Philipps Universität Marburg

controlware communicationssystems

Fraunhofer SIT

TECHNISCHE UNIVERSITÄT DARMSTADT

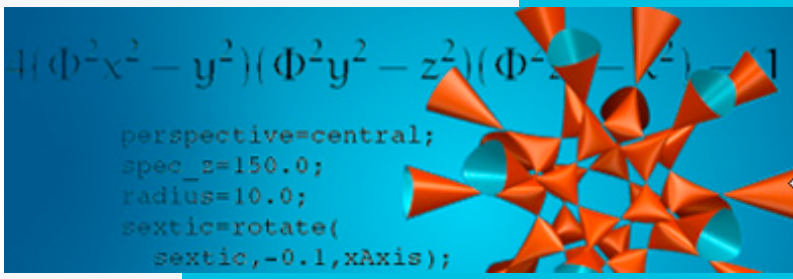
software AG

BMBF-Statustreffen IT-Sicherheitsforschung

ACCEPT

Bundesministerium für Bildung und Forschung

PROJECT OF Software-Cluster



Anomaliemanagement in Computersystemen durch Complex Event Processing Technologie

Bastian Hoßbach

Agenda

- Einführung
- Architektur
- Fallbeispiele
- Evaluation
- Ausblick

Most people spend more time and energy going around problems than trying to solve them.

— Henry Ford

EINFÜHRUNG

Problemstellung

- Aktuelle SIEM-Systeme („Security Information and Event Management“)
  - sind auf infrastrukturelle Ereignisse beschränkt
  - erfordern die Einbeziehung menschlicher Entscheider
  - haben oft eine sehr hohe Anzahl an falschen Alarmen
  - skalieren nicht mit einer sehr hohen Anzahl von Ereignissen

- In ACCEPT soll ein neuartiges SIEM-System dediziert für virtualisierte Rechnersysteme entwickelt werden, das die Nachteile aktueller SIEM-Systeme vermeidet.

Lösung: „Anomaly Management Inside“

- Virtualisiertes Rechnersystem mit “eingebauter Sicherheit”
  - Sensoren verteilt über alle Ebenen (A*pplication-, Operating System- und Hypervisor-Level*)
  - Sicherer Highspeed-Kommunikationskanal für Inter-VM Nachrichten
  - Neuartiges Analyse-Backend auf Basis von Complex Event Processing (CEP) zum Management von Anomalien in massiven Datenströmen in Echtzeit
  - Framework, um (Re-)Aktionen im System auszulösen

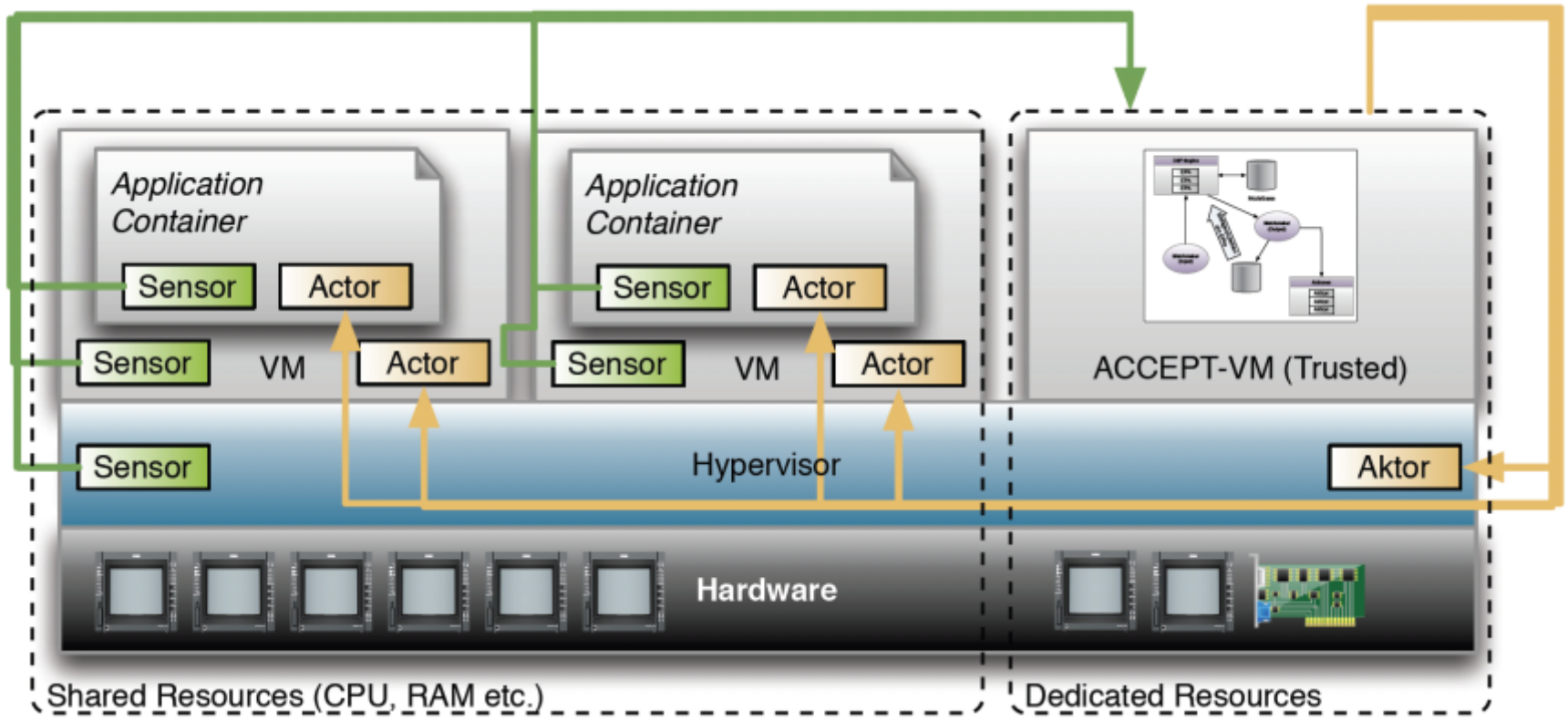
- Vorteile
  - Abstraktion, Korrelation und Aggregation der Ereignisse erfolgt horizontal in und vertikal über alle Virtualisierungsebenen
  - System kann sich dynamisch anpassen und falsche Alarme reduzieren
  - System kann selbständig Angriffe abwehren und potentielle Sicherheitslücken schließen

The user's going to pick dancing pigs over security every time.

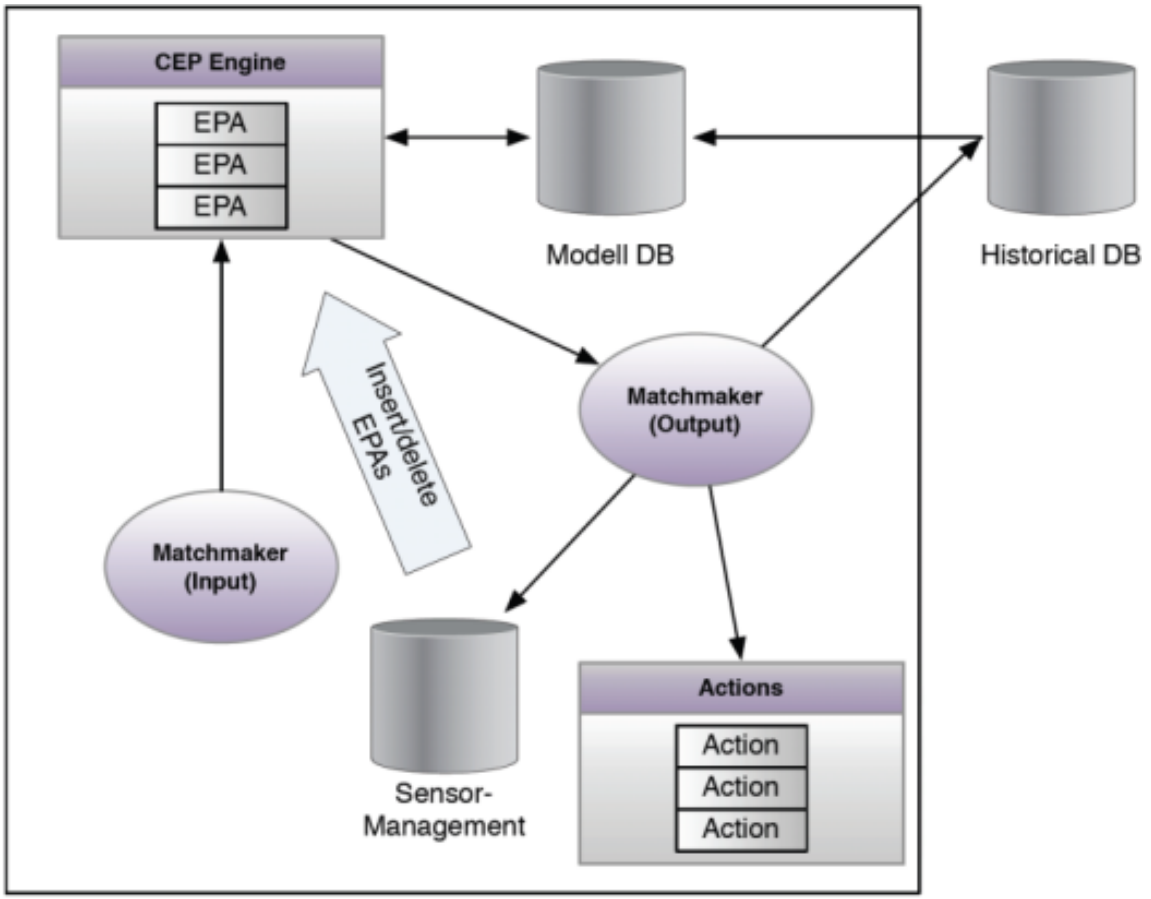
— Bruce Schneier

ARCHITEKTUR: EINGEBAUTE SICHERHEIT

ACCEPT Architektur: The Big Picture



ACCEPT Architektur: ACCEPT-VM



Kommunikationskanal

- Kommunikation zwischen Gast-Systemen und der ACCEPT-VM
- Vermeidung von TCP
  - Hoher Protokoll-Overhead
  - Erhöhte Sicherheit gegenüber Angriffen auf Kommunikationskanal
- Effizienter Kanal im Hypervisor zwischen paravirtuellen, seriellen Geräten durch gemeinsame Speicher mit Ringpuffer
- Verbindungsorientierte- und verbindungslose Kommunikation

Sensoren

- Sensoren auf allen Ebenen des virtuellen Systems
  - Hypervisor
  - Gast-System Kernel
  - Gast-System Userland
  - Gast-System (Java-)Applikationsschicht)

Sensoren
(Hypervisor)

- Sensoren nutzen das QEMU Monitor Protocol (QMP) für Anfragen und Befehle an den Hypervisor

- Ermöglicht alle Anfragen und Befehle des QEMU Hypervisors, u.a.:
  - Starten, Stoppen von virtuellen Maschinen
  - Memory Dumps
  - Trace Events
  - Netzwerkverbindungen

Sensoren
(Gast-Kernel)

- Sensoren realisiert als einzelne Linux Kernel-Module

- Monitoring von System-Calls durch spezielle Handler, u.a.:
  - open
  - close
  - fork
  - exec
  - read
  - write

Sensoren
(Gast-Userland)

- Sensoren zum Monitoring des Gast-Systems
- Nutzen von Schnittstellen und Tools des Betriebssystems
- Abfragen beliebiger Ressourcen durch angepasste Scripts, u.a.:
  - Laufende Prozesse
  - Geladene Kernel-Module
  - Angemeldete Benutzer und deren Gruppen und Berechtigungen
  - Sockets
  - Eingehängte Dateisysteme

Sensoren
(Gast-Userland, Java)

- Spezielle Java Sensoren zur Überwachung von JVMs

- Überwachung von Eigenschaften der virtuellen Umgebung
  - Heap-Größe
  - Geladene Klassen
  - Threads und deren Status

- Tracing von Events in Java Prozessen durch SUNs btrace
  - JDBC Anfragen
  - Runtime Events
  - Socket Events
  - WebService Events

Aktoren und Action Framework

- Integriert in allen Ebenen des virtuellen Systems
  - Hypervisor
  - Gast-Kernel
  - Gast-Userland

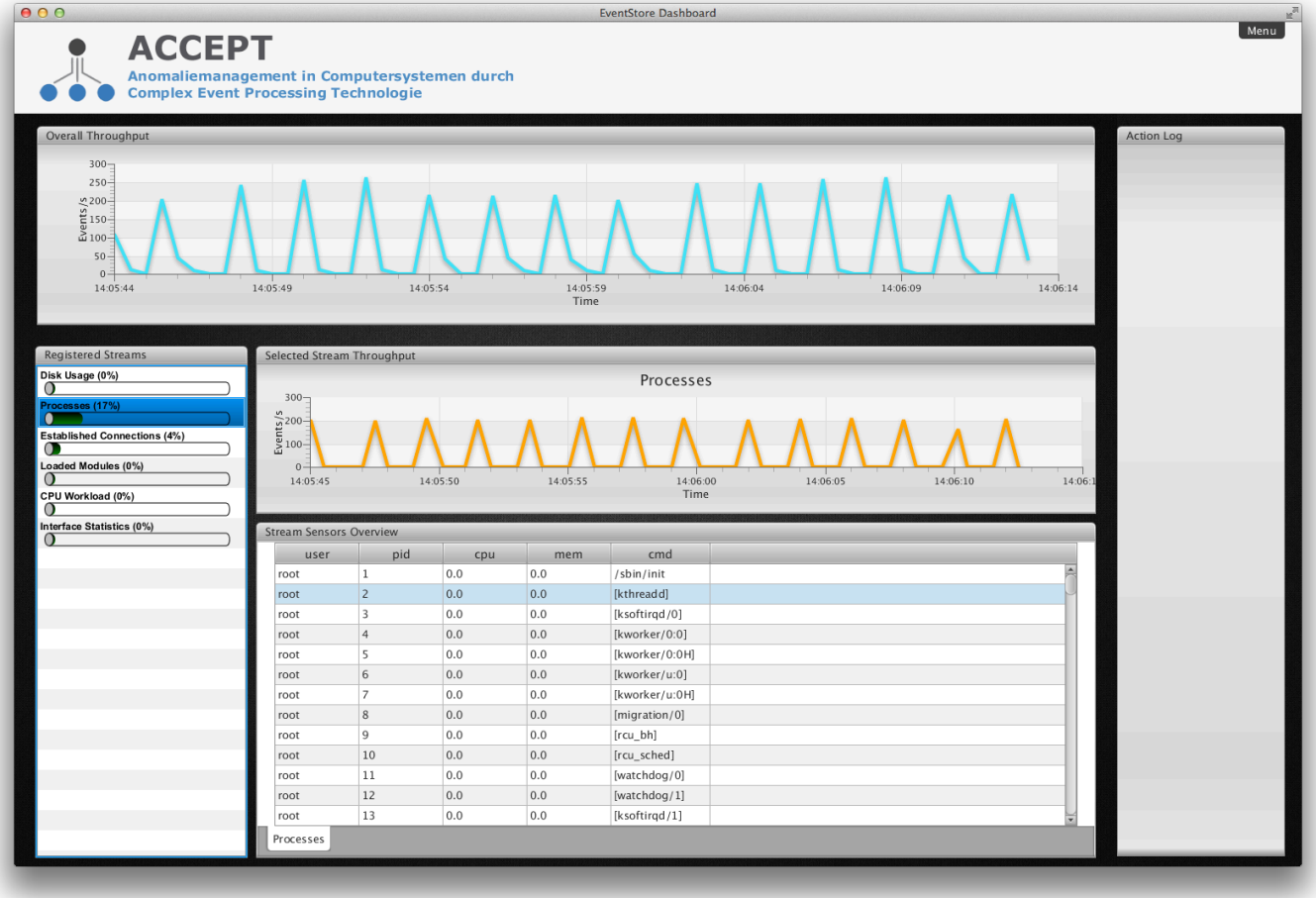
- Ermöglichen Event-basierte Ausführung von Skripten
  - Beenden von Prozessen
  - Herunterfahren der Maschine
  - Neustarten von Diensten
  - ...

- Stellen eine geschützte Umgebung bereit, in der signierte Python-Skripte ausgeführt werden

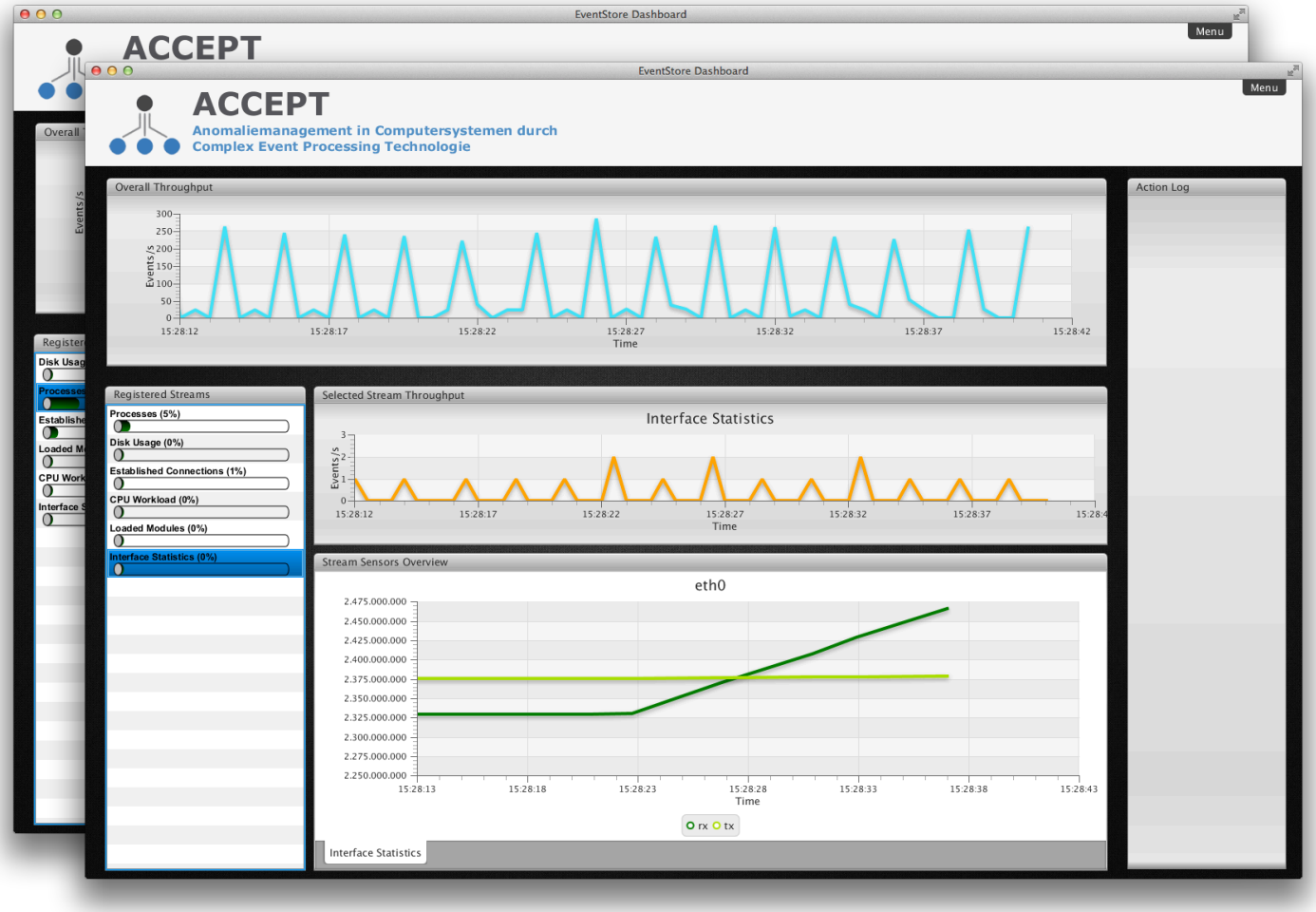
Aktuell implementierte Sensoren und Aktoren

- Sensoren gesamt: 60
  - Hypervisor: 4
  - Kernel: 7
  - Userland: 29 (regulär) + 6 (systemtap)
  - Java: 5 (regulär) + 9 (btrace)
- Aktoren gesamt: 6
  - Remote: 3
  - In-VM: 3

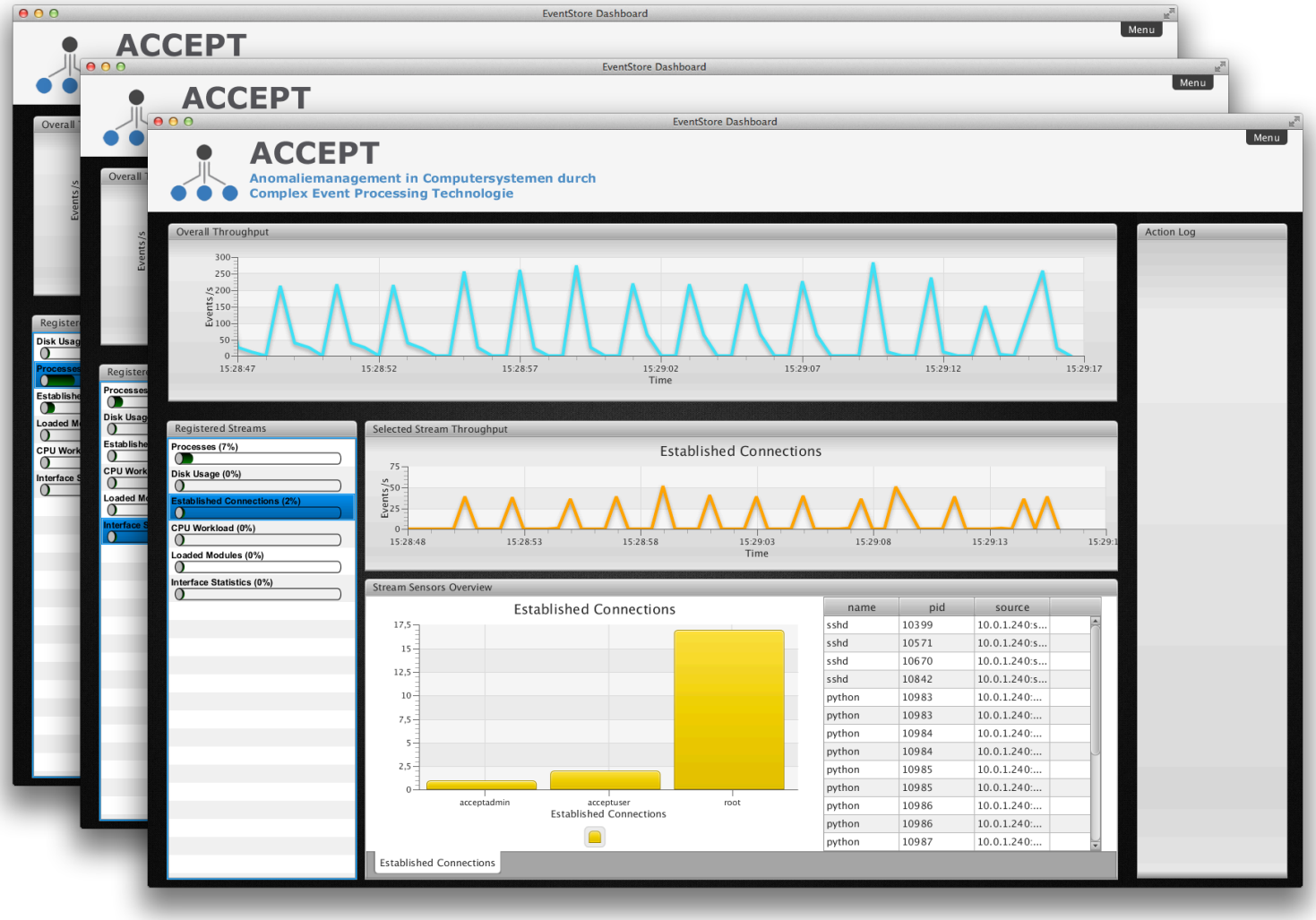
Sensoren in Aktion



Sensoren in Aktion



Sensoren in Aktion



CEP-Engine

- Ausführungsumgebung für Event Processing Agents

- Event Processing Agents (EPA)
  - Stehende Anfragen in Datenströmen
  - Korrelationen zwischen verschiedenen Datenströmen und damit auch Ebenen
  - Korrelationen zwischen Datenströmen und statischen Datenbanktabellen
  - Erkannte Anomalien werden sofort an das Action Framework berichtet

CEP-Engine: JEPC

- ACCEPT verwendet JEPC (Java Event Processing Connectivity)
- Kein Vendor lock-in
- Föderation von CEP-Systemen
- Dynamic CEP zum Management von Anomalien

CEP-Engine: Ausgewählte Features für ACCEPT

- Ein spezieller Filterindex erlaubt den parallelen Betrieb von tausenden von Filter-EPAs gleichzeitig bei nahezu konstant hohem Datendurchsatz

- Statische Datenbanktabellen können in-memory gehalten und mit Datenströmen korreliert werden
  - So performant wie Korrelationen von Datenströmen untereinander

Historische Datenbank (Event Store)

- Maßgeschneiderte Datenbank zum Aufzeichnen und Anfragen von hochfrequenten und massiven Datenströmen
- Grundlage für die Erzeugung von Modellen
- Essentiell für Simulationen und „what-if“ Analysen
- Block-weise Kompression mit LZ4
- Berechnung von Statistiken und Aggregaten on-the-fly
- Verlustbehaftete Verdichtung von sehr alten Daten

Modell Datenbank

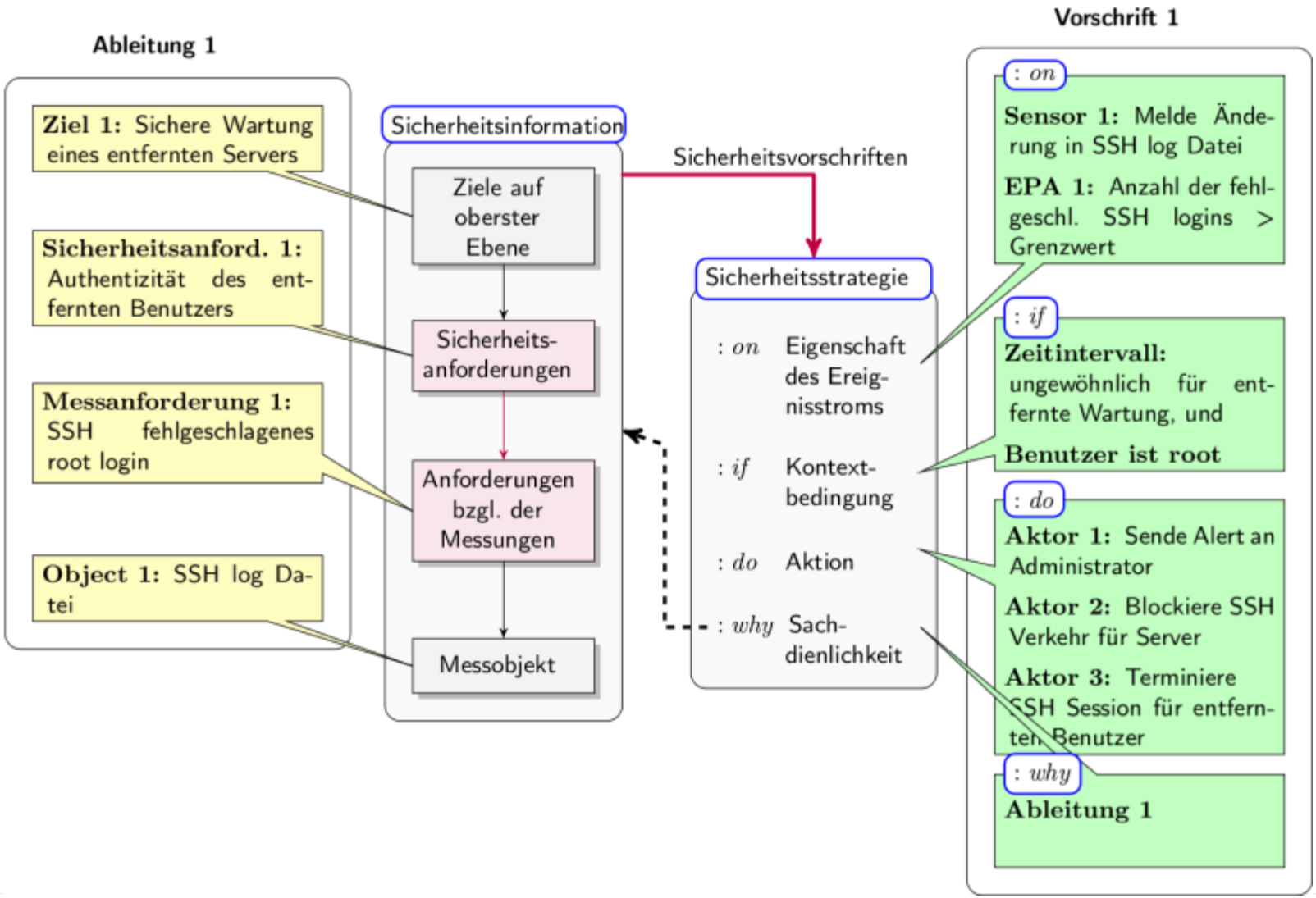
- Generiert und verwaltet Modelle auf Basis historischer Daten
- Algorithmen aus dem Bereich des maschinellen Lernens erzeugen Modelle, die das Normalverhalten beobachteter Entitäten beschreiben (z.B. Kerndichteschätzer)
- Ausdruck des Normalverhaltens in Form von statistischen Modellen
- Aus Modellen werden (semi-)automatisch EPAs abgeleitet, die beobachtete Entitäten auf Verhaltenskonformität überwachen

Metadatenmodell

- Das Metamodell für Sicherheitsinformationen beschreibt den Zusammenhang zwischen
  - übergeordneten Zielen
  - den daraus abzuleitenden Sicherheitsanforderungen
  - den passenden Anforderungen an die Messungen
  - den zu messenden Objekten bzw. Attributen

- Das dazu passende Metamodell der Sicherheitsstrategie enthält:
  - alle Quellen von Sicherheitsinformationen
  - den aktuellen Kontext
  - die Sicherheitsregeln für Reaktionen auf Angriffe
  - deren Begründung und Zusammenhang

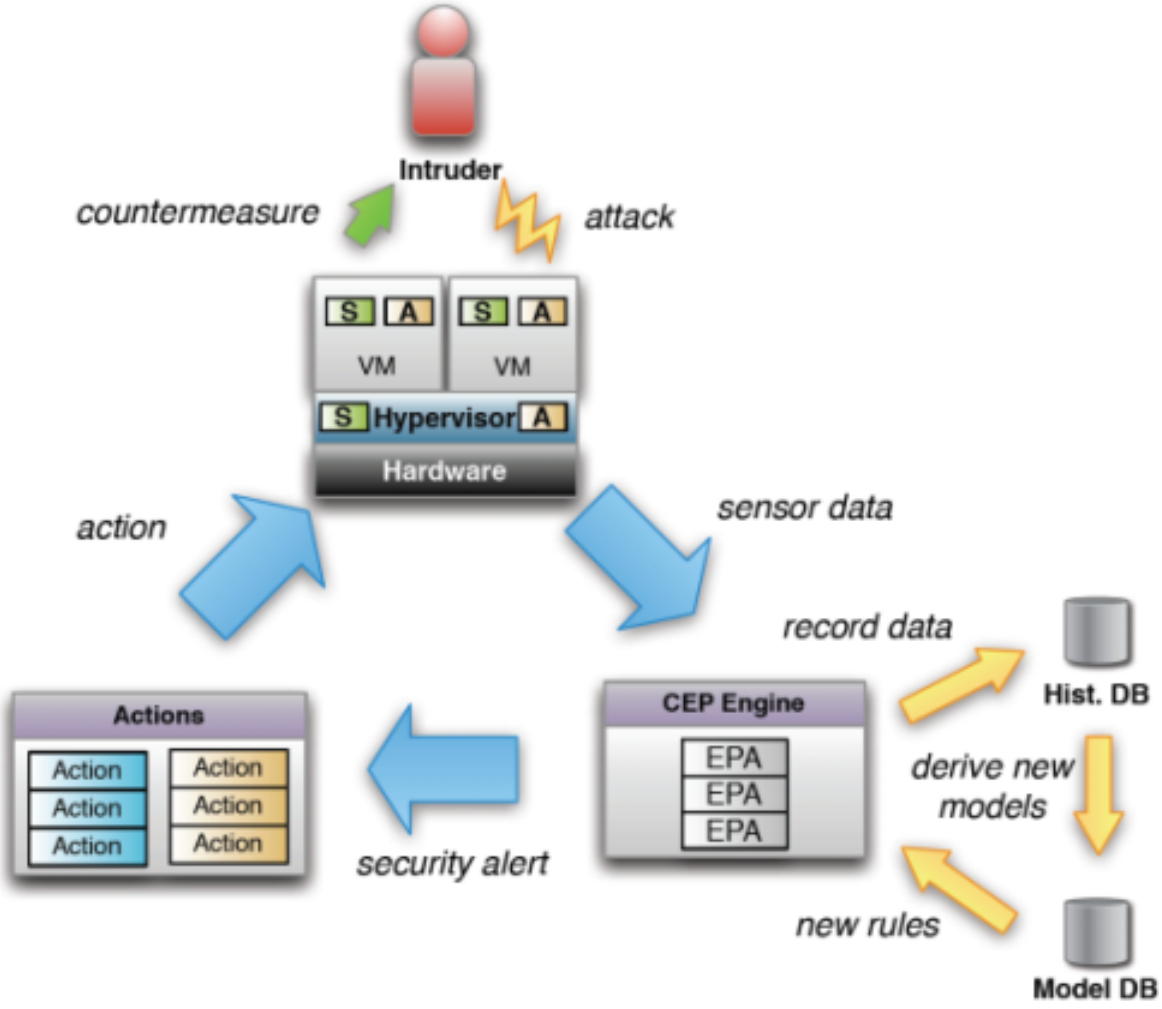
Beispiel: Sicherheitsvorschrift



Matchmaker

- Vollautomatisches Verbinden von
  - Sensoren mit EPAs
  - EPAs mit EPAs
  - EPAs mit Aktionen
- Hat Kenntnis über die Metadaten aller platzierten Sensoren, EPAs und Aktionen
- Löst unmittelbar entsprechende Aktionen aus, sobald eine Anomalie erkannt wurde
- Als Erweiterung für JEPC umgesetzt

ACCEPT Lifecycle



Security is always going to be a cat and mouse game because there'll be people out there that are hunting for the zero day award, you have people that don't have configuration management, don't have vulnerability management, don't have patch management.

-- Kevin Mitnick

FALLBEISPIELE

Von Real World zu ACCEPT

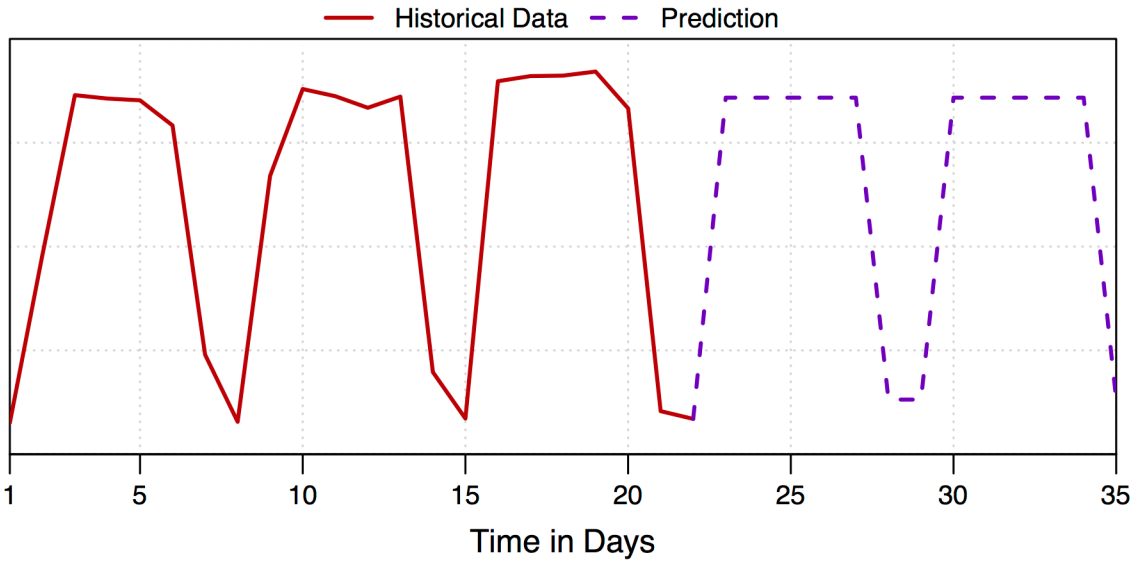
- Auswertung von echten Honeypot Daten
- Ableitung von typischen Angriffen
- Fokus
  - Angriffsvektor (z.B. SQL-Injection, Command-Injection, Buffer Overflow, gestohlene SSH-Zugangsdaten)
  - Post-Exploitation Phase (Aktionen auf kompromittierter Maschine analysieren)
  - Anomalien in Netzwerkverkehr

Beispiel: NetMon

- Ziel: Detektion von Anomalien in Netzwerkverkehr

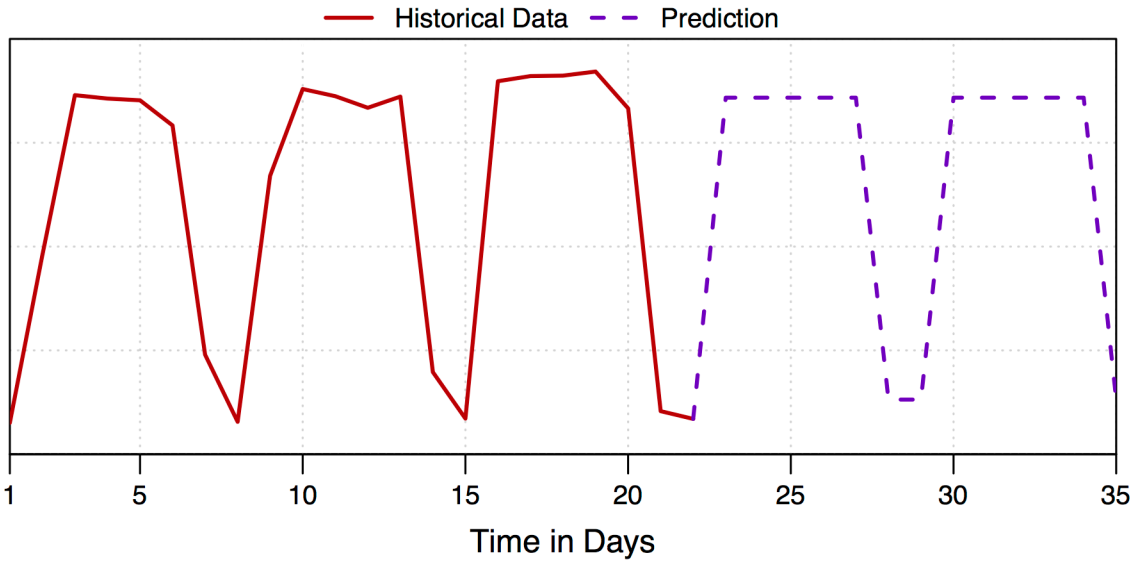
- Ablauf
  - Sensor, der periodisch die aktuelle Netzwerklast ermittelt, startet
  - Matchmaker verbindet automatisch Sensor mit Historischer Datenbank
  - Zeitreihe wird aufgezeichnet ...

Beispiel: NetMon



- Modell-DB erkennt neue Zeitreihe und stößt passenden Lernprozess an:
  - Zeitreihe simplifizieren (Douglas-Peucker-Algorithmus)
  - Zeitreihe clustern (hierarchischer Cluster-Algorithmus)
  - Hidden-Markov Modell trainieren und verfeinern
  - Hidden-Markov Modell in Modell-DB aufnehmen

Beispiel: NetMon



- Modell beschreibt (zukünftige) Normal-Werte im Kontext von Zeit
- Modell-DB generiert passenden, parametrisierten Filter-EPA
- Parameter `vorhergesagt` des EPAs wird angepasst, sobald das vorhergesagte Normalverhalten sich ändert:

```
SELECT * FROM Zeitreihe
 WHERE wert > vorhergesagt + toleranz
    OR wert < vorhergesagt - toleranz
```

Beispiel: NetMon

- EPA zur Erkennung von ungewöhnlichen Netzlasten fertig generiert und konfiguriert
- Matchmaker verbindet neuen EPA automatisch mit den passenden Sensoren und Aktionen
- Ab jetzt werden Modell und EPA durch Simulationen regelmäßig auf ihre Qualität geprüft und ggf. angepasst oder verworfen

Beispiel: JAppMon

- Kataloge mit den häufigsten Sicherheitslücken in Web Anwendungen
  - *Beispiele: SQL injection, command injection, path traversal, unchecked redirection*
  - Quellen:
    - **Bug tracker** verschiedener Projekt Repositiories
    - Analyse von **Sicherheitsreports: CWE, CVE, CAPEC, CERT**

- Entwicklung eines Analysetools, um diese Lücken zu erkennen
  - Basierend auf der WALA Bibliothek (IBM) zur statischen Analyse von Java Bytecode
  - Beschreibungen der Lücken aus Katalog
  - Fokus darauf, False-Positives zu verringern
  - Validierung durch synthetische Beispiele und mit echten Anwendungen (Stanford SecuriBench)

Beispiel: JAppMon

- Einsatz der statischen Analyse:
  - Offline: Angestossen vom **Backend**
  - Online: Als **Sensor**, angeschlossen an ein Application Container
- Lifecycle:
  - Stösst die Analyse zur Deploy-Time an
  - Ergebnis: potentiell gefährliche Stellen in der Anwendung
  - Installation von Sensoren zur Laufzeit um diese Stellen zu überwachen
- Reaktionen auf Exploits, z.B.
  - Abbruch von Anfragen, Blockieren des angreifenden Clients
  - Deinstallation der Anwendung

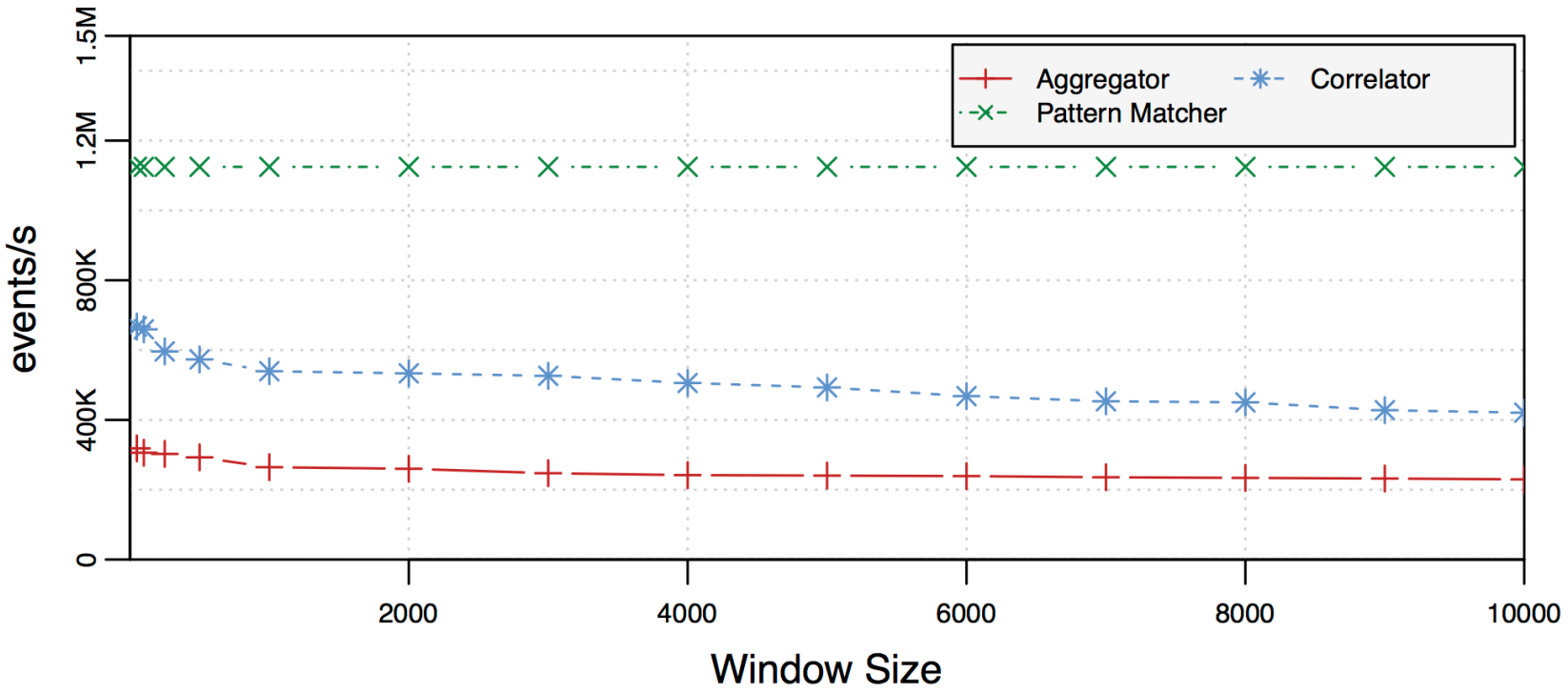
There are risks and costs to a program of action--but they are far less than the long range cost of comfortable inaction.

— John F. Kennedy

EVALUATION

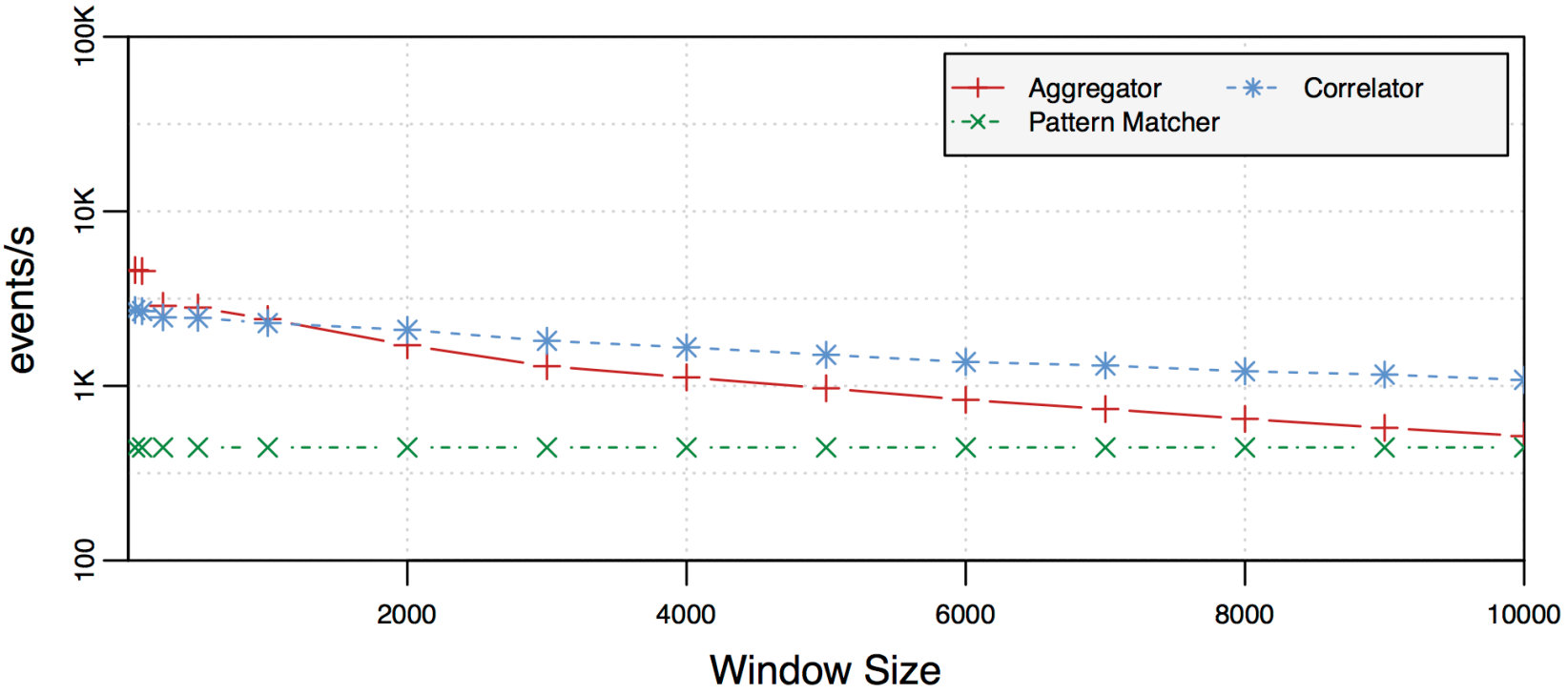
CEP-Technologie

- Datendurchsatz auf einem Core einer i7-CPU:
  - Filtern: Über 3,7 Mio. Events/Sekunde
  - Aggregieren: Über 300K Events/Sekunde
  - Korrelieren: Über 650K Events/Sekunde
  - Mustererkennung: Über 1,1 Mio. Events/Sekunde



Zum Vergleich: Standard DBS

- Datendurchsatz auf einem Core einer i7-CPU:
  - Filtern: 4.147 Events/Sekunde
  - Rest: siehe Grafik (beachte logarithmische Skala)



Historische Datenbank (Event Store)

- Schreibdurchsatz auf einer Maschine mit einer Disk
  - Standard Datenbanksysteme: Unter 10K Events/Sekunde
  - NoSQL Datenbanksysteme: Unter 100K Events/Sekunde
  - Historische Datenbank: Über 5 Mio. Events/Sekunde
- Besseres Maß: Durchsatz in MBs/Sekunde
  - Historische Datenbank reizt maximale Schreibrate von Magnetplatten aus
- Historische Datenbank hat höheren Durchsatz als CEP-Komponente
  => Wird nicht zum Flaschenhals

Matchmaker

- Verbindungen werden ausschließlich offline berechnet, wenn sich eine Änderung auf Seiten der Sensoren, EPAs oder Aktionen ergeben hat
- Kein Einfluss auf den Datendurchsatz von CEP-Komponente und Historischer Datenbank zur Laufzeit
- Neuberechnen aller Verbindungen dauert nur wenige Millisekunden bis Sekunden

JAppMon

- Hohe Precision- und Recall-Werte (Werte für beide zwischen 0.74 und 1.0; in vielen Fällen 1.0) über alle ausgewählten Sicherheitslücken und Anwendungen (d.h. niedrige Anzahl von „False Positives“ and „False Negatives“)

- Moderate Ausführungszeiten und Speicherverbrauch
  (18-991 Sekunden für 68-2368 Java-Klassen; Speicher 300-750 MB)

- Neue (verdächtige) Instanzen des „Confused Deputy Problems“ wurden in Java 7 gefunden (d.h. Methoden, die ohne Rechteüberprüfung Zugriff auf gesicherte Methoden erlauben)
  → Oracle wurde informiert

It is not in the stars to hold our destiny but in ourselves.

-- William Shakespeare

AUSBLICK

Zusammenfassung

- Beiträge
  - Mehrstufiger Ansatz zur automatischen Erkennung, Analyse und Behandlung von Sicherheitsanomalien
  - Detektion bisher unbekanntner Anomalien
  - Flexible Möglichkeit zur Hinzunahme/Wegnahme von Sensoren
  - Effiziente Verarbeitung von Ereignissen
- Nutzung von aktuellen Technologien (z.B. Virtualisierung und Complex Event Processing)
- CEP Backend erlaubt effektives Monitoring auf allen Ebenen
- Zukünftige Arbeiten: Implementierung, Fallbeispiele, Test und kommerzielles Deployment

Publikationen

- L. Baumgärtner, P. Graubner, M. Leinweber, R. Schwarzkopf, M. Schmidt, B. Seeger, B. Freisleben: Mastering Security Anomalies in Virtualized Computing Environments via Complex Event Processing. In: Proc. of eKNOW 2012
- B. Hoßbach, N. Glombiewski, A. Morgen, F. Ritter, B. Seeger: JEPC: The Java Event Processing Connectivity. In: DASP, to appear
- B. Hoßbach, B. Seeger: Anomaly Management using Complex Event Processing. In: Proc. of EDBT 2013
- N. Glombiewski, B. Hoßbach, A. Morgen, F. Ritter, B. Seeger: Event Processing on your own Database. In: Proc. of BTW 2013
- R. Rieke, J. Schütte, A. Hutchison: Architecting a security strategy measurement and management system. In: Proc. of MDSec 2012
- G. Salvaneschi, J. Drechsler, Joscha, M. Mezini: Towards Distributed Reactive Programming. In: Proc. of COORDINATION 2013
- J. Van Ham, G. Salvaneschi, M. Mezini, J. Noye: JEScala - Modular Coordination with Declarative Events and Joins. In: Proc of AOSD 2014, to appear