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 isAdmin;
  undefined *stackBase;

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

```
  }
  return 0;
}
```

Description

The revised code has variable names updated for clarity. **iVar1** has been renamed to **comparisonResult** to reflect its role in storing the result of **strcmp**. **local_1a** is now **passwordInput**, indicating it stores the user-input password. **local_14** is renamed to **isAdmin**, reflecting its role as a flag to check admin authorization. **local_10** is **stackBase**, denoting its relation to the stack base. The code asks for an admin password, compares the input to a hardcoded string "pass," and prints a message based on whether the password matched.

One significant security issue in this code is the use of **gets()**, which is unsafe because it doesn't perform bounds checking and can lead to buffer overflow attacks, where input data overwrites adjacent memory. A more secure alternative would be **fgets()** which allows specifying the buffer size, reducing the risk of overflow. Another potential issue is hardcoding the password in the source code, which could be easily found and exploited by an attacker with access to the binary. Using a more secure method of password storage and comparison would mitigate this risk.

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

_start.c

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

void processEntry _start(undefined4 arg_1, undefined4
arg_2)

{
  undefined stackBuffer[4];

  __libc_start_main(main, arg_2, &stackBuffer[0],
```

```
__libc_csu_init, __libc_csu_fini, arg_1, stackBuffer);
  do {
                    /* WARNING: Do nothing block with
infinite loop */
  } while (true);
}
```

Description

The provided pseudo code corresponds to a typical entry point of a C program that uses the **__libc_start_main** function to initialize the program. The variable names have been changed to **arg_1** and **arg_2** for better readability as they represent arguments. The array **auStack_4** has been renamed to **stackBuffer** to indicate its purpose as a temporary storage stack buffer.

This function uses **__libc_start_main** to start the program by calling **main** with its parameters along with initialization and finalization functions. However, the program contains an infinite loop that will keep it running indefinitely, unless deliberately terminated. The inclusion of a do-nothing infinite loop indicates either incomplete code or some form of vulnerability.

A significant security concern in this code includes the use of the function **__libc_start_main** without proper validation and error handling that may lead to undefined behavior if the arguments passed are not as expected. Another issue is that the definition of **undefined** types is not clear, leading to potential mismanagement of resources. The overuse of such generic types can introduce security risks such as buffer overflows if mishandled, potentially exposing the application to exploitation.

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

_init.c

```
Revised Code:

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

```
int initialize(EVP_PKEY_CTX *context)
{
  undefined *funcPtr;

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

Description

Explanation: The provided code defines a function named **_init** which has been renamed to **initialize** for clarity, indicating that it is likely related to an initialization process. The parameter **ctx** has been renamed to **context** to make it clear that it is referencing a context object. Within the function, the variable **puVar1** has been changed to **funcPtr** to suggest that it holds a function pointer. The code checks if **PTR*gmon*start_0804bffc** is not a null pointer and if true, invokes what appears to be the function pointed to by **PTR*gmon*start_0804bffc**.

One primary concern with this code is the use of an undefined type **undefined** and pointer manipulation which depends heavily on specific addresses. This can lead to undefined behavior and potential security vulnerabilities such as null pointer dereference or execution of unintended code. Additionally, the function pointer call **((*code*)PTR*gmon*start_0804bffc)()** can be problematic if the address is corrupted or tampered with, potentially leading to a function pointer hijacking attack. It is important to ensure that such pointers are validated and handled securely to prevent exploitation.

> 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.