iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 11 **Issue:** VI **Month of publication:** June 2023

DOI: https://doi.org/10.22214/ijraset.2023.53953

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: 7.538
Volume 11 Issue VI Jun 2023- Available at www.ijraset.com

An Efficient Approach for Mitigating Insecure Direct Object Reference (IDOR) Bug Bounty Method

Shivam Singh[1], Monika Dandotiya[2]
[1, 2]*Department of FCE, Poornima University, Jaipur, Rajasthan,*

Abstract: ***This real-time paper explores Insecure Direct Object Reference (IDOR) vulnerabilities in the context of bug bounty programs. It provides an overview of IDOR attacks and their impact on web application security. The paper discusses common IDOR vulnerabilities, including how they can be discovered and exploited by attackers. It also highlights the importance of implementing proper access controls and other security measures to prevent IDOR attacks. The paper concludes with a discussion of best practices for identifying and mitigating IDOR vulnerabilities in bug bounty programs, including the use of manual testing.***

Keywords: IDOR, Vulnerability, Sensitive data, authorized, unauthorized

I. INTRODUCTION

Insecure Direct Object Reference (IDOR) Vulnerabilities are among the Top 10 Most Important Vulnerabilities according to Open Web Application Security Project (OWASP) IDOR Vulnerabilities a common web application Security Flow is a type of access control vulnerability. IDOR Vulnerability with direct reference to the database objects, Web applications unintentionally expose sensitive internal objects such as files, databases, and user information leaks. vulnerability is a significant risk to web applications that provide users with direct access to objects through user-supplied input. This flaw allows attackers to bypass authorization and access system resources, such as database records or files, and manipulate data. By altering the value of a parameter used to point to an object, attackers can directly access resources without proper validation or authorization checks. This may include accessing files or database entries belonging to other users within the system[1]. The vulnerability arises because the application uses user-supplied input to retrieve objects without sufficient validation or authorization checks.

Insecure Direct object vulnerability occurs when the following Four conditions meet:-

1) The application exposes a direct link to an internal resource.
2) The user has the ability to alter a URL or form parameter in order to modify a direct reference[2].
3) The application allows access to the internal object without verifying if the user is authorized.
4) The attacker can change the HTTP request method and bypasses the authorization.

Fig.1: Insecure Direct Object Reference[3]

A. Control

Access control, also known as authorization, involves placing limitations on who or what can access requested resources or carry out actions. In web application settings, access control relies on authentication and session management[4]. Access control policies are established using methods such as authentication and authorization, which enable organizations to authenticate the users' identities and ensure that they are granted access at an appropriate level considering aspects such as their device, whereabouts, position, and other conditions.

1) Authentication is the process of verifying a user's identity and confirming that they are indeed the person they claim to be.
2) Session management involves determining which subsequent HTTP requests are originating from the same user.
3) Access control is responsible for deciding whether the user has permission to perform the action they are attempting to execute[5].

The various types of access controls can be classified into the following categories:

a) Vertical access controls - Vertical access controls refer to methods of limiting access to important functionality that is not accessible to other types of users.
b) Horizontal access controls - refer to mechanisms that limit access to resources only to users who have been granted explicit permission to access those resources.
c) Context-dependent access controls- Context-dependent access controls are methods of restricting access to resources and functionality that take into account the current state of the application or the user's actions within it[5].

Access control is a security measure that aims to prevent unauthorized users from accessing confidential information such as customer data and intellectual property. It also helps to reduce the risk of data exfiltration by employees and mitigates web-based attack threats. Rather than manually managing permissions, many security-aware organizations utilize identity and access management solutions to enforce access control policies.

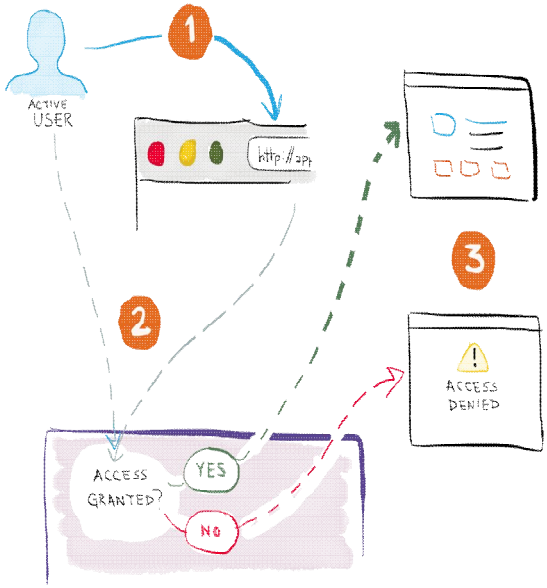


Fig. 2: Access control [6]

B. How an idor Vulnerability Works

IDOR (Insecure Direct Object Reference) is a vulnerability that arises when an application uses an identifier such as an ID, UID, PID, and Name, etc. to access resources such as files, records or URLs, without proper authorization checks. An attacker can exploit this vulnerability by manipulating the identifier to access unauthorized resources or sensitive information [7].

For Example, consider a web application that displays the details of a user's orders. The application uses an ID to retrieve the order information from the database and display it to the user. The ID is passed as a parameter in the URL.

Suppose the application does not check if the user has permission to access the order before displaying it. In that case, an attacker can modify the parameter in the URL to access the order of other users.

For instance, if the attacker changes the ID to 54321, they can view the order details of another user without proper authorization.

Like this - https://example.com/orders?id=54321

This way, the attacker can access sensitive information, such as the user's name, address, phone no, payment details, etc. Moreover, they can also modify or delete the orders of other users, leading to data breaches and financial losses

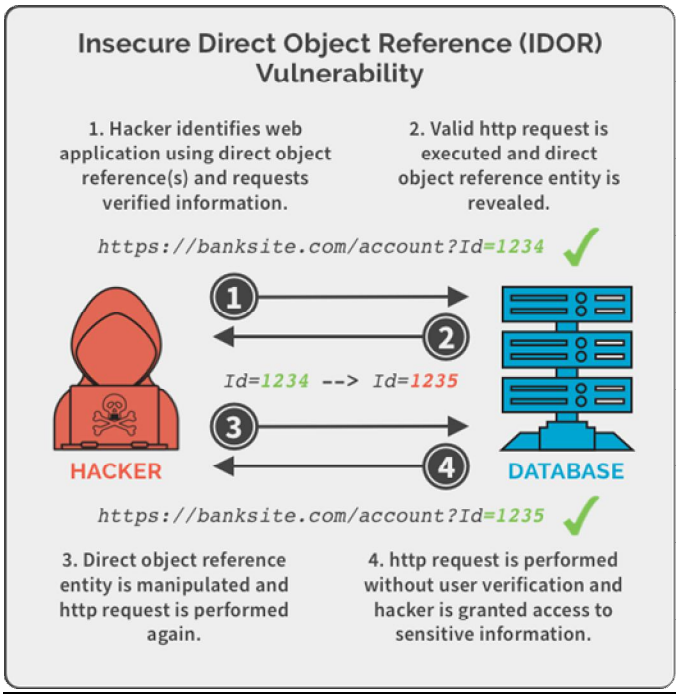


Fig.3: IDOR Vulnerabilities[8]

C. Types of Idor Vulnerability

There are four types of IDOR attacks that an attacker can use to exploit this vulnerability. They are

1) Integer-based IDOR attacks: This type of attack involves manipulating the ID parameter by incrementing, decrementing, or changing the value to another integer. For example, consider an application that displays the details of a user's order based on the order ID. An attacker can use an integer-based IDOR by changing the ID parameter to view the orders of other users[9].

2) Sequential IDOR attacks: This type of attack involves guessing the value of the ID parameter by iterating through a range of values. For example, an application uses a sequential ID for users or orders. an attacker can use Sequential IDOR attacks to access other users' data by guessing the IDs value.

3) Unverified object reference attacks: This type of attack occurs when an application does not properly verify that the user has access to the object referenced by the ID parameter. For example, consider an application that uses the ID parameter to access the file on the server. An attacker can use Unverified object reference attacks to access other files on the server by modifying the ID parameter[10].

4) Semi-verified object reference attacks: This type of attack occurs when an application verifies the user's authorization level and does not verify the object referenced by the ID parameter. For example, consider an application that uses the ID parameter to access a resource such as an order. An attacker can use Semi-verified object reference attacks to access other orders by modifying the ID parameter.

II. LITERATURE REVIEW

P. A. E. Pratama and A. M. Rhusuli [2022] examine the weaknesses and vulnerabilities of the web application by penetration testing using a method in the form of Insecure Direct Object References (IDOR), with a case study using one URL contained in the application. The test results obtained are the tested URLs that then show vulnerabilities to Insecure Direct Object References (IDOR)[11].

John Jackson [2022] describes some of the reasons a company may decide to ignore the idea of a bug bounty program: increased threat actor activity, security researchers scamming, applications being a small consideration, enormous budgetary requirements, and other security tooling as a priority. An enterprise may be fearful that establishing a bug bounty program will cause an increase in malicious threat actors attempting to hack into or successfully exploit applications[12].

J. Zhou et al.[2021] study bounties in open source projects on GitHub to better understand how bounties can be leveraged to evolve such projects in terms of addressing issue reports. We investigated 5,445 bounties for GitHub projects.

These bounties were proposed through the Bountysource platform with a total bounty value of $406,425. We find that in general, the timing of proposing bounties is the most important factor that is associated with the likelihood of an issue being addressed. More specifically, issue reports are more likely to be addressed if they are for projects in which bounties are used more frequently and if they are proposed earlier[13].

E. Shaji and N. Subramanian [2021] inspect six non-intrusive detection methodologies which are useful in the identification of web application vulnerabilities without hampering the organization's service model. The dataset that was used for real-world analysis consisted of government agencies and reputable organizations. They have assessed these vulnerabilities on the criterion of the statistical probability of finding these vulnerabilities on the dataset[14].

B. Heubl [2020] Proposed a HIGH chance that many of today's cybercriminals were introduced to hacking through modifying computer games. It's also likely that many ethical hackers, like bug bounty hunters, came into the craft the same way[15].

S. Mumtaz et al.[2020] propose a word embedding for the cyber security vulnerability domain. They train our embedding model on multiple, rich, and heterogeneous security vulnerability information sources publicly available on the web. The benefits of such specialized word embedding are demonstrated through a qualitative comparison of word similarity and the exemplary task of matching security professionals to vulnerability discovery tasks posted to bug bounty programs[16].

III. RESEARCH METHODOLOGY

IDOR (Insecure Direct Object Reference) is a security vulnerability that occurs when an application allows a user to manipulate a reference to an internal object such as a file, record, or database entry without proper authorization checks.This can result in an attacker being able to access, modify, or delete sensitive data they should not have access. IDOR can have adverse effects on the confidentiality, integrity, and availability of your organization's critical data.

Here are a few examples of how IDOR can impact data:

Access to sensitive data: An attacker can use IDOR to bypass access controls and gain unauthorized access to sensitive data. For example, if a website allows users to access their own account details by passing an account ID parameter in the URL, an attacker can change the parameter to access other users' accounts.

Modification of data: IDOR can also be used to modify data. For instance, if a web application allows users to update their profile information by submitting a form with a user ID parameter, an attacker can change the ID to update someone else's profile.

Deletion of data: IDOR can also allow an attacker to delete data they should not have access to. For example, if a website allows users to delete their own posts by passing a post ID parameter, an attacker can change the parameter to delete someone else's post.

Manipulation of critical data: In some cases, IDOR can be used to manipulate critical data, such as financial records or medical records. For example, if a banking application allows users to transfer money by passing an account ID parameter, an attacker can change the parameter to transfer money to their own account.

A. Proposed Methodology and Security Header

Here's a proposed methodology to identify and mitigate IDOR Vulnerability:

1) Identify potential IDOR attack surface: Look for areas where sensitive data is exposed or user input is accepted, such as APIs, search functions, and user profiles.

2) Fuzzing: use automated tools like ffuf, Burpsuit, dirsearch, and OWASp zap to send various inputs to the target application and observe its responses.

3) Manual Testing: conduct a manual test to identify hidden resources, unauthorized access, and manipulation opportunities.

4) Test For authorization flaws: check if the application implements proper authorization checks and if access control mechanisms can be bypassed.

5) Verify data integrity: Make sure that sensitive data is not leaked or manipulated through the application.

-Steps Involved in the Execution of the IDOR Attack:

Step 1-Capture the Request: The attacker begins by selecting a target website for the IDOR attack, and then adds it to the scope. Next, they use a spider tab to crawl the website and identify all the URLs that contain specific parameters.

Step 2-Request to filter the parameters: After the first step, we will filter our captured request with the parameter filters. An attacker will only choose that parameter or injection points where they can execute the attacks.

Step 3-Send request to Repeater: In case the attacker discovers any injection points that can be exploited for IDOR, they will then send the request to the repeater.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VI Jun 2023- Available at www.ijraset.com

Step 4-Captureing of Parameters: Once The person carrying out the attack has pinpointed the injection point that is susceptible, they intend to try and execute an IDOR attack. using social engineering techniques or following the pattern indicated by the injection point. For instance, the attacker might change the PID value from 123 to 321, thus gaining access to another user's account information.

B. HTTP Security Headers

Security Headers are commands that web applications use to establish protective measures within web browsers. These headers are utilized by both the client and server to exchange information as part of the HTTP protocol. During communication with the server, the behavior of the web page is determined by the directives in these headers. By adhering to these instructions, web browsers can avoid exposing clients to potential security risks such as clickjacking or cross-site scripting. Furthermore, headers can be utilized to configure the browser to only permit secure TLS communication with legitimate certificates, or even to require the use of a specific server certificate. Due to the increasing number of data breaches and website hacks resulting from misconfigurations or lack of protection, Security Headers can be employed to safeguard websites against common attacks like XSS, code injection, clickjacking, and others 1-HTTP Strict Transport Security (HSTS)- The HTTP Strict Transport Security (HSTS) security header is intended to protect websites from man-in-the-middle and cookie hijacking attacks. Let's imagine, for example, that you have a website called www.example.com and that you have gotten an SSL certificate to convert it from HTTP to HTTPS. However, some users might still attempt to use HTTP to access your website. You could reroute your HTTP users to HTTPS to solve this problem. But by using the website's non-encrypted version at first, man-in-the-middle assaults may be able to compromise it. This problem is addressed by the HTTP Strict Transport Security Header, which instructs the web browser to only access websites over HTTPS and converts all HTTP requests into HTTPS queries automatically. By doing this, you may protect your website from man-in-the-middle attacks and increase security.

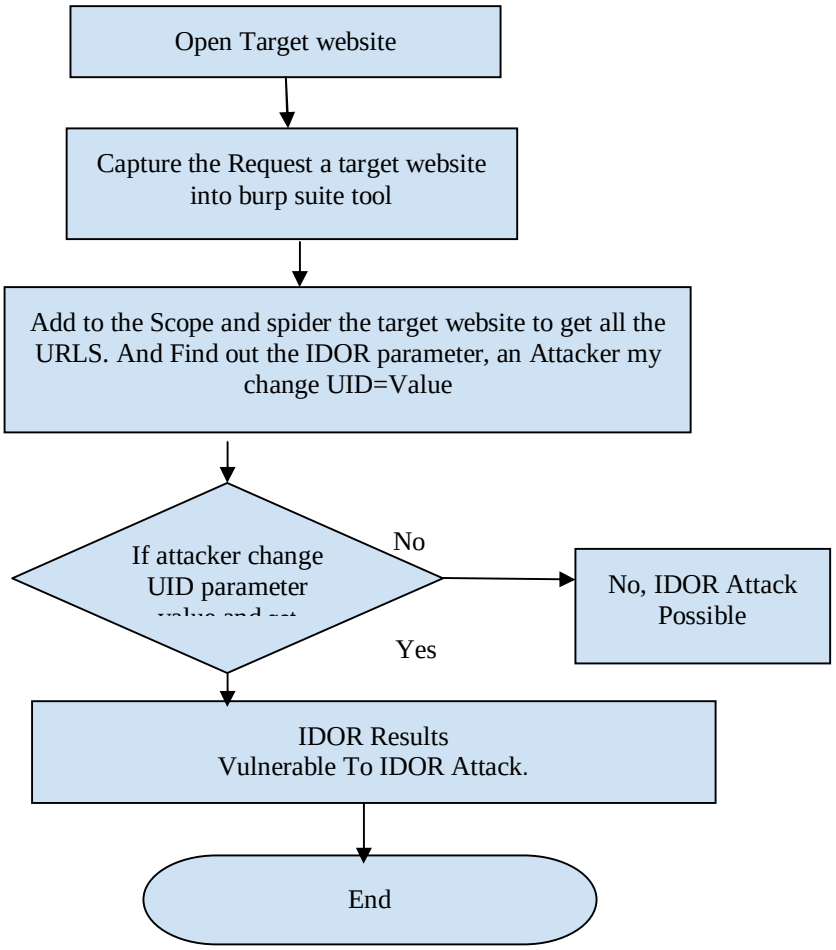


Fig.4: Flowchart of Proposed Methodology

Here is an example of an HSTS header:

Strict-Transport-Security:max-age=31536000; includeSubDomains

1) Cross-Site-Scripting Protection (X-Xss)- Websites are protected from script injection attacks by the X-XSS Protection header. The X-XSS-Protection header can stop browsers from putting malicious javascript code into an HTTP request to get sensitive data like session cookies. This header aids in the detection and prevention of cross-site scripting (XSS) assaults, which are often utilized and have the potential to seriously undermine the security of websites.

   Syntax:

   X-XSS-Protection: 0

   X-XSS-Protection: 1

   X-XSS-Protection: 1; mode=block

   X-XSS-Protection: 1; report=

2) Website Iframe Protection- The X-Frame-Options HTTP response header is used to determine whether a browser can display a page in a <frame>, <iframe>, or <object>. Including this header helps protect a website against clickjacking attacks by preventing its content from being embedded into other sites. This can be accomplished by specifying whether or not the site's content can be displayed in a frame or iframe, which can prevent attackers from tricking users into performing actions they did not intend to perform.

   Syntax:

   X-Frame-Options: DENY

   X-Frame-Options: SAMEORIGIN

3) Content Security Policy- The Content-Security-Policy Header is a security measure used to specify what content can be loaded on a web page. It follows a whitelisting approach, which allows the website owner to define which images, scripts, CSS, applets, and other content is allowed to be loaded by the browser. By doing so, it can effectively prevent the exploitation of cross-site scripting (XSS), ClickJacking, and HTML injection attacks, among others.

   Syntax:

   Content-Security-Policy:

4) Access-Control-Allow-Origin- The Access Control Allow Origin header is part of the CORS (Cross-Origin-Resources-Sharing) mechanism. It specifies whether the response to a request can be shared with the code that made the request from a particular origin. For example, if website A requests a resource from website B, website B must indicate in its If the Access-Control-Allow-Origin header doesn't authorize website A to access a certain resource, the request will be blocked according to the same-origin policy (SOP).

   Syntax:

   Access-Control-Allow-Origin: *

   Access-Control-Allow-Origin:

   Access-Control-Allow-Origin: null

   Mitigation of IDOR Vulnerability:

Here are some recommended best practices to prevent Insecure Direct Object Reference (IDOR) vulnerabilities:

Implement Access Controls: Use role-based access controls (RBAC) or attribute-based access controls (ABAC) to restrict access to sensitive resources. Implement strong authentication and authorization mechanisms to ensure that only authorized users can access the resources they are authorized to access.

Use Unique Identifiers: Use unique identifiers, such as UUIDs or hashes, to reference resources instead of using sequential or predictable values. This makes it harder for an attacker to guess or manipulate the identifier.

Validate User Input: Always validate user input, including parameters in URLs, to ensure that they are within expected ranges and formats. Sanitize input to remove any potentially malicious code that could be used to exploit vulnerabilities.

Implement Parameterized Queries: Use parameterized queries in your database interactions to prevent SQL injection attacks. Parameterized queries ensure that user input is treated as data rather than executable code. Conduct Thorough Testing: Conduct regular security testing and penetration testing to identify potential vulnerabilities, including IDOR vulnerabilities. Use both manual and automated testing tools to identify potential vulnerabilities.

Have a Strong Vulnerability Management Process: Have a strong vulnerability management process in place, including a clear vulnerability disclosure and patching process. Respond quickly to identified vulnerabilities and deploy patches as soon as possible.
By following these best practices, you can reduce the likelihood of IDOR vulnerabilities in your applications and systems.

IV. EXPERIMENTAL ANALYSIS IDOR

IDOR vulnerability typically involves identifying a web application that is vulnerable to IDOR attacks and testing the application to confirm the existence of the vulnerability.
The typical process for testing generally consists of the following steps:

1) Identify the target application and the functionality that uses a direct object reference.
2) Use the application's functionality to access the direct object reference (such as an account number or user ID).
3) Use a tool or the application's own functionality to modify the direct object reference to access another user's data.
4) Observe the application's response to the modified direct object reference.
5) If the application provides access to the other user's data, analyze the data to determine its sensitivity and usefulness for the attack.
6) Use the obtained data to execute the intended attack (such as stealing funds or accessing sensitive information).
7) Cover tracks by deleting any traces of the attack or altering logs to hide the attack.
8) Repeat steps 2-7 for other direct object references until the attacker achieves their goals or until no more vulnerable direct object references are found.
9) Report the vulnerability to the application owner with the steps taken to reproduce it.

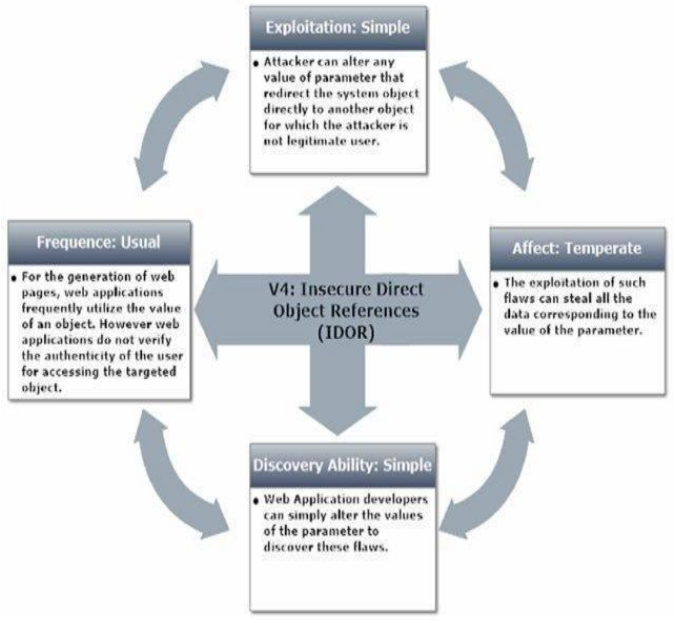


Fig. 5: STEPS INVOLVE

An example of IDOR vulnerability would be a banking application that allows users to view their account details by entering their account number. If the application uses the account number as the only input parameter to retrieve the account details, an attacker could potentially modify the input parameter to view the account details of other users without proper authorization checks.
To demonstrate this vulnerability, you could perform the following steps:

a) Access the banking application and log in with a valid user account.
b) Use the "Inspect element" feature of the browser to modify the account number parameter in the URL of the account details page.
c) Change the account number to a different value to see if you can access the details of another user's account.
d) If successful, try modifying the account details of the other user, such as changing the account balance or transaction history.
e) Report the vulnerability to the application owner with the steps taken to reproduce it.

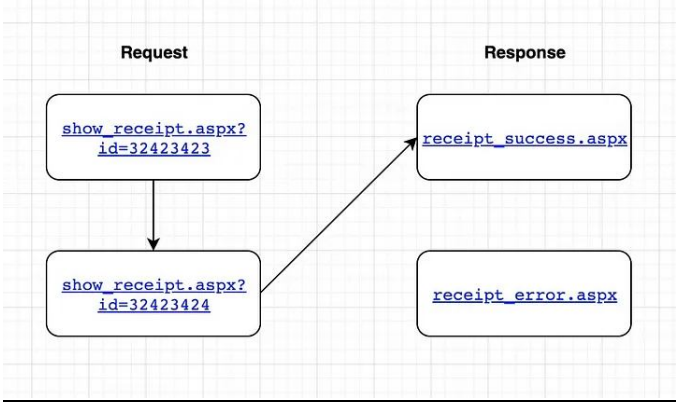


Fig 6: IDOR Vulnerability Exploit

During the testing process, it is important to ensure that the tests are conducted in a controlled and ethical manner, so as not to cause any harm to the application or its users. It is also important to obtain permission from the application owner before conducting any tests.

The experimental analysis of IDOR Vulnerability is an ongoing process, as new vulnerabilities may be discovered as the application evolves and new features are added. It is important for organizations to regularly conduct vulnerability assessments and penetration testing to identify and remediate any security weaknesses in their applications.

A. Experimental IDOR Vulnerability Particle Part-1

In this, particle I'll demonstrate a Real-World IDOR Vulnerability Find a bug bounty Hunter, Cyber Security Expert, and beginner. This Particle helps the real-time method to identify a Vulnerability in Google sub-domains.

NOTE: already Explaining the Steps Involved in the Execution of the IDOR Attack in the 2.2 Section. So, Direct Jump to the particle.

- The IDOR Vulnerability in Real-time.

Summary: The issue here is that the vulnerable web application doesn't check if the endpoint URL parameter provided the value.

Vulnerability Name: IDOR

Step To Reproduce This Vulnerability:

1) Open a target website -> trendsfromhome.withyou.com
2) Log in to your account at the trendsfromhome.withyou.com
3) Go to the profile

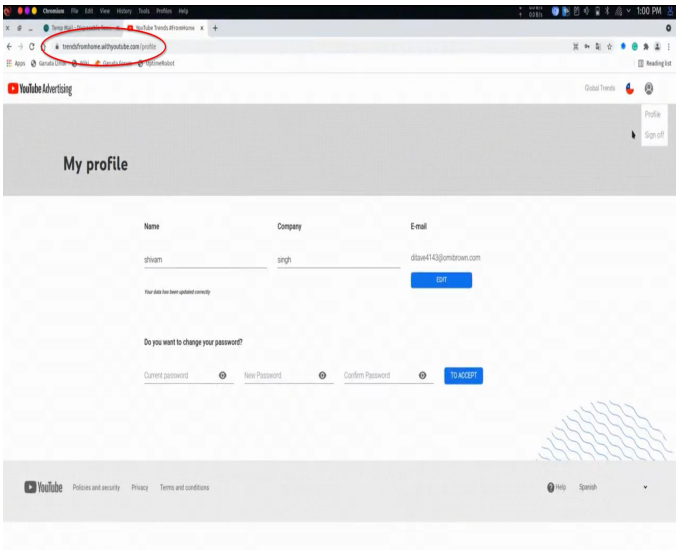


Fig 7: In the Images, show go to Profile

Change the Account Name and click the “Edit” Button.

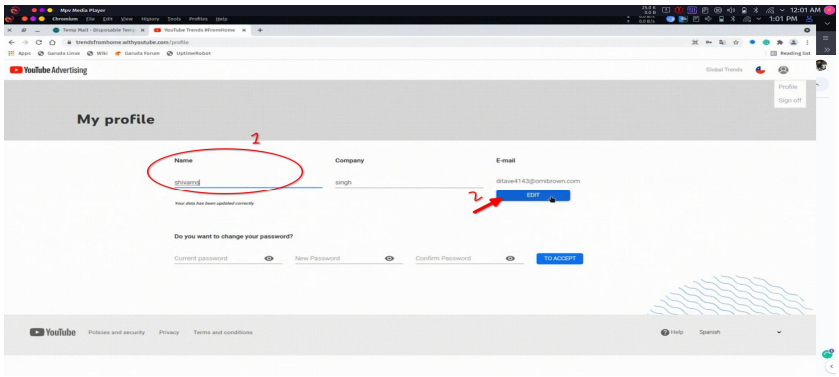


Fig 8: Name Change and intercept Request

Intercept The Request into Burpsuit and send the request to the repeater tab

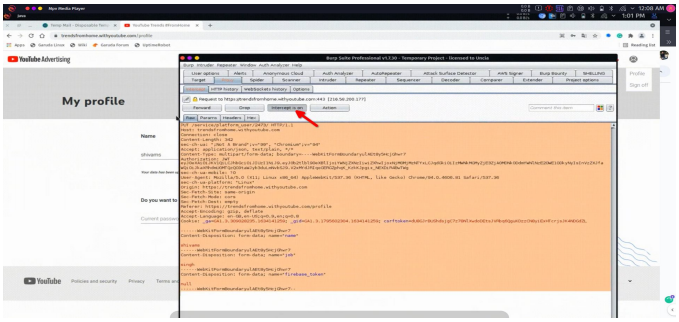


Fig 9: Request Intercept in burp-suite.

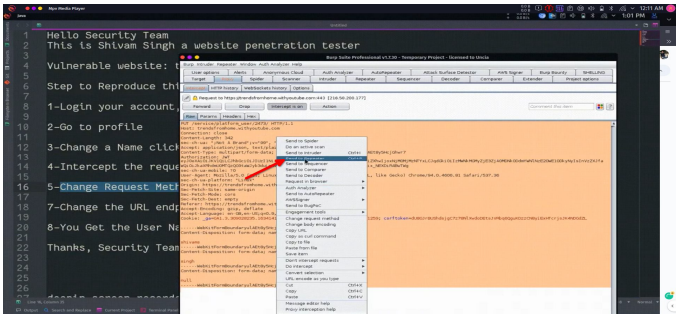


Fig 10: Send the request to the repeater tab

Right-click and Change Request Method “PUT” To “GET”

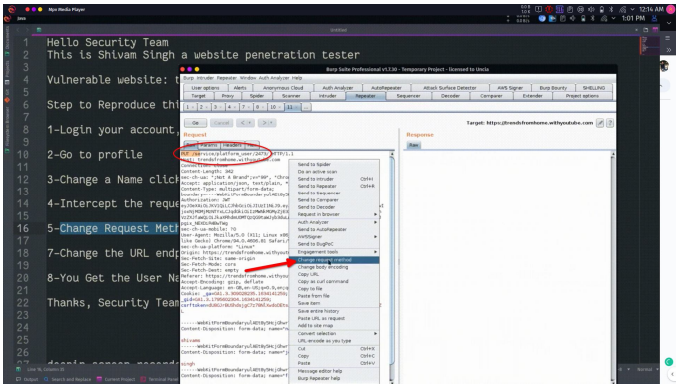


Fig 11: Right-click and Change Request Method

The attacker changes the URL Endpoint number value 2473 to a different number 241

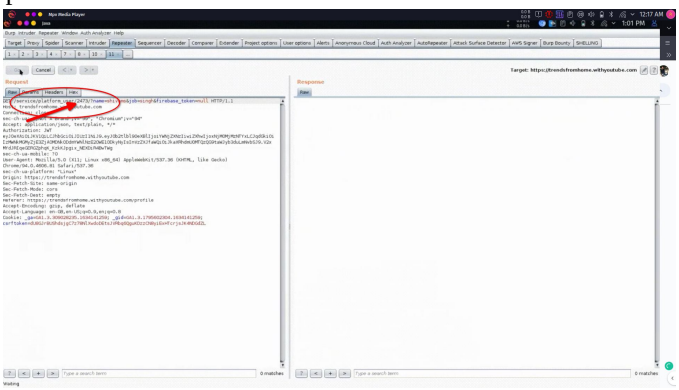


Fig 12: Change the URL Endpoint number value

The attacker Exposed the Victim's account information.

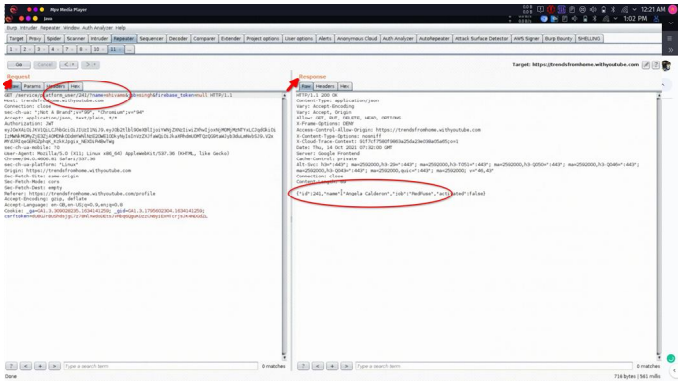


Fig 13: Exposed the Victim account information

Report the vulnerability to the Google Security Team and The VRP panel has decided to issue a reward of $500.00 for your report.

V. CONCLUSION

The conclusion regarding IDOR vulnerability is that it is a severe threat to the security of an application or system. It can lead to unauthorized access to sensitive information or resources, resulting in data breaches, financial loss, reputational damage, and legal liabilities. This paper describes IDOR vulnerability, Access control, IMPACT, and Mitigation. We have demonstrated a real-time IDOR vulnerability Exploit in Google Subdomain How Hackers access Users' sensitive information

VI. ACKNOWLEDGMENT

The heading of the Acknowledgment section and the References section must not be numbered. Causal Productions wishes to acknowledge Michael Shell and other contributors for developing and maintaining the IEEE LaTeX style files which have been used in the preparation of this template. To see the list of contributors, please refer to the top of file IEEETran.cls in the IEEE LaTeX distribution.

REFERENCES

[1] V. Atlidakis, P. Godefroid and M. Polishchuk, "RESTler: Stateful REST API Fuzzing," 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 748-758, doi: 10.1109/ICSE.2019.00083.

[2] V. Atlidakis, P. Godefroid and M. Polishchuk, "Checking Security Properties of Cloud Service REST APIs," 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020, pp. 387-397, doi: 10.1109/ICST46399.2020.00046.

[3] E. Viglianisi, M. Dallago and M. Ceccato, "RESTTESTGEN: Automated Black-Box Testing of RESTful APIs," 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020, pp. 142-152, doi: 10.1109/ICST46399.2020.00024.

[4] S. Karlsson, A. Čaušević and D. Sundmark, "QuickREST: Property-based Test Generation of OpenAPIDescribed RESTful APIs," 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020, pp. 131-141, doi: 10.1109/ICST46399.2020.00023.

[5] S. Karlsson, A. Čaušević and D. Sundmark, "Automatic Property-based Testing of GraphQL APIs," 2021 IEEE/ACM International Conference on Automation of Software Test (AST), 2021, pp. 1-10, doi: 10.1109/AST52587.2021.00009.

[6] N. Laranjeiro, J. Agnelo and J. Bernardino, "A Black Box Tool for Robustness Testing of REST Services," in IEEE Access, vol. 9, pp. 24738-24754, 2021, doi: 10.1109/ACCESS.2021.3056505.

[7] A. Arcuri, "Automated Black- and White-Box Testing of RESTful APIs With EvoMaster," in IEEE Software, vol. 38, no. 3, pp. 72-78, May-June 2021, doi: 10.1109/MS.2020.3013820.

[8] Viriya, Anthony, and Yohan Muliono. "Peeking and Testing Broken Object Level Authorization Vulnerability onto E-Commerce and E-Banking Mobile Applications." Procedia Computer Science 179 (2021): 962-965.

[9] Laurens Sion, Koen Yskout, Dimitri Van Landuyt, and Wouter Joosen. 2018. Solution-aware data flow diagrams for security threat modeling. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC '18). Association for Computing Machinery, New York, NY, USA, 1425–1432. DOI:https://doi.org/10.1145/3167132.3167285.

[10] Laurens Sion, Koen Yskout, Dimitri Van Landuyt, Alexander van den Berghe, and Wouter Joosen. 2020. Security Threat Modeling: Are Data Flow Diagrams Enough? In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20). Association for Computing Machinery, New York, NY, USA, 254–257. DOI:https://doi.org/10.1145/3387940.3392221.

[11] P. A. E. Pratama and A. M. Rhusuli, "Penetration Testing on Web Application Using Insecure Direct Object References (IDOR) Method," 2022 International Conference on ICT for Smart Society (ICISS), Bandung, Indonesia, 2022, pp. 01-07, doi: 10.1109/ICISS55894.2022.9915074.

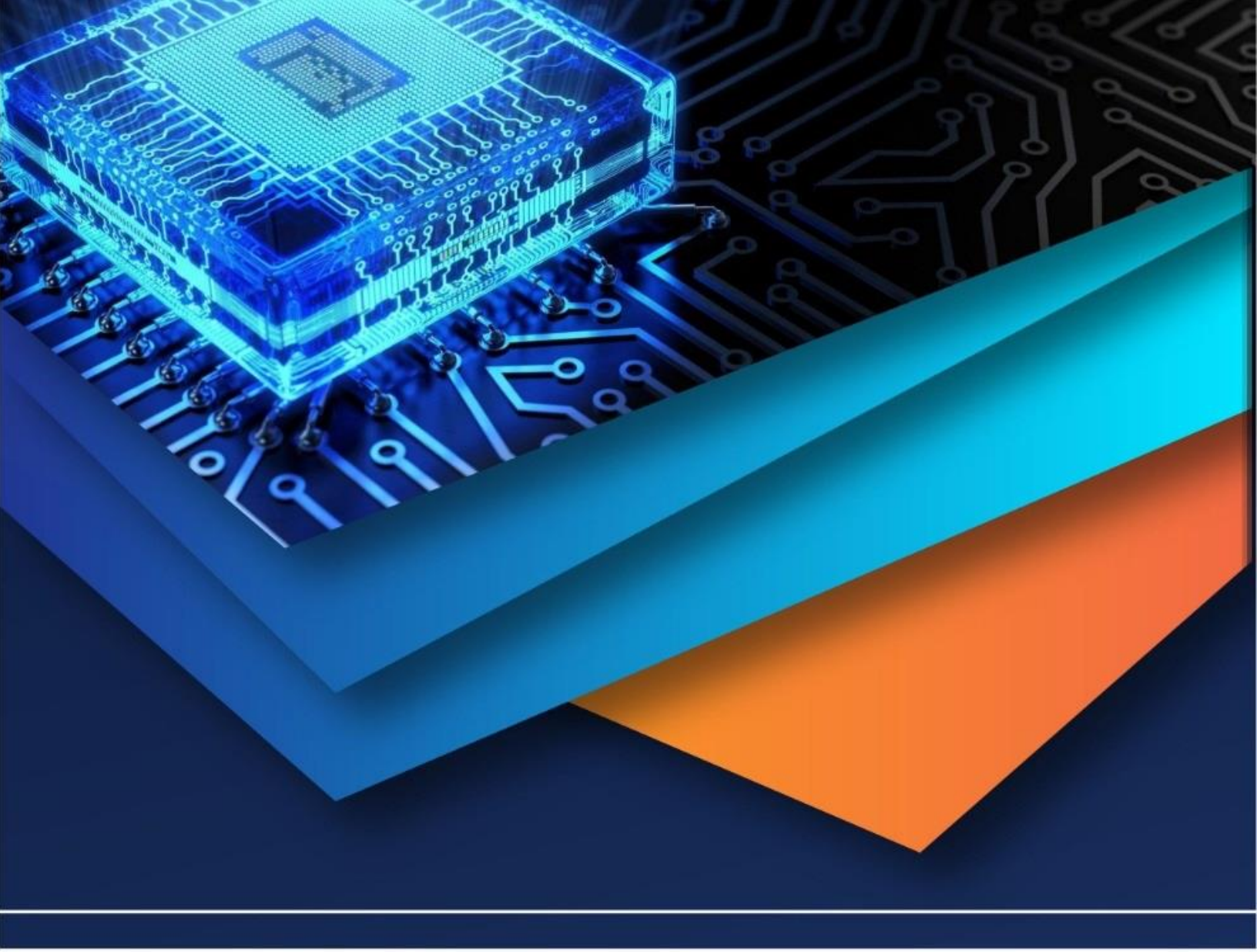
[12] John Jackson, "The Evolution of Bug Bounty Programs," in Corporate Cybersecurity: Identifying Risks and the Bug Bounty Program , IEEE, 2022, pp.1-9, doi: 10.1002/9781119782568.ch1.

[13] J. Zhou, S. Wang, C. -P. Bezemer, Y. Zou and A. E. Hassan, "Studying the Association Between Bountysource Bounties and the Issue-Addressing Likelihood of GitHub Issue Reports," in IEEE Transactions on Software Engineering, vol. 47, no. 12, pp. 2919-2933, 1 Dec. 2021, doi: 10.1109/TSE.2020.2974469.

[14] E. Shaji and N. Subramanian, "Assessing Non-Intrusive Vulnerability Scanning Methodologies for Detecting Web Application Vulnerabilities on Large Scale," 2021 International Conference on System, Computation, Automation and Networking (ICSCAN), Puducherry, India, 2021, pp. 1-5, doi: 10.1109/ICSCAN53069.2021.9526423.

[15] B. Heubl, "When cheating leads to crime: Many hackers are gamers and some gamers are hackers. But is it ethical, or even legal? As it turns out, the answer is far from straightforward," in Engineering & Technology, vol. 15, no. 2, pp. 28-31, March 2020, doi: 10.1049/et.2020.0212.

[16] S. Mumtaz, C. Rodriguez, B. Benatallah, M. Al-Banna and S. Zamanirad, "Learning Word Representation for the Cyber Security Vulnerability Domain," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 2020, pp. 1-8, doi: 10.1109/IJCNN48605.2020.9207140



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY