OWASP AppSec Germany 2009 Conference
http://www.owasp.org/index.php/Germany

11:55-12:30

Adaptive Sicherheit durch Anomalieerkennung

Hartmut Keil

AdNovum Informatik AG, Zürich
hartmut.keil@adnovum.ch
++41 44 272 61 11

OWASP
Nürnberg, 13.10.09

Agenda

- Überblick bisheriger Ansätze von WAFs
- Anomalieerkennung
  - Learning Phase
  - Detection Phase
- Herausforderungen
- Implementierung
- Zusammenfassung

Bisherige Ansätze – Blacklisting

- **Attack-Patterns werden beschrieben**
- **Vorteile**
  - Etablierte Technik (Hauptfokus bei vielen WAFs)
  - Generische Blacklists vorhanden, die auf dem aktuellen Stand gehalten werden (mod_security Core Rules, PHP-IDS, ... kommerzielle WAFs)
- **Nachteile**
  - Kein Schutz vor 'Zero-Day Exploits' (negatives Security-Modell)
  - Aufwändige Konfiguration, wenn generische Blacklists infolge von 'false Positives' nicht angewendet werden können
  - Aufwändige Konfiguration im Fall von proprietären Applikationen
  - Moderne Datenaustausch-Formate (XML, JSON etc.) ungenügend unterstützt

Bisherige Ansätze – Whitelisting

- **Legale Requests werden beschrieben**
- **Vorteile**
  - Schutz vor 'Zero-Day Exploits' (positives Security-Modell)
  - Weniger 'false Positives', da explizit konfiguriert wird, was erlaubt ist
- **Nachteile**
  - Konfiguration ist sehr aufwändig, da die zu schützende Applikation analysiert werden muss
  - moderne Datenaustausch-Formate (XML, JSON, etc.) ungenügend unterstützt

Bisherige Ansätze – Dynamic Whitelisting

- **Signierung der in der Response enthaltenen URLs & Forms**. Whitelist wird durch den Content bestimmt. href='/account.html' wird durch href='/account.html?sig=vbedjdh1ks...' ersetzt. (Anstelle von Signierung auch Encryption möglich.)
- **Vorteile**
  - Schutz vor 'Zero-Day Exploits' (positives Security-Modell)
  - Weniger 'false Positives'
  - Minimaler Konfigurationsaufwand
- **Nachteile**
  - Nur Struktur des Inputs wird validiert, nicht die Werte
  - Probleme mit Applikationen, bei denen die URLs und HTML-Formulare via JavaScript auf dem Client zusammengesetzt werden
  - Validierung von modernen Datenaustausch-Formaten (XML, JSON etc.) prinzipiell nicht möglich

Bisherige Ansätze – Limitierungen in der Praxis

- **Defizite bei der Input-Validierung**
  - Dynamisches Whitelisting validiert nur die Struktur
  - Ansonsten vor allem Blacklists → kein Schutz vor 'Zero-Day Exploits'
  - Whitelists zu aufwändig in der Konfiguration
  - Ungenügende Behandlung von komplexen Strukturen (z.B. XML, JSON)
- **Probleme mit Rich Internet Applications (z.B. AJAX)**
  - Dynamic Whitelisting (eines der mächtigsten Features) funktioniert nicht mit RIAs
  - Ernsthaftes Problem wegen zunehmender Verbreitung von RIAs

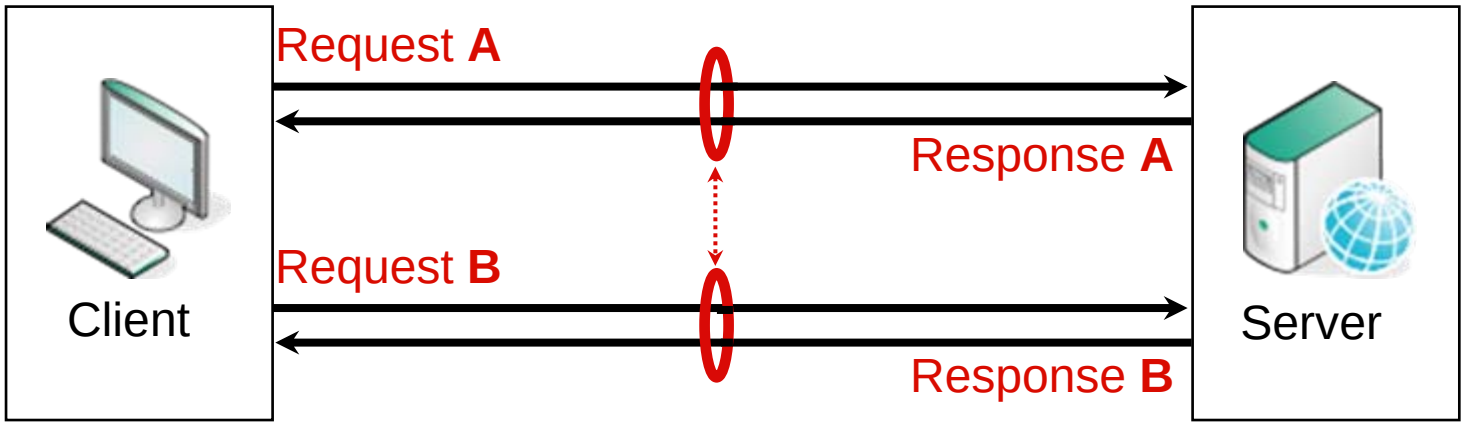
➢ **Fazit:**
In der Praxis ist Input-Validierung mit statischen Rules kein realistischer Ansatz mehr.

Anomalieerkennung – Allgemein

- **Was ist Anomalieerkennung**?
  - Detektieren von 'nicht normalem' Verhalten
  - Was ist 'normales' Verhalten?
    - Kann mit Regeln spezifiziert werden
    - Kann automatisch gelernt werden
- **Beispiele für Einsatzgebiete**
  - Spamfilter
  - Anitivirensoftware
  - Intrusion-Detection-Systeme
  - Fraud Detection
  - Web Application Firewalls

Anomalieerkennung – Im Kontext einer WAF

- **Welche Elemente sieht eine Web Application Firewall ?**
  - Requests
  - Response zu einem bestimmten Request
  - Abfolgen/Beziehungen von Requests innerhalb einer Session

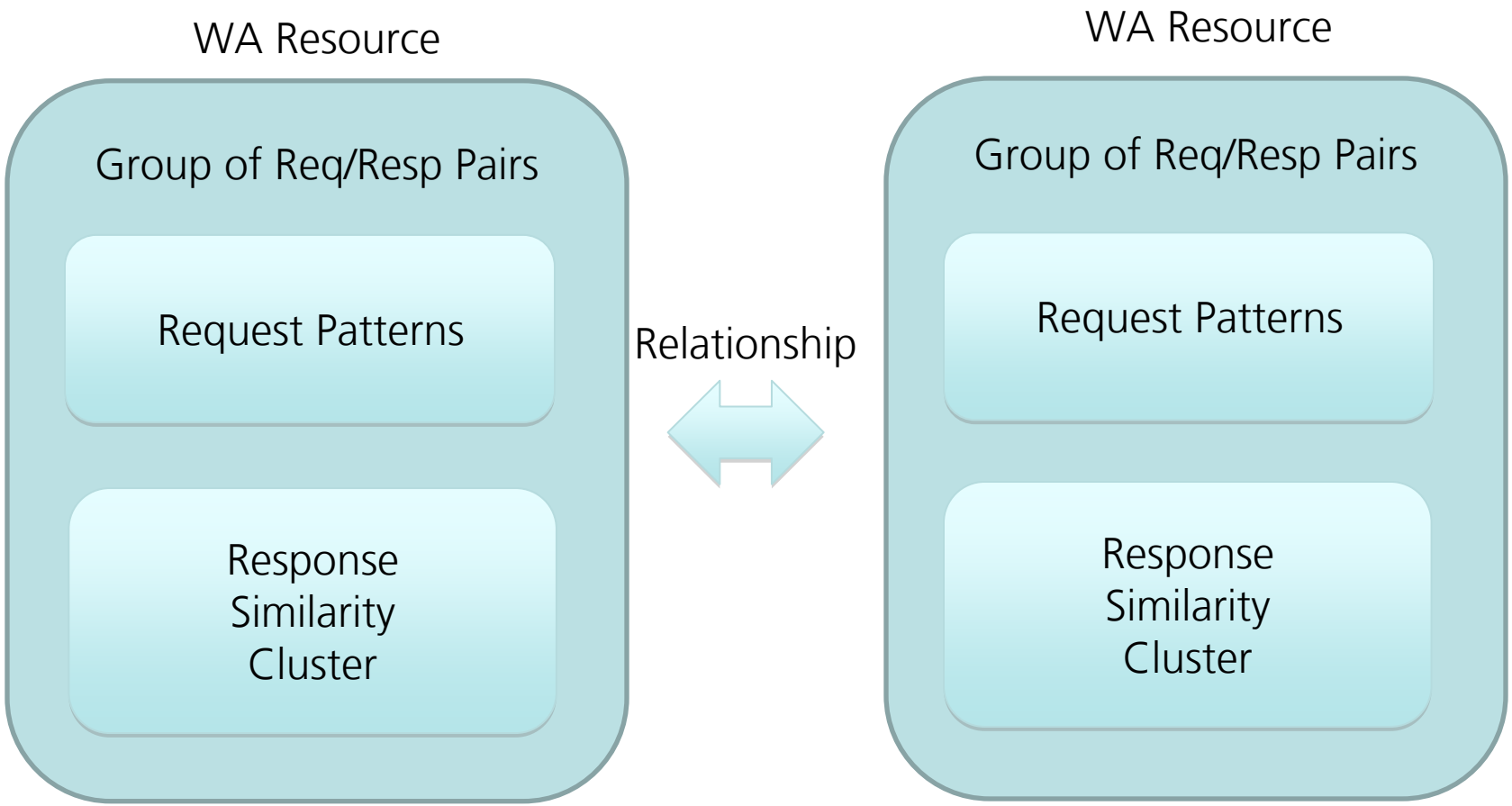


Anomalieerkennung – Phasen

- **Learning Phase**
  - Inkrementelle/adaptive Generierung von Regeln
    - für reguläre Responses *(Response Similarity Cluster)*
    - für reguläre Requests *(Request Patterns)*
  - Finden von Beziehungen zwischen Request/Response-Paaren aufgrund der zeitlichen Abfolge innerhalb einer Session *(Inter Resource Relationship)*
- **Detection Phase**
  - Validierung des Requests anhand der generierten *Request Patterns*
  - Überprüfen der *Inter Resource Relationship*
  - Validierung/Zuordnung der Response zu einem *Response Similarity Cluster*

➢ **Positives Security-Modell**

Anomalieerkennung – Web Application Model



Learning Phase – Grundidee

1. **Gruppierung von Request-Response-Paaren**
   basierend auf der Ähnlichkeit des Response-Inhalts

   ➡ Similarity Cluster bzgl. Response-Inhalt

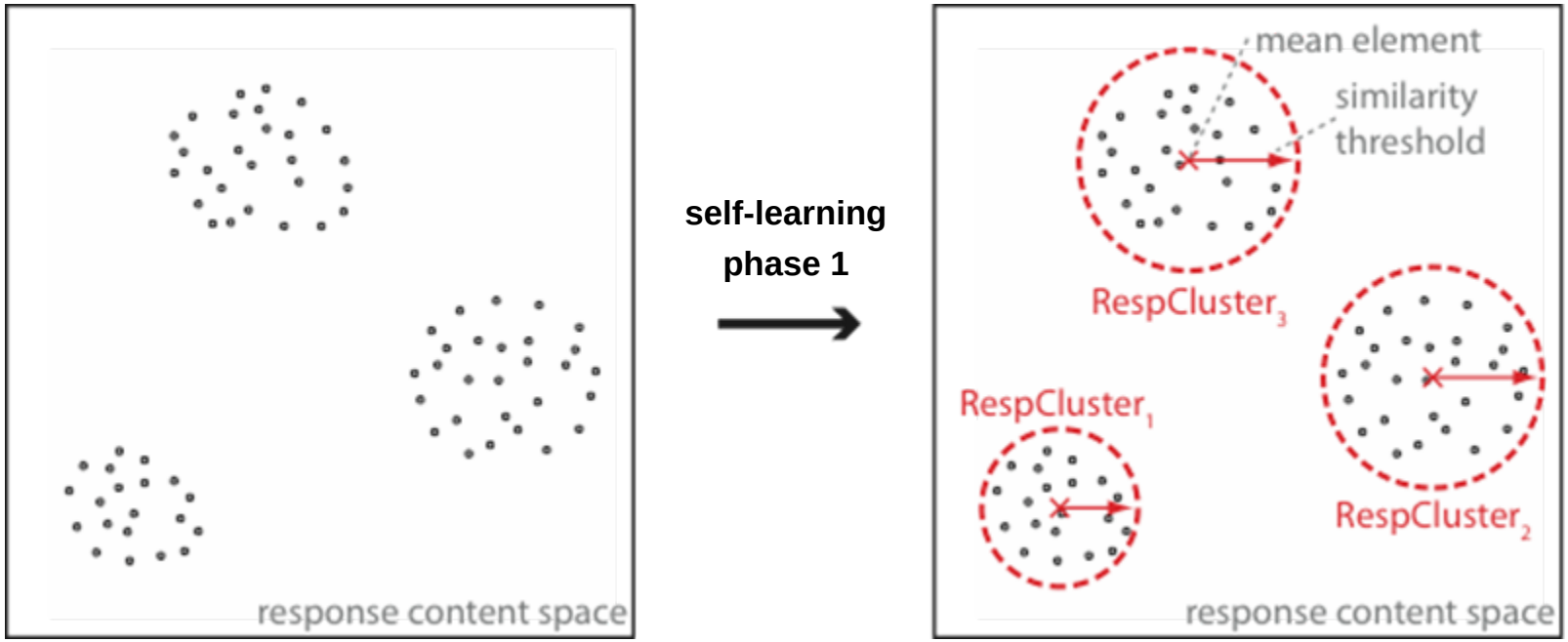
2. **Finden gemeinsamer Request Patterns,**
   die zu Responses aus dem gleichen Similarity Cluster führen

   ➡ Finden von Request Patterns

Grundlegende Annahme

➡ Ähnlichkeiten zwischen zwei Responses ergeben sich durch Ähnlichkeiten der entsprechenden Requests

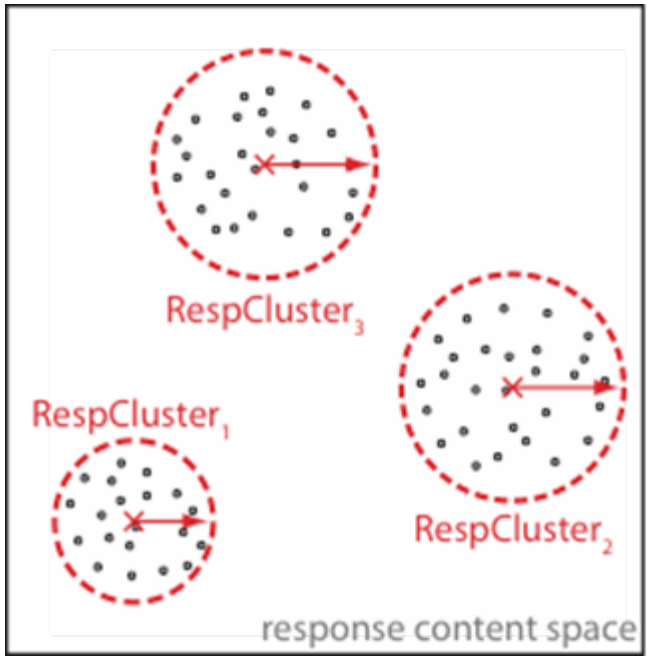
Learning Phase – Similarity-Cluster



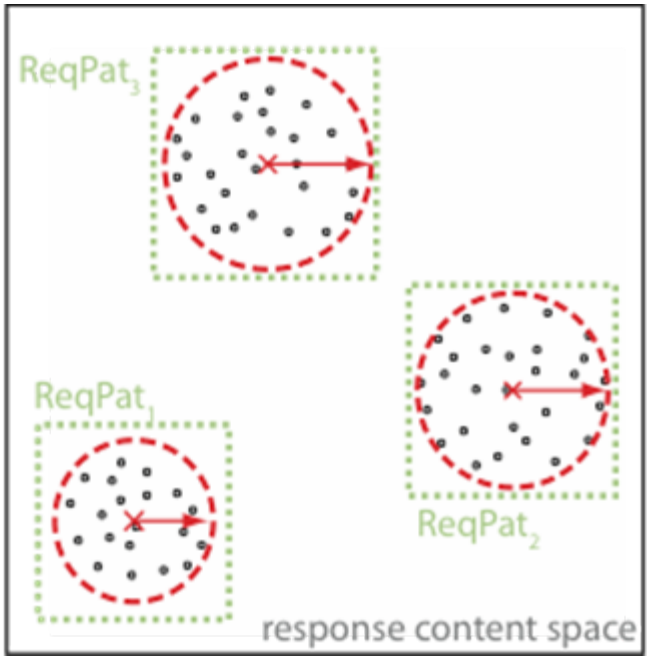
Learning Phase – Similarity Cluster

- **Idee/Beobachtung**
  - Zentraler Punkt ist die Darstellung von Response-Inhalten
  - Alle bekannten Response-Typen (HTML, XML, JSON etc.) lassen sich in einer Baumstruktur darstellen
  - Die Ähnlichkeit von Response-Inhalten lässt sich in obiger Darstellung mit etablierten Algorithmen bestimmen
  - Beim Clustering ist die Baumstruktur von Bedeutung
- **Vorteil**
  - Unabhängigkeit vom konkreten Response-Typ
  - Leicht zu erweitern

Learning Phase – Request Pattern



self-learning phase 2

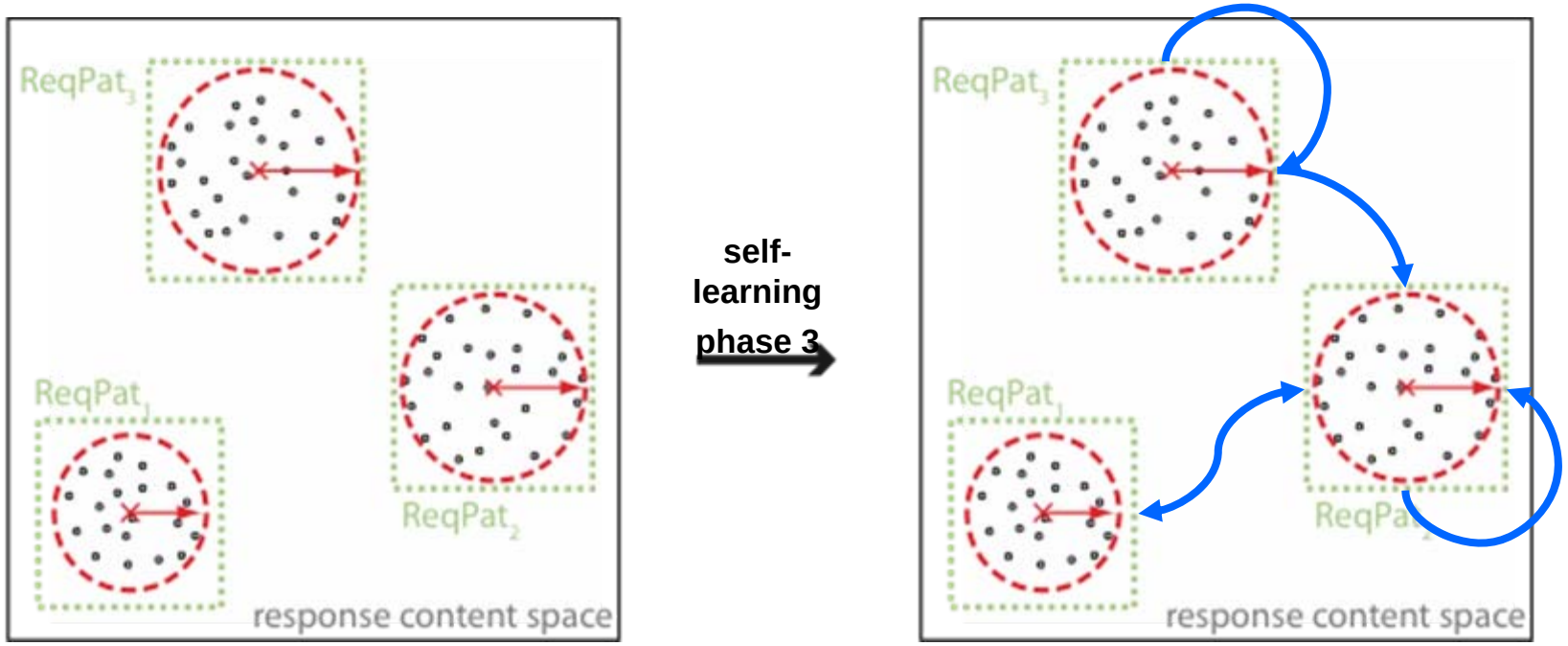


Learning Phase – Request Pattern

- **Idee/Beobachtung**
  - Zentraler Punkt ist wieder die Darstellung von Requests
  - Alle bekannten Request-Typen (Form-Post, XML, JSON etc.) lassen sich in einer Baumstruktur darstellen
  - Die Ähnlichkeit von Request-Inhalten lässt sich in obiger Darstellung mit etablierten Algorithmen bestimmen
  - Für die Request Patterns sind Baumstruktur und Inhalt von Bedeutung
    - Pattern für Inhalt: n-Gramme, regular expressions etc.
- **Vorteil**
  - Unabhängigkeit vom konkreten Request-Typ
  - Trennung von Struktur und Inhalt
  - Leicht zu erweitern

Learning Phase – Inter Resource Relationship

Korrelation zwischen verschiedenen Requests, das heisst, eine Session ist erforderlich



Detection Phase

- **Validierung eines Requests anhand der generierten Request Patterns**
  - Request wird blockiert, falls kein passendes Request Pattern gefunden wird
- **Überprüfen der Inter Resource Relationship**
  - Request wird blockiert, falls nicht erfolgreich (Session erforderlich)
- **Zuordnung/Validierung der Response zu einem Response Similarity Cluster**
  - Nachträgliches Entdecken von Attacken (a posteriori detection)
  - Verhindern von Information-Leakage, Daten-Diebstahl

➢ **Positives Security-Modell**

Herausforderungen

- **Vermeidung von 'false Positives'** verursacht durch
  - Unvollständiges Lernen
  - Sich ändernde Applikationen während der Detection Phase
- **Attacken während der Learning Phase**
  - Falls nicht in einem Clean-Room-Setup gelernt werden kann
  - Gelernte Attacken werden später als korrekt validiert
- **Fehlerhafter oder nicht standardkonformer Content**
  - Kann u.U. nicht geparst werden und verhindert das Response Clustering
- **Performance, Skalierbarkeit**
  - Parsen von Requests und Responses kostet Performance und erschwert Streaming
  - Degeneration des Response Clusterings

Implementierung – Stand

- **'Proof of Concept'-Implementierung als J2EE Servlet Filter**
- **Deployment-Varianten**
  - Als Teil der Web-Applikation
  - Als Filter in einem Reverse Proxy
  - (ICAP Service, Tomcat Valve etc.)
- **Setup für nicht-invasive Tests bei Kunden aufgebaut**

Implementierung – Next Steps

- **Testing der 'Proof of Concept'-Implementierung mit echten Daten**
  - Performance?
  - Exaktere Request Patterns durch Response Clustering? (Grundannahme erfüllt?)
  - Degeneration des Response Clusterings?
- **Abhängig von Testresultaten eventuell Implementierung weiterer Response-Clustering-Algorithmen**
- **Entwicklung einer Administrationsapplikation**
- **Productizing**

Zusammenfassung

- In der Praxis ist Input-Validierung mit statischen Rules kein realistischer Ansatz mehr

- Mehrwert des vorgestellten Ansatzes
  - Weitgehend konfigurationsfrei
  - Feingranulare Input- und Output-Validierung
  - Auch für Rich Internet Applications anwendbar
  - Erweiterung für neue Content-Typen möglich
  - Positives Security-Modell

Fragen

Fragen?

Feedback?

Danke für Ihre Aufmerksamkeit.