Mail-Sniper

Developer Guidline – Entwicklerhilfe zur Mozilla Thunderbird Erweiterung

Entwickelt von

Martin Schreiner
(agentsway@users.sourceforge.net)

Sascha Strasser
(lucypher@users.sourceforge.net)

Inhalt

1. Mail-Sniper – Die Software

1.1 Kurzbeschreibung

Diese Software ist zur geolokalisierung von E-Mails gedacht. Allerdings lassen sich nicht nur Spam-Mails lokalisiert, sondern auch jede andere E-Mail kann einer Analyse mit Mail-Sniper unterzogen werden.

1.2 Funktionsweise

1.2.1 Auslesen der Mail-Headers

Diese Anwendung analysiert den Header der gewählten E-Mail nach der IP-Adresse des Absenders bzw. des letzten Mailservers.

1.2.2 Whois-Abfragen

Anhand dieser IP-Adresse werden iterativ an verschiedene Server, welche den Whois-Dienst anbieten, anfragen hinsichtlich dieser Adresse gestellt. Der Client wertet die Antworten der Server aus und filtert die gewünschten Informationen heraus.

1.2.3 Geodaten

Der Client besitzt vorerst eine XML Datei, welche die Geodaten sämtlicher Länder enthält. Resultierend aus den Whois-Anfragen wird der Ergebnisdatensatz aus der XML Datei gefiltert und das Ergebnis entsprechend visualisiert.

2. Anforderungen

2.1 Hardwareanforderungen

Minimale Hardwareanforderungen

Windows

- Pentium 233 MHz (*empfohlen:* Pentium 500MHz oder schneller)
- 64 MB RAM (*empfohlen:* 128 MB RAM oder größer)
- 52 MB freier Festplattenspeicher

Linux

- Pentium 233 MHz (*empfohlen:* Pentium 500MHz oder schneller)
- 64 MB RAM (*empfohlen:* 128 MB RAM oder größer)
- 52 MB freier Festplattenspeicher

2.2 Softwareanforderungen

Betriebssysteme

Windows

- Windows 98
- Windows 98SE
- Windows ME
- Windows NT 4.0
- Windows 2000
- Windows XP (*empfohlen*)

Linux

- Linux kernel - 2.2.14 mit minimal folgenden Bibliotheken oder Packeten:
- glibc 2.3.2
- gtk+ - 1.2.0 (1.2.5 oder höher empfohlen)
- XFree86-3.3.6
- Thunderbird wurde getestet auf Red Hat Linux 7.0 und neuer

Weiter Software

Mozilla Thunderbird

Das Produkt ist Plattformunabhängig. Es wird lediglich eine installierte Version des Mozilla Thunderbird E-Mail Clients vorausgesetzt, welches für Linux, Mac OSX sowie Microsoft Windows erhältlich ist. Somit haben wir keine Einschränkungen hinsichtlich des verwendeten Betriebssystems.

JSLIB

Damit der Client mit den Whois-Server kommunizieren kann ist eine Socketverbindung zwischen dem Client und den Servern notwendig. Das Mozdev Projekt JSLIB bietet eine solche Funktionsbibliothek. Daher ist es essenziell wichtig vorher diese Extension zu installieren. Das Paket sowie die Installationsanleitung wird unter http://jslib.mozdev.org gefunden.

3. Benutzeroberfläche (Frontend)

3.1 Kontextmenü-Eintrag

Um die eigentliche Benutzeroberfläche und somit das Mail-Sniper-Programm zu starten muss der Benutzer im Kontextmenü der Email den Eintrag „Zurückverfolgen mit Mail-Sniper“ anwählen. Dieser Eintrag wird mit der Datei `mailsniperOverlay.xul` realisiert, die einen Menuitem für das Popup „threadPaneContext“ enthält. Das genannte Popup beschreibt das Mail-Kontextmenü, und kann durch überschreiben um neue Eintäge erweitert werden. Somit kann der Benutzer Mail-Sniper über einen Rechtsklick auf die E-Mail starten.

Des Weiteren werden in der `mailsniperOverlay.xul` noch zwei andere Dateien eingebunden: Die `mailsniper.js` und die `mailsniperOverlay.dtd` aus dem locale-Verzeichis. Erstere stellt die Methode `openMailSniper()` bereit, welche das Hauptprogramm startet, die zweite Datei sorgt für das Bereitstellen sprachenspezifischer Werte für die Menüeinträge.

3.2 Mail-Sniper-GUI

Nachdem das Mail-Sniper-Fenster über die Methode `openMailSniper()` geöffnet und auf dem Bildschirm zentriert wurde, erfolgt automatisch die Auswertung der Sender-IP. Diese wird dann in einem Eingabefeld des Mail-Sniper-Eröffnungsscreens angezeigt und kann somit auch vom User überschriben werden, um eine andere IP zu analysieren.

Diese Benutzeroberfläche besteht im Wesentlichen aus der Datei `mailsniper.xul`. Diese setzt die verschiedenen Ansichten zusammen und importiert auch gleich alle ausgelagerten Formatierungen (aus der `mailsniper.css`), sowie den Standardskin des Messengers (aus der `prefPanels.css`). Bei den Ansichten handelt es sich um das derzeit eingesetzte IP-Eingabefeld (`inputFiledOverlay.xul`), der Weltkarte (`worldmapOverlay.xul`) und der Dateilansicht (`detailViewOverlay.xul`). Inhalt und Aufbau dieser drei Views wird weiter unten noch erklärt.

Ebenfalls hier eingebunden wird die Datei `menuOverlay.xul`, welche die Menüs dieser GUI bereitstellt.

Den Rest der Mail-Sniper-Hauptdatei stellt dann ein Window-Bereich dar, der neben Keyset und Commandset für das Menü noch eine Toolbar als Statusanzeige, sowie die Ansichts-Boxen der drei oben genannten Ansichten enthält bzw. aufruft.

Auch die verschiedenen Funktionsdateien (`mailsniper.js`, `whoisrequest.js` und `searchxml.js`) einschließlich der `jslib.js` werden in die `mailsniper.xul` eingebunden.

3.3 Das Menü

3.3.1 Aufbau der Menüleiste

Über das Mail-Sniper-Menü kann einerseits zwischen den drei Ansichten umgeschaltet werden, andererseits besteht hierüber auch die Möglichkeit Information zum Programm (Version, Webseite…) angezeigt zu bekommen oder das Programm zu beenden.

Hierzu gliedert sich die `menuOverlay.xul` in drei wesentliche Teile auf: Die Menubar selbst, das Keyset und das Commandset. Die Menüleiste enthält die drei Menüs (File, View, Info) mit ihrem jeweiligen Menupopup. Darin sind dann die einzelnen Menüeinträge mit ihren Attributen id, key, label, command und accesskey untergebracht. Die Beschriftungen der Elemente werden zum Teil aus der `mailsniperMenu.dtd` importiert, oder sind im Keyset festgelegt.
Das Keyset definiert also für jeden Eintrag der Menüs einen Namen (ID), eine Beschriftung (Label), ein Observes-Attribut (zum Ausführen einer Aktion), und Modifiers-Attribut (für die Tastenkombination).
Die Befehle, die beim Anklicken eines Menüeintrags ausgeführt werden sollen, werden dann im Commandset beschreiben. Hier bekommt jeder Command neben seiner ID ein Attribut „oncommand“ zugewiesen, das die auszuführende Methode enthält. Natürlich wird hierfür noch die `mailsniperMenu.js` in die Datei eingebunden.

3.3.2 Die Menüfunktionen

Die Datei `mailsniperMenu.js`, welche die über das Menü aufrufbare Funktionen enthält, besteht aus vier verschiedenen Funktionen. `ShowInputField()`, `showWorldmap()` und `ShowDetailView()` sorgen hier für den Wechsel der drei Ansichten, indem zuerst die Boxen per Element-ID aus dem Dokument geholt und in lokale Variablen gespeichert werden.

Danach wird dann jeweils das „hidden“-Attribut entsprechend der gewünschten Ansicht auf „true“ bzw. „false“ gesetzt.
Die Boxen inputFieldBox, worldmapBox und detailViewBox sind ja wie bereits angesprochen alle in der `mailsniper.xul` aufgerufen. Die soeben beschriebenen Methoden sorgen nun dafür, dass auch immer nur eine Ansicht sichtbar ist.

Als vierte Methode dieser Datei ermöglicht es `showInfo()` eine Alert-Meldung mit Informationen über Mail-Sniper anzuzeigen. Die lokale Variable der Methode enthält dabei einen String mit Infos zu Version, Autoren und Webseite des Tools.

3.4 Die verschiedenen Ansichten

Im Allgemeinen bestehen die Ansichten-Dateien immer aus einem Overlay, welches zur eindeutigen Unterscheidung (wie alle anderen Komponenten auch) mit einer ID versehen ist. Weiter werden noch zwei Dateien eingebunden, nämlich die `mailsniperViews.dtd`, welche die Werte der verwendeten (sprachabhängigen)Textvariablen enthält, und die `mailsniper.js` um die Funktionalitäten einzubinden. Hier nun die Dateien im Einzelnen:

3.4.1 Der Startscreen

Die `inputFieldOverlay.xul` enthält als wichtigstes Element die „inputFieldBox“, die ja bei gewählter Ansicht in der `mailsniper.xul` angezeigt wird. Des Weiteren ein Label als Überschrift, die Description als Informationstext, sowie drei weitere Elemente für die eigentliche Eingabe der IP-Adresse (Label, Textbox und Button). Letztere sind in diverse Boxen verpackt, um eine optisch ansprechende Ausrichtung zu erzeugen. Die „felx“-Werte dienen hierbei zur Gewichtung der Boxen bzw. Zellen.

Einzige verwendete Methode die dem Button zugeordnet ist, ist die Methode `startProcess()`, die später dem Klick in das Kontextmenüs der E-Mail zugewiesen wird.

3.4.2 Die Weltkarte

Bei der Weltkarten-Ansicht (`worldmapOverlay.xul`) werden in der „worldmapBox“ lediglich zwei Bilder in einem Stack übereinander gelegt. Zum einen die Weltkarte, zum anderen das Fadenkreuz, welches die Position des Mail-Senders anzeigen soll. Die Karte wird hier direkt in dieser xul-Datei ausgerichtet, wohingegen vom

Fadenkreuz lediglich die Größe definiert wird. Die Position bestimmt die Methode `setCrosshais()` der `mailsniper.js`.

3.4.2 Die Detail-Ansicht

`detailView.xul`: Die dritte mögliche Ansicht bei Mail-Sniper ist die Detailansicht. Hier wir die „detailViewBox“ zunächst in zwei Spalten eingeteilt: Links wird die passende Landeskarte, rechts die Textinformation angezeigt. Nicht nur die vertical Box des Textes soll sich an die Breite des Bildes anpassen, auch die Label mit der eigentlichen Information sollen passend angeordnet werden. Daher wurden hier alle Elemente in ein Grid (eine Tabelle) innerhalb einer horizontal Box verteilt, und die Gewichtung der Zellen wieder mit dem Flex-Attribut festgelegt. Die verwendeten Spacer-Elemente dienen ebenfalls der Ausrichtung.

Um die angezeigten Daten nicht editierbar erscheinen zu lassen werden die Ergebnisse auch in Form von Labeln angezeigt, deren Wert per Methode (`setInformation()`) festgelegt wird. Die restliche Beschriftung ist wie üblich per Entity eingefügt.

3.5 Ausgelagerte Designangaben

Abgesehen vom Zusammensetzen der einzelnen Komponenten (Reihenfolge, Gruppierungen…) und einigen Größenangaben wurden alle Designangaben in eine CSS-Datei ausgelagert. Die `mailsniper.css` importiert als erstes den allgemeinen Skin von Mozilla, und definiert danach Farben, Ränder und Schriften der Mail-Sniper-Anwendung. Aber auch die Abstände der einzelnen Komponenten werden hier genau festgelegt.

Weiter lädt die `mailsniper.css` alle Bilder der Extension in Form von `list-style-images` ein, um diese den Image-Elementen mit den speziellen IDs zuzuweisen. Lediglich die Detailkarte des Landes wird nicht hier, sondern in der Methode `setInformation()` der `mailsniper.js` bestimmt.

Das Zuweisen des Designs zu den verschiedenen Komponenten geschieht wie üblich bei CSS über die Element-ID.

3.6 Sprachspezifische Variablenwerte

Um diese Erweiterung auf einfache Art und Weise in mehreren Sprachen anbieten zu können, wurden alle sprachspezifischen Angaben/Beschriftungen in DTD-Dateien ausgelagert. Hier enthalten nun also die Dateien `mailsniperOverlay.dtd`, `mailsniperMenu.dtd` und `mailsniperViews.dtd` die Textelemente in Form von Entities. Ein Entity ist wie folgt aufgebaut:

`<!ENTITY [id]  "[Wert]">`, also beispielsweise:
`<!ENTITY menu-info:label  "Info">`

Die Werte für den Kontextmenü-Eintrag sind dabei in der Datei `mailsniperOverlay.dtd` enthalten, alle Einträge der Mail-Sniper-Menüleiste in der `mailsniperMenu.dtd` und die Werte für Label, Texte oder Überschriften der einzelnen Ansichten enthält die `mailsniperViews.dtd`.

Derzeit ist Mail-Sniper in zwei verschiedenen Sprachen verfügbar: Englisch (en-US) und Deutsch (de-DE).

4. Funktionalität (Backend)

4.1 Grundlegende Methoden

Die meisten grundlegenden Funktionen werden in der Datei mailsniper.js umgesetzt. Dies enthält auch einige globale Variablen, in denen nach der Analyse die Ergebnisse gespeichert werden. Vorhanden sind:

- `ip_address` (auszuwertende IP-Adresse)
- `domain` (Toplevel-Domain zur IP)
- `gns_name` (GNS Kurzbeschreibung)
- `gns_name_full` (kompletter GNS-Name)
- `display_country` (Ländername)
- `area` (Beschreibt das Gebiet, den Kontinent)
- `longitude` (Längengrad)
- `latitude` (Breitengrad)

Diese Variablen werden vor jeder Analyse

4.1.1 OpenMailSniper()

In dieser Datei ist auch die bereits angesprochene Methode `openMailSniper()` vorhanden, die für das Öffnen des Mail-Sniper-Programms zuständig ist. Vor dem eigentlichen Öffnen dieser Extension muss natürlich erst einmal die IP aus dem Header der E-Mail ausgelesen werden, was die Methode `getIP()` umsetzt. Das Ergebnis der Auswertung wird dann in der globalen Variable `tmp_ip` für alle anderen Methoden abgelegt.

Nach einer Zeitverzögerung von 100 ms kann jetzt das eigentliche Fenster von Mail-Sniper geöffnet werden, und die ausgelesene IP-Adresse in das dafür vorgesehene Eingabefenster geschrieben. Zur Aktualisierung wird diese Ansicht dann noch einmal geladen. Nun kann der Benutzer den Analysevorgang zur Mail-IP per Knopfdruck starten.

Die Verzögerungen werden durch die Javascript-Methode `window.setTimeout(„Anweisung“, verzögerungInMs)` realisiert, und sind notwendig um die Objekte der Anwendung auch erzeugt zu bekommen bevor die Zuweisungen erfolgen.

4.1.2 OpenMailSniperWindow()

In dieser Datei wird das Öffnen des Mail-Sniper-Fensters realisiert. Dies geschieht über die Javascript-Funktion `window.open()`, der sowohl die Beschreibungsdatei der Extension, als auch ein Name und weitere Eigenschaften (Breite, Höhe, nicht größenänderbar, abhängig von Thunderbirs-Fenster) mit übergeben werden.
Damit das neu geöffnete Fenster stets zentriert angezeigt wird, muss es beim Erstellen zuerst der Variable `msWindow` zugewiesen, und nach dem Auslesen der Bildschirmauflösung (über `screen.width` und `srceen.height`) an eine neu errechnete Position geschoben werden. Diese Aufgabe erledigt die Methode `moveTo(position_x, position_y)`.

4.1.3 StartProcess()

Mittels der Funktion `startProcess()` wird die Datenauswertung gestartet. Derzeit ist der Aufruf dieser Methode noch dem Button des IP-Eingabeformulars zugewiesen, wird später aber direkt beim Klick auf den Kontextmenü-Eintrag ausgeführt.
`StartProcess()` setzt zunächst den Status der Abfrage (über den Wert des Labels `statusMsg`) auf „Mail Sniper startet“. Danach wird der Variablen `mailIP` die IP-Adresse der E-Mail zugewiesen. Das Auslesen dieser übernimmt hierbei die Methode `getMailIP()`.
Als Nächste muss eine Instanz der Klasse `whoisrequest` erzeugt und als lokale Variable `req` gespeichert werden.
Der Konstruktor der Klasse `whoisrequest` benötigt keine weiteren Parameter. Die Instanzvariable `isInitialized` prüft ob ob ein Socket Objekt erstellt wurde. Das Socket Objekt wird mittels der JsLib Erweitung zur Verfügung gestellt. Wenn dies ohne Schwierigkeiten generiert wurde, gibt diese Variable einen boolschen Wert zurück, welcher in einer Selektion abgefragt wird. Je nach Zustand wird die Anfrage an den Whois-Server gestartet oder nicht.
Die Methode `sendrequest` des Objekts `whoisrequest` benötigt zur korrekten Anfrage folgende Parameter.IP-Adresse, erster Whois-Server und Port.

4.1.4 GetMailIP()

Da Benutzer bei Mail-Sniper ja auch andere IPs eingeben und analysieren können, liest die Methode `getMailIP()` diese Daten aus dem Eingabefeld aus. verwendet. Vor diesem Prozess werden alle globalen Variablen der Datei initialisiert (geleert), um keine alten Werte angezeigt zu bekommen, wenn eine Analyse ergebnislos bleibt.

Nun erfolgt das eigentliche Auslesen, durch das Abfragen des Textfeldes `ipEntry` des inputFieldViews. Die lokal gespeicherte IP-Adresse wird nach dem Auslesevorgang der Methode `startProcess()` zurückgegeben.

4.1.5 StoreGlobal()

Diese Methode startet den Suchalgorithmus welcher in der Klasse `searchXML` implementiert ist. Über `new searchXML()` wird ein Objekt erzeugt. Nun steht die Funktionalität der Klasse `searchXML` der Methode `storeGlobel` zur Verfügung. Das Ergebnis wird von dem Objekt `searchXML` in ein Array gespeichert welches nun ausgewertet wird. Die Elemente des Arrays werden nun von der Methode `StoreGlobal()` global abgelegt, damit andere Methoden darauf zugreifen können. Schlussendlich wird die Methode `showWorldmap()` aufgerufen, welche diese globalen Daten nun ausliest und visualisiert.

4.1.6 SetInformation()

Nach der Analyse müssen die Ergebnisse den entsprechenden Elementen der Mail-Sniper-GUI zugewiesen. Hierfür wird der `Value` der Elemente ip, continent, land, longi und lati mit den Werten der entsprechenden globalen Variablen belegt.
Weiter wird in dieser Methode noch die passende Detailkarte des Landes ausgewählt. Die URL zur Karte setzt sich aus einer Teil-URL und der Domain zusammen. Der String wird in der lokalen Variablen `pic` gespeichert, und danach dem `detailMap`-Element als `listStyleImage` zugewiesen. Auch das Fadenkreuz wird als lokale Variabel gespeichert (`noCross`) und erst einmal angezeigt (`noCross.setAttribute("hidden","false");`).
Nun erfolgt noch eine Abfrage, ob auch wirklich eine Karte zum Ergebnis vorhanden ist, oder ob der Wert der Variablen `domain` „undefined“ bzw. leer war. Sollte dies der Fall sein wird ein Ersatzbild anstelle der Karte eingesetzt und das Fadenkreuz ausgeblendet.

4.1.7 SetCrosshairs()

Diese Methode dient zum Positionieren des Fadenkreuzes. Im Methodenanfang befinden sich einige Pixelangaben für bestimmte Bereiche der GUI. So werden je X- und Y-Wert des Ursprungs (0° Länge, 0° Breite) in den Variablen `originLongi` und `originLati` festgelegt, aber auch für die Kartenränder der Weltkarte werden die

Pixelwerte innerhalb der Mail-Sniper-GUI in den Variablen `myTop`(oben), `myBottom`(unten), `myLeft`(links) und `myRight`(rechts) gespeichert.
Danach erfolgt die Auswertung der Analyseergebnisse für Längen- und Breitengrad durch Interpolation vom Ursprung zum Kartenrand hin. Hier ist das Vorzeichen relevant dafür, zu welchem Rand interpoliert wird. Außerdem sitzt der Ursprung nicht mitten in der Weltkarte, sonder leicht nach links versetzt. Somit muss der Längengradbereich der am rechten Rand (aber mit negativem Vorzeichen) liegt separat abgefragt und interpoliert werden.
Zum Schluss wird nur noch das Fadenkreuz-Objekt in eine Variable ausgelesen und in der Position geändert. Nun wird die richtige Position auf der Weltkarte angezeigt.

4.2 Kommunikation mit dem Whois-Server

Für die Kommunikation mit den Whois-Servern sowie die Auswertung der Server Antworten wird die Datei `whoisrequest.js` verwendet. Innerhalb dieser Datei befinden sich im Prinzip zwei Klassen bzw. Pseudoklassen, da Javascript nicht wirklich Objektorientiert ist. Einmal die Klasse `whoisrequest`, welche die Kommunikation zum Server übernimmt. Zum anderen die Klasse `AsyncObserver`, welche die Antworten des Servers entgegen nimmt und diese auswertet, um schlussendlich die gesuchten Informationen zu extrahieren.
Die Pseudoklasse `whoisrequest` stellt über die zuvor eingebundene Erweiterung JsLib eine Socketverbindung zum Whois-Server her. Über die Methode `sendRequest` wird ein Socket Objekt erstellt und danach die extrahierte IP-Adresse an den Whois-Server ARIN gesendet. Für den Fall, dass dieser Server die gesuchte IP mit ihren entsprechenden Daten nicht in seiner Datenbank hat, bietet dieser eine Referral Information an, welche angibt auf welchen Server sich die gesuchte Information befindet. Natürlich muss der Anfrage Alogrithmus dies berücksichtigen. Sofern innerhalb des Datenstroms, welcher von dem Server zurück an den Client gesendet wird, diese Referral Information befindet, wird über einen regulären Ausdruck diese URL herausgefiltert und einen neue Anfrage an den neuen Server geschickt. Dies wird entweder RIPE, APNIC oder LANIC sein. Gibt nun der Server die Information zur gesuchten IP-Adresse zurück wird auch über einen regulären Ausdruck die Top-Level-Domain aus dem Datenstrom gefiltert. Anhand dieser Top-Level-Domain wird später in der XML Datei nach den Geodaten gesucht.

4.2.1 Whoisrequest

4.2.1.1 Konstruktor whoisrequest()

Dieser Konstruktor bindet die externe Datei `socket.js` der Extension JsLib ein. Diese ist für die Socket Kommunikation zwingend erforderlich. Des Weiteren wird das `SOCKET_XUL_ASYNC` Flag auf `true` gesetzt. Dies macht eine asynchrone Kommunikation mit dem Server möglich. Erst wenn dies geschehen ist, wird ein Socket Objekt erstellt. Nun ist die Klasse `whoisrequest` bereit zur kommunikation mit dem Server und setzt die Instanzvariable `isInitialized` auf.

4.2.1.2 Whoisrequest.sendRequest(IP,Whois-Server,Port)

Diese Methode benötigt drei Parameter. Die gesuchte IP, den Whois-Server und den dazugehörigen Port. Die Methode öffnet nun über die Externe Methode `socket.open(host,port,true)` einen entsprechende Verbindung zum Server. Da es manchmal dauern kann, bis eine Verbindung zustande kommt, darf nicht sofort eine Nachricht an den Server gesendet werden, da dies sonst einen Fehler verursacht. Deswegen wird über die Methode `socket.isAlive()` zyklisch überprüft ob die Verbindung besteht. Hierbei wird auch die Methode `delay(seconds)` verwendet was die Anzahl der notwendigen Überprüfungen reduziert. Da das `SOCKET_XUL_ASYNC` Flag auf wahr gesetzt wurde, kommt hier auch die Klasse `AsyncObserver` ins Spiel und es wird eine Instanz dieser Klasse erzeugt. Die Funktionalität dieser Klasse wird später beschrieben. Über die externe Methode `socket.write(Message)` wird die IP an den Whois-Server übermittelt.

4.2.1.3 Whoisrequest.delay(seconds)

Diese einfache Methode lässt eine über seconds eingestellte Zeit verstreichen bevor die nächste Operation ausgeführt werden kann.

4.2.2 AsyncObserver

Diese Klasse ist notwendig um asynchron mit dem Whois-Server zu kommunizieren. Diese Methoden werden über das Socket Objekt der Extension JsLib gesteuert. Es gibt die Methode `AsyncObserver.receivedData(theData)` welche gestartet wird wenn der Client

eine Antwort vom Server erhält. Die anderen beiden Methoden, `AsyncObserver.streamStarted(theSocket)`und `AsyncObserver.streamStopped(theSocket, status)` reagieren auf den entstehenden Datenstrom zwischen Client und Server.

4.2.2.1 Konstruktor AsyncObserver(IP)

Der Konstruktor initialisiert lediglich zwei Variablen. `showCountry` und `ipaddress`. Wichtig hierbei ist, dass der Konstruktor die gesuchte IP Adresse mitbekommt, da diese bei einer weiteren Anfrage durch das AsyncObserver Objekt gestartet wird.

4.2.2.2 AsyncObserver.streamStarted(theSocket)

Diese Methode wird gestartet wenn ein Datenstrom vom Client zum Server gesendet wird. Diese Methode wird genutzt um einen Status der Übermittlung in der Statusleiste anzuzeigen.

4.2.2.3 AsyncObserver.streamStopped(theSocket,status)

Wie auch die zuvor beschriebene Methode reagiert diese auf den Datenstrom. Aber im Gegensatz zur `AsyncObserver.streamStarted()`, wird diese aktiviert wenn der Datenstrom beendet ist.
Dazu wird der Socket über die Methode `socket.close()` auch geschlossen.

4.2.2.4 AsyncObserver.receiveData(theData)

Diese Methode wird aufgerufen wenn eine Antwort vom Whois-Server an den Client gesendet wird. Das Ergebnis wird der Methode über den Parameter `theData` übermittelt. Um diesen zu parsen muss er in den Datentyp String umgewandelt werden.
Diese Methode hat nun zwei Aufgaben. Neben der Aufgabe die entsprechende Länderinformation über den regulären Ausdruck `CountryExp` herauszufiltern, muss sie auch auf einen Weiterleitung zu einem anderen Whois-Server reagieren. Wie bereits schon angesprochen wir auch nach dem ReferralServer Eintrag gesucht (`ReferralServerExp`). Falls dieser nun vorhanden ist, wird in dieser Methode einen neue Anfrage an den neuen Server erstellt und abgeschickt. Wenn kein ReferralServer Eintrag vorhanden ist,

handelt es sich um den richtigen Server und die Länderinformation kann entnommen werden. Die Länder-information `Country` wird danach an die Methode `storeGlobal(Country,IP)` übergeben.

4.3 Auswerten der XML-Datei

Nachdem der IP-Adresse eine Top-Level-Domain zugeordnet wurde, geht der Client über, diese TLD in seiner eigenen XML-Datei zu suchen. Hierzu verwendet er die Datei `searchXML.js` welche das Laden einer XML Datei beinhaltet und einen entsprechenden Suchalgorithmus für diese. Die XML Datei ist in die einzelnen Anfangsbuchstaben kategorisiert, was ein Suchen über sämtliche Einträge verhindert. Der Parser sucht zuerst den Anfangsbuchstaben der TLD und fängt dort an die Einträge sequentiell zu vergleichen, bis er die gesuchte TLD findet. Die gefundenen Attribute werden gekapselt in einem Array an das Hauptskript zurückgegeben.

4.3.1 Konstruktor searchXML

Der Konstruktor der Klasse startet lediglich die Initialisierungs-methode searchXML.init()

4.3.2 SearchXML.init()

Diese Methode initialisiert den Pfad zur XML Datei (`filepath`) sowie das Array `specificTLDData`, in dem die gefundenen Attribute gespeichert werden.

4.3.3 SearchXML.loadXML()

Diese Methode ist zuständig für das Laden einer XML Datei. Diese geschieht über `this.xmldoc.load(this.filepath,"xml/text")`. Wobei der `filepath` bereits in der Initialisierung festgelegt wird. Falls der Algorithmus keine Datei finden sollte gibt die Methode eine `false` zurück was das Programm erkennen lässt das ein Fehler aufgetreten ist.

4.3.4 SearchXML.getTLDData(tld,ip)

Dahinter verbirgt sich der Parsing Algorithmus. In einer Schleife wird zuerst nach dem Anfangsbuchstaben der TLD gesucht. Wurde dieser gefunden geht der Alogithmus sequentiell die einzelnen Elemente der XML Datei durch bis die entsprechende TLD mit ihren Geodaten gefunden wurde. Diese Attribute werden dann in dem Array `specificTLDData` gespeichert, damit anderen Methoden auf das Ergebnis zugreifen können.

5. Kontakt

5.1 Webseite

Die Webseite dieser Extension ist unter folgender URL zu erreichen:
http://mail-sniper.sourceforge.net

Dort erhalten Sie weitere Informationen zu Mail-Sniper, wie Benutzerdokumentation oder andere Sprachpakete der Software.

5.2 Forum

Ein Forum rund um Mail-Sniper befindet sich hier:
http://sourceforge.net/forum/?group_id=122595

5.3 E-Mail

Martin Schreiner:
agentsway@users.sourceforge.net

Sascha Strasser:
lucypher@users.sourceforge.net