Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelorarbeit

Entwicklung eines Visualisierungswerkzeuges zur Demonstration datenschutzfreundlicher Dokumentspeicherdienste

vorgelegt von

David Kirchhausen Monteiro

geb. am 24. Januar 1994 in Hildesheim

Matrikelnummer 6530927

Studiengang Software-System-Entwicklung

eingereicht am 10. Juli 2018

Betreuer: Maximilian Blochberger, M. Sc.

Erstgutachter: Prof. Dr.-Ing. Hannes Federrath

Zweitgutachter: Tilmann Stehle, M. Sc.

Aufgabenstellung

Im Zuge dieser Bachelorarbeit soll ein einfacher Dokumentenspeicher entwickelt werden, welcher möglichst viele Nutzerdaten erfasst und speichert. Die erfassten Daten sollen anschaulich grafisch dargestellt werden können. Weiter sollen verschiedene Szenarien entwickelt werden, welche aufzeigen wie eine mögliche Benutzung des Services mit und ohne der Verwendung von datenschutzfreundlichen Methoden zum Anonymisieren von Daten aussieht. Anhand der Szenarien soll eine grafische Auswertung Unterschiede zwischen anonymisierten Daten und nicht anonymisierten Daten visuell sichtbar machen und die Unterschiede somit leicht zugänglich sein.

Zusammenfassung

1. Dokumentenspeicherdienste Vorteile (Problemstellung erläutern)
2. Mögliche Datenschutz unfreundliche Aspekte von gängigen Anbietern (Problemstellung erläutern)
3. Entwicklung des Dokumentenspeichers und der Visualisierung zur deutlich Veranschaulichung von Potentiellen Unterschieden zwischen der Verwendung von Datenschutz freundlichen Methoden zum Anonymisieren oder nicht. (Bearbeitung der Problemstellung)
    a) Implementation des Dokumentenspeichers
    b) Implementation der API zur Datenübergabe
    c) Implementation des Visualisierungswerkzeug
    d) Darstellung der Szenarien zur Benutzung des Visualisierungswerkzeug

Inhaltsverzeichnis

1 Einleitung

Dokumentenspeicherdienste sind nützliche Alltagsgegenstände, welche für private sowie kommerzielle Nutzer meist unverzichtbar sind. Sie bieten nicht nur den Speicherplatz für wichtige Dateien der Nutzer sondern stellen auch die Sicherheit der Dateien sicher und machen sie global jeder zeit verfügbar. Durch die große Datensammlung dieser Dienstleister machen sie sich nicht nur selbst zu lukrativen Zielen von gezielten Angriffen (Yahoo, UBER etc.), jedoch auch die Dienstleister selber können die Daten auswerten und meitere Meta-Daten über die Nutzer sammeln und weiter verwenden. Vor allem private Nutzer sind meist gar nicht über die Risiken und das Missbrauchspotenziale aufgeklärt, welche die Verwendung solcher Dienstleistungen mit sich bringen. Methoden zur Verschlüsselung oder des Anonymisieren von Daten sind Benutzern meist nicht bekannt, werden von den Dienstanbietern nicht angeboten oder sind schwer umzusetzen da es einen meist erheblichen Aufwand für die Benutzer bedeutet und Kompetenzen erfordert welche diese Benutzer nicht besitzen. Um genau die Risiken und Missbrauchspotenziale aufzuzeigen wird im Zuge dieser Arbeit ein Dokumentenspeicherdienst entwickelt, welcher prinzipiell alle Meta-Daten der Nutzer sammelt und diese in einer visuellen Darstellung zusammen fasst. Zur Implementation des Dokumentenspeichers wird dabei das Microsoft ASP.NET Core Framework verwendet. Das Framework wird benutzt um die Webbenutzeroberfläche sowie der Web-APIs des Dokumentenspeichers zu realisieren. Dazu wird das Javascript Framework Data-Driven Documents, i.d.R. d3.js genannt, zur Visualisierung der Daten verwendet. Der Dokumentenspeicher soll vor allem den Unterschied zwischen der Verwendung von Methoden zur Verschlüsselung oder des Anonymisieren von Daten visualisieren und verwaltet dazu zwei verschiedene Datensätze, wobei eine Datenmenge ohne, und eine Datenmenge mit der Verwendung von Methoden zur Verschlüsselung oder des Anonymisieren von Daten erzeugt wird. Der entstehende Unterschied der gesammelten Meta-Daten durch die verschiedenen Methoden führt dann zu einer Veränderung in der Visualisierung was dan den Effekt und Nutzen der Methoden deutlich sichtbar macht.

2 Grundlagen

2.1 Terminologie

1. HTTP-Header
   a) Teil des Hypertext Transfer Protocol (HTTP)
   b) Headerblock und Headerfeld nach RFC 2616(https://tools.ietf.org/html/rfc2616)
2. Geo lookup
   a) eine Methodik zur Bestimmung des geografischen Standort einer IP-Adresse
   b) keine eigenen Implementation der Standortbestimmung, sondern Verwendung eines öffentlich zugänglichen Service, aus Zeit und komplexitätsgründen
   c) Patent US7752210B2
3. Header Fingerprint
   a) eine Methodik zur Identifikation eines Benutzers anhand der von ihm Verwendeten HTTP Header
   b) Aggregation über allen Http-Header oder einem ausgewählten Set an Headerfeldern
   c) erzeugt Fingerprint wird gehasht und gespeichert
   d) bei übereinstimmenden Fingerprints wird angenommen das diese vom gleichen Benutzer erzeugt wurden

2.2 Set A / Set B

Die angeführten DbSets FileEntryItemsA und FileEntryItemsB verfügen jeweils über Daten welche zu visualisieren sind. Dabei wird das DbSet FileEntryItemsA weiterhin als die ungeschützte Datenmenge beschrieben, da diese Datenbank nur mit Daten befüllt werden soll, bei welchen keine datenschutzfreundlichen Methoden zum anonymisieren verwendet wurden. Das DbSet FileEntryItemsB wird weiterhin als geschützte Datenmenge beschrieben, da ausschließlich Daten , welche mit Methoden zum anonymisieren hochgeladen wurden, verwendet werden sollen. Dabei sollte die geschützte und ungeschützte Datenmenge in den Zugrunde liegenden Datenmenge identisch sein und lediglich die Verwendung von Methoden zum anonymisieren die Datensätze unterscheiden, sodass die beiden Datenmenge miteinander in Hinblick auf den Effekt der Verwendung von Methoden zum anonymisieren untersucht werden können. So wird im jedem beschriebenen Szenario zur Benutzung des Dokumentenspeichers angenommen das ein Benutzer eine Datei einmal ohne die Verwendung von Methoden zum anonymisieren in die ungeschützte Datenmenge via HTTP hochlädt und einmal mit der Verwendung von Methoden zum anonymisieren in die geschützte Datenmenge via HTTP hochlädt.

Das führt dazu das bei der Auswertung der Datei durch die Verwendung oder nicht Verwendung von Methoden zum anonymisieren, die erfassten Meta-Daten sich nur durch die jeweilige Art der Methoden zum anonymisieren unterscheiden und die anschließenden Visualisierung nur diesen Unterschied darstellt und keine anderen Faktoren.

3 Hauptteil

3.1 Implementation des Dokumentenspeichers

Der Dokumentenspeicher wurde mit Hilfe des ASP .Net Core Framework erstellt. Das Framework ist Microsofts aktuellste corss-platform Framework zur Realisierung von Web-Anwendungen. Das Framework unterstützt alle gängigen Betriebssysteme wie Windows, Mac OS und Linux durch die Verwendung des "Roslyn"Compilers. Mit dem ASP .Net Core entwickelte Web-Anwendungen lassen sich in gängige Hostingplatformen wie, z.B. das IIS von Microsoft, integrieren oder auch in einem eigenen Prozess selbst gehostet werden. Das ASP .Net Framework sieht dabei eine MVC-Architektur, der Projekte vor und verwenden diese Architektur intern zum Realisieren der Anwendungen. Modelle werden in diesem Kontext als Objekte zur Daten Repräsentation verstanden. Views sind die HTML Pages welche die Klienten ausgegeben werden. Controller sind Zentrale Elemente. Sie stellen die Funktionalität von Views auf der Server Seite dar. Sie sind zur Steuerung verschiedener Routen und die Implementation von API-Punkten vorgesehen. Dabei Verwalten die Controller ebenfalls die zu Grunde liegenden Datenbanken.

Der Entwickelte Dokumentenspeicher besteht aus einer Controller-Klasse, welche verschiedene API-Punkte implementiert. Die API-Punkte werden durch Routen angesprochen, welche vorher definiert worden. Die Implementierten Routen sind:

/api/GetA HTTP Get Methode welche die Daten des Set A, dem ungeschützten Datensatz zurückgibt

/api/GetB HTTP Get Methode welche die Daten des Set B, dem geschützten Datensatz zurückgibt

/api/uploadA HTTP Post Methode zum hochladen von ungeschützten Dateien ins Set A

/api/uploadB HTTP Post Methode zum hochladen von geschützten Dateien ins Set B

api/uploadEmu HTTP Post Methode zum erzeugen von Dummy Daten

In der Controller Klasse ist auch für das Sammeln der Meta-Daten für die hochgeladenen Dateien zuständig und speichert diese mit Hilfe der Modell Klasse in einem Datenbankschema ab. Die Modell Klasse besitzt verschiedene Propertys welche die gesammlten Meta-Daten widerspiegeln. Die Propertys sind :

ID Datenbank Index

Set Das Set bezeichnet die Gruppe welcher die Datei zugeordnet wurde

Filename Der Dateiname

Filepath Der Pfad zur gespeicherten temporären Datei

Size Die Dateigröße in Byte

IPAddress Die IP-Adresse von der die Datei hochgeladen wurde

Headers Ein String bestehend aus den Headern der Datei

HeaderFingerprint Ein zusammenschluss aus ausgewähten Headern um eine möglichst eindeutige Signatur zu erzeugen

DateTime Als Zeitstempel für das Hochladen der Datei

Country Land aus welches die Datei hochgeladen wurde

RegionName Region (Bundesland) aus welches die Datei hochgeladen wurde

City Stadt aus welches die Datei hochgeladen wurde

Lat Breitengrad welcher mit den bekannten IP-Adresse assoziiert wird

Lon Längengrad welcher mit den bekannten IP-Adresse assoziiert wird

Isp Der Internetanbieter welcher der IP zugeordnet ist

Diese Propertys enthalten alle gesammelten Meta-Daten und können zur Visualisierung verwendet werden. Mit Hilfe des Entity Framework Core von Microsoft wird ein Datenbankschema erzeugt was dieser Modell Klasse entspricht. Dabei wird eine SQLite Datenbank Datei erzeugt die dem Datenbankschema entspricht und im Verzeichnis der Anwendung liegt. Im Gegensatz zur Verwendung eines festen SQL-Server der die Datenbank verwaltet, bietet die Anwendung so mehr Flexibilität und erlaubt es mehrere Datenbank Dateien mit der Anwendung bereit zu stellen, um so verschiedene Szenarien in verschiedenen Dateien bereitzustellen.

Der Dokumentenspeicher besitzt 3 Views, welche HTML Pages darstellen welche ein Benutzer über bestimme Routen aufrufen kann.

/FileEntry Anzeige der Datenbank in Tabellarischer Form

/FileEntryCreate Bietet Möglichkeiten zum hochladen von Dateien oder dem erzeugen von Dummy Daten

/FileEntryVisual Visualisierung der gesammelten Daten

Die FileEntry-Page ist ist die StartPage der Web-Anwendung und verweist zu der FileEntry-Create und FileEntryVisual Page. Die Page zeigt lediglich die Datenbank in tabellarischer Form und ist hauptsächlich für die Entwicklung gebraucht worden. Dies gilt auch für die FileEntryCreate-Page. Die FileEntryVisual-Page ist der Hauptbestandteil dieser Arbeit und stellt die verschiedenen Visualisierungsmöglichkeiten dar.

Die Implementation des Dokumentenspeichers ist mit dem ASP.NET Core Framework von Microsoft umgesetzt worden.
„ASP.NET Core ist ein plattformübergreifendes, leistungsstarkes Open-Source-Framework zum Erstellen moderner, cloudbasierter mit dem Internet verbundener Anwendungen.“
Der Dokumentenspeicher ist mittels einer Model-View-Controller Architektur realisiert worden. Die Model-View-Controller Architektur definiert, Modelle als Klassen welche die zugrundeliegende Problemstruktur wiedergeben. Sie modellieren die Objekte welche keine Informationen über ihre Verwendung besitzen. Views sind die Klassen welche die Darstellung der Modelle implementieren und den Output einer Systems darstellen. Controller sind die Kontroll-Klassen welche Views steuern und Input in das System verarbeiten.

Das ausschlaggebende Modell im Dokumentenspeicher ist die FileEntryItem Klasse. Die Klasse definiert alle Porperties, welche die Meta-Data wieder spiegeln, welche vom Dokumentenspeicherdienst gesammelt werden. Mit Hilfe des Entity Framework Core von Microsoft werden aus dieser Modell-Klasse ein Datenbankschema erzeugt. Das Datenbankschema ist somit an die Modell-Klassen gekoppelt und führt dazu das Änderungen an den Modell-Klassen auch direkt Änderungen des Datenbankschemas erfordern, welche vom Entity Framework Core einfach erzeugt werden können. Das erzeugte Datenbankschema für FileEntryItem ist.

```
1         public int ID { get; set; }
2         public string Set { get; set; }
3         public string Filename { get; set; }
4         public string Filepath { get; set; }
5         public long Size { get; set; }
6         public string IPAddress { get; set; }
7         public string Headers { get; set; }
8         public string HeaderFingerprint { get; set; }
9         public DateTime DateTime { get; set; }
10        public string Country { get; set; }
11        public string RegionName { get; set; }
12        public string City { get; set; }
13        public float Lat { get; set; }
14        public float Lon { get; set; }
15        public string Isp { get; set; }
```

Die einzelnen Properties werden für bestimme Informationen verwendet.

1. ID: Datenbank Index
2. Set: Das Set bezeichnet die Gruppe welcher die Datei zugeordnet wurde
3. Filename: Der Dateiname
4. Filepath: Der Pfad zur gespeicherten temporären Datei
5. Size: Die Dateigröße in Byte
6. IPAddress: Die IP-Adresse von der die Datei hochgeladen wurde
7. Headers: Ein String bestehend aus den Headern der Datei
8. HeaderFingerprint: Ein zusammenschluss aus ausgewähten Headern um eine möglichst eindeutige Signatur zu erzeugen
9. DateTime: Als Zeitstempel für das Hochladen der Datei
10. Country: Land aus welches die Datei hochgeladen wurde
11. RegionName: Region (Bundesland) aus welches die Datei hochgeladen wurde
12. City: Stadt aus welches die Datei hochgeladen wurde
13. Lat: Breitengrad welcher mit den bekannten IP-Adresse assoziiert wird
14. Isp: Internetanbierter des Benutzers der die Datei hochgeladen hat
15. Lon: Längengrad welcher mit den bekannten IP-Adresse assoziiert wird
16. Isp: Der Internetanbieter welcher der IP zugeordnet ist

Für eine hoch geladene Datei werden diese 16 Informationen gespeichert und verwaltet.

Der UploadController, welche die API's implementiert besitzt eine Reihe von Methoden welche die API's die einem Benutzer zur Verfügung stehen implementieren.

Für das Set A, die ungeschützte Datenmenge sowie für das Set B, die ungeschützte Datenmenge gibt es API Punkt welcher das Hochladen in die entsprechende Datenmenge ermöglicht.

```
1         // POST api/upload
2         [HttpPost]
3         [Route ("api/uploadA")]
4         public RedirectToPageResult PostA()
5         {
6             ProcessPost (Request, "A");
7             return RedirectToPage ("/FileEntry");
8         }
9
10        // POST api/upload
11        [HttpPost]
12        [Route ("api/uploadB")]
13        public RedirectToPageResult PostB()
14        {
15            ProcessPost (Request, "B");
16            return RedirectToPage ("/FileEntry");
17        }
```

In Line zwei bis Line acht wird eine API Punkt implementiert.

```
1         [HttpPost]
```

Ist das Keyword um eine Methode als einen API Punkt zu bereit zu stellen. Das HTTPPost Keyword definiert dabei einen Post API Punkt welcher Daten, im zuge eines Request entgegen nimmt und diese in der Regal verarbeitet.

```
1         public RedirectToPageResult PostA()
2         {
3             ProcessPost (Request, "A");
4             return RedirectToPage ("/FileEntry");
5         }
```

Die Methode definiert als Rückgabewert ein RedirectToPageResult, welcher den Benutzer der diese API-benutzt als Antwort wieder zurück auf eine vordefinierte Page leitet. Da die Methode mit den HttpPost-Attribut getagt ist, ist im Methoden-Rumpf ein Request-Object verfügbar welches den HTTP-Request welcher gegen den API-Punkt gestellt wurde darstellt. Das Request-Object hält alle Informationen wie die Header, die hochgeladene Datei und die IP-Adresse des Benutzers und dient dazu die Meta-Daten des Uploads zu erfassen. Das Request-Object wird zusammen mit einen String Identifier für das Set, welches dem API-Punkt zugeordert wird, an eine Methode "ProcessPost"übergeben, welche den Request verarbeitet, indem die Meta-Daten gesammelt, die Datei, welche hochgeladen wurde, abspeichert und die Meta-Daten sowie den Pfad zu der hochgeladenen Datei in einer Datenbank geschrieben wird.

3.2 Darstellung: IP Tree Map und IP Google Map

Die Darstellung der gesammelten Daten über IP-Adressen Gruppierte Mengen.

Erzeugen des Gruppierungen aus der gegebenen Datenmenge. Nutzung des d3.js zur Darstellung der Tree Map. Dabei wird jeder IP-Gruppierung eine andere Farbe zugeordnet. Die Farbliche Visuelle Darstellung macht zusammengehörige Dateien nach IP-Adresse direkt sichtbar. Mit Hilfe der IP-Adressen und eines Geolookup kann ein ungefährer Standpunkt (Lat/Lon) der IP-Adresse gewonnen werden. Anhand dieses Standpunkts kann auf der Google Map der Standpunkt von wo eine Datei hochgeladen wurde aufgezeigt werden.

Bei Verwendung von Methoden zur Anonymisieren der IP-Adresse wie z.B. das verwenden eines Proxys oder der Verwendung des TOR-Netzwerks, werden die IP-Adressen-Gruppierungen verzerrt, durch verzerrte Gruppen können Dateien eines Benutzer nicht mehr zu 100% auf diesem Benutzer gemappt werden.

1. Proxy -> IP-Adresse wird maskiert
    a) Falls der Proxy wird nur durch einen Benutzer benutzt -> keine Gruppenverzerrung
    b) Proxy wird von mehreren Benutzern benutzt -> Gruppenverzerrung
    c) Benutzer wechselt Proxy mehrfach -> Gruppierungen werden in der Gesamtheit verzerrt -> pro Benutzer mehre Gruppierungen
2. IP-Gruppierung über die Endknoten des Tor-Netzwerks
    a) Benutzer die den gleichen Endknoten benutzen werden gruppiert -> Gruppenverzerrung
    b) Durch automatischen wechsle der Endknoten -> Pro Benutzer zwangsläufig mehrere Gruppen -> Gruppenverzerrung

Die Darstellung von IP bezogenen Meta-Daten wird an einem einfachen Beispiel vorgeführt. Gegeben sind Benutzer Alice und Bob. Alice und Bob befinden sich beide in Hamburg. Sie laden jeweils drei Dateien hoch, einmal mit und einmal ohne die Verwendung eines Proxys zum Maskieren ihrer IP-Adresse. Der verwendete Proxy ist bei Alice und Bob der selbe. Alice lädt die Dateien test0 bis test2 und Bob die Dateien test3-test5 hoch. Alice lädt ihre drei Dateien in einem Post hoch und Bob lädt die drei Dateien jeweils einzeln mit einem Abstand von ein paar Sekunden hoch.

Abbildung 3.1: Visualisierung von Dateien mittels einer TreeMap , welche nach der IP-Adresse gruppiert sind

In Abbildung 3.1 werden die Daten von Alice und Bob dargestellt welche ohne die Verwendung eines proxys hochgeladen worden sind. Die Darstellung erfolgt durch eine sog. TreeMap, welche eine Gegebene Datenstruktur, welche als Baum dargestellt werden kann, visualisiert. Die Baum Struktur besteht aus einer Wurzel-Knoten, sog. Root-Node und den davon abgehenden Child-Nodes, Nodes in der Baumstruktur welche keine weiteren Child-Nodes bestitzen werden Leaf-Nodes genannt. Die Root-Node wird künstliche erzeugt und als Überschrift für die Visualisierung verwendet und Zeigt den Key über dem die Baumstruktur erzeugt wurde, in diesem Fall die ÏpAddress". Die restlichen Nodes sind aus dem gegebenen Daten erzeugt worden, wobei der Key: ipAddress, die Ausschlag gebende Property ist über welcher die Baumstruktur erzeugt wird. Die Erzeugte Baumstrukur hat somit 3 Ebenen. Auf der 1. Ebene die Root-Node, welche zum Viusaliseren der Key Property benutzt wird. Auf der 2. Ebene die Nodes, welche die IP-Adressen darstellen welche im Datensatz vorhanden sind. Und auf der 3. Ebene die Leaf-Nodes, welche die Dateien selbst darstellen. In Abbildung 3.1 sind die IP-Adressen 95.91.226.214 von Alice und 95.91.225.215 Bob erkennbar. Zu jeder IP-Adresse können jeweils die drei Dateien der Benutzer zugeordnet werden. Die Dateien test0 bis test2 sind der IP-Adresse 95.91.226.214 zugeordnet und test3 bis test5 sind der IP-Adresse 95.91.225.215 zugeordnet. Damit ist das gegebene Beispiel richtig abgebildet und die echten Relationen zwischen Dateien und IP-Adressen sind richtig modelliert und nachvollziehbar.

Abbildung 3.2: Visualisierung von Dateien mittels einer Karte, welche die Geo-Position einer IP-Adresse anzeigt

Die Abbildung 3.2 zeigt uns die Geo-Location der IP-Adressen von Alice und Bob und zeigt deren ungefähre Position in Hamburg, wobei die Position nur einen ungefähren Standpunkt darstellt und die eigentliche Position bis zu einem Kilometer abweichen kann.

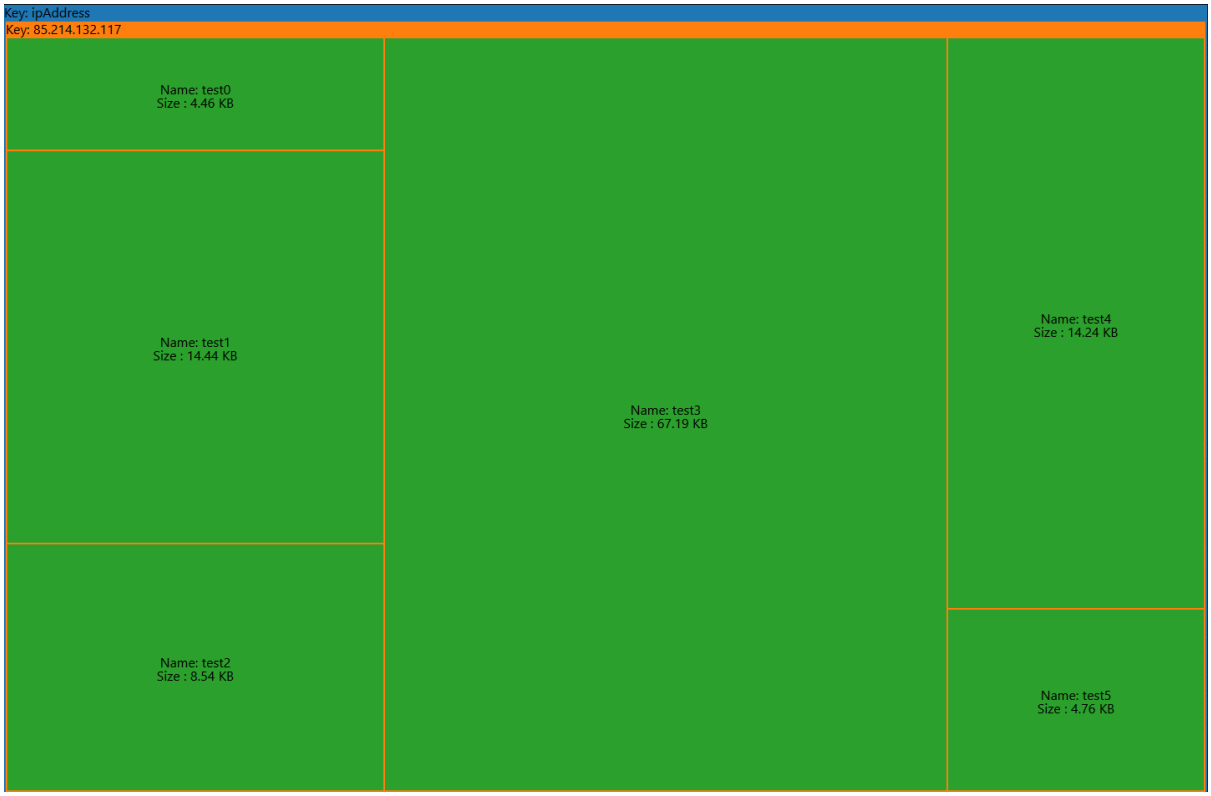

Abbildung 3.3: Visualisierung von Dateien mittels einer TreeMap , welche nach der IP-Adresse gruppiert sind

In Abbildung 3.3 sind die Daten Visualisiert, welche mittels eines Proxy hochgeladen wurden. Alle Dateien wurden über die IP-Adresse des Proxys 85.214.132.117 gruppiert und somit eine Anonymitätsmenge erzeugt. Die Daten von Alice und Bob sind nun in einer Anonymitätsmenge zusammengefasst und lassen sich anhand der IP-Adresse nicht mehr eindeutig den beiden Benutzern zuordnen.

Abbildung 3.4: Visualisierung von Dateien mittels einer Karte, welche die Geo-Position einer IP-Adresse anzeigt

Abbildung 3.4 zeigt und nun die Geo-Location des Proxys welches in Berlin steht. Damit sind auch die eigentlichen Geo-Locations von Alice und Bob durch die Verwendung des Proxys maskiert worden.

Anstelle eines Proxys könnten Alice und Bob das Tor-Netzwerk benutzen um ihre IP-Adressen zu maskieren. Alice und Bob laden drei Dateien, einmal ungeschützt und einmal geschütztm durch das Tor-Netzwerk hoch. Dabei nehmen wir an das alle Datei-Uploads einzeln passieren und jedem Dabei-Upload eine andere Route durch das Tor-Netzwerk zugewiesen wird, sodass jeder Upload einen anderen Exit-Server des Tor-Netzwerkes durchläuft.

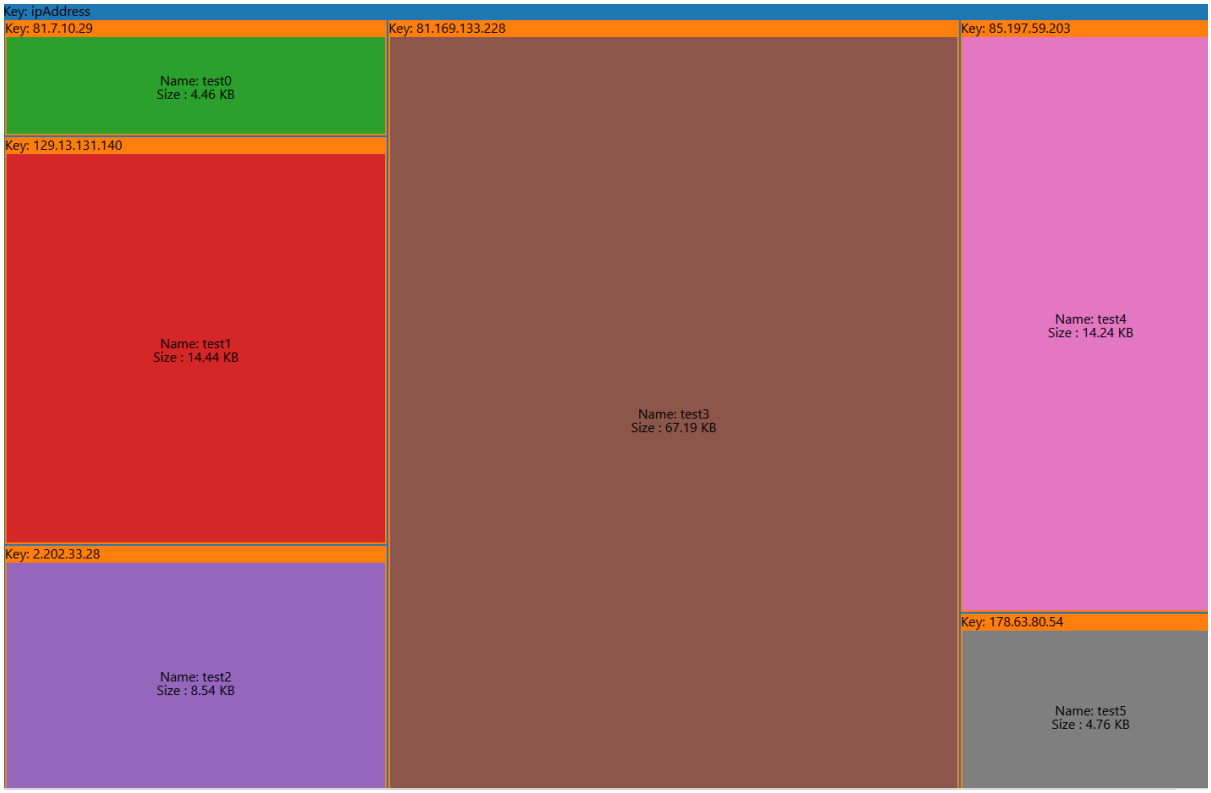

Abbildung 3.5: Visualisierung von Dateien mittels einer TreeMap , welche nach der IP-Adresse gruppiert sind

In Abbildung 3.5 sehen wir nun die IP-Adressen der verschiedenen Exit-Server des Tor-Netzwerks. Da jedem Datei-Upload ein neues Exit-Server zugewiesen wurde, können wir über die IP-Adresse die Dateien nicht mehr zuordnen.

Abbildung 3.6: Visualisierung von Dateien mittels einer Karte, welche die Geo-Position einer IP-Adresse anzeigt

Bei Betrachtung der Abbildung 3.6 ist auch erkennbar, das die verschiedenen Positionen der Exit-Server des Tor-Netzwerks variieren und es keinen Anhaltspunkt auf die eigentliche Postion von Alice und Bob gibt.

3.3 Darstellung: Headerfingerprinting

Der Headerfingerprint welcher beim hochladen der Datei erzeugt wird, wird nun verwendet um die Dateien jeweils

3.4 Darstelung: Time Line

4 Schluss

4.1 Zusammenfassung der Ergebnisse

4.2 kritische Bewertung des Ergebnisse

4.3 neue Problemstellungen, Möglichkeiten zur Weiterführung der Arbeit