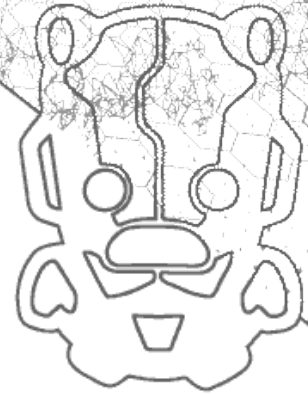


Slightly SOSL'ed

Locating and Testing SOSL Injection

Nick Dunn



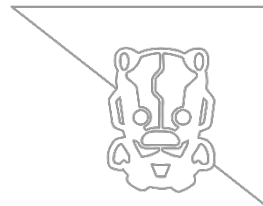
Standard Disclaimer



- ▶ As usual, these techniques can be used for good or evil.
- ▶ The techniques are intended for testing software, not breaking it.
- ▶ If you discover major issues in commercial software or libraries, report them/fix them, play nicely, and don't go down the path of world domination (or minor theft).



Agenda



- ▶ Intro & whoami
- ▶ Salesforce overview
- ▶ SOSL overview
- ▶ SOSL injection intro
- ▶ Finding SOSL injection vulnerabilities
 - ▶ Code review
 - ▶ Application testing
- ▶ Exploiting SOSL injection vulnerabilities
 - ▶ Code review
 - ▶ Application testing
- ▶ Mitigations and defences
- ▶ Conclusions



whoami

@N1ckDunn

@nickdunn@infosec.exchange



- ▶ Coming from software development and architecture
 - ▶ 6 years as software developer, architect, team lead, working in secure software for the financial sector
 - ▶ Worked as an in-house penetration tester and code reviewer in online gambling
 - ▶ Security consultancy
- ▶ Moved into security consultancy and worked on:
 - ▶ Code review
 - ▶ Penetration testing
 - ▶ Threat modelling, architecture review
 - ▶ Automating security testing with new tools, scripts, etc.
 - ▶ Security research



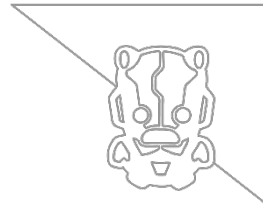
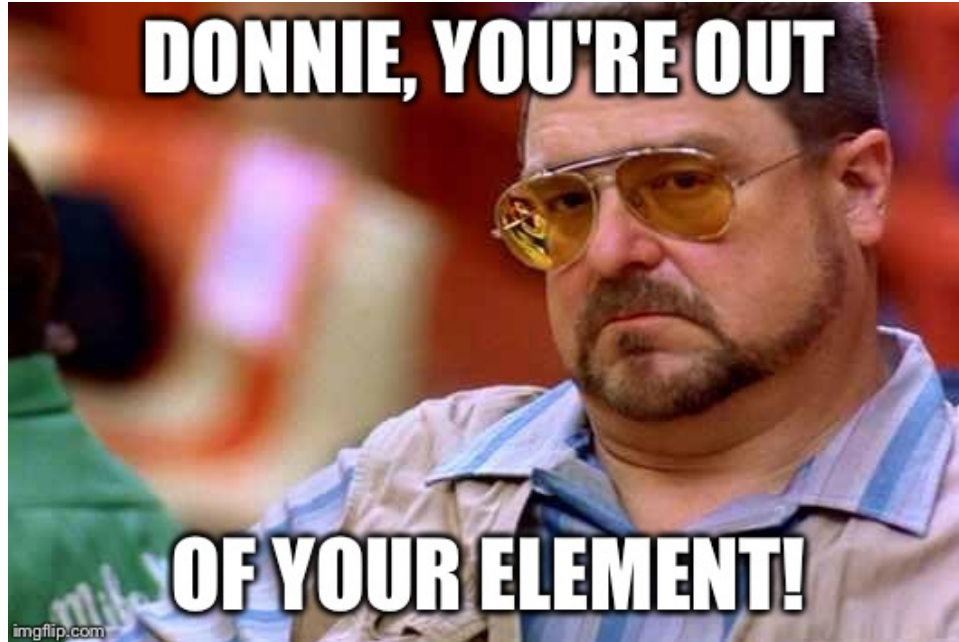
What This Talk Is (and Isn't) About



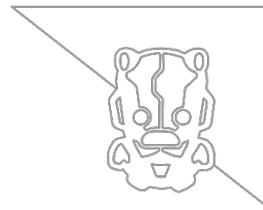
- ▶ This is a talk about a vulnerability that seems to be lurking in various APIs and websites built on top of Salesforce.
- ▶ The talk will show how to test for the issue and determine its risk level for the situation.
- ▶ This talk is happening because when I looked, I couldn't find example payloads anywhere. // *shocked_face*
- ▶ Examples will be provided to counter the lack of payloads/descriptions online.
- ▶ It is *not* a talk about Salesforce.
- ▶ The vulnerability is a coding issue, *not* a Salesforce issue.

An Important Detail

- ▶ I am *not* a Salesforce expert.



What is Salesforce?



- ▶ Salesforce is a CRM system.
- ▶ Salesforce Sites is a product that facilitates website creation, giving a tailored customer web front-end for their data, stored using Salesforce CRUD permissions, applied to Salesforce Objects.
- ▶ “CRUD (Create Read Update Delete) : Object-level security within the salesforce environment is referred as CRUD. It can be used to restrict the actions that users can take on each type of standard and custom object.”

What is Salesforce?

► It looks like this (but usually more polished):



The screenshot displays the Salesforce SOSLDemo user interface. At the top, there is a search bar containing the query "Q. ('a' OR 'b')". Below the search bar, the interface shows search results for "Accounts" and "Cases".

Accounts Section:

5+ Results • Sorted by Relevance

Account Name	Account Site	Phone	Account Owner Alias
United Oil & Gas, Singapore		(650) 450-8810	NDunn
United Oil & Gas, UK		+44 191 4956203	NDunn
University of Arizona		(520) 773-9050	NDunn
Express Logistics and Transport		(503) 421-7800	NDunn
United Oil & Gas Corp.		(212) 842-5500	NDunn

Cases Section:

5 Results • Sorted by Relevance

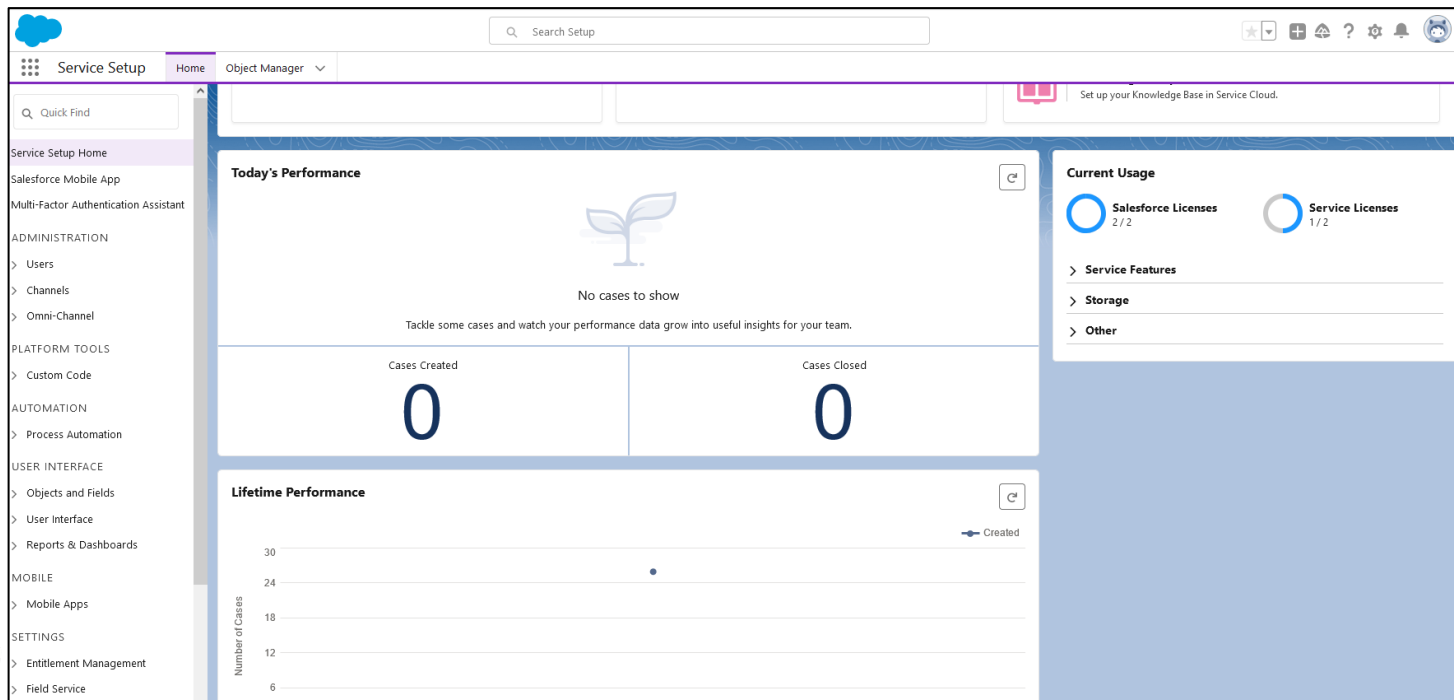
Case Number	Subject	Status	Date/Time Opened	Case Owner Alias
00001023	Electric surge damaging adjacent equipment	Closed	11/09/2023, 15:47	NDunn
00001018	Cannot start generator after electrical failure	Closed	11/09/2023, 15:47	NDunn
00001007	Structural breakdown of rotor assembly	Closed	11/09/2023, 15:47	NDunn
00001006	Generator assembly instructions unclear	Closed	11/09/2023, 15:47	NDunn
00001000	Starting generator after electrical failure	Closed	11/09/2023, 15:47	NDunn

At the bottom of the interface, there is a message: "Don't see your result?"

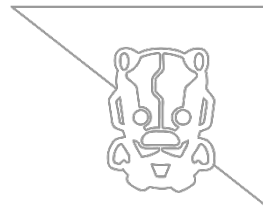
What is Salesforce?



- ▶ You can use a free developer version to learn more:

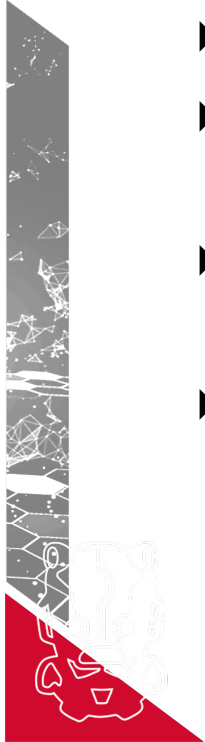


Salesforce Data Handling

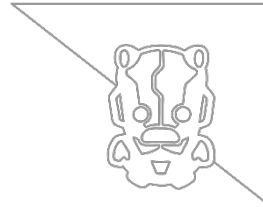


► Salesforce Objects

- A table holding data, conceptually similar to a spreadsheet.
- 'Objects' represent Salesforce database tables, and within these, 'Fields' refers to columns, and 'Records' represents the rows.
- In the context of CRUD, individual users may have read, write, or other permissions applied to Objects, Fields, or Records.
- For example, a user may have read permission only for certain Fields of a particular Object.



Salesforce Data Handling



▶ Apex

- ▶ Apex enables developers to access the Salesforce platform back-end data storage and client-server interfaces to create additional functionality and to write custom APIs.

▶ SOQL

- ▶ A query language for Salesforce data, similar syntax to SQL.

▶ DML

- ▶ An update language for Salesforce data, similar syntax to SQL

▶ SOSL

- ▶ A query language for Salesforce data, different syntax and behaviours to SQL.

What is SOSL?



- ▶ SOSL stands for Salesforce Object Search Language.
- ▶ It can be used within Salesforce to search all Objects.
- ▶ It is limited to queries – unlike SQL it cannot modify data.
- ▶ Salesforce gives a number of reasons or situations to help determine whether to use SOQL or SOSL.



SOQL vs SOSL



SOQL	SOSL
You know in which object holds the data.	You are not sure which object holds the data.
The data is to be retrieved from a single object or related objects.	You want to retrieve multiple objects and/or fields, that may not be related.
Can be used in Classes and Triggers.	Statements supported in Apex code.
Can pass query result into DML operations.	Cannot directly perform DML operations without additional coding.
Returns records.	Returns fields.
Can count retrieved records.	Cannot count retrieved records.

What does SOSL look like?



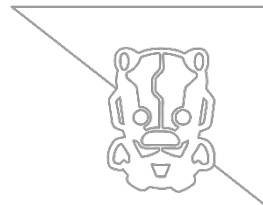
- ▶ A bit like a C++ programmer seeing Java or C# for the first time, SOSL has enough grammatical similarities to SQL to look comfortably familiar, with enough behavioural differences to cause you some confusion.
- ▶ **FIND {Search Text} [IN SearchGroup/SearchField] [RETURNING sObject(Fields to return)];**
- ▶ Examples:

```
FIND {jones} IN ALL FIELDS RETURNING account(username, phone);  
FIND {jones};
```
- ▶ This can be used in Apex code, like this:

```
List<List<SObject>> searchList = [FIND 'jones' IN ALL FIELDS RETURNING  
account(username, phone)];
```



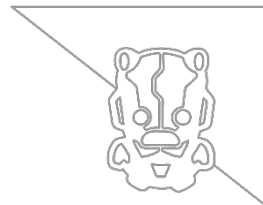
What is SOSL Injection?



- ▶ It is an accepted vulnerability type for unsafely written Apex code.
- ▶ Like SQL injection, it's the outcome of an attacker injecting script into dynamically built SOSL statements that rely on inadequately validated user input.
- ▶ It can be found in Salesforce, primarily in Apex code written by developers customizing Salesforce.



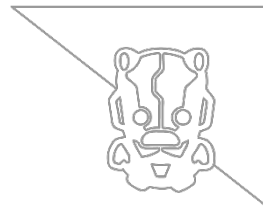
What is SOSL Injection?



- ▶ It has information disclosure issues, but no information integrity issues in normal use.
- ▶ In some (less common) circumstances it can theoretically result in data modification if the outcome of an injectable statement is used to pass conditions or data into a DML statement.



What is SOSL Injection?



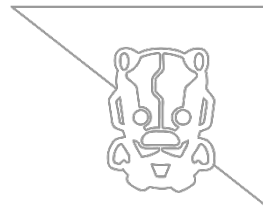
- ▶ The Salesforce SOSL page has this information on SOSL injection:

SOSL injection is a technique by which a user causes your application to execute database methods you did not intend by passing SOSL statements into your code. A SOSL injection can occur in Apex code whenever your application relies on end-user input to construct a dynamic SOSL statement and you do not handle the input properly.

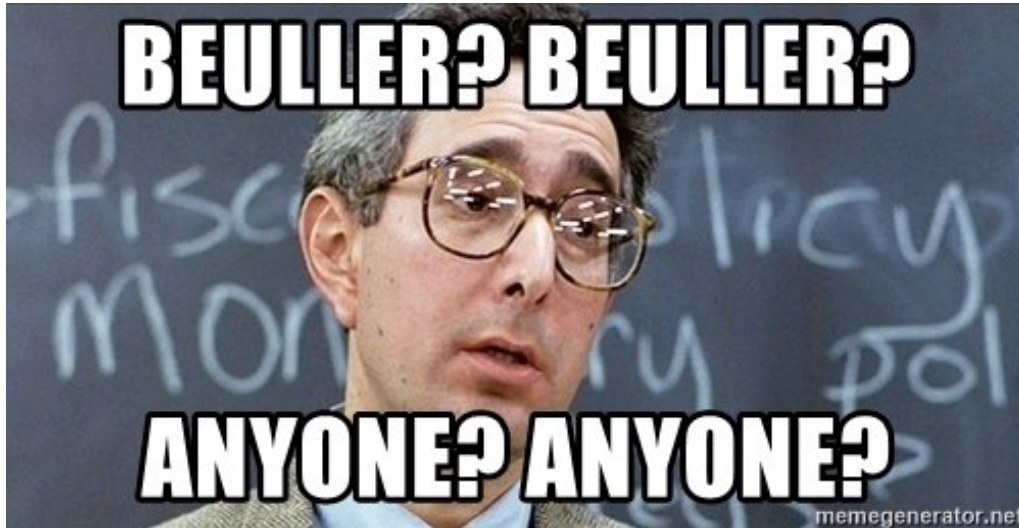
To prevent SOSL injection, use the `escapeSingleQuotes` method. This method adds the escape character (`\`) to all single quotation marks in a string that is passed in from a user. The method ensures that all single quotation marks are treated as enclosing strings, instead of database commands.



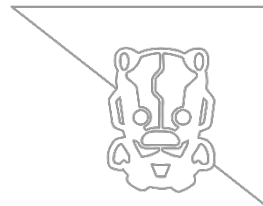
How Do I Find Out More?



- ▶ You can't!
- ▶ My search attempts brought back a couple of results stating that SOSL injection exists but no payloads/examples, or explanations.



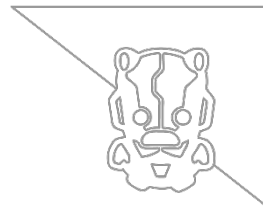
Finding Instances of SOSL Injection



- ▶ Three ways to discover instances:
 - ▶ Code review
 - ▶ Web application testing
 - ▶ API testing



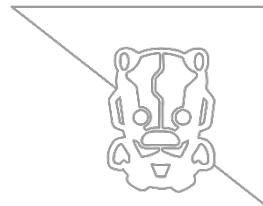
Discovery, Confirmation, and Exploitation



- ▶ Methodology:
 - ▶ Investigation & discovery
 - ▶ Confirmation
 - ▶ Test for sanitization and injection points
 - ▶ Check for exploitability and information disclosure



SOSL Code Review



- ▶ When reviewing Apex code, SOSL injection can be found in constructs similar to these:

```
public static List<List<SObject>> getAccountsAndContacts_dynamicSOSL(String searchString) {  
  
    String queryString = 'FIND \'' + searchString + '\'' IN Name Fields RETURNING Account, Contact';  
    List<List<SObject>> listToReturn = search.query(queryString);  
  
    return listToReturn;  
}
```

- ▶ Developers can use grep to check their code for issues (or some commercial tools if they have a licence).
- ▶ The primary (and frequently recommended) defence in Salesforce is the escapeSingleQuotes function.

SOSL Code Review



- ▶ When reviewing Java (or other) code, SOSL injection can be found in constructs similar to these, which perform a Salesforce API call:

```
public void searchSample() {  
    try {  
        // Perform the search using the SOSL query.  
        SearchResult sr = connection.search(  
            "FIND {4159017000} IN Phone FIELDS RETURNING "  
            + "Contact(Id, Phone, FirstName, LastName), "  
            + "Lead(Id, Phone, FirstName, LastName), "  
            + "Account(Id, Phone, Name)");  
  
        // Get the records from the search results.  
        SearchRecord[] records = sr.getSearchRecords();  
    }  
}
```



SOSL Code Review



- ▶ Note that in both of the previous code samples, the **FIND** keyword was present at the start of the statement.
- ▶ This will be universally present for any SOSL query, which requires:

```
FIND search_term [optional terms]
```

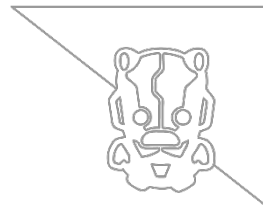
- ▶ When conducting a code review, regardless of language, SOSL queries can be found by grepping for 'FIND'.
- ▶ Note that SOSL (like SQL) is not case-sensitive.

- 

```
qry = "FIND {map*} IN ALL FIELDS RETURNING Account (Id, Name), Contact, Lead"
```

(?i) \bFIND\b\s+(\\"'|\'|\\{)
/\bFIND\b\s+(\\"'|\'|\\{)/i

What Do I Do With the Results?



- ▶ The grep/regex will bring back any SOSL instances in the codebase.
- ▶ There are a few key things to look for, to assess viability of SOSL injection:
 - ▶ Injectability – Is any user-supplied data used in dynamic statements
 - ▶ System Mode – What level of privilege does the code have
- ▶ We'll look at this in more detail when discussing exploitability.

Investigation and Confirmation (App Test)



- ▶ Identifying SOSL when testing a web application or API can be done with a few specific values for search input.
- ▶ Initial fuzzing or testing values to confirm that a SOSL search takes place should include:
 - ▶ A^*
 - ▶ "A"
 - ▶ "A*"
 - ▶ "AA" (or AA)



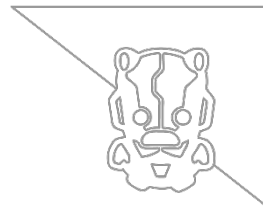
Investigation and Confirmation (App Test)



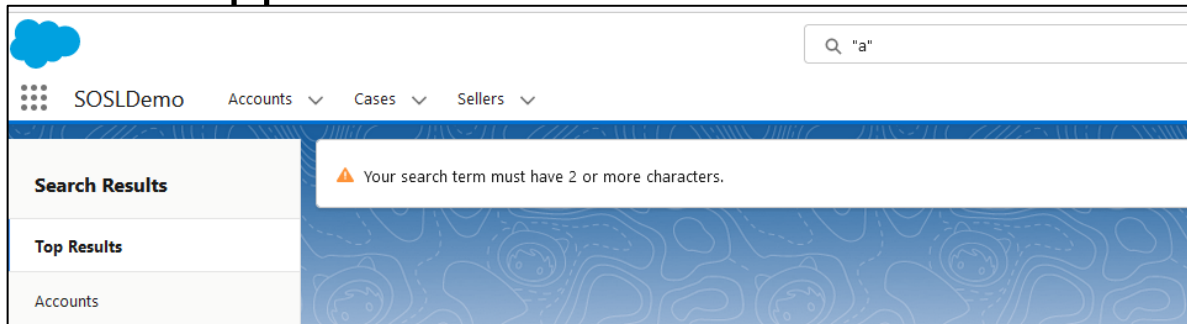
- ▶ The use of SOSL can be confirmed by these responses to A*, “A”, “A*”, and “AA”/AA.
- ▶ For the first three:
 - ▶ “Your search term must have 2 or more characters.”
- ▶ For the fourth:
 - ▶ Indication of a completed search either containing no results or results containing the characters ‘AA’.



Investigation and Confirmation (App Test)



► Web Application:



► API:

HTTP/1.1 500 Server Error

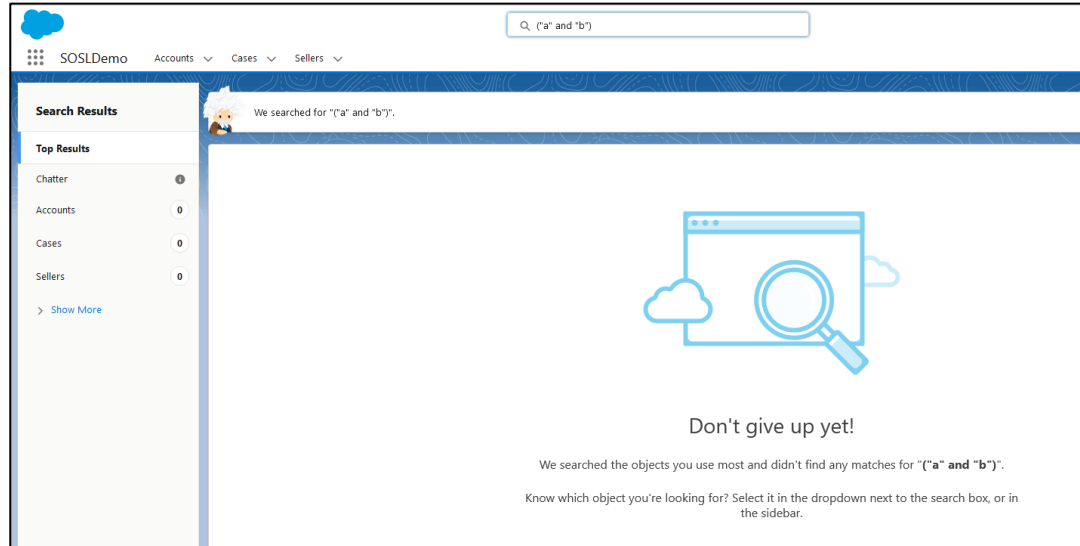
...

```
[{"errorCode": "APEX_ERROR", "message": "System.SearchException: search term must be longer than one character: *\n\nClass.AccSrch.buildSearchQuery: line 197, column 1\nClass.AccSrch.findCustomers [snip]...
```

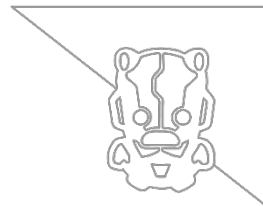
Investigation and Confirmation (App Test)



- ▶ Successful search result for values that don't exist with valid SOSL syntax:



Investigation and Confirmation (App Test)



- ▶ Other useful/alternative testing values for further confirmation (and later exploitation) can include:
 - ▶ "known_good" OR "a*"
 - ▶ "known_good" AND "a*"
 - ▶ "known_good" AND "known_bad"
- ▶ The lack of prior knowledge around what values are known good and known bad mean this type of test is manual, rather than part of the fuzzing process.
- ▶ These are likely to have further uses during the confirmation and exploitation phase, and likely to be far more useful when your test includes both web and API.

From Confirmation to Exploitation



- ▶ We now have a list of possible injection points. What next?

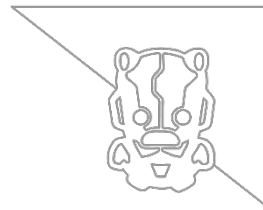


From Confirmation to Exploitation



- ▶ If all user roles should be able to see all data, then the issue is very low severity (or could be viewed as not an issue).
- ▶ You're more likely to find issues in APIs than in web applications, due to their custom/bespoke nature.
- ▶ Web applications will generally restrict searches to data that is allowed for your user.
- ▶ APIs written in Apex may be running in Salesforce's System Mode in some circumstances, User Mode in others.
- ▶ APIs running in System Mode would be able to read data that may not be visible to the user account that you have.

From Confirmation to Exploitation



- ▶ Main limitations on exploitation attempts:
 - ▶ No comment character
 - ▶ No data modification constructs



From Confirmation to Exploitation (Code Review)



- ▶ Our chief concern is exploitability, and levels of risk:

```
public static List<List<SObject>> getAccountsAndContacts_dynamicSOSL(String  
searchString) {
```

```
    String queryString = 'FIND \'' + searchString + '\'' IN Name FIELDS RETURNING  
Account, Contact';
```

```
    List<List<SObject>> listToReturn = search.query(queryString);
```

```
    return listToReturn;
```

```
}
```

- ▶ Check for the escapeSingleQuotes function, but also look at SOSL statement construction, and injection points.

Exploitability



Developers

Home

Documentation

APIs

Discover ▾

Build ▾

Connect ▾

Apex Developer Guide

Pages

v59.0

Search

CTRL J

Apex Developer Guide ▾

Release Notes

> Getting Started with Apex

▾ Writing Apex

> Data Types and Variables

> Control Flow Statements

> Classes, Objects, and Interfaces

▾ Working with Data in Apex

> Working with sObjects

> Data Manipulation Language

▾ SOQL and SOSL Queries

Working with SOQL and
SOSL Query Results

Accessing sObject Fields
Through Relationships

Note

The syntax of the `FIND` clause in Apex differs from the syntax of the `FIND` clause in SOAP API and REST API:

- In Apex, the value of the `FIND` clause is demarcated with single quotes. For example:

```
1 FIND 'map*' IN ALL FIELDS RETURNING Account (Id, Name), Contact, Opportunity, Lead
```

Note

- In the API, the value of the `FIND` clause is demarcated with braces. For example:

```
1 FIND {map*} IN ALL FIELDS RETURNING Account (Id, Name), Contact, Opportunity, Lead
```

Apex that is running in system mode ignores field-level security while scanning for a match using `IN ALL FIELDS`.

From Confirmation to Exploitation (Code Review)



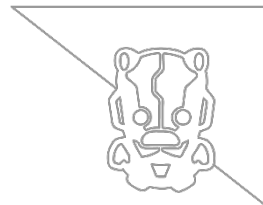
► SOSL injection with higher levels of risk:

```
public static List<List<SObject>> getAccountsAndContacts_dynamicSOSL(String
searchString, String searchField) {
    String safeSearch = String.escapeSingleQuotes(searchString);
    String safeField = String.escapeSingleQuotes(searchField);
    String queryString = 'FIND \'' + searchString + '\'' IN ' + searchField + '
FIELDS';
    List<List<SObject>> listToReturn = search.query(queryString);

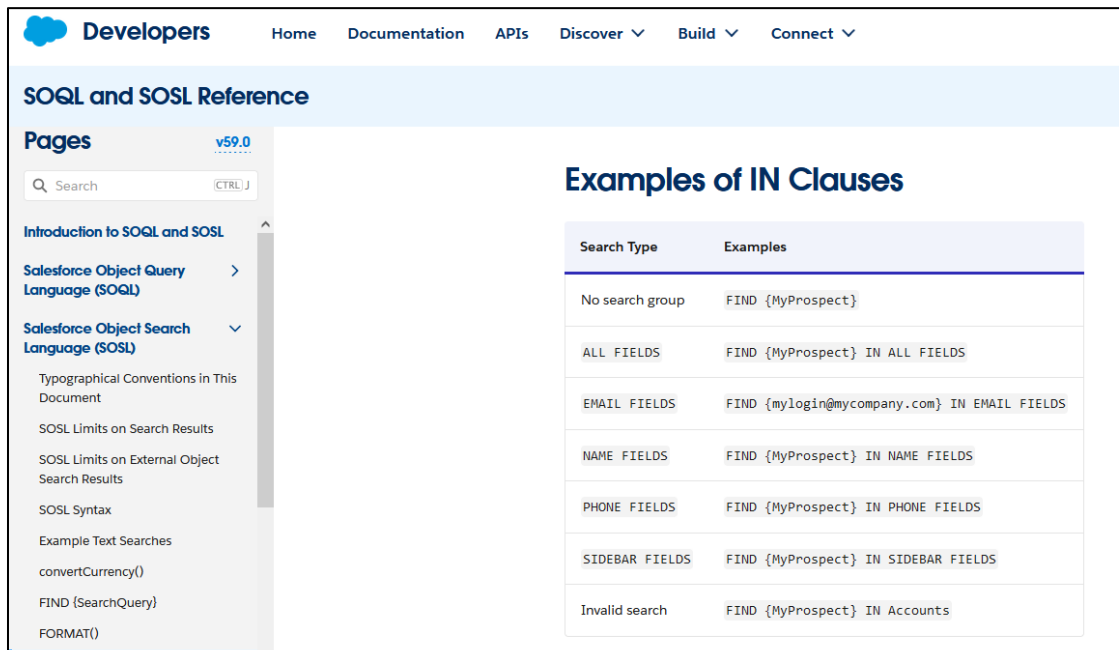
    return listToReturn;
}
```

► Despite use of escapeSingleQuotes, this is vulnerable to injection, and information disclosure.

From Confirmation to Exploitation



► The IN clause:



SOQL and SOSL Reference

Pages v59.0

Search (CTRL) J

Introduction to SOQL and SOSL

Salesforce Object Query Language (SOQL)

Salesforce Object Search Language (SOSL)

Typographical Conventions in This Document

SOSL Limits on Search Results

SOSL Limits on External Object Search Results

SOSL Syntax

Example Text Searches

convertCurrency()

FIND (SearchQuery)

FORMAT()

Examples of IN Clauses

Search Type	Examples
No search group	FIND {MyProspect}
ALL FIELDS	FIND {MyProspect} IN ALL FIELDS
EMAIL FIELDS	FIND {mylogin@mycompany.com} IN EMAIL FIELDS
NAME FIELDS	FIND {MyProspect} IN NAME FIELDS
PHONE FIELDS	FIND {MyProspect} IN PHONE FIELDS
SIDEBAR FIELDS	FIND {MyProspect} IN SIDEBAR FIELDS
Invalid search	FIND {MyProspect} IN Accounts

From Confirmation to Exploitation (Code Review)



▶ Two injection points:

```
public static List<List<SObject>> getAccountsAndContacts_dynamicSOSL(String searchString,
String searchField) {
    String safeSearch = String.escapeSingleQuotes(searchString);
    String safeField = String.escapeSingleQuotes(searchField);
    String queryString = 'FIND \'' + searchString + '\'' IN ' + searchField + ' FIELDS';
```

▶ In normal use, the function would construct harmless search:

```
FIND 'john.jones@example.com' IN email FIELDS;
```

▶ With injection of a suitable value, a broader search of data, normally inaccessible to the user:

```
FIND 'secret contract' IN ALL FIELDS;
```



From Confirmation to Exploitation (Code Review)



- ▶ Special Circumstances:
 - ▶ Apex code with further actions dependent upon a search result
- ▶ Check what is done with results by any following code.
- ▶ Determine whether injection could facilitate unexpected or unintended outcomes.



Exploitability - User Mode vs System Mode

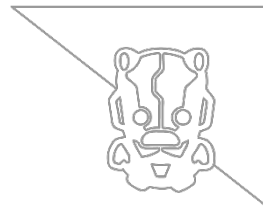


- ▶ Inside the web interface the user's searches are generally run in User Mode.
- ▶ By default Apex code runs in System Mode and may be exploitable if developer has not used appropriate settings.
- ▶ "search_term_1" OR "search_term_2" may bring back different results from injection into an Apex API call, compared to direct entry in standard components of the web interface.
- ▶ User Mode can be enforced in searches in Apex code:

```
String query = 'FIND \'Test*\'' IN ALL FIELDS RETURNING Account(Name), Contact, Lead';  
List<List<SObject>> searchResults = Search.query(query, AccessLevel.USER_MODE);
```



From Confirmation to Exploitation (App Test)



► Tools:

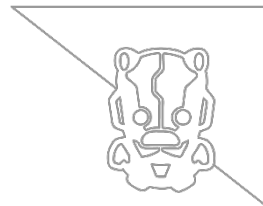
- Web Browser
- BurpSuite (other web proxies are available)
- Postman

► Injection Points and Payloads:

- Based on initial investigation/footprinting
- An ideal injection point would allow modification of which fields are to be returned or searched

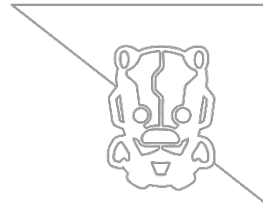


From Confirmation to Exploitation (App Test)



- ▶ **Testing Sanitization in Web/APIs:**
 - ▶ Depending on code constructs, a search can be enclosed in { } or “ ”
 - ▶ Check whether } or ‘ causes an Apex error to verify presence or absence of sanitization
- ▶ **Information Disclosure:**
 - ▶ Broader search results than intended
 - ▶ Look for any fields or data that should not be visible that have been exposed
 - ▶ Do API results differ from same search in web interface?

Exploitability – Response Comparison (1)



- ▶ Bring back data from API, compare this to data returned in web interface.
- ▶ Payload and Response:

```
POST /services/apexrest/FindFruit HTTP/1.1
```

```
...
{
  "Name": "",
  "Color": "",
  "LatinName": "",
  "Supplier": "",
  "Origin": "\"A*" OR "B*\"",
}
```

```
HTTP/1.1 200 OK
```

```
...
[{"name":"orange","color":"orange","latinname":"ipsum factum","supplier":{"name":"universal fruit","telephone":"+441234111222","email":united_fruit@example.com},"origin":"Spain"}, {"name":"lime","color":"green","latinname":"lorum ipsum factum","supplier":{"name":"universal fruit","telephone":"+441234111222","email":united_fruit@example.com},"origin":"Bulgaria"}]
```

Exploitability – Response Comparison (2)



- ▶ Does the API allow constructs that appear to be blocked or filtered in the web application?

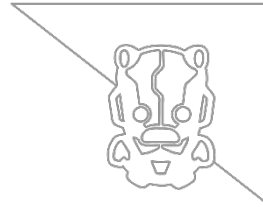
```
"Origin": "\"A*\" OR \"B*\""
```

- ▶ Does the API return data that the web application doesn't?

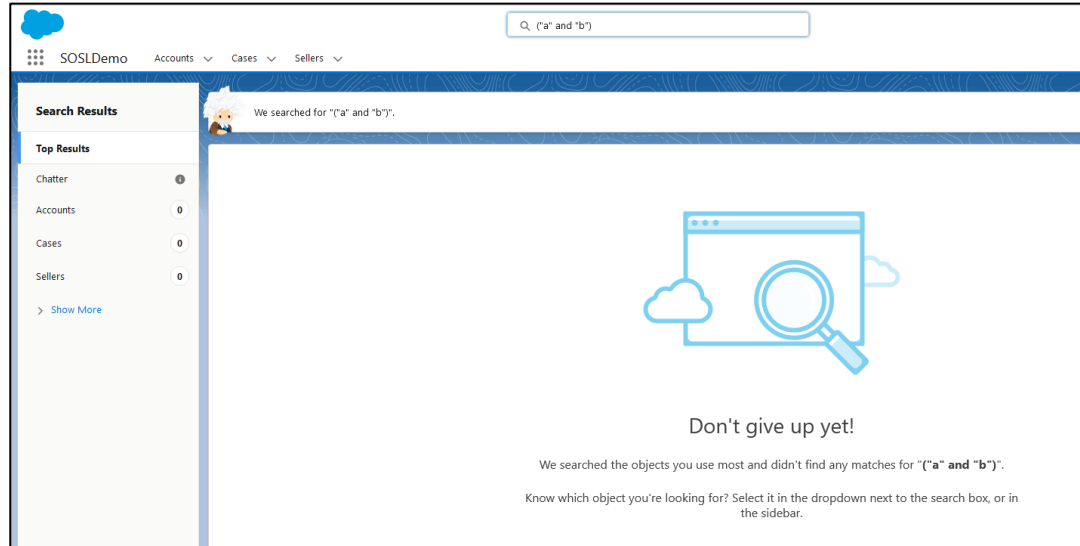
```
[{"name": "orange", "color": "orange", "latinname": "ipsum  
factum", "supplier": {"name": "universal  
fruit", "telephone": "+441234111222", "email": "united_fruit@example  
.com"}, "origin": "Spain"},  
{"name": "lime", "color": "green", "latinname": "lorum ipsum  
factum", "supplier": {"name": "universal  
fruit", "telephone": "+441234111222", "email": "united_fruit@example  
.com"}, "origin": "Bulgaria"}]
```



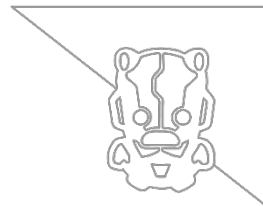
Exploitability – Response Comparison (3)



- ▶ Does the API return data that the web application doesn't?



Some Injection Points are Better than Others



- ▶ Remember the code review example from earlier:

```
public static List<List<SObject>> getAccountsAndContacts_dynamicSOSL(String searchString,
String searchField) {
    String safeSearch = String.escapeSingleQuotes(searchString);
    String safeField = String.escapeSingleQuotes(searchField);
    String queryString = 'FIND \'' + searchString + '\'' IN ' + searchField + ' FIELDS';
```

- ▶ This example was deliberately contrived but the construction might still be recognizable in an API (or web application) test:

```
{
  "Forename": "secret contract",
  "Surname": "",
  "Telephone": "",
  "Email": "",
},
{
  "Field": "ALL",
}
```

Exploitability (continued)



- ▶ Look for any possibility of injectable clauses.
- ▶ As already seen, the FIELDS clause can be a target.
- ▶ The RETURNING clause is another possibility, particularly if the WHERE clause has been included.

Developers Home Documentation APIs Discover Build Connect

SOQL and SOSL Reference

Pages v59.0

Search CTRL J

- SOQL LIMITS MY CURRENT PAGE
- Search Results
- SOSL Syntax
- Example Text Searches
- convertCurrency()
- FIND (SearchQuery)
- FORMAT()
- IN SearchGroup
- LIMIT n
- OFFSET n
- ORDER BY Clause
- RETURNING FieldSpec**
- toLabel(fields)

WHERE

Optional description of how search results for the given object are filtered, based on individual field values. If unspecified, the search retrieves all the rows in the object that are visible to the user.

If you want to specify a **WHERE** clause, you must include a *FieldList* with at least one specified field. The following example isn't proper syntax:

```
RETURNING Account (WHERE name like 'test')
```

But this syntax is:

```
RETURNING Account (Name, Industry WHERE Name like 'test')
```

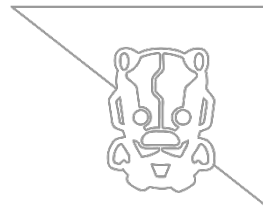
See [conditionExpression](#) for more information.

ORDER BY Clause

Optional description of how to order the returned result, including ascending and descending order, and how nulls are ordered. You can supply more than one **ORDER BY** clause.

If you want to specify an **ORDER BY** clause, you must include a *FieldList* with at least one specified field.

RETURNING and WHERE

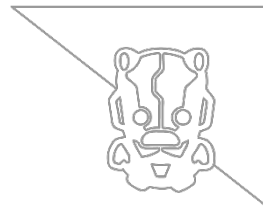


- ▶ The Salesforce website gives this illustration of RETURNING clause, with WHERE subclause:

```
FIND {test} IN ALL FIELDS
    RETURNING Contact(Salutation, FirstName, LastName,
        AccountId WHERE Name = 'test'),
        User(FirstName, LastName),
        Account(id WHERE BillingState IN ('California', 'New
York'))
```



RETURNING and WHERE (continued)



- ▶ Use of an injectable WHERE clause could potentially offer a workable attack vector:

```
Query = 'FIND \'' + param1 + '\'' IN ALL FIELDS
```

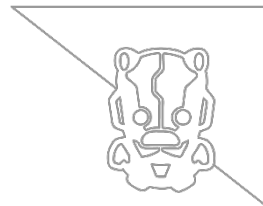
```
RETURNING Contact(Salutation, FirstName, LastName, AccountId WHERE  
Name = \'' + param2 + '\');
```

- ▶ A lack of sanitization would allow information disclosure using a payload such as:

```
test'), PrivateObject(id, text WHERE Description = 'Secret
```

- ▶ This standard SQL injection tactic allows closing of quotations and braces to avoid difficulties caused by lack of comment character.

RETURNING and WHERE (continued)



- ▶ The intended use produces a query in its correct form:

```
FIND 'foo' IN ALL FIELDS
```

```
    RETURNING Contact(Salutation, FirstName, LastName,  
    AccountId WHERE Name = 'test')
```

- ▶ The unintended use, with the attack payload produces a query that searches other potentially private objects:

```
FIND 'foo' IN ALL FIELDS
```

```
    RETURNING Contact(Salutation, FirstName, LastName,  
    AccountId WHERE Name = 'test'), PrivateObject(id, text WHERE  
    Description = 'Secret')
```

RETURNING and WHERE (continued)



- ▶ The use of a number of standard Fields in the RETURNING clause, eliminates a lot of guesswork.

Object Reference for the Salesforce Platform

Pages v59.0

Search CTRL J

Overview of Salesforce Objects and Fields

Reference

- > Associated Objects (Feed, History, OwnerSharingRule, Share, and...)
- > Custom Objects
- > Object Interfaces
- ▼ **Standard Objects**
 - AcceptedEventRelation
 - Account
 - AccountBrand
 - AccountContactRelation
 - AccountCleanInfo
 - AccountContactRole
 - AccountInsight
 - AccountOwnerSharingRule

Reference / Standard Objects

Standard Objects

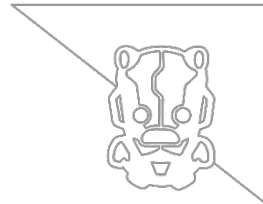
This section provides a list of standard objects and their standard fields.

Some fields may not be listed for some objects. To see the system fields for each object, see [System Fields](#).

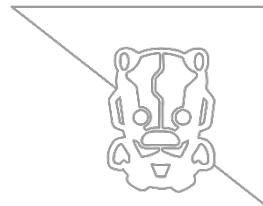
To verify the complete list of fields for an object, use a describe call from the API, or inspect with an appropriate tool. For example, inspecting the WSDL or using a schema viewer.

- **AcceptedEventRelation**
Represents event participants (invitees or attendees) with the status `Accepted` for a given event.
- **Account**
Represents an individual account, which is an organization or person involved with your business (such as customers, competitors, and partners).
- **AccountBrand**
Represents the brand details of a Partner Account. This object is available in API version 43.0 and later.
- **AccountContactRelation**
Represents a relationship between a contact and one or more accounts.
- **AccountCleanInfo**
Stores the metadata Data.com Clean uses to determine an account record's clean status. AccountCleanInfo helps you automate the cleaning or related processing of account records.

Mitigation



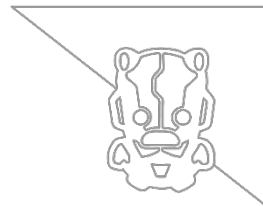
Mitigation



- ▶ User Mode for queries (where feasible).
- ▶ Sanitization:
 - ▶ The `:` operator
 - ▶ The `escapeSingleQuotes` function
 - ▶ Avoid direct use of user-controlled string literals, where possible
- ▶ Input validation:
 - ▶ If building SOSL queries in code, check for valid values.
 - ▶ Block invalid values, particularly the presence of SOSL keywords, where user input might specify search fields, or return values.



Safer Statement Construction



► Unsafe:

```
public static List<List<SObject>> getAccountsAndContacts_dynamicSOSL(String
searchString, String searchField) {
    String queryString = 'FIND \'' + searchString + '\\' IN ' + searchField + '
FIELDS';
```

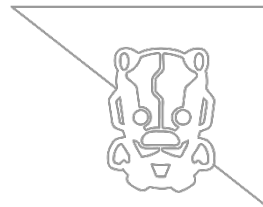
► Safer:

```
public static List<List<SObject>> getAccountsAndContacts_dynamicSOSL(String
searchString, String searchField) {
    String safeSearch = String.escapeSingleQuotes(searchString);
    String safeField = String.escapeSingleQuotes(searchField);
    String queryString = 'FIND \'' + searchString + '\\' IN ' + searchField + '
FIELDS';
```

► Even safer:

```
public static List<List<SObject>> getAccountsAndContacts_dynamicSOSL(String
searchString, String searchField) {
    String queryString = 'FIND \':searchString\' IN :searchField FIELDS';
```

Safer Statement Usage



► Unsafe:

```
public static List<List<SObject>> getAccountsAndContacts_dynamicSOSL(String
searchString, String searchField) {
    // Some code for input validation here
    String queryString = 'FIND \'' + searchString + '\'' IN ' + searchField + '
FIELDS';
    List<List<SObject>> listToReturn = search.query(queryString);
```

► Safe:

```
public static List<List<SObject>> getAccountsAndContacts_dynamicSOSL(String
searchString, String searchField) {
    // Some code for input validation here
    String queryString = 'FIND \':searchString\' IN :searchField FIELDS';
    List<List<SObject>> result = search.query(queryString, AccessLevel.USER_MODE);
```

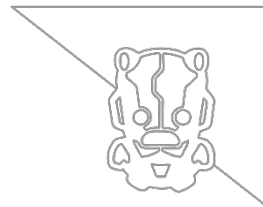
Safer Statement Usage



- ▶ Never allow direct user input to be used for field or object names!



Safer Statement Usage

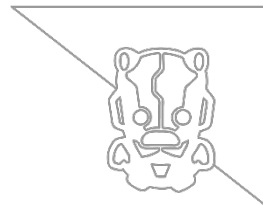


- ▶ In place of direct user input, map the input onto allowed fieldnames internally:

```
switch on input_val {  
  when 1 {  
    field_name = 'email'  
  }  
  when 2 {  
    field_name = 'telephone'  
  }  
  when 3 {  
    ...  
  }
```



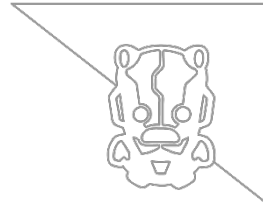
A Few Final Thoughts...



- ▶ Salesforce uses Apex, SOSL, SOQL, and DML.
- ▶ We've discussed SOSL injection and the specific payloads required.
- ▶ SQL injection payloads should identify SOQL injection (in most cases) – note the lack of comment character.
- ▶ DML injection should also be possible in a similar manner to SQL injection, with the usual warnings to beware of accidental mass deletions.

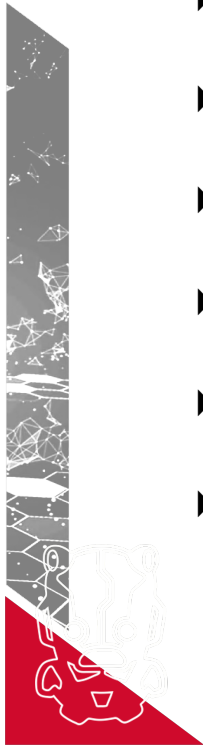


Conclusion



► We've seen:

- SOSL injection can have consequences in terms of information disclosure but is much less serious in terms of information integrity.
- Injection points may be more exploitable towards the end of a SOSL statement and can be counterintuitive compared to SQL injection.
- It is not a well-known issue, and you may find instances missed by other testers.
- It is mainly a concern if users should be limited to viewing certain subsets of Salesforce Objects.
- On a system where all users should be able to see all data, it is generally less of a concern.
- The information I've given is the result of my own reading and experimentation. I would love to hear other people's results and encourage further innovation and research.



Any Questions?



Thank You!

