Experimentelle Methoden zum Aufspüren von Einbrüchen

Stephan Neuhaus
Lehrstuhl für Softwaretechnik
Universität des Saarlandes
Stephan.Neuhaus@acm.org

23. März 2006

1 Einführung

Die Analyse von Computereinbrüchen zählt nach wie vor zu den anstrengendsten Tätigkeiten, die ein Informatiker ausüben kann. Warum gibt es für dieses Problem keine automatische Unterstützung? Weil vorhandenen Werkzeuge auf einer ungeeigneten Methodik basieren.

Die Analyse von Einbrüchen strebt danach, aus dem aktuellen Zustand des Systems zu rekonstruieren, wie es zum Einbruch kommen konnte. Dazu werden die vorhandenen Spuren analysiert und dann geschlossen, was in dem System passiert sein muss, damit diese Spuren entstehen konnten. Beispielsweise könnte eine Analyse des Linux Slapper-Wurms so aussehen: „Angreifer mit der IP-Adresse 10.120.130.140 haben eine besonders präparierte HTTPS-Anfrage an unseren Web-Server geschickt, der einen fehlerhaft formatierten Client-Schlüssel enthielt. Das verursachte einen Pufferüberlauf und rief eine Shell auf. Diese Shell speicherte dann eine mit *uuencode* kodierte Kopie des Wurm-Quellcodes, dekodierte und kompilierte sie und startete das erzeugte Programm unter dem Namen *.bugtraq*. Sobald das Programm ablief, versuchte der Wurm, andere Rechner des Netzwerks zu kontaktieren.“ (Beispiel aus [4].)

Soll nun ein Analyst diesen Vorfall untersuchen, und sieht er dabei nur den *.bugtraq*-Prozess, ist eine seiner ersten Aufgaben, die Prozesse zu isolieren, die an dem Angriff beteiligt sind, und zwar sowohl Prozesse, die noch ablaufen, als auch solche, die bereits terminiert sind. In diesem Beispiel sind das der Web-Server, die (terminierte) shell, die entsprechenden *cat*, *uudecode*, und C-Übersetzer-Aufrufe, und endlich der *.bugtraq*-Prozess.

Die übliche Vorgehensweise ist, bei einer Verletzung der Sicherheitsrichtlinien zu beginnen (hier wäre das der nicht erwünschte *.bugtraq*-Prozess) und sich dann mit Hilfe von Log-Dateien und Toolkits wie The Coroner's Toolkit rückwärts bis zur ursprünglichen Ursache zu bewegen (hier zu der fehlerhaft formatierten HTTPS-Anfrage). Dieses deduktive Verfahren hat aber gravierende Nachteile:

Vollständigkeit. Die Spuren sind möglicherweise nicht ausreichend, um die Ursache-Wirkungs-Kette verlässlich zu ermitteln.

Minimalität. Die wichtigen Spuren sind oft in einer Menge von anderen Spuren verborgen und müssen erst mühsam herausgearbeitet werden.

Korrekheit. Unsere Beweisführung könnte auf falschen Annahmen basieren, die unsere Schlussfolgerungen in Frage stellen.

Wir haben dazu eine Software namens Malfor (kurz für MALware FORensics) entwickelt, die in der Lage ist, durch *experimentelle* Verfahren die oben genannten Nachteile zu vermeiden. Anstatt Spuren zu interpretieren und rückwärts eine Ursache-Wirkung-Kette aufzubauen, geht Malfor experimentell vor: In einer ersten Phase werden während des laufenden Betriebs Ereignisse (hier Systemaufrufe) *aufgezeichnet*. Sobald ein Einbruch festgestellt wird, werden diese Ereignisse verwendet, um das System in Teilen zu *wiederholen*. Durch geschickte Auswahl der zu wiederholenden Ereignisse werden nach und nach diejenigen Ereignisse isoliert, die für den Einbruch relevant sind: Geschieht eine Wiederholung ohne ein bestimmtes Ereignis X und findet der Einbruch auch ohne X statt, kann X für den Einbruch nicht relevant gewesen sein. Malfor wiederholt ganze Prozesse und kann so die für den Einbruch relevanten Prozesse automatisch isolieren.

Kennt man die relevanten Prozesse, bieten sich eine Reihe von Möglichkeiten, die Anfälligkeit des Systems für weitere Angriffe zu reduzieren und damit die Qualität des Systems zu erhöhen. Dazu gehören Maßnahmen wie das Abschalten der betroffenen Dienste, aber auch die automatische Extraktion von Angriffsvektoren mit dem Ziel, Proxies damit ausstatten zu können, um gleichartige Angriffe in der Zukunft verhindern zu können.

2 Aufzeichnen und Wiederholen

Unser Subsystem zum Aufzeichnen und Wiederholen von Systemaufrufen funktioniert durch System Call Interposition. Bei diesem Verfahren werden Systemaufrufe wie *fork*, *execve*, *read*, *getpid* und so weiter auf eigene Routinen umgeleitet, welche die Ergebnisse in einer Datenbank speichern, beziehungsweise aus ihr laden. Im Sicherheitsbereich wurde dieses Verfahren beispielsweise in *systrace* verwendet, um Systemaufrufe zur Laufzeit mit Sicherheitsrichtlinien belegen zu können.

Beim Wiederholen von Systemaufrufen sind viele Details zu beachten, wenn die Wiederholung gelingen soll. Beispielsweise muss dem Aufrufer unter Umständen eine andere Prozessnummer vorgegaukelt werden, als er im Betriebssystem tatsächlich besitzt, damit Bibliotheksaufrufe wie *gethostbyname*, welche die Prozessnummer in DNS-

Anfragen kodieren, bei der Wiederholung noch genauso funktionieren wie bei der Aufzeichnung.

Unser Verfahren funktioniert, indem die aufgezeichneten Prozesse in immer neuen Konfigurationen wiederholt werden sollen. Dazu ist es nötig, Prozesse nach Belieben von der Wiederholung ausschliessen zu können. Jetzt kann man aber einen Prozess nicht daran hindern, *fork* aufzurufen und einen neuen Prozess zu erzeugen. Man kann aber die Prozesserzeugung auch nicht fehlschlagen lassen, da das eine zu starke Abweichung zum ursprünglichen Lauf darstellt. Um also den so erzeugten Prozess an der Berechnung zu hindern, wird er beim nächsten Systemaufruf mit dem aufgezeichneten Exit-Code beendet.

Die Verwendung solcher Maßnahmen ist typisch für die Wiederholung nur eines Teils des Systems.

3 Minimieren

Um aus allen Prozessen diejenigen herauszufinden, die für den Einbruch relevant sind, verwenden wir Delta Debugging [2]. Delta Debugging ist eine Technik, die durch wiederholte Experimente aus einer Menge von fehlerverursachenden Umständen (hier die aufgezeichneten Prozesse) die relevanten Umstände herausfindet.

Dabei geht Delta Debugging ähnlich wie binäre Suche vor: Zuerst wird eine Hälfte der Umstände weggelassen. Führt das bereits zum Fehlschlagen, wird mit dieser um die Hälfte reduzierten Umstände-Menge weitergearbeitet. Falls der Test jedoch nicht zum Fehlschlagen führt, wird die andere Hälfte weggelassen. Reicht das ebenfalls nicht, werden Komplemente der Teilmengen gebildet und getestet. Führt das auch nicht zum Ziel, wird die Ursprungsmenge in mehr als zwei Teile zerlegt und der Algorithmus beginnt von vorne.

Man kann dabei zeigen, dass das Ergebnis des Algorithmus nur Umstände enthält, die für das Fehlschlagen relevant sind. Gibt es anfangs n Umstände, die zu minimieren sind, benötigt Delta Debugging im schlechtesten Fall dazu $O(n^2)$ Tests.

4 Erste Ergebnisse

Um unseren Prototyp zu testen, haben wir einen Netzwerkserver geschrieben, der ein von uns eigens eingebrachtes Sicherheitsloch enthält: Schickt man eine besonders präparierte Nachricht, entsteht eine Datei */tmp/pwned* mit Administrator-Rechten. In einem simulierten Angriff haben wir unter 30 Anfragen eine versteckt, welche die Lücke ausnutzt.

Durch den Angriff entstanden etwa 1.500 Systemaufrufe, die vom Originalsystem in etwa 6 Sekunden abgearbeitet und aufgezeichnet wurden. Das ist eine Verlangsamung um 8% bezogen auf die Verarbeitungsgeschwindigkeit ohne Aufzeichnung. Die Bearbeitung und Aufzeichnung findet auf einer virtuellen Maschine statt, um die Wiederholung zu vereinfachen. Rechnet man den Aufwand für die Virtualisierung noch heraus, ergibt sich eine Gesamtverlangsamung um 13% gegenüber einer dedizierten Maschine. Diese Performancezahlen bewegen sich dabei im Rahmen des Üblichen [1] und sind geeignet, um Malfor auf Produktivesystemen einzusetzen.

Malfor war in der Lage, in knapp drei Minuten und 14 Tests alle relevanten Prozesse (hier drei von 32) zu isolieren [4]. Dabei war die Wiederholung etwa um den Faktor 2 langsamer als die Aufzeichnung. Auch diese Zahlen zeigen die Eignung für den Produktivbetrieb.

5 Ausblick

An erster Stelle steht zunächst die Erweiterung von Malfor auf ein realistisches Beispiel. Dazu arbeiten wir bereits an der Wiederholung eines Angriffs auf Apache. Dieser Angriff ist so konstruiert, dass er ein neues Administrator-Konto hinzufügt, ohne aber die Passwortdatei zu öffnen. Damit können auch Werkzeuge wie BackTracker in die Irre geführt werden, die einen Angriff analysieren, indem sie durch die Auswertung von Systemaufrufen Beziehungen zwischen Prozessen herstellen [1, 3]: In diesem Angriff öffnet kein Prozess die Passwortdatei, dennoch ist sie am Schluss verändert.

Als nächstes wollen wir Malfor auf verteilte Systeme anwenden. Malfor ist bereits auf den Einsatz in verteilten Systemen ausgelegt, aber beim Aufzeichnen und beim Wiederholen müssen zeitliche Nebenbedingungen betrachtet werden, damit die Konsistenz des Gesamtsystems gewährleistet bleibt.

6 Abschluss

Wir haben mit Malfor ein System vorgestellt, das mit Hilfe experimenteller Methoden in der Lage ist, Systemeinbrüche automatisch zu analysieren. Es kann auf Produktivsystemen eingesetzt werden und eignet sich besonders für die Analyse gezielter Angriffe.

Literatur

[1] DUNLAP, GEORGE W., SAMUEL T. KING, SUKRU CINAR, MURTAZA A. BASRAI und PETER M. CHEN: *ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay*. In: *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Seiten 211–224, New York, NY, USA, Dezember 2002. ACM Press.

[2] HILDEBRANDT, RALF und ANDREAS ZELLER: *Simplifying and Isolating Failure-Inducing Input*. IEEE Transactions on Software Engineering, 26(2):183–200, Februar 2002.

[3] KING, SAMUEL T. und PETER M. CHEN: *Backtracking intrusions*. In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, Seiten 223–236, New York, NY, USA, 2003. ACM Press.

[4] NEUHAUS, STEPHAN und ANDREAS ZELLER: *Isolating Intrusions by Automatic Experiments*. In: *Proceedings of the 13th Annual Network and Distributed System Security Symposium*, Seiten 71–80, Reston, VA, USA, Februar 2006. Internet Society.