IT-Security Tutorübung 14

Dorian Zedler

4. Februar 2024

Technische Universität München

Inhalt der Tutorübung

- Tooling für sicheres Programmieren
- Fragen und Wiederholung für die Klausur

Quizizz

Quizizz

https://quizizz.com/admin/quiz/65c09fb6247ab7243072942c?searchLocale=

Hausaufgaben Präsentationen

Aufgaben 1

Aufgabe 1a - Warnings bei gcc

Lade die Datei `ueb14-sources.zip` von Moodle herunter!

a) Kompiliere Programme `vuln1` bis `vuln3` und finde Warnungen die auf die Existenz von sicherheitskritischen Bugs hinweisen. Compilerflags wie `-Wall -Wextra -Wpedantic` oder verschiedene Optimierungsstufen (`-O2`, `-O3`) können hilfreich sein!
   `gcc -o vuln vuln{1,2,3}.c`

Aufgabe 1a - Warnings bei gcc

Lade die Datei `ueb14-sources.zip` von Moodle herunter!

a) Kompiliere Programme `vuln1` bis `vuln3` und finde Warnungen die auf die Existenz von sicherheitskritischen Bugs hinweisen. Compilerflags wie `-Wall -Wextra -Wpedantic` oder verschiedene Optimierungsstufen (`-O2`, `-O3`) können hilfreich sein!
   `gcc -o vuln vuln{1,2,3}.c`
   - Bei `vuln1` und `vuln2` wird der Buffer overflow von `name` ohne weitere Argumente erkannt.
   - Bei `vuln3` wird die implizite Umwandlung von `atio(line)` zu `unsigned` nicht erkannt.

Aufgabe 1b - Linter

b) Als Ergänzung zu gcc gibt es sogenannte **Linter**. Führe den Linter `clang-tidy` auf `vuln{1,2,3}.c` aus.
`clang-tidy vuln{1,2,3}.c`

Aufgabe 1b - Linter

b) Als Ergänzung zu gcc gibt es sogenannte **Linter**. Führe den Linter `clang-tidy` auf `vuln{1,2,3}.c` aus.
`clang-tidy vuln{1,2,3}.c`

- Bei `vuln1` und `vuln2` wird nur empfohlen, memset_s statt memset zu verwenden
- Bei `vuln3` werden keine Probleme gefunden

Aufgabe 1c - Einsatz in der Praxis

c) Schaue nun vuln4 an. Versuche, mithilfe von gcc und clang-tidy, Schwachstellen und einen Angriffsvektor zu finden!

Aufgabe 1c - Einsatz in der Praxis

c) Schaue nun vuln4 an. Versuche, mithilfe von gcc und clang-tidy, Schwachstellen und einen Angriffsvektor zu finden!

- gcc erkennt overflows von `fgets` und `gets`.
  $\rightarrow$reichen aber aufgrund des stack Canaries nicht für einen Angriffsvektor!

Aufgabe 1c - Einsatz in der Praxis

c) Schaue nun vuln4 an. Versuche, mithilfe von gcc und clang-tidy, Schwachstellen und einen Angriffsvektor zu finden!

- gcc erkennt overflows von `fgets` und `gets`.
  →reichen aber aufgrund des stack Canaries nicht für einen Angriffsvektor!

- clang-tidy findet die printf magic!
- Mit Hilfe von Printf und dem "Namen"%9$p kann der Stack Canary geleakt werden.

Aufgabe 1d - address sanitizer

d) Führe nun die selben Programme mit der Compileroption `-fsanitize=address` aus und beobachte was passiert, wenn das Programm mit valider Eingabe bzw. mit einer Eingabe, die einen Buffer Overflow auslöst, ausgeführt wird.

Aufgabe 1d - address sanitizer

d) Führe nun die selben Programme mit der Compileroption `-fsanitize=address` aus und beobachte was passiert, wenn das Programm mit valider Eingabe bzw. mit einer Eingabe, die einen Buffer Overflow auslöst, ausgeführt wird.

- Der address sanitizer erkennt den Buffer Overflow und gibt eine Fehlermeldung aus.
- **Achtung:** Der address sanitizer ist nur ein Werkzeug zur Fehlersuche und kein Ersatz für sicheren Code!

Fragen? Feedback?

Bis zum nächsten Mal!