



## MiniWeb Multiple Vulnerabilities



The open-source mini HTTP server - Small and elegant

### Introduction

MiniWeb is a mini HTTP server implementation written in C language, featuring low system resource consumption, high efficiency, good flexibility and high portability.

It is capable to serve multiple clients with a single thread, supporting GET and POST methods, authentication, dynamic contents (dynamic web page and page variable substitution) and file uploading. MiniWeb runs on POSIX complaint OS, like Linux, as well as Microsoft Windows.

vulnerability discovered by : Hamid Ebadi (ebadi\_AT\_bugtraq.ir)  
<http://www.bugtraq.ir>

vulnerable version :MiniWeb 0.8.19 (C)2005 Written by Stanley Huang  
<http://miniweb.sourceforge.net/>  
<http://sourceforge.net/projects/miniweb>

### Description:

#### directory traversals :

An input validation error in the URL request handling in mwGetLocalFileName() function ( http.c) can be exploited to disclose arbitrary files (and also Directory listing) outside the web root via directory traversals attacks via the ".%2e/" or "%2e%2e/" sequences

#### Proof of Concept :

Directory listing:

<http://127.0.0.1:80/.%2e/.%2e/.%2e/.%2e/.%2e/.%2e/>

disclose arbitrary files:

<http://127.0.0.1:80/.%2e/.%2e/.%2e/.%2e/.%2e/.%2e/boot.ini>

let's take look at the original mwGetLocalFileName() function in http.c , line 162

```
int mwGetLocalFileName(HttpFilePath* hfp)
{
    char ch;
    char *p=hfp->cFilePath,*s=hfp->pchHttpPath,*upLevel=NULL;

    hfp->pchExt=NULL;
    hfp->fTailSlash=0;
    if (hfp->pchRootPath) {
        p=_mwStrCopy(hfp->cFilePath,hfp->pchRootPath);
        if (*(p-1)!=SLASH) {
            *p=SLASH;
            *(++p)=0;
        }
    }
    while ((ch=*s) && ch!='?' && (int)p-(int)hfp->cFilePath<sizeof(hfp->cFilePath)-1) {
        if (ch=='%') {
            *(p++) = _mwDecodeCharacter(++s);
            s += 2;
        } else if (ch=='/') {
            *p=SLASH;
            upLevel=(++p);
            while (*(++s)=='/');
            continue;
        } else if (ch=='+') {
            *(p++)=' ';
            s++;
        } else if (ch=='.') {
            if (upLevel && GETWORD(s+1)==DEFWORD('.', '/')) {
                s+=2;
                p=upLevel;
            } else {
                *(p++)='.';
                hfp->pchExt=p;
                while (*(++s)=='.'); //avoid '..' appearing in filename for
security issue
            }
        } else {
            *(p++)=*(s++);
        }
    }
    if (*(p-1)==SLASH) {
        p--;
        hfp->fTailSlash=1;
    }
    *p=0;
    return (int)p-(int)hfp->cFilePath;
}
```

I think it's not clear for you , and unfortunately my English is not well , so let me add comments , that the best way to let you understand what I mean gaze at the if-else-if statements (Note : I also add my simple patch in the source code )

```

int mwGetLocalFileName(HttpFilePath* hfp)
{
    char ch;
    char *p=hfp->cFilePath,*s=hfp->pchHttpPath,*upLevel=NULL;
    //pchHttpPath : file path in the http request (GET PATH HTTP/1.1)
    //cFilePath = we fill it with cphRootPath+pchHttpPath
    hfp->pchExt=NULL;
    hfp->fTailSlash=0;
    if (hfp->pchRootPath) {
        p+=_mwStrCopy(hfp->cFilePath,hfp->pchRootPath);
        // _mwStrCopy : copy src to dsc and return the size of src string

        // here we check if after root path we have "/" ?
        // if does not , add slash+NULL)at the end
        if (*(p-1)!=SLASH) {
            *p=SLASH;
            *(++p)=0;
        }
    }
    // we have filled hfp->cFilePath with hfp->pchRootPath
    // here "p" points to the end of hfp->cFilePath
    // and "s" points to the hfp->pchHttpPath

    while ((ch=*s)
        && ch!='?' // repeat untill seeing "?" or NULL (http://site/file?arg)
        && (int)p-(int)hfp->cFilePath<sizeof(hfp->cFilePath)-1) //
    {
        if (ch=='%') { // check for unicodes like %2e

            *(p++) = _mwDecodeCharacter(++s); // if it's unicode return the char
            represent the %2e (as you know %2e='.' )
            // added by Hamid Ebadi http://www.bugtraq.ir
            // simple patch to avoid double dot directory traversal
            if (*(p-1)=='.' &&*(p-2)=='.' ) p--;
            // as you can see _mwDecodeCharacter can return "." but since we have "if-
            else-if" statement ,else if (ch=='.') can not catch this dot
            s += 2; // we have 2 character (number) after %
        } else if (ch=='/') {
            *p=SLASH;
            upLevel=(++p); // and save address of last "/"
            while (*(++s)=='/');// don't care how many "/" you see
            continue;
        } else if (ch=='+') { // check for space character
            *(p++)=' ';
            s++;
        } else if (ch=='..') {
            //define DEFWORD(char1,char2) (char1+(char2<<8))
            // so our WORD(16 bit) will be (8 bit char2) + (8 bit char1)
            //define GETWORD(ptrData) (*(WORD*)(ptrData))

            // if we set uplevel and we see "/"
            if (upLevel && GETWORD(s+1)==DEFWORD('.', '/')) {
                s+=2;
                p=upLevel;// came back to the last "/"
            } else {
                *(p++)='.'; // double dot ".."
                hfp->pchExt=p;
                while (*(++s)=='.'); //avoid '..' appearing in filename for
                security issue
            }
        } else {
            *(p++)=*s++; // it's normal alphabets , just copy them
        }
    }
    if (*(p-1)==SLASH) {
        p--;
        hfp->fTailSlash=1;
    }
    *p=0;
    return (int)p-(int)hfp->cFilePath;
}

```

## Heap based buffer overflow vulnerability

There is also heap based buffer overflow in this web server

The vulnerability is caused due to a boundary error in `_mwProcessReadSocket()` function (`http.c`) when handling HTTP requests.

This can be exploited by sending an overly long, specially crafted request, which can cause a heap overflow and allow arbitrary code execution with the privileges of the web service.

Proof of Concept :

```
GET /AAAA...[3600 - 4000]...AAAA/ HTTP/1.0
```

Before start talking about vulnerability :

`Httpint.h` : line 76

```
#define MAX_REQUEST_PATH_LEN (512/*bytes*/)
#define MAX_REQUEST_SIZE (2*1024 /*bytes*/)
```

And also look at the `httpint.h` :line 187

```
#define HTTP_BUFFER_SIZE (4*1024 /*bytes*/)

// per connection/socket structure
typedef struct _HttpSocket{
    struct _HttpSocket *next;
    SOCKET socket;
    int fd;
    HttpRequest request;
    HttpResponse response;
    unsigned char *pucData;
    int iDataLength;

    unsigned long flags;
    void* ptr;
    time_t tmAcceptTime;
    time_t tmExpirationTime;
    int iRequestCount;

    unsigned char buffer[HTTP_BUFFER_SIZE];
} HttpSocket;
```

In the above code we can see structure of sockets link-list, and also reserve 4\*1024 chars for our buffer ( if I were him I would use pointer since there is no difference between request with size of 50 bytes and 40000 bytes, and don't forget if I were him result will be spaghetti code , so take it easy ;-)

I think the problem is in `_mwProcessReadSocket()` function in `http.c` line 638

In this function we can see if `(phsSocket->iDataLength)>=MAX_REQUEST_SIZE` we don't accept the connection . it sounds good but let's continue ...

[NOTE : I add some comments for better understanding]

```

// p points to the end of buffer so if we send GET 3700*A HTTP/1.1 :
// siHeaderSize ~ 3716
// and p will point to buffer[3716+4]
p=phsSocket->buffer + phsSocket->request.siHeaderSize + 4;
p=(unsigned char*)((unsigned long)p & (-4)); //keep 4-byte aligned
*p=0;
//keep request path
{
    char *q;
    int iPathLen;
    // after this for , q will points to the end of request
    for (q=phsSocket->request.pucPath;*q && *q!=' ';q++);
    // calculate the length of path
    iPathLen=(int)q-(int)(phsSocket->request.pucPath);
    if (iPathLen>=MAX_REQUEST_PATH_LEN) {
        DBG("Request path too long and is stripped\n");
        // if path is too long , we should stripp it to MAX_REQUEST_PATH_LEN=500
        iPathLen=MAX_REQUEST_PATH_LEN-1;
    }
    if (iPathLen>0)
        // add path at the end of buffer (now p points to buffer[3720]
        // we have only 4069-3720=376 empty place in our buffer
        // and we are going to add 512 char of path at the end of buffer ,
        // what will happen ? yes, Heap based bufer Overflow
        memcpy(p,phsSocket->request.pucPath,iPathLen);
    *(p+iPathLen)=0;
    phsSocket->request.pucPath=p;
    p=(unsigned char*)((unsigned long)(p+iPathLen+4+1)&(-4)); //keep 4-byte aligned
}

```

to solve this vulnerability we can change the buffer size in struct \_HttpSocket , so we have enough free space for path(MAX\_REQUEST\_PATH\_LEN (512))

```

typedef struct _HttpSocket{
    struct _HttpSocket *next;
    SOCKET socket;
    int fd;
    HttpRequest request;
    HttpResponse response;
    unsigned char *pucData;
    int iDataLength;

    unsigned long flags;
    void* ptr;
    time_t tmAcceptTime;
    time_t tmExpirationTime;
    int iRequestCount;
    // added by Hamid Ebadi http://www.bugtraq.ir
    unsigned char buffer[HTTP_BUFFER_SIZE+512]; //MAX_REQUEST_PATH_LEN =512
} HttpSocket;

```

this vulnerability allow remote command execution  
(kaveh razavi successfully exploit this vulnerability , thank kaveh )

### How to find ?

First of all I fuzz the server using efuuz  
<http://packetstormsecurity.org/Win2k/efuzz01.zip>

Using :

efuzz localhost 80 http.cfg TCP

and also BED 0.5 by mjm ( [www.codito.de](http://www.codito.de) ) & eric ( [www.snake-basket.de](http://www.snake-basket.de) )

<http://snake-basket.de/bed.html>

Using :

bed.pl -s HTTP -t 127.0.0.1 -p 80 -o 2

but I didn't get any result ( no crash , no Dos )

Questions ( ??? ) :

1- do you know why most of fuzzer (with default setting ) like BED & efuzz can't find this vulnerability ?

2- do you know why this is a Heap based buffer overflow and not a stack based buffer overflow ?

(remember : `unsigned char buffer[HTTP_BUFFER_SIZE];` )

if you can't answer these questions read this article again (or read the source code).

Solution:Edit the source code .

```
//----- Httpint.h -----
```

Change :

```
unsigned char buffer[HTTP_BUFFER_SIZE];
```

to :

```
// added by Hamid Ebadi http://www.bugtraq.ir  
unsigned char buffer[HTTP_BUFFER_SIZE+512]; //MAX_REQUEST_PATH_LEN =512
```

```
//----- http.c -----
```

change :

```
if (ch=='%') {  
    *(p++) = _mwDecodeCharacter(++s);  
    s += 2;
```

to :

```
if (ch=='%') {  
    *(p++) = _mwDecodeCharacter(++s);  
    // added by Hamid Ebadi http://www.bugtraq.ir  
    if (*(p-1)=='.' &&*(p-2)=='.' ) p--;  
    s += 2;
```

Thanks :

Kaveh Razavi , Alireza Danesh ,Mohammad Mozaffary, Ali Abbasi , Amir Barati,Farhad Jafari, Ali Salarpour , Arash Setayeshi , Ali Nejad mazare, Mohammadreza Roohian and all [www.bugtraq.ir](http://www.bugtraq.ir) moderator

Copyright : <http://www.bugtraq.ir>