Technische Berichte in Digitaler Forensik

Herausgegeben vom Lehrstuhl für Informatik 1 der Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) in Kooperation mit dem Masterstudiengang Digitale Forensik (Hochschule Albstadt-Sigmaringen, FAU, Goethe-Universität Frankfurt am Main)

Analyse persistenter Spuren von Telegram Desktop 0.9.13.0 – 0.9.18.0 für Windows

Ralf Steinbrink
16.02.2016

Technischer Bericht Nr. 2

Zusammenfassung:
Telegram Desktop ist ein Multi-Plattform-Messenger zum Chat und Austausch von Medien. Er nutzt die Telegram-Infrastruktur (Cloud) und lädt erst bei Nutzer-Interaktion benötigte Inhaltsdaten nach. Dabei werden die Mediendaten regelmäßig lokal verschlüsselt gespeichert und vorgehalten. In diesem Dokument wird die grundlegende Struktur dieser Daten analysiert und dargestellt, so dass grundsätzlich eine Entschlüsselung der lokal gespeicherten Daten ermöglicht wird. Diese Arbeit entstand im Rahmen des Moduls Browser- und Anwendungsforensik des Studiengangs Digitale Forensik im Wintersemester 2015/2016 unter der Modulleitung von Felix Freiling, Holger Morgenstern und Michael Gruhn.

Inhalt

1. Aufgabe

1.1 Kurzbeschreibung

„Erstellen Sie im Rahmen der Analyse eines von Ihnen gewählten Programms eine Spurendokumentation, die für einen Forensiker hilfreich ist. Orientieren Sie sich dabei an Eintragungen im forensicswiki."

- Analyse einer Anwendung auf deren Spurenmenge
- (Detaillierte) Dokumentation der Spuren:
    - Welche Sachverhalte kann man wo finden?
    - Wie kann man die Spuren auslesen?

1.2 Abgrenzung

- Die Erstellung eines Werkzeugs, welches automatisch die Spuren ausliest und analysiert, ist nicht notwendig, wird jedoch positiv bewertet.

1.3 Konkretisierung

Durch den Autor wurde zur Analyse die Anwendung

Telegram Desktop für Windows[1]

in den Versionen 0.9.13.0, 0.9.15.0 bzw. 0.9.18.0 gewählt.

Bei diesem Programm handelt es sich um einen Cloud-gestützten Messenger, der laut Hersteller sowohl unverschlüsselte als auch verschlüsselte Chat-Kommunikation über mehrere Plattformen – sowohl über Mobiltelefone als auch über Standard-Betriebssysteme – ermöglicht.

Bei dem hiesigen Dokument handelt es sich um eine Zusammenfassung für informationstechnisch oder IuK-forensisch versierte Leser.

2. Untersuchungsmethoden

Zur Untersuchung wurde sowohl die Zustandsmethode verwendet als auch die Ereignismethode. Darüber hinaus wurde im Rahmen einer Live-Analyse der Netzwerkverkehr mittels Wireshark überprüft und Hauptspeichersicherungen durchgeführt.

2.1 Zustandsmethode

Als Basis der Untersuchungen wurden virtuelle Maschinen mit Windows XP und Windows 7 (jeweils 32Bit) verwendet. Jeweils vor einer Aktion wurde ein Snapshot gefertigt (Ursprung), die Aktion wurde durchgeführt, danach wurde ein weiterer Snapshot gefertigt (Endzustand). Beide Snapshots wurden miteinander

[1] https://tdesktop.com/win

verglichen. Darüber hinaus wurde in mehreren Versuchen das so genannte Grundrauschen ohne Durchführung einer Aktion erhoben und aus den Vergleichen von Usprung und Endzustand herausgefiltert. Die Vergleiche wurden unter Zuhilfename des Tools idifference2.py aus dem Projekt DFXML[2] durchgeführt.

2.2 Ereignismethode

Für die Durchführung der Ereignismethode wurden die Programme Procmon[3] aus der Sysinternals-Suite und Sandboxie[4] verwendet. Beide Werkzeuge wurden getrennt angewendet. Sie wurden zunächst gestartet, eine Konfiguration bzw. Filterung auf das zu untersuchende Programm konfiguriert und danach Telegram Desktop gestartet bzw. die entsprechende Funktion ausgeführt. Die erstellten Daten oder Protokolldateien wurden im Anschluss analysiert.

2.3 Live-Analyse (Wireshark)

Über den Netzwerkverkehr sind keine verwertbaren Spuren zu gewinnen, da der Netzwerkverkehr zwischen Telegram und den Telegram-Servern grundsätzlich SSL-verschlüsselt ist.

2.4 Hauptspeichersicherungen

Im Rahmen der labormäßigen Untersuchungen wurden flankierend mehrfach Hauptspeichersicherungen an den virtuellen Maschinen (mittels VBoxManage dumpvmcore) durchgeführt, um zu ermitteln, ob die von Telegram Desktop dargestellten Chat-Daten entschlüsselt im RAM verfügbar sind. Darauf liegt bzw. lag jedoch nicht der Schwerpunkt dieser Arbeit. Untersuchungen im Hinblick auf ein ggf. vorhandenes Programm-Passwort wurden nicht durchgeführt.

3. Ergebniszusammenfassung

3.1 Dateien und Ordner

Unmittelbar nach Standard-Installation von Telegram Desktop befinden sich in **%APPDATA%\Telegram Desktop** die Dateien

- **Telegram.exe**
- **Unins000.dat**
- **Unins000.exe**
- **Updater.exe**

wobei %APPDATA% für die Windows-Variable steht, die auf das Benutzer-spezifische Programmdaten-Verzeichnis verweist. Die Variable nimmt bspw. unter Windows XP den Wert

[2] http://www.forensicswiki.org/wiki/Category:Digital_Forensics_XML
[3] https://technet.microsoft.com/de-de/sysinternals/processmonitor.aspx
[4] http://www.sandboxie.com/index.php?DownloadSandboxie

C:\Dokumente und Einstellungen\Benutzername\Anwendungsdaten

und unter Windows 7 den Wert

C:\Users\Benutzername\AppData\Roaming

an.

Nach erstem Start von Telegram.exe befinden sich darüber hinaus dann in diesem Ordner die Dateien/Ordner

- Ordner **tdata**
- Ordner **tupdates** (nur dann, wenn eine neuere Version als die installierte verfügbar ist)
- **Log.txt**

Im Ordner tdata befinden sich

- Ordner **D877F783D5D3EF8C**
- Datei **D877F783D5D3EF8C1**
- **settings0**

Im Ordner D877F783D5D3EF8C befindet sich die Datei

- **map0**

Zur Laufzeit des Programms werden (nach Registrierung eines Benutzers) im angelegten Ordner D877F783D5D3EF8C in tdata beim Senden oder Empfangen von Nachrichten (Bilder, Audio, Video) weitere Dateien pro Nachricht nach dem Schema AABBCCDDEEFFGGHH[0|1] erzeugt. Beim Senden und Empfangen von reinen Text-Nachrichten werden nur manchmal Dateien nach obigem Schema angelegt oder modifziert.

Grundsätzlich ist aber feststellbar, dass die Dateien map[0|1] und D877F783D5D3EF8C[0|1] beim Senden oder Empfangen von Nachrichten inhaltlich verändert und neu geschrieben werden. Die Datei settings[0|1] wird inhaltlich verändert und neu geschrieben, sobald Änderungen von Programm-Optionen durchgeführt werden. Dabei variieren die letzten Zeichen der benannten Dateien immer zwischen 0 und 1.

Darüber hinaus konnte beobachtet werden, dass teilweise noch der Ordner

- **tdummy**

im Ordner tdata angelegt wurde (ohne Inhalt).

Eine genauere Eingrenzung der Spuren im Hinblick auf tatsächlich **charakteristische** Spuren bei einer **spezifizierten** Aktion ist jedoch nicht möglich.

3.2 Hauptspeicher zur Laufzeit von Telegram

Während der Laufzeit von Telegram konnten im Rahmen von Hauptspeichersicherungen stets Fragmente von Chats unverschlüsselt vorgefunden werden. Dabei handelte es sich regelmäßig um die Inhalte, die im aktiven Chatfenster dargestellt wurden. Eine tiefere Analyse dieser flüchtigen Spuren wurde nicht vorgenommen.

3.3 Windows-Registry

In der Windows-Registrierung befinden sich lediglich Eintragungen mit Installations- bzw. Deinstallations-Informationen im Key

HKCU/Software/Microsoft/Windows/CurrentVersion/Uninstall/{53F49750-6209-4FBF-9CA8-7A333C87D1ED}_is1

inkl. der Angabe des Installationszeitpunktes.

3.4 Unmittelbare Analyse der identifizierten Spuren

Bei den erzeugten Dateien handelt es sich jeweils um welche in einem proprietären Dateiformat, gekennzeichnet durch ein **„TDF$“** am Dateibeginn. Der weitere Dateiinhalt ist nicht im Klartext lesbar.

Der Dateiinhalt der durch Telegram angelegten Dateien ist ausweislich der hohen Entropie der Daten augenscheinlich verschlüsselt.

Bislang ist kein forensisches Werkzeug bekannt, mit dem die Dateien unabhängig von einer Installation von Telegram Desktop entschlüsselt bzw. interpretiert werden können.

In mehreren Versuchen konnte stets durch Sicherung des Ordners tdata mit allen darin befindlichen Dateien und Ordner und Hineinkopieren in eine frische Installation in bspw. eine virtuelle Maschine der Telegram Desktop mit gleichem Versionsstand lauffähig gestartet und zum Sichten und Auswerten der darin gespeicherten Daten genutzt werden, wenn

1) im Telegram Desktop nicht die Möglichkeit des Setzens eines Passcodes genutzt wurde bzw. dieser Passcode bekannt war,
2) zumindest zum Programmstart eine Internet-Anbindung vorhanden war und
3) die ursprüngliche Client-Installation nicht de-registriert wurde (!) – dies ist sowohl von dem eigenen Client als auch über einen anderen registrierten Client möglich.

Es sind gemäß der durchgeführten Versuche keine zur Entschlüsselung oder Interpretation notwendige Daten in der Windows-Registry gespeichert oder von der verwendeten Hardware-Umgebung abhängig.

Wurde im Telegram Desktop ein Passcode gesetzt oder der Client, dessen Daten vorliegen, de-registriert, sind die Daten nicht mehr ohne diesen Passcode bzw. gar nicht mehr über diese Methode entschlüssel- bzw. darstellbar.

3.5 Mittelbare Analyse der identifizierten Spuren durch Analyse des Quellcodes (Reverse Engineering)

Der Quellcode des Telegram Desktop ist frei verfügbar unter

https://github.com/telegramdesktop/tdesktop

Der für die Organisation der Ablage der Daten in Dateien maßgebliche Programmcode ist in den Dateien

https://github.com/telegramdesktop/tdesktop/blob/master/Telegram/SourceFiles/localstorage.cpp

und

https://github.com/telegramdesktop/tdesktop/blob/master/Telegram/SourceFiles/localstorage.h

enthalten. Weitere für die Analyse wichtige Variablen, Klassen und Strukturen sind in den Dateien

https://github.com/telegramdesktop/tdesktop/blob/master/Telegram/SourceFiles/config.h

und

https://github.com/telegramdesktop/tdesktop/blob/master/Telegram/SourceFiles/types.h

bzw.

https://github.com/telegramdesktop/tdesktop/blob/master/Telegram/SourceFiles/types.cpp

enthalten.

3.5.1 Allgemeiner Dateiaufbau

Grundsätzlich wird das Lesen der jeweiligen Dateien in Telegram Desktop über die Funktion

```
bool readFile(FileReadDescriptor &result, const QString &name, int options = UserPath | SafePath)
```

in localstorage.cpp realisiert, aus dieser lässt sich der ganz grundlegende Aufbau der Dateien wie folgt nachvollziehen:

- Jede Datei muss über einen Header verfügen, bestehend aus
  - Einem Magic als Char-Array, hier 4 Bytes TDF$
  - Der AppVersion als Int32 (4 Bytes little Endian), hier 9013, 9015 bzw. 9018
- Es folgen die Nutzdaten als Datenstream bis (Dateiende – 16 Bytes) eingelesen in einen QByteArray
- Die letzten 16 Bytes stellen eine Signatur der vorherigen Daten dar; diese Signatur errechnet sich mit der Funktion `HashMd5` aus Datei types.cpp aus

den Nutzdaten, der Datenmenge, der in der Datei vermerkten AppVersion und dem Magic.

Die Nutzdaten werden regelmäßig in weitere Datentypen (und Werte) gesplittet, dessen Bestandteile je nach Datentyp durch einen 4 Byte großen Längenwert angeführt werden (int32 Big Endian) und meist über verschlüsselte Inhaltsdaten verfügen.

3.5.2 Datei settings[0|1]

Maßgebliche Quellcode-Datei(en):

localstorage.cpp

Maßgebliche Funktion(en):

1. `void readSettings()`
2. `bool _readSetting(quint32 blockId, QDataStream &stream, int version)`

Ablauf/Struktur:

1) Herauslösen der Nutzdaten aus der Datei mittels in Ziffer 3.5.1 beschriebener Funktion
2) Aufsplitten in ein 32 Byte großes Salz (`salt`)[5] und verschlüsselte Inhaltsdaten (`settingsEncrypted`) – jeweils als QByteArray, d.h. auf Dateiebene angeführt durch je einen 4 Byte großen Längenwert (int32 Big Endian)
3) Erstellen eines Schlüssels *_settingsKey* mit leerem Passwort und dem eingelesenen Salz durch Aufruf der Funktion `createLocalKey(QByteArray(),&salt, &_settingsKey)`
4) Entschlüsseln der Inhaltsdaten mit diesem Schlüssel durch Aufruf der Funktion `decryptLocal(settings, settingsEncrypted, _settingsKey)` – die dekryptierten Daten befinden sich nun in dem QByteArray `settings`
5) Setzen der Programmeinstellungen durch schleifenweises Einlesen jeweils einer `blockId` mit folgendem Aufruf der Funktion `_readSetting(blockId, settings.stream, settingsData.version)`, in welcher mittels einer Switch/Case-Unterscheidung anhand der `blockId` die notwendigen Parameter folgend in Typ und Anzahl von der blockId abhängig (!) ausgelesen und entsprechend gesetzt werden.[6]

[5] Die Größe des Salz ergibt sich aus einer Definition in config.h (LocalEncryptSaltSize = 32 Bytes)
[6] Bei unbekannter blockId führt das zwangsläufig zu einem Abbruch des weiteren Auslesens dieser Datei

Besonderheiten/Inhalte:

- Hier werden Programmeinstellungen wie dbiAutoStart, dbiStartMinimized, dbiSendToMenu aber auch allgemeine Einstellungen wie MaxGroupCount oder MaxMegaGroupCount gespeichert.

3.5.3 Datei map[0|1]

Maßgebliche Quellcode-Datei(en):

localstorage.cpp

Maßgebliche Funktion(en):

1. `Local::ReadMapState _readMap(const QByteArray &pass)`

Ablauf/Struktur:

1) Herauslösen der Nutzdaten aus der Datei mittels in Ziffer 3.5.1 beschriebener Funktion
2) Aufsplitten der Nutzdaten in ein 32 Byte großes Salz (`salt`), einen 256 Byte langen verschlüsselten Schlüssel (`keyEncrypted`)[7] plus Signatur und verschlüsselte Inhaltsdaten (`mapEncrypted`) – jeweils als QByteArray, d.h. angeführt auf Dateiebene durch je einen 4 Byte großen Längenwert (int32 Big Endian)
3) Generierung eines aus dem eingelesenen Salz und einem ggf. gesetzten Passcode zuammenzusetzenden Schlüssels durch Aufruf der Funktion `createLocalKey(pass, &salt, &_passKey)`, um den verschlüsselten Schlüssel entschlüsseln zu können
4) (Versuch der) Entschlüsselung des verschlüsselten Hauptschlüssels durch Aufruf der Funktion `decryptLocal(keyData, keyEncrypted, _passKey)`
5) Nach Erfolg Setzens des lokalen Hauptschlüssels durch Aufruf der Funktion `_localKey.setKey(key)`
6) Entschlüsseln der Inhaltsdaten mittels Aufruf der Funktion `decryptLocal(map, mapEncrypted)`
7) Schleifenweises Einlesen jeweils eines `keyType` mit folgenden Parametern in Typ und Anzahl vom keyType abhängig (!)[8]

Besonderheiten/Inhalte:

- Der Ordnername D877F783D5D3EF8C, in dem sich die Datei map[0|1] befindet, wird generiert aus den höherwertigen 8 Bytes des MD5-Hashs von **data**, bei dem bei jedem Byte die Nibbles getauscht werden[9].

[7] Die Größe des Schlüssels ergibt sich aus einer Definition in config.h (LocalEncryptKeySize = 256 Bytes)
[8] Bei unbekanntem keyType führt das zwangsläufig zu einem Abbruch des weiteren Auslesens dieser Datei

- Die Datei beinhaltet einen 256 Byte großen kryptierten „Hauptschlüssel“, mit dem weitere Dateien entschlüsselt werden können.
- Die Datei beinhaltet offensichtlich die notwendigen Daten, um eine interne Datenbankstruktur über alle lokalen Nutzerdaten des Telegram Desktop zu erstellen. Insbesondere befinden sich darin Werte, mit denen die Klassen draftsMap, draftsPositionsMap, draftsNotReadMap, imagesMap, stickerImagesMap, audiosMap zur Laufzeit befüllt werden. Aber auch weitere Daten wie userSettingsKey, locationsKey, savedPeersKey, bei denen es sich **nicht** um Schlüssel sondern eher um Bezeichner von weiteren Daten(bank)dateien handeln dürfte, sind enthalten.
- Die Datei dürfte somit das „große Inhaltsverzeichnis und gleichzeitig Index“ aller im Ordner D877F783D5D3EF8C befindlichen Daten sein.

3.5.4 Datei D877F783D5D3EF8C[0|1]

Maßgebliche Quellcode-Datei(en):

localstorage.cpp

Maßgebliche Funktion(en):

1. `void _readMtpData()`

Ablauf/Struktur:

1) Herauslösen der (verschlüsselten) Nutzdaten aus der Datei mittels in Ziffer 3.5.1 beschriebener Funktion
2) Entschlüsseln der Nutzdaten erfolgt in der Funktion `readEncryptedFile(FileReadDescriptor &result, const QString &name, int options = UserPath | SafePath, const mtpAuthKey &key = _localKey)` durch Aufruf von `decryptLocal(data, encrypted, key)`, wobei für `key` der während des Einlesens von **map[0|1]** generierte/gesetzte `_localKey` genutzt wird.
3) Setzen von für die Kommunikation mit den Telegram-Servern notwendigen Einstellungen durch schleifenweises Einlesen jeweils einer `blockId` mit folgendem Aufruf der Funktion `_readSetting(blockId, settings.stream, settingsData.version)`, in welcher mittels einer Switch/Case-Unterscheidung anhand der `blockId` die notwendigen Parameter folgend in Typ und Anzahl von der blockId abhängig (!) ausgelesen und entsprechend gesetzt werden.[10]

[9] Nachvollziehbar über die Definitionen zu Anfang der benannten maßgeblichen Funktion sowie die Funktion `QString toFilePart(Filekey val)` in localstorage.cpp

[10] Bei unbekannter blockId führt das zwangsläufig zu einem Abbruch des weiteren Auslesens dieser Datei

Besonderheiten/Inhalte:

- Der Aufruf von `_readMtpData` erfolgt aus der in Ziff. 3.5.3 beschriebenen Funktion vor Beendigung der Funktion.
- Diese Datei ist mit dem „Hauptschlüssel“ verschlüsselt.
- In der Datei ist die UserId (UID) des Telegram-Benutzers gespeichert.
- Die Datei beinhaltet weitere Schlüssel für die Kommunikation mit der Telegram-Infrastruktur.

3.5.5 Restliche Dateien nach dem Schema AABBCCDDEEFFGGHH[0|1]

Maßgebliche Quellcode-Datei(en):

localstorage.cpp

Maßgebliche Funktion(en):

1.
```
bool readEncryptedFile(FileReadDescriptor &result, const FileKey &fkey, int options = UserPath | SafePath, const mtpAuthKey &key = _localKey) {
      return readEncryptedFile(result, toFilePart(fkey), options, key);
      }
```
2. `decryptLocal (data, encrypted, key)`

Ablauf/Struktur:

1) Herauslösen der (verschlüsselten) Nutzdaten aus der Datei mittels in Ziffer 3.5.1 beschriebener Funktion
2) Entschlüsseln der Nutzdaten mit `decryptLocal(data, encrypted, key)`, wobei für `key` der während des Einlesens von **map[0|1]** generierte/gesetzte `_localKey` genutzt wird.

Besonderheiten/Inhalte:

- Alle nicht explizit erwähnten Dateien können grundsätzlich Daten von beliebigem Format beinhalten.
- Um welchen Datentyp es sich handelt, ist aus den aus der map[0|1] ausgelesenen Indexen ersichtlich. Der Dateiname ist der so genannte FileKey, dessen Wert (Typ quint64) nach Umrechnung durch die Funktion `fromFilePart` in map[0|1] oder einer der darin deklarierten Datenbanken zu finden sein sollte.

4. Zusammenfassung

Das Programm Telegram Desktop für Windows wird per Standardeinstellung grundsätzlich in den Benutzer-spezifischen Ordner für Programmeinstellungen in den Ordner „Telegram Desktop“ installiert. Im Unterordner „tdata“ befinden sich

dann auch die durch das Programm angelegten Dateien mit – aus forensischer Sicht bzw. aus Ermittlersicht – interessanten Inhalten. In der Windows-Registrierung werden lediglich Informationen über die (De-) Installation des Programms gespeichert. Die von Telegram Desktop angelegten Dateien verfügen alle über einen einheitlichen Header TDF$, die restlichen Daten sind nicht im Klartext lesbar. Das Dateiformat ist proprietär. Die Inhaltsdaten sind verschlüsselt (AES). Eine Eingrenzung der persistenten Spuren im Hinblick auf tatsächlich **charakteristische** Spuren bei einer **spezifizierten** Aktion ist nicht möglich.

Der Quelltext von Telegram Desktop ist frei im Internet verfügbar. Anhand dessen ist durch Reverse Engineering der Aufbau der Strukturen der Dateien nachvollziehbar.

Die Datei settings[0|1] beinhaltet (codierte) Programmeinstellungen. Sie ist verschlüsselt durch ein leeres Passwort und ein in der Datei selbst vorhandenes Salz.

Die Datei map[0|1] beinhaltet „das große Inhaltsverzeichnis“, einen (codierten) Index, zu allen anderen angelegten Dateien. Enthalten ist wiederum ein Salz sowie ein mit ggf. vergebenem Passwort und dem in der Datei gespeicherten Salz verschlüsselter „Hauptschlüssel“. Der Hauptschlüssel ist entschlüsselbar, sofern kein Passwort vergeben war oder das Passwort bekannt ist.

Die Datei D877F783D5D3EF8C[0|1] beinhaltet Informationen zur Kommunikation mit der Infrastruktur von Telegram und die Telgram UserId (UID) des Nutzers. Diese Datei ist verschlüsselt mit dem Hauptschlüssel.

Alle übrigen Dateien im Ordner D877F783D5D3EF8C, deren Dateinamen nach dem Schema AABBCCDDEEFFGGHH[0|1] aufgebaut sind, enthalten entweder weitere Benutzerspezifische Programminformationen oder Benutzerdaten. Sie sind verschlüsselt mit dem Hauptschlüssel.

Bei jedem Programmstart und bei jeder Änderung von Programmeinstellungen wird die Datei settings[0|1] neu geschrieben. Genau so wird bei jedem Programmstart die Datei D877F783D5D3EF8C[0|1] neu geschrieben.

Die Datei map[0|1] wird immer dann neu geschrieben, wenn der Benutzer sich registriert (und damit den Telegram Desktop freischaltet), der Telegram Desktop neue Chat-Nachrichten zu verarbeiten hat, der Nutzer im Programm den lokalen Speicher aufräumt oder wenn er sich de-registriert.

Alle übrigen Dateien im Ordner D877F783D5D3EF8C, deren Dateinamen nach dem Schema AABBCCDDEEFFGGHH[0|1] aufgebaut sind, werden erzeugt, wenn Telegram Desktop insbesondere Medien neuer Chat-Nachrichten lokal abspeichert oder Benutzer-spezifische Einstellungen erzeugt oder geändert werden. Sie sind verschlüsselt mit dem Hauptschlüssel.

Mit einem entwickelten Proof of Concept „LocalstorageView“ zur Verifikation der Analyse-Ergebnisse der durch den Telegram Desktop verursachten persistenten Spuren war anhand der vorgelegten Analyse im aktuellen Entwicklungsstand eine Entschlüsselung und „Sichtbarmachung“ der Spuren im Rohformat möglich und damit auch der Nachweis über Besitz oder Kenntnis von den entsprechenden Medien zu führen. Eine weitere Auswertung der Spuren im Hinblick auf bspw. jeweilige Absender, Empfänger und weitere Daten bedarf einer weitergehenden und deutlich tieferen Analyse der Datenbankstruktur(en) sowie ggf. die Zuhilfename der Telegram-Infrastruktur.

Im Rahmen der Ergebnisverifikation konnte kein Nachweis für lokal gespeicherte Texte aus den Chats geführt werden, trotzdem das Generieren oder Modifizieren von verschlüsselten Dateien bei einer entsprechenden Aktion zunächst darauf hindeutete.

Dahingegen wurden in Chats übermittelte Medien (Audio, Video, Bilddateien) regelmäßig auch lokal (verschlüsselt) vorgehalten, wenn sie entweder vom Client aus an die Telegram-Infrastruktur übermittelt wurden oder mit dem Client ein entsprechnder Chat-Inhalt gesichtet wurde. Entsprechende Nachweise konnten erbracht werden.

5. Quellenverzeichnis

Internetquellen: (alle zuletzt aufgerufen am 16.02.2016)

Telegram Desktop
Telegram Desktop für Windows (Fußnote 1)
http://tdesktop.com/win

GitHub
dfxml (Fußnote 2)
https://github.com/simsong/dfxml

Projektseite Telegram Desktop (Ziffer 3.5)
https://github.com/telegramdesktop/tdesktop

Microsoft TechNet
Process Monitor (Fußnote 3)
http://technet.microsoft.com/de-de/sysinternals/bb896645.aspx

Sandboxie (Fußnote 4)
http://www.sandboxie.com