Fuzzing

Robert Buchholz, Kai Dietrich, Björn Lohrmann

Rechnersicherheit Praktikum WS 2007

26. Januar 2008

Fahrplan

Was ist Fuzzing?

Allgemein:

- Testmethode mit automatisch generierten Testfällen
- Testen von Grenzfällen mittels semi-valider Eingabedaten
- Ziel: Finden von Sicherheitslücken

Aus Sicht von Qualitätssicherung:

- Negatives Testen: Programm macht nichts, was es nicht soll
- Funktionales Testen: Programm macht, was es soll

Fuzzing = Negatives Testen

Beispiele für Fuzzing

Was lässt sich z.B. sinnvoll mittels Fuzzing testen?

- Implementierungen von Netzwerkprotokollen
  Beispiel: Ein DNS-Proxy :-)
- Programme zur Verarbeitung bestimmter Dateitypen
  Beispiel: Ein Video-Player
- Programmbibliotheken
  Beispiel: Eine Bibliothek für Reguläre Ausdrücke
- Programm das seine Kommandozeilenparameter verarbeitet
  Beispiel: Shell-Script, das für gegebene URL entsprechendes Programm startet

Motivation

Warum Fuzzing zum Finden von Sicherheitslücken?

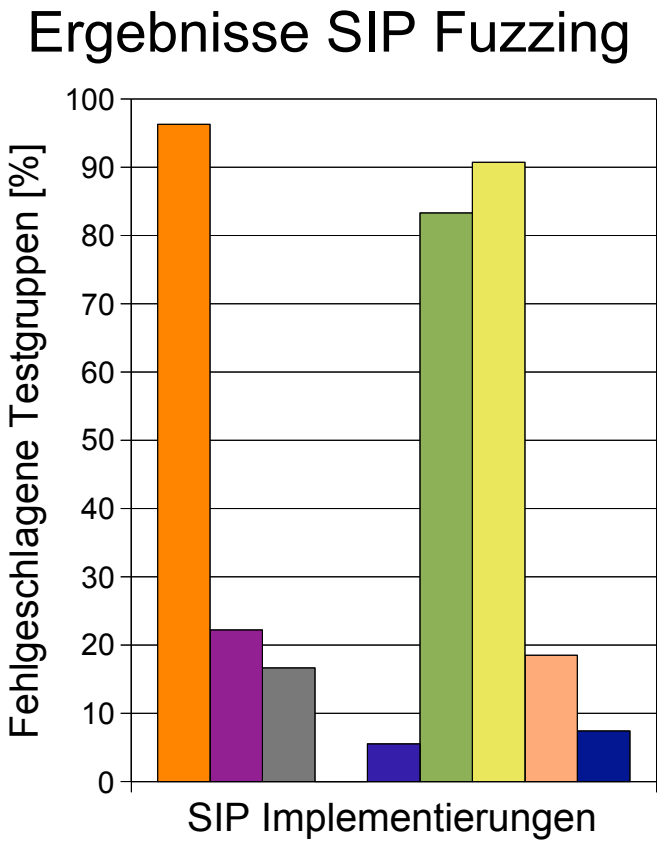
- Weniger personal- und zeitintensiv als z.B. Code Audits
- Stark automatisiert $\Rightarrow$ Hohe Anzahl an Testfällen
- Reproduzierbarkeit der Testfälle
- Hohe Effektivität

Motivation

Beispiel:
Fuzzing Ergebnisse ([OLOU])

- Gefuzztes Protokoll: Submenge von SIP
- 4527 Testfälle in 54 Testgruppen
- 9 Programme getestet (User Agents und Proxies)

Fazit: Potenzielle Schwachstellen in 8 Tools gefunden



Fahrplan

Identifikation von Eintrittspunkten

Identifikation von Eintrittspunkten

- Welche Kanäle für Programmzugriff?
- Welche Formate/Protokolle der einzelnen Kanäle?

Beispiele:

- Webapplication: HTTP Requests (Kanal), JSON (Format)
- Programmbibliothek: Funktionsaufrufe (Kanal), Datenstrukturen (Format)

Identifikation von Eintrittspunkten: Beispiel TeamSpeak 2

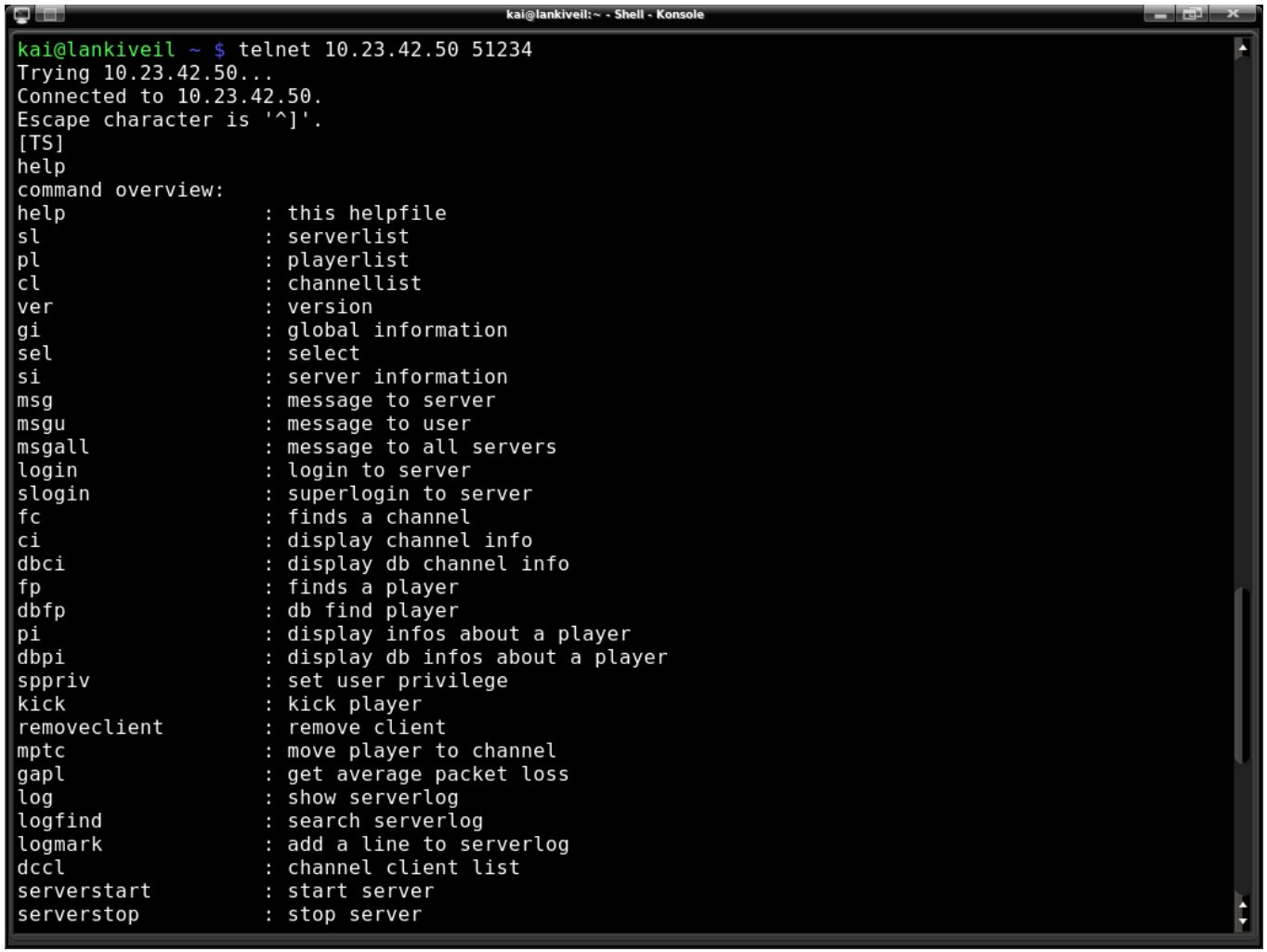
TeamSpeak 2:

- VoiceCom für Gruppen
- Sternförmige Architektur (Server/Client)
- UDP-basiertes Protokoll

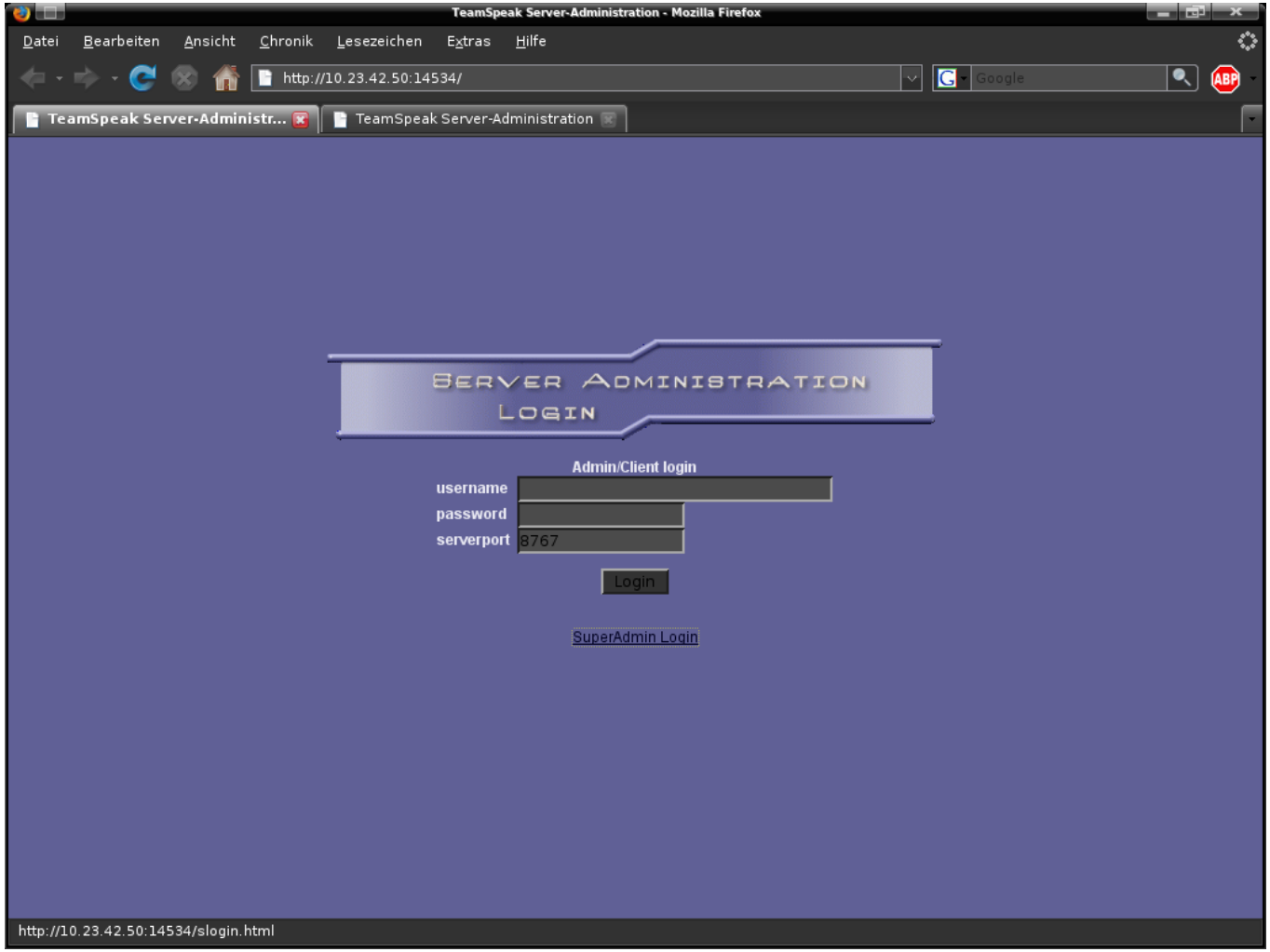
Identifikation von Eintrittspunkten: Beispiel TeamSpeak 2

```
voip ~ # netstat -tulpe | grep teamspeak
tcp        0      0 *:51234                 *:*                     LISTEN      teamspeak2 7208
    4317/server_linux
tcp        0      0 *:14534                 *:*                     LISTEN      teamspeak2 7207
    4317/server_linux
udp        0      0 *:8767                  *:*                                 teamspeak2 7205
    4317/server_linux
voip ~ #
```

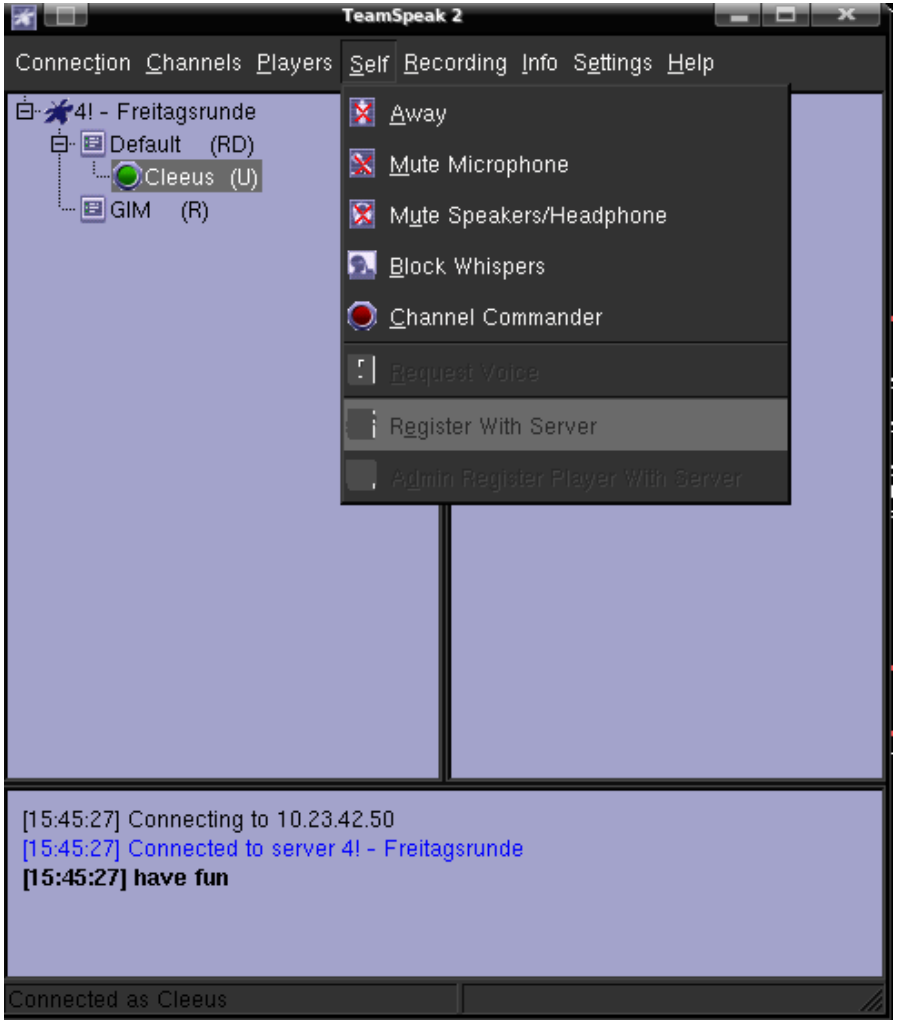
Identifikation von Eintrittspunkten: Beispiel TeamSpeak 2



Identifikation von Eintrittspunkten: Beispiel TeamSpeak 2



Identifikation von Eintrittspunkten: Beispiel TeamSpeak 2



Identifikation Überschrittener Trust Boundaries

Trust Boundary:

- Übergang von Daten/Kontrollfluss von einem Trust Level zu anderem
- Ort, an dem Schwachstellen zu Privilegienausweitung führen

Trust Level: Menge von Privilegien und Ressourcen

Frage: Welche Trust Boundaries bestehen bei vorher identifizierten Eintrittspunkten?

Beispiel:

- Aufruf einer Kernelfunktion
- Eingaben an SUID-Binary

Identifikation Überschrittener Trust Boundaries: Beispiel

Trust Boundaries in TeamSpeak:

- Web-Admin Authorisierungsdialog
- Telnet vor der Authorisierung
- VoIP Traffic im TeamSpeak Protokoll
  - UDP Connect Packet im TeamSpeak Protokoll

Selektieren der Eintrittspunkte

Optimierung von Kosten und Nutzen:

- Überschrittene Trust Boundary
- Komplexität der Eingabedaten
- Erwartete Fehlerdichte

Selektieren der Eintrittspunkte: Beispiel

Alle 3 Dienste gleich mächtig:

- User → Admin
- User → `teamspeak2` Systembenutzer

Ausgewählter Eintrittspunkt: UDP Connect Packet:

- Web und Telnet können durch Firewall gesperrt werden
- UDP Connect Packet kann nicht gesperrt werden

Generierung semi-valider Daten

Ursprung der Daten

- Generieren nach Formatdefinition
- Mutation existierender Daten

Format der Daten

- Protokollwissen im Fuzzer (intelligent)
- Blinder Fuzzer

Generierung semi-valider Daten (Beispiele)

	Generieren	Mutieren
Protokollwissen	PROTOS (SIP)	SPIKE, zzuf
Blind	`cat /dev/random | ./app`	QueFuzz

Generierung semi-valider Daten (Beispiele): TeamSpeak 2 Connect Packet

Demo

Erkennung von Fehlern

In-Band: über den Eingangskanal

- Beobachte Antworten
- Wie erkenne ich Fehler?
- Wie erkenne ich Funktion?

Out-of-Band: auf Zielplattform

- Instrumentierung / Dynamische Code-Analyse (Debugger, Valknut)
- Ausgaben, Logdateien
- Ressourcenverbrauch

Erkennung von Fehlern: Offene Fragen

- Was sind valide Antworten?
- Reproduzierbarkeit
- Ursprung vs. Manifestation von Fehlern

Fahrplan

Zusammenfassung

- Fuzzing ist eine Testmethode
- Negative, automatisch generierte Testfälle
- Implementation eines Fuzzers:
  1. Eintrittspunkte und Trust Boundaries
  2. Priorisierung der Eintrittspunkte
  3. Datengenerierung
  4. Fehlererkennung

?

[OLOU] „PROTOS Test-Suite: c07-sip“, Im Internet unter:
http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/
Computer Engineering Laboraty, University of Oulu, Stand vom 24.1.2008

[OE05] Peter Oehlert : „Violating Assumptions with Fuzzing“,
Security & Privacy Magazine, IEEE, March-April 2005

[DM06] Jared DeMott: „The Evolving Art of Fuzzing“
Unveröffentlicht. Im Internet unter:
http://www.vdalabs.com/tools/The_Evolving_Art_of_Fuzzing.pdf

[AI02] Dave Aitel: „The Advantages of Block-Based Protocol Analysis for Security“
Unveröffentlicht. Im Internet unter:
http://www.net-security.org/dl/articlesadvantages_of_block_based_analysis.pdf

[SP05] Ilja van Sprundel: „Fuzzing: Breaking software in an automated software“
Unveröffentlicht. Im Internet unter:
http://events.ccc.de/congress/2005/fahrplan/attachments/582-paper_fuzzing.pdf