Anomalieerkennung in XML-basierter Interaktion*

Harald Lampesberger

Christian Doppler Labor für Client-Centric Cloud Computing
Johannes-Kepler-Universität Linz
Softwarepark 21, 4232 Hagenberg
h.lampesberger@cdcc.faw.jku.at

Abstract: XML ist eine semistrukturierte Sprache mit Einsatz in Web- und Cloud-Umfeld und die Verarbeitung von Dokumenten ist ein potentielles Einfallstor für Angreifer. Angriffe auf XML-verarbeitende Komponenten manifestieren sich vor allem in der Dokumentenstruktur, z.B. in der logischen Baumstruktur oder in Inhalten, und Validierung kann die Angriffsfläche deutlich reduzieren. In der Praxis sind die für Validierung erforderlichen Schemas leider oft nicht vorhanden, werden ignoriert oder sind zu allgemein. Der Beitrag beschreibt hierzu einen grammatikalischen Inferenz-Algorithmus für XML, der aus einer Menge von Beispiel-Dokumenten einen Automaten erlernt. Dieser Automat kann wiederum für Stream-Validierung zukünftiger Dokumente eingesetzt werden und so potentielle Angriffe erkennen. Ein geplanter Anwendungsfall ist sprachbasierte Anomalieerkennung in Interaktion, wo XML als Protokollsprache zum Einsatz kommt.

1 Einführung

Web-Technologien spielen eine zentrale Rolle in *Software-as-a-Service* und dem Trend zu mobilen Geräten, z.B. das *Hypertext Transfer Protocol* (HTTP) für Datenübertragung, semistrukturierte Sprachen wie die *Hypertext Markup Language* (HTML), *Extensible Markup Language* (XML) oder *JavaScript Object Notation* (JSON) als Datenaustausch-Format und *JavaScript* für Interaktivität. Aktuell findet man diese Technologien im Web-Umfeld, wo unter anderem der client-seitige Browser ein Angriffsziel für z.B. Drive-by Downloads [PMM+07] oder generell *Exploit-Kits* [Fra12] sind. Dieses Einfallstor wird auch vermehrt von nationalstaatlichen Angreifern gegen Unternehmen oder Personen eingesetzt [Sch13, Wea13]. Im Kontext von Software-as-a-Service ist daher das Unterbinden von Angriffen gegen Client-Software eine zentrale Frage für Sicherheit und Privatsphäre.

Es ist davon auszugehen, dass ein Angreifer technisch in der Lage ist, Daten in der Übertragung für Angriffszwecke zu verändern und eine absehbare Gegenreaktion wird in Zukunft der vermehrte Einsatz von Ende-zu-Ende Verschlüsselung [Wea13]. Eine weitere Entwicklung ist steigende Serviceorientierung – Schwachstellen entstehen in immer höheren Abstraktionsebenen der Interaktion, was das Erkennen von Angriffen erschwert [MZ11]. Als

*Der Beitrag ist eine Zusammenfassung der englischsprachigen Veröffentlichung in [Lam13].

Konsequenz werden klassische Sicherheitsmaßnahmen, die Interaktion auf Netzwerkebene betrachten, wie z.B. Firewalls oder Intrusion Detection Systeme (IDS), in Zukunft die Clients immer weniger schützen können. Das Sicherstellen von korrekter und sicherer Interaktion unterliegt schlussendlich dem Client oder einer Middleware.

Dieser Beitrag konzentriert sich auf XML als Sprachklasse für Client-Cloud Interaktion. Praktische Einsatzgebiete sind z.B. Web Services [ACKM04], XHTML und *Asynchronous JavaScript and XML* (AJAX) [Gar05]. Ein Angriff gegen ein XML-verarbeitendes System ist typischerweise ein manipuliertes Dokument, dessen logische Baumstruktur oder Datentypen unerwartete Struktur aufweisen. Um das Aussehen von Dokumenten zu spezifizieren bietet der XML-Standard das Konzept von *Schema* und *Validierung*: Ein Schema entspricht hier einer Grammatik, die erlaubte Dokumente beschreibt, und strenge Validierung könnte so die Angriffsfläche deutlich reduzieren. Unglücklicherweise ist Validierung nicht verbreitet – nur $8,9\%$ der XML-Dokumente im Web sind valide und referenzieren ein gültiges Schema [GM11]. Des Weiteren sind Schemas oft zu allgemein bzw. in Softwareparadigmen wie AJAX nicht verpflichtend und führen letztendlich zu Ad-hoc-Design.

Der Beitrag diskutiert einen Algorithmus für *grammatikalische Inferenz*, um die zugrunde liegende Sprache von XML-Dokumenten als Automat zu erlernen [Lam13]. Ein gelernter Automat entspricht in gewisser Weiser der Spezifikation der Sprache und kann für *Stream-Validierung* eingesetzt werden, um zukünftige Dokumente mit abnormaler Struktur oder abweichenden Datentypen zu identifizieren. Der Algorithmus stellt die theoretische Grundlage für das angestrebte Anwendungsszenario „*Anomalieerkennung in semistrukturierten Sprachen*“ als Client- oder Middleware-Komponente dar.

1.1 Problematik von Intrusion Detection

Intrusion Detection ist das Erkennen von Angriffsaktivität durch Monitoring eines Zielsystems [DDW99, LKS05]. Ein IDS kann je nach Erkennungsstrategie in Missbrauchs- oder Anomalieerkennung [Den87] und je nach Lokalität des Monitors in host- und netzwerkbasiertes IDS unterschieden werden. Missbrauchserkennung setzt Wissen über Angriffe bzw. Exploits voraus und Beobachtungen werden mit den bekannten Mustern verglichen. Anomalieerkennung konstruiert hingegen ein Modell des *typischen Verhaltens* aus Beobachtungen, üblicherweise mit Hilfe von Maschinenlern- oder statistische Methoden. Dies erfordert jedoch eine Art Training oder Lernphase. Theoretisch können so unbekannte Angriffe (*0-days*) [BD12] bzw. gerichtete Angriffe, die bekannte Muster oder Signaturen vermeiden, erkannt werden.

Netzwerkdaten sind eine beliebte Datenquelle für Monitoring von Interaktion. *Deep Packet Inspection* (DPI) [BM11] ermöglicht den Zugriff auf Inhalte von in Übertragung befindlichen Netzwerkpaketen, z.B. Paket-Header, Paketinhalte, oder reassemblierte Ströme von Anwendungsdaten. Ptacek und Newsham [PN98] zeigen jedoch, wie DPI-Beobachtungen durch Manipulation des *Time-To-Live* Paket-Headers verfälscht werden können – dieses fundamentale Problem existiert weiterhin in vielen aktuellen DPI-Systemen [NLM12]. Alternativ ist der Zugriff auf vollständige Nachrichten durch einen Proxy bzw. durch einen

client- oder middleware-seitigen Filter möglich.

Anomalieerkennung wirkt vielversprechend, ist aber in der Praxis mit Problemen konfrontiert [SP10]: Allem voran ist die Anfälligkeit für Falschalarme und die damit assoziierten Kosten [Axe00], aber auch die schwierige Erklärung von Anomalien und die kontinuierliche Änderung der Normalität (*Concept Drift*).

Angreifer versuchen auch ihre Aktivitäten zu verschleiern z.B. mit *polymorphem Angriffscode* [SLS+07] oder sog. *Mimicry Attacks* [WS02], die sich nicht signifikant von normalem Verhalten unterscheiden. Zusätzlich ist das Training bzw. die Lernphase in Anomalieerkennung anfällig für *Poisoning* [RNH+09]: Wenn der Angreifer Einfluss auf die Trainingsdaten hat, kann fälschlicherweise ein Angriff als normales Verhalten gelernt werden. Chan-Tin et al. [CTHHK11] beschreiben dieses Phänomen als *Frog-Boiling Attack* gegen Systeme, die aus kontinuierlichem Feedback lernen. Der Angreifer verändert stetig die normale Interaktion, bis der eigentliche Angriff unentdeckt bleibt. Für zuverlässige Anomalieerkennung mit guten Falschalarm- und Erkennungsraten muss letztendlich das zugrunde liegende Schwachstellen-Problem genau verstanden werden.

1.2 Formale Sprachen und Sicherheit

Um die Ursache von Software-Schwachstellen besser zu verstehen, betrachten Sassaman et al. [SPB+11] das Schwachstellen-Problem mithilfe *formaler Sprachen*. In Softwareentwicklung sind Modularisierung und Komposition übliche Vorgänge, welche eindeutige Schnittstellen und Protokolle zwischen Komponenten und Systemen voraussetzen. Ein Protokoll spezifiziert in diesem Sinne die Syntax und Semantik einer formalen Sprache, um Information übertragbar zu machen, z.B. Dateiformate oder Netzwerkübertragungen.

Wenn zwei Komponenten S und E interagieren, kodiert der Sender S die Information mithilfe des Protokolls in ein übertragbares Objekt bzw. Signal, z.B. eine Datei oder Netzwerknachricht. Der Empfänger E *interpretiert* dieses Objekt oder Signal entsprechend den Regeln des Protokolls und verändert dadurch seinen eigenen Zustand. So betrachtet können zwei interagierende Komponenten ihre gegenseitigen Zustände beeinflussen, begrenzt durch das vereinbarte Protokoll. Doch in der Praxis können Fehler und Schwächen in Protokollen und deren Implementierungen auftreten und *unerwarteterweise* die Ausdruckskraft des Protokolls erhöhen [SPB+11]; Sender S kann eventuell einen *Exploit* generieren, sodass Empfänger E durch dessen Interpretation in einen unerwarteten oder unsicheren Zustand übergeht. Die ungewollt erhöhte Ausdruckskraft führt so zu einer Schwachstelle.

Um Schwachstellen zu schließen sind eine eindeutige Protokollspezifikation und eine fehlerfreie Implementierung notwendig, sodass ein Empfänger ungültige Objekte bzw. Signale ablehnen kann [SPB+11]. Dies entspricht genau dem *Wortproblem* in formalen Sprachen, welches abhängig von der Ausdruckskraft des Protokolls schwer bzw. unentscheidbar ist [HMRU00]. Eine weitere Schwierigkeit entsteht durch das *Verschachteln* von Protokollen mit überschneidenden Alphabeten, z.B. die Schichten im TCP/IP Modell.

Für Intrusion Detection bedeutet das konkret, dass die Sprachklasse des IDS und observierten Protokolls äquivalent sein müssen. Ist dies nicht der Fall, sind Falschalarme bzw.

schlechten Erkennungsraten die Konsequenz. In der Praxis ist dieses Problem allgegenwärtig, da IDS oft für eine spezifische Sprachklasse konstruiert sind, wo das Wortproblem effizient entscheidbar ist, z.B. reguläre Sprachen im IDS *Snort* [Roe99]. Auch Anomalieerkennung muss Sprachklassen respektieren: Hadžiosmanović et al. [HSB+12] evaluieren z.B. bekannte n-Gramm-basierte Algorithmen mit binären Protokollen, wo Sprachklasse und Alphabet nicht zusammenpassen, und erreichen folglich keine guten Ergebnisse.

2 Problemstellung

Um den erläuterten Problemen von Intrusion Detection bzw. Anomalieerkennung entgegenzuhalten, zielt dieser Beitrag auf *sprachbasierte Anomalieerkennung* ab. Der konkrete Fokus liegt auf XML-basierter Interaktion und folgende Kriterien wurden identifiziert:

Monitoring auf Nachrichtenebene Der Trend zu Ende-zu-Ende Verschlüsselung gestaltet DPI-Monitoring immer schwieriger bzw. unmöglich. Des Weiteren kann ein geübter Angreifer das Assemblieren der Anwendungsdaten aus Paketströmen in manchen Fällen umgehen. Es müssen deshalb ganze Nachrichten, in diesem Fall XML-Dokumente, von dem Client oder einer Middleware analysiert werden.

Äquivalente Sprachklassen Wenn sich die Sprachklasse des Protokolls und Sprachklasse des Analysealgorithmus unterscheiden, kann dies zu Falschalarmen und schlechten Erkennungsraten führen.

Grammatikalische Inferenz Anomalieerkennung kann als Spezialfall von *grammatikalischer Inferenz* [dlH10] betrachtet werden. Das Ziel ist hier, die *Repräsentation* einer formalen Sprache aus ihrer *Präsentation* zu erlernen, z.B. eine Grammatik oder Automat aus Beispielen, Gegenbeispielen oder einem Orakel. Zusätzlich wird von einem *versteckten Lernziel* ausgegangen, welches die Struktur der Sprache charakterisiert, z.B. eine Protokollspezifikation. Die Sprachklasse und Art der Präsentation bestimmen, ob Konvergenz zum Lernziel möglich ist. Konvergenz für Anomalieerkennung würde weniger Falschalarme und höhere Erkennungsraten bedeuten.

Abbildung 1 zeigt die Problemstellung: Der grammatikalische Inferenz-Algorithmus im Lerner konstruiert aus einer Menge von Beispiel-Dokumenten einen Automaten, der die syntaktische Struktur und Datentypen von zukünftigen Dokumenten validieren kann. Ein valides Dokument ist in diesem Sinne *normal* relativ zu der erlernten Sprache aus den Beispiel-Dokumenten.

Die logische Baumstruktur von XML kann als *Document Object Model* (DOM) [W3C05] verarbeitet werden, aber der Speicherbedarf von großen Dokumenten bzw. XML-Streams ist problematisch – Lerner und Automat müssen deshalb Stream-Verarbeitung ermöglichen. Dokumente werden über die *Simple API for XML* (SAX) [SAX04] Schnittstelle als Stream interpretiert und sprachlich äquivalente Repräsentation erfolgt durch *Visibly Pushdown Automaten* (VPA) [AM04, KMV07]. Die folgenden Abschnitte beschreiben allgemein, wie der Lerner einen XML VPA (XVPA) aus Beispiel-Dokumenten erzeugt.

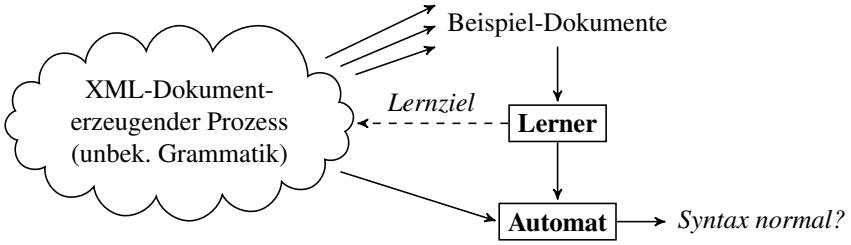

Abbildung 1: Der Lerner erzeugt aus einer Menge von Beispiel-Dokumenten einen speziellen Automaten, der zukünftige Dokumente syntaktisch validieren kann.

2.1 Angriffe in XML-Basierter Interaktion

XML erlaubt komplexe Strukturen und Fehler im Parsing und Interpretation erzeugen Schwachstellen. DOM-Parser sind grundsätzlich anfällig für Denial-of-Service (DoS) Angriffe durch *überlange Elementnamen*, *überlange Inhalte* oder viele *verschachtelte Tags* [FJS11]. Die *Document Type Definition* (DTD) in der Präambel eines Dokuments kann des Weiteren durch rekursive *Entity Expansion* für eine DoS-Attacke oder für Zugriff auf vertrauliche Daten mittels *External Entity References* missbraucht werden.

```
<transaction>
  <total>1000.00<total>
  <cc>
      1234</cc><total>1.00</total><cc>1234
  </cc>
</transaction>
```

(a) XML-Injection Angriff gegen DOM-Parser.

```
<transaction>
  <total>1000.00<total>
  <cc>
      1234' or '1'='1
  </cc>
</transaction>
```

(b) SQL-Injection Angriff.

Abbildung 2: Beispiele für XML-basierte Angriffe.

XML-Injection liegt vor, wenn der Angreifer Teile eines Dokuments kontrolliert und so gezielt modifiziert, dass Fehlinterpretation verursacht wird. Abbildung 2 zeigt fiktive Dokumente, wo der Angreifer die Kreditkartennummer im `cc`-Element steuert. Die spezielle Eingabe in Abbildung 2a kann eine XML-Injection in einem DOM-Parser bewirken, sodass Element `total` letztendlich den Wert 1.0 im DOM erhält [FJS11]. Auch SAX-Parser können für XML-Injection anfällig sein. *Cross-Site Scripting* (XSS) kann man als Spezialfall von XML-Injection sehen, wo der Angreifer ein unerlaubtes JavaScript oder Iframe im Dokument platziert. Auch klassische Angriffe wie *SQL-* (Abbildung 2b), *Command-* oder *XPATH-Injection* im Inhalt eines Elements können zur Gefahr für andere Komponenten im System werden. Generell verändern viele Angriffe die erwartete Dokumentenstruktur oder Datentypen von Inhalten und Validierung kann die Angriffsfläche reduzieren.

2.2 Verwandte Arbeiten

XML-Stream-Validierung entspringt der Forschung von Segoufin und Vianu [SV02]. Des Weiteren beschreiben Kumar et al. [KMV07] VPA als eine geeignete Repräsentation für XML-Streams, welche die Klasse der regulären Baumsprachen abbilden kann. VPA finden praktische Anwendung im *XML Schema* (XSD) Framework von Picalausa et al. [PSZ11].

Ein Überblick zum Thema grammatikalische Inferenz ist in [dlH10]. Zu Inferenz von DTDs aus XML-Dokumenten gibt es bereits Veröffentlichungen [BNSV10, BGNV10, Fer01, GGR$^+$03], in diesem Beitrag liegt jedoch das Augenmerk auf die weitaus größere Sprachklasse von XSD. Eine Bestandsaufnahme zu XSD-Inferenz ist in Mlýnková [Mly08]: Die allgemeine Herangehensweise leitet zuerst eine kontextfreie Grammatik als Sprachrepräsentation aus Beispiel-Dokumenten ab und fusioniert später Nichtterminal-Symbole [MN09]. Hegewald et al. [HNW06] und Chidlovskii [Chi01] behandeln zusätzlich auch Datentypen. Der vorgestellte Algorithmus ist konzeptuell ähnlich zu Bex et al. [BNV07]: Deren Inferenzalgorithmen verwenden Baumautomaten um die Sprachklasse von l-lokalen *Single Occurrence XSDs* ohne Datentypen zu lernen.

Für Informationsgewinnung aus HTML beschreiben Kosala et al. [KBBV06] and Raeymaekers et al. [RBdB08] Algorithmen zum Lernen von Baumautomaten. Bezogen auf die Sicherheitsthematik ist noch die Arbeit von Rieck [Rie09] hervorzuheben. Sie behandelt Baum-Kernel [RKBM10] als Ähnlichkeitsmaß für Baumstrukturen in Maschinenlernverfahren und als Anwendung wird unter anderem Anomalieerkennung in HTML angeführt.

In den nächsten Abschnitten wird der Algorithmus aus [Lam13] konzeptuell beschrieben. Die Vorgehensweise unterscheidet sich zu verwandten Arbeiten dahingehend, dass zum einen direkt ein Streaming-fähiger VPA als Sprachrepräsentation gelernt wird und zum anderen die Datentypen von Element-Inhalten berücksichtigt werden.

3 Sprachbasierte Anomalieerkennung in XML

Der konkrete Algorithmus in [Lam13] erfüllt zwei Aufgaben:

1. *Abstraktion von Dokumenten.* Ein XML-Dokument ist prinzipiell ein String aus Zeichen eines Zeichensatzes, z.B. Unicode. Die Abstraktion definiert Regeln, wie ein Dokument in eine Sequenz aus SAX-Ereignissen übersetzt wird. Zusätzlich werden XML-Inhalte durch Datentypen abstrahiert.

2. *Automatenkonstruktion.* Aus den Sequenzen von SAX-Ereignissen der Beispiel-Dokumente wird zuerst ein spezieller Automat, der *Visibly Pushdown Prefix Acceptor* (VPPA) erzeugt. Für den Lerneffekt werden ähnliche Zustände im VPPA fusioniert und letztendlich in einen XVPA transformiert. Der XVPA ist das Resultat der grammatikalischen Inferenz, welches wiederum für Stream-Validierung eingesetzt werden kann.

3.1 Abstraktion von Dokumenten

Für die Abstraktion sind folgende SAX-Ereignisse von Relevanz: *startElement* für Beginn-Tags, *endElement* für End-Tags und *characters* für Inhalt. Attribute werden grundsätzlich als spezielle Kind-Elemente interpretiert um die logische Baumstruktur zu erhalten. Zusätzlich werden Namensräume direkt in den Elementnamen eingebettet.

Inhalte in Elementen sind prinzipiell Strings aus dem Zeichensatz und müssen explizit behandelt werden, z.B. kann ein Inhalt eine einfache Nummer, eine Uhrzeit, aber auch natürliche Sprache oder JavaScript darstellen. Die Sprachklasse eines Dokuments ist in diesem Sinne mit den individuellen Sprachklassen von Inhalten verschachtelt, was grammatikalische Inferenz erschwert bzw. unmöglich macht. Inhalte von Elementen werden deshalb abstrahiert.

Die Schemasprache XSD [W3C] bietet für die Spezifikation von Dokumenten eine endliche Menge sog. *Datentypen*. Jeder Datentyp beschreibt einen semantischen Werteraum und einen lexikalischen Raum für die Darstellung als String. Als Beispiel kann der semantische Werteraum $\{true, false\}$ des Datentyps *boolean* durch Strings $\{\texttt{0}, \texttt{1}, \texttt{true}, \texttt{false}\}$ repräsentiert werden. Ein Lerner sieht nur den lexikalischen Raum und um das Problem der verschachtelten Sprachklassen aufzulösen wird ein *lexikalisches Datentypensystem* in [Lam13] definiert. Das Datentypensystem liefert für einen String die Menge der minimalen XSD Datentypen, die den String in ihrem lexikalischen Raum abbilden können. Ein Inhalt wird somit zu einem SAX-Ereignis aus Datentypen abstrahiert und Abbildung 3 zeigt ein einfaches Beispiel.

$$\langle a\rangle\ \langle a\rangle \texttt{10.0} \langle /a\rangle\ \langle b\rangle \texttt{etwas Text} \langle /b\rangle\ \langle b\rangle\langle /b\rangle\ \langle /a\rangle$$
$$\downarrow$$
$$aa\{decimal\}\bar{a}b\{string\}\bar{b}b\bar{b}\bar{a}$$

Abbildung 3: Ein Dokument wird als Sequenz von SAX-Ereignissen und Datentypen abstrahiert.

3.2 Automatenkonstruktion

In wohlgeformtem XML sind die Tags korrekt ineinander verschachtelt. Diese Eigenschaft und die klare Trennung zwischen Tags und Inhalt macht XML formal zu einer *Visibly Pushdown Language* [AM04], eine echte Teilmenge der kontextfreien Sprachen, welche durch VPA repräsentiert werden kann. Eine Sonderform von VPA, sog. XVPA [KMV07], sind spezielle Kellerautomaten für XML. Wie Kellerautomaten besitzen XVPA eine Menge von Zuständen und Zustandsübergängen. Jedoch haben XVPA drei disjunkte Alphabete jeweils für *startElement* SAX-Ereignisse, *endElement* SAX-Ereignisse und *Datentypen*. Des Weiteren ist das Stapelalphabet gleich der Menge der Zustände. Kumar et al. [KMV07] zeigen, dass jedes Schema einen äquivalenten XVPA hat, der XML-Streams validieren kann. Der Abschnitt beschreibt nun, wie man einen XVPA aus Beispiel-Dokumenten lernen kann.

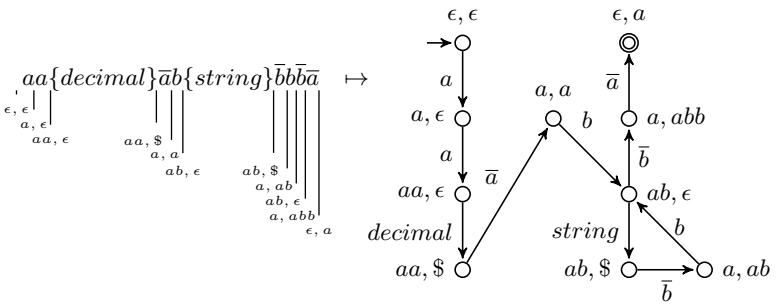

Abbildung 4: Jeder Präfix eines abstrahierten Beispiel-Dokuments beschreibt einen individuellen Zustand. Der VPPA (rechts) wird aus diesen Zuständen aller Beispiel-Dokumente konstruiert.

Im ersten Schritt wird ein VPPA, der exakt die Beispiel-Dokumente darstellt, konstruiert. Intuitiv ist ein VPPA vergleichbar mit einem Präfixbaum: Jeder Präfix von jedem Beispiel-Dokument führt zu einem eindeutigen Zustand im VPPA. Der individuell erreichbare Zustand eines Präfixes wird hierzu als Paar x, y definiert, wo x dem String von Elternelementen und y dem String der linken Geschwisterelemente entspricht. Das Symbol \$ dient als Platzhalter für Datentypen. Abbildung 4 zeigt, wie die individuellen Zustände aus den Präfixen eines Beispiel-Dokuments zu einem VPPA führen.

Der VPPA ist eine kompakte Darstellung der Beispiel-Dokumente, da er exakt diese akzeptiert. Um *zu lernen* werden nun ähnliche Zustände im VPPA fusioniert. Eine sog. Unterscheidungsfunktion [Fer03] kommt zum Einsatz – zwei Zustände können fusioniert werden, wenn sie ähnlich sind. Konkret gelten zwei Zustände als ähnlich, wenn die k linken Geschwisterelemente und l nächsten Elternelemente identisch sind. Im laufenden Beispiel in Abbildung 5 (links) werden mit Parametern $k = 1$ und $l = 2$ die Zustände a, ab und a, abb im VPPA zum neuen Zustand a, b fusioniert, da beide das gleiche Elternelement a und das gleiche linke Geschwisterelement b besitzen. Die Anzahl der Zustände nimmt dadurch ab und ein Lerneffekt durch Generalisierung tritt ein.

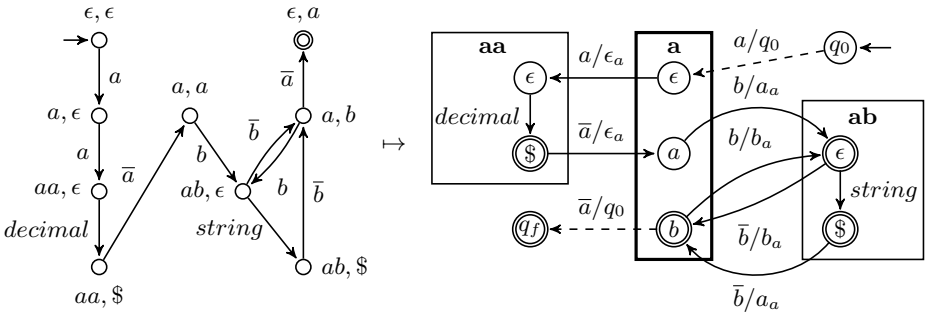

Abbildung 5: Ähnliche Zustände im VPPA werden zuerst fusioniert (links) und der resultierende Automat wird in einen validen XVPA für Stream-Validierung überführt (rechts).

Im letzten Schritt wird der VPPA mit fusionierten Zuständen in einen validen XVPA transformiert. Unter anderem wird die Menge der Zustände anhand deren x-Komponente in sog. Module partitioniert. Ein Modul entspricht dem *Typ* bzw. Inhaltsmodell eines Elements. Letztendlich wird der XVPA minimiert, indem äquivalente Module zusammengefasst werden. Im laufenden Beispiel in Abbildung 5 (rechts) ist ersichtlich, wie der Algorithmus für das Element a zwei unterschiedliche Module erzeugt – das Inhaltsmodell von Wurzelelement a unterscheidet sich deutlich vom Kindelement a. Solche kontextspezifische Inhaltsmodelle sind z.B. in der simplen Schemasprache DTD nicht ausdrückbar.

3.3 Diskussion

Das Lernszenario im beschriebenen Algorithmus ist „*Identifikation aus Positivbeispielen*“ von Gold [Gol67]. Der Algorithmus konvergiert wenn (1) die zu lernende Sprache durch endlich große Parameter k und l beschreibbar ist und (2) die Beispiel-Dokumente *charakteristisch* für die Sprache sind.

In der Unterscheidungsfunktion begrenzen Parameter k und l die Lokalität eines Zustands und schränken somit die generell lernbare Sprachklasse ein. Diese Einschränkung deckt immerhin einen großen Teil praktisch vorkommender Dokumente ab: Die Studie von Bex et al. [BNV04] untersucht die Komplexität praktischer Schemas und zeigt, dass Inhaltsmodelle grundsätzlich einfache Strukturen verwenden. Werden Parameter k und l für den Algorithmus kleiner gewählt, als die zugrunde liegende Sprache erfordert, konvergiert der Algorithmus zwar schneller, aber der resultierende Automat wird eine Vereinfachung der Sprache. Zu große Parameter erfordern wiederum deutlich mehr Beispiel-Dokumente um Konvergenz zu erreichen.

Für die angestrebte Anwendung „sprachbasierte Anomalieerkennung“ hat dieses Lernszenario noch Einschränkungen: Zum einen müssen Beispiel-Dokumente frei von Angriffen sein und der Lerner ist dadurch grundsätzlich anfällig für Poisoning. Zum anderen werden allmähliche Veränderungen der zugrunde liegenden Sprache (Concept Drift) noch nicht berücksichtigt. Zukünftige Forschung konzentriert sich auf diese Einschränkungen.

Anwendungsbeispiel Aktuell wird eine client-seitige Browser-Erweiterung implementiert, um XSS-Angriffe in XHTML-basierten Webseiten zu erkennen. Ein XSS-Angriff beruht auf der Einbettung von bösartigem JavaScript bzw. eines Iframes, welches das bösartige JavaScript nachlädt. Diese Art von Angriff wird z.B. im großen Stil von nationalstaatlichen Angreifern durch Paket-Injektion an zentralen Netzwerkknoten vollzogen [Sch13, Wea13]. Die Browser-Erweiterung würde hier im Angriffsfall die abweichende Struktur einer regelmäßig besuchten Webseite erkennen und den Benutzer informieren. Des Weiteren erlaubt die Sprachrepräsentation durch XVPA sog. Äquivalenztests: Zwei User an geographisch unterschiedlichen Orten können ihre individuell gelernten Automaten für eine bestimmte Webressource vergleichen, um strukturelle Differenzen festzustellen, die auf einen Angreifer oder Zensor rückschließen.

4 Zusammenfassung

Der Beitrag beschreibt einen grammatikalischen Inferenz-Algorithmus für die geplante Anwendung als sprachbasierte Anomalieerkennung [Lam13]. XML kommt oft als Sprache für Interaktion im Cloud- und Web-Umfeld zum Einsatz und typische Angriffe manifestieren sich vor allem in der Dokumentenstruktur oder Datentypen von Inhalten. Der Lerner erzeugt aus Beispiel-Dokumenten einen Automaten, der Struktur und Datentypen von zukünftigen Dokumenten validieren kann. Dadurch kann die Angriffsfläche von XML-verarbeitenden Komponenten deutlich reduziert werden.

Das Projekt ist noch in einem frühen Status. Mittlerweile existiert ein Softwareprototyp, der als Referenz für zukünftige Weiterentwicklungen dient. Neben der Anfälligkeit für Poisoning im Lernszenario haben erste Experimente gezeigt, dass das lexikalisches Datentypensystem basierend auf XSD-Datentypen in manchen Fällen zu allgemein ist und an approximierten Datentypen wird deshalb geforscht. Des Weiteren sind ein probabilistisches Automatenmodell und inkrementelles Lernen geplant, um Lernen von unvollständigen Trainingsdaten zu ermöglichen. Schließlich ist es ein großes Ziel, die Ergebnisse zu grammatikalischer Inferenz und Stream-Validierung auf andere bekannte Sprachklassen wie JSON und HTML5 auszudehnen und im Softwareprototyp zu implementieren.

Literatur

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi A. Kuno und Vijay Machiraj. *Web Services - Concepts, Architectures and Applications*. Springer, 2004.

[AM04] Rajeev Alur und P. Madhusudan. Visibly Pushdown Languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, STOC'04, Seiten 202–211. ACM Press, 2004.

[Axe00] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security*, 3(3):186–205, August 2000.

[BD12] Leyla Bilge und Tudor Dumitras. Before We Knew It: an Empirical Study of Zero-Day Attacks in the Real World. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS'12, Seiten 833–844. ACM Press, 2012.

[BGNV10] Geert Jan Bex, Wouter Gelade, Frank Neven und Stijn Vansummeren. Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data. *ACM Transactions on the Web*, 4(4):1–32, 2010.

[BM11] R. Bendrath und M. Mueller. The end of the net as we know it? Deep packet inspection and internet governance. *New Media & Society*, 13(7):1142–1160, April 2011.

[BNSV10] Geert Jan Bex, Frank Neven, Thomas Schwentick und Stijn Vansummeren. Inference of Concise Regular Expressions and DTDs. *ACM Transactions on Database Systems*, 35(2):1–47, 2010.

[BNV04] Geert Jan Bex, Frank Neven und Jan Van den Bussche. DTDs versus XML Schema: A Practical Study. In *Proc. of the 7th Int. Workshop on the Web and Databases*, WebDB '04, Seite 79. ACM Press, 2004.

[BNV07] Geert Jan Bex, Frank Neven und Stijn Vansummeren. Inferring XML Schema Definitions from XML Data. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB'07, Seiten 998–1009. VLDB Endowment, 2007.

[Chi01] Boris Chidlovskii. Schema Extraction from XML: A Grammatical Inference Approach. In *Proceedings of the 8th International Workshop on Knowledge Representation meets Databases*, KRDB'01, 2001.

[CTHHK11] Eric Chan-Tin, Victor Heorhiadi, Nicholas Hopper und Yongdae Kim. The Frog-Boiling Attack: Limitations of Secure Network Coordinate Systems. *ACM Transactions on Information and System Security*, 14(3):1–23, November 2011.

[DDW99] Hervé Debar, Marc Dacier und Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, April 1999.

[Den87] Dorothy E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, Februar 1987.

[dlH10] Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[Fer01] Henning Fernau. Learning XML Grammars. In *Machine Learning and Data Mining in Pattern Recognition (MLDM'01)*, Jgg. 2123 of *Lecture Notes of Computer Science*, Seiten 73–87. Springer Berlin Heidelberg, 2001.

[Fer03] Henning Fernau. Identification of Function Distinguishable Languages. *Theoretical Computer Science*, 290(3):1679–1711, 2003.

[FJS11] Andreas Falkenberg, Meiko Jensen und Joerg Schwenk. Welcome to WS-Attacks.org. Online: `http://www.ws-attacks.org`, 2011. Abruf 05.02.2013.

[Fra12] Howard Fraser. Exploring the Blackhole Exploit Kit. Online: `http://nakedsecurity.sophos.com/exploring-the-blackhole-exploit-kit/`, März 2012. Abruf 27.11.2013.

[Gar05] Jesse James Garrett. Ajax. Online: `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications`, Februar 2005. Abruf 27.03.2013.

[GGR+03] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri und Kyuseok Shim. XTRACT: Learning Document Type Descriptors from XML Document Collections. *Data Mining and Knowledge Discovery*, 7(1):23–56, 2003.

[GM11] Steven Grijzenhout und Maarten Marx. The Quality of the XML Web. In *Proceedings of the 20th ACM International Conference on Information and Knowledge management*, CIKM'11, Seiten 1719–1724. ACM Press, 2011.

[Gol67] E Mark Gold. Language Identification in the Limit. *Information and Control*, 10(5):447–474, 1967.

[HMRU00] John E. Hopcroft, Rajeev Motwani, Rotwani und Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computability*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2.. Auflage, 2000.

[HNW06] Jan Hegewald, Felix Naumann und Melanie Weis. XStruct: Efficient Schema Extraction from Multiple and Large XML Documents. In *22nd International Conference on Data Engineering Workshops*, ICDEW'06, Seiten 81–81. IEEE, 2006.

[HSB+12] Dina Hadžiosmanović, Lorenzo Simionato, Damiano Bolzoni, Emmanuele Zambon und Sandro Etalle. N-Gram against the Machine: On the Feasibility of the N-Gram Network Analysis for Binary Protocols. In *Research in Attacks, Intrusions, and Defenses (RAID'12)*, Jgg. 7462 of *Lecture Notes in Computer Science*, Seiten 354–373. Springer Berlin Heidelberg, 2012.

[KBBV06] Raymondus Kosala, Hendrik Blockeel, Maurice Bruynooghe und Jan Van den Bussche. Information Extraction from Structured Documents Using k-Testable Tree Automaton Inference. *Data & Knowledge Engineering*, 58(2):129–158, 2006.

[KMV07] Viraj Kumar, P. Madhusudan und Mahesh Viswanathan. Visibly Pushdown Automata for Streaming XML. In *Proceedings of the 16th International Conference on World Wide Web*, WWW'07, Seiten 1053–1062. ACM Press, 2007.

[Lam13] Harald Lampesberger. A Grammatical Inference Approach to Language-Based Anomaly Detection in XML. In *2013 International Conference on Availability, Reliability and Security*, ECTCM'13 Workshop, Seiten 685–693. IEEE, 2013.

[LKS05] Aleksandar Lazarevic, Vipin Kumar und Jaideep Srivastava. Intrusion Detection: A Survey. In *Managing Cyber Threats*, Jgg. 5 of *Massive Computing*, Seiten 19–78. Springer US, 2005.

[Mly08] Irena Mlynkova. An Analysis of Approaches to XML Schema Inference. In *IEEE International Conference on Signal Image Technology and Internet Based Systems*, SITIS'08, Seiten 16–23. IEEE, 2008.

[MN09] Irena Mlynkova und Martin Nečaský. Towards Inference of more Realistic XSDs. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC'09, Seiten 639–646. ACM Press, 2009.

[MZ11] Federico Maggi und Stefano Zanero. Is the Future Web more Insecure? Distractions and Solutions of new-old Security Issues and Measures. In *2nd Worldwide Cybersecurity Summit*, WCS'11, Seiten 1–9. IEEE, 2011.

[NLM12] Olli-pekka Niemi, Antti Levomäki und Jukka Manner. Dismantling intrusion prevention systems. *ACM SIGCOMM Computer Communication Review*, 42(4):285–286, September 2012.

[PMM+07] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang und Nagendra Modadugu. The Ghost in the Browser: Analysis of Web-Based Malware. In *Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets*, HotBots'07. USENIX Association, 2007.

[PN98] Thomas H. Ptacek und Timothy N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Secure Networks, Inc., Bericht, Online: `http://insecure.org/stf/secnet_ids/secnet_ids.html`, 1998. Zugriff 13.10.2013.

[PSZ11] François Picalausa, Frédéric Servais und Esteban Zimányi. XEvolve: an XML schema evolution framework. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC'11, Seiten 1645–1650. ACM Press, 2011.

[RBdB08] Stefan Raeymaekers, Maurice Bruynooghe und Jan den Bussche. Learning (k, l)-Contextual Tree Languages for Information Extraction from Web Pages. *Machine Learning*, 71(2):155–183, 2008.

[Rie09] Konrad Rieck. *Machine Learning for Application-Layer Intrusion Detection*. Dissertation, Berlin Institute of Technology, TU Berlin, Germany, 2009.

[RKBM10] Konrad Rieck, Tammo Krüger, Ulf Brefeld und Klaus-Robert Müller. Approximate Tree Kernels. *Journal of Machine Learning Research*, 11:555–580, 2010.

[RNH+09] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft und J. D. Tygar. ANTIDOTE: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC'09, Seiten 1–14. ACM Press, 2009.

[Roe99] Martin Roesch. Snort – Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA'99, Seiten 229–238. USENIX Association, 1999.

[SAX04] SAX Project. Simple API for XML (SAX). Online: `http://www.saxproject.org/`, April 2004. Abruf 08.12.2013.

[Sch13] Bruce Schneier. How the NSA Attacks Tor/Firefox Users With QUANTUM and FOXACID. Online: `https://www.schneier.com/blog/archives/2013/10/how_the_nsa_att.html`, Oktober 2013. Abruf 27.11.2013.

[SLS+07] Yingbo Song, Michael E. Locasto, Angelos Stavrou, Angelos D. Keromytis und Salvatore J. Stolfo. On the infeasibility of modeling polymorphic shellcode. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS'07, Seiten 541–551. ACM Press, 2007.

[SP10] Robin Sommer und Vern Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. *IEEE Symposium on Security and Privacy*, Seiten 305–316, 2010.

[SPB+11] Len Sassaman, Meredith L. Patterson, Sergey Bratus, Michael E. Locasto und Anna Shubina. Security Applications of Formal Language Theory. Bericht TR2011-709, Dartmouth College Computer Science Department, 2011.

[SV02] Luc Segoufin und Victor Vianu. Validating Streaming XML Documents. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems*, PODS'02, Seiten 53–64. ACM Press, 2002.

[W3C] W3C. XML Schema Part 2: Datatypes Second Edition. Online: `http://www.w3.org/TR/xmlschema11-2/`. Abruf: 22.03.2013.

[W3C05] W3C. Document Object Model (DOM). Online: `http://www.w3.org/DOM/`, 2005. Abruf 24.01.2013.

[Wea13] Nicholas Weaver. Our Government Has Weaponized the Internet. Here's How They Did It. Online: `http://www.wired.com/opinion/2013/11/this-is-how-the-internet-backbone-has-been-turned-into-a-weapon/`, November 2013. Abruf 27.11.2013.

[WS02] David Wagner und Paolo Soto. Mimicry Attacks on Host-Based Intrusion Detection Systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS'02, Seiten 255–264. ACM, 2002.