iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 7 **Issue:** III **Month of publication:** March 2019

DOI: http://doi.org/10.22214/ijraset.2019.3401

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 7 Issue III, Mar 2019- Available at www.ijraset.com

Enhancing Software Security for Salesforce Applications

Akshita D. Kadam[1], Seema Joshi[2]
[1]*Master of Computer Engineering (CyberSecurity), Graduate School of Engineering and Technology, GTU Campus*
[2]*Assistant Professor(Cyber Security), School of Engineering and Technology, GTU Campus*

Abstract:All software projects have one artifact in common: Source code. Code review from security standpoint ranks very high on the list of software security review best practices. Static Application Security Testing (SAST) should be implemented as a part of modern development process. Developing and deploying secure software is a challenging task. The code review from security standpoint is critical part in software security. The main aim is to benefit the Salesforce applications security review process for better performance and efficient findings for Cyber security practices. This approach will be modularized one that can facilitate developer to test Apex code against set of modules or classes or project as a whole as per the developer's concern and requirements. It will assist as developer's guide leveraging developer to implement secure coding practices without having prior paid automated scans for code review. Applications that intensify accuracy for better secure code analysis and reducing number of false positives. This approach will help salesforce application developer to pin point exact security issues during the development phase and help them to build secure salesforce application.

Keywords:Access Control, Accuracy, Apex code , Code review,Cyber security, False positives, , Intensify,Modularized,Salesforce, Salesforce application, Static code analysis, Source code, Source code analysis, Vulnerability.

I. INTRODUCTION

Salesforce is Cloud Service Provider which provides “Salesforce” as Platform as a Service to the customer where customer can deploy their customapplication on the Salesforce platform by writing application in the Apex Language along with Lightning/Visual force components using html/css and javascripts.You would need Salesforce login to compile and run the program written in Apex language, since it can only run on Salesforce platform which is on Cloud. If you compare web interface of typical web application Vs Salesforce application, both looks similar and vulnerabilities are similar (names are different but nature is same) but the methodology of detection vulnerability is different in both the application. From Static code analysis perspective, since both the languages are different, the methodology of detection is completely different. As per OWASP (The Open Web Application Security Project) Top 10 Vulnerabilities- “Broken Access Control” is one of the highly emerged vulnerability that needs to be checked and enforced in recent years

As per OWASP general term it is known as “Broken Access Control”, when we map this vulnerability for Salesforce platform the same term is referred to as is “CRUD/FLS and Sharing Violation”. As OWASP is the general consortium for variety of Web Application and therefore cannot be termed individually for different platforms. OWASP provides general terms for each type of category that is presented as in Web Application Context that we need to map as per our framework and platform schema. Below figure explains the mapping between OWASP and salesforce.

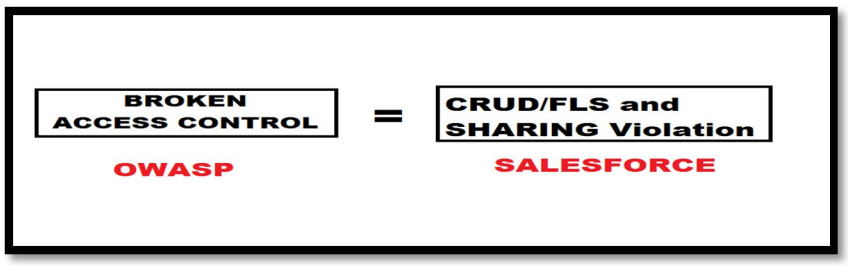


Fig. 1 Mapping of OWASP Vulnerability with Salesforce

Broken Access Control is general term that in terms of Salesforce can be mapped as CRUD (Create Read Update Delete)/FLS (Field Level Security, i.e. Sharing policy). CRUD/FLS and Sharing Implementation requires flaw pattern or logic for Enforcement that does not exist for any other API or Platform other than Apex

II. EXPERIMENTAL SETUP

This set-up is created in order to give a clear idea about how CRUD/FLS vulnerability arises in Salesforce Application and how it is mitigated:

Prerequisite:Salesforce Admin Account, Normal User Account, Internet.

A. Setup

1) As an admin I am creating "test" permission set for CRUD/FLS check which is the duty of Salesforce Adminto assign privileges to configure ACL(access control list) as per the schema of the individual application

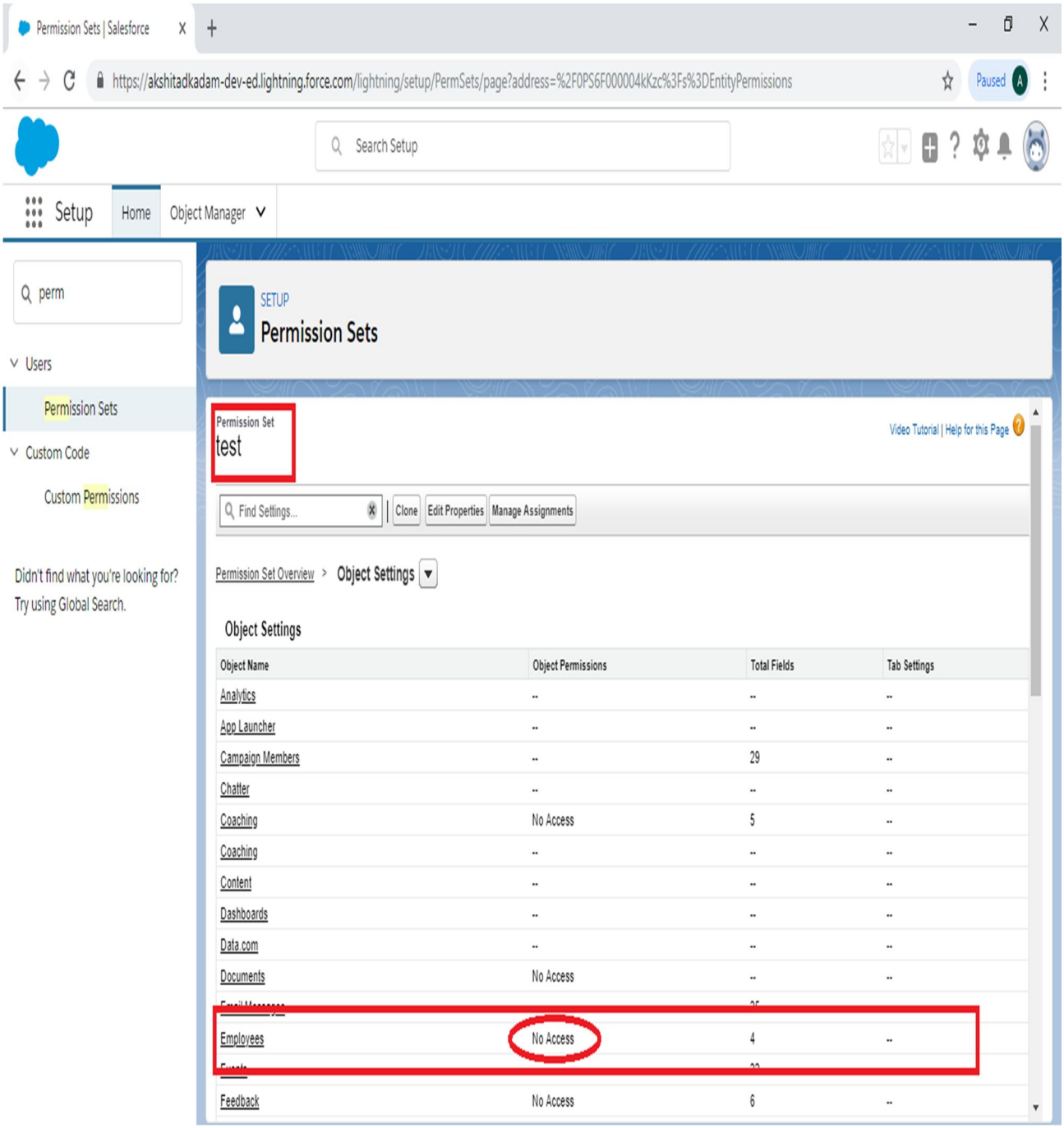


Fig. 2.1 Creating permission set: *test*

2) Then as an admin I am enforcing "no access" permission rights, i.e., no read write update delete can be performed on Employee object

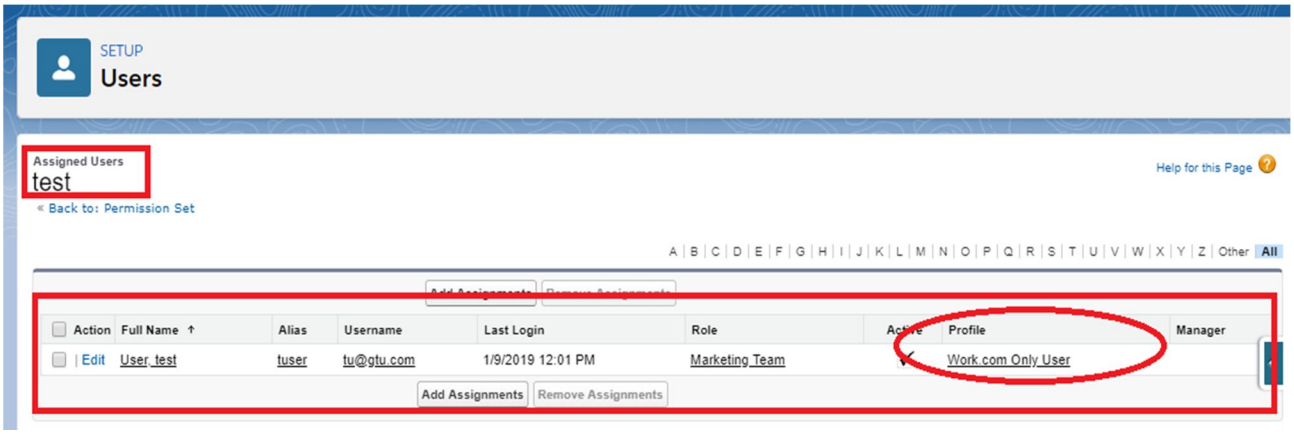


Fig. 2.2 Assigning test permission set access rights to Work.com Only (Normal user) profiles

Then "test" permission set created as an Admin is assigned to Work.com Only User Category of users which means that Work.com Only User cannot have any access on object Employee."Work.com OnlyUser" is one of profiles in Salesforce application that is manually selected for demonstration purpose and same experiment will give similar results if performed with other user profiles.

3) So, as per the privilege configuration assigned by an Admin to normal users that come under "Work.com Only User"profile should not be able to perform Create, Read, Update, or Delete operation on Employee object

4) Now we will perform a test case to verify that configuration assigned by Admin on Work.com User profiles works in case of Custom Salesforce Applications

B. Testing for CRUD/FLS

For testing application we created a demonstration test code which allows normal existing Work.com Only User to create otherWork.com User's Employee record and save it directly into the database server which is against Admin configuration privacy policy which bypasses write access denied bythe Salesforce Admin for salesforce domain. Another method was created which on reloading the application calls constructor that allows Work.com Only User to read records of other Employees which should not be displayed as user was assigned denied permission to read access on other user's employee records in test permission set..This test code was purely created to bypass access control privileges which should not bypassed as Admin had already configured permission for it and also to check how Apex code execution works by verifying whether permissions assigned by Admin are bypassed or respected.

On performing the testing process on application that was assigned no access permission set on Employee object following results were found: Though no access rights was assigned on "Employee" object there exists vulnerability in Salesforce application that "Work.com Only User" can still have access to Employee object type as Apex code runs in System context by default. Thus CIA triad (Confidentiality, Integrity and Availability) is breached.

This vulnerability exists in Apex because the code built runs in System's Context (Current User's authorization is ignored) rather than User's Context (Current User's authorizationis respected)

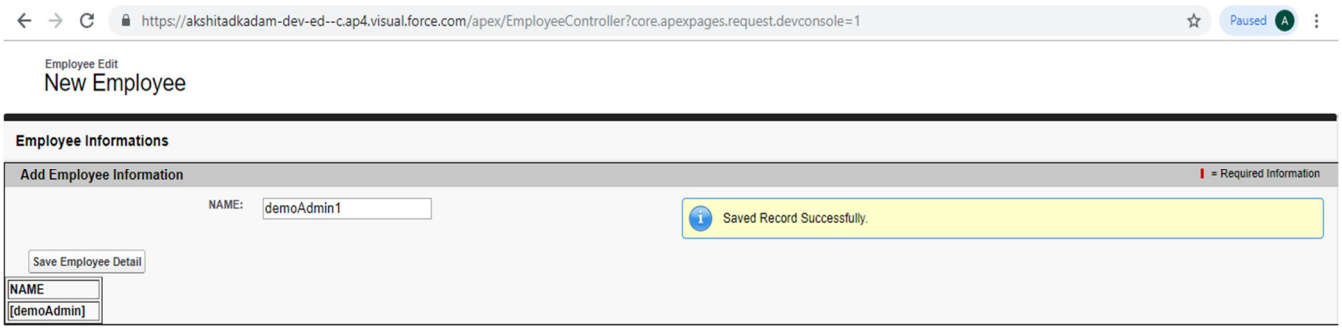


Fig. 2.3 Work.com User able to bypass write access denied by Admin

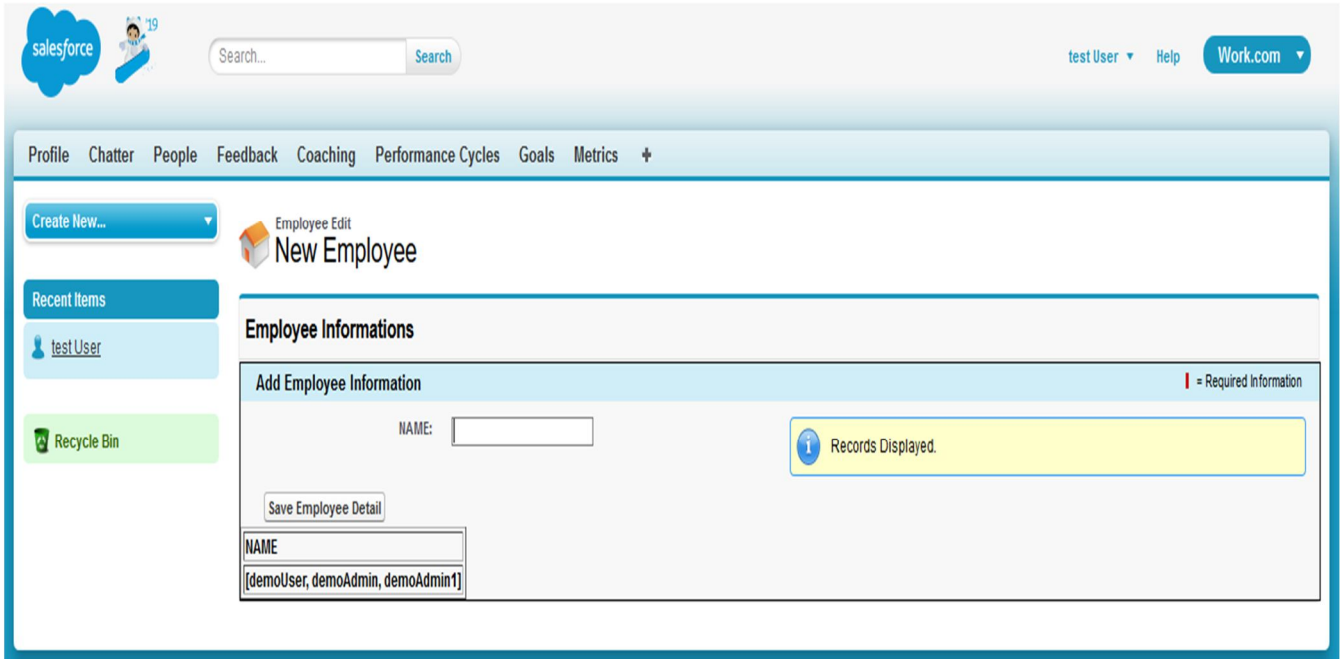


Fig. 2.4 Work.com User able to bypass read access denied by Admin

C. CRUD/FLS Enforcement

To overcome this vulnerability that exists in Salesforce Application, there is secure coding practices that need to be enforced and ensured:

For CRUD/FLS Enforcement

1) *On Create / Upsert: isCreateable,*
2) *Upsert: isCreateable and isUpdateable,*
3) *Read: isAccesible,*
4) *Update : isUpdateable,*
5) *Delete: isDeletable*
6) *class should be declared with "with sharing" in order to have field level access enforced on particular record*

We have demonstrated how to enforce Create and Read access check, other CRUD/FLS enforcement are done in similar manner.

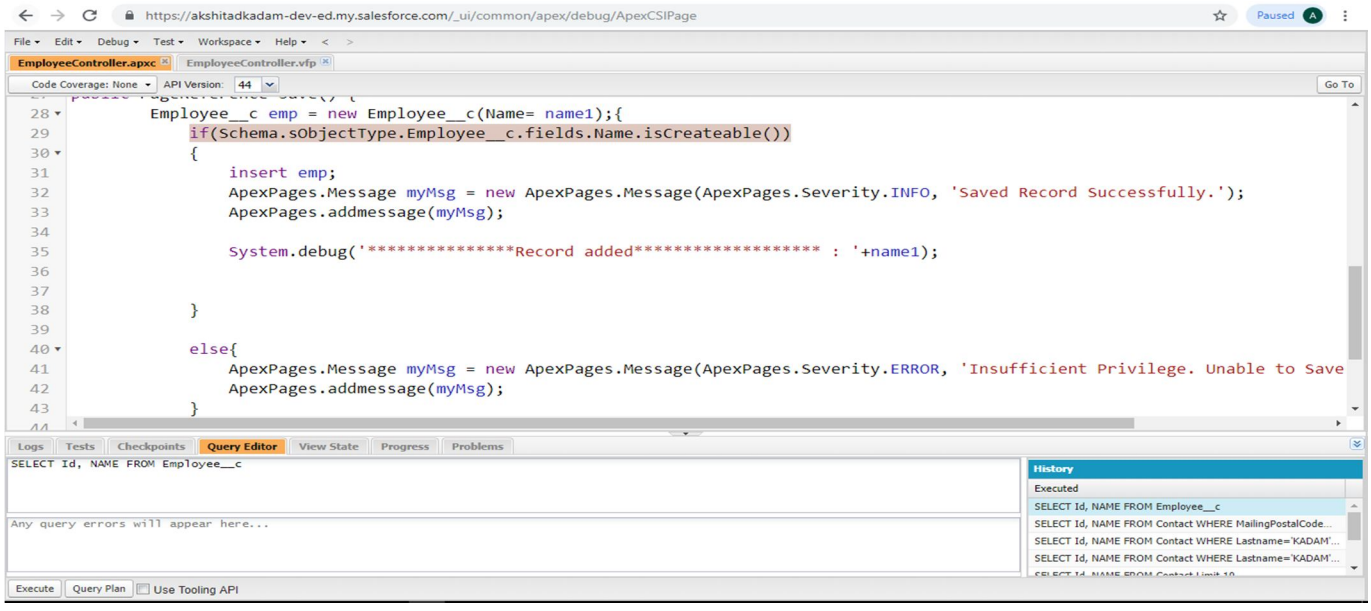


Fig. 2.5 Mitigation for CRUD writesaccess vulnerability

```
public with sharing class EmployeeController {

    public List<String> nameList = new List<String>();
    public List<Employee__c>  records;
    public List<String> records1 {get; set;}

    public String name1 {get;set;}

public EmployeeController(){
        if (Schema.sObjectType.Employee__c.fields.Name.isAccessible())    //Developer's Console
        {
            for(Employee__c records :[Select Name FROM Employee__c ]){
            nameList.add(records.Name);}
            records1 = nameList;
            System.debug('***************Record displayed******************');
            ApexPages.Message myMsg = new ApexPages.Message(ApexPages.Severity.INFO, 'Records Displayed.');
            ApexPages.addmessage(myMsg);
        }
        else{                                                  //User's Console
            ApexPages.Message myMsg = new ApexPages.Message(ApexPages.Severity.ERROR, 'Insufficient Privilege to View Records!!!'
            ApexPages.addmessage(myMsg);
            System.debug('You do not have permission');
        }
```

Fig. 2.6 Mitigation for CRUD read access vulnerability

D. After CRUD/FLS Enforcement

Current logged-in User (Work.com Only User) is not able to insert/ read another Employee records because he/she do not have privilege access rights for the same. Thus through CRUD/FLS enforcement privileges are not escalated and access control rights are maintained. Basically, in the typical web application, “broken access control” is checked by validating user session and user role at various levels such as business layer, repository layer, data access layer, etc. And in the Salesforce, it is to be checked in the Apex class during various CRUD operation APIs like insert, update, delete, upsert and selectperformed on Salesforce database server.

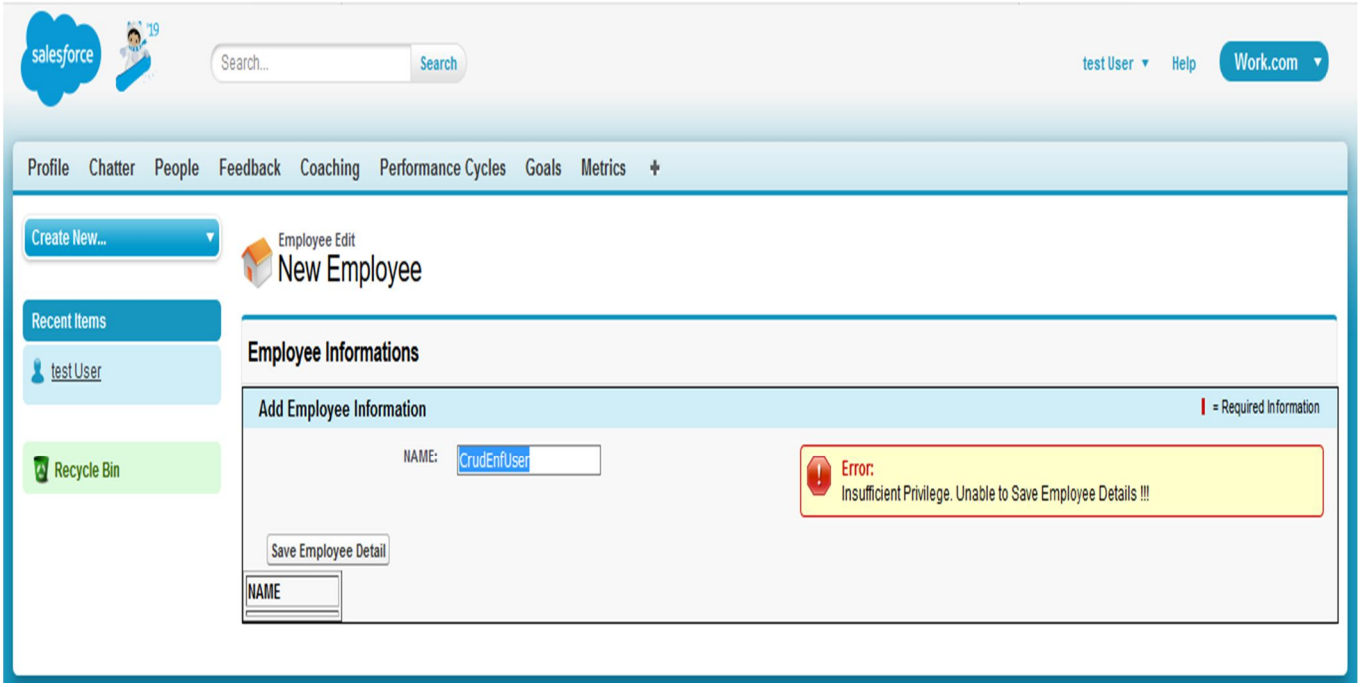


Fig. 2.7 CRUD Enforcement on write access

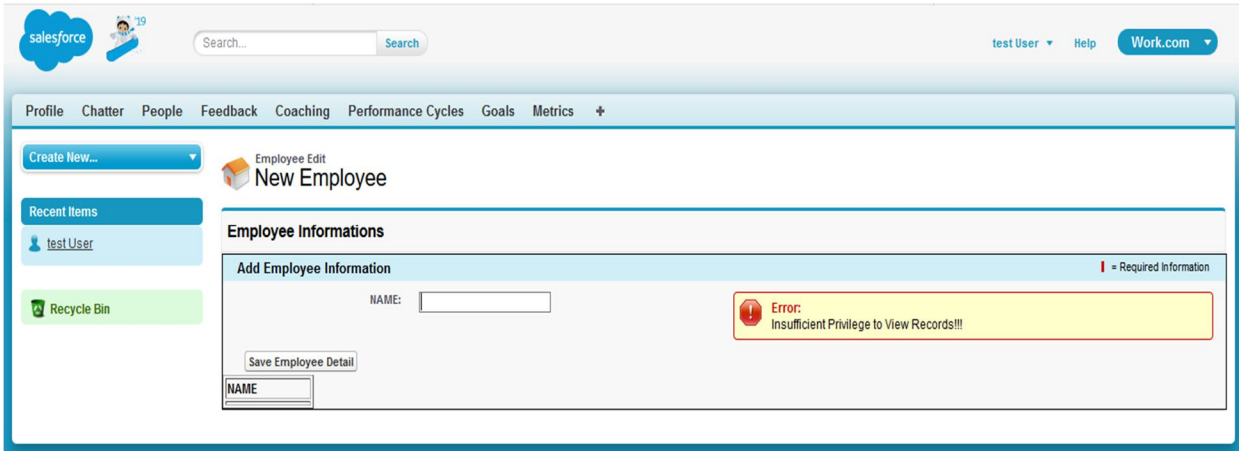


Fig. 2.8 CRUD Enforcement on read access

III. EXISTING SYSTEM

There are basically 2 types of secure code review techniques performed

1) *Source code :* where code is neither compiled nor executed
2) *Binary Code :* performs on converted byte code after compiling the code

For our research purpose we will be focussing on source code review technique where the code is static and never executed before testing and is simplyanalysed on its original code itself.

Automated SAST Technique

The ongoing Automated Static Application Security Tool makes use of an automated scanner that only concentrates on the keyword matching and does have support for advance techniques which are required in real-time case where code will be customised code that have customized method to performCRUD/FLS permission checks as per the business requirement. The following flowchart showcase on which parameters existing vulnerability testing tool works and in later topic we will discuss what are added parameters of our proposed model which makes it more efficient mechanism that provides sufficient logical parameters for accurate verification of whether source code vulnerability arises and exists till the final call from source to sink.

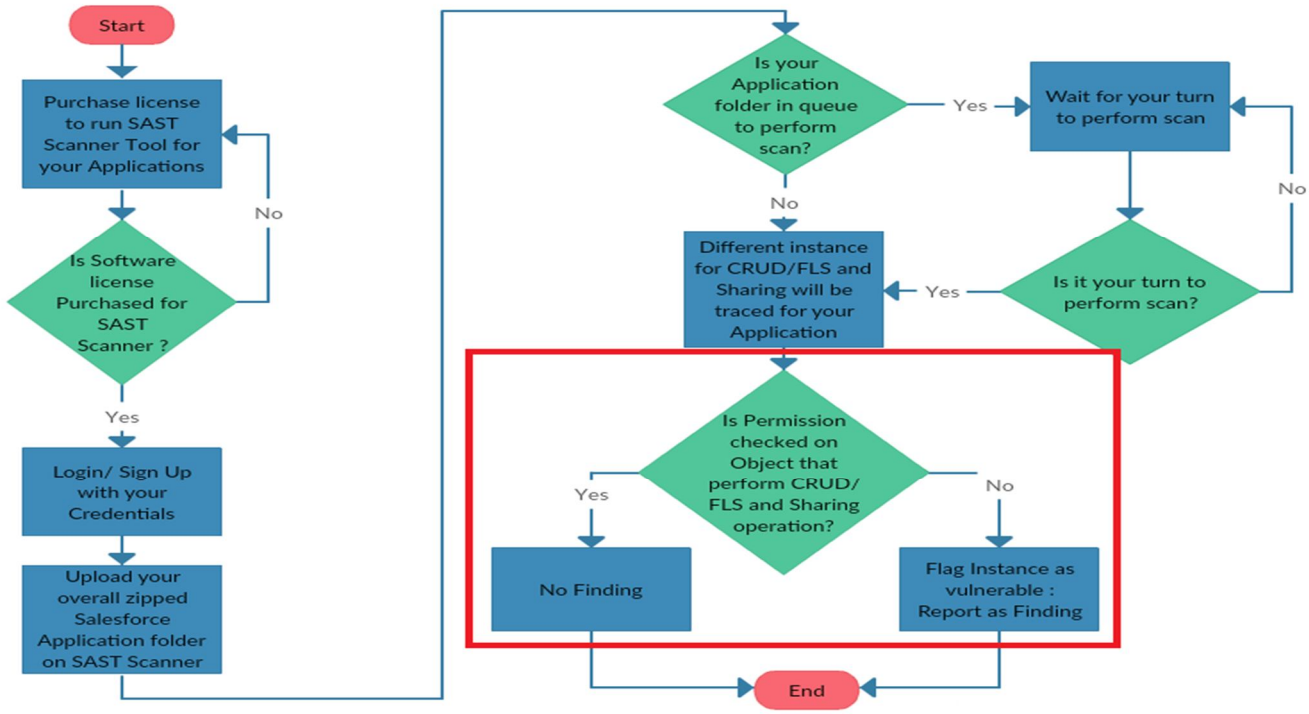


Fig. 3 Workflow of Existing Method of Salesforce CRUD/FLS Vulnerability Testing

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 7 Issue III, Mar 2019- Available at www.ijraset.com

IV. PROPOSED SYSTEM

The main idea behind the making of toolis:

A. Apex is on-demand language but the security perspective of coding still relies mostly on developers who is not yet practitioner that practices secure coding practices
B. Code review mechanism should be able to generate accurate results instead of more number of false positives which indirectly indicate more number of false negatives
C. A Source code review may prove efficient for one set of applications but at the same time cannot that useful for another set which simply means it depends on logic of patterns created that should accurately verify whether vulnerability can arise or not based on forward and backtracking to find out background of code snippet to be tested
D. SAST tool should be able to interpret the vulnerability such that non security person can easily implement and apply changes to the code

To fill the gap hybrid approach is used in which semi-automate scanner is used with different functionalities which requires combine effort of tool and the auditor. The main aim to have this approach is to: Reduce high number of false positive generated by automated scanners and to have efficient mechanism for secure code review practices
In this approach the step to be followed is explained in the following table:

TABLE I
STEPS FOR HYBRID APPROACH TECHNIQUE

1.	Open the tool and insert the packages(modules) that we want to scan
2.	Select the pattern file for which you want to trace instance for
3.	Click on run button to execute the scan
4.	Auditor will now verify one by one instance by hybrid approach
5.	There are different functionality to enhance the scan a. Trace : trace the keyword name (class/method/variable) in All files (you can also check based on specific folder or current file itself) b. HighlightAll : Forward trace and Back trace to find out object type of the associated instance that performs DML operation and maps it to see if the instance's object has called for permission checks on the object
6.	Auditor will back trace the instance in order to find out whether the instance detected by tool is already CRUD/FLS enforced. If Yes: Code review practice is already followed and there is no need to trace further. If No : Check if the instance is called in final page If Yes : FINDING If No : Check another class/method that traces the instance into the final page Yes : FINDING else NO FINDING
7.	For Sharing if class is declared without "sharing" keyword check if any DML operation performed. Check is similar to Step 6 for CRUD/FLS such that if the instance of the class that is declared without "sharing" is called in final page if Yes : FINDING else NO FINDING

Based on the above working functionalities workflow of the proposed system is explained below

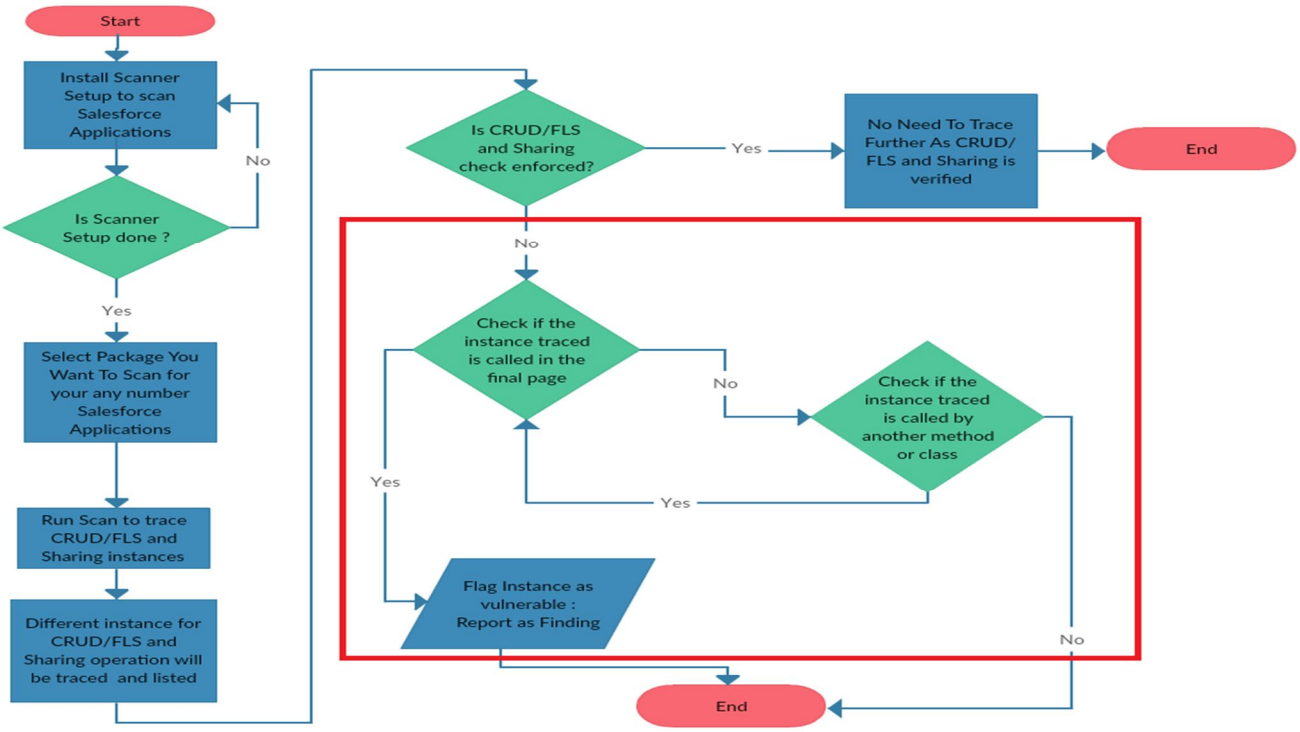


Fig. 4 Designed Proposed System Salesforce CRUD/FLS Vulnerability Testing

V. COMPARATIVE ANALYSIS

TABLE II

COMPARISION OF WORKINGS OF SAST AUTOMATED TOOL VS PROPOSED HYBRID APPROACH

SAST AUTOMATED SCANNER	PROPOSED SYSTEM
Cost is very high and paid in case if you want to scan your Apps	Free of Cost
Bound to perform limited number of scans as per your subscriptions and plans	Performs #m number of scans for #n Applications (# where m and n are finite integers)
Less accurate findings so is high number of false positives	More accurate findings of vulnerabilities so performance is high
It is purely based on native methods that check CRUD/FLS and Sharing Violation	Complex algorithm like Regex pattern match, backtracking and Forward tracking algorithm is applied
More time and effort required as Manual review is needed even after performing fully-automated scans	Lesser time is required as human auditor can recognize easily and trace custom methods used to perform CRUD/FLS custom checks
No interaction involved with auditor to verify in between the scans	Hybrid approach is followed involving mutual effort of auditor along with semi-automated algorithm patternsensuring whether the instance actually generates a loophole for vulnerability
It can only scan an entire project	It can scan class, (customized)modules as well as entire project

VI. HYPOTHSES

A dataset of 100 custom Salesforce Applications that were tested using SAST Automated Scanner as well as Proposed Hybrid Approachin order to draw line of conclusion about how much efficiency and performance is provided by individual approach. After successful testing of 100 customized salesforceapplicationsusing both approaches that were to be sent for Secure Code Review Process following results were drawn after comparing the result set of each approach individually for same applications:

TABLE III
COMPARISON TABLE OF FALSE POSITIVE REDUCTION RATIO

Category	**SAST Automated Scanner**	**Proposed System**
False Positive Ratio	*77% Accurate results* are generated as tool follows automated scanning without having deep analyses on code and hence generate more number of false positives than our proposed system. Automated scans provides 2/3 of static code vulnerability removal rate	*95% Accurate results* are generated as it follows semi-automated (hybrid) approach. Negligible number of false positives ratio. Thus generates accurate static code vulnerability removal rate *enhancing software security by 22% of false positive reduction ratio*compared to Automated SAST Scanner

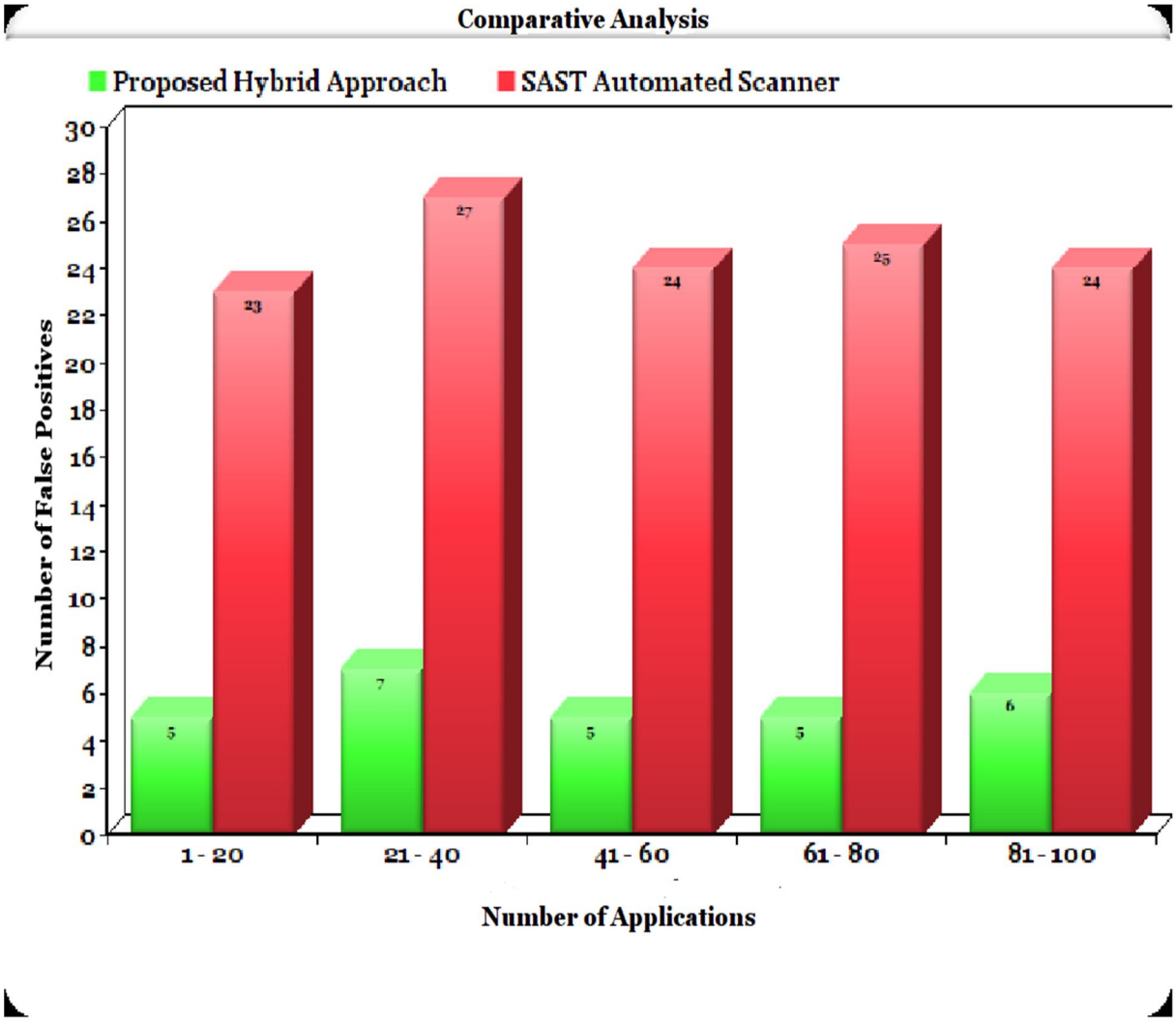


Fig. 6.1 Bar graph differentiating false positive ratio generated by Existing System Vs Proposed Hybrid System

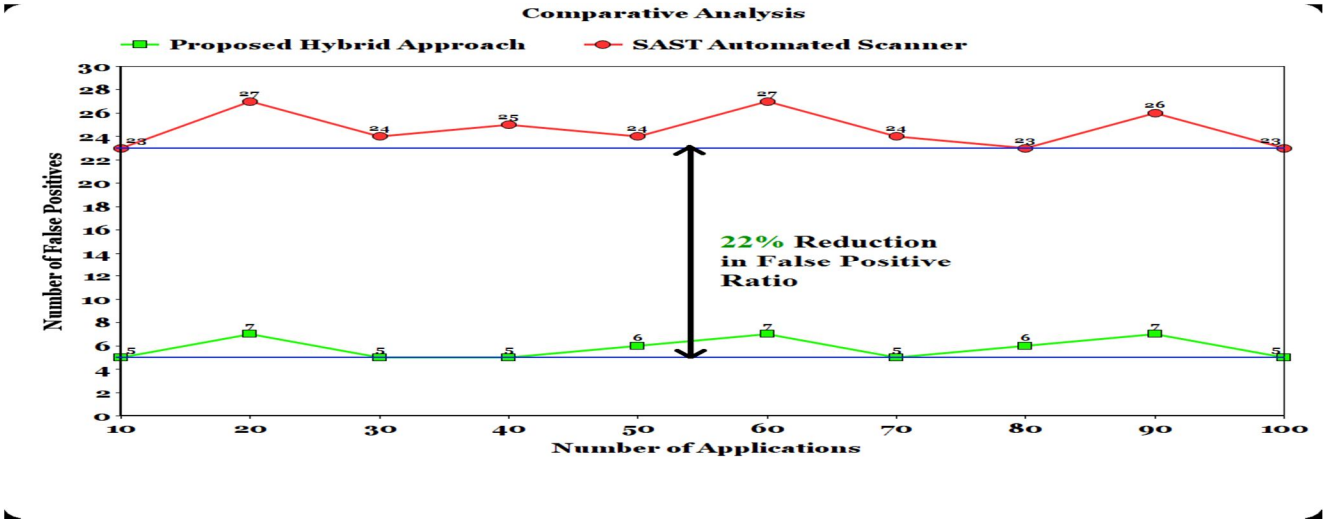


Fig. 6.2 Line graph demonstrating reduction in false positive ratio generated by Existing System by Proposed Hybrid System

VII. CONCLUSION

Detection methodology should be adopted such that it reduce the false positive ratio as a software team already using code reviews can gain a reasonable increase of 10% in defect removal, say moving from 85% to 95% , results in double the cost savings ($240,000 versus $120,000). The biggest is the cost multiplier of fixing bugs and security defects after a product is released (conservatively set at 5:1) [8]

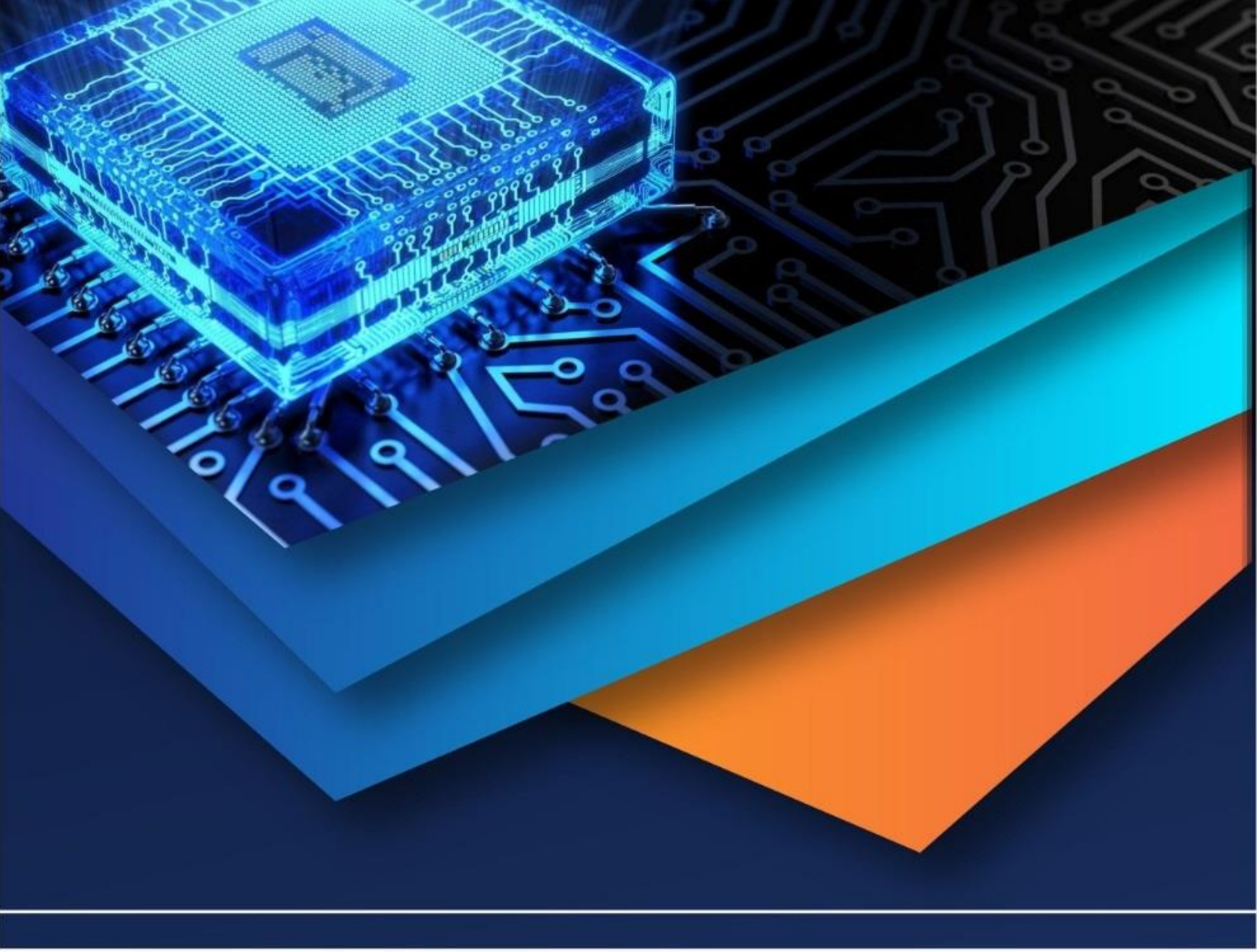
Too High a percentage of false positive leads to Number of false negatives [5].Reducing false positives are directly proportional to having reduce ratio of false negatives which enhances performance and provides least loopholes for vulnerabilities that arises from source code. Flagging a finding as vulnerable in code should be reported using efficient logical mechanism such that accuracy and performance are evaluated and information security is valued as per recent IBM study, the average cost for a stolen record raised 9% to $145 in 2014[11]

VIII. ACKNOWLEDGEMENT

I would like to thank my external guide Mr. Amish Shah for providing me enough knowledge for creating the experimental setup and guiding me throughout the research work. I would like to thank my organization for providing me datasets of real-time salesforce Applications which has been played a major role for analysis in my research work.

REFERENCES

[1] OWASP. “The Ten Most Critical Web Application Security Risks,” [Online], Available:https://www.owasp.org/index.php/Top_10-2017_Top_10 ,Accessed on: 3 August 2018.
[2] Gearset, Static code analysis for Apex, [Online], Available:https://gearset.com/assets/Static-code-analysis-for-apex.pdf, Accessed on: 5 August 2018.
[3] Checkmarx, Apex Vulnerabilities & How to Develop Securely on Force.com, [Online], Available:https://www.checkmarx.com/resources/white-papers/page/2/ , Accessed on: 5 August 2018.
[4] A. Saxena, S. Sengupta, P. Duraisamy, V. Kaulgud, and A. Chakraborty, “Detecting SOQL-injection vulnerabilities in SalesForce applications,”Proceedings of the 2013 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2013, pp. 489–493, 2013.
[5] B. Chess and G. Mcgraw, “Static analysis for security,” IEEE Security and Privacy, vol. 2, no. 6, pp. 76–79, 2004.
[6] D. Singh, V. R. Sekar, K. T. Stolee, and B. Johnson, “Evaluating how static analysis tools can reduce code review effort,” Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, vol. 2017–October, pp. 101–105, 2017.
[7] B. Maty and S. Checkmarx, “True Source Code Analysis VS Binary \ Byte Code Analysis,” vol. 1, no. 800, pp. 1–9.
[8] GrammaTech, Enhancing Code Reviews with Static Analysis [Online], Available: http://blogs.grammatech.com/enhancing-code-reviews-with-static-analysis, Accessed on: 9 September 2018.
[9] Salesforce Secure Coding Guide [Online], Available: https://developer.salesforce.com/docs/atlas.en-us.secure_coding_guide.meta/secure_coding_guide/secure_coding_guidelines.htm, Accessed on: 9 January 2019.
[10] Z. Zhioua, S. Short, and Y. Roudier, “Static code analysis for software security verification: Problems and approaches,”Proceedings - IEEE 38th Annual International Computers, Software and Applications Conference Workshops, COMPSACW 2014, pp. 102–109, 2014.
[11] T. Thomas, “Exploring the usability and effectiveness of interactive annotation and code review for the detection of security vulnerabilities,” Proceedings IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, vol. 2015–Decem, pp. 295–296, 2015.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY