Rust-C++-Robustheit

Implementierung meines Masterprojekts

🚀 Docker starten

🏗️ Build

```
docker build -t rust-cpp-robustness .
```

▶ Start

```
docker run -it  --memory=512m \
        --memory-swap=512m \
        --oom-kill-disable=false \
        --pids-limit=100 \
        --cpus=1 \
        --log-opt max-size=5m \
        --log-opt max-file=2 \
        --cap-add=SYS_PTRACE \
        --cap-add=SYS_ADMIN \
        --security-opt seccomp=unconfined \
        --security-opt apparmor=unconfined \
        --tmpfs /tmp:size=64m \
        --name rust-cpp-robustness \
        rust-cpp-robustness
```

🖥️ Zweite Bash starten

```
docker exec -it buffer_overflow /bin/bash
```

🔧 ASLR deaktivieren

Es kann hierfür entweder die `setup.sh` Datei genutz werden oder der folgende Befehl:

```
echo 0 | tee /proc/sys/kernel/randomize_va_space
```

✅ ASLR-Status prüfen

```
cat /proc/sys/kernel/randomize_va_space
```

⚙ C++-Code kompilieren

```
cd /cpp
rm -rf build
mkdir build
cd ./build
cmake -DCMAKE_BUILD_TYPE=Debug ..
make
```

▶ Programm ausführen

```
./src/[projektname]/[binary]
```

🦀 Rust-Code kompilieren

```
cd ./rust/[projektname]
cargo build
cargo run --bin [binary]
```

📂 Struktur des Repositories

Das Repository ist in verschiedene Verzeichnisse und Dateien unterteilt, die jeweils spezifische Funktionen und Inhalte beherbergen:

- `/cpp/`: Dieses Verzeichnis enthält den C++-Quellcode des Projekts. Hier befinden sich die Implementierungen der in C++ entwickelten Module. Die mit "_safe" bezeichneten Dateien geben eine sichere Version an, die keine Schwachstelle enthält.
- `/rust/`: In diesem Verzeichnis liegt der Rust-Quellcode des Projekts. Es umfasst die Implementierungen der in Rust entwickelten Komponenten. Die mit "_safe" bezeichneten Dateien geben eine sichere Version an, die keine Schwachstelle enthält.
- `.gitignore`: Eine Konfigurationsdatei, die festlegt, welche Dateien und Verzeichnisse von der Versionskontrolle durch Git ausgeschlossen werden sollen.
- `Dockerfile`: Ein Skript, das die Anweisungen zum Erstellen eines Docker-Images des Projekts enthält. Es definiert die Umgebung und die Schritte, die für den Aufbau der Anwendung erforderlich sind.
- `README.md`: Diese Datei bietet eine Übersicht über das Projekt, einschließlich Anweisungen zur Installation, Nutzung und anderen relevanten Informationen.
- `setup.sh`: Ein Shell-Skript zur deaktivierung des ASLRs

💥 Buffer Overflow in C++

1. Server-Datei kompilieren

2. Zweite Bash starten:

```
docker exec -it buffer_overflow /bin/bash
```

3. Adresse von `print_abracadabra` ermitteln:

```
nm ./server | grep "print_abracadabra"
```

4. Die Adresse von `print_abracadabra` in `client.cpp` eintragen
5. `client.cpp` kompilieren
6. Die Offset-Größe muss als Parameter übergeben werden. In Docker beträgt der Wert: **40**

🦀💥 Buffer Overflow in Rust

1. Rust-Datei kompilieren:

```
cd /rust/bufferoverflow/
cargo build
```

2. Client starten:

```
cargo run
```

3. Zweite Bash starten:

```
docker exec -it buffer_overflow /bin/bash
```

4. Die Adresse von `print_abracadabra` in `client.cpp` eintragen
5. C++-Datei kompilieren:

```
g++ -fno-stack-protector -z execstack -O0 -g -o client client.cpp
```

6. Die Offset-Größe muss als Parameter übergeben werden. In Docker beträgt der Wert: **88**