Bin1:

1. For each custom function in the binary, provide a minimum of 3 sentence describing what the function does. Your description should include any functions called, variables created or modified, and the purpose of the function. (3 pts)

|  |  |
| --- | --- |
| ***Function name*** | ***Description*** |
| Main | Main has a nested if/else structure, that checks two conditions before passing the command line argument to myCopy(). The first condition confirms that argc==2, meaning only one command line argument was submitted – failing this check prints “need password” to the screen. The nested if/else checks if the length of the provided argument is shorter than 0x29 (less than 41 decimal), printing a warning that the length must be less than 20 characters. |
| coolFunction | This function is not reachable from main. All it does is print “this shouldn’t execute” and then returns. |
| myCopy | This function receives a char\* to argv[1] (which can be at most 0x28 (40) characters), and then copies up to 0x24 (36) bytes to a local array with allocated length of 0x28 (40) bytes. It returns upon completion. |
|  |  |
|  |  |

1. Determine if the binary contains a vulnerability. If one exists, document it through screenshots and describe it in a minimum of 3 sentences. (5 pts)

. this binary does not contain a vulnerability, to my knowledge.

1. If a vulnerability exists, determine if it’s exploitable or not. In this context exploitable means that a malicious user can:
   * Reach the vulnerability. .
   * Take control of EIP. .

Write a minimum of 3 sentences explaining how a malicious user could (or could not) reach the vulnerability and if EIP could (or could not) be taken over. (5 pts)

.

1. If the vulnerability is exploitable, use “***python -c ‘print(…)’***” to write a simple Proof of Concept (POC) exploit to execute a function contained within the binary. Provide a screenshot documenting your POC and the result of executing it. If it is not exploitable, describe what, if any, negative side effects the vulnerability has. If relevant provide a screenshot document the negative side effect. (7 pts)

.

Bin2:

1. For each custom function in the binary, provide a minimum of 3 sentence describing what the function does. Your description should include any functions called, variables created or modified, and the purpose of the function. (3 pts)

|  |  |
| --- | --- |
| ***Function name*** | ***Description*** |
| main | This program asks for user input (an integer length), then calls generatePassword to generate a random string of that length, and then main prints that string to the screen before returning. Main in particular initializes an array of length 10, and passes that along with the password length into the generatePassword function. |
| coolFunction | This function is not reachable from main. All it does is print “this shouldn’t execute” and then returns. |
| generatePassword | this function takes two parameters, a char\* to the locally defined array in main (length 10), and an integer for the length of the password to generate. It loops over the array pointer given, up to the length provided. There are no checks against the given length, and we can overflow the buffer. If it is long enough we can overwrite the EIP value on the stack that is used when main returns. The values are random however, so at best we can just break the program after intended execution – and we exert any intentional will over the program due to the randomness. |
|  |  |
|  |  |

1. Determine if the binary contains a vulnerability. If one exists, document it through screenshots and describe it in a minimum of 3 sentences. (5 pts)

. There are no checks against the given length, and we can overflow the buffer. If it is long enough we can overwrite the EIP value on the stack that is used when main returns. The values are random however, so at best we can just break the program after intended execution – and we exert any intentional will over the program due to the randomness.

1. If a vulnerability exists, determine if it’s exploitable or not. In this context exploitable means that a malicious user can:
   * Reach the vulnerability. .yes
   * Take control of EIP. . ehhh kind of? We can mess with it, but we can’t take control. behavior depends on the random values generated

Write a minimum of 3 sentences explaining how a malicious user could (or could not) reach the vulnerability and if EIP could (or could not) be taken over. (5 pts)

. follow normal execution, when asked for a value to enter, enter 42. The array has defined 10 bytes on the stack, it is preceeded on the stack by a 4 byte integer (iterator) and a 4 byte char\* (8 bytes between them). Then Ebp is 4 bytes on the stack. Esp zeroed it’s least significant bit before getting pushed to ebp (0 to 16 bytes added to the buffer we’ll go with 16 just to be safe), and lastly EIP from when main was called, is another 4 bytes. 10+8+4+16+4 = 42 bytes or 42 characters.

1. If the vulnerability is exploitable, use “***python -c ‘print(…)’***” to write a simple Proof of Concept (POC) exploit to execute a function contained within the binary. Provide a screenshot documenting your POC and the result of executing it. If it is not exploitable, describe what, if any, negative side effects the vulnerability has. If relevant provide a screenshot document the negative side effect. (7 pts)

.

Bin3:

1. For each custom function in the binary, provide a minimum of 3 sentence describing what the function does. Your description should include any functions called, variables created or modified, and the purpose of the function. (3 pts)

|  |  |
| --- | --- |
| ***Function name*** | ***Description*** |
| Main | This program uses generatePassword() to generate a 10 character random string or password, then asks the user for a 10 character string or password. It calls verifyPassword() to compare the two strings, and if they match coolFunction() is called. Because the values of the generated password are random and seeded by the time library, it’s infeasible to actually get the coolfunction to pop as intended. If we overflow the unchecked buffer, however, we can overwrite the eip that main will use in it’s return call after execution, and drop the address of coolFunction() into that stack space. |
| coolFunction | This function is technically reachable from main through a conditional check on the flag variable set by verifyPassword. But let’s be real. what are the odds. All it does is print “Correct Password Congrats. |
| generatePassword | this function takes two parameters, a char\* to the locally defined array in main (length 10), and an integer for the length of the password to generate. It loops over the array pointer given, up to the length provided. The program does not ask for user input, it just feeds this function a value of 10. |
| verifyPassword |  |
|  |  |

1. Determine if the binary contains a vulnerability. If one exists, document it through screenshots and describe it in a minimum of 3 sentences. (5 pts)

.

1. If a vulnerability exists, determine if it’s exploitable or not. In this context exploitable means that a malicious user can:
   * Reach the vulnerability. . yes
   * Take control of EIP. . yes

Write a minimum of 3 sentences explaining how a malicious user could (or could not) reach the vulnerability and if EIP could (or could not) be taken over. (5 pts)

. the overflow is from user provided data. If we want to influence control flow and get coolFunction to pop (when it clearly isn’t intended to) we can stop after overflowing the generatedArray buffer (20 bytes total). If we want control of eip, we just keep going. It should be 44 characters/bytes, and then the 4 byte address that we want to jump to.

1. If the vulnerability is exploitable, use “***python -c ‘print(…)’***” to write a simple Proof of Concept (POC) exploit to execute a function contained within the binary. Provide a screenshot documenting your POC and the result of executing it. If it is not exploitable, describe what, if any, negative side effects the vulnerability has. If relevant provide a screenshot document the negative side effect. (7 pts)

$(python -c “print(‘a’\*44+’<reverse sequence of the hex address of coolFunction>’)”)

Bin4:

1. For each custom function in the binary, provide a minimum of 3 sentence describing what the function does. Your description should include any functions called, variables created or modified, and the purpose of the function. (3 pts)

|  |  |
| --- | --- |
| ***Function name*** | ***Description*** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. Determine if the binary contains a vulnerability. If one exists, document it through screenshots and describe it in a minimum of 3 sentences. (5 pts)

.

1. If a vulnerability exists, determine if it’s exploitable or not. In this context exploitable means that a malicious user can:
   * Reach the vulnerability. .
   * Take control of EIP. .

Write a minimum of 3 sentences explaining how a malicious user could (or could not) reach the vulnerability and if EIP could (or could not) be taken over. (5 pts)

.

1. If the vulnerability is exploitable, use “***python -c ‘print(…)’***” to write a simple Proof of Concept (POC) exploit to execute a function contained within the binary. Provide a screenshot documenting your POC and the result of executing it. If it is not exploitable, describe what, if any, negative side effects the vulnerability has. If relevant provide a screenshot document the negative side effect. (7 pts)

.