Enforce Security With the stripInaccessible Method

In this post we will talk about the new way to enforce the security in apex with stripInaccessible() method. From Winter 20, stripInaccessible() security feature for field-level data protection is available for beta in production. In winter 20 Salesforce extended the feature and added enum value UPSERTABLE to System.AccessType.

stripInaccessible() is useful to strip the field that current user don't have access from query and sub-query. We can use it to remove inaccessible field from sObjects before DML operation to avoid exceptions. This method also provides the option an option to enforce the **Object level access** check.

Syntax :-

```
public static System.SObjectAccessDecision stripInaccessible(System.AccessType accessCheckType,
                                                             List<SObject> sourceRecords,
                                                             Boolean enforceRootObjectCRUD )
```

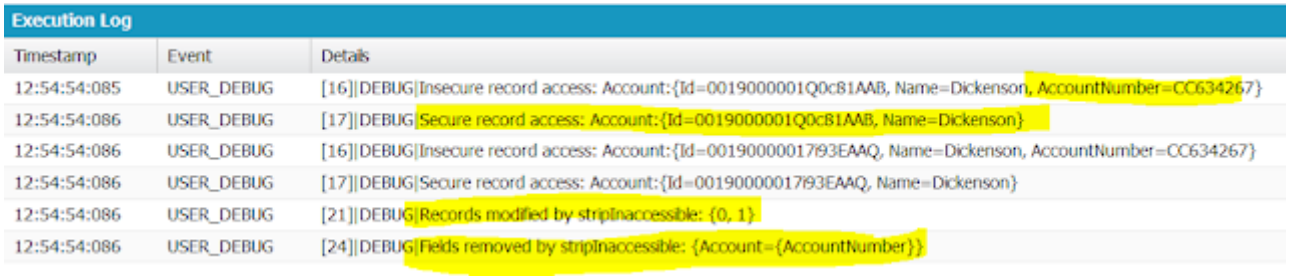
- ***accessCheckType*** : This parameter determines the type of field-level access check to be performed
- ***sourceRecords*** : A list of sObjects to be checked for fields that aren't accessible in the context of the current user's operation
- ***enforceRootObjectCRUD*** : Indicates whether an object-level access check is performed

USE CASE 1: NO ACCESS ON FIELD

In this example, User don't have access on field called accountNumber on Account Object.

```
List<Account> accounts =[SELECT Id, Name,AccountNumber
                         FROM Account limit 2];

// Strip fields that are not readable
SObjectAccessDecision decision = Security.stripInaccessible(
                                 AccessType.READABLE,
                                 accounts);

// Print stripped records
for (Integer i = 0; i < accounts.size(); i++) {
    System.debug('Insecure record access: '+accounts[i]);
    System.debug('Secure record access: '+decision.getRecords()[i]);
}

// Print modified indexes
System.debug('Records modified by stripInaccessible: '+decision.getModifiedIndexes());

// Print removed fields
System.debug('Fields removed by stripInaccessible: '+decision.getRemovedFields());
```

In above example user dont have access on accountNumber field. After using the Security.scripInaccessible method we can simply strip out the same field. Which all field are removed we can check with "**getRemovedFields**" method. Here is output of above code.



Execution Log

Timestamp	Event	Details
12:54:54:085	USER_DEBUG	[16]|DEBUG|Insecure record access: Account:{Id=0019000001Q0c81AAB, Name=Dickenson, AccountNumber=CC634267}
12:54:54:086	USER_DEBUG	[17]|DEBUG|Secure record access: Account:{Id=0019000001Q0c81AAB, Name=Dickenson}
12:54:54:086	USER_DEBUG	[16]|DEBUG|Insecure record access: Account:{Id=0019000001f93EAAQ, Name=Dickenson, AccountNumber=CC634267}
12:54:54:086	USER_DEBUG	[17]|DEBUG|Secure record access: Account:{Id=0019000001f93EAAQ, Name=Dickenson}
12:54:54:086	USER_DEBUG	[21]|DEBUG|Records modified by stripInaccessible: {0, 1}
12:54:54:086	USER_DEBUG	[24]|DEBUG|Fields removed by stripInaccessible: {Account={AccountNumber}}

USE CASE 2: NO ACCESS ON SOBJECT

Let see another example when user don't have access on object itself. For demo I simply removed the access from Account object. Then we got the below exception.

System.NoAccessException: No access to entity: Account

we have "***enforceRootObjectCRUD***" optional parameter which is true by default we can set that as false to get null value.

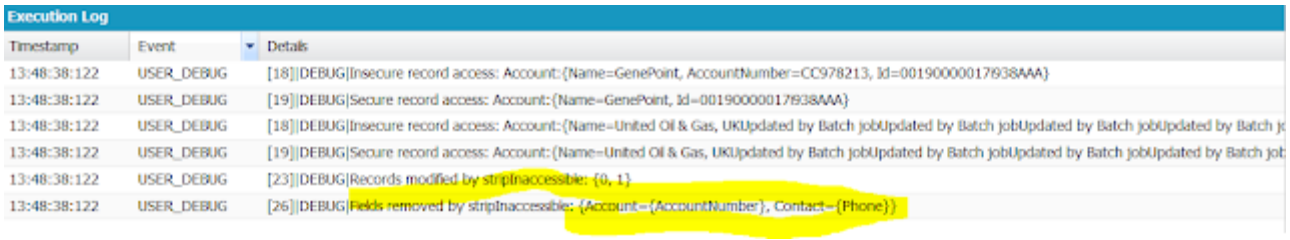
USE CASE 3: SUBQUERY.

What about if you are try to access field from subquery ?

```
List<Account> accountsWithContacts =
   [SELECT Name, AccountNumber,
      (SELECT LastName, Phone FROM Account.Contacts)
   FROM Account  limit 2];

// Strip fields that are not readable
   SObjectAccessDecision decision = Security.stripInaccessible(
                                AccessType.READABLE,
                                accountsWithContacts);

// Print stripped records
   for (Integer i = 0; i < accountsWithContacts.size(); i++)
```

```
 {
    System.debug('Insecure record access: '+accountsWithContacts[i]);
    System.debug('Secure record access: '+decision.getRecords()[i]);
  }

// Print modified indexes
  System.debug('Records modified by stripInaccessible: '+decision.getModifiedIndexes());

// Print removed fields
  System.debug('Fields removed by stripInaccessible: '+decision.getRemovedFields());
```

This will remove the field on which user dont have access.



Execution Log

Timestamp	Event	Details
13:48:38:122	USER_DEBUG	[18]|DEBUG|Insecure record access: Account:{Name=GenePoint, AccountNumber=CC978213, Id=0019000001793 8AAA}
13:48:38:122	USER_DEBUG	[19]|DEBUG|Secure record access: Account:{Name=GenePoint, Id=0019000001793 8AAA}
13:48:38:122	USER_DEBUG	[18]|DEBUG|Insecure record access: Account:{Name=United Oil & Gas, UKUpdated by Batch jobUpdated by Batch jobUpdated by Batch jobUpdated by Batch jo
13:48:38:122	USER_DEBUG	[19]|DEBUG|Secure record access: Account:{Name=United Oil & Gas, UKUpdated by Batch jobUpdated by Batch jobUpdated by Batch jobUpdated by Batch jobUpdated by Batch jo
13:48:38:122	USER_DEBUG	[23]|DEBUG|Records modified by stripInaccessible: {0, 1}
13:48:38:122	USER_DEBUG	[26]|DEBUG|Fields removed by stripInaccessible: {Account={AccountNumber}, Contact={Phone}}

USE CASE 4: DML.

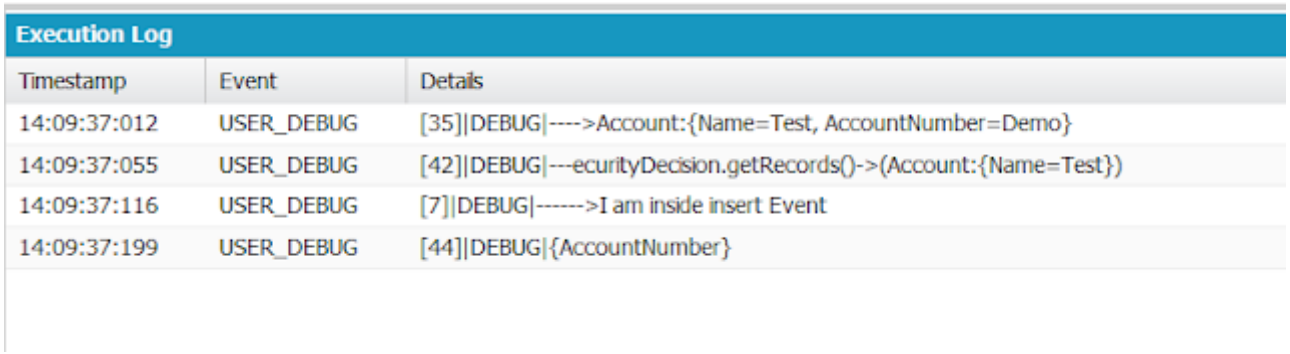
What about if use dont have access on AccountNumber field and we will try to add value of AccountNumber by DML ?

```
public static void testDML(){
    Account acc = new Account(Name='Test', AccountNumber ='TestRating');
    insert acc ;
    System.debug('---->'+acc );
}
```

This will insert the record with AccountNumber value in same user context even user dont have access on same field. How we can stop the same with Security.stripInaccessible. Let see

```
public static void testDML()
{
    Account acc = new Account(Name='Test' ,AccountNumber ='Demo');
    System.debug('---->'+acc );
    List<Account> lstAcc = new List<Account>();
    lstAcc.add(acc);

    SObjectAccessDecision securityDecision = Security.stripInaccessible(
                                        AccessType.CREATABLE,
                                        lstAcc );
    System.debug('---ecurityDecision.getRecords()->'+securityDecision.getRecords());
    insert securityDecision.getRecords();
    System.debug(securityDecision.getRemovedFields().get('Account'));
}
```

here is output



Execution Log

Timestamp	Event	Details
14:09:37:012	USER_DEBUG	[35]|DEBUG|---->Account:{Name=Test, AccountNumber=Demo}
14:09:37:055	USER_DEBUG	[42]|DEBUG|---ecurityDecision.getRecords()->(Account:{Name=Test})
14:09:37:116	USER_DEBUG	[7]|DEBUG|------>I am inside insert Event
14:09:37:199	USER_DEBUG	[44]|DEBUG|{AccountNumber}

You can also use the method to sanitize sObjects that have been deserialized from an untrusted source