MOPS-2010-023: Cacti Graph Viewer SQL Injection Vulnerability

May 13th, 2010

An SQL Injection vulnerability was discovered in Cacti that allows to retrieve all data from the database. In Cacti installations with publicly viewable graphs this vulnerability is a pre-auth SQL injection vulnerability.

Affected versions

Affected is Cacti <= 0.8.7e

Risk

High.

Credits

The vulnerability was discovered by Stefan Esser as part of the SQL Injection Marathon.

About Cacti

Cacti is a complete network graphing solution designed to harness the power of RRDTool's data storage and graphing functionality. Cacti provides a fast poller, advanced graph templating, multiple data acquisition methods, and user management features out of the box. All of this is wrapped in an intuitive, easy to use interface that makes sense for LAN-sized installations up to complex networks with hundreds of devices.

Detailed information

This vulnerability was discovered during SQL Injection Marathon a PHP code auditing marathon performed by Stefan Esser. The basic idea of this initiative is to select random PHP applications and perform a short code audit on them. The maximum time spent on each application is 30 minutes and after the first found SQL injection usually the next application is audited.

During SQL Injection Marathon Cacti was also audited and due to previous knowledge about the source code structure it was possible to find a SQL injection vulnerability caused by wrong variable filtering in less than 30 minutes. The offending code is located in graph.php which is depending on the configuration accessible by all.

```
case 'zoom':
   /* find the maximum time span a graph can show */
   $max_timespan=1;
   if (sizeof($rras) > 0) {
      foreach ($rras as $rra) {
         if ($rra["steps"] * $rra["rows"] * $rra["rrd_step"] > $max_timespan) {
            $max_timespan = $rra["steps"] * $rra["rows"] * $rra["rrd_step"];
         }
      }
   }

   /* fetch information for the current RRA */
   $rra = db_fetch_row("select id,timespan,steps,name from rra where id=" . $_GET["rra_id"]);
```

It is quite obvious here that `$_GET['rra_id']` is inserted into the SQL query directly, which usually results in an SQL injection vulnerability. However knowing the source code structure of Cacti one knows that it is usually required to go to scroll backwards and check if there are any filters. And indeed, when scrolling backwards in the file one will find.

```
/* ================= input validation ================= */
input_validate_input_regex(get_request_var_request("rra_id"), "^([0-9]+|all)$");
input_validate_input_number(get_request_var("local_graph_id"));
input_validate_input_regex(get_request_var_request("view_type"), "^([a-zA-Z0-9]+)$");
/* ==================================================== */
```

From that piece of code it seems that the ‘rra_id’ parameter is verified to be either ‘all’ or a number. But is it really? In order to check that it is necessary to look deeper into the code. So lets start with looking into the input_validate_input_regex() function.

```
function input_validate_input_regex($value, $regex) {
   if ((!ereg($regex, $value)) && ($value != "")) {
      die_html_input_error();
   }
}
```

In this function we can see that the deprecated ereg() function is used to check if the value is okay or if Cacti should error out with a validation error. So is the use of the deprecated binary unsafe ereg() function a problem here?

No it is not, because Cacti will emulate magic_quotes_gpc=On behaviour if it is deactivated which results in all NUL-bytes in the input the be converted to `'\0'`. So this is not an SQL injection vulnerability after all?

Well it is an SQL injection vulnerability, but the cause it not a wrong regex filter. The cause the wrong value being filtered. In order to understand that it is necessary to look into the get_request_var_request() function, because its return value is checked.

```
function get_request_var_request($name, $default = "")
{
   if (isset($_REQUEST[$name]))
   {
      return $_REQUEST[$name];
   } else
   {
      return $default;
   }
}
```

We can see here that the value that is filtered is taken from `$_REQUEST['rra_id']`. But wait a second that is not the same as `$_GET['rra_id']`, which is used in the SQL query. And yes this is indeed what results in the SQL injection vulnerability. `$_REQUEST['rra_id']` will only contain the same value as `$_GET['rra_id']` if there is no POST or COOKIE variable with the same name. This means all an attacker needs to exploit this SQL injection vulnerability is to submit his SQL injection payload through the URL variable ‘rra_id’ and in addition to that send a harmless ‘rra_id’ variable via POST or COOKIE.

Proof of concept, exploit or instructions to reproduce

No proof of concept will be provided for this vulnerability.

Notes

This vulnerability has been disclosed to the Cacti authors at the same time as the public.

We strongly recommend to the Cacti authors to rethink their current handling of security issues. Right now they do not produce a new security patched Cacti version, although there have been plenty of patches since the release. Instead they have a website listing official patches one should apply to the current version of Cacti.