DefenseCode ThunderScan SAST Advisory

SugarCRM Community Edition
Multiple SQL Injection Vulnerabilities

SugarCRM Community Edition – Multiple SQL Injection Vulnerabilities	
Advisory ID:	**DC-2018-01-011**
Software:	**SugarCRM Community Edition**
Software Language:	**PHP**
Version:	**6.5.26 and below**
Vendor Status:	**Vendor contacted, vulnerability confirmed**
Release Date:	**2018/01/23**
Risk:	**Medium**

1. General Overview

During the security audit of SugarCRM Community Edition, multiple SQL injection vulnerabilities were discovered using DefenseCode ThunderScan application source code security analysis platform.

More information about ThunderScan is available at URL:
http://www.defensecode.com

2. Software Overview

According to its developers, Sugar offers commercially licensed software built using open source components. This provides users with greater control, ultimate flexibility, and lower lifetime costs. SugarCRM's open source Community Edition allows developers a free and easy entry into learning how to customize and build upon the core Sugar platform, but is not recommended for production use.

Homepage:
https://www.sugarcrm.com

3. Vulnerability Description

During the security analysis, ThunderScan discovered multiple SQL injection vulnerabilities in SugarCRM Community Edition.

3.1 SQL injection in parameter $_REQUEST['track']	
Vulnerable Function:	**$db->query()**
Vulnerable Variable:	**$_REQUEST['track']**

File: SugarCE-Full-6.5.26\modules\Campaigns\Tracker.php

```
if(empty($_REQUEST['track'])) {
    $track = "";
} else {
    $track = $_REQUEST['track'];            <------------------ USER INPUT WHERE
VULNERABILITY STARTS
}
if(!empty($_REQUEST['identifier'])) {
    $keys=log_campaign_activity($_REQUEST['identifier'],'link',true,$track);

}else{
    //if this has no identifier, then this is a web/banner campaign
    //pass in with id set to string 'BANNER'
    $keys=log_campaign_activity('BANNER','link',true,$track);  <-----------------------
VULNERABLE FUNCTION CALL

}

$track = $db->quote($track);

if(preg_match('/^[0-9A-Za-z\-]*$/', $track))
{
    $query = "SELECT tracker_url FROM campaign_trkrs WHERE id='$track'";
    $res = $db->query($query);
```

File: SugarCE-Full-6.5.26\modules\Campaigns\utils.php

```
function log_campaign_activity($identifier, $activity, $update=true, $clicked_url_key=null)
{

    $return_array = array();

    $db = DBManagerFactory::getInstance();



     //check to see if the identifier has been replaced with Banner string
    if($identifier == 'BANNER' && isset($clicked_url_key)  && !empty($clicked_url_key))
    {
        // create md5 encrypted string using the client ip, this will be used for tracker id
purposes
        $enc_id = 'BNR'.md5($_SERVER['REMOTE_ADDR']);

        //default the identifier to ip address
        $identifier = $enc_id;

        //if user has chosen to not use this mode of id generation, then replace identifier
with plain guid.
        //difference is that guid will generate a new campaign log for EACH CLICK!!
        //encrypted generation will generate 1 campaign log and update the hit counter for
each click
```

```
        if(isset($sugar_config['campaign_banner_id_generation'])  &&
$sugar_config['campaign_banner_id_generation'] != 'md5'){
            $identifier = create_guid();
        }

        //retrieve campaign log.
        $trkr_query = "select * from campaign_log where target_tracker_key='$identifier' and
related_id = '$clicked_url_key'"; <------------- RAW SQL QUERY FORMATION
        $current_trkr=$db->query($trkr_query);  <----------------------------------------
-------- SQL INJECTION
        $row=$db->fetchByAssoc($current_trkr);
```

3.2 SQL injection in parameter $_REQUEST['default_currency_name']

Vulnerable Function: **$this->db->query()**
Vulnerable Variable: **$_REQUEST['default_currency_name']**

File: SugarCE-Full-6.5.26\modules\Configurator\controller.php

```
$currency = new Currency;
$currency->retrieve($currency->retrieve_id_by_name($_REQUEST['default_currency_name'])); <--
----- USER INPUT COMING HERE
```

File: SugarCE-Full-6.5.26\modules\Currencies\Currency.php

```
function retrieve_id_by_name($name) {
        $query = "select id from currencies where name='$name' and deleted=0;"; <-----------
--- USER INPUT COMING HERE AS $name VARIABLE
        $result = $this->db->query($query);  <---------------------- SQL INJECTION
VULNERABILITY
        if($result){
```

3.3 SQL injection in parameter $_POST['duplicate']

Vulnerable Function: **$db->query()**
Vulnerable Variable: **$_POST['duplicate']**

File: SugarCE-Full-6.5.26\modules\Contacts\ShowDuplicates.php

```
$duplicates = $_POST['duplicate'];       <------------- USER INPUT COMES FROM HERE
$count = count($duplicates);
if ($count > 0)
{
    $query .= "and (";
    $first = true;
    foreach ($duplicates as $duplicate_id)
    {
        if (!$first) $query .= ' OR ';
        $first = false;
        $query .= "id='$duplicate_id' ";     <--------------- USER INPUT STACKED TO THE RAW
SQL QUERY
    }
    $query .= ')';
}

$duplicateContacts = array();

$db = DBManagerFactory::getInstance();
$result = $db->query($query);   <----------------------- SQL INJECTION VULNERABILITY
```

3.4 SQL injection in parameter $_REQUEST['mergecur']

Vulnerable Function: **$this->db->query()**
Vulnerable Variable: **$_REQUEST['mergecur']**

File: SugarCE-Full-6.5.26\modules\Currencies\index.php

```
$currencies = $_REQUEST['mergecur'];   <---------------- USER INPUT COMES FROM HERE

$opp = new Opportunity();
$opp->update_currency_id($currencies, $_REQUEST['mergeTo'] ); <------- HERE IS PASSED
```

File: SugarCE-Full-6.5.26\modules\Opportunities\Opportunity.php

```
function update_currency_id($fromid, $toid){
    $idequals = '';

    $currency = new Currency();
    $currency->retrieve($toid);
    foreach($fromid as $f){
        if(!empty($idequals)){
            $idequals .=' or ';
        }
        $idequals .= "currency_id='$f'";         <---- USER INPUT IS STACKED to $idequals
varialbe
    }

    if(!empty($idequals)){
        $query = "select amount, id from opportunities where (". $idequals. ") and deleted=0
and opportunities.sales stage <> 'Closed Won' AND opportunities.sales stage <> 'Closed
Lost';"; <----- CONCATED HERE
        $result = $this->db->query($query);   <------------------- SQL Injection
        while($row = $this->db->fetchByAssoc($result)){
```

3.5 SQL injection in parameter $_POST['load_signed_id']

Vulnerable Function: **$this->db->query()**
Vulnerable Variable: **$_POST['load_signed_id']**

File: SugarCE-Full-6.5.26\modules\Documents\Document.php

```
if ((isset($_POST['load_signed_id']) and !empty($_POST['load_signed_id']))) {
    $query="update linked_documents set deleted=1 where id='".$_POST['load_signed_id']."'";
<---- USER INPUT
    $this->db->query($query);   <---- SQL INJECTIOn
}
```

4. Solution

Vendor should resolve the security issues in next release. All users are strongly advised to update SugarCRM Community Edition to the latest available version as soon as the vendor releases an update that fixes the vulnerability.

5. Credits

Discovered by Leon Juranic using DefenseCode ThunderScan source code security analyzer.

6. Disclosure Timeline

2017/09/25	**Vendor contacted**
2017/09/26	**Vendor contacted on more addresses**
2017/09/28	**Vendor responded. Fix expected.**
2017/10/10	**Vendor contacted**
2018/01/23	**Advisory released to the public**

7. About DefenseCode

DefenseCode L.L.C. delivers products and services designed to analyze and test web, desktop and mobile applications for security vulnerabilities.

DefenseCode ThunderScan is a SAST (Static Application Security Testing, WhiteBox Testing) solution for performing extensive security audits of application source code. ThunderScan performs fast and accurate analyses of large and complex source code projects delivering precise results and low false positive rate.

DefenseCode WebScanner is a DAST (Dynamic Application Security Testing, BlackBox Testing) solution for comprehensive security audits of active web applications. WebScanner will test a website's security by carrying out a large number of attacks using the most advanced techniques, just as a real attacker would.

Subscribe for free software trial on our website http://www.defensecode.com

E-mail: defensecode[at]defensecode.com

Website: http://www.defensecode.com
Twitter: https://twitter.com/DefenseCode/