

Hauptseminar Malicious Code

Sandboxing

Megdich Mohamed Ali
megdich@informatik.tu-muenchen.de

Technische Universität München

Zusammenfassung Welche Bedeutung hat der Begriff „sandboxing“? Was ist eine Sandbox? Diese Themen sind hier, in der Ausarbeitung vom Vortrag „sandboxing“ im Rahmen des Hauptseminar Malicious Code behandelt. Es werden auch ein paar Beispiele von Sanboxing Techniken bei verschiedenen Kontexten kurz erklärt.

1 Einführung

Eine „sandbox“ ist eine vom eigentlichen System abgeschottete Umgebung. Eine solche Umgebung kann sehr nützlich sein, wenn man den Zugriff auf das Host System beschränken will. Diese Technik ist in verschiedenen Kontexten sehr beliebt. Das Konzept bleibt aber immer das Selbe. Die mehr oder weniger verstärkte Isolierung vom richtigen Betriebssystem schränkt alle Schäden ein.

2 Kontexte der Sanboxing-Techniken

Sandboxing Applikationen sind reich in Ihrer Vielfalt. Das geht vom kleinen Modul das in der Software eingebaut ist (*built-in*) bis zur Kompletten Software die ganze Betriebssysteme emulieren kann. Um ein paar Beispiele zu nennen, werde ich folgende Punkte vorstellen:

Man bezeichnet zum Beispiel sandboxing bei der Software Installation die Technik bei der die Programme erstmal einmal in einem geschützten Bereich des Systems installiert beziehungsweise kompiliert werden. Erst nachdem man sicher ist das bei diesem Vorgang keine Probleme aufgetreten sind, werden die Dateien in den richtigen Verzeichnisse kopiert. So bleibt das System ohne Schäden wenn zum Beispiel ein Fehler bei der Kompilierung stattfinden würde. Der normale Betrieb wird daher nicht gestört.

Virtuelle Maschinen Können eine komplette Laufzeitumgebung und sogar auch einen kompletten Betriebssystem emulieren. Das ist sehr hilfreich bei der Erhöhung der Userlevel Sicherheit. Da der Benutzer dann tun und lassen darf was er will ohne die Sicherheit des Host *container* System zu beeinträchtigen. Kommerziell werde mehrere Lösungen angeboten. Die bekanntesten sind zum Beispiel VMWare[6] oder VirtualPC von Micorsoft[2]. Die JVM (*Java Virtual Machine*) ist auch eine Virtuelle Machine bietet aber keinen Kompletten Betriebssystem zur Verfügung, sondern nur eine Plattform zum Ausführen von JAVA Programme. Sie stellt einen Mechanismus dar, das verhindert, dass die geladene Applets, Schaden auf den Lokalen Rechner einrichten können. Da die Programme dann in einer geschützten Laufzeitumgebung laufen, kann nicht vieles Zerstört werden aber das ist manchmal viel zu eingeschränkt dass, die Programme genug Rechte haben, um ihre Aufgaben zu erfüllen.

Eine Sandboxing Schicht könnte auch im Betriebssystem Kernel implementiert sein. Das ist aber durchaus schwer zu *maintainen*, weil man dann immer einen Administrator braucht der den Systemkernel nochmal kompiliert wenn man einen Update machen wollte. Nicht zu Schweigen dass, das Modifikationen an den Kernel sehr gefährlich sind und damit die Chancen erhöhen dass das System instabil wird.

3 Potenzielle Gefahren

Ein Browser, zum Beispiel, braucht andere Hilfsprogramme um richtig zu funktionieren. Bei diesen kann es sich um eine JPEG, GIF oder auch einen PostScript Viewer handeln. Das letztere ist es eine komplette Programmiersprache mit vollem Zugriff auf das Dateisystem. Die Daten, die damit behandelt werden, sind meistens aus dem Internet und es ist daher nicht selbstverständlich, dass sie vertrauenswürdig beziehungsweise sicher sind.

Die Topologie des Internet hat sich, in den letzten Jahren, sehr viel verändert. In der Anfangszeit war der Austausch von Informationen so gut wie unverfälscht und vertrauenswürdig. Im Gegensatz zu damals, als das weltweite Netzwerk ausschließlich von kooperierenden Wissenschaftlern bevölkert war, hat es sich zu einer sehr heterogenen Landschaft weiterentwickelt. Da aber die Sicherheit nicht im gleichen Maße wie der Informationsaustausch weiterentwickelt wurde, ist es heutzutage umso wahrscheinlicher, auf nicht vertrauenswürdige Dateien zu stoßen.

Wenn ein Benutzer zum Beispiel auf ein Dokument, das vom Browser automatisch mit Ghostscript bearbeitet wird, herunterlädt, ist es nicht auszuschließen, dass ein Dritter dieses manipuliert hat. Wenn dies der Fall ist, kann dieser Zugriff auf das komplette Dateisystem haben. Hacker können auch potenzielle Fehler ausnutzen, um Schaden anzurichten. Da die Software, die sie sich zu Nutzen machen, aber oftmals sehr mächtig ist, kann niemals vorausgesagt werden, welches Ausmaß der Schaden annehmen wird. Schlimmer ist jedoch, dass in demselben Maße wie die Komplexität der Software zunimmt leider auch die Anfälligkeit derselben wahrscheinlicher wird.

Da viele der benutzten Hilfsprogramme ursprünglich nicht nur als Hilfsprogramme konzipiert wurden, sondern in einen anderen Kontext, sind sie viel zu wenig eingeschränkt. Es ist nicht gedacht das die Daten mit die sie umgehen müssen aus dem Internet kommen und korrumpiert sein können.

4 Historisches Beispiel: Hydra

Als erstes Beispiel werde ich einer der ersten Sandbox Modelle nennen und zwar Hydra (1975). Das Modell der Hydra ist recht flexibel und ist hat als Grundprinzip die Trennung zwischen die *Policy* (die Grundregeln) und den *Mechanism* (den Mechanismus). Bei der User-level Software werden die Policies definiert die vom Kernel verwirklicht werden. Das Konzept der Hydra baut auf folgenden Punkten auf:

- Es ist leichter Information zu schützen wenn sie auf verschiedenen Objekte zerlegt ist.

- Jedes Objekt wird ein Typ zugewiesen. Manche Typen sind schon von Hydra gegeben (Prozedur oder Prozess). Neue Typen können auch mithilfe macher Mechanismen erstellt werden. Hydra gibt auch allgemeine Operationen damit diese Objekte bearbeitet können.
- Objekte habe keine Besitzer. Die Zugriffe auf den Objekte werden definiert.
- Jedes Programm hat genau soviel Macht wie viel es braucht, NICHT mehr.
- Bei der Hydra wird die Repräsentation der Operationen jeder Objekttyp in Modulen abstrahiert die *Subsystems* genannt werden. Der Benutzer kann nur ein Objekt durch die richtige Subsystem Prozeduren ansprechen.

Hydra ist interessant in verschiedenen Sichten wie Sicherheit oder Sandboxing das es sich um einen früheren Konzept handelt.

5 FreeBSD Jails

5.1 Einführung

FreeBSD[1] ist das am weitesten verbreitete Derivat der BSD Familie (NetBSD[3], OpenBSD[5]). Es hat den Ruf sehr stabil zu sein und wird deshalb vor allem im Server-Bereich eingesetzt. Laut Netcraft[4] laufen derzeit zirka 25 Millionen Websites auf FreeBSD Systemen. FreeBSD besticht, entgegen der weit verbreiteten Meinung, auch durch seine gute Hardware-Unterstützung. Ebenfalls beeindruckend ist der Umfang der Ports-Kollektion. Derzeit sind über 11000 Programme Dritter in den Ports verfügbar.

5.2 Jails

Der Jail-Systemruf „chroot“et eine gesamte Umgebung und setzt diverse Restriktionen auf einen Prozess und dessen Kindprozesse. Ein so eingesperrter Prozess kann auf Prozesse außerhalb seines „Gefängnisses“in *keinster* Weise zugreifen. Jede *gejailte* Umgebung besitzt ihre eigene IP-Adresse. Ein Jail kann im einfachsten Fall einen Prozess/Dienst enthalten. Der Normalfall ist jedoch, daß ein „gesamtes“virtuelles Betriebssystem *gejailt* wird. Jails sind also keine echten virtuellen Maschinen. Das heißt es läuft auf dem System nur ein Kernel. Dies hat den Vorteil das da keine Emulation virtueller Geräte usw. nötig ist. Somit ist der Jail-Mechanismus anderen Realisierungen dieser Art in Sachen Performance stark überlegen. Kurz gesagt ist man ist mittels Jail-Mechanismus in der Lage mehrere virtuelle Systeme auf einem Host zu fahren und somit die Sicherheit wesentlich zu erhöhen. Im Falle *böswilliger* Aktivitäten bleiben diese auf die jeweilige Jail begrenzt. Jails sind ein Feature des Basissystems und existieren seit FreeBSD 4.0.

5.3 Hilfsmittel

Der Jail-Mechanismus besteht aus zwei Teilen: dem User-Space Programm jail(8), dem Systemruf jail(2) und Code der an diversen Stellen im Kernel implementiert ist, um *gejailte* Prozesse an der Ausführung sicherheitsrelevanter Systemrufe zu hindern.

jail(8): Hier passiert nichts spektakuläres. Die Parameter werden verarbeitet und der jail(2)-Systemruf abgesetzt. Anschließend wird mittels execv(2)-Systemruf der Code des übergebenen Arguments ausgeführt. Aufzurufen ist das Binary jail(8) folgendermaßen.:

```
#> jail /usr/jails/j_www wwwserver 192.168.0.103 /bin/sh
```

Wobei das erste Argument das Root-Verzeichnis der Jail ist. Danach kommt der Hostname der Jail und die IP an welche die gejaillte Umgebung gebunden werden soll. Das letzte Argument ist der auszuführende Prozess. Würde man das Kommando so absetzen, hätte man eine Shell gejaillt.

Das jail(2) Systemaufruf: Hier alles zu erklären würde zu weit gehen. Wichtig ist aber, dass außer dem Systemruf selbst, Jail-Code noch an einigen anderen Stellen des Kernels auftritt. Und zwar wird in sicherheitsrelevanten Systemrufen geprüft ob der rufenden Prozess einem Jail angehört oder nicht. Ist dies der Fall wird dieser Systemruf nicht ausgeführt und ein Fehler zurückgegeben.

5.4 Einschränkungen in den Jails

Folgende Aufzählung gibt einen Überblick über die Einschränkungen innerhalb einer Jail.

- Kernel lässt sich nicht modifizieren.
- Netzwerkkonfiguration, Routing-Tabellen usw. sind unveränderlich.
- Kein mount bzw. umount möglich
- Kein anlegen von Gerätedateien möglich.
- Keine Raw-Sockets (somit kein ping, ettercap, nmap ...).
- Herunterfahren Jail-intern nicht möglich (halt(8), reboot(8), shutdown(8)).
- System V IPC (*Inter-Process Communication*) ist nicht möglich.

Durch diese Einschränkungen ist selbst der Superuser (im Jail) derart beschnitten, daß er nicht in der Lage sein sollte ausbrechen oder Schaden am Hostsystem bzw. in anderen Jails anzurichten. Das kann sehr hilfreich wenn normale Benutzer Buffer-Overflows ausnutzen um root rechte zu erlangen.

5.5 Die Problematischen Dienste

Bedingt durch die Einschränkungen innerhalb einer Jail-Umgebung, vor allem aber durch das Verbot von System V IPC, sind einige Dienste nicht für den Betrieb in Jails geeignet. Dazu gehören NFS und Samba. Dies sollte aber kein Problem darstellen, da diese beiden Dienste in der Regel nur in internen, also vertrauenswürdigen Netzen zum Einsatz kommen sollten.

5.6 Fazit

Der Jail-Mechanismus ist eine mächtige Form des Sandboxing und nützliches Werkzeug um unter anderem folgende Dinge zu realisieren:

- Aufbau virtueller Server, Aufbau einer DMZ.
- Trennung administrativer Aufgaben.
- Aufbau virtueller Test/Entwicklungs-Systeme.
- Anbieten von (Root-)Shells.

Ein wichtiger Pluspunkt gegenüber Lösungen in anderen freien Betriebssystemen ist, daß Jails zum FreeBSD Basissystem gehören. Somit gestaltet sich die Installation und Aktualisierung einfacher. Umständliche Maßnahmen wie Kernel patches usw. entfallen.

6 Java und die Sandkästchen

Wie oben erwähnt koennen Java Programme und besonders Java Applets, die aus nicht vertrauenswürdigen Quellen kommen, theoretisch eine echte Gefahr für einen System darstellen da Java eine mächtige Programmiersprache ist und ohne Einschränkungen kann man leicht viel Schaden einrichten. Das wäre durchaus im Bereich des Möglichen, wenn die Entwickler von Java keine mehrstufige Gegenmaßnahme gegen solche Angriffe entworfen hätten: Alle Java-Applets laufen in einer Sandbox, was ihnen nur eingeschränkten Zugriff auf lokale Ressourcen ermöglicht. Beispielsweise können Java-Applets Text auf Ihrem Bildschirm ausgeben, aber keine Daten von Ihrem lokalen Dateisystem lesen oder gar darauf schreiben, wenn Sie das nicht ausdrücklich erlauben. Dieses Sandkasten-Paradigma schränkt zwar die Nützlichkeit von Applets ein, erhöht aber die Sicherheit Ihrer Daten. Dies wurde dann in den neueren Java-Versionen immer weiterentwickelt dass es auch mehr Flexibilität hat. So könnte man selbst bestimmen, wieviel Sicherheit man benötigen, und sich so auf Kosten der Sicherheit zusätzliche Flexibilität erkaufen.

6.1 JDK 1.0

Einen sehr einfachen Ansatz verfolgte das JDK1.0: es teilte die Welt ein in die eigene Festplatte, der man vertraute, und die *böse weite Welt*. Java-Code, der über das Netz geladen wurde (im JDK1.0 waren das nur Applets), hatte nur sehr begrenzte Zugriffsmöglichkeiten auf den lokalen Rechner. Das Programm lief gewissermaßen in einer "sandbox", in der es nichts zerstören konnten. Die Sicherheit wurde durch Compiler, "Bytecode Verifier", "Class Loader" und "Security Manager" garantiert. Dabei gäbe es durchaus kleinere Probleme, aber im Großen und Ganzen kann man behaupten, dass sich dieses Modell für bestimmte Anwendungsfälle zwar bewährt hat, aber mit den gewichtigen Einschränkungen konnte man mit Applets wenig anzufangen.

Selbst im Intranet mit ausreichender Bandbreite waren Applets weitgehend nutzlos, da zum Beispiel jeder Zugriff auf die lokale Festplatte genauso unmöglich war wie der Zugriff auf andere Server-Rechner als den Web-Server. Damit war eine typische Client-Server-Anwendung, die den Client (das Applet) vom Intranet-Server lädt und welcher dann zu einem Datenbank-Server Kontakt aufnimmt, schwierig zu realisieren. Es gab natürlich einen Ausweg: den Class Loader und den Security Manager selbst zu implementieren. Aber das war eher eine theoretische Möglichkeit.

6.2 Größere Sandkästen in den nächsten Versionen

In der nächsten JDK-Variante wurde das Sandbox-Modell erweitert: verwendete ein Applet eine *zertifizierte* JAR-Datei, so wurde dem Code genauso getraut, als würde er lokal von der Platte geladen. Dumm an der ganzen Sache war nur, dass die beiden Firmen Netscape und Microsoft wieder eigene, inkompatible Verfahren entwickelten - mit kostenpflichtigen Zertifikaten. Also aus der Traum, in der Firma mal eben schnell einen Sys-Admin als Zertifizierungsstelle zu definieren. Außerdem ist auch mit diesem Modell eine Alles-oder-Nichts Situation vorhanden: das Applet darf fast nichts oder alles. Mehr war nicht drin.

Das JDK1.2 bietet hier endlich eine überzeugende Lösung. Das Sicherheitsmodell ist beliebig fein strukturierbar, einfach zu konfigurieren, erweiterbar und auf alle Programme (also auch lokalen Code) anwendbar. Zentraler Begriff hierbei ist die "Protection Domain". Eine Domain umfasst konzeptionell eine Gruppe von Klassen mit gleichen Rechten. Typischerweise unterscheidet man System- und Application-Domains. Netzwerk-I/O, Tastatur- und Graphik-Ausgabe sind Beispiele für System-Domains. Eine Klasse wird gelegentlich eine Methode aus einer anderen Domain aufrufen und das Security-System muss sicherstellen, dass auf diese Weise keine unerlaubten Rechte erworben werden. Implementiert wird der Zugriff über Sub-Klassen von `java.security.Permission` beziehungsweise `java.security.BasicPermission`. Den Zugriff auf das lokale Filesystem regelt etwa die Klasse `java.io.FilePermission`. Hiermit läßt sich steuern, auf welchen Teil des Datei-Systems welche Rechte (lesen, schreiben etc.) vergeben werden. Alle zum aktuellen Zeitpunkt gültigen Permissions sind in einem "Singleton Object" (Objekt mit nur einem Member) der Klasse `java.security.Policy` zusammengefasst (genauer gesagt in einer implementierenden Sub-Klasse, da `Policy` selbst abstrakt ist).

6.3 Fazit

Ganz im Gegensatz zu ActiveX und anderen Techniken verfügt Java über ein eingebautes Sicherheitskonzept und läuft auf einer eigenen VMachine (*Virtuelle Maschine*). Die Applets laufen in der hermetisch abriegelten Sandbox ab und so sicherstellen dass das Programm nicht irgendwelche Systemschäden anrichten kann.

7 Das Ende

Das Konzept der Sandästen ist simple und beruht auf dem Folgenden Prinzip:

** Was man nicht sehen bzw. berühren kann, kann man auch keinen Schaden zufügen.*

Das hat sich auch im Laufe der Jahre bewert.

Literatur

1. FREEBSD: <http://www.freebsd.org>.
2. MICOROSOT: <http://www.microsoft.com>.
3. NETBSD: <http://www.netbsd.org>.
4. NETCRAFT: <http://www.netcraft.com>.
5. OPENBSD: <http://www.openbsd.org>.
6. VMWARE: <http://www.vmware.com>.