Report for crackme at 2024-06-20

Enkidu by Debrunbaix.

Summary

Enumeration

Binary Information

File Name	Path	Format	Bit
crackme	crackme	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	yes

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

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

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

Strings

- (*) -Syntaxe: %s [password]
- rification de votre mot de passe..
- (!) L'authentification a
- '+) Authentification r
- sh 3.0 # password: %s
- le voie de la sagesse te guidera, tache de trouver le mot de passe petit padawaan
- .debug_pubnames
- dtor_idx.6627

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

Vulnerable Functions

- strcpy
- 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
push 0x8048830
push 0x8048840
push ecx
push esi
push 0x8048554
call 0x804840c
hlt
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
push ebp
mov ebp, esp
push ebx
sub esp, 4
cmp byte ptr [0x804a030], 0
jne 0x8048520
```

```
mov edx, dword ptr [0x804a034]
mov eax, 0x8049f18
sub eax, 0x8049f14
sar eax, 2
lea ebx, [eax - 1]
cmp edx, ebx
jae 0x8048519
lea esi, [esi]
lea eax, [edx + 1]
mov dword ptr [0x804a034], eax
call dword ptr [eax*4 + 0x8049f14]
mov edx, dword ptr [0x804a034]
cmp edx, ebx
jb 0x8048500
mov byte ptr [0x804a030], 1
add esp, 4
pop ebx
pop ebp
ret
lea esi, [esi]
lea edi, [edi]
push ebp
mov ebp, esp
sub esp, 8
mov eax, dword ptr [0x8049f1c]
test eax, eax
je 0x8048551
mov eax, 0
test eax, eax
je 0x8048551
mov dword ptr [esp], 0x8049f1c
call eax
leave
ret
nop
lea ecx, [esp + 4]
and esp, 0xfffffff0
```

```
push dword ptr [ecx - 4]
push ebp
mov ebp, esp
push edi
push ecx
sub esp, 0xb0
mov eax, dword ptr [ecx + 4]
mov dword ptr [ebp - 0x9c], eax
mov eax, dword ptr gs:[0x14]
mov dword ptr [ebp - 0xc], eax
xor eax, eax
cmp dword ptr [ecx], 2
je 0x80485ae
xor eax, eax
xor ebx, ebx
xor ecx, ecx
xor edx, edx
mov eax, dword ptr [ebp - 0x9c]
mov eax, dword ptr [eax]
mov dword ptr [esp + 4], eax
mov dword ptr [esp], 0x80488f0
call 0x804843c
mov dword ptr [esp], 0
call 0x804848c
mov dword ptr [esp], 0x1d
call 0x804844c
mov dword ptr [ebp - 0x94], eax
mov dword ptr [esp + 8], 0x1f
mov dword ptr [esp + 4], 0x8048910
mov eax, dword ptr [ebp - 0x94]
mov dword ptr [esp], eax
call 0x804841c
lea edx, [ebp - 0x8e]
mov dword ptr [ebp - 0xa4], edx
mov dword ptr [ebp - 0xa8], 0
mov eax, 0x64
cmp eax, 4
```

```
jb 0x804861c
mov dword ptr [ebp - 0xac], 0x19
mov edi, dword ptr [ebp - 0xa4]
mov ecx, dword ptr [ebp - 0xac]
mov eax, dword ptr [ebp - 0xa8]
rep stosd dword ptr es:[edi], eax
mov dword ptr [esp + 8], 0xd
mov dword ptr [esp + 4], 0x804892f
lea eax, [ebp - 0x8e]
mov dword ptr [esp], eax
call 0x804841c
mov eax, dword ptr [ebp - 0x94]
add eax, 5
mov byte ptr [eax], 0x63
mov eax, dword ptr [ebp - 0x94]
add eax, 0x16
mov byte ptr [eax], 0
mov dword ptr [0x804a038], 0x80486c4
mov eax, dword ptr [ebp - 0x9c]
add eax, 4
mov eax, dword ptr [eax]
mov dword ptr [esp + 4], eax
lea eax, [ebp - 0x2a]
mov dword ptr [esp], eax
call 0x804842c
mov eax, dword ptr [ebp - 0x94]
add eax, 8
mov byte ptr [eax], 0x5f
mov eax, dword ptr [ebp - 0x94]
add eax, 9
mov byte ptr [eax], 0x2e
mov edx, dword ptr [0x804a038]
mov eax, dword ptr [ebp - 0x94]
mov dword ptr [esp + 4], eax
lea eax, [ebp - 0x2a]
mov dword ptr [esp], eax
call edx
```

```
mov edx, dword ptr [ebp - 0xc]
xor edx, dword ptr gs:[0x14]
je 0x80486b7
call 0x804845c
add esp, 0xb0
pop ecx
pop edi
pop ebp
lea esp, [ecx - 4]
ret
push ebp
mov ebp, esp
sub esp, 8
mov eax, dword ptr [ebp + 0xc]
add eax, 0xb
mov byte ptr [eax], 0xd
mov eax, dword ptr [ebp + 0xc]
add eax, 0xc
mov byte ptr [eax], 0xa
mov dword ptr [esp], 0x804893c
call 0x804846c
mov eax, dword ptr [ebp + 0xc]
mov dword ptr [esp + 4], eax
mov eax, dword ptr [ebp + 8]
mov dword ptr [esp], eax
call 0x804847c
test eax, eax
jne 0x804870f
call 0x804872c
mov dword ptr [esp], 0
call 0x804848c
call 0x8048803
mov dword ptr [esp], 0x8048964
call 0x804846c
mov dword ptr [esp], 1
call 0x804848c
push ebp
```

```
mov ebp, esp
sub esp, 0x58
mov eax, dword ptr gs:[0x14]
mov dword ptr [ebp - 4], eax
xor eax, eax
mov dword ptr [ebp - 0x40], 0x6562696c
mov dword ptr [ebp - 0x3c], 0xa9c37472
mov dword ptr [ebp - 0x38], 0x21
lea edx, [ebp - 0x34]
mov dword ptr [ebp - 0x44], edx
mov dword ptr [ebp - 0x48], 0
mov eax, 0x17
cmp eax, 4
jb 0x804879b
mov eax, 0x17
mov ecx, eax
and ecx, 0xfffffffc
mov dword ptr [ebp - 0x50], ecx
mov dword ptr [ebp - 0x4c], 0
mov eax, dword ptr [ebp - 0x48]
mov ecx, dword ptr [ebp - 0x44]
mov edx, dword ptr [ebp - 0x4c]
mov dword ptr [ecx + edx], eax
add dword ptr [ebp - 0x4c], 4
mov eax, dword ptr [ebp - 0x50]
cmp dword ptr [ebp - 0x4c], eax
jb 0x804877d
mov eax, dword ptr [ebp - 0x4c]
add dword ptr [ebp - 0x44], eax
movzx ecx, word ptr [ebp - 0x48]
mov edx, dword ptr [ebp - 0x44]
mov word ptr [edx], cx
add dword ptr [ebp - 0x44], 2
movzx eax, byte ptr [ebp - 0x48]
mov edx, dword ptr [ebp - 0x44]
mov byte ptr [edx], al
add dword ptr [ebp - 0x44], 1
```

```
mov dword ptr [ebp - 0x1d], 0x4763305f
mov dword ptr [ebp - 0x19], 0x6d35636a
mov dword ptr [ebp - 0x15], 0x54352e5f
mov dword ptr [ebp - 0x11], 0x3887c333
mov dword ptr [ebp - 0xd], 0xc3304a43
mov dword ptr [ebp - 9], 0x39483980
mov byte ptr [ebp - 5], 0
lea eax, [ebp - 0x40]
mov dword ptr [esp + 4], eax
mov dword ptr [esp], 0x8048998
call 0x804843c
mov dword ptr [esp], 0
call 0x804848c
push ebp
mov ebp, esp
sub esp, 8
mov dword ptr [esp], 0x80489e0
call 0x804843c
mov dword ptr [esp], 0
call 0x804848c
push ebp
mov ebp, esp
pop ebp
ret
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
push ebp
mov ebp, esp
pop ebp
```

```
ret
lea esi, [esi]
lea edi, [edi]
push ebp
mov ebp, esp
push edi
push esi
push ebx
call 0x804889a
add ebx, 0x17a9
sub esp, 0xc
call 0x80483bc
lea edi, [ebx - 0xe8]
lea eax, [ebx - 0xe8]
sub edi, eax
sar edi, 2
test edi, edi
je 0x8048892
xor esi, esi
mov eax, dword ptr [ebp + 0x10]
mov dword ptr [esp + 8], eax
mov eax, dword ptr [ebp + 0xc]
mov dword ptr [esp + 4], eax
mov eax, dword ptr [ebp + 8]
mov dword ptr [esp], eax
call dword ptr [ebx + esi*4 - 0xe8]
add esi, 1
cmp esi, edi
jb 0x8048870
add esp, 0xc
pop ebx
pop esi
pop edi
pop ebp
ret
mov ebx, dword ptr [esp]
ret
```

```
nop
nop
push ebp
mov ebp, esp
push ebx
sub esp, 4
mov eax, dword ptr [0x8049f0c]
cmp eax, -1
je 0x80488c4
mov ebx, 0x8049f0c
nop
sub ebx, 4
call eax
mov eax, dword ptr [ebx]
cmp eax, -1
jne 0x80488b8
add esp, 4
pop ebx
pop ebp
ret
nop
nop
```

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

Code Analysis

Pseudo C Code

RS4.c

```
void exitProgram(void)
{
  printf("The path of wisdom will guide you, try to find
the password, little padawan \n ");
  exit(0); // WARNING: This subroutine does not return
}
```

Description

The provided code consists of a function that prints a message to the console and then exits the program. The original function name "RS4" has been renamed to "exitProgram" to clearly indicate its functionality, which is to print a message and then terminate the program. The string message has also been translated to English for better clarity. One potential security issue with this function is the lack of proper input validation and handling, though in this specific function, there are no user inputs. Moreover, the use of printf inherently does not have immediate security risks in this context; however, if it were to use string formatting with user inputs, it could lead to format string vulnerabilities. The function also uses exit(0) to terminate the program, which is a standard safe operation but should be used with caution to ensure that it does not lead to an unexpected termination of the program in a real-world application.

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

_start.c

```
void processEntry _start(undefined4 entryPoint, undefined4
stackAddress) {
  undefined localStack[4];
  __libc_start_main(main, stackAddress, &stack0x00000004,
__libc_csu_init, __libc_csu_fini, entryPoint, localStack);

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

Description

The code provided is the entry point of a program, typically seen in system-level or embedded code where low-level access and setup are required. It calls the **__libc_start_main** function, which is responsible for setting up the environment and starting the **main** function of the program. The processEntry _start function takes two parameters: entryPoint and stackAddress, which are placeholders for values passed by the operating system at the program's start.

Security concerns arise with the usage of the stack in this code. The localStack array is defined with an undefined type and lacks proper initialization, posing potential risks of undefined behavior and security vulnerabilities. Further, C functions like __libc_start_main do not inherently perform bounds-checking, and an ill-defined stack setup could lead to buffer overflows, memory corruption, and other unpredictable behaviors. These issues highlight the need for careful management of stack variables and rigorous bounds-checking practices to minimize risks and ensure secure execution.

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

_init.c

```
int initialize(EVP_PKEY_CTX *context)
{
  int returnValue;
  if (PTR___gmon_start___08049ff0 != (undefined *)0x0) {
    __gmon_start__();
  }
  setup_frame_dummy();
  invoke_global_constructors();
  return returnValue;
}
```

Description

The provided pseudo code represents a typical initialization routine in C. The **_init** function takes a pointer to **EVP_PKEY_CTX** named **context** as an argument. The **iStack_c** variable has been renamed to **returnValue** for clarity. The function first checks if the pointer **PTR*gmon*start_08049ff0** is not null, and if so, it calls the function **gmon_start()**. It then calls two other functions: **frame_dummy()** and **__do_global_ctors_aux()**, which have been renamed to **setup_frame_dummy()** and **invoke_global_constructors()** respectively. These functions are typically the result of compiler-generated code for setting up the runtime environment.

One potential security issue in this code is that the value of **returnValue** (previously **iStack_c**) is used without initialization, which could lead to undefined behavior or potential security vulnerabilities if the function relies on this return value. Furthermore, any reliance on the function **gmon_start**, which is typically used for debugging purposes, might expose the code to potential security risks if not properly controlled. Additionally, invoking global constructors in **invoke_global_constructors** could lead to unexpected side-effects if these constructors perform unsafe operations. It is crucial to review the security aspects of all invoked functions to ensure they do not introduce any vulnerabilities.

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

WPA.c

```
void checkPassword(char *userInput, char *correctPassword)
{
  int comparisonResult;

  correctPassword[11] = '\r';
  correctPassword[12] = '\n';
  puts(&DAT_0804893c);
  comparisonResult = strcmp(userInput, correctPassword);
  if (comparisonResult == 0) {
    executeBlowfishEncryption();
                    /* WARNING: Subroutine does not return
*/
    exit(0);
  }
  executeRS4Encryption();
  puts(&DAT_08048964);
                    /* WARNING: Subroutine does not return
*/
  exit(1);
}
```

Description

The code above defines a function checkPassword which takes two character pointers, userInput and correctPassword, as parameters. It modifies correctPassword by inserting carriage return and newline characters at specific positions. It then prints a message from a memory address using the puts function. It compares the userInput with correctPassword using strcmp. If they match, it calls the function executeBlowfishEncryption and exits the program with a status of 0. Otherwise, it calls the function executeRS4Encryption, prints another message, and exits the program with status 1. Potential security issues include the use of hard-coded indices for modifying the correctPassword, which could result in buffer overruns if the length is not properly managed. Additionally, the use of functions that do not return (marked by the warnings in the comments) without proper cleanup could cause resource leaks and undefined behavior in a larger application. The comparison operator used for matching passwords should also be checked for timing attacks.

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

blowfish.c

```
void blowfish(void) {
  undefined4 savedRegister;
  uint index;
  int stackOffset;
  uint loopCounter;
  undefined4 var1;
  undefined4 var2;
  undefined4 var3;
  undefined4 var4;
  undefined2 shortVar;
  undefined buffer[17];
  undefined4 keyPart1;
  undefined4 keyPart2;
  undefined4 keyPart3;
  undefined4 keyPart4;
  undefined4 keyPart5;
  undefined4 keyPart6;
  undefined singleByteTerminator;
  undefined4 localVar;

  savedRegister = *(undefined4 *)(stackOffset + 0x14);
  var1 = 0x6562696c;
  var2 = 0xa9c37472;
  var3 = 0x21;
  loopCounter = 0;
  index = loopCounter;
  do {
    loopCounter = index;
    *(undefined4 *)((int)&var4 + loopCounter) = 0;
    index = loopCounter + 4;
  } while (loopCounter + 4 < 0x14);
  localVar = savedRegister;
  *(undefined2 *)(buffer + (loopCounter - 2)) = 0;
  buffer[loopCounter] = 0;
  keyPart1 = 0x4763305f;
  keyPart2 = 0x6d35636a;
  keyPart3 = 0x54352e5f;
  keyPart4 = 0x3887c333;
  keyPart5 = 0xc3304a43;
  keyPart6 = 0x39483980;
  singleByteTerminator = 0;
  printf(&DAT_08048998,&var1);
  exit(0);
}
```

Description

The provided code has been cleaned up and variable names changed to enhance readability and logical understanding. The function **blowfish** sets up a series of initialization operations, manipulates several buffers, and eventually prints a formatted string before terminating the program. The pattern of variable names such as **var1** to **var6** and **buffer** are more indicative of their roles within this context.

The critical security issues to notice involve the use of deprecated or non-portable constructs such as **undefined4** and handling of raw memory operations. The **printf** function is directly provided an untrusted format string which is a severe vulnerability (format string vulnerability). This can be potentially leveraged for exploits such as information leak or arbitrary code execution. The use of **gets** or other unsafe functions can lead to buffer overflows if the input size is not controlled. Always prefer secure functions (e.g., using snprintf) and provide secure initialization and termination processes that don't rely on magic values or stack manipulation for correctness.

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

Exploit

Fuzzing

Exploit success with this input :

- (*) -Syntaxe: %s [password]
- rification de votre mot de passe..
- (!) L'authentification a
- '+) Authentification r
- sh 3.0 # password: %s
- le voie de la sagesse te guidera, tache de trouver le mot de passe petit padawaan
- .debug_pubnames
- dtor_idx.6627

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.

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.