Talos Vulnerability Report

TALOS-2020-1067

Glacies IceHRM Admin Reports SQL injection Vulnerability

JULY 10, 2020

CVE NUMBER

CVE-2020-6114

Summary

An exploitable SQL injection vulnerability exists in the Admin Reports functionality of Glacies IceHRM v26.6.0.OS (Commit bb274de1751ffb9d09482fd2538f9950a94c510a) . A specially crafted HTTP request can cause SQL injection. An attacker can make an authenticated HTTP request to trigger this vulnerability.

Tested Versions

Glacies IceHRM v26.6.0.OS (Commit bb274de1751ffb9d09482fd2538f9950a94c510a)

Product URLs

https://icehrm.com/

CVSSv3 Score

6.6 - CVSS:3.0/AV:N/AC:L/PR:H/UI:N/S:C/C:L/I:L/A:L

CWE

CWE-89 - Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Details

IceHrm is free online HR Software with Leave Management, Recruitment and Payroll supporting many countries and a lot of other functionality to cover all HR management needs such as timesheet, attendance and documents.

An SQL injection has been found and confirmed within IceHrm. A successful attack could allow an attacker to access information such as usernames and password hashes that are stored in the database.

The parameter `ob` in `/app/data.php` suffers from an SQL injection. As demonstrated by the following GET request

```
GET /icehrm/app/data.php?t=Report&sm={}&cl=
[%22id%22,%22icon%22,%22name%22,%22details%22,%22parameters%22]&ft=%22{\%22type\%22:\%22Reports\%22}%22&ob=report_group%2c[SQL Injection]
&sorting=0&sEcho=1&iColumns=6&sColumns=&iDisplayStart=0&iDisplayLength=15&mDataProp_0=0&mDataProp_1=1&mDataProp_2=2&mDataProp_3=3&mDataProp_
4=4&mDataProp_5=5&sSearch=&bRegex=false&sSearch_0=&bRegex_0=false&bSearchable_0=true&sSearch_1=&bRegex_1=false&bSearchable_1=true&sSearch_2=
&bRegex_2=false&bSearchable_2=true&sSearch_3=&bRegex_3=false&bSearchable_3=true&sSearch_4=&bRegex_4=false&bSearchable_4=true&sSearch_5=&bReg
ex_5=false&bSearchable_5=true&iSortCol_0=0&sSortDir_0=asc&iSortingCols=1&bSortable_0=true&bSortable_1=false&bSortable_2=true&bSortable_3=tru
e&bSortable_4=true&bSortable_5=true&_=1570226150837 HTTP/1.1
Host: [IP]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:69.0) Gecko/20100101 Firefox/69.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
DNT: 1
Connection: close
Referer: http://[IP]/icehrm/app/?g=admin&n=reports&m=admin_Reports
Cookie: PHPSESSID=XXX
```

The vulnerable Code is located across a number of source files which eventually leads to the SQL injection. Exploitation of this vulnerability starts in `core/data.php` at line 52 where the order by (`ob`) parameter is passed to `core` functionality in `core/src/Classes/BaseService.php` to formulate part of the order by part of the query.

First, the `ob` parameter forms part of function call to BaseService in 'core/data.php':

```
 42 $skipProfileRestriction = false;
 43 if (isset($_REQUEST['skip']) && $_REQUEST['type'] = "1") {
 44     $skipProfileRestriction = true;
 45 }
 46
 47 $sortData = \Classes\BaseService::getInstance()->getSortingData($_REQUEST);
 48 $data = \Classes\BaseService::getInstance()->getData(
 49     $_REQUEST['t'],
 50     $_REQUEST['sm'],
 51     $_REQUEST['ft'],
 52     $_REQUEST['ob'],
 53     $sLimit,
 54     $_REQUEST['cl'],
 55     $_REQUEST['sSearch'],
 56     $isSubOrdinates,
 57     $skipProfileRestriction,
 58     $sortData
 59 );
```

Next, the `ob` parameter, is transated into the `$orderBy` paramteer by the getData function and forms the base of 'order by' query as seen on line 323 in `core/src/Classes/BaseService.php`:

```
316
317         if (!empty($sortData) && $sortData['sorting']."" == "1" && isset($sortData['column'])) {
318             $orderBy = " ORDER BY ".$sortData['column']." ".$sortData['order'];
319         } else {
320             if (empty($orderBy)) {
321                 $orderBy = "";
322             } else {
323                 $orderBy = " ORDER BY ".$orderBy;
324             }
325         }
```

Finally, the '$orderBy' parameter ends up as part of 'Find' function which calls the query as seen below on line 499:

```
497             }
498         } else {
499             $list = $obj->Find("1=1".$query.$orderBy.$limit, $queryData);
500         }
```

The Find function then builds the query that gets executed in the database and uses the `$orderBy` variable without further sanitization, ending up in the following query for this vulnerability, allowing an attacker to inject SQL statements:

```
select * from Reports WHERE 1=1 and report_group <> 'Payroll' ORDER BY $orderBy LIMIT 0, 15
```

Timeline

2020-05-04 - Vendor Disclosure
2020-05-07 - Vendor acknowledged issue under review
2020-06-01 - Vendor patched
2020-07-10 - Public Release

CREDIT

Discovered by Yuri Kramarz of Cisco Talos.
