Accessibility: Remote
Severity: Critical
Author: Wolfgang Hotwagner (AIT Austrian Institute of Technology)

SUMMARY

QCubed is a PHP Model-View-Controller Rappid Application Development framework. (https://github.com/qcubed/qcubed)

VULNERABILITY DESCRIPTION

A SQL injection vulnerability in qcubed (all versions including 3.1.1) in profile.php via the strQuery parameter allows an unauthenticated attacker to access the database by injecting SQL code via a crafted POST request.

strQuery parameter of the serialized array in profile.php could lead to a sql-injection:

https://github.com/qcubed/qcubed/blob/master/assets/php/profile.php

```
171 <?php
172             $intIndex = 1;
173
174             foreach( $objProfileArray as $objProfile) {
175                     $objBacktrace = $objProfile['objBacktrace'];
176                     $strQuery = $objProfile['strQuery'];
177                     $dblTimeInfo = $objProfile['dblTimeInfo'];
178
179                     $objArgs = (array_key_exists('args', $objBacktrace)) ? $objBacktrace['args'] : array();
180                     $strClass = (array_key_exists('class', $objBacktrace)) ? $objBacktrace['class'] : null;
181                     $strType = (array_key_exists('type', $objBacktrace)) ? $objBacktrace['type'] : null;
182                     $strFunction = (array_key_exists('function', $objBacktrace)) ? $objBacktrace['function'] : null;
183                     $strFile = (array_key_exists('file', $objBacktrace)) ? $objBacktrace['file'] : null;
184                     $strLine = (array_key_exists('line', $objBacktrace)) ? $objBacktrace['line'] : null;
185 ?>
186                     <span class="function">
187                             Called by <?php _p($strClass . $strType . $strFunction . '(' . implode(', ', $objArgs) . ')');
188                     </span>  <br/>
189                     <span class="function_details">
190                             <b>File: </b><?php _p($strFile); ?>;   <b>Line: </b><?php _p($strLine); ?>
191                     </span>
192 <?php
193                     //mark slow query - those that take over 1 second
194                     if($dblTimeInfo >= 1)
195                             echo('<div class="time slow">');
196                     else
197                             echo'<div class="time">';
198                     printf("Query took %.1f ms", $dblTimeInfo * 1000);
199 ?>
200                     <?php $explainStatement = PrintExplainStatement($strQuery); ?>
201                     <a href="#" onClick="return Toggle('query<?php _p($intIndex); ?>')" id="buttonquery<?php _p($intIndex);
202                             Show SQL
203                     </a>
```

1. pass the sql-statement from $_POST

2. pass the sql-statement to DB

(/fileadmin/mc/digital_safety_security/images/CVE2.jpg)

Inside the PrintExplainStatement()-function the query is passed to the ExplainStatement-method of the specific database-provider:

```
qcubed/qcubed/blob/master/assets/php/profile.php

3
4           //Exit gracefully if called directly or profiling data is missing.
5           if ( !isset($_POST['intDatabaseIndex']) && !isset($_POST['strProfileData'])
6                   exit('Nothing to profile. No Database Profiling data recived.');
7
8           if ( !isset($_POST['intDatabaseIndex']) || !isset($_POST['strProfileData'])
9                   throw new Exception('Database Profiling data appears to have been co
10
11          $intDatabaseIndex = intval($_POST['intDatabaseIndex']);
12          $strReferrer = QApplication::HtmlEntities($_POST['strReferrer']);
13
14          $objProfileArray = unserialize(base64_decode($_POST['strProfileData']));
15          $objProfileArray = QType::Cast($objProfileArray, QType::ArrayType);
16          $intCount = count($objProfileArray);
17
18          function PrintExplainStatement($strOriginalQuery) {
19                  global $intDatabaseIndex;
20                  if (substr_count($strOriginalQuery, "AUTOCOMMIT=1") > 0) {
21                          return null;
22                  }
23                  $result = "";
24
25                  $objDb = QApplication::$Database[$intDatabaseIndex];
26                  $objDbResult = $objDb->ExplainStatement($strOriginalQuery);
27                  if (!$objDbResult) {
28                          return "";
29                  }
30
```

[/fileadmin/mc/digital_safety_security/images/CVE2_2.jpg]

The ExplainStatement is a method of the specific database-provider. We tested the sql-injection with postgres and mysql(i). This could be exploited with both databases. Let's see how it looks in the postgres-provider:

```
https://github.com/qcubed/qcubed/blob/master/includes/database/QPostgreSqlDatabase.class.php

492                 }
493
494                 public function ExplainStatement($sql) {
495                         return $this->Query("EXPLAIN " . $sql);
496                 }
497         }
498
499         /**
500          * Database exception class
```

Postgres-Database

[/fileadmin/mc/digital_safety_security/images/CVE2_3.jpg]

VENDOR CONTACT TIMELINE

2020-04-19	Anrufen (tel:0043505500)	Contacting the vendor	E-Mail	Standorte (/ueber-das-ait/standorte-und-tochterunternehmen)

https://github.com/qcubed/qcubed/blob/master/includes/framework/QDatabaseBase.class.php

```php
536                 * Sends the 'SELECT' query to the database and returns the result
537                 *
538                 * @param string $strQuery query string
539                 *
540                 * @return QDatabaseResultBase
541                 */
542                public final function Query($strQuery) {
543                        $timerName = null;
544                        if (!$this->blnConnectedFlag) {
545                                $this->Connect();
546                        }
547
548
549                        if ($this->blnEnableProfiling) {
550                                $timerName = 'queryExec' . mt_rand() ;
551                                QTimer::Start($timerName);
552                        }
553
554                        $result = $this->ExecuteQuery($strQuery);
555
556                        if ($this->blnEnableProfiling) {
557                                $dblQueryTime = QTimer::Stop($timerName);
558                                QTimer::Reset($timerName);
559
560                                // Log Query (for Profiling, if applicable)
561                                $this->LogQuery($strQuery, $dblQueryTime);
562                        }
563
564                        return $result;
565                }
566
```

Query is passed unfiltered to ExecuteQuery

[/fileadmin/mc/digital_safety_security/images/CVE2_4.jpg]

In ExecuteQuery it finally gets executed. Unfiltered:

https://github.com/qcubed/qcubed/blob/master/includes/database/QPostgreSqlDatabase.class.php

```php
239                }
240
241                protected function ExecuteQuery($strQuery) {
242                        // Perform the Query
243                        $objResult = pg_query($this->objPgSql, $strQuery);
244                        if (!$objResult)
245                                throw new QPostgreSqlDatabaseException(pg_last_error(), 0, $strQuery);
246
247                        // Return the Result
248                        $this->objMostRecentResult = $objResult;
249                        $objPgSqlDatabaseResult = new QPostgreSqlDatabaseResult($objResult, $this);
250                        return $objPgSqlDatabaseResult;
251                }
252
```

[/fileadmin/mc/digital_safety_security/images/CVE2_5.jpg]

We were able to write proof-of concept exploit for mysql and postgres. Unfortunately with mysql we were not able to use a stacked-queries-payload and we had to exploit this vulnerability with a timebased approach.

VULNERABLE VERSIONS

All versions including 3.1.1 are affected.

TESTED VERSIONS

QCubed 3.1.1

IMPACT

An unauthenticated attacker could access the database remotely. In worst case scenarios an attacker might be able to execute code on the remote machine.

MITIGATION

A patch was delivered by QCubed that allows to disable the profile-functionality. (https://github.com/qcubed/qcubed/pull/1320/files)

2020-04-19	Vendor replied
2020-05-01	Vendor released a patch at Github
2021-02-15	Public disclosure

ADVISORY URL

https://www.ait.ac.at/ait-sa-20210215-02-unauthenticated-sql-injection-qcubed (https://www.ait.ac.at/ait-sa-20210215-02-unauthenticated-sql-injection-qcubed)

WOLFGANG HOTWAGNER

Research Engineer /
Security & Communication Technologies

+43 664 88335483 (tel:+43 664 8833 5483)

+43 50550-4150

wolfgang.hotwagner(at)ait.ac.at (mailto:wolfgang.hotwagner@ait.ac.at)

WACA Web Accessibility Certificate Austria

www.waca.at

zertifiziert **09/2021**

Domain **ait.ac.at**

© 2022 AIT Austrian Institute of Technology