Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jonas Schäufler

Die Erweiterung einer Web-Application Firewall durch Reporting- und Monitoring Schnittstellen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jonas Schäufler

Die Erweiterung einer Web-Application Firewall durch Reporting- und Monitoring Schnittstellen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus-Peter Kossakowski
Zweitgutachter: Prof. Dr.-Ing. Martin Hübner

Eingereicht am: 11. September 2015

Jonas Schäufler

Thema der Arbeit
Die Erweiterung einer Web-Application Firewall durch Reporting- und Monitoring Schnittstellen

Stichworte
IT-Sicherheit, Web-Application Firewall, ModSecurity, Verschlüsselung, SSL, TLS, ICAP

Kurzzusammenfassung
In dieser Arbeit wird eine Web-Application Firewall derart erweitert, dass sicherheitsrelevante Informationen über Clients und Server gewonnen werden können, die nicht aus der Verbindung auf dem Application-Layer zu entnehmen sind. Diese zusätzlich gewonnenen Informationen können dazu genutzt werden, administrativ einzelne kontaktierte Server zu blockieren oder User/Clients im eigenen Netz zu informieren oder zu blockieren.

Jonas Schäufler

Title of the paper
The extention of a web-application firewall with reporting- and monitoring functions

Keywords
IT-Security, Web-Application Firewall, ModSecurity, Encryption, SSL, TLS, ICAP

Abstract
This thesis describes an extention of a web-application firewall in order to gather security related information about clients and servers that can not be extracted from a connection on the application layer. This information can be used to block contacted servers and to report or block users inside the own network.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Ziel und Abgrenzung . . . 1
- 1.3 Zielgruppe . . . 2
- 1.4 Aufbau der Arbeit . . . 2

2 Grundlagen **4**
- 2.1 OWASP Top 10 Sicherheitsrisiken von Webanwendungen . . . 4
  - 2.1.1 Injection . . . 5
  - 2.1.2 Fehler in Authentifizierung und Session Management . . . 5
  - 2.1.3 Cross-Site Scripting (XSS) . . . 6
  - 2.1.4 Unsichere direkte Objektreferenzen . . . 7
  - 2.1.5 Sicherheitsrelevante Fehlkonfiguration . . . 7
  - 2.1.6 Verlust der Vertraulichkeit sensibler Daten . . . 8
  - 2.1.7 Fehlerhafte Autorisierung auf Anwendungsebene . . . 8
  - 2.1.8 Cross-Site Request Forgery (CSRF) . . . 8
  - 2.1.9 Verwendung von Komponenten mit bekannten Schwachstellen . . . 9
  - 2.1.10 Ungeprüfte Um- und Weiterleitungen . . . 9
- 2.2 Protokolle . . . 9
  - 2.2.1 Content Vectoring Protocol . . . 9
  - 2.2.2 Internet Content Adaption Protocol . . . 10
- 2.3 Web-Application-Firewall . . . 12
  - 2.3.1 Modes of Operation . . . 12
  - 2.3.2 Security Model . . . 12
  - 2.3.3 Protection Techniques . . . 13
  - 2.3.4 SSL . . . 13

3 Konzept für die Softwareentwicklung **15**
- 3.1 WAF Servermodule . . . 15
  - 3.1.1 ModSecurity . . . 15
  - 3.1.2 OWASP NAXSI Project . . . 17
- 3.2 Analyse . . . 18
  - 3.2.1 Zielbestimmung . . . 18
  - 3.2.2 Akteure . . . 18
  - 3.2.3 Nichtfunktionale Anforderungen . . . 18

3.2.4 Funktionale Anforderungen . . . 19
3.3 Architektur . . . 19
3.3.1 Proxy/WAF . . . 20
3.3.2 DiscoveryServer . . . 21
3.3.3 InfoServer . . . 23
3.3.4 ICAP-Server . . . 23

4 Implementierung **25**
4.1 Bibiliotheken und Frameworks . . . 25
4.2 Proxy . . . 26
4.2.1 Model . . . 27
4.2.2 View . . . 27
4.2.3 SSLCheck . . . 31
4.2.4 ICAP-Client . . . 32
4.3 DiscoveryServer . . . 32
4.3.1 ServeScript . . . 32
4.3.2 SSLCheckServer . . . 33
4.3.3 SSLFootprintServer . . . 34
4.3.4 DiscoveryScript . . . 35
4.4 InfoServer . . . 35
4.5 ICAP-Server . . . 36

5 Tests **38**
5.1 Testumgebung . . . 38
5.2 Testläufe . . . 39
5.2.1 Fallbeispiel 1: Blockieren eines Clients aufgrund Plugins . . . 40
5.2.2 Fallbeispiel 2: Informieren über Cipher Suite . . . 41
5.2.3 Fallbeispiel 3: Blockieren aufgrund von unsicheren SSL/TLS-Versionen 42
5.2.4 Fallbeispiel 4: Informationen über Cipher Suiten des Ziel-Servers . . . 42
5.2.5 Fallbeispiel 5: Blockieren von Ziel-Servern aufgrund von Blacklist-Eintrag 42
5.2.6 Fallbeispiel 6: Blockieren von Content-Types durch ICAP . . . 43
5.2.7 ModSecurity Test . . . 44
5.2.8 Die gefüllten Tabellen exemplarisch . . . 45

6 Fazit **50**
6.1 Zusammenfassung . . . 50
6.2 Ausblick . . . 52

Tabellenverzeichnis

2.1 OWASP Top 10 - 2013 4
2.2 Header der REQMOD und RESPMOD Methoden 10
2.3 Entitäten des Encapuslated-Headers 11

3.1 Systemfunktionen 19
3.2 Administratorfunktionen 19

4.1 Felder der Client Tabelle 28
4.2 Felder der Server Tabelle 29

Abbildungsverzeichnis

3.1 Architektur des Netzwerks 20
3.2 Client TLS/SSL Überprüfung 21
3.3 Sequenzdiagramm: Client Überprüfung 22
3.4 Sequenzdiagramm: Server Überprüfung 23

4.1 Ablauf eines HTTP Requests 26

5.1 Aufbau der Testumgebung 38
5.2 Daten der MIME-Type Tabelle 39
5.3 Daten der Plugin-Tabelle 40
5.4 Zum InfoServer weitergeleiteter Client 41
5.5 Blockierte Webseite 43
5.6 Blockierter Content-Type 44
5.7 Durch WAF blockierter Aufruf 45

Listings

2.1 Pseudocode als SQL-Injection Beispiel . . . 5
2.2 search.php: Beispiel eines für XSS anfälligen Codes . . . 6
2.3 search.php: Beispiel Aufruf . . . 6
2.4 include.php: Beispiel einer direkten Objektreferenz auf eine Datei . . . 7
2.5 Encapsulated-Header Beispiel . . . 11

4.1 Client Model . . . 27
4.2 Proxy URLPatterns . . . 30
4.3 Pseudocode der main Funktion . . . 30
4.4 Google Safe Browsing API Request . . . 30
4.5 SSLCheck Pseudocode . . . 31
4.6 ICAPreqmod Pseudocode . . . 32
4.7 Pseudocode DiscoveryServer . . . 33
4.8 Pseudocode runserver . . . 33
4.9 runallservers Beispiel . . . 34
4.10 Pseudocode SSLFootprintServer . . . 34
4.11 Pseudocode DiscoveryScript . . . 35
4.12 InfoServer Pseudocode . . . 35
4.13 REQMOD Methode Pseudocode . . . 36
4.14 RESPMOD Methode Pseudocode . . . 37

5.1 Benutzer mit bestimmtem Plugins Blockieren . . . 40
5.2 Benutzer über CipherSuite informieren . . . 41
5.3 Benutzer über veraltete SSL/TLS Version informieren und blockieren . . . 42
5.4 Server CipherSuiten für SSLv3 von google.com abfragen . . . 42
5.5 Zeus Status von benchblog.com . . . 43
5.6 Unzulässigen MIME-Type hinzufügen . . . 44
5.7 include.php . . . 45
5.8 Beispiel Befehle und Ausgabe in der Administrator Console . . . 46

1 Einleitung

1.1 Motivation

Seit mehreren Jahren steigt die Anzahl der neu entdeckten Sicherheitslücken und der Anteil der davon als kritisch eingestuften Lücken ist hoch. Das Internet erlaubt es, ortsunabhängig und mit wenig Kapital und Risiko, Angriffe durchzuführen. Zusätzlich bieten die zunehmende Vernetztheit, Smartphones, Tablets und neue Endgeräte des *Internet of Things* eine breite Angriffsfläche. Diese abstrakte und scheinbar allgegenwärtige Gefahr sowie die Enthüllungen von Edward Snowden über das Projekt PRISM haben zusätzlich dazu beigetragen, dass sich, in Anbetracht der Methoden und des Ausmaßes der Überwachung, hinsichtlich der Informationssicherheit ein Gefühl der Machtlosigkeit eingestellt hat. Dies führt zu einer "digitalen Sorglosigkeit", welche sich wiederum negativ auf die Gesamtsicherheit auswirkt. Durch die Verwendung veralteter Software, fehlende Sicherheitsupdates oder das unaufmerksame Verhalten der Benutzer ist es selbst Hobby-Hackern möglich, in Systeme von Unternehmen einzudringen und Schaden zu verursachen. Gefahr geht nicht nur von gezielten Angriffen aus Motiven wie Wirtschaftsspionage aus, sondern auch von Spam-Mails oder Drive-By Downloads, die kein gezieltes Opfer haben. [BSI14]

In Unternehmen, Bildungseinrichtungen und anderen Organisationen wird zunehmend gestattet, dass Benutzer eigene Endgeräte mitbringen und an das interne Netzwerk anschließen. Dieses *Bring your own device* genanntes Konzept bringt der jeweiligen Institution ökonomische und ökologische Vorteile und erhöht die Produktivität deren Mitglieder, stellt jedoch ein Sicherheitsrisiko dar. Nahezu jedes mobile Endgerät besitzt einen Browser und das Web bietet viele Möglichkeiten ein System zu kompromittieren und an Informationen zu gelangen die geheim gehalten werden sollten. Jedoch gibt es keine geeigneten Werkzeuge um diese Endgeräte zu kontrollieren. [Sic13]

1.2 Ziel und Abgrenzung

Module wie ModSecurity für Apache oder NAXSI für nginx sind Firewalls auf Anwendungsebene, die den Inhalt der Anfragen und Antworten der Webserver auf verdächtige Inhalte

überprüfen, um gängige Angriffstechniken zu erschweren und gezielt Angriffe auf bekannte Schwachstellen zu verhindern. Andere Web Application Firewalls werden als Reverse Proxy oder Appliance in ein Netzwerk eingebunden. Jedoch werden hier nicht alle zur Verfügung stehenden Informationen ausgewertet, an welchen Sicherheitsrisiken erkannt werden könnten. Das Ziel dieser Arbeit ist die Entwicklung einer Software, um über HTTP- und HTTPS-Verbindungen sicherheitsrelevante Informationen über Clients und Server zu sammeln, und diese direkt zum Schutz von Anwendern und deren Organisationen zu nutzen. Weiterhin sollen solche Informationen für Administratoren verfügbar sein, um weiterführende Reaktionen zu ermöglichen.

1.3 Zielgruppe

Diese Arbeitet richtet sich an Studenten im Bereich der Informatik, Web-Programmierer sowie Personen, die für die Sicherheit eines Netzes, mit entsprechender mit Web-Infrastruktur, verantwortlich sind.

1.4 Aufbau der Arbeit

In Kapitel 2 werden die Grundlagen für diese Arbeit behandelt. Zu Beginn werden die verbreitesten Angriffstechniken (anhand OWASP Top Ten) auf Web-Applikationen vorgestellt. Die Funktionsweise dieser Angriffe liefert inhärent auch das Rüstzeug für Gegenmaßnahmen, die im Einzelfall ebenfalls diskutiert werden. Grundfrage: Welchen Typen von Angriffen muss begegnet werden. Kapitel 2 geht zudem auf die Protokolle ICAP (Internet Content Adaption Protocol) und CVP (Content Vectoring Protocol) ein. Am Ende von Kapitel 2 wird das Konzept einer Web-Application Firewall vorgestellt.

Kapitel 3 befasst sich mit der Architektur der in dieser Arbeit entwickelten Software. Es werden die Kernfunktionen des Prototypen vorgestellt. Es werden die verschiedenen Komponenten mit Hilfe von Diagrammen beschrieben und deren Funktion und deren Zusammenspiel erläutert.

Kapital 4 beschreibt die Implementierung des Prototypen. Kernmethoden und Schnittstellen der Komponenten werden anhand von Pseudocode und Diagrammen erläutert, die verwendeten Datenstrukturen besprochen.

Kapitel 5 deckt die notwendigen Tests ab. Hier wird zunächst der Aufbau der Testumgebung beschrieben. Anschließend werden an Fallbeispielen die implementierten Funktionen getestet und dokumentiert.

Abschließend wird die Arbeit in Kapitel 6 zusammengefaßt. Um den hier gebauten Prototypen in eine reale Anwendung integrieren zu können, müssen noch substanzielle Überlegungen angestellt werden. Kapitel 6 widmet sich diesen Integrations-TODOs.

2 Grundlagen

2.1 OWASP Top 10 Sicherheitsrisiken von Webanwendungen

Alle drei Jahre veröffentlich das Open Web Application Security Project (OWASP) eine Liste der zehn häufigsten Sicherheitsrisiken für Webanwendungen. Wie hoch ein Risiko bewertet wird ist abhänging von vier Faktoren:

- Wahrscheinlichkeit, dass die Schwachstelle in einer Applikation vorhanden ist.
- Wahrscheinlichkeit, dass ein Angreifer die Schwachstelle entdeckt.
- Wahrscheinlichkeit, dass ein Angreifer die Schwachstelle erfolgreich ausnutzt.
- Die technischen Auswirkungen bei erfolgreichem Ausnutzen der Schwachstelle.

Tabelle 2.1 zeigt die OWASP Top 10 Sicherheitsrisiken für das Jahr 2013 und gibt deren Rang in den vorhergehenden Veröffentlichungen an.

Rang	**Sicherheitsrisiko**	**2010**	**Trend**
1	Injection	1	=
2	Fehler in Authentifizierung und Session-Management	3	↑
3	Cross-Site Scripting (XSS)	2	↓
4	Unsichere direkte Objektreferenzen	4	=
5	Sicherheitsrelevante Fehlkonfiguration	6	↑
6	Verlust der Vertraulichkeit sensibler Daten	7/9	↑
7	Fehlerhafte Autorisierung auf Anwendungsebene	8	↑
8	Cross-Site Request Forgery (CSRF)	5	↓
9	Nutzung von Komponenten mit bekannten Schwachstellen	-	neu
10	Ungeprüfte Um- und Weiterleitungen	10	=

Tabelle 2.1: OWASP Top 10 - 2013

Die Kategorien *Unzureichende Absicherung der Transportschicht* und *Unzureichende Absicherung auf der Transportschicht* (welche 2010 den siebten und neunten Platz belegten), sind 2013 zur neuen Kategorie *Verlust der Vertraulichkeit sensibler Daten* zusammengefasst worden.

Die Entdeckung von Angriffen wie POODLE, FREAK, Logjam oder Heartbleed weisen darauf hin, dass im letzten Jahr die Transportschicht im Fokus der Forscher lag. Auch die Veröffentlichung von Schwachstellen der Kategorie *Fehler in Authentifizierung und Session-Management* ist seit 2010 gestiegen. In den Top 10 von 2007 noch auf Platz 7, belegt diese Kategorie nun den zweiten Platz, vor Cross-Site Scripting.

Die in 2.1 aufgelistetet Arten von Risiken werden den in folgenden Abschnitten genauer beschrieben. [Pro13] [Pro10]

2.1.1 Injection

Injection-Schwachstellen entstehen, wenn eine Applikation nicht vertrauenswürdige Daten verwendet, um Kommandos in einem Interpreter auszuführen. Werden die Eingabedaten nicht validiert, kann die (daraus formatierte) Eingabe des Interpreters so manipuliert werden, dass unautorisiert auf Daten zugegriffen werden kann, oder dass Kommandos ausgeführt werden, die nicht vorhergesehen sind. Injection-Schwachstellen sind sehr einfach auszunutzen und können schwerwiegende Auswirkungen haben. Obwohl die Schwachstelle im Code einfach zu erkennen ist, tritt sie häufig auf und ist seit 2013 Platz 1 der OWASP Top 10.
Als Beispiel ist in Listing 2.1 Pseudocode dargestellt der, für die Authentifizierung eines Benutzers, einen Datenbank-Eintrag anfordert.

```
1 $QUERY = "SELECT user FROM member_table
2         WHERE username = '" + $NAME + "'
3          AND password = '" + $PASSWORD + "'
4         ;"
5 runSQLQuery($QUERY)
```

Listing 2.1: Pseudocode als SQL-Injection Beispiel

Die Variablen *$NAME* und *$PASSWORD* sind die vom Benutzer übergebenen Anmeldedaten. Nimmt die String-Variable *$NAME* den Wert [*Administrator';--*] an, wird die AND-Klausel, in SQL, auskommentiert. Enthält die Variable *$PASSWORD* den Text [*password' OR '1'<'2*] würde die Klausel in jedem Fall als Wahr interpretiert werden. Ein Angreifer könnte sich so als einen beliebigen existierenden Benutzer authentifizieren.

2.1.2 Fehler in Authentifizierung und Session Management

Da HTTP ein zustandsloses Protokoll ist, muss (für die Wiedererkennung bereits authentifizierter Benutzer) mit jeder Anfrage ein Sitzungs-Token übertragen werden. Gelingt es einem Angreifer an ein gültiges Sitzungs-Token zu gelangen, kann er ohne Authentifizierung die

Identität des Benutzers annehmen. Zu der Kategorie *Fehler in Authentifizierung und Session-Management* zählt nicht nur die unsichere Übertragung der Sitzungstokens und Anmeldedaten, sondern auch deren unverschlüsselte Speicherung, sowie fehlerhafte Funktionen zur Wiedererkennung und Erstellung des Benutzers, Änderung des Passwords oder Abmeldung. Die Kompromittierung eines priviligierten Benutzers stellt ein hohes Risiko dar.

2.1.3 Cross-Site Scripting (XSS)

Cross-Site Scripting ist eine Unterkategorie der Injection-Schwachstelle. Sie ist die häufigste Art von Schwachstelle in Webanwendungen und leicht von einem Angreifer zu entdecken. An den Benutzer zurückgegebene Eingabedaten werden so manipuliert, dass der Browser sie als ausführbaren Code interpretiert. Bei den meisten Cross-Site Scripting Angriffen handelt es sich um JavaScript-Code, da dieser von fast allen Browsern standardmäßig unterstüzt wird. Andere clientseitige Scriptsprachen die für XSS verwendet werden sind VBScript, ActiveX, Flash oder auch einfaches HTML. Die Auswirkunge sind vielfältig. Mit eingeschläustem Code kann auf Cookies zugegriffen werden um an Sitzungs-Tokens zu gelangen, es kann auf Phishing- oder Malware verbreitende Seiten umgeleitet werden, oder einer Seite können sonstige ungewollte Inhalte hinzugefügt werden.
Beispielsweise könnte eine Suchmaschine nach einer Suche den, vom Client eingegebenen, Suchbegriff auf der Seite anzeigen. In Listing 2.2 ist ein Minimalbeispiel dargestellt, bei welchem der Suchbegriff per GET in der URL übergeben wird.

```
1 <?php
2 ...
3 echo $_GET["keyword"];
4 ...
5 ?>
```

Listing 2.2: search.php: Beispiel eines für XSS anfälligen Codes

Als Suchbegriff eingegebener Script-Code wird so zum Client zurückgesendet und, sofern der Browser ihn unterstützt, ausgeführt. Der Suchbegriff wird dem Server als Parameter in der URL übergeben. Wird ein Benutzer, durch das Unterschieben eines Links, dazu gebracht eine URL der in Listing 2.3 abgebildeten Form aufzurufen, kann ihm mit dem Link beliebiger dann clientseitig ausgeführter Script-Code untergeschoben werden.

```
1 http://www.example.net/search.php?keyword=
2         <script>alert(hello)</script>
```

Listing 2.3: search.php: Beispiel Aufruf

Das übergebene Scirpt, kann so verschleiert werden, dass der Nutzer nicht erkennen kann ob es sich um Code handelt. Zum Beispiel durch die Dekodierung der ASCII Zeichen in ihre Hex-Werte.

2.1.4 Unsichere direkte Objektreferenzen

Unsichere direkte Objektreferenzen treten auf wenn Object Identifier für serverseitige Resourcen vom Benutzer direkt übergeben werden. Dabei kann es sich zum Beispiel um Dateinamen oder Schlüssel von Datenbankeinträgen handeln. Diese Art von Schwachstelle ist einfach aufzufinden und auszunutzen, da direkte Referenzen auf Resourcen, zum Beispiel in der URL, oft leicht erkannt und manipuliert werden können. In Listing 2.4 ist php-Code als Beispiel aufgeführt, der statischen Inhalt (aus einer Datei) in eine Seite einbinden soll. Hier wird der Dateiname direkt referenziert und in der URL, mit dem *content*-Parameter, übergeben.

```
1 <?php
2 ...
3 include $_GET["content"];
4 ...
5 ?>
```

Listing 2.4: include.php: Beispiel einer direkten Objektreferenz auf eine Datei

Ein Link auf einer Seite die ähnlichen Code wie in Listing 2.4 verwendet könnte folgendes Format haben:

```
1 http://www.somepage.com/include.php?content=faq.html
```

Durch Anpassung des *content*-Parameters können beliebige Dateien eingebunden werden. Dies können lokale Dateien sein, die Passwörter enthalten könnten, oder Dateien anderer Webserver, welche Schadecode verbreiten. Man spricht von *Local File Inclusion (LFI)* beziehungsweise *Remote File Inclusion (RFI)*.

2.1.5 Sicherheitsrelevante Fehlkonfiguration

Oft sind Standardeinstellung von Anwendungen nicht für den laufenden Betrieb geeignet. Standardkonten, deren Passwort nicht geändert wurde, können automatisiert ausfindig gemacht werden und können zur Kompromittierung des Systems führen. Nicht unterdrückte Fehlermeldungen können sicherheitsrelevante Informationen, wie zum Beispiel Felder oder Tabellennamen von Datenbanken, preisgeben. Zur Kategorie *Sicherheitsrelevante Fehlkonfiguration* gehört auch das serverseitige Zulassen von zwar performanteren aber unsicherer Cipher Suiten bei der SSL/TLS-Konfiguration (siehe auch nächster Abschnitt).

2.1.6 Verlust der Vertraulichkeit sensibler Daten

Die gänzlich fehlende Verschlüsselung vertraulicher Daten ist sicher die häufigste Schwachstelle in dieser Kategorie. Dennoch gehören hierzu auch Schwachstellen wie die Verwendung schwacher Cipher Suiten oder Hashing Algorithmen. Schwachstellen dieser Art kommen oft in Verbindung mit anderen Angriffen, wie Man-in-the-Middle oder SQL-Injections, zum Tragen. Aus Sicht eines Unternehmens, das im Netz seine Geschäfte tätigen möchte, ist das Zulassen älterer Clients durchaus wünschenswert (es möchte keine potentiellen Kunden ausschließen). Aus Sicht des Kunden/Clients ist eine https:// Verbindung aktiv, aber sie könnte unbemerkt mit bereits als eher unsicherer SSL-Version oder mit schwacher Cipher Suite aufgebaut worden sein.

2.1.7 Fehlerhafte Autorisierung auf Anwendungsebene

Durch fehlende oder fehlerhafte serverseitige Prüfung bei Zugriff auf eine privilegierte URL wird es Angreifern erlaubt, Funktionen zu nutzen, für die sie nicht berechtigt sind. Um Schwachstellen dieser Art auszunutzen genügt oft eine Änderung der URL. Die Folge ist, dass der Angreifer auf privilegierte Seiten gelangt. Diese Art von Schwachstellen lassen sich leicht entdecken und ausnutzen, jedoch besteht die Schwierigkeit darin, herauszufinden welche angreifbaren URLs existieren.

Zum Beispiel könnte die Datei *adduser.php* Code enthalten, um einen neuen Benutzer anzulegen. Auch wenn ein Link auf diese Seite nur durch den Administratorenbereich zugänglich ist, muss innerhalb der Datei trotzdem eine serverseitige Prüfungen erfolgen. Angreifer könnten sonst durch Aufrufen folgender URL einen neuen Benutzer hinzufügen.

```
1 http://www.somepage.com/adduser.php?user=new&pw=password
```

2.1.8 Cross-Site Request Forgery (CSRF)

Je nach Verfahren müssen Sitzungstokens oder Anmeldedaten(bei jeder HTTP Anfrage innerhalb einer Sitzung) an den Server gesendet werden. Die Zuordnung der Daten erfolgt automatisch durch den Browser je nach Zielhost der Anfrage. Bei einem CSRF-Angriff wird der Browser des Opfers dazu gebracht, eine (vom Angreifer definierte) Anfrage zu senden. Eine Anfrage kann durch einen Link oder clientseitigen Script-Code, der durch XSS oder das Aufrufen einer bösartigen Seite ausgeführt wird, untergeschoben werden. Hat das Opfer beim Host der untergeschobenen Anfrage eine aktive Sitzung, können so Aktionen mit den Privilegien des jeweiligen Nutzers durchgeführt werden.

2.1.9 Verwendung von Komponenten mit bekannten Schwachstellen

Diese Kategorie war 2006 noch Teil von *Sicherheitsrelevante Fehlkonfiguration*, wurde jedoch wegen sehr häufiger Verbreitung als eigene Kategorie eingeführt. Durch die steigende komplexität von Software werden bei der Entwicklung oft nicht alle Abhängigkeiten auf ihre Sicherheit überprüft. Dies führt dazu, dass alte Versionen von Komponenten verwendet werden, die bekannte Schachstellen aufweisen.

2.1.10 Ungeprüfte Um- und Weiterleitungen

Nutzt eine Seite Eingaben des Benutzers für eine Um- oder Weiterleitung, ohne sie zu prüfen, kann dies ausgenutzt werden, um Benutzer auf eine bösartige Seite, oder den Angreifer auf eine privilegierte Seite, umzuleiten. Schwachstellen dieser Kategorie sind einfach aufzufinden, da URLs als Parameter leicht zu erkennen sind. Findet keine serverseitige Prüfung statt, kann ein Benutzer durch Unterschieben folgender URL auf *evil.net* umgeleitet werden,

```
1 http://www.somepage.com/redirect.php?url=evil.net
```

Interne Weiterleitungen können dazu führen, dass ein Angreifer auf privilegierte Seiten zugreifen kann, da die Zugangskontrolle umgangen wird.

```
1 http://www.somepage.com/forward.php?url=admin.php
```

2.2 Protokolle

2.2.1 Content Vectoring Protocol

Normale Firewalls, auf Netzwerkebene, überprüfen keine Pakete auf höhergelegenen Schichten. Das Content Vectoring Protocol (CVP) wurde entworfen, damit Firewalls Daten von Nachrichten, der Protokolle auf Anwendungsschicht, an dazugehörige Content-Inspection Server weiterleiten können, um nach Schadcode zu suchen. Häufige Anwendungsprotokolle sind HTTP, FTP und SMTP. Es können aber auch Daten von TCP-Nachrichten, unabhängig vom Anwendungsschicht-Protokoll, an generische Content-Inspection Server weitergeleitet werden. Die erste Spezifikation des Protokolls erfolgte 1996 durch das Softwareunternehmen Check Point. Heute ist CVP, in Form einer API, Teil des Open Platform for Security (OPSEC) Frameworks. Diese API erlaubt es Dienste und Anwendungen, für Content-Inspection, von Drittanbietern in Check Points kommerzielle Lösungen, wie der Firewall-1, zu integrieren. Da viele Funktionen dieser API für diese Software nicht relevant sind, macht die hohe Komplexi-

tät und Abhängigkeiten von anderen Bibiliotheken das *ICAP* Protokoll zu einer geeigneten Alternative.

[Che06]

2.2.2 Internet Content Adaption Protocol

Für diese Arbeit wurde das IACP-Protokoll zur Weiterleitung von HTTP-Requests an einen ICAP-Server gewählt. Daher soll hier auf die ICAP-Spezifikationen intensiver eingegangen werden.

Das Internet Content Adaption Protocol (ICAP) ist ein zustandsloses Protokoll der Anwendungsebene nach dem Request-Response Prinzip. Ein ICAP-Server stellt Funktionen zur Manipulation von HTTP-Nachrichten zur Verfügung. Das Weiterleiten der möglicherweise modifizerten HTTP-Nachrichten, an Clients und Quellserver, liegt in den Händen des ICAP-Clients. ICAP-Nachrichten entsprechen dem Internet Message Format. Sie sind also unterteilt in Statuszeile, Header und Body, in welchem sich die Nutzdaten befinden. Ursprünglich sollte es im Protokollstapel über HTTP angesiedelt werden, jedoch hätten so einige Features nicht, oder nur sehr umständlich, realisiert werden können. Es ist nun an HTTP angelehnt und läuft über TCP. Die meisten ICAP-Header werden durch die Spezifikation von HTML 1.1 definiert.

Header	**Request**	**Response**	**ICAP spezifisch**
Cache-Control	X	X	
Connection	X	X	
Date	X	X	
Expires	X	X	
Pragma	X	X	
Trailer	X	X	
Upgrade	X	X	
Encapsulated	**X**	**X**	**X**
Allow	X		
From	X		
Host	X		
Referer	X		
User-Agent	X		
Preview	**X**		**X**
Server		X	
ISTag		**X**	**X**

Tabelle 2.2: Header der REQMOD und RESPMOD Methoden

Tabelle 2.2 zeigt die zulässigen Header für die Methoden REQMOD und RESPMOD. Bis auf die ICAP spezifischen Header *Encapsulated*, *Preview* und *ISTag* sind deren Funktionen die selben wie in HTTP 1.1.
Es wird unterschieden zwischen der Bearbeitung von HTTP-Anfragen (Request Modification) und Antworten (Response Modification). ICAP besitzt drei Methoden: REQMOD, für die Modifikation von HTTP-Anfragen, RESPMOD, für die Modifikation von Antworten, und OPTIONS, um die Konfiguration des Servers abzufragen. Dass ein ICAP-Server die Methoden REQMOD und RESPMOD anbietet ist optional, OPTIONS muss jedoch von jedem ICAP-Server zur Verfügung gestellt werden. Außerdem erlaubt es das Protokoll, dass Server benutzerdefinierte Methoden anbieten.

Der *Encapsulated* Header muss in jeder ICAP-Nachricht vorhanden sein. Er zeigt welche Teile einer HTTP-Nachricht im ICAP-Body gekapselt sind und gibt deren Position, relativ zu Begin des Bodies, in Byte an. Listing 2.5 zeigt ein Beispiel eines zulässigen *Encapsulated* Headers einer ICAP-Nachricht, deren Body Header einer HTTP-Anfrage enthält, wobei die Gesamtgröße der Header 514 Bytes beträgt:

```
1 Encapsulated: req-hdr=0, null-body=514
```

Listing 2.5: Encapsulated-Header Beispiel

Die folgende Tabelle 2.3 zeigt die erlaubten Entitäten des Encapsulated-Headers für die jeweiligen Methoden.

Entität	**REQMOD**		**RESPMOD**		**OPTIONS**	
	Anfrage	**Antwort**	**Anfrage**	**Antwort**	**Anfrage**	**Antwort**
req-hdr	X	X	X			
req-body	X	X				
res-hdr		X	X	X		
res-body		X	X	X		
opt-body						X
null-body	X	X	X	X	X	X

Tabelle 2.3: Entitäten des Encapuslated-Headers

Die *null-body* Direktive wird verwendet, wenn kein gekapselter Body in der ICAP-Nachricht vorhanden ist. Gekapselte Bodys müssen mit dem *Chunked Transfer Coding* übermittelt werden, wie es für HTTP spezifiziert ist. Zum Beispiel kann die Antwort, auf eine Anfrage nach einer Request Modification, HTTP-Response-Header und -Body enthalten. Vom ICAP-Client wird dann erwartet, dass diese an den HTTP-Client übermittelt wird ohne einen Quellserver zu kontaktieren. Das Protokoll bietet die Möglichkeit, dass Server die Bearbeitung einer

Anfrage abbrechen, bevor der komplette gekapselte Body übermittelt wird, sofern es vom Client angeboten wird. Dazu muss bei einer Anfrage der Preview Header verwendet, und die maximale Größe des Previews in Byte angegeben werden. Daraufhin sendet der Client die gekapselten Header und maximal die angegebene Anzahl an Bytes des gekapselten Bodies. Wenn der vollständige gekapselte Body im Preview enthalten ist, muss im terminierenden Chunk die ICAP-spezifische Chunk-Extention *ieof\r\n\r\n* verwendet werden. [IET03]

2.3 Web-Application-Firewall

Eine Web-Application-Firewall (WAF) ist ein Sicherheitssystem, das Nachrichten des HTTP-Protokolls, also auf Anwendungsebene, analyisert und Sicherheitsmechanismen implementiert, die technisch unabhängig von der zu schützenden Web-Anwendung sind. Die Art der Integration bestimmt deren Funktionstyp. Die wesentlichen Parameter einer Web-Application Firewall werden im folgenden dargestellt. [Con06] [ICS14]

2.3.1 Modes of Operation

Es werden vier verschiedene *Modes of Operation* für WAFs unterschieden, je nach dem auf welcher Schicht sie im Netzwerk eingebunden werden.

1. Bridge: Die WAF wird auf der Sicherungsschicht in das Netzwerk eingebunden. In diesem Modus ist die WAF für darüber liegende Schichten transparent und erfordert keine weitere Konfiguration des Netzwerks.

2. Router: Das Netzwerk wird auf Vermittlungsschicht so konfiguriert, dass Anfragen an die WAF weitergeleitet werden.

3. Reverse Proxy: Die WAF wird als Reverse Proxy verwendet. Die Umleitung der Anfragen erfolgt per DNS oder Routing auf Vermittlungsschicht.

4. Teil des Webservers: WAF ist ein Modul oder monolithisch integriert in den abgefragten Webserver.

2.3.2 Security Model

Das Security Model beschreibt das Verhalten der WAF im Umgang mit Transaktionen und die Herangehensweise bei der Erkennung von Angriffen.

Negative Security Model: Standardmäßig sind alle Transaktionen erlaubt. Dieses Model basiert darauf, bösartige Anfragen zu blockieren (Blacklisting). Angriffe werden mit Hilfe von Pattern-Matching von Signaturen und generischen Regeln erkannt. Signaturen und Regeln sind angriffsspezifisch. Es wird also Wissen über den Angriff bzw die Art von Angriff benötigt, gegen den das System geschützt werden soll.

Positive Security Model: Standardmäßig sind alle Transaktionen blockiert. Regeln dieses Models beschreiben, welche Transaktionen erlaubt sind. Dies erfordert Wissen über die zu schützende Anwendung. Die Erstellung dieser Regeln kann teilweise automatisiert werden (Training), jedoch ist es nahezu unmöglich, jeden Fall abzudecken.

2.3.3 Protection Techniques

WAFs werden nicht nur verwendet, um spezifische Angriffe auf generische Art zu erkennen und zu verhindern, sondern sind auch geeignet, um zusätzliche Schutzmechanismen zu implementieren:

Brute-Force Erkennung: An der hohen Wiederhohlungsrate von Requests können Brute-Force Angriffe und automatisierte Clients erkannt und blockiert bzw verlangsamt werden.

Cookie Schutz: Cookies können von einer WAF signiert, verschlüsselt oder versteckt werden (Virtualisierung) um deren Manipulation zu erkennen bzw zu verhindern.

Session-Management: Sitzungs-Tokens können von der WAF mit Virtualisierung geschützt werden oder mit anderen Daten wie SSL Sitzungstokens oder IP-Adressen der Clients verknüpft werden, um Angriffe zu erkennen.

2.3.4 SSL

Die Verschlüsselung der Transaktion mittels TLS/SSL ist essentiell für eine sichere Übertragung im Web. Da der WAF die Daten in unverschlüsseltem Zustand vorliegen müssen, bestehen zwei Möglichkeiten:

- Terminierend: Alle serverseitigen SSL Operationen (und Zertifikate der Webserver) werden auf die WAF ausgelagert. Die WAF kann für die Transaktion mit dem Webserver eine neue SSL Verbindung aufbauen. (aktiv)

- Tunnel: Serverseitige SSL-Operationen tätigt der Webserver. Die Transaktion auf Anwendungsebene wird, mit dem Private-Key des Webservers, entschlüsselt. (passiv)

3 Konzept für die Softwareentwicklung

Die prinzipielle Vorgehensweise besteht darin, eine vorhandene Web-Application Firewall einzurichten und diese um weitere Funktionen zu ergänzen. Ich habe mich für WAFs entschieden, die im Kontext eines Webservers laufen, der das Common-Gateway-Interface (CGI) unterstützt. Innerhalb dieser Einschränkungen bleibt die entwickelte Software kompatibel zu anderen Servern und Modulen. Die Funktionsweise und die Detection-Mechanismen der WAF bleiben unangetastet. In einem ersten Schritt sollen daher zwei WAF-Implementierungen vorgestellt werden. Darauf aufbauend werden die Anforderungen für eine Funktionserweiterung der gewählten WAF abgeleitet. Diese Funktionserweiterungen sind gleichzeitig die Anforderungen für die in dieser Arbeit erstelle Software.

3.1 WAF Servermodule

3.1.1 ModSecurity

ModSecurity ist ein plattformunabhängiges Modul für die Webserver Apache, Nginx und IIS. Es ist Open Source und steht unter der Apache 2.0 Lizenz. Eher ein WAF Framework als eine WAF, bietet das Modul von selbst keinen Schutz und implementiert keine Sicherheitsmechanismen. Das Modul bietet jedoch volle Einsicht in HTTP Anfragen und Antworten bevor diese an Client beziehungsweise Server weitergeleitet werden. ModSecurity betreibt nicht nur Buffering der Nachrichten um diese vollständig untersuchen zu können bevor sie weitergereicht werden, sondern parst auch Formate wie XML oder JSON um Information zu extrahieren. Die *Rule Engine* wendet die vom Administrator konfiguierten Regeln auf die vorhanden Daten an und entscheidet wie mit der Transaktion im weiteren umgegangen wird.
ModSecurity Regeln haben folgende Sytax:

```
1 SecRule VARIABLES OPERATOR ACTIONS
```

Folgende Regel überprüft mittels Regular Expressions alle Request Parameter auf den *<script>* Tag. Wird das Pattern erkannt, wird die Transaktion blockiert und der HTTP Status Code 404 (Page Not Found) an den Client gesendet.

```
SecRule ARGS "<script>" log,deny,status:404
```

Dies ist jedoch eine stark vereinfachte Regel, um vor script-Tag basierten XSS Attacken zu schützen. Effektive Regeln erreichen hohe Komplexität, was wiederum ein Nachteil von ModSecurity darstellt, da Personal mit geeignetem Fachwissen für die Konfiguration benötigt wird oder Regeln von Drittanbietern erworben werden müssen.

Standardmäßig ist bei einer Installation von ModSecurity das OWASP ModSecurity Core Rule Set (CRS) enthalten, um vor den geläufigsten Angriffen zu schützen. Als Beispiel einer effektiven Regeln folgt die verwendete Regel des OWASP ModSecurity CRS um vor Script-Tag basierten XSS Angriffen zu schützen:

```
SecRule ARGS "(?i)(<script[^>]*>[\s\S]*?<\/script[^>]*>|
<script[^>]*>[\s\S]*?<\/script[[\s\S]]*[\s\S]|
<script[^>]*>[\s\S]*?<\/script[\s]*[\s]|
<script[^>]*>[\s\S]*?<\/script|<script[^>]*>[\s\S]*?)"
"id:'973336',phase:2,rev:'1',
ver:'OWASP_CRS/2.2.9',maturity:'1',accuracy:'8',t:none,
t:urlDecodeUni,t:htmlEntityDecode,t:jsDecode,t:cssDecode,log,capture,
msg:'XSS Filter - Category 1: Script Tag Vector'"
```

Bevor das Pattern auf die Argumente angewandt wird, erfolgen verschiedene Transformationen um gängige Umgehungstechniken mit Encoding zu verhindern. Ab der Version 2.5 von ModSecurity ist es außerdem möglich mit der Scriptsprache Lua Regeln zu schreiben, um Techniken zu implementieren, die mit der normalen Rule Engine von ModSecurity nicht realisiert werden können.

[Ris13]

3.1.2 OWASP NAXSI Project

Eine weitere WAF in Form eines Servermodules ist das OWASP Project NAXSI (Nginx Anti XSS SQL Injection) für den Nginx Webserver. NAXSI soll Webserver gegen Angriffe wie SQL Injections, Cross Site Scripting, Request Forgery und File Inclusion schützen. Es ist ausgelegt auf hohe Performanz und geringem Konfigurationsaufwand der Regeln. Es funktioniert nach einem positiven Security Model. Ein Minimum an generischen Regeln (Core-Rule) blockiert alle als möglichen Angriff erkannte Anfragen. Und NAXSI versucht den Konfigurationsaufwand der Regeln durch ein positives Security Model (White Listing) mit automatisierter Regelerstellung, zu vermindern.

Folgende Core-Rule blockiert Anfragen in welchen SQL Schlüsselwörter in Body, URL, Parametern, Headern oder Cookies erkannt werden:

```
1 MainRule
2 "rx:select|union|update|delete|insert|
3         table|from|ascii|hex|unhex|drop"
4 "msg:sql keywords"
5 "mz:BODY|URL|ARGS|$HEADERS_VAR:Cookie" "s:$SQL:4" id:1000;
```

Das *rx* Schlüsselwort gibt an, dass es sich hierbei um Regular-Expression und kein einfaches String matching handelt. Jeder Core-Rule ist als Unique Identifier eine id zugewiesen. Die Regeln der White-List deaktivieren die nötigen Core-Rules für das in der Regel beschriebene Request.

```
1 BasicRule wl:1000,1015 "mz:$BODY_VAR:submit";
```

Diese Whitelist Regel für Wordpress deaktiviert Core-Rules 1000 und 1015 bei POST-Parametern mit dem Namen *submit*.

[Nax15]

3.2 Analyse

Eine Web Application Firewall, oder allgemeiner ein Proxy, ist eine gute Möglichkeit um Daten aus einer HTTP-Transaktion zu gewinnen. Die Möglichkeit (in zentraler Position) in eine Verbindung einzugreifen, kann genutzt werden um Informationen über Clients und Server zu gewinnen, die nicht durch das Beobachten der Transaktionen zu erschließen, aber relevant für die Sicherheit des Netzwerks und dessen Benutzern sind.

3.2.1 Zielbestimmung

Eine Web Application Firewall soll erweitert werden um Informationen über Client und Server zu gewinnen, welche nicht aus der Verbindung extrahiert werden können. Diese sollen Administratoren möglichst leicht zugänglich sein. Außerdem sollen diese in der Lage sein, den Nutzer über unsichere Konfigurationen zu informieren.

3.2.2 Akteure

Client: HTTP-Clients des Netzwerks. Der Browser des Benutzers.
Administrator: Der Adminstrator des Netzwerks / Systems.
Webserver: Interne Webserver, welche durch die WAF geschützt sind.
Externe Webserver: Server im WAN, die von internen Clients kontaktiert werden,

3.2.3 Nichtfunktionale Anforderungen

Korrektheit: Die Verwendung der Software sollte die Funktion von aufgerufenen Webseiten oder Applikationen nicht einschränken.
Skalierbarkeit: Da das System ein Flaschenhals für HTTP/S Verbindungen schafft, sollte es skalierbar sein, um eine steigende Anzahl von Anfragen bewältigen zu können
Portierbarkeit: Die Software sollte leicht auf andere Betriebssysteme portiert werden können.
Leistung / Benutzbarkeit: Die Software sollte die Bearbeitungszeit einer Anfrage nicht so verlängern, dass der Benutzer gehindert wird.

3.2.4 Funktionale Anforderungen

Tabelle 3.1: Systemfunktionen

A1.1	Das System ist in der Lage sicherheitsrelevante Informationen über die Clients herauszufinden.
A1.1.1	Betriebssystem
A1.1.2	Browser Typ
A1.1.3	Browser Version
A1.1.4	Unterstützte SSL/TLS Versionen
A1.1.5	Unterstützte CipherSuiten
A1.1.6	Verwendete Browser Plugins
A1.1.7	Version verwendeter Browser Plugins
A1.1.8	Unterstützte MIME-Typen
A1.2	Das System ist in der Lage sicherheitsrelevanten Informationen über die kontaktierten Server herauszufinden.
A1.2.1	Unterstützte SSL/TLS Versionen
A1.2.1	Unterstützte CipherSuiten
A1.2.3	Reputation bei vertrauenswürdigen Dritten
A1.3	Das System speichert alle gewonnen Informationen in einer Datenbank
A1.4	Das System kann in HTTP-Transaktionen eingreifen
A1.4.1	Anfragen blockieren
A1.4.2	Anfragen modifizieren
A1.4.3	Antworten blockieren
A1.4.4	Antworten modifizieren

Tabelle 3.2: Administratorfunktionen

A2.1	Der Administrator kann die Informationen über Client und Server abfragen
A2.2	Der Administrator kann Clients blockieren
A2.3	Der Administrator kann Zugriffe auf Server blockieren
A2.4	Der Administrator kann Benutzer informieren

3.3 Architektur

Ein Webserver mit WAF Modul wird in einen Proxy umfunktioniert. Für diese Bachelorarbeit wurde Apache mit dem ModSecurity gewählt. In Abbildung 3.1 sind die verschiedenen

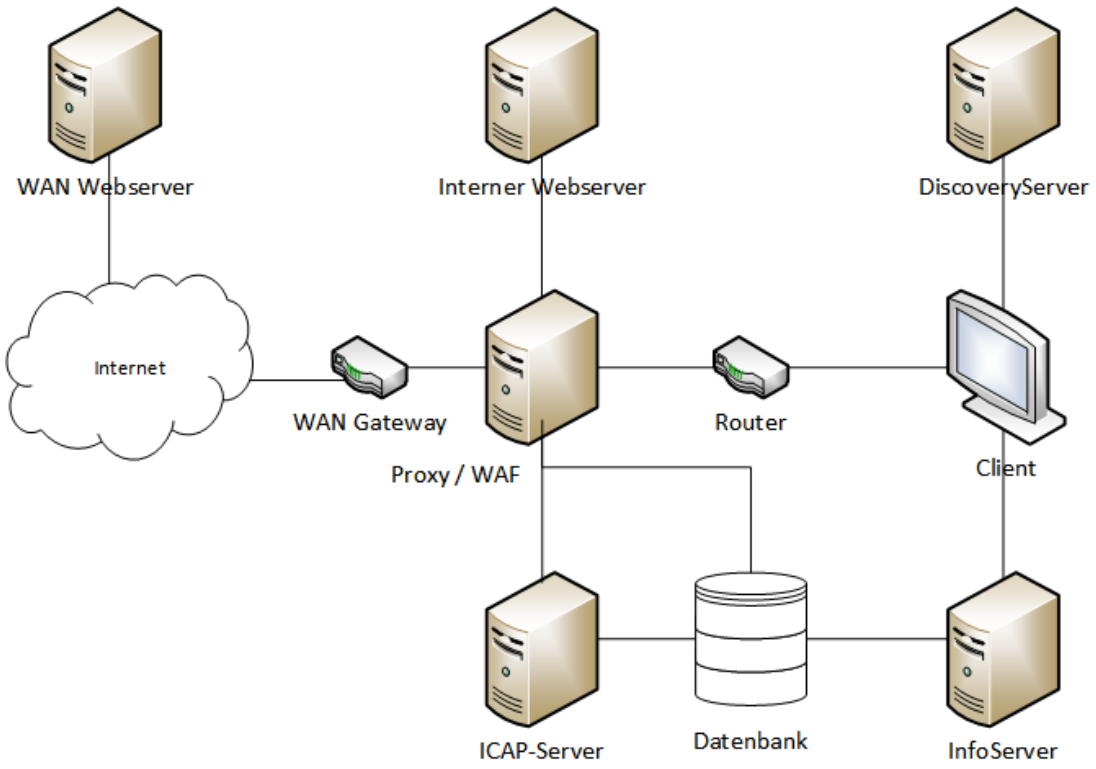

Abbildung 3.1: Architektur des Netzwerks

Teilkomponenten des Softwaresystems dargestellt. Deren Funktion wird in den folgenden Abschnitten erläutert.

3.3.1 Proxy/WAF

Der Proxy ist der zentrale Knotenpunkt des Aufbaus. Ein Router leitet Verbindungen der Ports 80 und 443 an den Proxy weiter. Dieser entscheidet, anhand von aus der HTTP Verbindung extrahierten und in der Datenbank hinterlegten Informationen, das weitere Vorgehen gegen einen Request.

Der Proxy besitzt folgende Funktionalitäten:

- Umleitung des Clients zu DiscoveryServer, zur Informationsgewinnung
- Umleitung des Clients auf InfoServer um den Benutzer zu informieren
- Informationsgewinnung über kontaktierte Quellserver

- Verwalten der Datenbankeinträge
- Kontaktieren des ICAP-Servers

3.3.2 DiscoveryServer

Der DiscoverySerer dient zur Gewinnung von Informationen über den Client, die nicht aus der HTTP Transaktion auf Applikationsebene zu extrahieren sind, durch clientseitigen Scriptcode. Die über die Clients gesammelten Informationen gehen über die im engeren Sinne sicherheitsrelevanten Informationen (Verschlüsselungs-Protokoll und Cipher Suiten) hinaus. In größeren Unternehmen mit zentralisiertem Bestandsmanagement mögen solche hier gesammelten Clientdaten weniger interessant sein. In kleineren Netzwerken und einer heterogenen Client-Struktur können die in diesem Prototypen gesammelten Client-Informationen aber durchaus hilfreich sein. Zum Beispiel für Migrationsplanungen, um herauszufinden wo der größte Handlungsbedarf zum Update oder Tausch eines Rechners besteht.

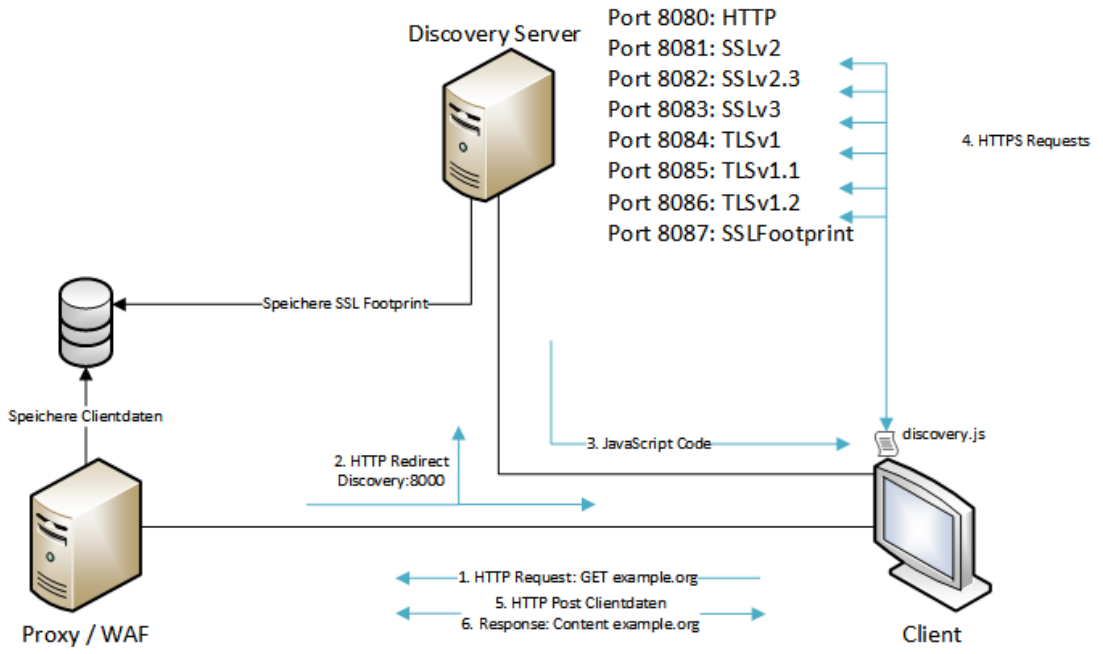

Abbildung 3.2: Client TLS/SSL Überprüfung

Der Proxy antwortet auf ein HTTP Request mit einem HTTP 301 Redirect, wenn die Konfiguration eines Clients abgefragt werden soll. Der DiscoveryServer sendet JavaScript Code, welcher die Konfiguration des Browser abfragt und alle Daten mittels HTTP Post Request

an den Proxy sendet. Dieser speichert sie in der Datenbank ab und gibt die Response des ursprünglichen Requests zurück, welcher mittels Redirect umgeleitet wurde. Dieser Vorgang kann bei beliebegen Requests eingeleitet werden. Der Ablauf ist in Abbildung 3.2 in seinen sechs Schritten noch einmal dargestellt.

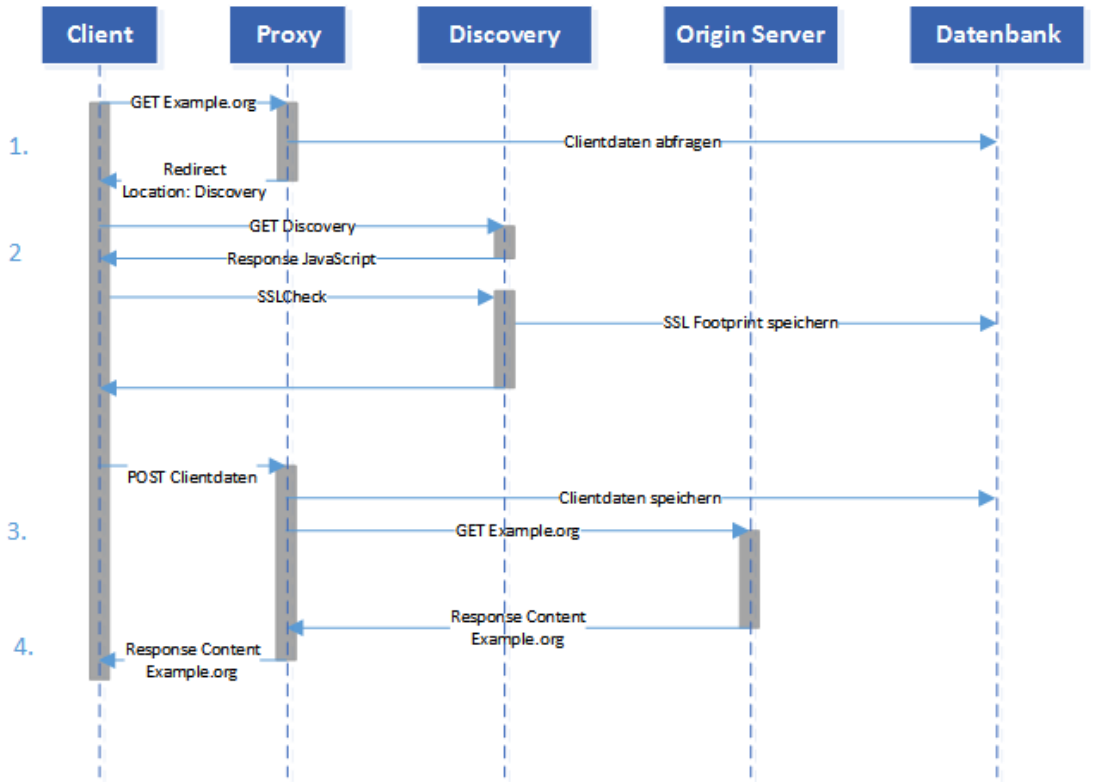

Abbildung 3.3: Sequenzdiagramm: Client Überprüfung

Das Sequenzdiagramm in Abbildung 3.3 zeigt (nochmals in anderer Darstellung) die Informationsgewinnung über den Client. Empfängt der Proxy ein HTTP Request, kann er mittels Redirect auf den DiscoveryServer den Prozess einleiten (1). Darauf sended er Client ein HTTP Request an den DiscoveryServer. Dieser sendet JavaScript Code zurück welcher Plugins und SSL Konfiguration des Clients überprüft (2). Mittels POST Request werden die gewonnen Daten an den Proxy übergeben (3). Dieser speichert sie in der Datenbank ab und liefert die Response (4), des zu Begin umgeleiteten, HTTP Requests in (1). Dieser Vorgang erfordert keine Interaktion des Benutzers und ist für diesen transparent.

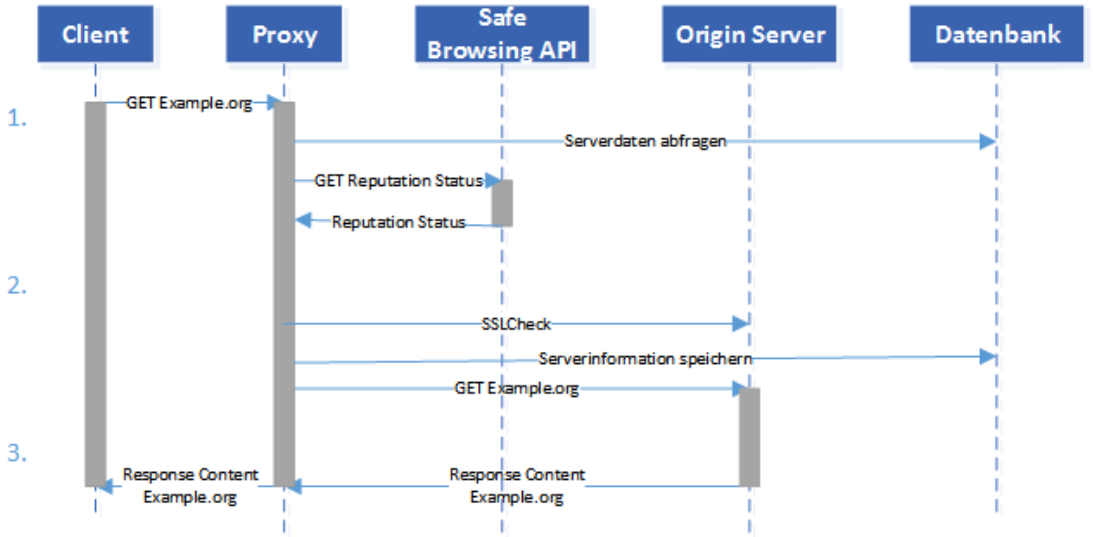

Abbildung 3.4: Sequenzdiagramm: Server Überprüfung

Stellt ein Client eine Anfrage an einen unbekannten Quellserver, startet der Proxy die Informationsgewinnung über den Server. Der Ablauf ist in Abbildung 3.4 dargestellt. Das SSLCheck Modul überprüft die unterstützten Versionen von TLS/SSL und CipherSuiten des Servers. Weiterhin überprüft der Proxy mit Hilfe Services von Drittanbietern die Reputation des Quellservers. Die gewonnen Informationen werden in der Datenbank abgespeichert.

3.3.3 InfoServer

Der InfoServer ist für das Reporting an den Benutzer zuständg. Wird in der Datenbank vermerkt, dass der Benutzer informiert werden soll oder blockiert ist, wird er vom Proxy auf den InfoServer umgeleitet. Der InfoServer informert den Nutzer über seinen derzeitigen Status und liefert wenn möglich Anweisung für weiteres Vorgehen. Wird zum Beispiel ein Plugin des Browsers als unsicher eingestuft, oder verwendet der Benutzer unsichere Verschlüsselungen oder CipherSuiten, kann dieser darüber informiert werden.

3.3.4 ICAP-Server

Der ICAP-Server kann Anfragen und Antworten modifizeren bevor sie zum Quellserver beziehungsweise an den Client weitergeleitet werden. Er dient dem Eingriff in eine Transaktion (auf Anwendungsebene) zum Schutze des Clients. Hierfür verwendet er die in der Datenbank hinterlegten Informationen. So können Header verborgen, deren Werte überprüft und gegebe-

nenfalls geändert werden. Zusätzlich können Antworten mit Content-Types, die beim Client mit einem unsicheren Plugin geöffnet werden, blockiert werden. Beim Download von Dateien kann der ICAP-Server, mit Hilfe von Signaturen, schädliche Dateien erkennen.

4 Implementierung

4.1 Bibiliotheken und Frameworks

- *Debian Linux*: Zum Entwickeln und Testen der Software wurde Debian als Betriebssystem gewählt, da es gut dokumentiert ist und alle benötigten Programme im Repository zugänglich sind.
- *Apache Webserver*: Für die Implementierung des Prototypen wird der Webserver Apache verwendet. Er ist quelloffen, gut dokumentiert und der meist verbeitete Server im Web. Benötigte Funktionalitäten wie SSL und CGI können in Form von Modulen aktiviert werden. [Fou15a]
- *Django Framework*: Zur Implementierung des Python Backend wurde das Django Framework verwendet. Der HTTP Request ist bereits geparst und wird als Python Objekt übergeben. Dadurch ist es einfach gewünschte Informationen zu extrahieren. Die in Django enthaltene Datenbank API (mit der Objekte verwaltet werden können) ist geeignet, um die gesammelten Informationen zu speichern. [Fou15b]
- *Requests*: Die Python Library Requests wird für die HTTP/S Anfrage an den Quellserver verwendet.
- *pyOpenSSL*: pyOpenSSL ist ein Python Binding für die Bibiliothek OpenSSL [Cal15]

4.2 Proxy

Der Proxy ist mittels des Django Framework implementiert. Der Webserver verarbeitet die Requests auf Transportschicht, inklusive Entschlüsselung (SSL-Terminierung). Der geparste HTTP Request der Anwendungsebene wird mittels des *Common-Gateway-Interface* an das Python Backend übergeben. Der Informationsfluss ist in Abbildung 4.1 abgebildet.

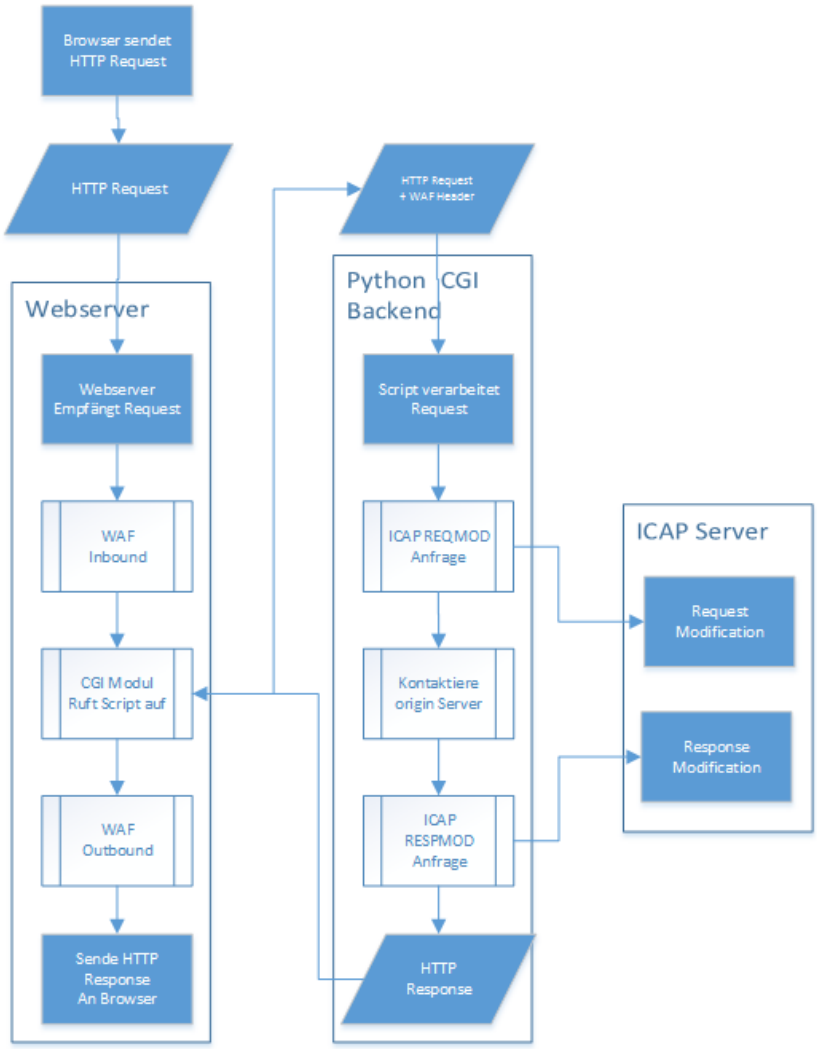

Abbildung 4.1: Ablauf eines HTTP Requests

Das Django Framework folgt dem Model-View-Controller Prinzip. In der Konfigurationsdatei *urls.py* können verschieden URLs Funktionen zugeordnet werden, welche bei einem Request ausgeführt werden. Diesen (in der *views.py*) definierten Funktionen werden (bei einem Aufruf)

der geparste Request als *HTTPRequest* Objekt übergeben. Ein Django-Model entspricht einer Tabelle in der Datenbank. In der Datei *models.py* werden die Felder des Models als Python Klasse definiert und von Django, bei der *migration*, in ein Datenbankschema verarbeitet.

4.2.1 Model

Um ein Model zu erstellen wird die Django Klasse *model* abgeleitet. Diese enthält die Funktionen um die zum Model gehörige Tabelle zu manipulieren. Für den Prototypen werden vier verschiedene Models erstellt: *Client* und das Model *Server*. Das Listing 4.1 zeigt einen Ausschnitt aus der datei *models.py*. Felder einer Model Klasse werden mit vorgefertigten Typen initialisiert welche Typen der Datenbank entsprechen.

```
1  class Client(models.Model):
2      ip = models.CharField(max_length=15)
3      user_agent = models.CharField(max_length=255)
4      appName = models.TextField(default="")
5      appCodeName = models.TextField(default="")
6      appVersion = models.TextField(default="")
7          plugins = models.ManyToMany(Plugin)
8          mimeTypes = models.ManyToMany{mimeType}
9
10         ...
11         class Meta:
12                 unique_together = ("ip", "user_agent")
```

Listing 4.1: Client Model

Die Tabellen 4.1 und 4.2 listen die Felder der Client und Server Tabelle auf. Wird im Model kein Feld als Primary-Key deklariert fügt Django der Tabelle eine autoinkrementierende ID hinzu. Identifiziert wird ein Client anhand der IP-Addresse und dem User-Agent String. So dass verschiedene Browser unter der gleichen IP-Addresse unterschieden werden können. Django bietet die Möglichkeit Relationen bei der Deklaration des Attributes anzugeben. So werden die Attribute, für die Zuordnung der Clients zu den verwendeten Plugins und unterstützten MIME Typen, mittels der *ManyToMany* Klasse initialisiert. Die Tabellen für Many-to-Many Relationen werden bei der Migration automatisch erstellt.

4.2.2 View

In Listing 4.2 ist die Konfigurationsdatei dargestellt, in der mittels Expressions Views zu URLs zugeordnet werden können. Jeder URL wird die Funktion *main* zugeordnet.

Tabelle 4.1: Felder der Client Tabelle

Name	**Beschreibung**	**Type**
id	Primary Key	serial
ip	IP-Addresse	varchar(16)
user_agent	User-Agent String	varchar(255)
appName	Browser Bezeichnung	text
appCodeVersion	Browser Version	text
plugins	Plugins	ManyToMany
mimeTypes	mimeTypes	ManyToMany
javaEnabled	Java Unterstützung im Browser	bool
language	Spracheinstellung des Browsers	text
cookieEnabled	Cookie Unterstützung im Browser	bool
platform	Betriebssystem	text
product	Browser Engine	text
SSLv2_enabled	SSL v2 Status	text
SSLv3_enabled	SSL v3 Status	text
SSLv23_enabled	SSL v2.3 Status	text
TLSv1_enabled	TLS v1 Status	text
TLSv1_1_enabled	TLS v1.1 Status	text
TLSv1_2_enabled	TLS v1.2 Status	text
SSLFootprint	SSL Footprint	text
blocked	Blockiert Flag	bool
blockedmessage	Block Nachricht	bool
info	Info Flag	text
infomessage	Info Nachricht	text
accept	Accept-Header	text

Tabelle 4.2: Felder der Server Tabelle

Name	**Beschreibung**	**Type**
hostname	Hostname	varchar(255)
ip	IP-Addresse	varchar(16)
sb_status	Reputation Google Safe Browsing API	integer
sb_desc	Beschreibung (z.b. phishing, malware, unwanted software)	varchar(128)
SSLv2_ciphers	CipherSuiten mit SSL v2	text
SSLv3_ciphers	CipherSuiten mit SSL v3	text
SSLv23_ciphers	CipherSuiten mit SSL v2.3	text
TLSv1_ciphers	CipherSuiten mit TLS v1	text
TLSv1_1_ciphers	CipherSuiten mit TLS v1.1	text
TLSv1_2_ciphers	CipherSuiten mit TLS v1.2	text
city	Stadt	bool
regioncode	Region Code	text
regionname	Region Name	text
countrycode	Land Code	text
countryname	Land Name	text
blacklisted	Anzahl der Blacklist treffer	integer
bl_feodo	Feodo Tracker Status	text
bl_dragon	Dragon Research Group Status	text
bl_mdl	Malware Domain List Status	text
bl_openbl	OpenBL Status	text
bl_ptank	Phish Tank Status	text
bl_spamhs	The Spamhaus Project Status	text
bl_zeus	Zeus Tracker Status	text
bl_alien	Alien Vault Status	text
bl_cleanmx	Clean MX Status	text
bl_bfb	Brute Force Blocker Status	text
bl_chaos	Chaos Reigns Status	text
bl_malcode	Malc0de Status	text
blocked	Blockiert Flag	text
blockedmessage	Block Nachricht	text
info	Info Flag	text
infomessage	Info Nachricht	text

```
urlpatterns = [

    url(r'.*', views.main, name='main'),
]
```

Listing 4.2: Proxy URLPatterns

Erhält der Proxy eine Anfrage wird diese in der *main* Funktion verarbeitet, deren Pseudocode in Listing 4.3 dargestellt ist. Die Identifikation geschieht mittels der IP-Addresse und User-Agent String. Ist dieser nicht in der Datenbank vorhanden, überprüft der Proxy anhand der URL, ob es sich um Daten des DiscoveryServer handelt, wenn nicht leitet der Proxy mittels Djangos *redirect* Funktion den Request auf den DiscoveryServer um.

```
if client not in database:
        if request is from discoveryScript:
                addclient(client)
        else:
            return redirect(discoveryServer)
else:
        if client.info or client.blocked
                return redirect(infoServer)
        ...
```

Listing 4.3: Pseudocode der main Funktion

Ist der Client in der Datenbank vorhanden, wird anhand der Statusfelder *info* und *blocked* überprüft, ob ein redirect zum InfoServer erfolgen muss.

Bevor der Request an den Origin-Server gestellt wird, wird dessen Status aus der Datenbank abgefragt. Ist der Server als blockiert markiert oder sollte der Nutzer beim Aufrufen der Seite informiert werden, wird dieser auf den InfoServer umgeleitet. Besteht für den Server noch kein Datenbankeintrag, wird dieser überprüft und hinzugefügt. Bei der Überprüfung wird die Reputation der Google Safe Browsing API abgefragt und die vom Server akzeptierten Verschlüsselungen und CipherSuiten überprüft. Ist der Server als unsicher eingestuft, wird der Client auf den InfoServer weitergeleitet.

Für das Abfragen der Reputation wird ein *GET* Request an den von Google bereitgestellten Server gesendet. Die URL muss das in 4.4 abgebildete Format haben.

```
https://sb-ssl.google.com/safebrowsing/api/lookup?
        client=CLIENT&key=APIKEY&appver=APPVER&pver=PVER&url=URL
```

Listing 4.4: Google Safe Browsing API Request

Die *client* und *appver* Parameter dienen zur Identifzierung des Clients, und können beliebig gewählt werden. Der *pver* Parameter gibt an welches Protokoll, der Lookup API verwendet wird (die aktuelle Version ist 3.1). Der API-Schlüssel des *Google Developers Project* wird im *key* Parameter übergeben.

Ist die URL als vertrauenswürdig eingestuft, antwortet der Server mit dem HTTP Response Code 204 (No Content). Gilt die URL als schädlich, liefert der Server ein 200 Status Code (OK). Der body enthält die Typen der Blacklists, in welchem die URL gefunden wurde. Möglich sind: *phishing*, *malware*, *unwanted*.
Mit der Metascan Public API, welche von OPSWAT bereitgestellt wird, werden URLs auf ihren Status in verschiedenen öffentlichen Blacklists überprüft. Hierfür reicht es, wie bei der Google API, ein GET-Request an eine spezielle URL zu senden. Zusätzlich muss das Request den HTTP-Header *apikey* mit dem Schlüssel des Benutzers enthalten, damit die Anfrage bearbeitet wird. Die Antwort ist ein json-Objekt, in welchem der Status in den verschiedenen Blacklists und Metadaten, wie Geoinformationen, enthalten sind. Diese Informationen werden in den in Tabelle 4.2 dargestellten Feldern gespeichern. [OPS15]

4.2.3 SSLCheck

Das SSLCheck Modul überprüft mittels erschöpfender Suche einen Server auf unterstützte Verschüsselungen und verwendeten Cipher-Suites. In Listing 4.5 ist Pseudocode dargestellt der das Vorgehen des Modules beschreibt. Die Liste der Bezeichner, der CipherSuiten, für OpenSSL wird aus einer csv-Datei gelesen. Diese enthält alle unterstützen CipherSuiten nach [Ope15].

```
for sslVersions:
        Context = SSLContext(version)
        for ciphersuite in sslVersions:
                Context.setCipherList(ciphersuite)
                socket = connect(Context, ip, port)
                try:
                        socket.do_handshake()
                        supportedCiphers.add(cipher)
                except:
                        unsupportedCiphers.add(cipher)
                socket.close()
```

Listing 4.5: SSLCheck Pseudocode

Das Programm besteht aus zwei Schleifen. Die äußere Schleife iteriert über die Liste verschiedener Verschlüsselungen, die innere über die Liste der CipherSuites. Für die SSL Operationen wird ein Python Binding der OpenSSL Library verwendet.

4.2.4 ICAP-Client

Der ICAP-Client implementiert die Funktion *build_header*, für das Erstellen der ICAP-Header und Anfragezeile. Diese Funktion wird die gewünschte ICAP-Methode und eine Liste der zu kapselten Entitäten übergeben. Elemente der Liste sind Tup l aus dem Bezeichner der Entität, und deren Inhalt. In Listing 4.6 ist ein Aufruf dargestellt. Soll kein HTTP-Body gekapselt werden, wird die *nullbody* Entität hinzugefügt.

```
1  entities = []
2  entities.append(ENC_TYPE_REQHDR, http_request_headers)
3  entities.append(ENC_TYPE_NULL, None)
4  icap_headers = self.build_header(METHOD_REQMOD, entities)
5
6  send(icap_headers)
7  for e in entities
8          send(e)
9
10 <empfange response>
```

Listing 4.6: ICAPreqmod Pseudocode

4.3 DiscoveryServer

Der DiscoveryServer verwendet die Komponenten *ServeScript*, *DiscoveryScript* und *SSLCheck-Server*. Deren Implementation wird in den folgenden Abschnitten beschrieben.

4.3.1 ServeScript

Das *ServeScript*-Modul verwendet die *HTTPServer*-Klasse des *BaseHTTPServer* Modules. Zur Instanziierung wird dem Konstruktor der *HTTPServer* Klasse eine Ableitung des *BaseHTTPRequestHandlers* übergeben. Dieser implementiert Funktionen die beim Eintreffen eines Requests aufgerufen werden. Für die ServeScript Komponente wird die *do_GET* Funktion implementiert. Auf eine Anfrage gibt er eine HTML Datei mit eingebettetem JavaScript Code zurück. Listing 4.7 zeigt den Python Code der *ServeScript* Klasse.

```
class ServeScript(BaseHTTPRequestHandler):
    with open (os.path.join(os.path.dirname(__file__),
        'DiscoveryScript.js'), "r") as myfile:
        txdata=myfile.read().replace('\n', '')

    def do_GET(self):

        self.send_response(200)
        self.send_header('Content-type','text/html')
        self.send_header('Access-Control-Allow-Origin','*')
        self.end_headers()
        self.wfile.write("<body></body><script>"
                         +self.txdata
                         +"</script>")

        return
```

Listing 4.7: Pseudocode DiscoveryServer

Damit das DiscoveryScript privilegiert ist, Cross-Origin-Requests an den *SSLCheckServer* und *SSLFootprintServer* zu senden, wird der *Access-Controll-Allow-Origin* Header hinzugefügt. Das Script wird in HTML eingebettet, damit es eine HTML Form für das Einsenden der Daten erstellen kann.

4.3.2 SSLCheckServer

Der SSLCheckServer verwendet das Python Bindung pyOpenSSl um SSL/TLS Sockets zu erstellen. Die Hauptfunktion des SSLCheckServers ist die *runserver* Methode, deren Ablauf in Listing 4.8 dargestellt ist. Dieser wird die SSL/TLS Version und ein Port übergeben. Daraufhin erstellt sie den SSL Context mit der angegebenen Version, bei welchem, mit der *set_options* Methode, die restlichen Protokollversionen abgeschaltet sind, um eine Renegotiation zu verhindern. Dies geschieht in der *setupContext* Funktion. Die *runserver*-Methode erstellt ein SSL Socket und bindet diesen an den übergebenen Port.

```

ctx = setupContext(SSLVersion)
socket = SSLSocket(ctx, Port)

```

```
while running:
        client = socket.accept()
        while recieving:
                client.recv()


        client.send(toHTTP(dummydata))
        client.close()
```

Listing 4.8: Pseudocode runserver

Die *runservers* Funktion erstellt für jedes übergebene Tuple aus Methode und Port einen Thread der *runserver* Funktion. Ein Aufruf dieser Funktion, wie in Listing 4.9 dargestellst, startet die Komponente.

```
    runservers([SSL.SSLv2_METHOD, SSL.SSLv3_METHOD,
                SSL.SSLv23_METHOD, SSL.TLSv1_METHOD,
                SSL.TLSv1_1_METHOD, SSL.TLSv1_2_METHOD],
               [8081, 8082, 8083, 8084, 8085, 8086])
```

Listing 4.9: runallservers Beispiel

4.3.3 SSLFootprintServer

Der SSLFootprintServer wertet die Informationen aus, die ein Client (während des Handshakes) mit der Client Hello Nachricht an den Server sendet, und speichert diese in der Datenbank ab. Zur Implementierung wird ein einfaches Socket verwendet. Der Pseudocode in Listing 4.10 stellt den Ablauf des Modules dar.

```
ctx = setupContext(SSLVersion)
socket = SSLSocket(ctx, Port)

while running:
        client = socket.accept()
        client_hello = client.recv()
        client.close()
        data = parseHello(client_hello)
        update_database(data)
```

Listing 4.10: Pseudocode SSLFootprintServer

Die Versionsnummern des Records, des Handshakes und die Liste der Cipher Suiten werden extrahiert und in der Datenbank gespeichert.

4.3.4 DiscoveryScript

Das vom DiscoveryServer an den Client gesendete Script, dessen Ablauf in Listing 4.11 dargestellt ist, verwendet das *navigator* JavaScript-Objekt, um die verwendeten Plugins und die Metadaten des Browser abzufragen. Die JavaScript API XMLHTTPRequest wird verwendet um Cross-Origin-Requests an den SSLCheckServer zu senden. Alle Informationen werden in einem Dictionary gespeichert.

```
data = getBrowserData(navigator);
data['SSLv2Enabled'] = cors_request(SSLCheckServer:8081);
data['SSLv3Enabled'] = cors_request(SSLCheckServer:8082);
data['SSLv23Enabled'] = cors_request(SSLCheckServer:8083);
data['TLSv1Enabled'] = cors_request(SSLCheckServer:8084);
data['TLSv11Enabled'] = cors_request(SSLCheckServer:8085);
data['TLSv12Enabled'] = cors_request(SSLCheckServer:8086);
data['SSLFootprint'] = cors_request(SSLFootprintServer:8087);
postData(proxy, data);
```

Listing 4.11: Pseudocode DiscoveryScript

Die gewonnen Daten werden mit der *postData* Funktion an den Proxy weitergeleitet. Diese Funktion erstellt ein HTML *form* Objekt, mit Feldern für die übergebenen Daten, und sendet dieses mittels *POST* Request an den Proxy.

4.4 InfoServer

Der InfoServer ist in Python implementiert. Verwendet werden das *BaseHTTPServer* und das *sqlite3* Modul.

Zur Instanziierung wird dem Konstruktor der *HTTPServer* Klasse eine Ableitung des *BaseHTTPRequestHandlers* übergeben. Dieser implementiert Funktionen, die beim Eintreffen eines Requests aufgerufen werden. Für die abgeleitete InfoServer Klasse wird die *do_GET* Funktion implementiert, deren Ablauf in Listing 4.12 dargestellt ist,

```

connection = sqlite3.connect(database)
c = connection.cursor()

```

```
def do_GET(self):

        c.execute(query % client_ip, user_agent)
        userobject = c.fetch()
        if userobject.blocked:
                response = "You are blocked. Message: "
                + userobject.blockedmessage
        else if userobject.info:
                response = "Attention. Message: "
                + userobject.infomessage
        else:
                response = "unknow user"
```

Listing 4.12: InfoServer Pseudocode

4.5 ICAP-Server

Für die Implementation des ICAP-Servers wird das *pyicap*-Modul verwendet. Dieses Modul stellt die Klassen *ICAPServer*, eine Ableitung des *SocketServer.TCPServer*, und *BaseICAPRequestHandler*, eine Ableitung des *SocketServer.StreamRequestHandler*, in welchem Operationen für das ICAP-Protokoll implementiert sind, zur Verfügung.

In der, von *BaseICAPRequestHandler* abgeleiteten, Klasse *ICAPHandler* sind die Funktionen für die vom Server angebotenen ICAP-Methoden *OPTIONS* und *REQMOD* und *RESPMOD* implementiert.

```
def _REQMOD(self):

        c.execute(query % client_ip, user_agent)
        userobject = c.fetch()

        for mime in userobject.filtertypes:
                http_headers['accept'].remove(mime)

        icap_send_modified_request()
```

Listing 4.13: REQMOD Methode Pseudocode

In Listing 4.13 ist der Ablauf der REQMOD Methode dargestellt. Sie modifiziert den HTTP *Accept* Header und entfernt alle verbotenen MIME-Types. Da Webserver den *Accept* Header ignorieren können, wird der *Content-Type* Header der Antwort, in der RESPMOD Methode,

überprüft. Ist der MIME-Type in der Liste der (als nicht zulässig eingestuften) Typen enthalten, wird statt der ursprünglichen Antwort eine präparierte Nachricht mit dem HTTP Status-Code *403 Forbidden* zurückgegeben. Der Ablauf der *RESPMOD* Methode ist in Listing 4.14 dargestellt.

```
1 def _RESPMOD(self):
2
3         c.execute(query % client_ip, user_agent)
4         userobject = c.fetch()
5
6         if content_type in userobject.filtertypes:
7                 return http_forbidden('unzulässiger content-type')
```

Listing 4.14: RESPMOD Methode Pseudocode

5 Tests

5.1 Testumgebung

Der Aufbau der Testumgebung ist in Abbildung 5.1 dargestellt. Das Netzwerk besteht aus vier virtuellen Maschinen: Router, Proxy, Webserver und Client. Für den Router sind vier Netzwerkadapter konfiguriert. Eine Netzwerkbrücke stellt die Verbindung zum Host-Computer und dem Internet her. Die drei restlichen Netzwerkadapter sind Proxy, Webserver und Client zugeordnet und bilden drei Subnetze die mittels Router verbunden sind. DiscoveryServer und InfoServer werden auf dem Host-Computer gestartet, der ICAP-Server auf der Maschine des Proxies.

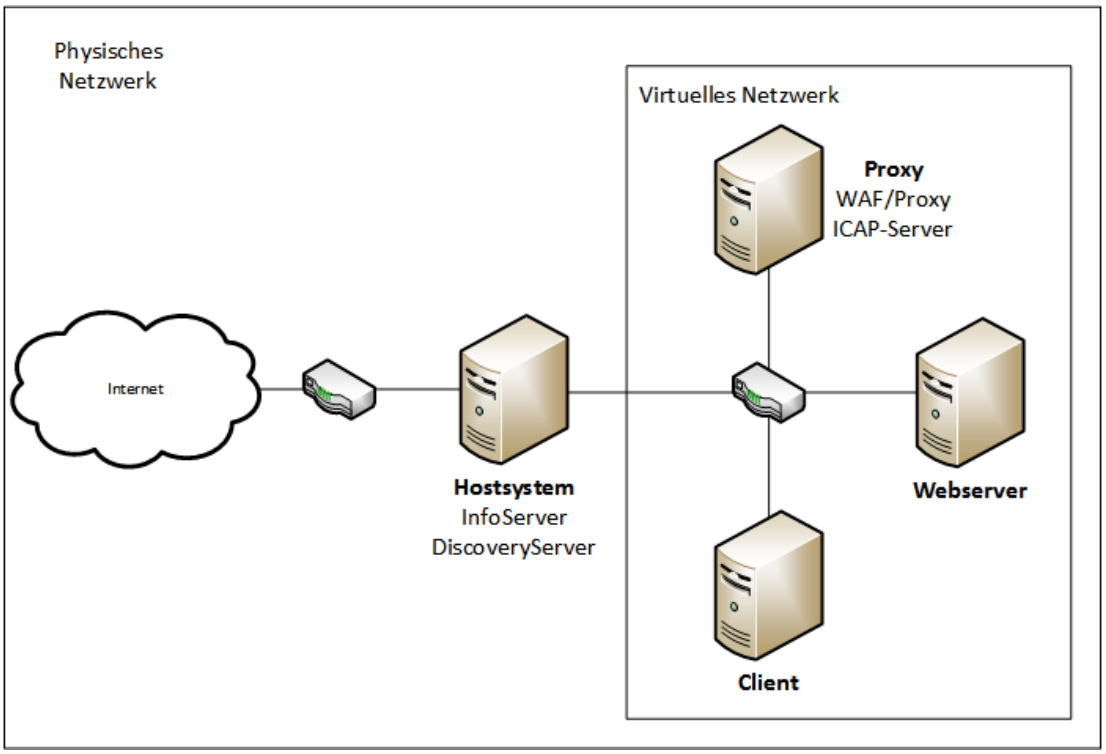

Abbildung 5.1: Aufbau der Testumgebung

Mit dem Programm *iptables* ist der Router so konfiguriert, dass ein- und ausgehende Verbindungen auf den Ports 80 und 443 an den Proxy umgeleitet werden. Mit statischen Routen wird die Verbindung zwischen den verschiedenen Netzwerken hergestellt.

5.2 Testläufe

Auf der Client VM sind für Testzwecke die Browser *Google Chrome* und *Internet Explorer* installiert. Nach einem Aufruf der Domain *google.com* sind die in den Abbildungen 5.2 und 5.3 dargestellten Tabelleneinträge entstanden.

	id	mimeType	fileExt	ugin_	llowe
	Filter	Filter	Filter	Filter	Filter
1	81	application/x-shockwave-flash	swf	92	1
2	82	application/futuresplash	spl	92	1
3	83	application/pdf	pdf	87	1
4	84	application/x-shockwave-flash	swf	88	1
5	85	application/futuresplash	spl	88	1
6	86	application/vnd.chromium.remoting-viewer		89	1
7	87	application/x-nacl		90	1
8	88	application/x-pnacl		90	1
9	89	application/x-google-chrome-pdf	pdf	91	1

Abbildung 5.2: Daten der MIME-Type Tabelle

	id	name	fileName	description	llowe
	Filter	Filter	Filter	Filter	Filter
1	87	Chrome PDF Viewer	mhjfbmdgcfjbbpaeojof...		1
2	88	Shockwave Flash	pepflashplayer.dll	Shockwave Flash 18.0 r0	1
3	89	Chrome Remote Desktop Viewer	internal-remoting-viewer	This plugin allows you to...	1
4	90	Native Client	internal-nacl-plugin		1
5	91	Chrome PDF Viewer	internal-pdf-viewer	Portable Document For...	1
6	92	Shockwave Flash	Flash32_18_0_0_209.ocx	Shockwave Flash 18.0 r0	1

Abbildung 5.3: Daten der Plugin-Tabelle

Nun können Plugins oder die Zuordnung von MIME-Typ und Plugin als blockiert markiert werden. Mittels Datenbank-Triggern können Clients automatisch markiert werden. Damit sie auf den InfoServer umgeleitet werden. Trigger sind jedoch im Prototypen nicht implementiert.

5.2.1 Fallbeispiel 1: Blockieren eines Clients aufgrund Plugins

Als Fallbeispiel werden nun manuell alle Clients blockiert welche *swf* Dateien mit einem Plugin öffnen, dessen Dateinamen *Flash32_18_0_0_209.ocx* ist. In Listing 5.1 werden die dafür nötigen Befehle aufgelistet, welche in der Python Shell des Django Environments ausgeführt werden müssen.

```
1 from ProxyApp.models import mimeType, Client
2 mimeType.objects.filter(fileExt="swf",
3         plugin_fileName="Flash32_18_0_0_209.ocx")
4         .update(allowed=0)
5 Client.objects.filter(mimeTypes__allowed=0)
6         .update(blocked=1, blockedmessage="unerlaubtes Flash Plugin")
```

Listing 5.1: Benutzer mit bestimmtem Plugins Blockieren

Nachdem die benötigten Models importiert sind, wird mit dem Befehl in Zeilen 2-4 das *allowed* Feld aller mimeType-Einträge auf *false* gesetzt bei welchen *swf*-Dateien mit dem Programm *Flash32_18_0_0_209.ocx* geöffnet werden. Mit dem Befehl in Zeile 5 und 6 werden alle Clients blockiert, welchen MIME-Typen zugeordnet sind, die nicht erlaubt sind, also das *allowed*-Feld des MIME-Typ auf *false* gesetzt ist.

In diesem Fall wird für den Client der Internet Explorer als blockiert markiert. Wird mit diesem ein Request getätigt, wird dieser auf den InfoServer umgeleitet und es erscheint die in der Datenbank hinterlegte Nachricht.

5.2.2 Fallbeispiel 2: Informieren über Cipher Suite

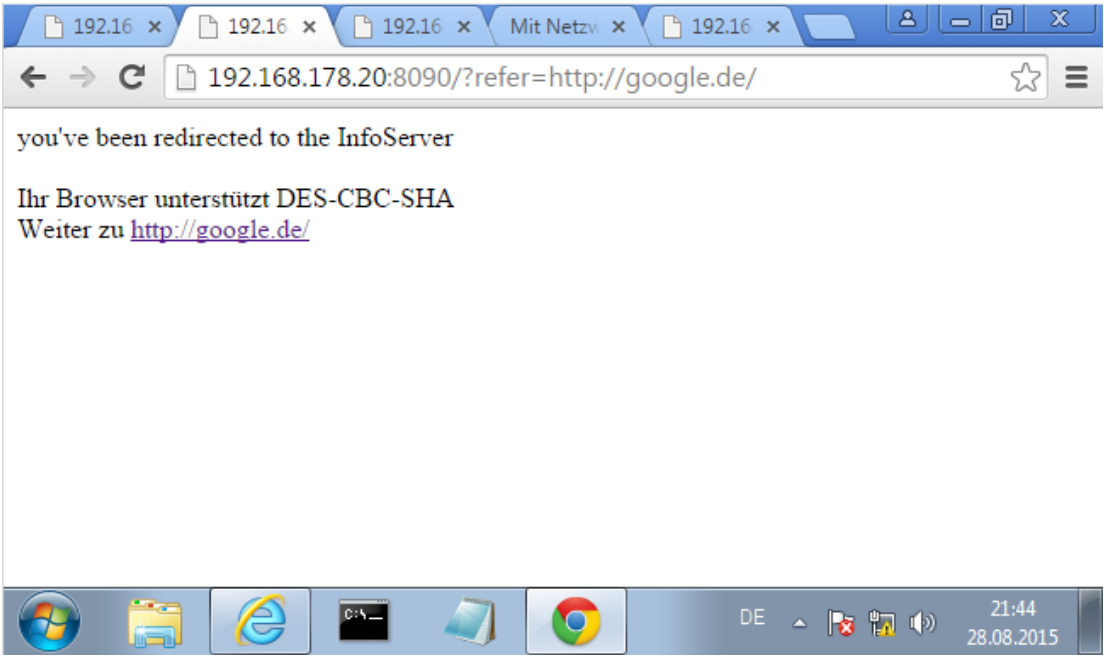

Abbildung 5.4: Zum InfoServer weitergeleiteter Client

Ein weiterer Anwendungsfall wäre, zu überprüfen ob ein Client eine bestimme CipherSuite unterstützt und diesen zu Informieren, sollte diese nicht sicher sein. Der Befehl in Listing 5.2 informiert alle Benutzer, dass sie die RSA Cipher *DES-CBC-SHA*-Cipher anbieten (hex-Code 0x00 0x09).

```
1 Client.objects.filter(sslfootprint__contains="00:09")
2         .update(info=1, infomessage="Ihr Browser unterstüzt
3         DES-CBC-SHA")
```

Listing 5.2: Benutzer über CipherSuite informieren

Ruft der Benutzer eine Webseite auf, wird er zu erst zum InfoServer umgeleitet

5.2.3 Fallbeispiel 3: Blockieren aufgrund von unsicheren SSL/TLS-Versionen

Ist eine SSL/TLS Version vom Client nicht unterstützt, ist der Wert des Feldes *Disabled*. Mit einem Befehl wie in Listing 5.3, wird mit der *exclude* Methode nach Clients gefiltert, die SSL3 unterstützen.

```
1 Client.objects.exclude(ssl3enabled="Disabled")
2         .update(blocked=1, blocked="Sie haben SSLv3 aktiviert.")
```

Listing 5.3: Benutzer über veraltete SSL/TLS Version informieren und blockieren

5.2.4 Fallbeispiel 4: Informationen über Cipher Suiten des Ziel-Servers

Da nach der Client-Anfrage die Informationsfindung über den Ziel-Server stattgefunden hat, können nun zum Beispiel die Cipher Suiten des Servers abgefragt werden, wie in Listing 5.4 dargestellt.

```
1 >>> Server.objects.get(hostname="www.google.com").SSLv3_ciphers
2         "['ECDHE-RSA-AES256-SHA', 'AES256-SHA',
3         'ECDHE-RSA-DES-CBC3-SHA', 'DES-CBC3-SHA',
4         'ECDHE-RSA-AES128-SHA', 'AES128-SHA',
5         'ECDHE-RSA-RC4-SHA', 'RC4-SHA', 'RC4-MD5']"
```

Listing 5.4: Server CipherSuiten für SSLv3 von google.com abfragen

5.2.5 Fallbeispiel 5: Blockieren von Ziel-Servern aufgrund von Blacklist-Eintrag

Wird vom Client eine Seite aufgerufen die in einer (durch die Metascan überprüften) Blacklist enthalten ist, oder von der Safe Browsing API als nicht vertrauenswürdig eingestuft wird, erfolgt eine Weiterleitung an den InfoServer.

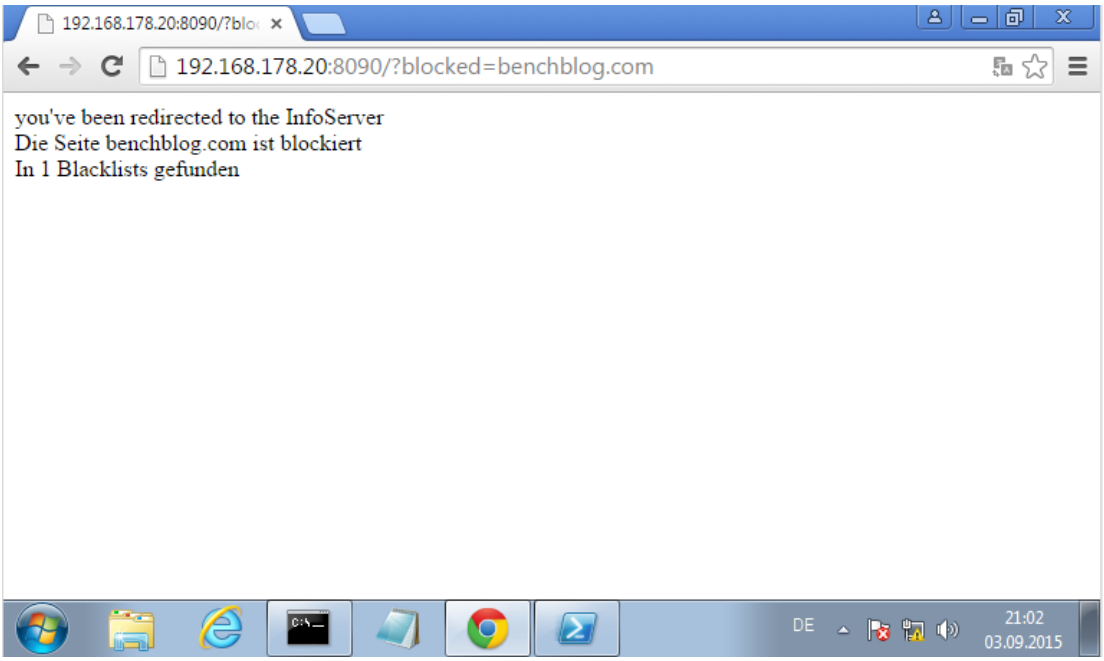

Abbildung 5.5: Blockierte Webseite

In Abbildung 5.5 ist ein Beispielaufruf der Seite *Benchblog.com* dargestellt, welche auf der *Zeus Tracker* Blacklist vermerkt ist (wie in Listing 5.5 gezeigt ist).

```
>>> s = Server.objects.get(hostname='benchblog.com')
>>> s.bl_zeus
u'detecttime:2015-09-03T01:52:07Z,
updatetime:2015-09-03T18:41:52Z,
alternativeid:https://zeustracker.abuse.ch/
        monitor.php?search=216.194.169.100,
confident:65,
result:blacklisted,
assessment:botnet'
```

Listing 5.5: Zeus Status von benchblog.com

5.2.6 Fallbeispiel 6: Blockieren von Content-Types durch ICAP

Ist bekannt, dass ein Benutzer ein unsicheres Plugin für einen *Content-Type* verwendet, kann dieser mit dem ICAP-Server daran gehindert werden, diesen Inhalt zu empfangen. Wie in Listing 5.6 dargestellt, können MIME-Types, welche für einen Benutzer blockiert werden sollen,

eingetragen werden. Im Beispiel werden PDFs für den Client blockiert, dessen Eintrag in der Tabelle an erster Stelle steht.

```
>>> c = Client.objects.all()[0]
>>> c.filterTypes = "application/pdf"
>>> c.save()
```

Listing 5.6: Unzulässigen MIME-Type hinzufügen

Empfängt der Client nun eine Antwort, deren *Content-Type*-Header den Wert *application/pdf* besitzt, wird diese (vom ICAP-Server) in eine Antwort mit dem Status-Code *403 Forbidden* umgewandelt, deren Inhalt eine Nachricht an den Benutzer ist. Ein Beispiel ist in Abbildung 5.6 dargestellt.

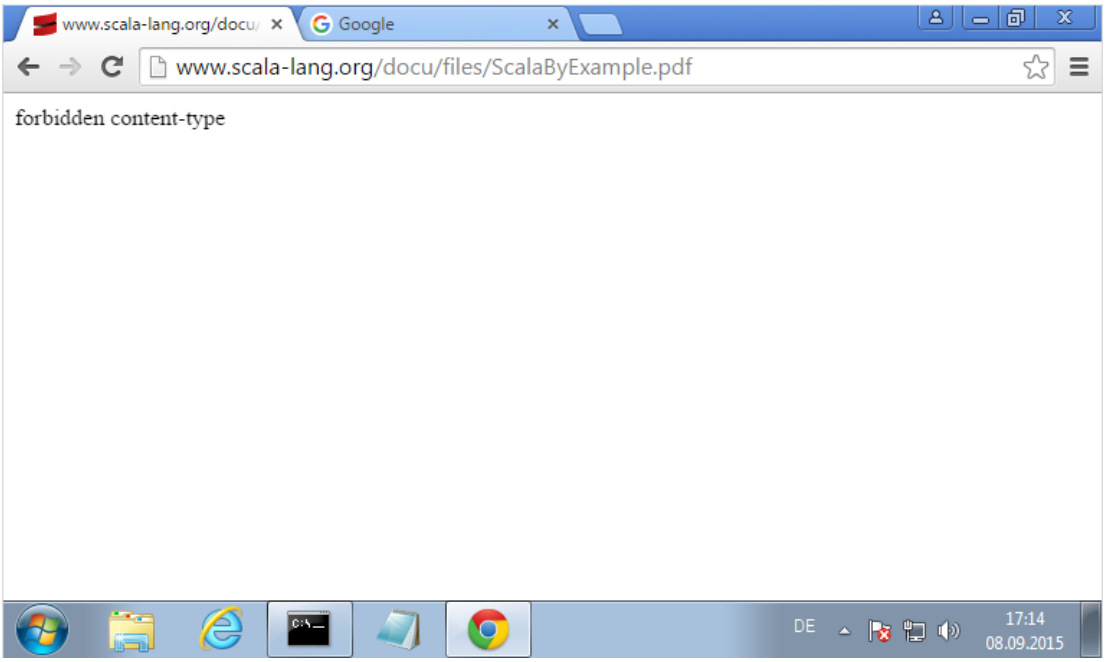

Abbildung 5.6: Blockierter Content-Type

5.2.7 ModSecurity Test

Zum Testen der WAF ist auf dem internen Webserver die Datei *include.php* erstellt worden, deren Inhalt in Listing 5.7 abgebildet ist. Die Datei ist präpariert, dass *File Inclusion* und reflektives *XSS* möglich ist.

```
<?php
$file = $_GET{'file'];
print($file);
include($file);
?>
```

Listing 5.7: include.php

Abbildung 5.7 zeigt eine Anfrage an den Webserver, welche aufgrund der *script*-Tags im GET-Parameter von ModSecurity blockiert wurde.

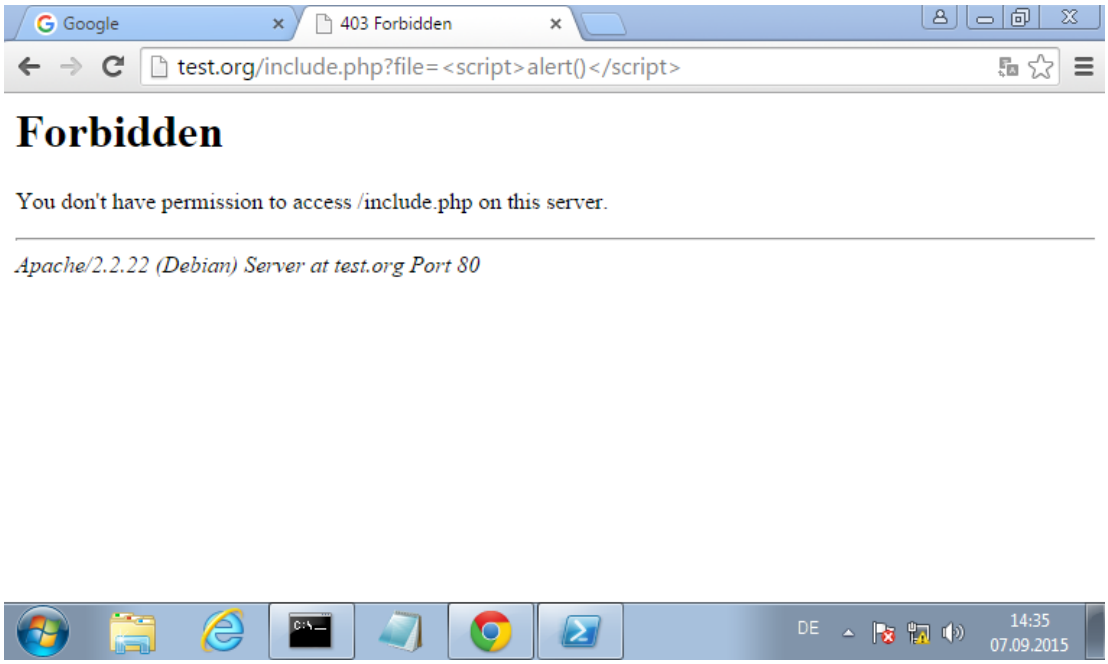

Abbildung 5.7: Durch WAF blockierter Aufruf

5.2.8 Die gefüllten Tabellen exemplarisch

In einem längeren Listing 5.8 werden abschließend drei Beispielausgaben auf die Administrator Console gezeigt. Es handelt sich um die Datenbankeinträge, die entstanden sind für Server und Client (siehe Tabellen 4.1 und 4.2) sowie die gefüllte Plugin-Tabelle.

```
>>> s = Server.objects.get(hostname='google.com')
>>> pprint(vars(s))

{'SSLv23_ciphers': u"['AES128-GCM-SHA256', 'AES128-SHA',
'AES128-SHA256', 'AES256-GCM-SHA384', 'AES256-SHA',
'AES256-SHA256', 'DES-CBC3-SHA', 'ECDHE-RSA-AES128-GCM-SHA256',
'ECDHE-RSA-AES128-SHA', 'ECDHE-RSA-AES128-SHA256',
'ECDHE-RSA-AES256-GCM-SHA384', 'ECDHE-RSA-AES256-SHA',
'ECDHE-RSA-AES256-SHA384', 'ECDHE-RSA-DES-CBC3-SHA',
'ECDHE-RSA-RC4-SHA', 'RC4-MD5', 'RC4-SHA']",
 'SSLv2_ciphers': u'None',
 'SSLv3_ciphers': u"['AES128-SHA', 'AES256-SHA',
 'DES-CBC3-SHA', 'ECDHE-RSA-AES128-SHA',
 'ECDHE-RSA-AES256-SHA',  'ECDHE-RSA-DES-CBC3-SHA',
 'ECDHE-RSA-RC4-SHA', 'RC4-MD5', 'RC4-SHA']",
 'TLSv1_1_ciphers': u"['AES128-SHA', 'AES256-SHA',
 'DES-CBC3-SHA', 'ECDHE-RSA-AES128-SHA',
 'ECDHE-RSA-AES256-SHA', 'ECDHE-RSA-DES-CBC3-SHA',
 'ECDHE-RSA-RC4-SHA', 'RC4-MD5', 'RC4-SHA']",
 'TLSv1_2_ciphers': u"['AES128-GCM-SHA256',
 'AES128-SHA', 'AES128-SHA256',
 'AES256-GCM-SHA384', 'AES256-SHA',
 'AES256-SHA256', 'DES-CBC3-SHA',
 'ECDHE-RSA-AES128-GCM-SHA256',
 'ECDHE-RSA-AES128-SHA', 'ECDHE-RSA-AES128-SHA256',
 'ECDHE-RSA-AES256-GCM-SHA384', 'ECDHE-RSA-AES256-SHA',
 'ECDHE-RSA-AES256-SHA384', 'ECDHE-RSA-DES-CBC3-SHA',
 'ECDHE-RSA-RC4-SHA', 'RC4-MD5', 'RC4-SHA']",
 'TLSv1_ciphers': u"['AES128-SHA', 'AES256-SHA',
 'DES-CBC3-SHA', 'ECDHE-RSA-AES128-SHA',
 'ECDHE-RSA-AES256-SHA', 'ECDHE-RSA-DES-CBC3-SHA',
 'ECDHE-RSA-RC4-SHA', 'RC4-MD5', 'RC4-SHA']",
 '_state': <django.db.models.base.ModelState object at 0x03154B10>,
 'bl_alien': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:65, result:unknown, assessment:',
 'bl_bfb': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:60, result:unknown, assessment:',
 'bl_chaos': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:60, result:unknown, assessment:',
```

```
 'bl_cleanmx': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:70, result:unknown, assessment:',
 'bl_dragon': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:65, result:unknown, assessment:',
 'bl_feodo': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:60, result:unknown, assessment:',
 'bl_malcode': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:60, result:unknown, assessment:',
 'bl_mdl': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:60, result:unknown, assessment:',
 'bl_openbl': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:60, result:unknown, assessment:',
 'bl_ptank': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:70, result:unknown, assessment:',
 'bl_spamhs': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:70, result:unknown, assessment:',
 'bl_zeus': u'detecttime:, updatetime:2015-09-01T00:26:12Z,
 alternativeid:, confident:65, result:unknown, assessment:',
 'blacklisted': 0,
 'blocked': False,
 'blockedmessage': u'',
 'city': u'Mountain View',
 'countrycode': u'US',
 'countryname': u'United States',
 'hostname': u'google.com',
 'info': False,
 'infomessage': u'',
 'ip': u'173.194.113.161',
 'regioncode': u'CA',
 'regionname': u'California',
 'sb_desc': u'',
 'sb_status': 204}


>>> c = Client.objects.get(id=54)
>>> pprint(vars(c))
{'_state': <django.db.models.base.ModelState object at 0x03239990>,
 'appCodeName': u'Mozilla',
 'appName': u'Netscape',
 'appVersion': u'5.0 (Windows NT 6.1)
```

```
        AppleWebKit/537.36 (KHTML, like Gecko)
        Chrome/44.0.2403.89 Safari/537.36',
 'blocked': False,
 'blockedmessage': u'',
 'cookieEnabled': True,
 'filterTypes': u'application/x-shockwave-flash',
 'id': 62,
 'info': False,
 'infomessage': u'',
 'ip': u'10.10.11.22',
 'javaEnabled': True,
 'language': u'de',
 'platform': u'Win32',
 'product': u'Gecko',
 'ssl23enabled': u'Enabled',
 'ssl2enabled': u'Disabled',
 'ssl3enabled': u'Disabled',
 'tls11enabled': u'Enabled',
 'tls12enabled': u'Enabled',
 'tls1enabled': u'Enabled',
 'sslfootprint': u'record version major: 03
        record version minor: 01
        hello version major: 03
        hello version major: 01
        ciphers: c0:0a:c0:14:00:39:c0:09:c0:13:00:33:
        c0:07:c0:11:00:35:00:2f:00:05:00:04:00:0a:00:
        ff:56:00
        extentions: 00:00:00:0f:00:0d:00:00:0a:31:30:
        2e:31:30:2e:31:32:2e:32:00:17:00:00:00:23:00:
        00:00:05:00:05:01:00:00:00:00:33:74:00:00:00:
        12:00:00:00:10:00:14:00:12:08:68:74:74:70:2f:
        31:2e:31:08:73:70:64:79:2f:33:2e:31:00:0b:00:
        02:01:00:00:0a:00:06:00:04:00:17:00:18',
 'user_agent': u'Mozilla/5.0 (Windows NT 6.1)
        AppleWebKit/537.36 (KHTML, like Gecko)
        Chrome/44.0.2403.89 Safari/537.36'}
```

```
>>>for plugin in c.plugins.all():
>>>   pprint(vars(plugin))

{'_state': <django.db.models.base.ModelState object at 0x03195770>,
 'allowed': True,
 'description': u'\r',
 'fileName': u'mhjfbmdgcfjbbpaeojofohoefgiehjai',
 'id': 87,
 'name': u'Chrome PDF Viewer'}
{'_state': <django.db.models.base.ModelState object at 0x03195C10>,
 'allowed': True,
 'description': u'Shockwave Flash 18.0 r0\r',
 'fileName': u'pepflashplayer.dll',
 'id': 88,
 'name': u'Shockwave Flash'}
{'_state': <django.db.models.base.ModelState object at 0x03195DB0>,
 'allowed': True,
 'description': u'This plugin allows you to securely access other
 computers that have been shared with you. To use this plugin you
 must first install the
 <a href="https://chrome.google.com/remotedesktop">
 Chrome Remote Desktop</a> webapp.\r',
 'fileName': u'internal-remoting-viewer',
 'id': 89,
 'name': u'Chrome Remote Desktop Viewer'}
{'_state': <django.db.models.base.ModelState object at 0x03195E10>,
 'allowed': True,
 'description': u'\r',
 'fileName': u'internal-nacl-plugin',
 'id': 90,
 'name': u'Native Client'}
{'_state': <django.db.models.base.ModelState object at 0x03195E50>,
 'allowed': True,
 'description': u'Portable Document Format\r',
 'fileName': u'internal-pdf-viewer',
 'id': 91,
 'name': u'Chrome PDF Viewer'}
```

Listing 5.8: Beispiel Befehle und Ausgabe in der Administrator Console

6 Fazit

6.1 Zusammenfassung

Die immer weiter fortschreitende Vernetzung in Arbeitswelt und Freizeit (Stichwort "Industrie 4.0" und "Internet of Things") ermöglicht enorme Produktivitätsfortschritte, aber ebendiese Vernetzung erlaubt es auch, dass die beteiligten Systeme angreifbar sind und Sicherheitslücken einen neuen "Berufstyp" schaffen: den Internetkriminellen. Die Schäden der Internetkriminalität gehen schon in die Milliarden, aber (v.a. Mittelstands- oder Klein-) Firmen haben bisweilen keinen Überblick, welche potentiellen Schwachstellen und Sicherheitslücken die im Unternehmen eingesetzten Rechner aufweisen.

Offen werden apokalyptische Bedrohungsszenarien diskutiert (Angriffe auf Energieversorgung und sonstige daseinsvorsorgende Infrastruktur). Aber auch ohne diese Szenarien im Hintergrund, muss sich die Erkenntnis durchsetzen, dass die Vorsorge und Prävention vor Internetkriminalität eine unabdingbare Voraussetzung zum weiteren Funktionieren eines offenen Netzes mit allgemeinem Zugang sein wird. Und diese Prävention erfordert Aufwand und ist ohne Zusatzkosten nicht zu haben. Diese "Zusatzkosten" sind auch im übertragenen Sinne zu sehen. Eine Geldautomatenverfügung dauerte vor 15 Jahren kaum 20 Sekunden. Heute, drei Betriebssysteme weiter, mit Skim-Schutz und mit Verschlüsselung zwischen den internen Devices, dauert eine Transaktion an die 40 Sekunden [interne Auskunft von Wincor Nixdorf]. Warum muss also der Aufruf einer Seite weniger als eine Sekunde dauern?

HTTP ist das meist verwendete Protokoll und über HTTP-Requests können initial Angriffspunkte in Client-Software gesetzt werden, bis hin zur möglichen vollständigen Überwachung und Steuerung dieses Clients.

Das Konzept der Web-Application Firewall (und deren kommerziellem Einsatz) setzt hier an, um Angriffe (auf der Anwendungsschicht) erkennen und abwehren zu können. Die Diskussion der OWASP Top10 zeigt aber, wie vielfältig die Angriffsszenarien sein können. Eine universelle Lösung zur Abwehr der verschiedensten Angriffe gibt es nicht. Es wird immer sinnvolle (und neuen Herausforderungen begegnende) Erweiterungen geben.

Der Prototyp ist aber nicht "umsonst" zu haben (im übertragenen Sinne). Er verlängert Antwortzeiten und er bedarf zusätzlicher Ressourcen. Wie aber in der Diskussion zu den Web-Application Firewalls gesehen, verlangen die entprechenden Rules (ModSecurity: Core Rule Set) zur Angriffsabwehr Know-How und müssen unter Umständen von extern bezogen werden. Eine Web-Application Firewall mag also auch Zeiten haben, in denen sie weniger Schutz bietet. Umso wichtiger ist es daher, nur Clients mit hohem Sicherheitslevel dort ins Netz zu lassen, wo sie auf gut administrierte Server treffen. Die Informationsgewinnung über Client und kontaktierten Server wäre auch als "Standalone" möglich gewesen. Diese Arbeit möchte jedoch einen größeren Zusammenhang herstellen und daher bereits im Einsatz befindliche Lösungen erweitern. Voraussetzung war, dass diese Systeme als quelloffen vorliegen und die Wiederverwendbarkeit der eingesetzten Software/Klassen gegeben ist.

Größeres Gewicht wurde auf das ICAP-Protokoll gelegt. ICAP-Server sind ebenfalls äußerst dienliche und mächtige Tools, um ein- und ausgehende HTTP-Transaktionen auf Angriffe oder Schadsoftware zu überprüfen. Mit der ICAP-Schnittstelle können auch externe Services eingebunden werden. Die Firma Kaspersky bietet zum Beispiel ein Anti-Viren Programm in Form eines ICAP Servers an, die so einfach in einen Proxy eingebunden werden können. ICAP kann nicht nur gegen Schadcode verwendet werden, sondern kann auch für *Data Loss Prevention* verwendet werden, wie es zum Beispiel von McAfee angeboten wird.

Der im Prototyp implementierte ICAP-Server weist hier nur geringe Funktonalität auf. Wichtig war die Einbindung überhaupt (und die Schnittstellendiskussion), damit eine reale Implementierung (siehe nächstes Kapitel) die Möglichkeit zur weiteren funktionalen Erweiterung der ICAP-Dienste hat.

Die in dieser Arbeit entwickelte Software ist ein Prototyp, der sich (weiterentwickelt) in eine angenommene / vielleicht real existierende Anwendungsarchitektur integrieren lässt. Einzig ein Webserver mit CGI-Unterstützung ist Voraussetzung für die Portabilität. Die Tests gegen die aufgebaute Testumgebung zeigen die prinzipielle Funktionstüchtigkeit (sammeln, administrieren, informieren, blockieren). Für die Administration der Client- und Server-Info-Tabellen wurde im Prototyp auf eine GUI verzichtet. Auch werden hinsichtlich Robustheit und Ausfallsicherheit noch nicht alle Optionen ausgeschöpft (z.B. Reaktion auf Downtime, Ausfall des DiscoveryServer u.a.). Auch machen Lasttests in der aufgebauten Testumgebung nur beschränkt Sinn. Ein kommerziell eingesetzter Apache-Server läuft selten auf Hardware zur Textverarbeitung.

6.2 Ausblick

Diese Arbeit sieht als architektonische Grundüberlegung eine maximale Anzahl an physichen Server-Instanzen vor (wenn im Prototypen auch als VM umgesetzt). Eine reale Implementierung (bzw. Integration) in eine vorhandene IT-Infrastruktur muss berücksichtigen, dass diese im Regelfall eine gewachsene IT-Struktur ist, mit wahrscheinlich diversen physischen Rechnern, die die im Prototypen verwendeten Serverinstanzen aufnehmen können.

Dies gilt umso mehr für den Aspekt des Alertings, der im Prototypen den virtuellen dauer-aufmerksamen Administrator voraussetzt. In der Regel wird gelten, dass in einer gewachsenen IT-Infrastruktur bereits Software-Lösungen für das Alerting eingesetzt werden. Es wurde im Prototypen bewußt darauf verzichtet, z.B. die Error/Alerting Logs der WAF und die Logs aus der hier entwickelten Funktionserweiterung bereits zu konsolidieren. Diese Konsolidierung erfolgt in einer realen Implementierung (bzw. Integration) gegen existiernde Alerting-Tools wie zum Beispiel HP Open View. Ein weniger umfangreiches aber geeignetes Tool wäre z.B. EventLog Analyser (von Manage Engine). Eine einfache aber verläßliche Methode ist das "Einhängen" und Scannen von Error-/Alerting-Logs und die Reaktion auf Schlüsselwörter. In den Alertings-Tools können Reaktionsmöglichkeiten definiert werden (Konsolen PopUp, Mail, ...), um eine schnelle administrative Reaktion zu ermöglichen.

Zudem können in den Alerting-Tools Trigger definiert werden (z.B. x-maliges Auftreten eines bestimmten Alerts innerhalb eines Zeitraums y). Die in der vorliegenden Implementierung durch den Administrator definierten Regeln (v.a. Blockierung von Seiten und Verschlüsselungsprotokollen) könnten fehlerhaft sein und ungewollte Blockierungen erzwingen. Hier kann ein variables Alerting sehr sinnvoll werden.

Auch für den ICAP-Server wurde eine eigene Server-Instanz im Prototypen vorgesehen. In einer realen Implementierung (bzw. Integration) wäre diese einzelne Funktion "billiger" zu haben. Sie könnte als Zusatzfunktion in die WAF-Erweiterung aufgenommen werden. Der Prototyp soll aber die Bandbreite der Eingriffsmöglichkeiten aufzeigen:

- Transaktion blockierende Funktionalität (WAF)
- Transaktion veränderde Funktionalität (ICAP)
- Sammeln von Zusatzinformationen über die Transaktion hinaus (Discovery)

Daher wurde im Prototypen (eine einzelne ICAP-Funktionalität exemplarisch darstellend) diese Funktion als Server ausgelagert. Ohne diese Auslagerung müßte eine reale Implementierung,je nach Anforderungen, mehr Code-Erweiterungen in die WAF aufnehmen.

Das Thema Ausfallsicherheit konnte im Rahmen des Prototypen nur ansatzweise erforscht werden. Prinzipiell ist eine Mehrfachauslegung von Serverinstanzen sinnvoll. Sie dient sowohl der Skalierung als auch der Ausfallsicherheit des Gesamtsystems. Auch die Fehlertoleranz des Gesamtsystems konnte im Prototypen nur ansatzweise verfolgt werden. Der DiscoveryServer und der InfoServer müssen als nicht obligatorische Instanzen definiert werden, sie sollen also den Weiterbetrieb des grundlegenden Systems nicht gefährden, sollten diese ausfallen. Natürlich gilt auch für diese Server, dass in einer realen Implementierung ein Alerting-Mechanismus installiert wird, der den Ausfall sofort anzeigt.

Priorisierte (Wunsch-)TODO-Liste bei einer realen Implementierung / Integration:

- Einbindung in existierende Alerting-Mechanismen/Tools
- Fehlertoleranz verbessern; ein Ausfall von DiscoveryServer und InfoServer darf nicht zum Ausfall des Gesamtsystems führen
- GUI für Administrator (ggf. auf Template-Basis zur Minimierung von Fehleingaben)
- Optionale Nutzung der ICAP-Möglichkeiten (incl. ggf. Ablösung von vorhandenen Insellösungen z.B. beim Virenscan)
- Einzelne im Prototypen verwendete Aufrufe/Funktionen können statt synchron auch asynchron umgesetzt werden (Performance)

Literatur

[BSI14] BSI. *Die Lage der IT-Sicherheit in Deutschland*. Techn. Ber. Bundesamt für Sicherheit in der Informationstechnik, 2014.

[Cal15] Jean-Paul Calderone. *pyOpenSSL Documentation*. 2015. URL: https://pyopenssl.readthedocs.org/en/stable/ (besucht am 24. 08. 2015).

[Che06] Checkpoint. *CVP API Specification*. Checkpoint, 2006.

[Con06] Web Application Security Consortium. "Web Application Firewall Evaluation Criteria". In: (2006).

[Fou15a] Apache Software Foundation. *Apache Documentation*. 2015. URL: http://httpd.apache.org/docs/ (besucht am 24. 08. 2015).

[Fou15b] Django Software Foundation. *Django Documentation*. 2015. URL: https://docs.djangoproject.com/ (besucht am 24. 08. 2015).

[ICS14] ICSALabs. "Web Application Firewall Certification Criteria". In: (2014).

[IET03] IETF. *RFC3507: Internet Content Adaptation Protocol (ICAP)*. 2003. URL: https://tools.ietf.org/html/rfc3507 (besucht am 24. 07. 2015).

[Nax15] Naxsi. *NAXSI Project Home*. 2015. URL: https://github.com/nbs-system/naxsi (besucht am 24. 08. 2015).

[Ope15] OpenSSL. *Cipher Suite Names*. Aug. 2015. URL: https://www.openssl.org/docs/manmaster/apps/ciphers.html.

[OPS15] OPSWAT. *Metascan Public API Documentation*. 2015. URL: https://www.metascan-online.com/public-api/ (besucht am 24. 08. 2015).

[Pro10] Open Web Application Security Project. "The ten Most Cirtical Web Application Security Risks". In: (2010).

[Pro13] Open Web Application Security Project. "The ten Most Cirtical Web Application Security Risks". In: (2013).

[Raz13] Abdul Razzaq. "Critical Analysis on Web Application Firewall solutions". In: (2013).

[Ris13] Ivan Ristic. *ModSecurity Handbook*. Feisty Duck Limited, 2013.

[Sic13] Bundesamt für Sicherheit in der Informationstechnik. *IT-Grundschutz Kataloge*. Sep. 2013. URL: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/kataloge.html.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 11. September 2015 Jonas Schäufler