Report for login at 2024-06-18

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 comparisonResult;
  char passwordInput[6];
  int isAuthenticated;
  undefined *stackPointer;

  stackPointer = &stack0x00000004;
  isAuthenticated = 0;
  puts("Enter admin password: ");
  gets(passwordInput);
  comparisonResult = strcmp(passwordInput,"pass");
  if (comparisonResult == 0) {
    puts("Correct Password!");
    isAuthenticated = 1;
  }
  else {
    puts("Incorrect Password!");
  }
  if (isAuthenticated == 0) {
    printf("Failed to log in as Admin (authorised=%d) :
(\n",0);
  }
  else {
    printf("Successfully logged in as Admin
(authorised=%d) :)\n",isAuthenticated);
```

```
  }
  return 0;
}
```

Description

The provided code prompts the user to enter an admin password, reads the input using gets, and then compares it with the string "pass" to determine if the entered password is correct. If the password matches, it prints a success message and sets the isAuthenticated variable to 1. Otherwise, it prints a failure message. The value of isAuthenticated is then used to print whether the login was successful or not.

The primary security issue with this code is the use of the gets function to read the password input. The gets function does not perform bounds checking and is therefore susceptible to buffer overflow attacks, where an attacker could overflow the passwordInput buffer to execute arbitrary code or manipulate the program's execution. This vulnerability can be mitigated by using safer functions like fgets or scanf with a specified limit on input length. Additionally, it is a bad practice to use hardcoded passwords in the code as it can lead to security vulnerabilities if the code is exposed or reverse-engineered. To improve the security of handling passwords, consider using secure password hashing mechanisms and proper password storage strategies.

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 entryPoint _start(undefined4 mainFunc, undefined4
stackSize)
```

```
{
  undefined stackMemory [4];

  __libc_start_main(main, stackSize, &stack0x00000004,
__libc_csu_init, __libc_csu_fini, mainFunc, stackMemory);
  do {
                    /* WARNING: Do nothing block with
infinite loop */
  } while(true);
}
```

Description

The code initializes a program using the **__libc_start_main** function, which is common in Linux systems to set up the main execution environment and prepare to call the main function. The function **entryPoint** (originally **processEntry**) takes two parameters which were renamed for clarity: **mainFunc** and **stackSize**. The local variable **auStack_4** was renamed to **stackMemory** to reflect more clearly that it represents some allocated stack space. After initialization, the code enters an infinite loop that does nothing.

There are several security considerations with this code, primarily stemming from the use of **undefined** types and the lack of proper error checking which can lead to vulnerabilities such as buffer overflows or stack corruption. Additionally, the infinite loop could be a potential point of Denial of Service (DoS) attack if not properly managed. Functions like **__libc_start_main** are generally safe if used correctly, but misuse or erroneous parameters can compromise the programâ€™s security, making robust and secure handling crucial.

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

_init.c

```
/* 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

The provided C pseudo code has a function named _init which has been renamed to initialize for better readability. The parameter ctx has been renamed to context for clarity. The variable puVar1 has been renamed functionPointer to give more context as to its role in pointing to a function.

This code includes a check on a pointer PTR***gmon*start**_0804bffc, and if it is not null, it attempts to call the function at that location. One important security consideration here is the dereferencing and indirect call of a function pointer. It is critical to ensure that this pointer points to a legitimate and safe function to avoid arbitrary code execution or corruption. Furthermore, the use of undefined pointers (type undefined *) can be unsafe because it relies on implicit casting, which can lead to undefined behavior if not handled carefully. Proper type definitions and checks should be enforced to mitigate risks. Ensuring that pointers are valid and operations on them are secure is paramount to avoid vulnerabilities like buffer overflows, code injection, and other security issues.

> 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.