Fortgeschrittene Softwaretechnik
Wintersemester 2024/25
Name: Simon Szulik
Matr.Nr: 1474315

Datum: 31. Oktober 2024
Reading Group Sessions
Inhaltsskizze zu Block: 01

Titel des Papiers

An empirical study of the reliability of UNIX utilities

Kurzzusammenfassung

Das Paper untersucht die Zuverlässigkeit von UNIX-Diensten durch Eingabe eines Streams von zufälligen Zeichen. Getestet wurden knapp 90 Dienste (z. B. Editoren, Kompilierer, Mail-Programme, Shells, ...) auf 7 Unix Versionen. Simulierter Zufallsweg über einem DEA. Hierbei konnte bei jeweils 24% - 33% der Dienste ein *Crash* oder *Hang* verursacht werden. Unterschiede in Systemarchitekturen und Versionen beeinflussten die Anfälligkeit der Programme, da einige Systeme bestimmte Fehler tolerieren, während andere abstürzen.
Verwendete Tools zum Testen:

1. **fuzz**: Generiert zufällige Zeichenfolgen (druckbare, Steuerzeichen und NULL-Bytes)
2. **ptyjig**: Teste Interaktive Dienste mit Pseudo-Terminal-Datei (Da wo | nicht reicht)
3. Scripte zum Automatisieren der Tests

Mögliche Testergebnisse und „Zustände“ der Dienste & Klassifizierung der *Crash-Fehler*:

1. **Crash**: wenn das Programm sich abnormal beendet; Core-Dump wird erzeugt
   (a) **Pointer & Arrays**: Array-Grenzen und Null-Pointer-Referenzen führen oft zu Abstürzen; jedoch mögliche Unvorhersehbarkeit der Manifestation von unabsichtlichen Speicherüberschreibungen; benötigt Grenz-Test
   (b) **Rückgabewerte Ignorieren**: Das nicht Überprüfen von Dateienenden kann häufig zu Fehlern führen; extra Flags oder GoTo's ec.
   (c) **Overflow bei Inputs**: Eingabefunktionen wie *gets()* und *scanf()* haben teilweise keine Grenzen beim Lesen von Zeichenketten; Grenze einbauen
   (d) **Unterprogramme**: Schlechte Überwachung der Übergabeparameter und keine Fehlertoleranz für Unterprogramme; Schwer zu beheben
   (e) **Interaktionskonflike**: Unerwartete Zusammenarbeit von Eingaben und der Verarbeitung. Übergabe teilweise nicht an Parameter angepassen; simpleren Stringroutinen
   (f) **Schlechtes Error Handling**: Erneute Ausführung von Fehlerhaftem Code; Abbruch
   (g) **Signed Integer**: Negative Werte führen zu ungültigen Hashs; Unsigned Ints benutzen
   (h) **Race Conditions**: Einlesen von Steuerzeichen während Wiederherstellung des Terminalzustandes;
   (i) **Undefinierte Error**: Quellcode nicht verfügbar oder Ursache nicht klar
2. **Hang**: wenn sie endlos liefen, ohne auf Eingaben zu reagieren oder weiterhin Ausgaben produzierten, nachdem die Eingabe gestoppt wurde
3. **Succeed**: wenn das Programm normal terminiert; die Korrektheit der Ausgabe wird jedoch nicht überprüft.

Eigene (offene) Diskussionspunkte

1. „**It is useless to report bugs**“:
   Viele Nutzer melden Fehler nicht, da dies oft kompliziert und zeitaufwändig ist oder sie das Gefühl haben die Bug-Reports haben keinerlei Auswirkung. Mitlerweile sind Bug-Reports meistens eine One-Click Angelegenheit die einen Fehlerbericht mitliefern. Das senkt die Hemmschwelle Feedback zu geben. Ein Vorschlag, die Nutzer weiter zu ermutigen Bugs zu finden, versuchen zu reproduzieren oder zu veröffentlichen wäre eine Art Belohnungssystem. (Digital-Badges im Microsoft Account, Rabatte auf Software, ..., ähnlich zu den heutigen Bug Bounties) CI/CD könnte ebenfalls helfen um Fehler vor einem Release fest zu stellen.

2. **Bad Standards**:
   Viele der Fehler waren auf einfache Programmierfehler zurück zu führen wie das Missachten von Array-grenzen, falschen Speicherzugriffen, fehlerhaften Rückgabewerten oder nicht vorhandender Catches. Coding Standards, Programmier-Konventionen und intensive Code-Reviews sollten heutzutage dafür sorgen, dass solche Fehler vermieden werden.

3. **Immer mehr Distros könnten ein Problem sein**:
   Die Vielzahl an UNIX-Systemen und Linux-Distributionen sowie die unterschiedlichen Hardware-Architekturen (wie x86, ARM, RISC-V) machen umfassende Tests schwierig. Jede Systemvariante hat spezielle Anforderungen und Abhängigkeiten, wodurch ein Programm auf einer Distribution fehlerfrei läuft, aber auf einer anderen Probleme macht. Unterschiedliche Bibliotheken und Versionen sowie die Kombination verschiedener Hard- und Software ergeben zahllose Testmöglichkeiten. Virtualisierung und Containerisierung, etwa mit Docker, können hier teilweise helfen, doch hardware-nahe oder sicherheitskritische Tests lassen sich oft nur auf echter Hardware zuverlässig durchführen.