Professional Pentesting Report
Tiny Web Server

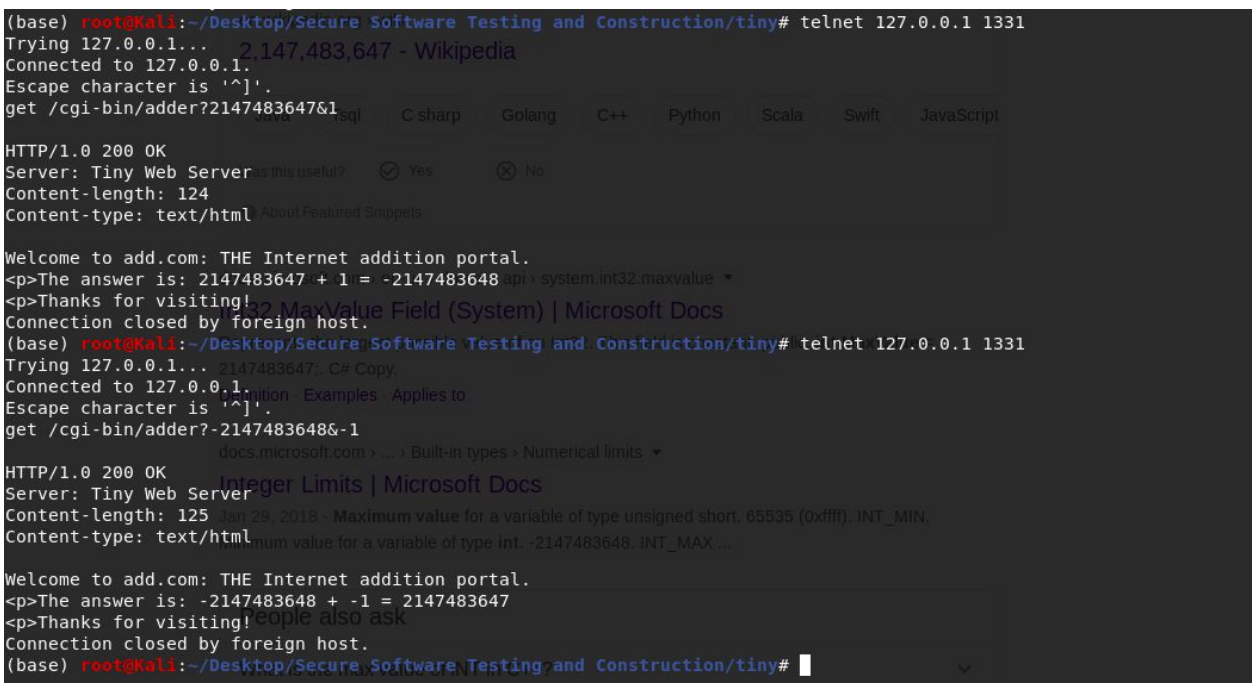
Vulnerability Report

1. Integer Overflow

a. Integer Overflow occurs when an arithmetic operation tries to generate a numeric value that is out of range that can be represented with the given number of digits. The Tiny Web Server is vulnerable to Integer Overflow. This vulnerability was found in adder.c file in the cgi-bin directory.

b. Steps to reproduce the attack

- This vulnerability was exploited by creating a telnet connection and a specific set of inputs were used.
- The attack was successful using INT_MAX and 1 as inputs to the adder file, as it will output INT_MIN. Even inputs such as INT_MIN and -1 exploited the vulnerability, which produced INT_MAX as the output.
- This attack was possible because there was no user input sanitization present.
- Following screenshot represents the successfulness of the attack.



c. Patch

- The vulnerability was exploited because there was no user input sanitization.
- For patching this vulnerability, user input sanitization was incorporated, where each of the integer values and their sums were compared. If both the inputs are positive and the summation is negative (INT_MAX + 1) or if

both inputs are negative and summation is positive, the program would terminate (INT_MIN - 1).

- Following are the screenshots that represent the patch added to the adder.c file and successful execution of the patch.

```c
sprintf(content, "Welcome to add.com: ");
sprintf(content, "%sTHE Internet addition portal.\r\n<p>", content);
int sum =n1 + n2;

if((n1>0 && n2>0 && sum<0) || (n2<0 && n1<0 && sum>0)){
        printf("Integer Overflow Detected!!!");
        exit(0);
}
else
{
sprintf(content, "%sThe answer is: %d + %d = %d\r\n<p>",
    content, n1, n2, n1 + n2);
}
sprintf(content, "%sThanks for visiting!\r\n", content);
//printf("%d",atoi(arg[0]));
/* Generate the HTTP response */
printf("Content-length: %d\r\n", (int)strlen(content));
```

```
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# telnet 127.0.0.1 1331
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /cgi-bin/adder?21474836647&1

HTTP/1.0 200 OK
Server: Tiny Web Server
Integer Overflow Detected!!!Connection closed by foreign host.
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# telnet 127.0.0.1 1331
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /cgi-bin/adder?-2147483648&-1

HTTP/1.0 200 OK
Server: Tiny Web Server
Integer Overflow Detected!!!Connection closed by foreign host.
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny#
```

- Same approach was undertaken while handling POST requests, which will be covered in the later section of the report.

2. File Path Traversal Vulnerability

a. File Path Traversal Vulnerability is a web security vulnerability that allows an attacker to read arbitrary files on the server that is running on an application. This is a serious vulnerability because this might end up allowing an attacker to read confidential files containing sensitive information such as server code, login credentials, config files and many more. The File Path Traversal vulnerability was detected in Tiny Web Server that allows an attacker to read sensitive files such as /etc/passwd, source code, binary and many more.

b. Steps taken to reproduce attack

- This vulnerability was exploited by creating the telnet connection and GET method was used to look for the file.
- Sensitive files in directories such as etc were accessed using "../" string in URI. This string represents jumping in the previous directory. The total number of jumps needed to reach "/" directory were 4 i.e. "/../../../../".
- After successfully guessing the number of jumps to reach the "/" directory, the file was accessed and the vulnerability was exploited.
- Even using "GET /tiny", the vulnerability was exploited as it displayed the entire binary file. Same was with "tiny.c" file which is the source code of the server.
- Following screenshots represents the successful exploitation of the vulnerability.

- File accessed was /etc/passwd.

```
(base) root@kali:~/Desktop/Secure Software Testing and Construction/tiny# telnet 127.0.0.1 1331
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /../../../../etc/passwd

HTTP/1.0 200 OK
Server: Tiny Web Server
Content-length: 3221
Content-type: text/plain

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:102:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
mysql:x:104:110:MySQL Server,,,:/nonexistent:/bin/false
ntp:x:105:111::/nonexistent:/usr/sbin/nologin
messagebus:x:106:112::/nonexistent:/usr/sbin/nologin
arpwatch:x:107:113:ARP Watcher,,,:/var/lib/arpwatch:/bin/sh
Debian-exim:x:108:114::/var/spool/exim4:/usr/sbin/nologin
uuidd:x:109:115::/run/uuidd:/usr/sbin/nologin
redsocks:x:110:116::/var/run/redsocks:/usr/sbin/nologin
tss:x:111:117:TPM2 software stack,,,:/var/lib/tpm:/bin/false
rwhod:x:112:65534::/var/spool/rwho:/usr/sbin/nologin
iodine:x:113:65534::/var/run/iodine:/usr/sbin/nologin
miredo:x:114:65534::/var/run/miredo:/usr/sbin/nologin
dnsmasq:x:115:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
postgres:x:116:121:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
usbmux:x:117:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
rtkit:x:118:123:RealtimeKit,,,:/proc:/usr/sbin/nologin
stunnel4:x:119:127::/var/run/stunnel4:/usr/sbin/nologin
sshd:x:120:65534::/run/sshd:/usr/sbin/nologin
Debian-snmp:x:121:128::/var/lib/snmp:/bin/false
ssln:x:122:129::/nonexistent:/usr/sbin/nologin
```

- Tiny Binary was accessed.

```
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /tiny

HTTP/1.0 200 OK
Server: Tiny Web Server
Content-length: 35728
Content-type: text/plain

ELF>
```

- Tiny Server's Source Code was accessed.

```
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /tiny.c

HTTP/1.0 200 OK
Server: Tiny Web Server
Content-length: 10702
Content-type: text/plain

/* $begin tinymain */
/*
 * tiny.c - A simple, iterative HTTP/1.0 Web server that uses the
 *     GET method to serve static and dynamic content.
 */
#include "csapp.h"
#include<pthread.h>

//int flag=0;
void doit(int fd);
void read_requesthdrs(rio_t *rp);
int parse_uri(char *uri, char *filename, char *cgiargs);
void serve_static(int fd, char *filename, int filesize);
void get_filetype(char *filename, char *filetype);
void serve_dynamic(int fd, char *filename, char *cgiargs);
void clienterror(int fd, char *cause, char *errnum,
                 char *shortmsg, char *longmsg);


struct ThreadArgs{
    int client_sock;
};

void *ThreadMain(void *threadargs){
    int client_sock;
    pthread_detach(pthread_self());
    client_sock = ((struct ThreadArgs *) threadargs) -> client_sock;
    free(threadargs);
    doit(client_sock);
    Close(client_sock);
    return;


}
int main(int argc, char **argv)
{
    int listenfd, connfd, port, clientlen;
    struct sockaddr_in clientaddr;
```

c. Patch

- This vulnerability was patched using user input sanitization.
- The user input was taken using Rio_readinitb() function and was stored in the buffer and entire stored information in the buffer was distributed to different strings such as method, uri and version using sscanf() function, as referred from the source code.
- Uri contained the string given by the attacker after GET method.
- User Input Sanitization was done by checking the substring inside the string using strstr function in C. It would check the presence of "../" substring in the uri string.
- Arbitrary file access was controlled by adding the check of whether the uri contained the relevant substring i.e. Uri which contained "cgi-bin", ".gif", ".jpg", "README", ".html", ".ico" and "/"(For controlling "GET /" and once when an attacker opens the browser).

- What kind of file to be allowed can be checked by running gobuster or dirbuster, and the one with 200 Status code should be allowed.

- Following are the screenshots that represent the patch added in the doit() function of tiny.c file and successful execution of patch.

```
if (strstr(uri,"cgi-bin") == NULL && strstr(uri, "../"))
    {

    clienterror(fd, uri, "403", "Forbidden","File Traversal Detected! Program ends here");     //File Traversal
    return;
    }
```

```
if (strstr(uri,"cgi-bin") || strstr(uri,".gif") || strstr(uri,".jpg") || strstr(uri,"README") || strstr(uri,".html") || strstr(uri,".ico") || uri[strlen(uri)-1] == '/')
{
    is_static = parse_uri(uri, filename, cgiargs);
                                                    //line:netp:doit:staticcheck
    if (stat(filename, &sbuf) < 0) {                //line:netp:doit:beginnotfound
        clienterror(fd, filename, "404", "Not found",
        "Tiny couldn't find this file");
        return;
    }                                                   //line:netp:doit:endnotfound

    if (is_static) { /* Serve static content */
    if (!(S_ISREG(sbuf.st_mode)) || !(S_IRUSR & sbuf.st_mode)) { //line:netp:doit:readable
        clienterror(fd, filename, "403", "Forbidden",
        "Tiny couldn't read the file");
        return;
    }
    serve_static(fd, filename, sbuf.st_size);        //line:netp:doit:servestatic
    }
    else
    { /* Serve dynamic content */
        if (!(S_ISREG(sbuf.st_mode)) || !(S_IXUSR & sbuf.st_mode))
        { //line:netp:doit:executable
            clienterror(fd, filename, "403", "Forbidden",
            "Tiny couldn't run the CGI program");
            return;
        }
        serve_dynamic(fd, filename, cgiargs);                //line:netp:doit:servedynamic
    }

}
else{
    clienterror(fd, uri, "403", "Forbidden",
            "You are not allowed to execute this file!!");
    //flag=1;
    return;
}
```

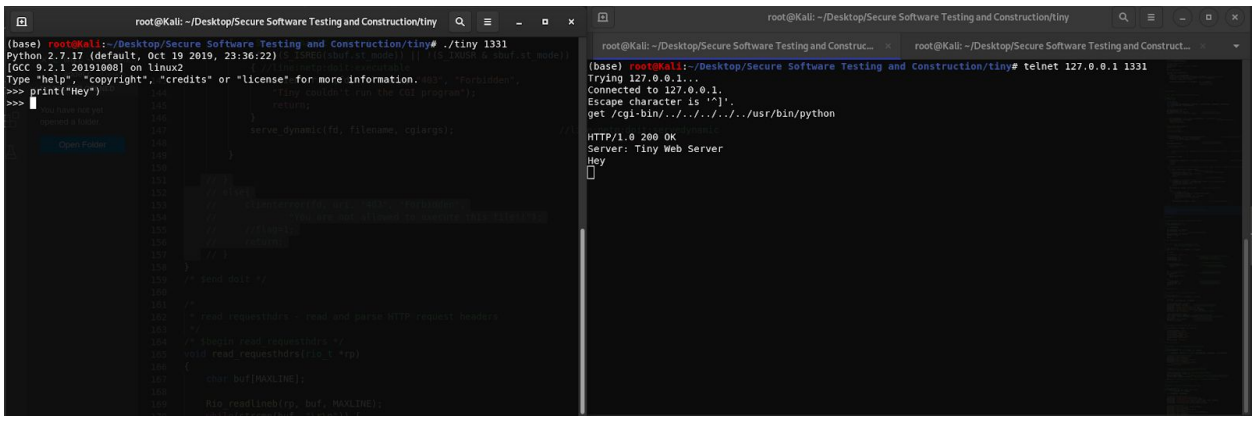
```
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /../../../../etc/passwd

HTTP/1.0 403 Forbidden
Content-type: text/html
Content-length: 203

<html><title>Tiny Error</title><body bgcolor=ffffff>
<h1 style="color:red;">403: Forbidden
<p><h1>File Traversal Detected! Program ends here: /../../../../etc/passwd
<hr><em>The Tiny Web server</em>
Connection closed by foreign host.
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# telnet 127.0.0.1 1331
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /tiny

HTTP/1.0 403 Forbidden
Content-type: text/html
Content-length: 185

<html><title>Tiny Error</title><body bgcolor=ffffff>
<h1 style="color:red;">403: Forbidden
<p><h1>You are not allowed to execute this file!!: /tiny
<hr><em>The Tiny Web server</em>
Connection closed by foreign host.
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# telnet 127.0.0.1 1331
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /tiny.c

HTTP/1.0 403 Forbidden
Content-type: text/html
Content-length: 187

<html><title>Tiny Error</title><body bgcolor=ffffff>
<h1 style="color:red;">403: Forbidden
<p><h1>You are not allowed to execute this file!!: /tiny.c
<hr><em>The Tiny Web server</em>
Connection closed by foreign host.
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny#
```

3. Command Injection Vulnerability

a. Command Injection Vulnerability is a web application vulnerability that allows an attacker to execute arbitrary OS commands on the web application's server. For example, an attacker can execute OS commands such as ping, python, bash and many more. Command Injection Vulnerability was found in Tiny Server in the cgi-bin where an attacker can traverse the file path and reach the usr/bin/ directory to execute OS commands.

b. Steps to reproduce the attack

- The exploitation of this vulnerability was quite same as that of the file path traversal where an attacker enters "/cgi-bin/../../../../../usr/bin/" to reach the binaries directory.
- After jumps were calculated correctly as 5, exploit was executed successfully.
- Python binary was executed from the /usr/bin directory which popped the python shell on the server side.
- Following screenshot represents successful exploitation of the vulnerability.



c. Patch

- This vulnerability was patched basically using the same patch used for file path traversal.
- User input sanitization was done and uri containing "../" were detected.
- Even arbitrary file execution was controlled using almost the same patch as that of file path traversal's patch.
- Following screenshots represents changes in the source code and successful execution of the patch.

```
if (strstr(uri,"cgi-bin") && strstr(uri, "../"))
    {

    clienterror(fd, uri, "403", "Forbidden","Command Injection Detected!");                //Commmand Injection
    return;
    }
```

```c
if (strstr(uri,"cgi-bin") || strstr(uri,".gif") || strstr(uri,".jpg") || strstr(uri,"README") || strstr(uri,".html") || strstr(uri,".ico") || uri[strlen(uri)-1] == '/')
{
    is_static = parse_uri(uri, filename, cgiargs);
                                                    //line:netp:doit:staticcheck
    if (stat(filename, &sbuf) < 0) {                        //line:netp:doit:beginnotfound
        clienterror(fd, filename, "404", "Not found",
        "Tiny couldn't find this file");
        return;
    }                                                       //line:netp:doit:endnotfound

    if (is_static) { /* Serve static content */
    if (!(S_ISREG(sbuf.st_mode)) || !(S_IRUSR & sbuf.st_mode)) { //line:netp:doit:readable
        clienterror(fd, filename, "403", "Forbidden",
        "Tiny couldn't read the file");
        return;
    }
    serve_static(fd, filename, sbuf.st_size);        //line:netp:doit:servestatic
    }
    else
    { /* Serve dynamic content */
        if (!(S_ISREG(sbuf.st_mode)) || !(S_IXUSR & sbuf.st_mode))
        { //line:netp:doit:executable
            clienterror(fd, filename, "403", "Forbidden",
            "Tiny couldn't run the CGI program");
            return;
        }
        serve_dynamic(fd, filename, cgiargs);                //line:netp:doit:servedynamic
    }

}
else{
    clienterror(fd, uri, "403", "Forbidden",
            "You are not allowed to execute this file!!");
    //flag=1;
    return;
}
```

```
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# telnet 127.0.0.1 1331
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /cgi-bin/../../../../../usr/bin/python

HTTP/1.0 403 Forbidden
Content-type: text/html
Content-length: 203

<html><title>Tiny Error</title><body bgcolor=ffffff>
<h1 style="color:red;">403: Forbidden
<p><h1>Command Injection Detected!: /cgi-bin/../../../../../usr/bin/python
<hr><em>The Tiny Web server</em>
Connection closed by foreign host.
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny#
```

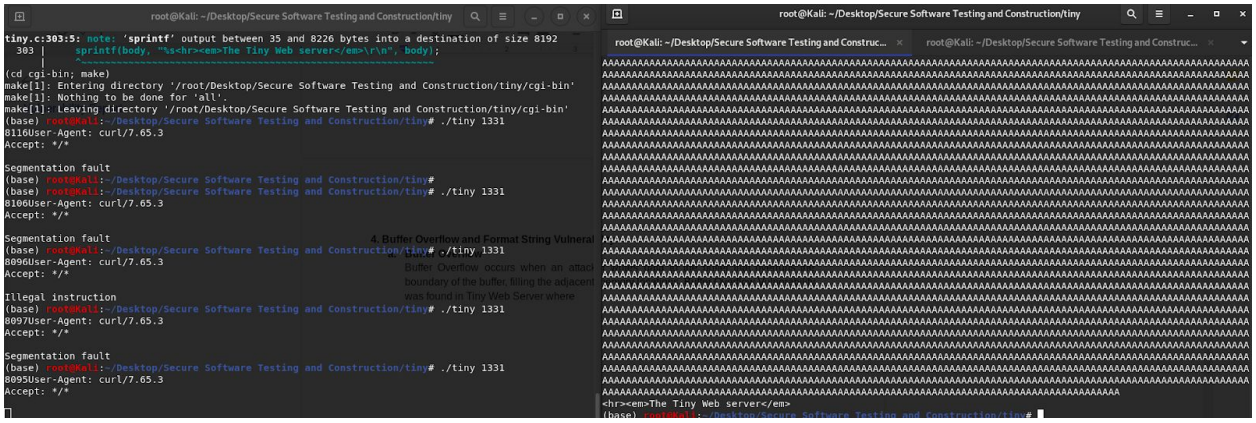
4. Buffer Overflow and Format String Vulnerability

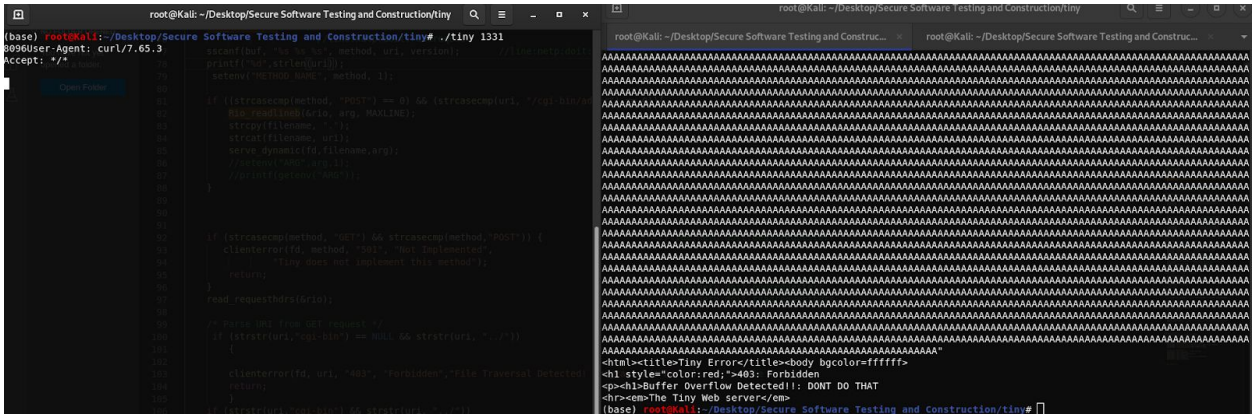
a. Buffer Overflow

Buffer Overflow occurs when an attacker writes data to the buffer that overruns the boundary of the buffer, filling the adjacent memory locations. Buffer Overflow Vulnerability was found in Tiny Web Server in strcpy and strcat function. Overflowing the buffer would crash the server representing Segmentation Fault.

b. Steps taken to reproduce the attack

- The exploit for this vulnerability was developed by checking where the server crashes i.e. on what input length the server crashes.
- Following screenshot represents the trials made in order to check the server status.


root@Kali: ~/Desktop/Secure Software Testing and Construction/tiny
tiny.c:303:5: note: 'sprintf' output between 35 and 8226 bytes into a destination of size 8192
303 | sprintf(body, "%s<hr>The Tiny Web server\r\n", body);
(cd cgi-bin; make)
make[1]: Entering directory '/root/Desktop/Secure Software Testing and Construction/tiny/cgi-bin'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/root/Desktop/Secure Software Testing and Construction/tiny/cgi-bin'
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# ./tiny 1331
8116User-Agent: curl/7.65.3
Accept: */*
Segmentation fault
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny#
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# ./tiny 1331
8106User-Agent: curl/7.65.3
Accept: */*
Segmentation fault
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# ./tiny 1331
8096User-Agent: curl/7.65.3
Accept: */*
Illegal instruction
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# ./tiny 1331
8097User-Agent: curl/7.65.3
Accept: */*
Segmentation fault
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# ./tiny 1331
8095User-Agent: curl/7.65.3
Accept: */*
root@Kali: ~/Desktop/Secure Software Testing and Construction/tiny
root@Kali: ~/Desktop/Secure Software Testing and Construc...
root@Kali: ~/Desktop/Secure Software Testing and Construc...
<hr>The Tiny Web server



a. Format string Vulnerability

This vulnerability was found in Tiny Web Server in clienterror() function which contained ‘cause’ string as an argument, which printed the filename. Attackers can exploit this vulnerability to determine the presence of a file in the server.

b. Steps taken to reproduce the attack

- This vulnerability was exploited by trying to access files that were not present in the server.
- “Cause” argument in clientError() function, caused this exploit to work.

```
void clienterror(int fd, char *cause, char *errnum,
         char *shortmsg, char *longmsg)
{
    char buf[MAXLINE], body[MAXBUF];

    /* Build the HTTP response body */
    sprintf(body, "<html><title>Tiny Error</title>");
    sprintf(body, "%s<body bgcolor=""ffffff""> \r\n", body);
    sprintf(body, "%s<h1 style=\"color:red;\">%s: %s\r\n", body, errnum, shortmsg);
    sprintf(body, "%s<p><h1>%s: %s\r\n", body, longmsg, cause);
    sprintf(body, "%s<hr><em>The Tiny Web server</em>\r\n", body);

    /* Print the HTTP response */
    sprintf(buf, "HTTP/1.0 %s %s\r\n", errnum, shortmsg);
    Rio_writen(fd, buf, strlen(buf));
    sprintf(buf, "Content-type: text/html\r\n");
    Rio_writen(fd, buf, strlen(buf));
    sprintf(buf, "Content-length: %d\r\n\r\n", (int)strlen(body));
    Rio_writen(fd, buf, strlen(buf));
    Rio_writen(fd, body, strlen(body));


}
```

- Following screenshot represents the approach.

```
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# telnet 127.0.0.1 1331
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /config.php

HTTP/1.0 404 Not found
Content-type: text/html
Content-length: 178

<html><title>Tiny Error</title><body bgcolor=ffffff>
<h1 style="color:red;">404: Not found
<p><h1>Tiny couldn't find this file: ./config.php
<hr><em>The Tiny Web server</em>
Connection closed by foreign host.
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny#
```

- This should not be displayed i.e. filename tried to access. This would allow an attacker to brute-force various filenames and that might end up revealing the presence of confidential files.

c. Patch

- Simplest approach was undertaken to fight this vulnerability. “Cause” string was removed from the clientError() function.
- This would not allow an attacker to know the presence of specific files in the server.
- Following screenshot represents the changes in code and successful implementation of patch.

```c
void clienterror(int fd, char *cause, char *errnum,
        char *shortmsg, char *longmsg)
{
    char buf[MAXLINE], body[MAXBUF];

    /* Build the HTTP response body */
    sprintf(body, "<html><title>Tiny Error</title>");
    sprintf(body, "%s<body bgcolor=""ffffff"">\r\n", body);
    sprintf(body, "%s<h1 style=\"color:red;\">%s: %s\r\n", body, errnum, shortmsg);
    sprintf(body, "%s<p><h1>%s:\r\n", body, longmsg);
    sprintf(body, "%s<hr><em>The Tiny Web server</em>\r\n", body);

    /* Print the HTTP response */
    sprintf(buf, "HTTP/1.0 %s %s\r\n", errnum, shortmsg);
    Rio_writen(fd, buf, strlen(buf));
    sprintf(buf, "Content-type: text/html\r\n");
    Rio_writen(fd, buf, strlen(buf));
    sprintf(buf, "Content-length: %d\r\n\r\n", (int)strlen(body));
    Rio_writen(fd, buf, strlen(buf));
    Rio_writen(fd, body, strlen(body));


}
/* $end clienterror */
```

```
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# telnet 127.0.0.1 1331
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get /config.php

HTTP/1.0 404 Not found
Content-type: text/html
Content-length: 165

<html><title>Tiny Error</title><body bgcolor=ffffff>
<h1 style="color:red;">404: Not found
<p><h1>Tiny couldn't find this file:
<hr><em>The Tiny Web server</em>
Connection closed by foreign host.
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny#
```

5. Implementation of POST method

- The default method that was implemented in the tiny web server was GET.
- Strcasecmp() function was used to know whether the method buffer contained GET or any other method.
- The return value of strcasecmp() is zero, if two strings are the same, excluding the case i.e. case insensitive comparison.
- POST was added to that comparison “if” condition.
- For implementing the adder function using POST method, two environment variables were created using setenv() method, named “METHOD” and “ARG” respectively, and were accessed in the adder.c file.
- For taking the arguments as input in POST method, Rio_readlineb() method was used.
- Changes in adder.c were made in order to handle POST requests using “METHOD” environment variable.
- Argument input was stored and accessed in adder.c using “ARG” environment variable.
- After successfully making the relevant changes in the adder.c file, POST request for adding numbers was handled.
- Following screenshots represents successful implementation of POST method.

Changes in tiny.c file

```
if ((strcasecmp(method, "POST") == 0) && (strcasecmp(uri, "/cgi-bin/adder") == 0)){
    Rio_readlineb(&rio, arg, MAXLINE);                                  //For taking arguments input
    setenv("ARG",arg,1);
    //printf(getenv("ARG"));
}

setenv("METHOD_NAME", method, 1);

if (strcasecmp(method, "GET") && strcasecmp(method,"POST")) {          //line:netp:doit:beginrequesterr
   clienterror(fd, method, "501", "Not Implemented",
            "Tiny does not implement this method");
    return;
}                                                                       //line:netp:doit:endrequesterr
```

Changes in adder.c file

```c
int main(void) {
    char *buf, *p, *method_name;
    char method[MAXLINE], uri[MAXLINE], version[MAXLINE], arg[MAXLINE];
    char arg1[MAXLINE], arg2[MAXLINE], content[MAXLINE];
    int n1=0, n2=0;

    if (strcasecmp(getenv("METHOD_NAME"),"GET") == 0){
        if ((buf = getenv("QUERY_STRING")) != NULL)
        {
        p = strchr(buf, '&');
        *p = '\0';
        strcpy(arg1, buf);
        strcpy(arg2, p+1);
        n1 = atoi(arg1);
        n2 = atoi(arg2);
        }
    }
    else if (strcasecmp(getenv("METHOD_NAME"),"POST") == 0)
    {
        sscanf(getenv("ARG"), "%s", arg);
        p=strchr(arg,'&');
        *p='\0';
        strcpy(arg1,arg);
        strcpy(arg2,p+1);
        n1=atoi(arg1);
        n2=atoi(arg2);
        printf("n1=%d&n2=%d\n\n",n1,n2);

    }
```

```
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# telnet 127.0.0.1 1331
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
post /cgi-bin/adder
5&5

HTTP/1.0 200 OK
Server: Tiny Web Server
n1=5&n2=5

Content-length: 106
Content-type: text/html

Welcome to add.com: THE Internet addition portal.
<p>The answer is: 5 + 5 = 10
<p>Thanks for visiting!
Connection closed by foreign host.
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny#
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny# telnet 127.0.0.1 1331
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
post /cgi-bin/adder
2147483647&1

HTTP/1.0 200 OK
Server: Tiny Web Server
n1=2147483647&n2=1

Integer Overflow Detected!!!Connection closed by foreign host.
(base) root@Kali:~/Desktop/Secure Software Testing and Construction/tiny#
```

- Integer Overflow was also handled using POST method.

Another Approach

- More generalized approach was developed in which only some of the changes in tiny.c is required. No other changes are required.
- Instead of creating environment variables and accessing them in an adder.c file, directly serve_dynamic() was called once the arguments were taken as input.
- Following screenshot represents the changes made:

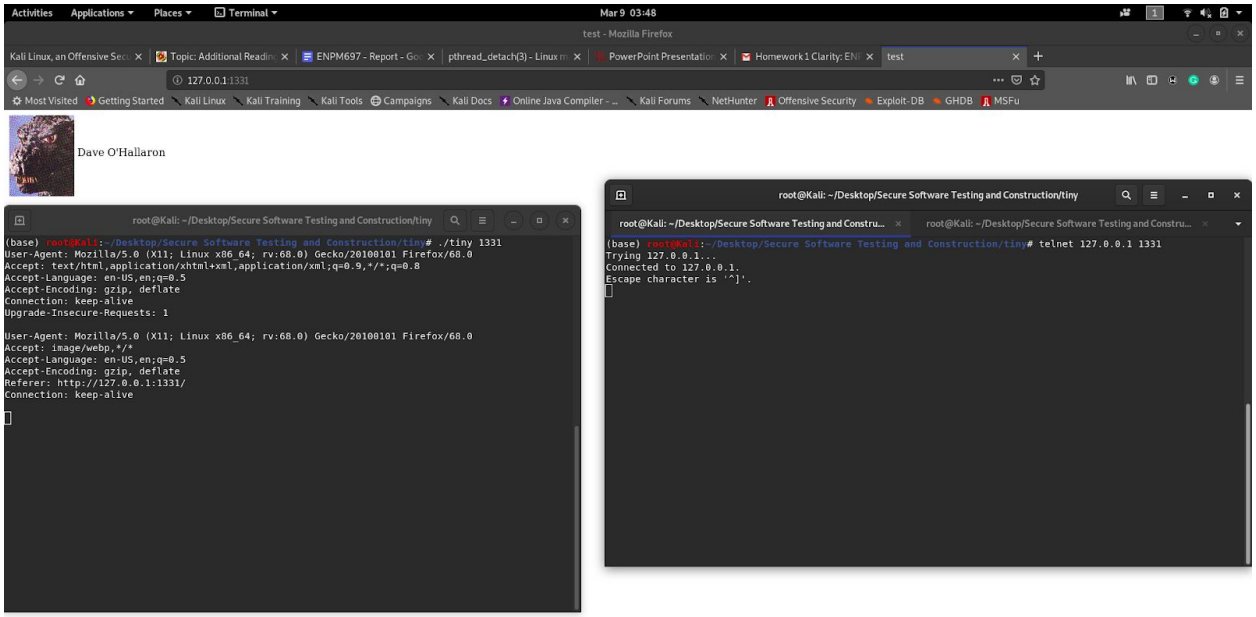
```
if ((strcasecmp(method, "POST") == 0) && (strcasecmp(uri, "/cgi-bin/adder") == 0)){
    Rio_readlineb(&rio, arg, MAXLINE);
    strcpy(filename, ".");
    strcat(filename, uri);
    serve_dynamic(fd,filename,arg);
    //setenv("ARG",arg,1);
    //printf(getenv("ARG"));
}
```

- This approach was successfully implemented and was tested for all set of inputs, covering Integer Overflow.

6. Handling Multiple Connections

- Default Tiny Web Server was configured to handle only a single client request.
- Once the telnet connection was created with the server, the user was not able to access the server from the browser.
- In order to handle multiple client requests, two methods can be used. MultiProcessing and Multithreading.
- Multithreading was incorporated in order handle multiple client requests as forking a new process in Multiprocessing is expensive. Entire state of memory, stack, file descriptors and many more are duplicated in multiprocessing.
- While, Multithreading is less expensive as it allows multitasking within the same process. Threads share the same address space (Code and data).
- Incorporation of multithreading was done by firstly importing the pthread.h header file.
- A structure called ThreadArgs was made with member client_sock.
- Dynamic memory allocation using malloc was done for threads.
- Client_sock was assigned a descriptor of the socket.
- A new thread was created using ThreadMain routine and an argument for the function using pthread_create() function.
- pthread_create() starts a new thread in the calling process. Every new thread starts the execution of the routine, which in this case is the ThreadMain function.
- For every new thread created, ThreadMain function was basically calling the doit function, which was basically communication with the server, along with detaching the thread using pthread_detach() function. When the detached thread is terminated, its resources are released back to the system. Memory stored by the thread is freed using the free() function.
- Following screenshots represents the changes in the code and successful implementation.

```c
struct ThreadArgs{
    int client_sock;
};

void *ThreadMain(void *threadargs){
    int client_sock;
    pthread_detach(pthread_self());
    client_sock = ((struct ThreadArgs *) threadargs) -> client_sock;
    doit(client_sock);
    Close(client_sock);
    free(threadargs);
    return;
}
int main(int argc, char **argv)
{
    int listenfd, connfd, port, clientlen;
    struct sockaddr_in clientaddr;
    pthread_t threadID;
    struct ThreadArgs *threadargs;

    /* Check command line args */
    if (argc != 2) {
    fprintf(stderr, "usage: %s <port>\n", argv[0]);
    exit(1);
    }
    port = atoi(argv[1]);

    listenfd = Open_listenfd(port);
    while (1) {
        clientlen = sizeof(clientaddr);
        connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen); //line:netp:tiny:accept
        threadargs = (struct ThreadArgs *) malloc(sizeof(struct ThreadArgs));
        threadargs -> client_sock = connfd;

        pthread_create(&threadID, NULL, ThreadMain, (void *) threadargs);
    }
}
```



References:

- https://www.csd.uoc.gr/~hy556/material/tutorials/cs556-3rd-tutorial.pdf