**Requirement 1:**

**/home/test/Desktop/sprint2/riskXSS.py**

Vulnerable as `\` is a simple restriction which can be avoided using `img` or other techniques and weak code sanitisation uses replace, better methods such as whitelisting are more secure. This can lead to being vulnerable to XSS (cross site scripting). A simple fix could be to use whitelisting based input validation and sanitise server side inputs to prevent injection

YAML file and file to be scanned

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Corrected code without the error and semgrep scan

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

**/home/test/Desktop/sprint2/RiskDos.py**

Vulnerable as an inefficient sort algorithm is used -- attackers can send a large array of elements causing algorithm to take too long to complete resulting in DoS attack. Simple fix is using a more efficient sort algorithm

YAML file and file to be scanned

A screen shot of a computer code

Description automatically generated A computer screen shot of a code

Description automatically generated  
A screenshot of a computer

Description automatically generated

Corrected code with semgrep scan

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

**/home/test/Desktop/sprint2/riskLeak.c**

The code is vulnerable to a memory leak because if fopen fails the memory isn’t freed before the program exits which leads to a memory leak. A simple fix to this is adding free(buffer) if fopen potentially fails.

YAML file and file to be scanned

A computer screen shot of a computer error

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Corrected code with semgrep scan

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Requirement 2**

I installed docker and pulled a container for this requirement

**Sudo docker pull ollama/ollama**

**sha256:722ce8caba5f8b8bd2ee654b2e29466415be3071a704e3f4db1702b83c885f76**

**A computer screen shot of white text

Description automatically generated**

**syft ollama/ollama -o cyclonedx-json > ollama\_sbom.cyclonedx.json**

**jq . ollama\_sbom.cyclonedx.json**

**A computer screen shot of a computer program

Description automatically generated**

Dependency track SBOM uploaded vulnerabilities in browser at localhost:8080

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**Command for grype is **grype ollama\_sbom.cyclonedx.json** which created the txt file you can see in the screenshot below

**A screenshot of a computer

Description automatically generated**

**Requirement 3:**

I installed clang and libfuzzer

This is my vulnerable code I will run commands on

A white background with black text

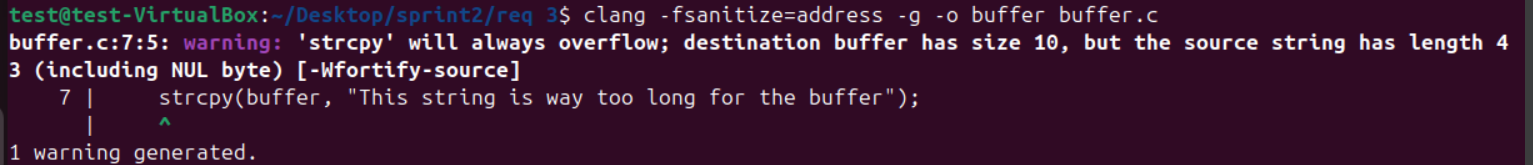
Description automatically generated

**gcc -Wall -Wextra -Wpedantic -o buffer buffer.c**

A screenshot of a computer program

Description automatically generated

**clang -fsanitize=address -g -o buffer buffer.c**



After running **./buffer** this was the whole output

A computer screen with text

Description automatically generated

A screenshot of a computer

Description automatically generated

A black and purple background

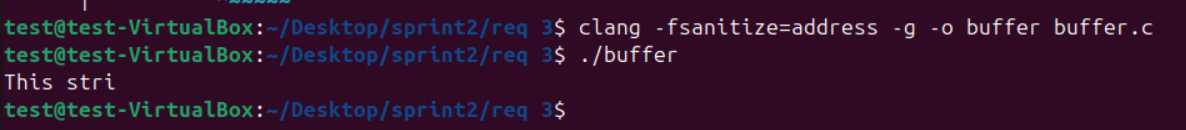
Description automatically generated with medium confidence

This is the updated and fixed c file

A computer screen shot of a computer code

Description automatically generated

After fixing the code and rerunning these commands this is my output



I made another vulnerable file which I would fuzz for the next section A screenshot of a computer code

Description automatically generated

The commands I used were

**clang -fsanitize=fuzzer -g fuzzTarget.c -o fuzz\_target**

**./fuzz\_target -runs=1**

I originally just used **./fuzz\_target** but it would give me the infinite output of ‘Crash!’ so I made it only fuzz the target once to give me a proper output

The sanitizer was able to pick up on the crash because it detected the undefined behaviour like dividing by 0 during the runtime

A screenshot of a computer program

Description automatically generated  
For the next task I used 3 different files

This file is my fuzz\_target.c, this file is the fuzzing harness for libfuzzer to catch the bugs

A computer screen shot of a code

Description automatically generated

This file is my target\_file.c, this file contains the implementation of the functions Im testing for

A computer code with text

Description automatically generated

This file is my target\_file.h, this file contains the declatrations of the functions from target\_file.c to be used in the fuzzing harness

A computer screen shot of text

Description automatically generated

The commands I used to get the result you see below are

**clang -fsanitize=fuzzer,address,undefined -g target\_file.c fuzz\_target.c -o fuzz\_target**

**./fuzz\_target -runs=1**

A computer screen shot of a computer code

Description automatically generated