Report for login at 2024-06-25

Enkidu by Debrunbaix.

Summary

Enumeration

Binary Information

File Name	Path	Format	Bit
login	app/testFile/login	ELF	32-bit

This information comes from the **file** command.

Security of the Binary

Basic Security Features			
Linked	**Stripped**	**RELRO**	**Canary**
dynamically linked	no	partial	no

Advanced Security Mechanisms		
NX	**PIE**	**RPath**
no	no	no

Security Meta-Information		
RunPath	**Symbols**	**Fortify Source**
no	yes	no

This information comes from the **checksec** command.

Strings

- Enter admin password:
- pass
- Correct Password!
- Incorrect Password!
- Successfully logged in as Admin (authorised=%d) :)
- Failed to log in as Admin (authorised=%d) :(
- login.c
- .note.gnu.build-id

> This information comes from Binary secions and the **strings** command.

Vulnerable Functions

- gets
- printf

Libraries

- linux-gate.so.1
- libc.so.6
- /lib/ld-linux.so.2

> This information comes from the **ldd** command.

Assembly Code

```
xor ebp, ebp
pop esi
mov ecx, esp
and esp, 0xfffffff0
push eax
push esp
push edx
call 0x80490b3
add ebx, 0x2f70
lea eax, [ebx - 0x2d40]
push eax
lea eax, [ebx - 0x2da0]
push eax
push ecx
push esi
mov eax, 0x8049192
push eax
call 0x8049070
hlt
mov ebx, dword ptr [esp]
ret
nop
nop
nop
nop
nop
ret
nop
nop
nop
nop
nop
nop
nop
```

```
nop
mov ebx, dword ptr [esp]
ret
nop
nop
nop
nop
nop
nop
mov eax, 0x804c028
cmp eax, 0x804c028
je 0x8049110
mov eax, 0
test eax, eax
je 0x8049110
push ebp
mov ebp, esp
sub esp, 0x14
push 0x804c028
call eax
add esp, 0x10
leave
ret
lea esi, [esi]
nop
ret
lea esi, [esi]
lea esi, [esi]
nop
mov eax, 0x804c028
sub eax, 0x804c028
mov edx, eax
shr eax, 0x1f
sar edx, 2
add eax, edx
sar eax, 1
je 0x8049158
```

```
mov edx, 0
test edx, edx
je 0x8049158
push ebp
mov ebp, esp
sub esp, 0x10
push eax
push 0x804c028
call edx
add esp, 0x10
leave
ret
lea esi, [esi]
ret
lea esi, [esi]
cmp byte ptr [0x804c028], 0
jne 0x8049180
push ebp
mov ebp, esp
sub esp, 8
call 0x80490e0
mov byte ptr [0x804c028], 1
leave
ret
lea esi, [esi]
ret
lea esi, [esi]
lea esi, [esi]
nop
jmp 0x8049120
lea ecx, [esp + 4]
and esp, 0xfffffff0
push dword ptr [ecx - 4]
push ebp
mov ebp, esp
push ebx
push ecx
```

```
sub esp, 0x10
call 0x80490d0
add ebx, 0x2e57
mov dword ptr [ebp - 0xc], 0
sub esp, 0xc
lea eax, [ebx - 0x1ff8]
push eax
call 0x8049060
add esp, 0x10
sub esp, 0xc
lea eax, [ebp - 0x12]
push eax
call 0x8049050
add esp, 0x10
sub esp, 8
lea eax, [ebx - 0x1fe1]
push eax
lea eax, [ebp - 0x12]
push eax
call 0x8049030
add esp, 0x10
test eax, eax
jne 0x804920c
sub esp, 0xc
lea eax, [ebx - 0x1fdc]
push eax
call 0x8049060
add esp, 0x10
mov dword ptr [ebp - 0xc], 1
jmp 0x804921e
sub esp, 0xc
lea eax, [ebx - 0x1fca]
push eax
call 0x8049060
add esp, 0x10
cmp dword ptr [ebp - 0xc], 0
je 0x804923b
```

```
sub esp, 8
push dword ptr [ebp - 0xc]
lea eax, [ebx - 0x1fb4]
push eax
call 0x8049040
add esp, 0x10
jmp 0x8049250
sub esp, 8
push dword ptr [ebp - 0xc]
lea eax, [ebx - 0x1f80]
push eax
call 0x8049040
add esp, 0x10
mov eax, 0
lea esp, [ebp - 8]
pop ecx
pop ebx
pop ebp
lea esp, [ecx - 4]
ret
nop
push ebp
call 0x80492c1
add ebp, 0x2d9a
push edi
push esi
push ebx
sub esp, 0xc
mov ebx, ebp
mov edi, dword ptr [esp + 0x28]
call 0x8049000
lea ebx, [ebp - 0xf0]
lea eax, [ebp - 0xf4]
sub ebx, eax
sar ebx, 2
je 0x80492b5
xor esi, esi
```

```
lea esi, [esi]
sub esp, 4
push edi
push dword ptr [esp + 0x2c]
push dword ptr [esp + 0x2c]
call dword ptr [ebp + esi*4 - 0xf4]
add esi, 1
add esp, 0x10
cmp ebx, esi
jne 0x8049298
add esp, 0xc
pop ebx
pop esi
pop edi
pop ebp
ret
lea esi, [esi]
ret
mov ebp, dword ptr [esp]
ret
```

> This information comes from the Capstone's library and elftools command.

Code Analysis

Pseudo C Code

main.c

```
/* WARNING: Function: __x86.get_pc_thunk.bx replaced with
injection: get_pc_thunk_bx */
undefined4 main(void) {
  int passwordComparisonResult;
  char userInputPassword[6];
  int isAuthorized;
  undefined *returnAddress;

  returnAddress = &stack0x00000004;
  isAuthorized = 0;
  puts("Enter admin password: ");
  gets(userInputPassword);  // POTENTIALLY UNSAFE
FUNCTION!
  passwordComparisonResult = strcmp(userInputPassword,
"pass");
  if (passwordComparisonResult == 0) {
    puts("Correct Password!");
    isAuthorized = 1;
  } else {
    puts("Incorrect Password!");
  }
  if (isAuthorized == 0) {
    printf("Failed to log in as Admin (authorised=%d) :
(\n", 0);
  } else {
    printf("Successfully logged in as Admin
(authorised=%d) :)\n", isAuthorized);
  }
```

```
  return 0;
}
```

Description

The provided code is a simple C program that prompts the user to enter a password, checks if the entered password matches "pass", and then displays appropriate messages based on the result. The major components of the code include declaring variables for user input and authorization status, using the function gets to get user input, and comparing the entered password with the correct password using strcmp. One primary security concern in this code is the use of the gets function to capture user input. The function gets does not perform bounds checking and can lead to buffer overflow vulnerabilities, making the program susceptible to malicious attacks. A more secure alternative to gets is the function fgets, which requires the size of the buffer and thus helps in preventing buffer overflow by limiting the number of characters read. Additionally, hardcoding the password in the source code is a security risk because anyone with access to the source can easily find the password.

This information comes from Ghidra CLI and the OpenAi's API.

_start.c

```
Changed Code:

/* WARNING: Function: __i686.get_pc_thunk.bx replaced with
injection: get_pc_thunk_bx */

void processEntry _start(undefined4 unknownParam1,
undefined4 unknownParam2)
{
  undefined placeholderStack[4];

  __libc_start_main(main, unknownParam2,
&placeholderStack, __libc_csu_init, __libc_csu_fini,
unknownParam1, placeholderStack);
```

```
  do {
                    /* WARNING: Do nothing block with
infinite loop */
  } while (true);
}
```

Description

Explanation: The provided C pseudo code initializes a program entry point with the function "processEntry _start". The function takes two parameters of type undefined4, which have been renamed to "unknownParam1" and "unknownParam2" to better reflect their purpose as they are not utilized directly within the code. The variable "auStack_4" has been renamed to "placeholderStack" to provide a clearer indication of its role.

In this function, the "__libc_start_main" function is called with several parameters: the "main" function, "unknownParam2", and a pointer to the "placeholderStack". It then includes initializers and finalizers of the C standard library as well as "unknownParam1" and the "placeholderStack" itself.

There is a potential problem related to security in this code due to the lack of proper data types and the use of the "undefined" keyword. Notably, the "__libc_start_main" function is generally a part of the C runtime and is used to set up and start the main program execution. If the arguments passed to it are not validated properly, it could lead to undefined behavior, security vulnerabilities, or stack corruption. Additionally, the infinite loop at the end with the "do nothing" block could potentially cause a Denial of Service (DoS) condition if not handled properly.

This information comes from Ghidra CLI and the OpenAi's API.

_init.c

```
Revised Pseudocode:

/* WARNING: Function: __x86.get_pc_thunk.bx replaced with
injection: get_pc_thunk_bx */
```

```
int initialize(EVP_PKEY_CTX *context)

{
  undefined *functionPointer;

  functionPointer = PTR___gmon_start___0804bffc;
  if (PTR___gmon_start___0804bffc != (undefined *)0x0) {
    functionPointer = (undefined *)(*(code
*)PTR___gmon_start___0804bffc)();
  }
  return (int)functionPointer;
}
```

Description

Explanation: The function named **_init** has been renamed to **initialize** to reflect its purpose better, which is to initialize some context or pointer. The variable **puVar1** has been renamed to **functionPointer** to indicate that it is pointing to a function.

The code initializes a pointer named **functionPointer** to the value of a global variable **PTR*gmon*start_0804bffc**. It then checks if this pointer is not null. If it is not null, it calls the function that **functionPointer** points to, and sets **functionPointer** to its return value. Finally, it returns the integer cast of **functionPointer**.

One significant security concern with this code is the lack of verification on the pointer **PTR*gmon*start_0804bffc** before it's dereferenced and called. This could potentially lead to undefined behavior or expose the program to injection attacks if this pointer is somehow manipulated. Adding proper null or validity checks before using function pointers and ensuring the integrity of function pointers is crucial for preventing such security vulnerabilities.

> This information comes from Ghidra CLI and the OpenAi's API.

Exploit

Fuzzing

Exploit success with this input :

- pass
- login.c

Buffer Overflow

To determine if a buffer overflow is possible, the process involves injecting progressively larger payloads into the target binary and observing the results. By starting with a small payload and incrementally increasing its size, the goal is to trigger a **segmentation fault**, which indicates a buffer overflow vulnerability. If such a fault occurs, the binary is deemed vulnerable, and the specific payload size at which this happens is noted. This method ensures a systematic approach to identifying potential vulnerabilities within a predefined limit.

The memory of the binary is writable with this offset : 12

Credits

The development of Enkidu utilized various tools and libraries to achieve its functionality:

file: For determining file types.

checksec: To check the security properties of binaries.

strings: For extracting printable strings from files.

ldd: To list dynamic dependencies of executables.

elftools & **capstone**: For parsing and analyzing ELF files and disassembling binaries.

Ghidra: Used for decompiling binaries into C-like pseudocode through the AnalyseHeadless script.

ChatGPT API: For enhancing code comprehension and generating explanatory paragraphs.

markdown: For converting text formatted in Markdown to HTML, facilitating report generation.

WeasyPrint: To convert HTML documents into PDF files for easy distribution and archiving of reports.