JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING

TERM PAPER MID EVALUATION REPORT

TEACHER SUPERVISOR :
Dr. Sangeeta Mittal, Assistant Professor (Senior Grade)

EVALUATION PANEL :
Dr. K Vimal Kumar, Assistant Professor (Senior Grade)
Mr. Mahendra Gurve, Assistant Professor

SUBMITTED BY:

A Malik	**16103145**	Batch B7
Ujjwal Sharma	**16103136**	Batch B2
Darsh Macwan	**16103275**	Batch B3

Submitted in partial fulfillment of the degree of Bachelor of Technology, Department of Computer Science and Engineering, Jaypee Institute of Information Technology, Sector 62, Noida.

INTRODUCTION AND RATIONALE

JavaScript often abbreviated as **JS**, is a high-level, interpreted scripting language that conforms to the ECMAScript specification. JavaScript has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. JavaScript is one of the core technologies of the world wide web. It enables interactive pages and is an essential part of web applications. In enabling various functions JavaScript relies heavily on JavaScript Resources. The direct client side inclusion of these cross origin resources in Web Applications has become an excessively pervasive practice to consume third-party services and to utilize externally provided code libraries.

Even though this inclusion increases the functionality manyfold, there are several downsides to it. Most of the issues arise as a result of the access that is granted to these third party services/libraries to access the resources essential for their functioning. The problem is that such a code runs in the same context and with the same privileges as the first party code. This gives rise to major security concerns. For starters, all potential security problems in this third party code now directly affect the site that included these. Secondly, these third party libraries/services can be used as gateways to mount possible attacks on the sites that use these.

This problem is not something new and has been known to developers for quite sometime now. However the problem lies in the solution to this problem. All the methods proposed up till today to solve this issue were methods involving either a very big computational or time overhead or methods that restricted the functionality of the applications. This problem was owed partly to the fact that it was logistically very problematic to scan each and every module being uploaded and all the updates there after and keep a track of any vulnerability therein and partly because the nature of web browsers and the way they function makes it a difficult task to perform and achieve a real time security scan at runtime. This is owed to the fact that the parsers that convert the user codes from ECMAScripts (JavaScript) for syntactic analysis have their own limitations in form of performance and functionality. These limitations are majorly credited to the language in which these parsers are often written. The fact remains that the programming language plays a major role in the way a program functions as well as it heavily affects the performance parameters of a program and hence in a way restricts the program from utilising its full potential.

Upon analysis we found this to be the case with our problem statement. We as a group have been associated with applications that make use of NodeJS and realise the importance of modules and the diversity they bring to this technology, however, we have also realised that in its present state NodeJS and all technologies using third party JavaScripts are highly vulnerable. Up until now this security concern was seen as a trade off to achieve the desired functionality but with the recent rise of light and fast Web Assembly languages, there is hope that these can be used to complement ECMAScripts and rid us of the issues associated with JavaScript implementations of resources like parsers and analysers.

This study is an attempt to understand the working of Web Browsers and the technologies that utilize these web browsers along with third party modules to enhance functionality of applications. Considering that the security of our resources is a legitimate concern, through this study, we aim to identify the ways in which we can mitigate the concerns by making the smallest trade off possible. For the purpose of intensive study and circumstantial understanding, we have restricted ourselves to Rust for coding purposes. The reason for doing so has been illustrated further. The practical implementation of this research study is being used in our project titled **"Security Model for ECMAScript runtimes."**

REFERENCED LITERATURE

Considering the fact that 'Security' is a complex domain and its implementation is dependent on a wholesome understanding of the structure in which it works. To understand and segment our problem statement, we have divided the study into certain segments to make the research process simpler and more understandable to us, as well as, to anybody who takes interest in this research project. Also, since our study focuses a lot on the recent trends in technology, there is lack of recent academic literature on the same. So, for the purpose of gathering relevant and useful information, we have taken help of books, journal papers and white papers alike. Due care has been taken that the Literature used adds value to the research and that it is procured from authentic and authoritative sources. The major sections in which we have classified our literature are as follows:

1. Classic Security Concepts
2. Identification and Previously Employed Approaches
3. Platform and Programming Language
4. Contemporary work going on in this domain

Majority of the papers selected are very recent publications, in or after 2015, to keep abreast with new technology trends. We have also used a couple of books to fill the gap between the understanding of the academia and the industry. The list of papers and books used for the purpose of study are attached as a list below.

S. NO	PAPER NAME AND PUBLICATION	AUTHOR AND YEAR OF PUBLICATION	OBJECTIVE
1]	Mapping software faults with web security vulnerabilities - IEEE - 2008	Jose Fonseca, Marco Vieira (CISUC, University of Coimbra, Portugal)	A field study analyzing security patches of widely used web applications.
2]	Control-Flow Analysis of Higher-Order Languages - Research Thesis - 1991	Olin Shivers (School of Computer Science Carnegie Mellon University)	Techniques for recovering the control-flow graph of a program at compile time
3]	CrowdFlow: Efficient Information Flow Security - White paper - 2015	Christoph Kerschbaumer , Eric Hennigan, Per Larsen, Stefan Brunthaler, and Michael Franz (University of California, Irvine, USA)	An approach to information flow security probabilistic-ally switches between two JavaScript execution modes.
4]	A Flexible containment mechanism for executing untrusted code - USENIX Security Symposium - 2002	David S. Peterson, Matt Bishop, and Raju Pandey (Department of Computer Science, University of California, Davis)	A detailed analysis of the options available to designers of sandboxing mechanisms

5]	XSS-SAFE: A Server-Side Approach to Detect and Mitigate Cross-Site Scripting (XSS) Attacks in JavaScript Code - Springer - 2015	Shashank Gupta, B. B. Gupta (King Fahd University of Petroleum & Minerals)	Exploring the XSS vulnerability and its solution approaches
6]	Native Client: A Sandbox for Portable, Untrusted x86 Native Code - IEEE - 2009	Bennet Yee, David Sehr, Gregory Dardyk, J. Bradley Chen, Robert Muth, Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar (Google Inc.)	A sandbox for untrusted native code. It aims to give browser-based applications the computational performance of native applications without compromising safety.
7]	The JavaScript and Web Assembly Function Analysis to Improve Performance of Web Application - IJAST- 2018	Jin-Tae Park , Hyun-Gook Kim and Il-Young Moon (KoreaTech University, Republic of Korea)	Examining application development through performance comparison when code is operated with web assembly and JavaScript.
8]	js.rs – A Rustic JavaScript Interpreter - white paper - 2015	Terry Sun and Sam Rossi (University of Pennsylvania)	A prototype server-side JavaScript interpreter in Rust
9]	Small World with High Risks: A Study of Security Threats in the npm Ecosystem - ACM - 2019	Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny and Michael Pradel (Department of Computer Science TU Darmstadt)	Mitigation techniques, such as trusted maintainers and total first-party security, and analyze their potential effectiveness
10]	ScriptProtect: Mitigating Unsafe Third-Party JavaScript Practices - ACM - 2019	Marius Musch, Marius Steffens, Sebastian Roth , Ben Stock and Martin Johns	A nonintrusive transparent protective measure to address security issues introduced by external script resources
11]	Benchmarking Static Analysis Tools for Web Security - IEEE- 2018	Paulo Nunes , Iberia Medeiros, Jose C. Fonseca, Nuno Neves, Miguel Correia , and Marco Vieira	A benchmark for assessing and comparing static analysis tools in terms of their capability to detect security vulnerabilities.
12]	On the impact of security vulnerabilities in the npm package dependency network - ACM - 2018	Alexandre Decan, Tom Mens, Eleni Constantinou	an empirical study of nearly 400 security reports over a 6-year period in the npm dependency network containing over 610k JavaScript packages.

13]	Current Research and Open Problems in Attribute-Based Access Control - ACM - 2017	DANIEL SERVOS and SYLVIA L. OSBORN (University of Western Ontario)	Attribute based access control model
14]	A novel approach for enhancing performance of javascript engine for web applications	Ayrapetyan R.B., Gavrin E.A., Shitov A.N (Moscow, Russia)	approach for performance improvement of Web applications by ahead-of-time optimizations
15]	Writing parsers like it is 2017 - IEEE- 2017	Pierre Chifflier and Geoffroy Couprie	how to implement safe parsers in Rust and insert them in large projects.
16]	Accelerate JavaScript Applications by Cross-Compiling to WebAssembly - ACM - 2017	Micha Reiser and Luc Bläser (Department of Computer Science Rapperswil, Switzerland)	a cross-compiler that translates JavaScript/TypeScript to WebAssembly
17]	The Rust Programming Language - Book	Steve Klabnik and Carol Nichols	*The Rust Programming Language* is the official book on Rust
18]	Compilers: Principles, Techniques, and Tools, 2nd Edition	Alfred V. Aho(Columbia University), Monica S. Lam (Stanford University), Ravi Sethi (Avaya Labs) and Jeffrey D. Ullman(Stanford University)	Compilers: Principles, Techniques and Tools, known to professors, students, and developers worldwide as the "Dragon Book," of compiler principles

LITERATURE REVIEW

1. Title: Mapping software faults with web security vulnerabilities - IEEE - 2008

Critical Analysis:
The paper talks about Web applications that are typically developed with hard time constraints and are often deployed with critical software bugs, which are invisible at the time of deployment either due to bad development practices or intentionally, making them vulnerable to attacks. The paper further dwells into classification and knowledge of the typical software bugs that lead to security vulnerabilities. This paper presents a field study analyzingsome major security patches of widely used web applications. The vulnerabilities used are those most commonly listed on CVE and the analysis results are compared against other field studies on general software faults (i.e., faults not specifically related to security), showing that a subset of software fault types is related to security. Furthermore, the detailed analysis of the code of the patches shows that web application vulnerabilities result from software bugs affecting a restricted collection of statements or overuse of a system resource or unrestricted access to a particular system resource.

Shortcoming:
The paper fails to connect the identified security breaches to the core issue in the implementation, rather provides a walk around solution through patching as and when the need arises. It does not touch upon the need to tackle the underlying issue to loose monitoring of web based code.

2. Control-Flow Analysis of Higher-Order Languages - Research Thesis - 1991

Critical Analysis:
This doctoral thesis, though outdated talks about a very essential element of our problem statement.
It outlines how programs written in powerful, higher-order languages should run as fast as their C counterparts. They should, but they don't. A major reason for this is the level of optimisation applied to these two classes of languages. This paper highlights the importance of the programming language when developing a software or code. The fact is that javascript has come to be known as a very fast and indispensible language when it comes to web development but the fact remains that in certain conditions implementations like parsing and analysis, assembly languages still beat javascript and the recent rise of strong, robust and unambiguous assembly languages like Rust and Go has reinforced the findings of this thesis that says assembly languages are the way to go to achieve best case time complexities when working in a service dependent ecosystem.

Shortcomings:
This being a very old paper, it does not encompass a lot of work that has happened since. In the past couple of decades, a lot of alternate solutions to translating code into newer languages have sprung up and this thesis does not touch on those. Nevertheless, the thesis helped us form a strong foundation as to how control flow in applications has transformed over time and what are the characteristics that have not changed since.

3. **CrowdFlow: Efficient Information Flow Security - White paper - 2015**

Critical Analysis:
The paper talks about how modern web pages have become complex web applications that mash up scripts from different origins inside a single execution context in a user's browser. This execution scheme opens the door for attackers, too. Vulnerability studies consistently rank Cross Site Scripting (XSS) highest in the list of the most prevalent type of attacks on web applications. Attackers use XSS to

gain access to confidential user information. The paper goes on to discuss how all the approaches up untill now that prevents misappropriation of sensitive data introduce runtime overheads that make execution of JS code at least two to three times slower. The paper discussed this infeasibility due to which the industry will never adopt the information flow approach without a substantial reduction in this overhead. The solution that the paper introduces is one of distributing the tracking workload across all page visitors by probabilistically switching between two JavaScript execution modes.
Though the model does not talks about security concerns, it gave us a head start as to why we need to take in account the time and resource overhead when developing a feasible solution.

Shortcoming:
This is not a shortcoming of the paper per say but the fact that the approach discussed in the paper is of no significant use to us apart from the fact that it discusses at length the various ways in which time overhead increases and the ways in which it can possibly be tackled.

4. A Flexible containment mechanism for executing untrusted code - USENIX Security Symposium - 2002

Critical Analysis:
One of the popular security mechanism being used in certain web applications these days, This paper discusses the viability of a sandboxing mechanism for executing untrusted code and enforce established security policies.Although sandboxing techniques have individual strengths, they also have limitations that reduce the scope of their applicability. This paper gives a detailed analysis of the options available to designers of sandboxing mechanisms. It discusses the tradeoffs of various design choices, and proposes a sandboxing facility that combines the strengths of a wide variety of design alternatives.

Shortcoming:
Even though a widely used security mechanism that works for ensuring safety of applications, this technique also has a major flaw. The fact that third party modules when integrated with core modules, act on behalf of these core modules and even through the sandbox gain access to all the resources that the core module has access to. This design serves as an attack window for exploitation and this design flaw is not tackled by sandboxing methods.

5. **XSS-SAFE: A Server-Side Approach to Detect and Mitigate Cross-Site Scripting (XSS) Attacks in JavaScript Code - Springer - 2015**

Critical Analysis:
This paper discusses a framework known as XSS- SAFE which is a server-side automated framework for the detection and mitigation of XSS attacks. It is designed based on the idea of injecting the features of JavaScript and introduces an idea of injecting the sanitization routines in the source code of JavaScript to detect and mitigate the malicious injected XSS attack vectors.The sanitization routine checks for known and unknown XSS attacks with minimum false positives.

Shortcoming:
The framework is a damage control mechanism and does not provide preemptive security. It works on the detection after the XSS has been injected, hence making it a passive defender when it comes to security.

6. **Native Client: A Sandbox for Portable, Untrusted x86 Native Code - IEEE - 2009**

Critical Analysis:
This paper describes the design, implementation and evaluation of a Client, a sandbox for untrusted native code. It aims to give browser-based applications the computational performance of native applications without compromising safety. It uses software fault isolation and a secure runtime to direct system interaction and side effects through interfaces managed by Native Client. The paper proposes a client that handles untrusted modules from any web site with comparable safety to accepted systems such as JavaScript. An untrusted module may contain arbitrary code and data. The client proposes some rules that the modules have to conform to. If the modules don't conform to these rules, they are rejected by the system.

Shortcoming:
The problem with this implementation is that even though it enforces certain rules that the code must adhere to, it does not provide a defense in depth mechanism for modules that may conform to the convention and yet contain malicious or dangerous code.

7. **The JavaScript and Web Assembly Function Analysis to Improve Performance of Web Application - IJAST- 2018**

Critical Analysis:
This paper explores the differences between Native languages and web based languages and their performance. It discusses about the JIT (Just-In-Time) compiler that appeared in JavaScript that resulted in the dramatic development of the performance of the web. But with increasing interdependence on modules the problem of performance has risen once again.The paper discusses an ahead of time compiler implementation in web assembly meaning that you can develop an application using an existing native language (C type) and convert it to be operable on a browser equipped with a JavaScript based engine.The reason why web assemblies have better performance than JavaScript is that the code is concise, the parsing time is small, and it takes less time to compile and optimize.

8. **Js.rs – A Rustic JavaScript Interpreter - white paper - 2015**

Critical Analysis:
The paper is very closely related to our implementation of the ECMAScript parser in the RUST web assembly. JavaScript is an incredibly widespread language, running on virtually every modern computer and browser, and interpreters such as NodeJS allow JavaScript to be used as a server-side language. The paper talks about Js.rs which is a prototype server-side JavaScript interpreter in Rust, a new systems programming language for building programs with strong memory safety guarantees and speeds comparable to C++. The paper intends to demonstrate the viability of using Rust to implement JavaScript by implementing a core subset of language features.

9. **Small World with High Risks: A Study of Security Threats in the npm Ecosystem - ACM - 2019**

Critical Analysis:
The paper touches on an extremely important finding of the research process. The fact is that the popularity of JavaScript has lead to a large ecosystem of third-party packages available via the npm software package registry. The open nature of npm has boosted its growth, providing over 800,000 free and reusable software packages. Unfortunately, This open nature also causes security risks, as evidenced by recent incidents of single packages that broke or attacked software running on millions

of computers. This paper studies security risks for users of npm by systematically analyzing dependencies between packages, the maintainers responsible for these packages, and publicly reported security issues. Studying the potential for running vulnerable or malicious code due to third-party dependencies, we find that individual packages could impact large parts of the entire ecosystem.

10. ScriptProtect: Mitigating Unsafe Third-Party JavaScript Practices - ACM - 2019

Critical Analysis:
This paper proposes a framework to mitigate the security effects of using third party code in web applications. The direct client-side inclusion of cross-origin JavaScript resources in Web applications is a pervasive practice to consume third-party services and to utilize externally provided libraries. The downside of this practice is that such external code runs in the same context and with the same privileges as the first-party code. The paper develops into ScriptProtect, a nonintrusive transparent protective measure to address security issues introduced by external script resources. It automatically strips third-party code from the ability to conduct unsafe string-to-code conversions.

Shortcoming:
This implementation does not require changes to the browser however incurs a runtime overhead. The adaption of this standard will positively affect only about 25% of the systems as expected by the authors.

11. **Benchmarking Static Analysis Tools for Web Security - IEEE- 2018**

Critical Analysis:
Static analysis is an indispensible tool used by developers to search for vulnerabilities in the source code of web applications. However, distinct tools provide different results depending on factors such as the complexity of the code under analysis and the application scenario; thus, missing some of the vulnerabilities while reporting false problems. Benchmarks can be used to assess and compare different systems or components, however, existing benchmarks have strong representativeness limitations, disregarding the specificities of the environment, where the tools under benchmarking will be used. This paper proposed benchmarks for assessing and comparing static analysis tools in terms of their capability to detect security vulnerabilities. This paper provided us with valuable insight into the static analysis process that we shall be doing in a later stage in our implementation, to shortlist the vulnerabilities we wish to target.

12. **On the impact of security vulnerabilities in the npm package dependency network - ACM - 2018**

Critical Analysis:
The paper discusses various security vulnerabilities in open source software package libraries. It also talks about how it is a very tedious task to discover and fix vulnerabilities in packages. In addition, vulnerabilities may propagate to dependent packages, making them vulnerable too. This paper presents an empirical study of nearly 400 security reports over a 6-year period in the npm dependency network containing over 610k JavaScript packages. Taking into account the severity of vulnerabilities,they analyse how and when these vulnerabilities are discovered and fixed, and to which extent they affect other packages in the packaging ecosystem in presence of dependency constraints. They also provide guidelines for package maintainers and tool developers to improve the process of dealing with security issues.

Shortcomings:
Even though the paper discusses about vulnerabilities and discusses on ways to fix them, it does not highlight why these vulnerabilities arise in the first place and what are the most susceptible paths through which these vulnerabilities propagate.

13. **Current Research and Open Problems in Attribute-Based Access Control - ACM - 2017**

Critical Analysis:
This paper discusses Attribute-based access control (ABAC) as a promising alternative to traditional models of access control (i.e., discretionary access control (DAC), mandatory access control (MAC), and role-based access control (RBAC)) that is drawing attention in both recent academic literature and industry application.This paper provides a basic introduction to ABAC and a comprehensive review of recent research efforts toward developing formal models of ABAC.This paper is in line with our usage of restricted access policy to the module through specification of the resource it can utilize and thus provided valuable insight into a new and uprising trend in the technology.

Shortcoming:
This paper has ignored Issues like delegation, administration, auditability, scalability, hierarchical representations etc and left them to future work.

14. **A novel approach for enhancing performance of javascript engine for web applications**

Critical Analysis:
The paper discusses Web Assembly and its ability to boost performance of Web based code. The paper discusses the idea of ahead-of-time optimizations similar to what we plan on using in our model.

15. **Writing parsers like it is 2017 - IEEE- 2017**

Critical Analysis:
This paper emphasis on the use of RUST as a viable programming language for developing web parsers. It highlights the fact that a pragmatic solution of fixing vulnerabilities is to fix not only bugs, but classes of bugs. It uses Rust as a fast and safe language, and then using it creates a parser combinator.It discusses the advantages and difficulties of this solution, and presents cases of how to implement safe parsers and insert them in large projects. The implementation is provided as a set of parsers and projects in the Rust language.

16. **Accelerate JavaScript Applications by Cross-Compiling to WebAssembly - ACM - 2017**

Critical Analysis:
The paper discusses how cross compilation can help accelerate Javascript applications. Although the performance of today's JavaScript engines is sufficient, faster and more predictable runtimes are desired for performance-critical web code. Therefore, the paper presents Speedy.js, a cross-compiler that translates JavaScript/TypeScript to WebAssembly, a new standard for native execution supported by all major browsers. With this approach, the authors claim that they managed to make compute-intense web code up to four times faster, while reducing runtime fluctuations to the half.

17. **The Rust Programming Language - Book**
The Rust Programming Language is the official book on Rust: an open source systems programming language that helps you write faster, more reliable software. Rust offers control over low-level details

(such as memory usage) in combination with high-level ergonomics, eliminating the hassle traditionally associated with low-level languages.

The authors of The Rust Programming Language, members of the Rust Core Team, share their knowledge and experience to show you how to take full advantage of Rust’s features—from installation to creating robust and scalable programs. You’ll begin with basics like creating functions, choosing data types, and binding variables and then move on to more advanced concepts.

18. Compilers: Principles, Techniques, and Tools, 2nd Edition - Book

This is the dragon book for all compiler concepts and introduces the theory and practice of compiler design.It also covers topics like context-free grammars, fine state machines, and syntax-directed translation.