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 13. Juni 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

1.1 Problemstellung

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.

1. Dokumentenspeicherdienste biten vorteile und Nachteile
2. möglichen Datenschutzmissbrauch durch Anbieter aufzeigen
3. Mögliche Verfahren/Methoden zum Datenschutz des Klienten aufzeigen
4. Implementation des Dokumentenspeicherdienstes als negativ Beispiel.
5. Nutzung des Dokumentenspeichers zur Gegenüberstellung von Datenschutz freundlichen und Datenschutz unfreundlichen Methoden zum Anonymisieren von Daten
6. Gewinn an Visuellen klar erkennbaren Unterschieden für z.B. nicht Informatiker zum besseren Verständnis des Problems

1.2 Problembearbeitung

1. Implementation des Dokumentenspeichers
    a) Vorstellung des Dotnet Core Framework
    b) Implementierung der Api des Dokumentenspeichers
        i. Api für die Verwendung des Dienstes
        ii. Api für die Ausgabe der Relevanten Daten
    c) Vorstellung des D3.js Framework zur Visualisierung
    d) Vorstellung der verschiedenen Komponenten zur Visualisierung
2. Schematische Verwendung des Dokumentenspeichers für die Problembearbeitung

2 Grundlagen

2.1 Terminologie

1. Header
2. Geolookup
3. Headerfingerprint

2.2 Implementation des Dokumentenspeichers

Die Implementation des Dokumentenspeichers ist mit dem ASP.NET Core Framework von Microsoft umgesetzt worden. Das ASP.NET Core Framework basiert prinzipiell auf der Model-View-Controller Architektur und lässt sich auf den gängigsten Betriebssystemen mit Hilfe von Thirdparty Applikationen oder selbst in einem eigenen Prozess Hosten.

Der Dokumentenspeicher besteht aus einer Controller-Klasse welche die API's implementiert, einer Datenbankkontext Klasse welche den Datenbankzugriff steuert und den dazugehörigen Modell-Klassen welche das Datenbank-Scheme implementieren.

Für das erstellen der Datenbank und dem Datenbankschema wurde eine Code-First-approach gewählt, wobei das Entity Framework Core (EF Core) verwendet wird um die Datenbankmodelle zu verwalten. Dabei wird eine Modell-Klasse erzeugt welche dem gewünschten Datenbankschema entspricht.

```
1  using Microsoft.AspNetCore.Http;
2  using System;
3  using System.Collections.Generic;
4  using System.ComponentModel.DataAnnotations;
5  using System.Linq;
6  using System.Net;
7  using System.Threading.Tasks;
8
9  namespace ba_Vofpffs.Models
10 {
11     public class FileEntryItemA
12     {
13         public FileEntryItemA()
14         {
15
16         }
17
18         public FileEntryItemA(string hash, byte[] file, int size,
               string ipAddress, string headers, string
               headerFingerprint, DateTime dateTime,
19              string country, string regionName, string city, string
                    lat, string lon, string isp)
```

```
20          {
21              this.Filename = hash;
22              this.File = file;
23              this.Size = size;
24              this.IPAddress = ipAddress;
25              this.Headers = headers;
26              this.DateTime = dateTime;
27              this.HeaderFingerprint = headerFingerprint;
28              this.Country = country;
29              this.RegionName = regionName;
30              this.City = city;
31              this.Lat = float.Parse (lat);
32              this.Lon = float.Parse (lon);
33              this.Isp = isp;
34          }
35
36          public int ID { get; set; }
37          public string Filename { get; set; }
38          public byte[] File { get; set; }
39          public int Size { get; set; }
40          public string IPAddress { get; set; }
41          public string Headers{ get; set; }
42          public string HeaderFingerprint { get; set; }
43          public DateTime DateTime { get; set; }
44          public string Country { get; set; }
45          public string RegionName { get; set; }
46          public string City { get; set; }
47          public float Lat { get; set; }
48          public float Lon { get; set; }
49          public string Isp { get; set; }
50      }
51  }
```

Listing 2.1: Modell-Klasse

Die Propertys der FileEntryA-Klasse werden hier definiert und die entsprechenden Konstruktoren implementiert.

Dazu muss eine Kontext-Klasse erzeugt werden welche anhand dieses Modells eine Datenbank-Tabelle erzeugt.

```
1   using Microsoft.EntityFrameworkCore;
2   using System;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Threading.Tasks;
6
7   namespace ba_Vofpffs.Models
8   {
9       public class FileEntryContext : DbContext
10      {
11          public FileEntryContext(DbContextOptions<FileEntryContext>
                options) : base (options) { }
12
13          public DbSet<FileEntryItemA> FileEntryItemsA { get; set; }
14          public DbSet<FileEntryItemB> FileEntryItemsB { get; set; }
15
16          protected override void OnModelCreating(ModelBuilder
                modelBuilder)
```

```
[ID]                 INT              IDENTITY (1, 1) NOT NULL,
    [DateTime]           DATETIME2 (7)    NOT NULL,
    [File]               VARBINARY (MAX)  NULL,
    [Filename]           NVARCHAR (MAX)   NULL,
    [Headers]            NVARCHAR (MAX)   NULL,
    [IPAddress]          NVARCHAR (MAX)   NULL,
    [Size]               INT              NOT NULL,
    [HeaderFingerprint] NVARCHAR (MAX)   NULL,
    [City]               NVARCHAR (MAX)   NULL,
    [Country]            NVARCHAR (MAX)   NULL,
    [Isp]                NVARCHAR (MAX)   NULL,
    [Lat]                REAL             NOT NULL,
    [Lon]                REAL             NOT NULL,
    [RegionName]         NVARCHAR (MAX)   NULL,
    CONSTRAINT [PK_FileEntryA] PRIMARY KEY CLUSTERED ([ID] ASC)
```

Listing 2.3: Datenbank Schema für das Set A

```
        {
            modelBuilder.Entity<FileEntryItemA> ().ToTable
                ("FileEntryA");
            modelBuilder.Entity<FileEntryItemB> ().ToTable
                ("FileEntryB");
        }
    }
}
```

Listing 2.2: Kontext-Klasse

Hier erbt er FileEntryContext den DbContext des Entitiy Frameworks und erzeugt zwei DbSets jeweils von Typ FileEntryA und FileEntryB und implementiert die Getter und Setter dieser Datenbanken und setzt die ModelBuilder Entitiys zum erzeugen der Datenbanken.

Es wurden zwei API-Schnittstellen für jede Datenbank implementiert. Zum einen eine Schnittstelle zum hochladen von Dateien und eine zum Abfragen der gesamten gespeicherten Datenmenge. Beim hochladen einer Datei werden die Header und Meta-Daten der Datei ausgewertet und in der Datenbank gespeichert. Die Auswertung beinhaltet das Erzeugen eines Headerfingerprint anhand der vorhandenen Header und eines Geolookup der IP-Adresse zur Ermittlung des Standortes von welchem die Datei hochgeladen wurde.

In der Datenbank werden Informationen gespeichert.

1. ID: Datenbank Index
2. DateTime: Als Zeitstempel für das Hochladen der Datei
3. File: die Datei selbst als Byte-Array
4. Filename: Der Dateiname
5. Headers: Ein String bestehend aus den Headern der Datei
6. IPAddress: Die IP-Adresse von der die Datei hochgeladen wurde
7. Size: Die Dateigröße in Byte

8. HeaderFingerprint: Ein zusammenschluss aus ausgewähten Headern um eine möglichst eindeutige Signatur zu erzeugen
9. City: Stadt aus welches die Datei hochgeladen wurde
10. Country: Land aus welches die Datei hochgeladen wurde
11. Isp: Internetanbierter des Benutzers der die Datei hochgeladen hat
12. Lat: Breitengrad welcher mit den bekannten IP-Adresse assoziiert wird
13. Lon: Längengrad welcher mit den bekannten IP-Adresse assoziiert wird
14. RegionName: Region (Bundesland) aus welches die Datei hochgeladen wurde

Die Schnittstelle zum abrufen der gesamten gespeicherten Daten erzeugt aus den vorhandenen Daten eine Json (Javascript Object Notation) Struktur und sendet diese als Antwort auf den HTTP-Request.

2.3 Set A / Set B

Die angesprochenen DbSets FileEntryItemsA und FileEntryItemsB verfügen jeweil über eine eigene Tabelle welche die zu visualisierenden Daten beinhaltet. 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.

Das Visualisierungswerkzeug besitzt zwei verschiedene Datenbanken welche Set A und Set B genannt werden. Jedes Datenbank Set verfügt über eine eigene API und funktioniert identisch. Die angestrebte Benutzung sieht den Vergleich von Set A und Set B mit den gleichen Datensatz vor wobei die nicht Anwendung von Datenschutz freundlichen Methoden zum Anonymisieren bei Set A und die Anwendung von Datenschutz freundlichen Methoden zum Anonymisieren bei Set B. Set A ist somit die Basis und zeigt auf was ein Dokumentenspeicherdienst an Nutzerdaten sammeln kann und Set B kann im direkten Vergleich zeigen wie Datenschutz freundlichen Methoden zum Anonymisieren diese Daten verfälschen.

3 Hauptteil

3.1 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

3.2 Darstellung: Headerfingerprinting

3.3 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