Bundesamt
für Sicherheit in der
Informationstechnik

Whitepaper

Java IDS-

Intrusion Detection für die Java-Laufzeitumgebung

Intrusion-Detection-Systeme (IDS) sind ein verbreitetes und bewährtes Mittel zur Erkennung sicherheitsrelevanter Anomalien in IT-Systemen. Insbesondere im Serverbereich kommen solche Systeme zur Anwendung, um eine bestmögliche Detektion sowie eine anschließende Reaktion auf Sicherheitsvorkommnisse zu gewährleisten.

Dieser Beitrag stellt ein Konzept zur Umsetzung eines speziell für Java geeigneten IDS vor, welches sich sowohl zur Absicherung von Client- als auch von Serversystemen eignet. Die Realisierung erfolgt durch die Integration geeigneter IDS-Komponenten in ein bekanntes IDS, welches unter einer freien Lizenz verfügbar ist.

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
Tel.: +49 228 99 9582-5599
E-Mail: java@bsi.bund.de
Internet: http://www.bsi.bund.de

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Sichere Anwendungen auf Basis von Java

Die Java-Virtual-Machine (Java-VM) [JAV] hat als Infrastruktur zur Ausführung von plattformunabhängigen Anwendungen bei Behörden und Unternehmen sowohl für Endanwender als auch im Serverumfeld weite Verbreitung erlangt. Um den sicheren Betrieb von Anwendungen zu gewährleisten, bringt die Java-Plattform durch die Sprachspezifikation und die interne Architektur bereits viele Sicherheitsmerkmale mit. Zu nennen sind hier beispielsweise Typensicherheit, Codeüberprüfung und Zugriffsschutzmechanismen.

2. Intrusion-Detection-Systeme

Ein Intrusion-Detection-System (IDS) dient der Erkennung von sicherheitsrelevanten Anomalien in IT-Systemen und Computernetzen. Der Einsatz eines IDS stellt eine sinnvolle Ergänzung zu einem bestehenden Sicherheitskonzept dar. Derzeit existiert jedoch kein IDS, welches interne Informationen aus der Java-VM zur Einbruchserkennung nutzt.

Prinzipiell werden IDS nach netzwerkbasierten (NIDS) und hostbasierten Intrusion-Detection-Systemen (HIDS) kategorisiert. NIDS dienen der Angriffserkennung in Netzwerken, in dem z.B. die transportierten Daten auf bekannte Angriffsmuster untersucht werden. HIDS hingegen verwenden unterschiedliche System- und Performanzmessgrößen, um diese auf bekannte Muster hin zu analysieren. Die so genannten hybriden IDS kombinieren den Ansatz von NIDS und HIDS.

Die Funktionsweise eines IDS stellt sich als wiederholtes Durchlaufen der folgenden Teilschritte dar:

1. **Informationssammlung**: Die unterschiedlichen Parameter werden durch sogenannte Sensoren aus Kenngrößen oder Nutzdaten eines IT-Systems gesammelt und an eine zentrale Stelle (IDS-Manager) übermittelt. Die Informationssammlung kann sowohl in Intervallen als auch auf Basis von Ereignissen erfolgen.
2. **Datenanalyse**: Der IDS-Manager sammelt die Daten aller Sensoren und wertet diese unter Anwendung unterschiedlicher Analysemethoden aus.
3. **Alarmierung/Reaktion**: In Abhängigkeit der Ergebnisse der Datenanalyse werden zuvor definierte Eskalationsschritte zur Alarmierung bzw. der Reaktion auf (vermeidliche) Sicherheitsvorfälle initiiert.

Sofern in Schritt 3 aktive Gegenmaßnahmen ergriffen werden, spricht man statt von IDS oftmals auch von Intrusion-Prevention-Systemen (IPS). Diese können Angriffe abwehren bzw. auf Auffälligkeiten oder auf konkrete Angriffe aktiv reagieren.

Eines der bekanntesten IDS ist Prelude [PRE], welches unter Open-Source-Lizenz zur Verfügung steht. Hierbei handelt es sich um ein hybrides IDS, das für eine Vielzahl unterschiedlicher Plattformen verfügbar ist. Darüber hinaus sind eine große Anzahl unterschiedlicher Sensoren verfügbar, die kompatibel zu Prelude sind. Dies wird dadurch gewährleistet, dass Prelude den offenen Standard Intrusion-Detection-Message-Exchange-Format (IDMEF) implementiert und somit eine offene und fest definierte Schnittstelle bereitstellt.

Weitere Informationen zur Sicherheit von IDS finden sich unter anderem in der IDS-Studie des BSI [BSI].

3. Konzept für ein Java-spezifisches IDS

Dieser Abschnitt beschreibt die Architektur sowie geeignete Standards und Technologien zur Umsetzung eines Intrusion-Detection-Systems, welches den Anforderungen an die Einbruchserkennung innerhalb einer Java-Virtual-Machine (JVM) gerecht wird.

3.1 Architektur

Das modular aufgebaute Gesamtsystem besteht aus einer Auditkomponente zur Erkennung von Systemereignissen innerhalb der JVM und einer Detection-Komponente. Letztere dient der Analyse dieser Ereignisse und der Erkennung von Angriffen. Die Architektur des Java-IDS – dargestellt in Abbildung 1 – beinhaltet folgende wesentliche Bestandteile:

- Auditsubsystem zur Informationsgewinnung über Ereignisse der JVM,
- Spezifikation einer formalen Beschreibungssprache für Ereignisse,
- Analyse- und Erkennungssystem (Detectionsubsystem),
- Spezifikation einer formalen Beschreibungssprache für Bedrohungsszenarien.

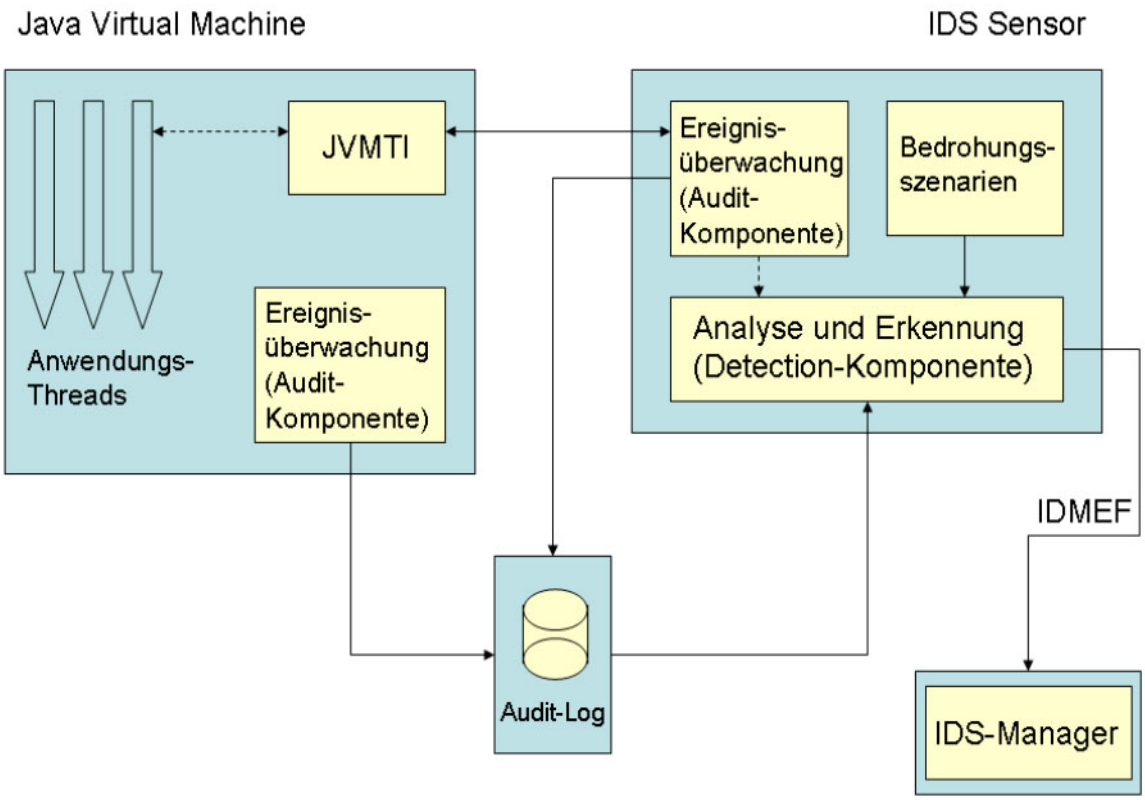


Abbildung 1: Architektur des Java IDS

Die Auditkomponente erfasst über Auditmodule die Ereignisse aus der Java-VM und leitet entsprechende Informationen zur Protokollierung an das Audit-Log weiter. Diese Daten dienen dann als Input für die Detection-Komponente. Zusätzlich enthält das Java-IDS ein Responsemodul, über das Reaktionen auf Angriffe an die Java-VM zurückgegeben werden können. Nach außen hin verhält sich das Java-IDS gegenüber einem IDS-Manager wie ein gewöhnlicher IDS-Sensor, der seine Daten in einer IDMEF-konformen XML-Struktur bereitstellt. Eine stärker technisch orientierte Darstellung der Gesamtarchitektur ist in Abbildung 2 zu sehen.

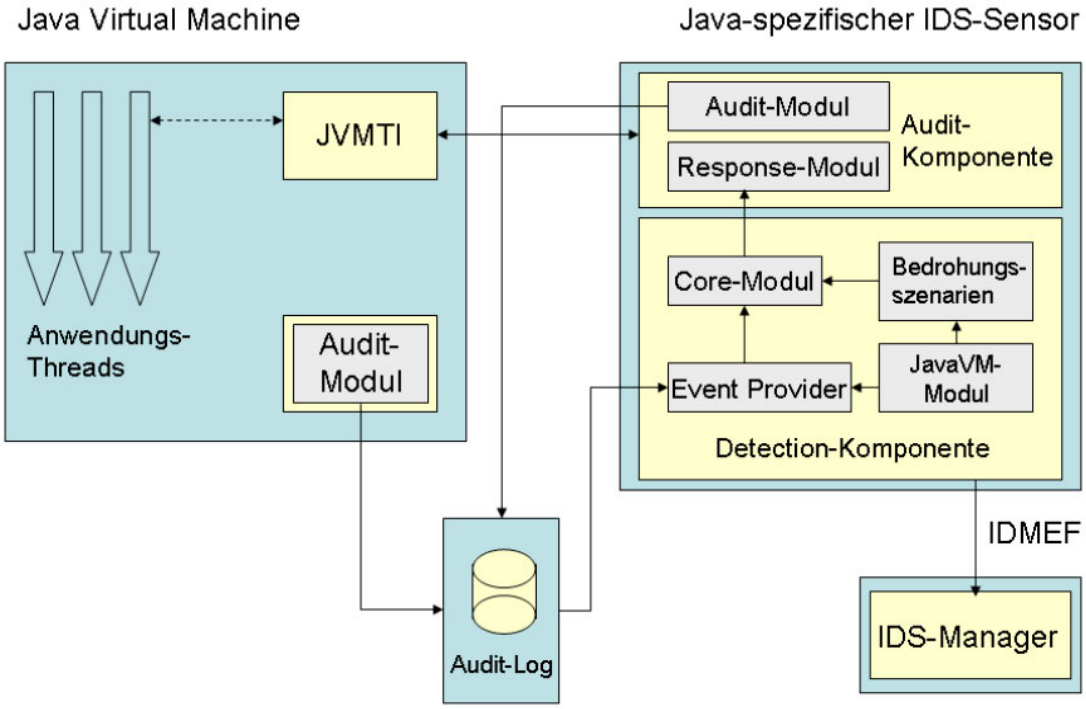


Abbildung 2: Architektur des Java IDS

3.2 Auditsubsystem

Das Auditsubsystem der Architektur des Java-IDS ist in zwei Teile unterteilt. Die interne Komponente läuft selbst in der JVM und gewinnt dort durch Instrumentierung bestimmter Klassen Informationen zu den laufenden Prozessen. Die externe Komponente greift von außen geeignete Informationen von der JVM ab. Somit entstehen zwei Möglichkeiten, Informationen aus der Java Laufzeitumgebung zu sammeln. Hierdurch können mitunter Redundanzen entstehen, die aber durchaus von Vorteil sein können (vgl. hierzu den Ansatz „Cross-View-based Detection“ aus dem Bereich Malware-Detection [CVD]).

Aufbau und Funktionsweise des Auditsubsystems sind in Abbildung 3 dargestellt. Sowohl die interne als auch die externe Teilkomponente des Auditsubsystems sammeln Ihre gewonnenen Daten in einem Audit-Log. Dieses bildet die Schnittstelle zwischen den Subsystemen Audit und Detection.

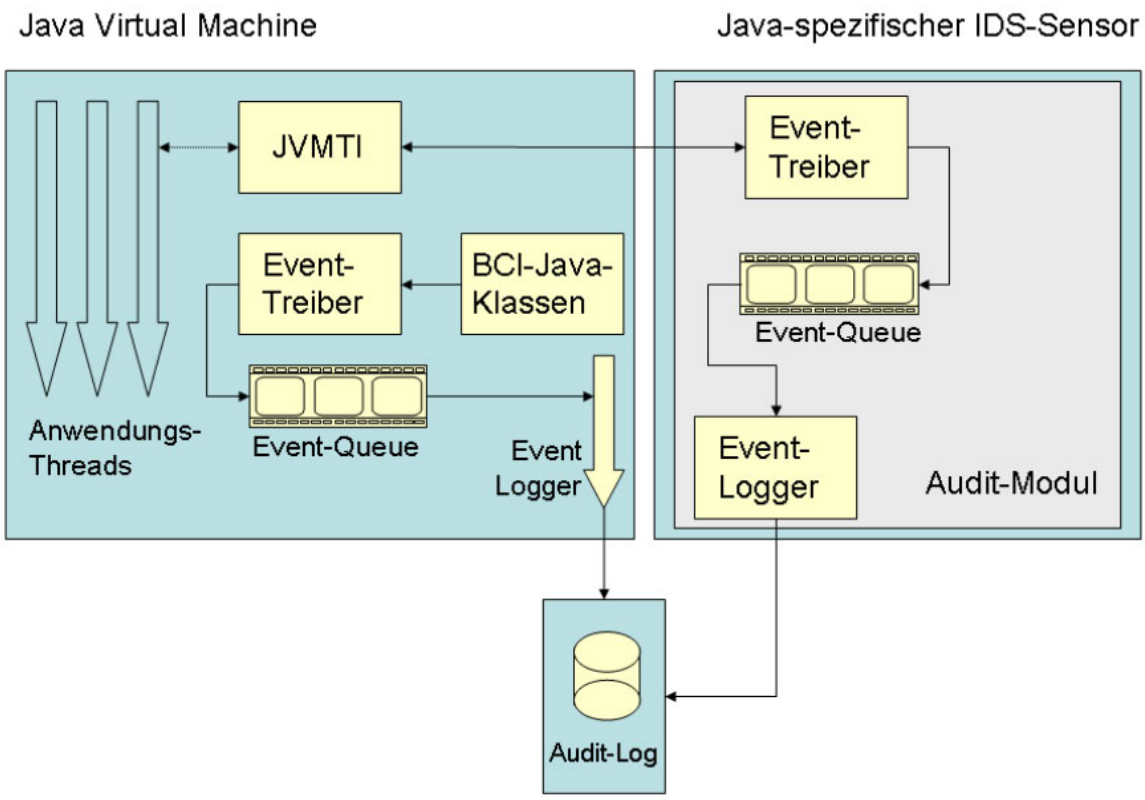


Abbildung 3: Architektur des Auditsubsystems

Die interne Auditteilkomponente verwendet die sog. Byte-Code-Instrumentation (BCI). Dabei werden bestimmte Java-Klassen mit Aufrufen eines Event-Treibers instrumentiert, sodass entsprechende Aktionen innerhalb der JVM direkt über einen Loggingmechanismus protokolliert werden können.

Das Java-Virtual-Machine-Tool-Interface (JVMTI) bildet die technische Grundlage für die externe Auditteilkomponente. JVMTI löst die zuvor in Java verfügbaren Schnittstellen JVMPI und JVMDI ab und stellt eine Schnittstelle für Entwicklungs- und Monitoring-Anwendungen bereit. JVMTI bietet die Möglichkeit zur Überwachung des Zustands sowie des Programmflusses von Anwendungen. Da die Verwendung bestimmter Ereignisse (MethodEntry, MethodExit) auf einigen Plattformen zu signifikanten Performanzverlusten führen kann, ist in Einzelfällen die Informationsgewinnung bestimmter Parameter durch BCI vorzuziehen.

3.3 Detectionsubsystem

Das Detectionsubsystem dient der Analyse der Daten, welche das Auditsubsystem im Audit-Log zusammenträgt. Die grundlegende Funktionsweise des Detectionsubsystems umfasst die Analyse von Zuständen der JVM und den Übergängen (Transitionen) zwischen diesen, um Bedrohungen bzw. Systemeinbrüche zu detektieren.

Die Detection-Komponente zur Erkennung von Angriffen setzt auf dem generischen STAT-Framework [STA] auf und erweitert es um spezifische Module zur Abbildung der besonderen Eigenheiten von Java. Mögliche Bedrohungsszenarien werden ebenso wie ein speziell an Java angepasster Eventprovider als ausführbare Erweiterungsmodule zum Framework hinzugefügt,

das dadurch seine spezielle Java-IDS-Funktionalität bekommt. Das Gesamtsystem wird als Sensor in ein IDS-Framework wie z.B. Prelude integriert. Abbildung 4 zeigt den grundlegenden Aufbau des Detectionsubsystems incl. der Entwicklungsumgebung für die Bedrohungsszenarien.

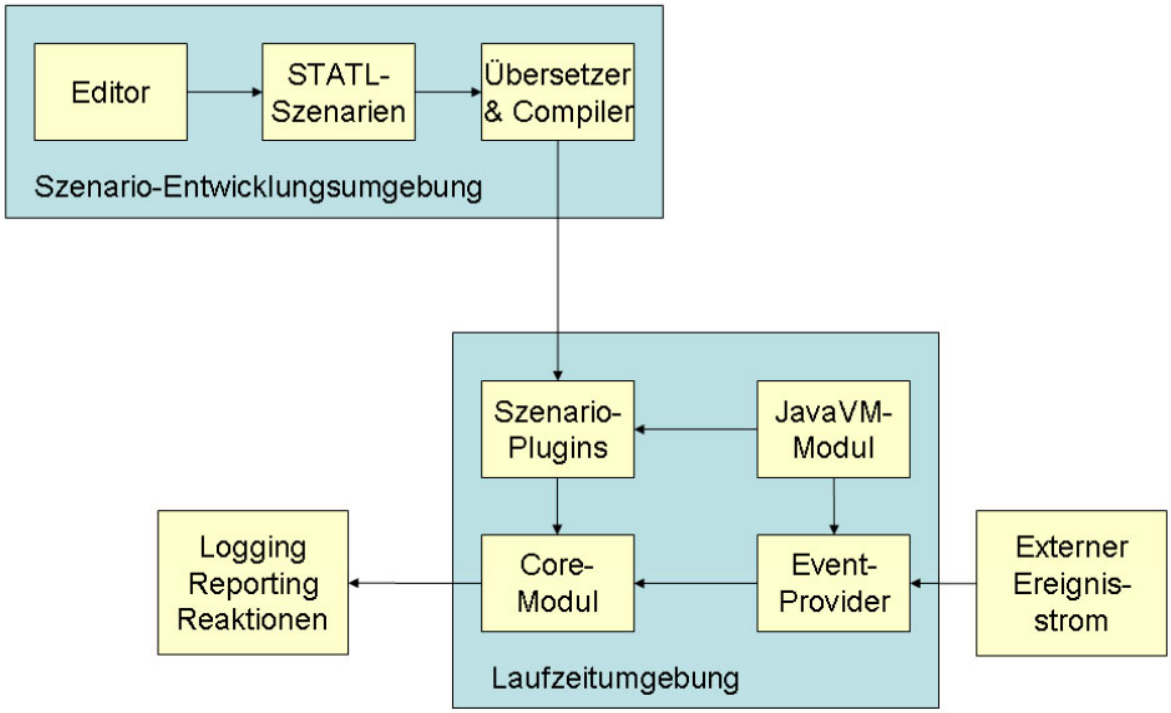


Abbildung 4: Architektur des Detectionsubsystems

Mit dem STAT-Framework (State-Transition-Analysis-Technique) sowie der zugehörigen STAT-Language (STATL) baut das Java-IDS auf einer systemunabhängigen Plattform für die Analyse und Erkennung von Angriffen auf IT-Systeme auf. Mittels STATL werden Angriffs- und Bedrohungsszenarien mit Mitteln der Graphen-Theorie modelliert. Die zentralen Begrifflichkeiten in diesem Zusammenhang sind Szenarien, Zustände und Transitionen (Übergänge). Ein einfaches Beispiel für eine solche Modellierung ist in Abbildung 5 dargestellt. Neben der Beschreibungssprache beinhaltet STAT insbesondere eine Laufzeitumgebung zur Verarbeitung und Analyse gemäß des STAT-Ansatzes. Daneben ist auch eine Architektur zur Erstellung und Übersetzung von Bedrohungsszenarien (Szenario-Plugins) Bestandteil von STAT.

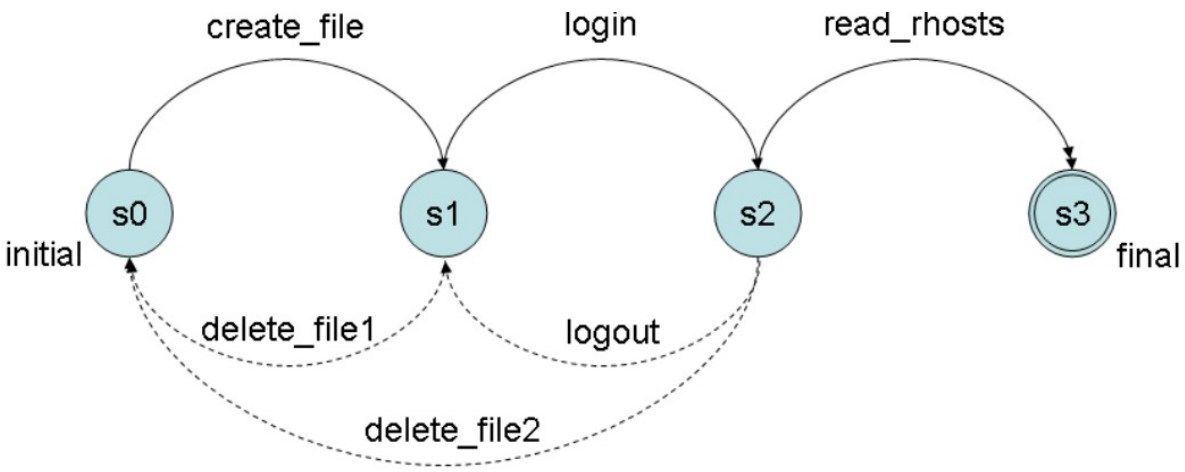
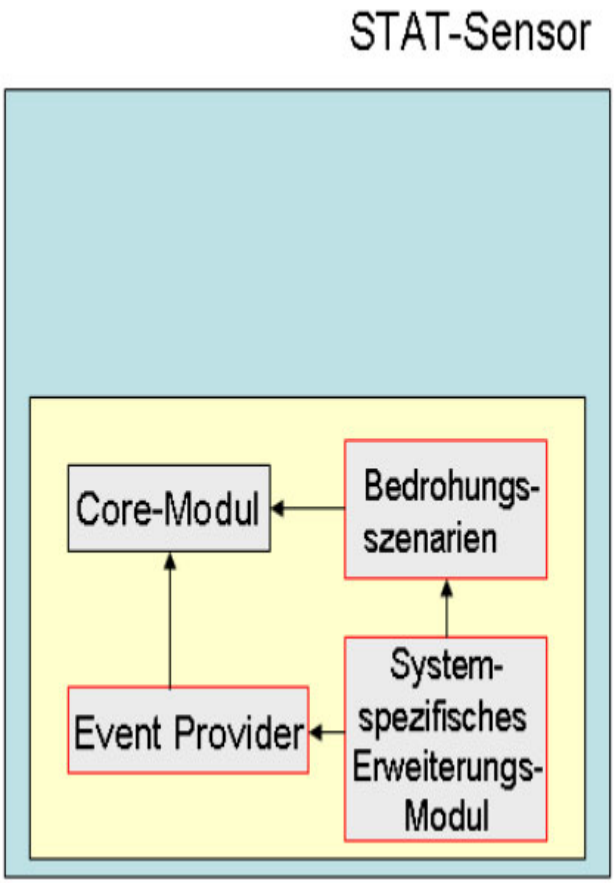


Abbildung 5: Beispiel für eine Modellierung mit Zuständen und Transitionen mit STAT



Abbilung 6: Schematische Darstellung eines STAT-Sensors

Abbildung 6 zeigt den Aufbau eines typischen STAT-Sensors, wovon folgende Komponenten für ein Java-spezifisches IDS zu entwickeln sind:

- Systemspezifisches Erweiterungsmodul (zur Definition JVM-spezifischer Typen und Ereignisse), genannt javastat,
  1. JVM-spezifischer Event-Typ
  2. Methoden zur Verwaltung der Ereignisse

- Event-Provider,
- Beschreibung zu detektierender Angriffssituationen in Form von Bedrohungsszenarien.

Als Sprache für die Java-IDS-interne Beschreibung von Ereignissen erscheint IDMEF etwas zu umfangreich. Daher wird ein alternativer Ansatz gewählt. IDMEF kommt nicht zwischen den IDS-Komponenten, sondern lediglich zur Anbindung an einen (externen) IDS-Manager zum Einsatz. Dies bewirkt eine effizientere Kommunikation zwischen den Komponenten des Java-IDS. Zu den Bestandteilen der intern verwendeten Ereignisbeschreibung gehören unter anderem folgende Aspekte:

1. Timestamp: Wann ist das Ereignis aufgetreten?
2. Source: Welche Anwendung oder welcher Anwendungs-Thread hat das Ereignis ausgelöst?
3. Action: Welche Aktion wurde durchgeführt (bzw. sollte durchgeführt werden)?
4. Target: Was war das Zielobjekt?
5. Result: Welches Ergebnis hatte die Aktion?
6. Type: eine Klassifizierung des Ereignisses.

Diese Informationen sind z.B. wie folgt dargestellt in eine entsprechende XML-basierte Beschreibungssprache zu integrieren.

```
<event>
    <timestamp>
        <year>2008</year>
        <month>01</month>
        <day>30</day>
        <hour>08</hour>
        <minute>05</minute>
        <second>12</>
     </timestamp>
     <type>dnssuspect</type>
    <source>
        <thread>
            <id>7925</id>
        </thread>
    </source>
    <action>
        <type>ThreadOverflow</type>
    </action>
    <target>
        de.target.sample.classname
    </target>
    <result>
        <status>TerminatedByIds</status>
        <returncode>3</returncode>
    </result>
</event>
```

Das Response-Modul des Detection-Subsystems wandelt diese Informationen in das Format IDMEF um und leitet sie anschließend an einen IDS-Manager weiter. Darüber hinaus ist eine geeignete Schnittstelle vorgesehen, um neben einem IDS auch ein Frühwarnsystem wie dem Open-Source-Projekt Nagios [NAG] mit Informationen über eine JVM versorgen zu können. Neben rein informativen Aktionen muss das Response-Modul auch in der Lage sein, als Reaktion auf einen Angriff Gegenmaßnahmen zu initiieren. Mögliche Gegenmaßnahmen sind z.B. das Terminieren eines Threads oder die dynamische Anpassung von Sicherheitsrichtlinien.

4. Fazit

Das Java-IDS stellt insgesamt einen deutlichen Sicherheitsvorteil für Java-basierte Client- oder Serveranwendungen dar. Dabei können sowohl Angriffe auf bekannte Schwachstellen als auch bislang unveröffentlichte Exploits aufgespürt werden. Neben der reinen Detektion kann auch eine geeignete Reaktion auf eine Auffälligkeit erfolgen, sodass das Java-IDS um eine umfangreiche Funktionalität zur Intrustion Prevention erweitert wird. Auch die Möglichkeit zur Integration in ein Frühwarnsystem macht das Java-IDS zu einer wertvollen Ergänzung für eine domänenweite oder auch domänenübergreifende IT-Sicherheitsarchitektur.

5. Referenzen

- [BSI] IDS-Studie, Bundesamt für Sicherheit in der Informationstechnik, http://www.bsi.bund.de/literat/studien/ids02/index.htm
- [CVD] Thoughts about Cross-View based Rootkit Detection, Joanna Rutkowska, http://invisiblethings.org/papers/crossview_detection_thoughts.pdf
- [JAV] Java, Sun Microsystems, http://java.sun.com
- [NAG] Nagios, http://www.nagios.org/
- [PRE] Prelude, http://www.prelude-ids.org/
- [STA] STAT Framework, http://www.cs.ucsb.edu/~seclab/projects/stat/index.html