

Fuzzing

Daniel Leese, Fabian Brunner

Index Terms

Fuzzing, Netzwerk Sicherheit, Axman, TAOF, FTPStress

1 WAS IST FUZZING?

IM Gegensatz zu anderen prägnanten Begriffen oder Akronymen der Informationstechnologie ist es nur schwer möglich aus dem Begriff "Fuzzing" seine Bedeutung abzuleiten. Hinter dem eher lustig klingenden Namen verbirgt sich jedoch eine erprobte und zuverlässige Software-Testing-Technik.

Im Software-Testing kann man prinzipiell zwischen White-Box und Black-Box Verfahren unterscheiden. Bei den erstgenannten handelt es sich, vereinfacht gesagt, um Verfahren bei denen die genaue interne Funktionsweise, oder gar der Quellcode, der zu evaluierenden Software bekannt ist. Bei den letztgenannten Black-Box Verfahren hingegen ist über das Programm nicht mehr bekannt als seine Ein- und Ausgabefunktionen.

Fuzzing eignet sich sehr gut zum finden von Programmfehlern, Sicherheitslücken und allgemein um ein Programm auf seine Robustheit zu testen. Hierzu gibt es drei verschiedene Arten von Fuzzing-Software:

- Spezielle Fuzzing-Tools, welche nur ein bestimmtes Programm oder Protokoll angreifen bzw. Fuzzern.
- Komplette Fuzzing-Suiten, die ein breiteres Spektrum angreifen können, wie z.B. alle verbreiteten Layer 7 Anwendungsprotokolle. Solche Suiten enthalten oft hunderte oder gar Tausende Unterprogramme.
- Selbstgeschriebene oder gesciptete Tools. Gerade unter unixoiden Betriebssystemen, welche mächtige Scripting und Netzwerkfunktionalitäten bereitstellen, kann ein solches Fuzzingtool mit wenigen Zeilen und geringen Programmierkenntnissen erzeugt werden.

1.1 Geschichte

- Daniel Leese, Fabian Brunner, Studenten der Hochschule Furtwangen, Studiengang Computer Networking, 4. Semester
E-mail: leese@hs-furtwangen.de, fabian.brunner@hs-furtwangen.de

TESTTOOLS die mit Zufallswerten arbeiten gibt es vermutlich schon seit den Anfängen der Softwareentwicklung. Als eigenständiger Aspekt im Softwaretesting und vor allem in der IT-Sicherheit wurde es jedoch erst 1989, von Professor Barton Miller der Universität von Wisconsin-Madison, etabliert.

Dieser schrieb ein Semesterprojekt mit dem Titel "Operation System Utility Program Reliability - The Fuzz Generator" aus, welches zum Ziel hatte ein Programm mit dem Namen "fuzz" zu entwickeln, das bekannte und gebäuchliche UNIX-Konsolenanwendungen mittels zufällig erzeugter Eingaben auf Robustheit zu testen. Neben dem simplen crashen der Programme sollte hier ebenfalls ausgewertet werden welche Eingabe der Programm letztendlich zum Absturz brachte. Dies ist noch heute, fast 20 Jahre später, ein anspruchsvoller Aspekt von Fuzzing. Insbesondere das Monitoring, d.h. herauszufinden zu exakt welchem Zeitpunkt, und so bei welcher Eingabe, das getestete Programm seinen Dienst quittiert, ist oftmals schwierig.

Im Umfeld von Professor Miller entstanden im Laufe der Zeit mehrere Arbeiten zum Thema Fuzzing. Hier ein Ausschnitt aus, nach Meinung der Autoren, interessanten Arbeiten und deren Ergebnissen:

- Der "Fuzz Revisited" Report von 1995
In diesem Report wurden die 1989 erstmalig getesteten UNIX Konsolenanwendungen erneut getestet und um Tests von graphischen "X"-Anwendungen ergänzt. Diese Studie hatte unter anderem zum Ergebnis das unter Linux 9% der getesteten Anwendungen für Fuzzing anfällig waren und abstürzten.
- Der "Windows NT Fuzz Report" aus dem Jahr 2000
Hier wurden populäre Windowsanwendungen wie z.B. Visual C++ 6.0, Office 2000, IE5, usw einem Fuzzing Test unterzogen, wobei Zufalls generierte Daten in deren Window Message Queues geschrieben wurden. Hierbei versagten erschreckenderweise 85.7% der getesteten "professionellen" Anwendungen den Dienst.
- Das aktuellste Paper, "Mac OS X Fuzz Report" von 2006
Was bisher verschiedene UNIXe und Microsoft-Betriebssysteme über sich ergehen lassen mussten traf nun das aufstrebende kommerzielle "MAC-OS X" Betriebssystem. Auch hier wurden "professionelle" GUI Applikationen von Apple und Microsoft (z.B. Adobe Reader 7, Finder 10.4.3, iTunes 6.0.1, Microsoft Office 11) sowie teilweise von UNIX bekannte Konsolenanwendungen getestet. Erstaunlicherweise lies hier das Apple Betriebssystem sogar Linux knapp hinter sich, es zeigten sich nur bei 7% der getesteten Anwendungen Anfälligkeiten für Fuzzing.

Hieraus zogen die Autoren, obwohl es sich, durch die Unterschiede in den getesteten Anwendungen und die großen zeitlichen Abstände, nicht um einen gänzlich fairen Vergleich handelt, den Schluss das Apple und Linux sich etwa auf gleichem Niveau befinden. Microsoft jedoch spielt bei der Ro-

bustheit seiner Applikationen, und somit auch indirekt bei deren Sicherheit, in einer völlig anderen Liga.

1.2 Funktionsweise

MAN kann prinzipiell zwei verschiedene Vorgehensweisen des Fuzzing unterscheiden: Das senden völlig zufälliger Daten oder die vorherige Analyse der möglichen Eingabedaten. Beide Methoden haben unterschiedliche Vor- und Nachteile:

Beim senden völlig zufälliger Datenströme kann eine Vielzahl von Fehlern aufgedeckt werden, auch solche auf die der Entwickler bzw. Tester bei einer Analyse nicht gekommen wäre. Allerdings kann der Fuzzing-Vorgang sehr lang dauern, da der komplette Raum möglicher Eingaben durchlaufen werden muß.

Bei der zweiten Methode wird der Eingaberaum von Anbeginn durch Analyse des Protokolls stark eingegrenzt. Somit kann ein wesentlich gezielterer "Angriff" erfolgen da zu unsinnige Eingaben, die sowieso gefiltert werden würden und nur Zeit kosten, wegfallen. Jedoch können so auch manche Fehler übersehen werden, da nicht alle Möglichkeiten durchlaufen werden.

Diesen Methoden bedienen sich drei Arten von Fuzzing-Programmen: Manuelle, bei denen jede Eingabe einzeln generiert und abgesendet wird, Automatische, welche die Eingaben nach vorher definierten Regeln generieren und massenhaft senden und auswerten und halbautomatische, die eine Mischform darstellen.

1.3 Was fuzzen?

GEFUZZT werden können eine Vielzahl von "Angriffszielen", nicht nur Webbrowser, auch wenn Fuzzing durch diese populär wurde:

- Netzwerkprotokolle und Anwendungen wie z.B. FTP
 - Argumente, Umgebungsvariablen, stdin, file-descriptors
 - APIS (syscalls, library calls)
 - Binär- sowie ASCII-Dateien, welche von Programmen gelesen werden
- Auf einige dieser Angriffsziele wird im Kapitel 3 noch näher eingegangen werden.

1.4 Ergebnisse

EIN erfolgreiches Fuzzing hat in der Regel einen Absturz des gefuzzten Programms zur Folge. Dies kann verschiedenste Gründe haben, wie z.B. Array out of bounds exceptions, Buffer overflows oder Stack overflows. Seltener kommt ein Einfrieren des Programms vor, d.h. es werden keine Benutzereingaben mehr entgegengenommen. Schliesslich kann auch noch,

vor allem bei sog. embedded Geräten, schlichtweg die Hardware überlastet werden, was zu einem Neustart führt.

Aus vielen "Fuzzing Schwachstellen", vor allem aus den o.g. Overflows können bei entsprechend tiefgehender Analyse Code injection Möglichkeiten entstehen die das System kompromittieren können (sog. Exploits).

2 BEKANNTE FÄLLE

2.1 MoBB von H.D.Moore

BEIM Month of Browser Bug veröffentlichte H.D. Moore täglich eine Schwachstelle in Browsern, die meisten im Internet Explorer 6 und einige in Firefox, Opera, Konqueror und Safari.

2.2 PROTOS

IN folgenden Geräten/Software wurden mit Hilfe von PROTOS Schwächen bei der Verarbeitung von INVITE-messages in SIP gefunden.

- Cisco IP Phone Model 7940/7960
- Ingate Firewall
- Alle Versionen von SIP Express Router bis 0.8.9
- Mozilla Firefox IBM SecureWay V3.2.1 (Solaris und Windows)
- Network Associates PGP Keyserver 7.0
- ...

2.3 mangleme

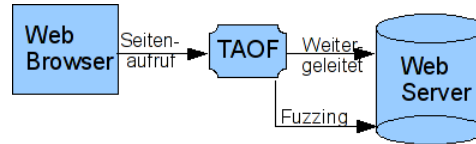
DURCH mangleme wurden in nahezu allen bekannten Browsern Fehlern gefunden. Beispielsweise in:

- Microsoft Internet Explorer: u.A. IFrame Bug
- Netscape Navigator / Mozilla Firefox
- Links
- Lynx
- Opera
- Safari
- ...

3 FUZZING DEMONSTRATION

3.1 TAOF (The Art Of Fuzzing)

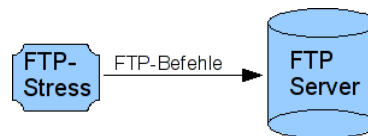
TAOF ist ein GUI-basiertes, einfach einzurichtendes Fuzzingtool für Netzwerkprotokolle.



Im ersten Schritt setzt sich TAOF zwischen Browser und Webserver um den Traffic zu loggen. Anschließend kann der Teil des Protokolls ausgewählt werden, welcher gefuzzt werden soll. Im zweiten Schritt wird dann, der in Schritt 1 ausgewählte Teil des Protokolls mit zufälligen Zeichen gefüllt und an den Server gesendet. Das wird solange wiederholt bis keine Antworten mehr vom Server kommen oder der Anwender abbricht.

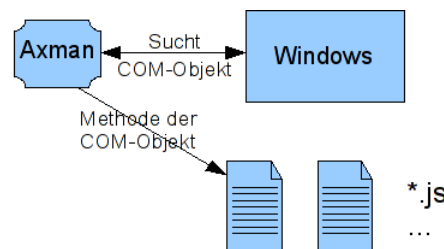
3.2 FTPStress

MIT FTPStress lassen sich die verschiedenen FTP-Befehle (wie beispielsweise `cd`, `dir`, `get`, ...) mit zufälligen Zeichenketten (incl. Umlaute und Sonderzeichen) unterschiedlicher Länge füllen um so die korrekte Verarbeitung der Eingaben vom Server zu prüfen.

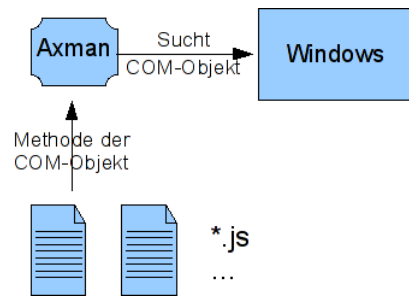


3.3 AxMan

DER webbasierte ActiveX Fuzzer AxMan besteht aus zwei Teilen
Teil 1:



Das Tool `axman.exe`, welches nach installierten COM-Objekte sucht, dessen Methoden analysiert und in ein JavaScript schreibt. Teil 2:



Wenn die Javascript Skripte erstellt sind, kann über das Webinterface Ax-Man gestartet werden. Es werden alle Javascript Skripte nacheinander mit zufälligen Parametern aufgerufen.

3.4 Shellscripting

ALS Beispiel wie ein an der Funktionsweise von Computern interessierter Anwender mit einem potenten Betriebssystem leicht selbst Sicherheitslücken finden kann dient folgendes Beispiel:

Der DrayTek Vigor ist eine auf kleinere Betriebe ausgerichtete Security Appliance mit der die Autoren in einem Projekt bei Prof. Dr. Karduck in Kontakt kamen. Das eigentliche Ziel des Projektes war der Aufbau eines VPN; jedoch fühlten sich die Autoren durch die, fast schon ironisch wirkenden, vollmundigen Marketingtexte zu dem Gerät herausgefordert.

So wurde ein simples Script entwickelt welches per netcat /dev/urandom auf die Ports der von der Appliance bereitgestellten Dienste leitet.

```

xterm
#!/bin/bash

# endless loop
while [ 1 ]
do
    #cat /dev/urandom | netcat 192.168.0.1 21 & # FTP
    #does nothing

    #cat /dev/urandom | netcat 192.168.0.1 22 & # SSH
    #sure crash ~10sec

    #cat /dev/urandom | netcat 192.168.0.1 23 & # Telnet
    #no crash, but intresting behaviour

    cat /dev/urandom | netcat 192.168.0.1 80 & # HTTP
    #sure crash ~10sec

    #cat /dev/urandom | netcat 192.168.0.1 443 & # HTTPS
    #does nothing
done
"easyHacking.sh" 20L, 430C          20,1          Alles
  
```

Dieses Script zwingt das Gerät auf verschiedenen Ports nicht nur zuverlässig zum Neustart, es produzierte auch folgende interessante Ausgabe:

```

xterm

Password: *****
Password: ****
Password: *****
Bad Password, Bye-Bye :-(<

System administrator is connecting from 192.168.1.10
Reject the connection request !!!
0000

Password: *****

*** WARNING *****
* System has no password. *
* Please set password, using "sys passwd" commands. *
*****

Type ? for command help

> J0sT?^D
^C^C

```

Hier heraus einen funktionsfähigen Exploit zu entwickeln überstieg die ohnehin sehr knapp bemessene Zeit der Autoren bei weitem, könnte jedoch ein interessantes Thema für zukünftige Untersuchungen sein.

3.5 Weitere Fuzzingtools

- Netzwerk
 - Spike (Netzwerkprotokolle, Sprache: C)
 - **TAOF** (The Art of Fuzzing, alle TCP/UDP Protokolle)
 - Bed (Netzwerkprotokolle, Sprache: Perl)
 - Smudge (Netzwerkprotokolle)
 - DHCPfuzz (DHCP-Clients)
 - **FTPStress** (FTP-Server)
 - Fuzzball2 (TCP/IP-Header)
 - IRCfuzz (IRC-Clients)
 - Protos (Tool-Sammlung für SIP, SNMP, DNS und andere)
 - SSHredder (Sammlung mutierter SSH-Pakete)
- Browser
 - COM-Bust (Common Object Model)
 - **Axman** (Microsoft ActiveX)
 - CSS-Die (Stylesheet Implementierung)
 - DOM-Hanoi (Data Object Model)
 - mangleme (automated broken HTML generator)

4 QUELLEN

- 22C3, Fuzzing - Breaking software in an automated fashion (<http://events.ccc.de/congress/2005/fahrplan/events/537.de.html>)
- 2006 Mac OS X Fuzz Report, Barton Miller, University of Wisconsin-Madison
- 2000 Windows NT Fuzz Report, Barton Miller, University of Wisconsin-Madison
- B.P. Miller, L. Fredriksen, and B. So, An Empirical Study of the Reliability of UNIX Utilities, Communications of the ACM 33
- <http://www.heise.de/security/Die-Axt-im-Walde-/artikel/76512>
- <http://theartoffuzzing.com/>

DANKSAGUNGEN

Der Autor dankt hiermit den folgenden Personen:

- Prof. Dr. Achim Karduck, für die Bereitstellung des DrayTek Vigor
- Prof. Dr. Christoph Reich, für die Latex-Vorlage
- Philip Waldhauer, für die Mithilfe bei der Entwicklung des easyhack.sh