee foundation errands. It is essential to restrict and monitor the quantity of concurrent activities behind the scenes, in any case the aggressor may begin an enormous number of complex assignments to get to worker assets.

▪ Malicious Installation Scripts in Packages

The usefulness of Node.js can be upgraded by bundles comparing to the CommonJS bundle design. The bundle portrayal script gives the proprietor the alternative to determine which records ought to be run at various occasions during the establishment or utilization of the module. This will permit the

assailant to make a content that will be utilized when introducing the npm pack. In spite of the fact that fresher npm adaptations bring down these rights, an assailant can forestall a vacation by setting an unstable client stop boundary, which will incapacitate the plunge.

This implies that most introduced bundles are inadequate in both code quality and backing. It likewise implies that before dynamic shipment, bundles should be appropriately investigated as unintentionally or coincidentally contain security weaknesses.

▪ **Server Poisoning**

On web application workers, each application makes another (introduced) measure for the kid and all handling inside the string is ended when that string or cycle closes. As the whole Node.js framework works in a worldwide setting with a solitary link, at that point if the application can debase the worldwide climate, the worker's conduct may change.

The amazing JavaScript mode can assist aggressors with distinguishing the source code of the application.

Since no split is given by JavaScript, the aggressor can check and change the worldwide status. JavaScript additionally permits you to grow classes and rethink errands during activity. With these chances accessible, the vital elements of the framework can be effectively modified by the aggressor.

Chapter-2

LITERATURE SURVEY

2.1 Paper 1: Understanding and Automatically Preventing Injection Attacks on NODE.JS (Cristian-Alexandru Staicu, Michael Pradel, and Benjamin Livshits)

The Node.js ecosystem has introduced many modern applications such as web-based web applications and desktop applications. In contrast to JavaScript-based customer code, Node.js applications communicate directly with the underlying operating system without the need for a security sandbox. This paper provides an in-depth study of the dangers of injections (editing errors that allow the attacker to inject and use malicious code) on the Node.js platform. The platform offers two API families that can accidentally allow injections. The eval API and its variants capture the string argument and interpret it as a JavaScript code, enabling the attacker to create a code that conflicts with the context of the current application. The exec API and its variants capture the string argument and interpret it as a Shell command, enabling the attacker to execute commands of a certain level of programming, beyond the context of the current application. Attackers can combine both APIs by injecting JavaScript code with eval, which uses the exec to execute Shell commands. Due to these two APIs and the lack of a security sandbox, the injection vulnerability can cause much more damage than browsers, e.g., it can modify the local file system or even hack a complete machine.

To address this issue, this paper introduces SYNODE, an automated mitigation system that enables the ability to use vulnerable modules safely by combining consistent analysis and enforcement of security policies. The main idea here is to statistically model the values transferred to APIs that are prone to injections and to compile a time-based policy based on grammar from these variables. This method requires no developer involvement and can be automatically installed as part of the installation of the module, further simplifying the delivery process.

2.2 Paper 2: Security Assessment of Node.js Platform (Andres Ojamaa and Karl Duuna)

Node.js is an event-based network application platform that enables developers to use asynchronous planning connectors in I / O functionality. Little research has been done on how the security of Node.js applications is affected by its design decisions. This paper provides an in-depth study of a number of potential security pitfalls on the Node.js platform and in JavaScript such as applications such as the first applications that work without security sandboxing on the server side that opens the attacker's path and potentially catastrophic. Second, the formation of a single event based on both threads is a major factor and a major weakness in many ways. Mainly because any error of a different throwing system breaks the event loop, so disconnecting the entire application and time-consuming process such as image measurement or complex calculations will block the main thread so that no new connections are accepted and the entire server is compromised.

The paper also describes the two risks involved. One has to do with common metaphor. Since normal speech simulation is faster in normal mode and does not cause any I / O, the node-validator creates the same, in the event loop. Now, if the app transfers a user's specified thread (any regular expression) to the validator, the malicious user will be able to block the event loop, which is why it blocks the entire system. In other resource management, Node.js does not have the basic functionality such as file uploads to be implemented using code. The code simply includes incoming data in the variable without security testing. An attacker with a fast network can easily load GBs of data that will be stored in memory, until the memory allocated to Node.js is completed and the process crashes, leading to service resistance.

2.3 Paper 3: Identification of Dependency-based Attacks on Node.js (Brian Pfretzschner and Lotfi ben Othmane)

Node.js is a stage side run-time stage for JavaScript applications that utilizes outsider modules as a reliance. To utilize Node.js applications, one requirements to introduce every one of the conditions. This paper centers around malevolent code got from outsider devotion and endeavors to assault applications dependent on JavaScript and Node.js weaknesses.

This paper talks about 3 weaknesses of JavaScript and Node.js dependent on the chance of 4 unique assaults. In the first place, manage shortcomings:

1. **Worldwide Variables:** Many JavaScript capacities are utilized as capacities and factors put away across the globe. This worldwide rating is divided between all modules of a specific Node.js program which incorporates their conditions. This sharing of land variety is viewed as an awful practice as it tends to be utilized for outside transformations.
2. **Monkey getting:** The powerful alteration of classes and errands during dispatch in JavaScript is known as monkey combination undertakings. Along these lines, another assignment can get the primary monkey and compose it over and the appended work will be continually mentioned. Also, in JavaScript we can't decide if a capacity has a mandrill tag or not.
3. **Store of downloaded modules:** In request to utilize the capacities and varieties of the application, the module should be stacked expressly on the sizes of the application utilizing the JavaScript require work. This capacity accompanies file material stacked with stacking modules and their exportable properties. As a worldwide variable, this reserve is rearranged between all modules in a particular utilization of Node.js and any module might be responsible for deceitful application store content.

This paper identified 4 dependence attacks due to the 3 weaknesses mentioned above.

1. **Global Leakage:** There is a possibility that dependence can reach global volatility and reward their content. For example, values for all natural variables can be disclosed on an attack-controlled

server by writing only one line of code. This can lead to the leakage of sensitive information and also this evidence can be misused to commit wrongdoing using the app.

2.Global Management: This type of attack can exploit dynamics or functionality that can be achieved globally to alter application performance. This change in behavior can be made using a pair of monkeys.

3.Location Fraud: The variables in Node.js may have the range of location found within the caller context but not outside. However, there is a weakness in Node.js known as full cache module and this can be used to deceive other dependents such as the control flow of a given system can be taken over by the attacker.

4.Reliable tree management: There are two variations of this attack. The first is about loading the dangerous dependency instead of the victim's dependence. Now, any given system modules that load the victim's dependence, will use the risky dependence found in the repository instead of the real one. The second uses the resolution and calls the victim's dependence function and then changes it. Now, this modified method is stored in the dependency file.

This type of attack can be used to bypass security controls. For example, the behavior of a module that addresses risk dependence may be modified in such a way that the environment will never be regenerated with the latest versions of dependence.

In this paper, static code is analyzed to detect attacks using the TJ code analysis tool. Watson Update Libraries (WALA). It is a fast and efficient analysis framework available for JavaScript source code analysis. The analytics tool currently has several limitations including unlimited application size limit (maximum 9 MB), dependence on binary code content, etc.

This paper concludes by identifying 4 dependency-based attacks based on Node.js's 3 vulnerabilities and the creation of JavaScript by creating static code analysis using TJ. Watson Update Libraries (WALA). The performance of the analysis is satisfactory without the onset of ground leaks, which take more than 2 minutes. The analysis is integrated with OpenWhisk, a free open source platform. Before accepting a downloaded NodeJS application, the extended OpenWhisk platform undergoes analysis. The request is denied when the attack is detected.

Chapter-3

SYSTEM DEVELOPMENT

3.1 Attack Scenario

Firstly I wrote a simple code using Node.js and express framework that prints the history of changes of a file.

```
admin.py — form2\  models.py — form2\myform  x  urls.py — form2\myform  x  view
1  const exec = require('child_process').exec;
2
3  app.get('/history', (req, res) => {
4    // Read file name from the URL query string.
5    const file = req.query.file;
6
7    // Prepare command to run
8    const command = `git log --oneline ${file}`;
9
10   // Execute the external program and send the response back
11   exec(command, (err, output) => {
12     // Respond with HTTP 500 if there was an error
13     if (err) {
14       res.status(500).send(err);
15       return;
16     }
17
18     // If the program ran successfully, send the output in
19     // the HTTP response
20     res.send(output);
21   });
22 });
```

Fig 1: Code snippet to print history of changes of a file

Then I send an http request to read the file history and to see if the program was working fine and the output was

```
f38482b Call git log command
5444afb Initial commit
```

Then I invoked this command - `git log --oneline app.js;ls` , and the result lists all information of current directory.

```
f38482b Call git log command
5444afb Initial commit
app.js
bin
node_modules
package-lock.json
package.json
public
routes
views
```

So this command clearly lists all the information about my current directory in which `app.js` is present. This type of security vulnerability is called **command injection**.

3.2 Proposed Model

The model I have proposed is developed with the help of synode technique and TJ Watson libraries.

This model provides a consistent analysis not only of the values transferred to the injection APIs but also to reliability-based attacks such as land leakage, land deception, site management, tree deception.

First the static analysis is performed on the numbers transferred to the injectable APIs, and then for each such API call, the analysis summarizes all the possible values transferred to the task called for the drug representation. of cable prices that may be transferred to the service. Finally, based on the templates, the analysis determines each input location of the API injection call whether it is statistically secure or whether to include an operating time check that prevents malicious cables from accessing the API.

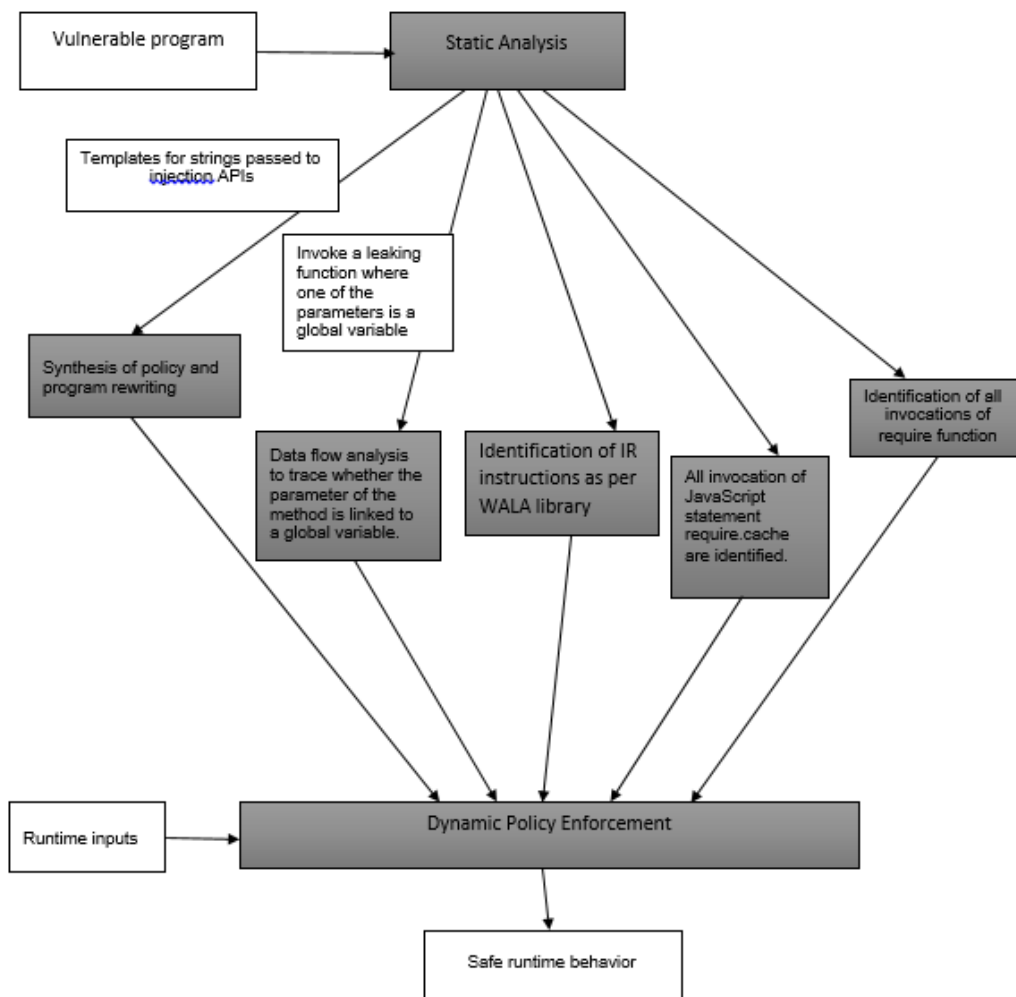


Fig 2 : Architectural diagram for mitigating injection and dependency based attacks.

To identify dependency based attacks, source code is converted to Intermediate Representation (IR) with the help of WALA framework. Then code analysis is performed to detect attacks, for global leakage leaking function is invoked where one of the parameters is a global variable. Analysis is done to trace whether the parameter of the method is linked to a global variable using http protocol.

If an IR instruction is of type AstGlobalWrite and the variable is a built-in JavaScript global variable, then Global manipulation attack is confirmed.

If analysis tracks IR instructions JavaScriptPropertyWrite and SSAPutInstruction then attack of type Local manipulation is identified.

To detect a type of tree exploitation, all requests for a JavaScript require.cache are identified. Items returned by this request are tracked using analytics data analysis to detect fraud.

After Static analysis then strong policy enforcement is applied. The purpose of this machine is to reject the values found to be dangerous in terms of policy. By default, we want to prevent prices that extend the template integrated into the call site in a way that the engineer would not consider.

3.3 Techniques used:

3.3.1 SYNODE

It is an automated mitigation method that combines static and operational analysis of security policies to allow vulnerable modules to be used in a safe manner. The basic idea is to statistically modify the values transferred to the APIs that are prone to injections, and to compile a grammatical policy of operating time from these templates.

Here the principle thought is to take a gander at all the outsider modules during their establishment and change them to empower protected mode. The mix of the two strategies is utilized as a component of the reworking. To begin with, we intend to break down genuinely the qualities that can be moved to injectable APIs. The static investigation delivers a format that portrays the qualities moved to the APIs. Second, in coding regions where static investigation can't ensure the shortfall of infusions, we present a force escalated measure that stops vindictive contributions prior to moving them to APIs. The mix of these techniques is applied to the module during establishment utilizing NODE.JS input snares, viably constraining the experimental method of outsider modules.

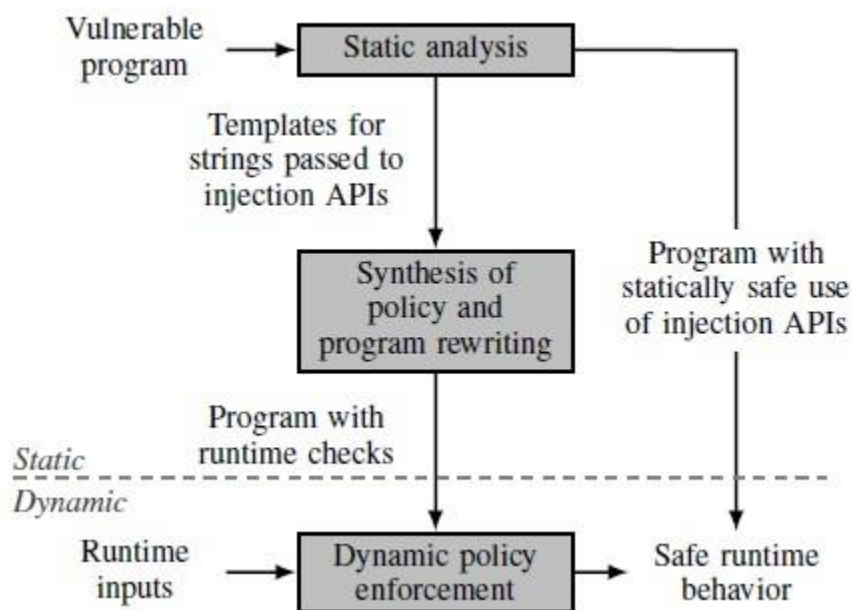


Fig 3: Architectural diagram for Synode

A) Static analysis

For each call site for such an API, static examination sums up all qualities that can be moved to a capacity called tree portrayal. From that point forward, the measurable examination assesses the tree to acquire a bunch of layouts, demonstrating the known and obscure pieces of the string esteems that might be sent to the work. At long last, in light of the layouts, the investigation decides each info area of the API infusion call whether it is genuinely secure or whether to incorporate a working time watch that keeps malevolent links from getting to the API.

i. Eliminating Template Trees

Reverse examination. From the information site of the infusion API, the examination spreads data about the potential upsides of the link debate moved to the API call with the contrary stream controls. Dispersed information is a medication that addresses current logical data about esteem. The format tree is a non-cyclic, associated, arranged chart (N, E) where hub N addresses the adaptability of the wire, the representative turn, the administrator, or the alternate way, and the edge E addresses the relationship of the home between the two territories.

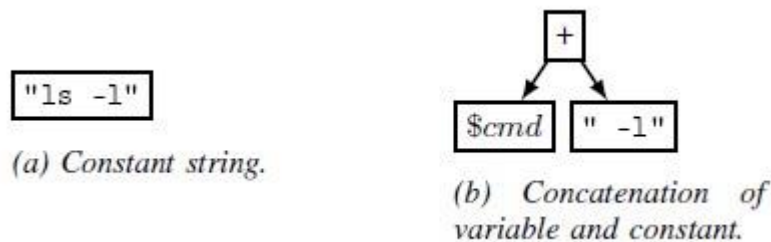


Fig 4: Examples of Template trees

By automatically removing such trees with templates, data analysis is used, which transfers template trees by system. Starting with an empty tree inject call API, the analysis uses the following transfer functions:

- Constants- Regular cord reading has a node containing a fixed value.
- Variable - A study of local variables or performance parameters produces a node representing symbolic variables.
- Operation- Installing a function, such as connecting two wires with a +, leads to a tree where the root node is the operator and its offspring represent the operator.

- Calls- The call function identifies the tree where the root node is the called function and its offspring represent the basic object and the telephone resistance.
- Assignments- The provision of the form $lhs = rhs$ replaces the current tree by replacing any appearance of the symbolic variable associated with lhs by the tree which results in the use of the switch function in rhs.

At whatever point the back control stream meets, the examination incorporates two tree formats of the joined branches. The blending capacity includes another hub with two trees combined as its posterity.

ii. Investigating Template Trees

Contingent upon the layout trees removed the information back investigation. To discover the examples of a given tree, the examination continually inspects the subtrees in a sub-design up to the roots. Testing modifies execution hubs with semantics known for execution result. Our present execution is demonstrating semantics of string connection, `Array.push`, `Array.join`, and `String.replace` where contentions are a consistent string. These capacities cover a significant number of the format trees that are removed for examination from the real JavaScript code.

At long last, the investigation changes over each tried tree into a format by joining a persistent grouping of letters into reliable strands and addressing every one of the emblematic qualities and the obscure capacity between these components as obscure qualities.

iii. Solidly Identified Secure Calls

On the off chance that all formats for a specific call site are persistent strings, that is, there are no obscure qualities in the layout, at that point the investigation presumes that the call site is genuinely secure. For non-unsafe call locales, no presentation time test is required. Conversely, the examination can't overestimate the requirement for infusions if call site formats contain obscure qualities. For this situation, the test is past due.

B) Dynamic Enforcement

On the off chance that call site formats contain obscure qualities, we give an amazing support strategy. The reason for this strategy is to dismiss the qualities discovered to be risky as far as strategy. Normally, we need to stay away from costs that broaden the layout ordered by the call site in a manner that is probably going to suddenly design. This objective can be accomplished in two stages:

I. Merging a tree-based arrangement

The motivation behind the initial step is to coordinate each phone territory a bunch of trees addressing the best qualities that can be moved to the API infusion. The fixed sentence structure tree (PAST) of the API infusion site layout is non-cyclic, associated, situated chart (N, E) where the N set of hubs address the just subtree of which the solitary known root hub is known, (N, E) is a medication that can be ventured into a legitimate AST for language endorsed API.

To join PAST with a layout, the technique plays out the accompanying advances. At first, it builds up the layout by filling its obscure parts with straightforward string esteems realizing it is organized. A bunch of realized benevolent qualities ought to be characterized just a single time for every infusion API. This technique completely endeavors to give a potential delivering of these qualities to obscure upsides of the layout. From that point forward, each resulting string is furnished with a language alternate route, e.g., JavaScript or Bash parser. Furthermore, just if the string is acknowledged as an endorsed individual from the language, that methodology will keep the AST drove into a substantial arrangement of AST models. Given the authority illustration of AST format, the subsequent stage is to consolidate them all into one PAST. Until now, this technique distinguishes basic hubs across all ASTs, i.e., hubs shared by all ASTs however with an alternate subtree for all ASTs. At first, ASTs are allotted to their foundations, which should be predictable on the grounds that all ASTs are in a similar language. From there on, this technique at the same time cuts all the ASTs in a profound manner and looks for nlc hubs with kids that contrast at any rate two ASTs. Each such hub is a standard hub. At long last, this strategy makes

solitary PAST that contains shared segments of all ASTs and where standard hubs are not continually broadened and structure a fixed Nsub.

ii. Evaluating the Period of Opposition to the Policy

The arrangement of PASTs coordinated with the call site is the premise of the approach utilized by our cycles for each link passed to the call site. We apply this authorization by reworking the fundamental JavaScript code to the call site. At the point when the measure of working time contacts the reworked telephone region, the working time measure scraps it into AST and matches it to the PASTs of the call site. During the examination, the arrangement utilizes two designs:

- The quantity of working hours ought to be a reasonable augmentation of the exhibition of any current PAST. Such extension gives the subtree a certain subtree so the arising tree (I) is lawful as far as language and (ii) efficiently compares to the AST of the functioning time esteem.
- PAST development of the PAST hub is restricted to contain just AST hubs from a predefined set of secure hub types. A bunch of secure hub types is characterized once per language, that is, free of a specific call site and its PAST. With the Shell orders executed, this technique just considers hubs addressing the content as secure. In the JavaScript-encoded JavaScript code, the technique permits a wide range of AST hubs that happen in JSON code, to be specific, messages, records, structures, arrangement articulations, object articulations, part articulations, and articulation explanations. Fundamentally, none of these sorts of hubs permit the assailant to enter noxious code.

C) Implementation

Exact investigation: Using static examination in Java, we expand on Google Closed Compiler4. To deal with circles and copies, static information examination restricts the occasions an assertion is refreshed while adding a particular information stream factor to ten. At the point when we apply predictable examination to a module, we force a one-minute end per module. In the wake of

finishing the module investigation, the execution composes a bunch of formats for each call site in a content record that will be utilized for dynamic examination.

Working time examination: We remember dynamic investigation for JavaScript. Prior to making the module, the PAST examines every one of the calls previously dependent on the layouts gathered by static investigation. While utilizing the module, the investigation catches every one of the calls to make and eliminate and eliminates the links moved to this capacity to be tried against our strategy. Automatic Shipping: To make SYNODE deployment as easy as possible without relying on third-party code providers, we suggest a way for a module engineer or system administrator to add an input script to an installed application as npm module .

3.3.3 T.J. Watson Libraries for Analysis (WALA)

Node.js applications utilize outsider JavaScript modules as reliant upon the Community to share and send these modules through the Node Package Manager (NPM) archive. NPM tallies roughly 500,000 bundles and accordingly, is the biggest language-explicit bundle administrator. The reliance esteem, which is typically the outsider reliance found in NPM, for the standard Node.js application can without much of a stretch surpass a couple hundred. (Reliance may apply to different conditions.) The incorporation of all conditions is a necessity of the framework. Node.js reliance has a similar degree of natural access as the primary application.

Doing the assaults talked about in this paper requires the aggressor to effectively identify its perilous reliance on the casualties' projects utilizing for example Reliance circulation on NPM or changing over conditions found in NPM. The reason for the assailant is to surpass the host demands, to oversee or spill information handled into applications, to hurt the working climate, or to contact different managers of the working environment.

	Scope	Aims for	Source	Weaknesses
Global Leakage	Global	Data	JavaScript	Global variable
Global Manipulation	Global	Data & Control	JavaScript	Global variable and monkey-patching
Local Manipulation	Local	Data & Control	Node.js	Loaded module cache and monkey-patching
Dependency Tree Manipulation	Local	Data & Control	Node.js	Loaded module cache and monkey-patching

Fig 5 : Comparison of attacks

This method uses code analysis to identify that attack. The analysis needs to be applied to the nature of the refusal to make claims that involve dependency-based attacks. We used this approach and created a TJ static code analysis. Watson Analysis Libraries (WALA) to identify trust-based attacks.

DEPENDENCY-BASED ATTACKS

There are 3 JavaScript and Node.js feebleness based on which a total of 4 different attacks are possible. Firstly, addressing the weaknesses:

1. Global Variables: Many JavaScript functions are used in the form of functions and variables stored across the globe. This global rating is shared among all modules of a particular Node.js program which includes their dependencies. This sharing of land diversity is considered a bad practice as it can be used for external conversions.

2. Monkey catching: The dynamic modification of classes and tasks during launch in JavaScript is known as monkey integration tasks. In this way, another task can catch the first monkey and write it over and the attached work will be constantly requested.

Additionally, in JavaScript we cannot determine whether a function has a baboon tag or not. Below is a basic example of a baboon. The MyClass class task is drawn with a function in the function performMonkeyPatch.

```

1  function MyClass() {};
2  MyClass.prototype.someFunction = function () {
3      // Initial implementation
4  };
5
6  function performMonkeyPatch() {
7      var originalFunction = MyClass.prototype.someFunction;
8      MyClass.prototype.someFunction = function () {
9          // New monkey-patched implementation
10         // originalFunction can be invoked here
11     };
12 }

```

Let the function of someFunction described in lines 2 to 4. The performMonkeyPatch function is described in 6 to 12 lines of monkey dot operation First, the first application is copied to the original originalFunction variable. After that, the Function function is rewritten. As long as the performMonkeyPatch function has not been performed, a call to someFunction will request the actual function. Once the monkey catch-up statements have been implemented, the announced work will be requested instead.

3. Cache of loaded modules: In order to use the functions and variations of the application, the module must be loaded explicitly on the scales of the application using the JavaScript require function. This function comes with archive material loaded with loading modules and their exportable properties. As a global variable, this cache is redistributed between all modules in a specific application of Node.js and any module may be liable for fraudulent application cache content.

This paper identified 4 dependence attacks due to the 3 weaknesses mentioned above.

1. Global Leakage: There is a possibility that dependence can reach global volatility and reward their content. For example, values for all natural variables can be disclosed on an attack-controlled server by writing only one line of code. This can lead to the leaking of sensitive information and also this evidence can be misused to commit wrongdoing using the app.

Captions of code reward the environmental dynamics stored in the process.env:

```

1 //definition of the leak method
2 function leak(data) { ... }
3
4 //call of the method
5 leak(process.env);

```

2.Global Management: This type of attack can exploit dynamics or functionality that can be achieved globally to alter application performance. This change in behavior can be made using a pair of monkeys.

The manifestation of the Global Manipulation Attack that transforms the performance of the guarantor's dependence:

```

1 > var validator = require('validator');
2 > validator.isEmail('name@example.de"␣hello="world');
3 false
4 > require('./index.js'); //Contains the Manip. attack
5 > validator.isEmail('name@example.de"␣hello="world');
6 true
7
8 //Content of ./index.js
9 var validator = require('validator');
10 var isEmail = validator.isEmail;
11 validator.isEmail = function(val){
12     if (val==='de"␣hello="world') return true;
13     else return isEmail.apply(this, arguments);
14 };

```

The application transfers an affirmation library on line 1. Reliance checks the client contribution to line 2; The aftereffect of this solicitation is bogus as demonstrated in line 3. The vindictive code found in the record ./index.js (content lines 8 to 14) is transferred to line 4, which modifies the usefulness of isEmail () work. In line 5, a similar link is tried once more. Meanwhile, the invalid email address is viewed as a substantial email address as shown the genuine return esteem on line.

Note that the record ./index.js has been transferred to utilize works other than the isEmail () work. In any case, this capacity supersedes the first capacity (Email). This alteration of the validator library clearly could influence security delicate capacities, for example, input-empowered capacities contrasted with XSS or SQL Injections.

3.Location Fraud: The variables in Node.js may have a wide range of location found within the caller context but not outside. However, there are weaknesses in the Node.js known as loaded

module cache and this can be used to manipulate other dependencies so that the control flow of a given system can be taken over by the attacker.

4.Reliable tree management: There are two variations of this attack. The first is about loading the dangerous dependency instead of the victim's dependence. Now, any given system modules that load the victim's dependence, will use the risky dependence found in the repository instead of the real one. The second uses the resolution and calls the victim's dependence function and then changes it. Now, this modified method is stored in the dependency file.

This type of attack can be used to bypass security controls. For example, the behavior of a module that addresses risk dependence may be modified in such a way that the environment will never be regenerated with the latest versions of dependence.

Example of Basic Dependency Tree Manipulation (Variation 1):

```
1  require('./malicious-lib');
2  require.cache[require.resolve('victim-lib')]
3  = require.cache[require.resolve('./malicious-lib')];
```

Attack of Tree Dependency Tree Manipulation to manage dependence that removes a function instead of an object (Variation 2):

```
1  var original = require('victim-lib');
2  require.cache[require.resolve('victim-lib')].exports
3  = function () {
4      // Monkey-patch function body
5      return original.apply(this, arguments);
6  };
```

You may use a temporary reserve of the required task directly rather than defrauding the retrieved object of the required function, such as in a previous attack. This attack uses the `require.resolve` function and gives the file name that uses the attack module as a parameter.

Static code analyses for dependency-based attacks:

Stable code analysis is used to detect attacks using the TJ code analysis tool. Watson Update Library (WALA). It is a fast and efficient analysis framework available for JavaScript source code analysis. The analysis tool currently has several limitations including unlimited application size

limit (maximum 9 MB), depending on content binary code, etc. The framework transforms the source code to an Intermediate Representation (IR) form that we use in the analysis.

A brief description of the code analysis to detect 4 attacks follows:

1. Global Leakage: The measure for recognizing this assault is: call the break work where one of the boundaries is earth changes. The issue with this assault is the trouble of tallying every single compensating movement - this is simple on account of Android as Apps need to call certain application administrations.

We are utilizing the HTTP convention execution to act as an illustration of a hole strategy. Our investigation analyzes the utilization of the HTTP application object to spill land variety esteems. The investigation recognizes the brings in the HTTP demand design. We use information stream examination to decide if the strategy boundary is connected to topographical variety.

2. Global Manipulation: There are two sorts of assaults: (1) the world's item is changed or (2) any of its properties are modified. The main assortments are not difficult to spot. In WALA, abrogate worldwide fluctuation is addressed by a solitary IR AstGlobalWrite order. On the off chance that the IR order is AstGlobalWrite type and the variable is an inherent variety inside JavaScript, there is an assault identification. Two methodologies require the satisfaction of the subsequent variety: the variable material is composed and that the variable is associated with one of the 32 worldwide JavaScript factors. The investigation distinguishes all AstGlobalRead orders (IR) for 32 forms and spreads them over the module to recognize the associated factors. Then, the IR JavaScriptPropertyWrite guidelines utilizing the factors from this rundown are determined.

3. Local Manipulation: The techniques for distinguishing proof of assaults are: reliance stacked utilizing a capacity requires the size of the module and is utilized in the extent of different modules. To discover such an assault, we start by distinguishing every one of the requests for work searchers, expecting that work titles might be a hindrance to different positions. Then, information

examination is performed ridiculous things and the administration of the articles or their properties. that is, we adhere to the guidelines of IR JavaScriptPropertWrite and SSAPutInstruction.

4. Reliance tree manipulation: The recognizable proof mode for an assault is: the switch interface is utilized on a size of another module without stacking or inclining module is supplanted by another record. To get such an assault, all JavaScript articulation demands. Things returned by this solicitation are followed utilizing investigation information examination to recognize misrepresentation.

Attack	# cases	Global Leak		Global Man.		Local Man		Dep. Tree Man.	
		#	Time	#	Time	#	Time	#	Time
Global Leak	3	2	133 s	147	138 s	1	5 s	0	0.484 s
Global Man.	9	0	0.063 s	2	0.519 s	0	0.019 s	0	0.027 s
Local Man.	4	0	0.430 s	0	2.611 s	1	0.081 s	0	0.067 s
Dep. Tree Man.	4	0	0.364 s	0	2.676 s	0	0.060 s	1	0.069 s

Fig 8: Assessment of the code analysis

The current analysis has several limitations including unlimited application size limit (Max. 9 MB.), Dependence containing binary code, and the use of certain functions such as eval. Future work will address these issues.

Chapter-4

PERFORMANCE ANALYSIS

The performance analysis includes test cases run on the developed system and results & outputs at various stages according to various inputs.

Our analysis has shown that while the platform is ready for use of the product in some applications but still needs further development and testing, the safety of critical applications should be avoided. The reasons are that the platform is not mature and has some security vulnerabilities. Many of the problems we have highlighted can be avoided with a security alert. However, errors are more common in Node.js applications because programmers do not have much experience in writing JavaScript applications that last longer on the server side. As the project progresses, speaker security is expected to improve over time.

Proposed model covers many aspects of Node.js weakness. It not only statically analyzes values passed to injection APIs but also analyzes dependency based attacks. The technique can be deployed automatically as part of module installation, further easing the process of deployment.

After successful complete implementation , we shall be checking outputs at each and every stage and then Performance Analysis section would be updated accordingly.

Chapter-5

CONCLUSIONS

This chapter contains brief description of what we have done and what can be added in the future.

5.1 Conclusion

So far, we have thoroughly studied the research papers and have discovered the critical vulnerabilities that may affect a web application based on Node.js platform and designed a model to overcome those vulnerabilities.

As the model is in initial stages, it needs to be tested on a several codes in different environments to make it resilient, robust and error free. The model covers both the bases of attacks, injection attacks and dependency based attacks.

Given the time, proposed model is in initial phase and will be ready for problem solving in the near future very soon.

Our project is developed partially and complete implementation will take place very soon. Moreover, the project will be concluded after the implementation.

5.2 Future Scope

Till now we have worked upon the methods which are already being used in the industry and we have become familiar with the model structure.

The platform's design decisions affect the security of its applications. This project outlines several possible security pitfalls to be aware of when using Node.js platform and server-side JavaScript. It will also address several other vulnerabilities and will contribute towards developing and configuring resilient web applications on the Node.js platform.

Next, we will be implementing our own developed Security Assessment tool by working and finding out better implementation techniques for the same. We will make the design more efficient both in terms of cost and user experience and make it much more user friendly. Complete implementation will be done in the near future.

5.3 Applications Contributions

Once the project is implemented in its full glory, we would like to contribute our own built tool to the Node.js platform so that it can use its full benefits towards securing it from security breaches.

REFERENCES

- (i) <https://nodejs.org>

- (ii) <https://en.wikipedia.org>

- (iii) <https://www.w3schools.com>

- (iv) <https://www.netguru.com/blog/top-companies-used-nodejs-production>

- (v) <https://scholar.google.com>

- (vi) C.-A. Staicu, M. Pradel, and B. Livshits. “Understanding and Automatically preventing injection attacks on Node.js”. Technical Report TUD-CS-2016-14663, TU Darmstadt, Department of Computer Science, 2016.

- (vii) A. Ojamaa and K. D  una, "Assessing the security of Node.js platform," 2012 International Conference for Internet Technology and Secured Transactions, London, 2012, pp. 348-355.

- (viii) B. Pfretzschner and L. b. Othmane, “Identification of Dependency-Based Attacks on Node.js” ARES '17: Proceedings of the 12th International Conference on Availability, Reliability and Security, August 2017. Article No. 68, Page 1-6. <https://doi.org/10.1145/3098954.3120928>