DuD
Datenschutz und Datensicherheit

Fuzzing

Dirk Fox

Hintergrund

Die Entwicklung sicherer Software ist eine der derzeit größten Herausforderungen für die Anbieter von Betriebssystem-, Web- und Anwendungs-Software. Dabei geht es im Kern darum zu verhindern, dass Programmierfehler von einem Angreifer ausgenutzt werden können, um auf diese Weise Zugriff auf geschützte Daten zu erhalten oder sogar den gesamten Rechner „fernzusteuern“.[1]

Häufig wird dies dadurch erreicht, dass z. B. eine Web-Anwendung dazu gebracht wird, eine Dateneingabe als Programm zu interpretieren und auszuführen („Cross Site Scripting“) oder durch Fehleingaben (zu lange Eingabe, falsches Datenformat, zu große oder negative Werte etc.), die vom Programm nicht verhindert werden, den Programmablauf zu manipulieren.

Angreifer haben in der Regel keinen Zugriff auf den so genannten Quellcode, d.h. das in einer Programmiersprache geschriebene „Original“ eines Programms, sondern höchstens auf den daraus erzeugten Programmcode; bei Web-Anwendungen meist nicht einmal das. Daher sind sie auf andere Methoden zur Fehlersuche angewiesen – wie zum Beispiel Fuzzing.

Verfälschte Eingaben

Kennt man den Programmcode nicht, lassen sich Fehler nur finden, indem man ihr Auftreten beobachtet, dokumentiert und analysiert. Nun treten Fehler bei einem getesteten Programm in der Regel nicht bei „normaler“ Nutzung auf, sondern dann, wenn das Programm eine seltene Ablaufvariante durchläuft, die bei den Tests vergessen oder übersehen wurde. Solche Fehler lassen sich im Normalbetrieb meist nicht beobachten – man kann ihr Auftreten aber gezielt auslösen. Wie kann das funktionieren, wenn man noch gar nicht weiß, wo ein Fehler steckt?

Die Antwort darauf gaben 1990 drei amerikanische Forscher, die „Fuzzing“ als effiziente Methode zur Suche nach Programmierfehlern in Unix-Tools erfanden. Die Idee: Etwas vereinfacht lässt sich ein Programm als eine Funktion f(x) betrachten, die abhängig von den Eingabedaten (Input) x eine Ausgabe (Output) y erzeugt. Füttert man nun ein Programm über die Eingabeschnittstelle mit allen möglichen, insbesondere „unüblichen“ Eingabewerten, dann ist die Wahrscheinlichkeit hoch, dass man einen seltenen Programmpfad oder einen unerwarteten Status im Programmablauf erwischt (z.B. die Fehlerüberprüfungsroutine oder eine unzulässige Parametergröße). Hat das Programm hier einen Fehler, reagiert es unkontrolliert oder stürzt ab. Ein gutes Drittel der von den Autoren 1990 so getesteten Unix-Tools quittierte den Dienst.

Diese Test-Technik lässt sich auf jede Anwendung übertragen, wenn man nicht nur manuelle Eingaben berücksichtigt, sondern auch Datenpakete, die eine Anwendung empfängt und auswertet, Dateien, die sie öffnet und Speicherinhalte, die sie gemeinsam mit anderen Anwendungen nutzt (bspw. die „Zwischenablage“ des Systems).

Systematik

Im Allgemeinen ist die Zahl der möglichen Werte eines Eingabeparameters viel zu groß, um mit einem Durchprobieren aller Varianten zum Ziel zu kommen – das kann schon bei einem nur acht Byte langen Parameter auf einem schnellen System leicht einige Millionen Jahre dauern. Dabei ist nur für wenige Eingabewerte zu erwarten, dass sie einen Fehler provozieren: So ist die Längenangabe einer Bilddatei viel interessanter als jeder einzelne Bildpunkt. Hinzu kommt, dass viele Eingaben wie beispielsweise empfangene Datenpakete standardisierten Formaten genügen müssen. Haben die Eingaben das falsche Format, werden sie häufig schon abgewiesen, bevor sie das eigentliche Angriffsziel erreichen – z.B. durch den Netzwerkprotokolltreiber oder den Syntax-Checker eines Webservers.

Effizientes Fuzzing erfordert daher eine möglichst genaue Kenntnis der Formate der jeweiligen Eingabeparameter und Erfahrungswissen darüber, bei welchen Teilen der Eingabe (wie z.B. Längenangaben) häufig Fehler gemacht werden. Neben dem Ausprobieren von bereits in anderen Anwendungen gefundenen, Fehler provozierenden Eingaben verfolgen Fuzzing-Tools bei der Selektion der Testwerte, mit denen sie die Eingabeparameter füttern, zwei verschiedene Strategien (oder eine Kombination aus beiden): (pseudo-)zufällige Wahl oder Bit Flipping. Beim Bit Flipping wird die untersuchte Anwendung mehrfach mit einer zulässigen Eingabe (ein Wert, ein Datenpaket) gefüttert, in der jedes Mal zuvor ein einzelnes Datenbyte verändert wurde. Damit wird gewissermaßen „um die zulässige Eingabe herum“ gesucht – häufig stößt man dabei schneller auf Fehler als bei einer rein zufälligen Wahl des Eingabewertes.

Nachanalyse

Mit dem Finden eines Fehlers und der Herbeiführung eines Programmabsturzes ist das Ziel aber noch nicht erreicht. Es gilt nun, den Fehler genauer einzugrenzen: Was hat den Absturz verursacht? Welcher Fehler ist dem Programmierer dabei wahrscheinlich unterlaufen? Und wie lässt sich der Programmfehler zu einem Angriff auf das System missbrauchen? „Denial of Service“-Attacken, bei denen das Zielsystem kurzzeitig außer Gefecht gesetzt wird, sind zwar hässlich (und für das betroffene Unternehmen ggf. auch kostspielig); weit interessanter für einen Angreifer ist es jedoch, die Kontrolle über das angegriffene System zu gewinnen.

Bei dieser Analyse kann Fuzzing helfen, indem z.B. ähnliche Eingabewerte ausprobiert werden wie der, der den Fehler oder Programmabsturz verursacht hat. Mit klassischen Analyse-Tools (z.B. Debuggern), einer Untersuchung des Prozessor- und Systemstatus beim Programmabsturz und, falls möglich, einer parallelen Untersuchung des Quellcodes lässt sich der Fehler weiter eingrenzen.

Bewertung

Spezielle Fuzzing-Tools sind ein sehr effizientes Hilfsmittel, um versteckte Programmierfehler aufzudecken, die in Tests „durchrutschen“. Allerdings finden Fuzzer nie alle Fehler, die beispielsweise eine systematische Codeanalyse aufdecken würde. Der Erfolg von Fuzzing-Analysen spricht dafür, dass sich durch eine systematische Nutzung von Fuzzing durch die Hersteller die Zahl sicherheitskritischer Fehler erheblich senken ließe.

[1] Siehe DuD 10/2006, Schwerpunkt „Sichere Software-Entwicklung“.