Security in Software Applications 2024 - Project 1

Tommaso De Nicola 2006686
Professor: Daniele Friolo

Flawfinder is a static source code analyzer that scans for vulnerable and dangerous functions and patterns, it's lightweight and easy to use and provides a ranked list of potential issues, helping prioritize higher-risk vulnerabilities; on the other hand, it cannot scan in a dynamic environment or for more complex contexts, generating a lot of false positives. Here an example of the output for the provided sample:

```
project1_SSA24.c:42:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy
  easily misused).
project1_SSA24.c:56:  [4] (format) fprintf:
  If format strings can be influenced by an attacker, they can be exploited
  (CWE-134). Use a constant for the format specification.
project1_SSA24.c:8:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
project1_SSA24.c:28:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
project1_SSA24.c:33:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
project1_SSA24.c:35:  [2] (buffer) strcat:
  Does not check for buffer overflows when concatenating to destination
  [MS-banned] (CWE-120). Consider using strcat_s, strncat, strlcat, or
  snprintf (warning: strncat is easily misused). Risk is low because the
  source is a constant string.
project1_SSA24.c:8:  [1] (buffer) strlen:
  Does not handle strings that are not \0-terminated; if given one it may
  perform an over-read (it could cause a crash if unprotected) (CWE-126).
project1_SSA24.c:9:  [1] (buffer) strncpy:
  Easily used incorrectly; doesn't always \0-terminate or check for invalid
  pointers [MS-banned] (CWE-120).
project1_SSA24.c:9:  [1] (buffer) strlen:
  Does not handle strings that are not \0-terminated; if given one it may
  perform an over-read (it could cause a crash if unprotected) (CWE-126).
project1_SSA24.c:10:  [1] (buffer) strlen:
  Does not handle strings that are not \0-terminated; if given one it may
  perform an over-read (it could cause a crash if unprotected) (CWE-126).
project1_SSA24.c:17:  [1] (buffer) read:
  Check buffer boundaries if used in a loop including recursive loops
  (CWE-120, CWE-20).
project1_SSA24.c:22:  [1] (buffer) read:
  Check buffer boundaries if used in a loop including recursive loops
  (CWE-120, CWE-20).
project1_SSA24.c:34:  [1] (buffer) strncpy:
  Easily used incorrectly; doesn't always \0-terminate or check for invalid
  pointers [MS-banned] (CWE-120).
```

Flaws Analysis

1. **project1_SSA24.c:42: [4] (buffer) strcpy**
   True Positive: the strcpy is done on a buffer of size 10 without checking the bounds of the source string.

2. **project1_SSA24.c:56: [4] (format) fprintf**
   True Positive: the error message can be manipulated by an attacker by inserting a non alphanumeric character in the first byte of input of the user id.

3. **project1_SSA24.c:8: [2] (buffer) char**
   True Positive: if the source string is too long the allocated buffer on stack can overflow the stack.

4. **project1_SSA24.c:28: [2] (buffer) char**
   False Positive: the buffer for its entire lifetime is used properly with the right sizes in the operations.

5. **project1_SSA24.c:33: [2] (buffer) char**
   False Positive: (the buffer is used properly)

6. **project1_SSA24.c:35: [2] (buffer) strcat**
   False Positive: the strcat cannot overflow given the maximum size of the first and second source strings.

7. **project1_SSA24.c:8: [1] (buffer) strlen**
   True Positive: as vulnerability 3, it can trigger a stack overflow and with a non-nullterminated string can also hit some non readable section/data.

8. **project1_SSA24.c:9: [1] (buffer) strncpy**
   False Positive: the strncpy is safely used with the right size of the copy.

9. **project1_SSA24.c:9: [1] (buffer) strlen**
   True Positive: as vuln 7, it can hit some non readable section/data.

10. **project1_SSA24.c:10: [1] (buffer) strlen**
    True Positive: as the previous vuln, it can hit some non readable section/data.

11. **project1_SSA24.c:17: [1] (buffer) read**
    False Positive: this read is safe even if it gets random bytes in input.

12. **project1_SSA24.c:22: [1] (buffer) read**
    True Positive: this read can leak some data if provided with a small number of characters, given that the nullbyte is not inserted at the end of the input, but at the end of the buffer.

13. **project1_SSA24.c:34: [1] (buffer) strncpy**
    False Positive: as vuln 6, this strncpy is safe, due to its size, input and output buffers.

Bug Fixes

1. **line 2/3/4** missing the "#" for the "include" preprocessors.
2. **line 17/22** missing include of "unistd.h" to use the "read" function.
3. **line 31** missing include of "ctype.h" to use the "isalpha" function.
4. **line 47** missing return type for "main" function.
5. **line 54** extra "{".
6. **line 54** the "try/catch" block does not exists in standard C.
7. **line 57** useless ";" (not really a bug).
8. **line 58** the file "aFile" is used but not declared.
9. **line 58** missing ";".
10. **line 61** the array "a" is accessed with the int "y", but "y" at line 49 is defined as 10, and in line 50, "a" is defined with size 10, so it will write out of bounds of the array "a".

Conclusion

After the correction of the reported bugs the code can now compile, but is preferable to even fix all the previously listed vulnerabilities. Also considering the capabilities of Flawfinder, is highly recommended to check by hand for vulnerabilities, mainly because it cannot find it all (in this case it found them all).