Bundesamt
für Sicherheit in der
Informationstechnik

Sicherheitsanalyse TrueCrypt

16. November 2015

Autoren

Die Studie *Sicherheitsanalyse TrueCrypt* wurde vom Fraunhofer-Institut für Sichere Informationstechnologie (SIT) im Auftrag des Bundesamts für Sicherheit in der Informationstechnik (BSI) erstellt. An der Erstellung waren beteiligt:

Mauro Baluda
Andreas Fuchs
Philipp Holzinger
Lotfi ben Othmane
Andreas Poller
Jürgen Repp
Johannes Späth
Jan Steffan
Stefan Triller
Eric Bodden

vom Fraunhofer-Institut für Sichere Informationstechnologie (SIT)
Rheinstraße 75
64285 Darmstadt

Die Autoren danken weiterhin Andreas Junestam und Nicolas Guigo vom Open Crypto Audit Project für klärende Worte zu ihrer eigenen Sicherheitsanalyse von TrueCrypt. Dank geht außerdem an Andreas Follner, Technische Universtät Darmstadt, für seine Unterstützung bei der Evaluation der Ausnutzbarkeit einiger im OCAP-Report genannten Pufferüberläufe.

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
Tel.: +49 22899 9582-0
E-Mail: bsi@bsi.bund.de
Internet: https://www.bsi.bund.de
© Bundesamt für Sicherheit in der Informationstechnik 2015

Inhaltsverzeichnis

1 Zusammenfassung 6
1.1 Zielsetzung . . . 6
1.2 Vorgehensweise . . . 6
1.3 Ergebnisse . . . 7

2 Untersuchung der Unterschiede von Version 7.0a zu 7.1a 9
2.1 Vorgehen . . . 9
2.2 Sicherheitsrelevante Unterschiede . . . 9
2.3 Weitere Unterschiede . . . 10
2.4 Zusammenfassung . . . 11

3 Bewertung des OCAP Phase 1 Prüfberichts 13
3.1 Kommentare zum OCAP . . . 13
3.1.1 Umfang des Berichts . . . 13
3.1.2 Kommentare zu den einzelnen Ergebnissen des OCAP-Berichts . . . 13
3.1.3 Detaillierte Ausführung der manuellen Analyse zu Resultat 3 . . . 15
3.1.4 Überprüfung der Ergebnisse mittels Coverity . . . 16
3.2 Zusammenfassung . . . 17

4 Überprüfung der Verschlüsselungsmechanismen 18
4.1 Verschlüsselungsalgorithmen . . . 18
4.2 Schlüsselableitung . . . 22
4.3 Zufallszahlengenerierung . . . 24
4.4 Zusammenfassung . . . 25

5 Untersuchung mittels automatisierter Code-Analyse 26
5.1 Übersicht über die Analyse-Ergebnisse . . . 29
5.2 Details zu den Fehlerberichten und Empfehlungen . . . 30
5.2.1 Clang Static Analyzer . . . 30
5.2.2 Coverity . . . 31
5.2.3 Cppcheck . . . 36
5.3 Zusammenfassung . . . 36

6 Bewertung der Code-Qualität und Dokumentation 37
6.1 Bewertung der Code-Qualität . . . 37
6.1.1 Programmierrichtlinien und Best-Practices . . . 37
6.1.2 Komplexität des Quelltextes . . . 39
6.1.3 Codeduplikate . . . 39
6.1.4 Fazit . . . 40
6.2 Bewertung der Dokumentation . . . 40
6.3 Zusammenfassung . . . 42

7 Konzeptionelle Bewertung der Architektur 43

7.1 Einleitung 43
7.2 Kontext und Anwendungsfälle 43
7.3 Schutzziele und Anforderungen 45
7.4 Angriffsstrategien 46
7.4.1 Vorbetrachtungen 46
7.4.2 Überblick über Angriffe mit physischen Zugriff 47
7.4.3 Angriffsstrategien im Detail 49
7.4.4 Zwischenfazit 53
7.4.5 Gegenüberstellung mit Sicherheitsmodell von TrueCrypt 54
7.5 Bewertung der Architektur 55
7.5.1 Die Komponenten im Überblick 55
7.5.2 Die Kernfunktionalität von TrueCrypt 57
7.5.3 Wartbarkeit und Testbarkeit 62
7.5.4 Empfehlungen zur Verbesserung 63
7.6 Zusammenfassung 65

8 Identifikation entbehrlicher Code-Teile 66
8.1 Zielsetzung 66
8.2 Vorgehensweise 66
8.3 Ergebnis 66

9 Bewertung des OCAP Phase 2 Prüfberichts 68
9.1 Kommentare zum OCAP-2 68
9.2 Kommentare zu den Ergebnissen 68
9.3 Weitergehende Ergebnisse 71
9.4 Zusammenfassung 72

A Funktionen mit zyklomatischer Komplexität über 15 73

B Codeduplikate (Exzerpt) 76

C Detail of the static analysis tools' warnings 78

Tabellenverzeichnis

4.1 Verfügbare Algorithmen 18
4.2 Format Testdaten symetrische Verschlüsselung 22
4.3 Schlüsselableitungsfunktionen TrueCrypt / OpenSSL 22
4.4 Format Testdaten Schlüsselableitung 22

5.1 Risikoeinstufung der Clang-Ergebnisse 30
5.2 Risikoeinstufung der Coverity-Ergebnisse für Linux 31
5.3 Risikoeinstufung der Coverity-Ergebnisse für Windows 33

7.1 Arten von Angriffsszenarien und Beispiele 49

1 Zusammenfassung

1.1 Zielsetzung

TrueCrypt ist ein bis vor Kurzem frei erhältliches Verschlüsselungsprogramm. Ende Mai 2014 wurde die Entwicklung und der Vertrieb von TrueCrypt in der Version 7.1a ohne Vorankündigung eingestellt. Über die Gründe hierfür ist wenig bekannt. Auf der offiziellen TrueCrypt-Webseite [10] findet man ganz oben in roter Schrift: »WARNING: Using TrueCrypt is not secure as it may contain unfixed security issues«. Da Teile von TrueCrypt auch im zugelassenen Produkt Trusted Disk enthalten sind, könnte eine Sicherheitsschwäche von TrueCrypt auch Trusted Disk betreffen.

Aus diesem Grund hat das Bundesamt für Sicherheit in der Informationstechnik (BSI) das Fraunhofer-Institut für sichere Informationstechnologie (SIT) mit der Durchführung einer Sicherheitsanalyse von TrueCrypt beauftragt. Dieser Bericht fasst die Ergebnisse der Sicherheitsanalyse zusammen. Ziel war es dabei neben der Aufdeckung möglicher Schwachstellen auch Verbesserungsmöglichkeiten für zukünftige Entwicklungen aufzuzeigen.

1.2 Vorgehensweise

Sicherheitsprobleme können vielfältige Ursachen haben, etwa falsche Designentscheidungen, Programmierfehler, aber auch eine missverständliche Dokumentation. Das Projekt wurde daher in mehrere Arbeitspakete untergliedert, die die Untersuchung von TrueCrypt unter den unterschiedlichen Gesichtspunkten beinhalten. Die Ergebnisse der einzelnen Arbeitspakete sind in diesem Bericht jeweils in einem eigenen Kapitel zusammengefasst.

Untersuchung der Unterschiede von Version 7.0a zu 7.1a Das vom BSI zugelassene Produkt Trusted Disk basiert auf der vorherigen Version 7.0a von TrueCrypt. In diesem Arbeitspaket wurden die Veränderungen der aktuellen Version 7.1a gegenüber 7.0a erfasst und im Hinblick auf ihre Sicherheitsrelevanz bewertet. So kann eingeschätzt werden, ob Schwachstellen der aktuellen Version potenziell auch Trusted Disk betreffen.

Bewertung des OCAP Phase 1 Prüfberichts Vor der Einstellung des TrueCrypt-Projekts wurde eine durch Crowd-Funding finanzierte Sicherheitsanalyse begonnen. Die Ergebnisse der ersten Phase dieser Analyse wurden in einem Bericht veröffentlicht [14]. Inhalt dieses Arbeitspakets war es, diese Ergebnisse sowie die verwendete Vorgehensweise zu bewerten.

Überprüfung der Verschlüsselungsmechanismen Kryptographische Verfahren zur Schlüsselherleitung und zur Verschlüsselung von Daten bilden die Kernfunktion von TrueCrypt. Schwachstellen in diesen Funktionen haben ein besonders hohes Potenzial, die Schutzziele von TrueCrypt zu gefährden. Daher wurde in einem eigenen Arbeitspaket untersucht, ob die kryptographischen Funktionen in TrueCrypt korrekt umgesetzt werden.

Untersuchung mittels automatisierter Code-Analyse Der vollständige Quelltext von TrueCrypt wurde mit verschiedenen aktuellen Werkzeugen auf Schwachstellen untersucht. Alle Fundstellen wurden anschließend manuell oder werkzeuggestützt untersucht und bewertet.

Bewertung der Code-Qualität und Dokumentation Viele Sicherheitslücken beruhen auf Fehlern oder falschen Annahmen durch Entwickler oder Anwender. Gut verständliche,

wartbare Quelltexte und eine vollständige, gut strukturierte und zielgruppengerechte Dokumentation sind daher sehr wichtig zur Vermeidung von Sicherheitsproblemen. Die Qualität von Quelltexten und Dokumentation liefert außerdem Indizien dafür, welchen Stellenwert nicht-funktionale Aspekte wie Sicherheit bei der Entwicklung hatten.

Konzeptionelle Bewertung der Architektur In diesem Arbeitspaket wurden Angriffswege gegen die Schutzziele von TrueCrypt ermittelt und geprüft, ob angemessene Design- und Architekturentscheidungen zur Abwehr dieser Bedrohungen getroffen wurden.

Identifikation entbehrlicher Code-Teile Eine Strategie die Wahrscheinlichkeit von Schwachstellen zu verringern ist es, unnötige Angriffsflächen zu vermeiden und generell die Komplexität des Systems zu verringern. Aufbauend auf der Architekturanalyse wurde daher geprüft, ob TrueCrypt Code-Teile enthält, die entfernt werden könnten, ohne auf wesentliche Funktonen verzichten zu müssen.

Bewertung des OCAP Phase 2 Prüfberichts Vor der Einstellung des TrueCrypt-Projekts wurde eine durch Crowd-Funding finanzierte Sicherheitsanalyse begonnen. Die Ergebnisse der zweiten Phase dieser Analyse wurden im März 2015 in einem Bericht veröffentlicht [2]. Inhalt dieses Arbeitspakets war es, diese Ergebnisse zu bewerten sowie Handlungsempfehlungen abzugeben.

1.3 Ergebnisse

Der Gegenstand der Untersuchungen war TrueCrypt in seiner letzten vollständigen Version 7.1a. Da zwischen den Versionen 7.0a und 7.1a nur geringfügige, nicht sicherheitsrelevante Änderungen festgestellt wurden, können die Ergebnisse jedoch auch auf die ältere Version übertragen werden, die u. a. als Grundlage für das Produkt »Trusted Disk« gedient hat.

Insgesamt wurden bei der Untersuchung keine Hinweise darauf gefunden, dass die Implementierung von TrueCrypt die zugesicherten Verschlüsselungseigenschaften nicht erfüllt. Insbesondere ergab ein Vergleich der kryptografischen Funktionen mit Referenzimplementierungen bzw. Testvektoren keinerlei Abweichungen.

Teilweise zeitgleich mit dieser Studie wurden durch Crowdfunding finanzierte Reviews des Quelltextes und der kryptographischen Funktionen durchgeführt (Open Crypto Audit Project, OCAP). Die durch OCAP publizierten 15 Schwachstellen wurden manuell verifiziert. Eine der Schwachstellen stellt eine hohe praktische Bedrohung dar.

Die Nutzung von Kryptographie in TrueCrypt ist nicht optimal. Die AES-Implementierung ist nicht timing-resistent, Keyfiles werden in nicht kryptographisch sicherer Weise genutzt und die Volume Header werden nicht korrekt integritätsgeschützt. Es gibt viele Mehrfachimplementierungen (teilweise zur Nutzung von Hardwarebeschleunigung) und aussortierte Algorithmen sind immer noch in deaktivierter Form im Source-Code vorhanden. Insbesondere wurde im Rahmen dieses Projekts eine verbesserungswürdige Implementierung für den Zufallszahlengenerator für Linux entdeckt und OCAP hat eine potentiell gefährliche Fehlimplementierung für den Zufallszahlengenerator unter Windows aufgedeckt.

Der Quelltext von TrueCrypt wurde mit drei verschiedenen Werkzeugen zur statischen Codeanalyse auf mögliche Fehler und Schwachstellen untersucht. Durch eine sorgfältige manuelle Überprüfung der werkzeuggestützten Ergebnisse konnten alle potentiell sicherheitskritisch gemeldete Verdachtsfälle wie mögliche Bereichsüberschreitungen als falsch-positiv identifiziert werden.

Qualitätsmängel gibt es vor allem in Bezug auf die Wartbarkeit und Dokumentation der Quelltexte. Sowohl die statischen Code-Analysen, verschiedene automatisch berechnete Bewertungsmetriken, als auch die manuelle Durchsicht der Quelltexte ergab zahlreiche Anhaltspunkte für Mängel und Abweichungen von der allgemein anerkannten Praxis. Zudem fehlt es an geeigneter

Dokumentation für Entwickler. Quelltexte sind nur sporadisch kommentiert. Eine Architekturbeschreibung ist nicht vorhanden. Dies ist nicht unmittelbar sicherheitsrelevant. Der aus den Qualitätsmängeln resultierende überdurchschnittliche Wartungsaufwand und die mangelnde Dokumentation erschweren jedoch eine mögliche Fortführung des Projekts durch Dritte.

Das Benutzerhandbuch von TrueCrypt ist umfangreich, aber schlecht strukturiert. Viele Detailinformationen sind schwer auffindbar und nur mit entsprechenden technischen Vorkenntnissen verständlich. Dies ist problematisch, da die Sicherheitseigenschaften von TrueCrypt teilweise vom Benutzerverhalten abhängen (z. B. Vermeidung des Ruhezustands, oder beim Einsatz von Hidden Volumes).

Aus Sicherheitssicht bedeutsam ist, dass TrueCrypt als reine Software-Lösung prinzipbedingt nicht vor allen relevanten Bedrohungen schützen kann. Ein wirksamer Schutz ist nur dann gegeben, wenn ein verschlüsselter Datenträger im ausgeschalteten Zustand verloren oder entwendet wird. Vor aktiven Angriffsszenarien wie der Installation eines Key-Loggers oder Malware bietet TrueCrypt keinen Schutz. Hierfür wären hardwarebasierte Sicherheitsmaßnahmen erforderlich, etwa mittels TPM oder Smartcard.

2 Untersuchung der Unterschiede von Version 7.0a zu 7.1a

Dieses Kapitel beschreibt das Vorgehen zum Vergleichen der beiden TrueCrypt-Versionen auf Basis deren Quelltexte. Es beschreibt das Vorgehen, sowie die gefundenen Unterschiede und hebt solche hervor, die als *sicherheitsrelevant* eingestuft werden können.

2.1 Vorgehen

Als Grundlage zum Finden der Unterschiede zwischen den beiden TrueCrypt-Versionen dient der Quelltext beider Versionen. Version 7.0a wurde von der Webseite `https://github.com/DrWhax/truecrypt-archive/blob/master/TrueCrypt%207.0a%20Source.zip` heruntergeladen und Version 7.1a von `https://github.com/AuditProject/truecrypt-verified-mirror?files=1`. Version 7.1a wurde zunächst nicht aus demselben Archiv genommen wie 7.0a, weil das Open Crypto Audit Project die unter dem Link genannte Version verwendet hat. Ein späterer Vergleich der beiden 7.1a Versionen aus den verschiedenen Quellen zeigte jedoch, dass diese identisch sind.

Verglichen wurden die Versionen 7.0a und 7.1a mit Hilfe des Programms *KDE Kompare* in Version 4.1.3. Kompare verarbeitet entweder einzelne Textdateien oder ganze Verzeichnisbäume. Letzteres wurde für den TrueCrypt-Vergleich verwendet. Kompare vergleicht dabei jede einzelne Datei im ersten Verzeichnisbaum mit der im zweiten und zeigt dessen Unterschiede an. Kam eine neue Datei im Zweiten Verzeichnisbaum hinzu, so wird diese gegen eine leere Datei verglichen und ist folglich vollständig als verschieden gekennzeichnet. Angezeigt werden nur die Teile des Verzeichnisbaumes, die unterschiedliche Dateien beinhalten. Für den TrueCrypt-Vergleich wurden die zwei Verzeichnisbäume, die jeweils auf eine TrueCrypt-Version zeigen, in Kompare ausgewählt. Anschließend sind wir den Verzeichnisbaum, welcher nur die Unterschiede angezeigt hat, chronologisch durchgegangen und haben uns manuell die geänderten Quelltextzeilen angeschaut.

Zunächst wurde versucht die unterschiedlichen Zeilen in einen Kontext zu bringen, um bewerten zu können, ob die Änderungen eventuell Auswirkungen auf die Sicherheit von TrueCrypt haben könnten. Um den Kontext einer Änderung zu erschließen wurde der Name der Methode, in der die Änderungen erfolgte, herangezogen, sowie der Quelltext in ihr. Ergab sich daraus bereits, dass die Änderung höchstwahrscheinlich nicht sicherheitsrelevant ist, weil sie z.B. nur eine Verschiebung eines Buttons in der grafischen Benutzeroberfläche bewirkt, wurde die Änderung als solche notiert und nicht weiter gesucht. Gab es jedoch Hinweise wie z. B. Schlüsselwörter wie »password«, so wurde auch danach gesucht, von wo im Quelltext diese Methode aufgerufen wurde und welche Auswirkungen die Änderung damit haben könnte.

2.2 Sicherheitsrelevante Unterschiede

Änderungen die für die Sicherheit von TrueCrypt interessant sind, gibt es zwischen den Versionen 7.0a und 7.1a nur zwei:

1. Veraltete Verschlüsselungsalgorithmen werden auf der Kommandozeile nicht mehr als Option angeboten
2. Funktion zum Einrechnen des Inhaltes eines Verzeichnisses in das Benutzerpasswort ignoriert nun versteckte Dateien

Die erstere der beiden o. g. Änderungen befindet sich in der Datei `CommandLineInterface.cpp` im Verzeichnis `Main` in Zeile 259. Die Verschlüsselungsalgorithmen AES/Blowfish, AES/Blowfish/Serpent, Blowfish, Cast5, und TripleDES waren bereits in der Version 7.0a als veraltet im Quelltext markiert worden und in Version 7.1a werden sie nun auch nicht mehr auf der Kommandozeile akzeptiert, falls ein Benutzer versuchen sollte diese zu verwenden.

Auswirkungen auf die Sicherheit von TrueCrypt hat diese Änderung jedoch nur wenige, da erfahrene Benutzer ohnehin schon veraltete Algorithmen ignoriert haben, beziehungsweise TrueCrypt in der Version 7.0a solche Algorithmen bereits als veraltet markiert hatte, und nun in der Version 7.1a es dem Benutzer auch anzeigt.

Die zweite Änderung befindet sich u. a. in der Datei `Keyfiles.c` im Verzeichnis `Common` ab Zeile 333. Betroffen ist die Funktion `ApplyListToPassword`. Sie ignoriert nun versteckte Dateien. Hintergrund zu dieser Funktion ist die Möglichkeit den Verschlüsselungsschlüssel für Container nicht nur aus dem vom Benutzer gesetzten Passwort abzuleiten, sondern zusätzlich eine Liste von Dateien mit einfließen zu lassen. Dieses Vorgehen erhöht die Entropie des Schlüssels, da Benutzer häufig dazu neigen nur Zahlen und/oder Buchstaben als Passwort zu wählen.

Aus sicherheitstechnischer Sicht betrachtet ist diese Änderung ebenso unkritisch zu sehen wie die erste. Zunächst ist es ist dem Benutzer freigestellt diese Option überhaupt zu aktivieren. Falls er sich dafür entscheidet sie zu aktivieren, bekommt er einen Hinweis, dass versteckte Dateien in dem von ihm gewählten Verzeichnis ignoriert werden. Sollte der Benutzer diesen Hinweis ignorieren und wählt im schlimmsten Fall ein Verzeichnis aus, welches nur versteckte Dateien enthält, so erhält er eine Fehlermeldung, weil keine Dateien gefunden wurden. Somit macht die Änderung sogar aus Usability-Sicht Sinn, weil unerfahrene Benutzer meist den Unterschied zwischen versteckten und nicht versteckten Dateien gar nicht kennen und versteckte Dateien eventuell auch noch nie gesehen haben.

2.3 Weitere Unterschiede

Im Folgenden werden Unterschiede zwischen den beiden Versionen aufgelistet, welche unserer Meinung nach keinen Einfluss auf die Sicherheit der Software haben. Es handelt sich dabei vor allem um kleinere Änderungen von Funktionen der Software, wie z. B. die Änderung von Knöpfen in der grafischen Benutzeroberfläche, oder um Behebung von Fehlern die mit einigen PC BIOS auftraten. Die gefundenen Unterschiede werden anhand des Verzeichnisbaumes der Software, gruppiert nach dessen Verzeichnissen, dargestellt. Sehr kleine Änderungen, wie z.B. geänderte Kommentare, werden ignoriert.

- Verzeichnis `Boot/Windows`
  - Die Datenstruktur `BootArguments` hat ein weiteres Feld `BootDriveSignature` bekommen, wodurch Partitionen von denen gestartet werden soll einfacher erkannt werden können
  - Einige BIOS haben I/O Fehler zu früh gemeldet, weswegen nun bestimmte Aktionen öfter versucht werden, bevor ein Fehler gemeldet wird
- Verzeichnis `Common`
  - Methode zum Parsen der Argumente von Befehlen auf der Kommandozeile akzeptiert kein »`-`« oder »`--`« mehr vor einem Parameter
  - Kleinere Änderungen der GUI (Größen, Buttons, Wortwahl etc.)
  - Neue Methoden für Tooltips über dem Taskbar Icon von TrueCrypt
  - Die Klasse `BootEncryption` hat eine neue Helfermethode namens `SystemdriveContainsNonStandardPartitions`, welche testet ob auf einem Laufwerk

  Partitionen sind, die nicht so geläufig sind, sprich andere als z.B. Fat16, Fat32, Extended, etc.
  - `Dlgcode.c` hat eine Methode namens `MountVolume`, deren Parameter `mountOptions` nun laut Kommentar zwingend »const« sein muss, weil sonst an anderer Stelle im Quelltext Probleme auftreten könnten

- Verzeichnis `Driver`
  - Einige Mutexe wurden entfernt
  - Quelltext zum Auslesen des »boot« Flags einer Partition hinzugefügt
  - Workaround zum Abschicken von »I/O Control Requests«, anstatt direkt abzubrechen
  - In der Methode `TCCreateDeviceObject`, wurde eine »magische Zahl« entfernt, welche aber nur dazu gedient hat gemountete Volumes zu erkennen
  - Fehlerbehebung beim Unmounten von Volumes wenn sie verschachtelt sind
  - Kompatibilitätsprobleme mit Windows-Tools wie diskmgmt, diskpart, vssadmin behoben

- Verzeichnis `Format`
  - Unter bestimmten Umständen sollte das Betriebssystem nicht in den Schlafmodus schalten dürfen, dies wird nun erzwungen (siehe Datei `TcFormat.c`)
  - Änderungen an der grafischen Benutzeroberfläche: Einige Warnungen Windows Vista-SP1 betreffend entfernt, Warnungen hinzugefügt für Benutzer die Dateien größer als 4 GB im Hidden-OS Modus auf einer nicht versteckten Partition speichern wollen
  - Andere Wizzards und Dialoge im Multibootbetrieb

- Verzeichnis `Main/Forms`
  - Button zum Aufruf des Spendenformulars entfernt

- Verzeichnis `Mount`
  - Mounten von Favoriten leicht verändert und Hilfe als »Baloon-Popup« über dem TrueCrypt TrayIcon hinzugefügt
  - Im Falle eines Absturzes wird nun auch die Datei `(winDir)\textbackslash MEMORY.DMP` ausgewertet

- Verzeichnis `Platform`
  - Methode `Erease` in Datei `Memory.cpp` verwendet nun unter Windows die Methode `RtlSecureZeroMemory` zum Löschen von Speicher, die von Microsoft extra dafür entworfen wurde, sicherer zu sein als die Standardmethode *memset*

2.4 Zusammenfassung

Wir haben den Quelltext der beiden TrueCrypt-Versionen 7.0a und 7.1a mit Hilfe des Open-Source-Tools *KDE Kompare* in Version 4.1.3 miteinander verglichen. Das Tool vergleicht zwei Verzeichnisbäume auf Dateiebene miteinander. Wir haben die gefundenen Unterschiede daraufhin analysiert, ob sie einen Einfluss auf die Sicherheit von TrueCrypt haben können. Dabei sind uns lediglich zwei Unterschiede aufgefallen, die man als »sicherheitsrelevant« betrachten kann. Zum einen dem Auslassen von versteckten Dateien beim Erhöhen der Entropie zum Ableiten eines Passwortes für verschlüsselte Container, und zum anderen das nun veraltete Verschlüsselungsalgorithmen auf der Kommandozeile ignoriert werden. Weitere Änderungen die es zwischen

den beiden Versionen gab, waren eher Fehlerkorrekturen und Änderungen an der grafischen Oberfläche. Aufgrund des eher kleinen Versionsnummernsprunges zwischen 7.0a und 7.1a sind auch eher keine großen Änderungen von uns erwartet worden. Dies geschieht meist in sogenannten *Major-releases* bei denen die vordere Ziffer der Versionsnummer erhöht wird. Sollten sich somit sicherheitsrelevante Fehler im Quelltext von TrueCrypt befinden, so sind diese bereits schon in der Version 7.0a enthalten, da es außer den zwei oben erwähnten keine Änderungen diesbezüglich gab.

Zusammenfassung der Ergebnisse aus Kapitel 2

- Lediglich zwei kleinere sicherheitsrelevante Änderungen zwischen den TrueCrypt-Versionen 7.0a und 7.1a, die aber nicht kritisch sind
  - Veraltete Verschlüsselungsalgorithmen werden auf der Kommandozeile nicht mehr als Option angeboten
  - Funktion zum Einrechnen des Inhaltes eines Verzeichnisses in das Benutzerpasswort ignoriert nun versteckte Dateien
- Die weiteren Änderungen sind Fehlerbehebungen oder Veränderungen der grafischen Benutzeroberfläche

3 Bewertung des OCAP Phase 1 Prüfberichts

3.1 Kommentare zum OCAP

Im Rahmen des Open Crypto Audit Project[1] wurde TrueCrypt von Junestam und Guido auf Sicherheitslücken überprüft. Der Bericht erschien im Februar 2014 und beruht auf der Analyse der TrueCrypt-Version 7.1a. Innerhalb dieses Kapitels evaluieren wir den Rahmen des Berichts, diskutieren die Erkenntnisse und überprüfen, ob die dargelegten Lücken durch automatisierte oder manuelle Analyse auffindbar sind.

3.1.1 Umfang des Berichts

Der Bericht bewertet Teilkomponenten des TrueCrypt Projekts: den *Bootloader*, den *Windows Kernel Treiber* und auch den Setup Prozess. Im Projekt wurde TrueCrypt gezielt auf Schwachstellen im Bereich Informationspreisgabe, Zugriffsrechte und ähnliche sicherheitsrelevante Lücken untersucht. Die Bewertung erfolgte mit diversen proprietären und öffentlich-verfügbaren Tools, manuellem Testen und direkter Quelltext-Analyse.

Insgesamt wird von 11 Schwachstellen berichtet, die sich wie folgt kategorisieren lassen: Unsichere kryptographische Funktionen, Pufferüberläufe und Speicherüberläufe.[2] Die Autoren des Berichts haben keinem der Funde ein hohes Ausmaß zugeschrieben. Zuletzt sprechen sie Empfehlungen für kurzfristige Lösungen der einzelnen Probleme aus.

In den beiden Anhängen liefert der Bericht Details über die verschiedenen Typen von Schwachstellen und Stellen von mangelnder Code-Qualität. Wir bewerten die Beschreibung der Funde, die bereitgestellten Beispiele und Empfehlungen als informativ und einleuchtend, jedoch mangelt es an Informationen, die es ermöglichen die Funde zu reproduzieren. Eine klare Beschreibung der Analyse-Methoden fehlt, daher ist es nicht möglich eine Aussage darüber zu treffen, ob die Ergebnisse der Analyse vollständig im Bezug auf die verwendeten Werkzeuge sind oder nur Teilergebnisse berichtet wurden. Weiterhin ist zu bemängeln, dass nicht genügend Quelltext zu den einzelnen Schwachstellen geliefert wird. Beispielsweise liegen uns nicht genügend Informationen vor, um Schwachstelle 2 des Berichts exakt zu verifizieren. Zusätzlich muss bedacht werden, innerhalb des OCAP-Bericht werden lediglich drei der sechs Teilprojekte: `Boot`, `Driver`, und `Setup` erwähnt. Die Projekte `Mount`, `Format` und Crypto werden im Bericht nicht behandelt.

3.1.2 Kommentare zu den einzelnen Ergebnissen des OCAP-Berichts

Im Folgenden kommentieren wir die Resultate des OCAP-Berichts.

Resultat 1 – Weak volume header key derivation algorithm. TrueCrypt verwendet zum Ableiten von Schlüsseln den PBKDF2 Algorithmus. Um ausreichend Sicherheit zu gewähren, sollten bei der Verwendung von PBKDF2 genügend Wiederholungen des Algorithmus ausgeführt werden. Im TrueCrypt Projekt in der Methode `get_pkcs5_iteration_count` (Date

[1] https://opencryptoaudit.org/

[2] Die OCAP-Klassifizierung der Resultate ist: 1 kryptographische Schwachstelle, 4 datenenthüllende Lücken, 3 mangelnde Datenvalidierungen, 2 Denial-of-Services und 1 mangelnde Fehlerbehandlung

`Pkcs5.c` des Teilprojekts `Boot`) wird die Anzahl der Wiederholungen gesetzt. In Abhängigkeit des Hash-Algorithmus, wählt TrueCrypt 1000 oder 2000 Wiederholungen, 1000 entspricht tatsächlich der minimal empfohlenen Anzahl [22]. Für kritische Schlüssel empfiehlt das NIST hingegen 10 000 000 [22]. Die Autoren gehen nicht konkret auf die zu selektierende Anzahl an Wiederholungen von PBKDF2 ein.

Resultat 2 – Sensitive information might be paged out from kernel stacks. Nach Angaben der Autoren kann das Betriebssystem durch TrueCrypt einen Speicherüberlauf erfahren, was dazu führt, dass Daten des Stacks in den Page-File des Betriebssystems auf die Festplatte geschrieben werden. Somit könnten Angreifer Zugriff auf nicht verschlüsselte, geheime Daten im Speicher erhalten. Werkzeuge zum Erkennen solcher Speicherüberläufe in dem Szenario sind rar. Daher kommen wir zu dem Schluss, dass diese Schwachstelle sehr schwer auszunutzen ist.

Resultat 3 – Multiple issues in the bootloader decompresser. Die Autoren fanden Buffer Overreads und Buffer Overflows in der `Decompressor.c` Datei im `Boot` Projekt. Leider werden keine Details beschrieben, wie sich dies auf die Sicherheit von Truecrypt auswirkt.

Da automatisierte Tools wie Coverity Probleme beim Scannen der Windows-Version von TrueCrypt Probleme haben (siehe 3.1.4), haben wir uns zu einer manuellen Analyse der Implikationen der genannten Schwachstellen entschlossen und sind zu dem Schluss gekommen, dass diese, falls sie von einem Angreifer ausgenutzt werden können, keine Gefahr für die Sicherheit von TrueCrypt darstellen. Die Details zur manuellen Analyse sind in Abschnitt 3.1.3 zu finden.

Resultat 4 – Windows kernel driver uses memset() to clear sensitive data. TrueCrypt benutzt die Funktion `memset` um Speicherstellen zu überschreiben. Die Autoren beschreiben, dass der Compiler diese Funktionsaufrufe wegoptimieren kann. Die Suche nach dem Schlüsselwort »memset« im TrueCrypt Projekt zeigt, dass die Funktion tatsächlich an mehreren Stellen benutzt wird. Allgemein wird die Methode benutzt, um Variablen zu initialisieren, bevor sie benutzt werden. In manchen Fällen ist die Benutzung jedoch gefährlich, wie die zwei Beispiele der Autoren in der Funktion `RMD160Final()` in Datei `RMD160.c`. zeigen. Wir können bestätigen, dass die Methodenaufrufe durch den Compiler wegoptimiert werden können. [5]. Wir können außerdem bestätigen, dass hierdurch möglicherweise vertrauliche Informationen aus dem Speicher ausgelesen werden können.

Resultat 5 – TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG kernel pointer disclosure. Der Bericht erwähnt eine Schwachstelle, über die ein Angreifer durch das Ausführen eines bösartigen Programms im Userspace eine Adresse im Kernelspace ermitteln kann. Dies kann, wie von den Autoren korrekt erkannt, verwendet werden, um Kernelspace ASLR zu brechen, was in der Praxis jedoch von eher geringer Bedeutung ist.

Resultat 6 – IOCTL_DISK_VERIFY integer overflow. In diesem Resultat berichten die Autoren einen Integer-Overflow in der Methode `ProcessVolumeDeviceControlIrp()` der Datei `Ntdriver.c`. Weiterhin beschreiben sie, dass der Fehler theoretisch auftreten könnte. Durch eine manuelle Datenfluss-Analyse der Variable, die die Schwachstelle auslöst, konnten wir feststellen, dass der Wert über die Methode `ExInterlockedRemoveHeadList()` der Microsoft Bibliothek `Ntoskrnl.lib` [3] in TrueCrypt fließt. Die Schwachstelle ist daher abhängig davon, ob die Methode `ExInterlockedRemoveHeadList()` einen Integer-Overflow abfängt. Daher liegt die Schwachstelle nicht in TrueCrypt, sondern in der Microsoft Bibliothek vor. Dennoch ist es möglich ggfs. Informationen aus dem Speicher auszulesen. Es ist daher sicherheitsrelevant und sollte behoben werden.

[3] https://msdn.microsoft.com/en-us/library/windows/hardware/ff545427%28v=vs.85%29.aspx

Resultat 7 – TC_IOCTL_OPEN_TEST multiple issues. Nach Angaben der Autoren erlaubt die Benutzung der Methode `zwCreateFile()` innerhalb von Funktion `ProcessVolumeDeviceConrrolIrp()` in Datei `Ntdriver.c` einem Angreifer beispielsweise Informationen abzuleiten, ob Dateien existieren (er kann sie jedoch nicht öffnen). Die Funktion `zwCreateFile()` wird ebenfalls in der Methode `TCOpenFsVolue()` innerhalb `Ntdriver.c` aufgerufen. Innerhalb der Funktion wird vorher die Methode `InitializeObjectAttributes()` mit dem Parameter `OBJ_KERNEL_HANDLE` aufgerufen. Diese setzt den Modus des aktuellen Aufrufers auf *Kernel Modus*[4]. Im Anschluss wird `zwCreateFile()` aufgerufen. Die Datei die dabei erzeugt wird, wird im Kernel Modus angelegt. Bei Zugriffen auf die Datei (auch externe) wird dabei keine Zugriffsberechtigung mehr überprüft. Es können Meta-Informationen wie etwa der Datei-Pfad extern abgegriffen werden, Einblick auf den Dateiinhalt ist dennoch von außen nicht möglich.

Resultat 8 – MainThreadProc() integer overflow In dieser Fundstelle berichten die Autoren von einer weiteren Schwachstelle im Bezug zu einem Integer-Overflow, der in der Funktion `MainThreadProc()` der Datei `EncryptedIoQueue.c` lokalisiert wurde. Die Schwachstelle kann einen Integer-Überlauf auslösen und Informationen preisgeben.

Resultat 9 – MountVolume() device check bypass. In Methode `VolumeThreadProc()` der Datei `Ntdriver.c` wird nach Angaben der Autoren der Zugriff mit `pThreadBlock->mount->wszVolume` nicht validiert, bevor er benutzt wird. Dies kann zu unbeabsichtigtem Verhalten TrueCrypts führen. Im Bericht wird nicht näher auf die Auswirkung durch einen möglichen Exploit eingegangen.

Resultat 10 – GetWipePassCount() / WipeBuffer() can cause BSOD. Innerhalb der Analyse durch das OCAP-Projekt konnte ein *Blue Screen of Death* ausfindig gemacht werden. Dieser befindet sich innerhalb der Funktionen `GetWipePassCount()` und `WipeCount()` der Datei `Wipe.c`. Der default-Handler einer switch-Anweisung, verursacht hier durch eine Aktion, die Administratorrechte benötigt, den Blue Screen. Wir können dies bestätigen, eine bessere Fehlerbehandlung sollte hier in Betracht gezogen werden.

Resultat 11 – EncryptDataUnits() lacks error handling. Diese Fundstelle des Prüfberichts befindet sich in der Funktion `EncryptDataUnits()` der Datei `Crypto.c`. Sollte ein Aufruf innerhalb der Funktion scheitern, wird dennoch wie gehabt im Programm fortgefahren. Insbesondere prüfen Methoden, die diese Funktion aufrufen, nicht ob die Methode erfolgreich ausgeführt wurde. Unsere Analyse ergab, dass die Methode beispielsweise durch `SetupThreadProc()` in der Datei `DriverFilter.c` aufgerufen wird. Im Anschluss werden die Rückgabewerte von `EncryptDataUnits()` unmittelbar – ohne zu Testen ob `EncryptDataUnits()` überhaupt erfolgreich war – auf das Speichermedium geschrieben. Dies kann zu korrupten Dateien führen.

3.1.3 Detaillierte Ausführung der manuellen Analyse zu Resultat 3

In den folgenden Absätzen legen wir die Gründe unseres Ergebnisses zu Resultat 3, Multiple issues in the bootloader decompresser, des OCAP-Berichts dar.

Der Dekompressor kommt prinzipiell nur zum Einsatz, wenn eine Vollverschlüsselung der Festplatte angewendet wurde. Die ersten Schritte im Startprozess sind dann wie folgt: nach dem Einschalten des PCs wird das BIOS ausgeführt. Danach wird der Code im Bootsektor ausgeführt, welcher die folgenden Schritte beinhaltet: da der TrueCrypt Bootloader in komprimierter Form vorliegt, muss dieser vor dem Ausführen vom Dekompressor entpackt werden. Hierfür wird zuerst der Dekompressor in den Speicher geladen. Danach wird die Prüfsumme des Dekompressors

[4] https://www.osronline.com/article.cfm?id=257

berechnet um dessen Integrität zu prüfen. Ist diese bestätigt, wird der komprimierte Truecrypt Bootloader in den Speicher geladen. Auch hier wird wieder mittels Prüfsumme die Integrität überprüft. Danach wird der Bootloader mit dem Dekompressor entpackt und ausgeführt. Erst hier wird der Benutzer nach seinem Passwort gefragt.

Zu beachten ist, dass die Berechnung der Prüfsumme nicht garantiert, dass die betroffene Datei nicht von einem Angreifer manipuliert wurde. Da es für einen Angreifer leicht möglich ist die Prüfsumme, mit der verglichen wird, zu ändern, kann mit diesem Vorgehen lediglich festgestellt werden, ob eine Datei aufgrund von Hardwareproblemen oder Ähnlichem beschädigt wurde, was auch der Intention der Programmierer entspricht.

Wir beschreiben nun, warum weder Buffer Overflows noch Buffer Overreads im Dekompressor eine Gefahr für die Sicherheit darstellen. In jedem Fall muss ein Angreifer aber den auf der Festplatte gespeicherten, komprimierten Bootloader manipulieren bzw. ganz austauschen, da dieser der einzige Input für den Dekompressor ist. Dies impliziert, dass der Angreifer physischen Zugang zu der Festplatte haben muss.

Buffer Overflows Die Gefahr von Buffer Overflows liegt im Allgemeinen darin, dass ein Angreifer unter Umständen den Programmfluss manipulieren und zuvor eingeschleusten, bösartigen Code ausführen kann. Es sind zeitaufwändige Analysen vonnöten um herauszufinden, ob eine Schwachstelle auch tatsächlich von einem Angreifer ausgenutzt werden kann, was im vorliegenden Fall jedoch aus folgendem Grund nicht notwendig ist: wie bereits erwähnt müsste ein Angreifer, um diese Schwachstelle ausnutzen zu können, den originalen, komprimierten Bootloader durch einen manipulierten ersetzen. Dieser würde die Schwachstelle im Dekompressor ausnutzen um den Programmfluss zu Code umzuleiten, den der Angreifer ebenfalls seinem manipulierten Bootloader hinzugefügt hat. Dies ist jedoch offensichtlich nicht zielführend, da es für den Angreifer einfacher ist, den Bootloader dahingehend zu ändern, dass dieser den zusätzlichen Code standardmäßig ausführt. Eine Ausnutzung des Buffer Overflows ist somit überflüssig.

Buffer Overreads Die Gefahr von Buffer Overreads liegt im Allgemeinen darin, dass ein Angreifer unter Umständen einen Speicherbereich auslesen kann. Sollten sich in diesem Schlüssel oder Passwörter im Klartext befinden, so könnten diese dem Angreifer in die Hände fallen. Wie groß der Speicherbereich ist, den ein Angreifer eventuell auslesen kann, ist von Fall zu Fall unterschiedlich und bedarf zeitaufwändiger, manueller Analysen. Im Fall der zwei im Dekompressor gefundenen Buffer Overreads ist dies jedoch nicht notwendig, da, wie beschrieben, der Dekompressor ausgeführt wird, bevor der Benutzer sein Passwort eingibt. Zu diesem Zeitpunkt befindet sich somit noch nichts von Interesse im Speicher, was eine Ausnutzung der Schwachstellen nutzlos macht.

3.1.4 Überprüfung der Ergebnisse mittels Coverity

Um die Resultate des OCAP-Berichts ergänzend zu überprüfen haben wir TrueCrypt für Windows mit dem automatisierten Code-Scanner Coverity analysiert (Details siehe Kapitel 5) und dessen Ergebnisse mit dem Bericht verglichen. Zunächst ist zu sagen, dass TrueCrypt für Windows zum Kompilieren auf drei verschiedene Microsoft Übersetzer zurückgreift. Insbesondere wird der Microsoft Visual C++ 1.52c Compiler aus dem Jahr 1994 verwendet[5]. Coverity unterstützt diesen alten Compiler nicht mehr. Der Compiler wird im Projekt `Boot` verwendet, dieses Projekt kann daher nicht mit Coverity analysiert werden. Die Analyse der Ergebnisse ergab, dass keine der Schwachstellen im OCAP-Bericht durch Coverity gefunden wurde, für Coverity sind diese Schwachstellen daher Falsch-Negative.

[5] http://support.microsoft.com/kb/145669

3.2 Zusammenfassung

Zusammenfassung der Ergebnisse aus Kapitel 3

- Der OCAP-Bericht beschreibt insgesamt 11 Schwachstellen von TrueCrypt.
- Klare Mängel weist der Bericht beim Beschreiben der zugrundeliegenden Analyse-Techniken und -Methoden auf. Die Resultate sind daher größtenteils nicht eindeutig reproduzierbar.
- Die automatisierte Code-Analyse konnte die im OCAP-Bericht dargelegten Schwachstellen nicht identifizieren.
- Für einige Schwachstellen, die der Bericht liefert, konnten wir mittels manueller Analyse nachweisen, dass sie in der realen Umgebung von TrueCrypt für Angreifer nur schwer auszunutzen sind.

4 Überprüfung der Verschlüsselungsmechanismen

4.1 Verschlüsselungsalgorithmen

Die untersuchte TrueCrypt Version 7.1a unterstützt für neu erstellte Volumes die Verschlüsselungsverfahren AES, Serpent, sowie Twofish. Als Hashfunktionen werden für diese Volumes RIPEMD-160, SHA-512 und Whirlpool verwendet. Die Verschlüsselung wird hierbei immer im XTS Mode durchgeführt. Im ersten Teil der Untersuchung wurden interne Schnittstellen und Datenflüsse in der TrueCrypt-Implementierung untersucht, um Punkte zu identifizieren, die einen Vergleich der implementierten Crypto-Funktionen mit analogen Funktionen in Open Source-Implementierungen zu ermöglichen. Für die Planung der durchzuführenden Tests wurde ein Call Graph für die unter Linux kompilierte TrueCrypt Software erstellt. Die Auswertung und Generierung der Graphen in erfolgte mit Hilfe der Scriptsprache Ruby in Kombination mit den Werkzeugen cscope [6] und rtags [1]. Zur Unterstützung der Analyse wurden auch Abstraktionen auf dem Graphen (z. B Schnittstellen zwischen Klassen) implementiert. Die Kontrolle der Einstiegspunkte des Call Graphen ergab, dass es sich bei diesen Funktionen um Teile des User Interfaces oder um Testfunktionen handelte. Lediglich die Funktionen `aes_encrypt_key`, `aes_encrypt_key192` und `aes_encrypt_key128` wurden nicht verwendet. In der untersuchten Version wird lediglich die Funktion `aes_encrypt_key256` benutzt. Abbildung 4.1 illustriert die Komplexität des berechneten Call Graphen, Abbildung 4.2 zeigt den Ausschnitt aus dem Call Graphen der alle Aufrufpfade zu der in Assembler implementieren Basisfunktion `aes_encrypt` enthält.

Um die Tests zur Überprüfung zu implementieren, wurden verfügbare Algorithmen und die entsprechenden Schnittstellen in den gängigsten Open Source-Bibliotheken untersucht. Tabelle 4.1 listet die verfügbaren Algorithmen, sowie die Verfügbarkeit des XTS Modes für die zu testenden Algorithmen, die in TrueCrypt, OpenSSL und Libgrypt implementiert sind, auf.

Algorithmus	**TrueCrypt**	**OpenSSL**	**Libgcrypt**
AES	+	+	+
Twofish	+	-	+
Serpent	+	-	+
AES-XTS	+	+	-
Twofish-XTS	+	-	-
Serpent-XTS	+	-	-

Tabelle 4.1: Verfügbare Algorithmen

Aus der Tabelle ergibt sich, dass Libgrypt für den Test der Implementierung aller symmetrischen Verschlüsselungsalgorithmen verwendet werden kann. Für diesen Test wurde die zu TrueCrypt kompatible Open Source Alternative tcplay [12] gewählt, die eine generische XTS-Implementierung enthält und die in Kombination mit Libcgrypt kompiliert werden kann. tcplay wurde so modifiziert, dass die für den Test benötigten Funktionen in einer Bibliothek bereitgestellt wurden.

Abbildung 4.1: TrueCrypt Call Graph

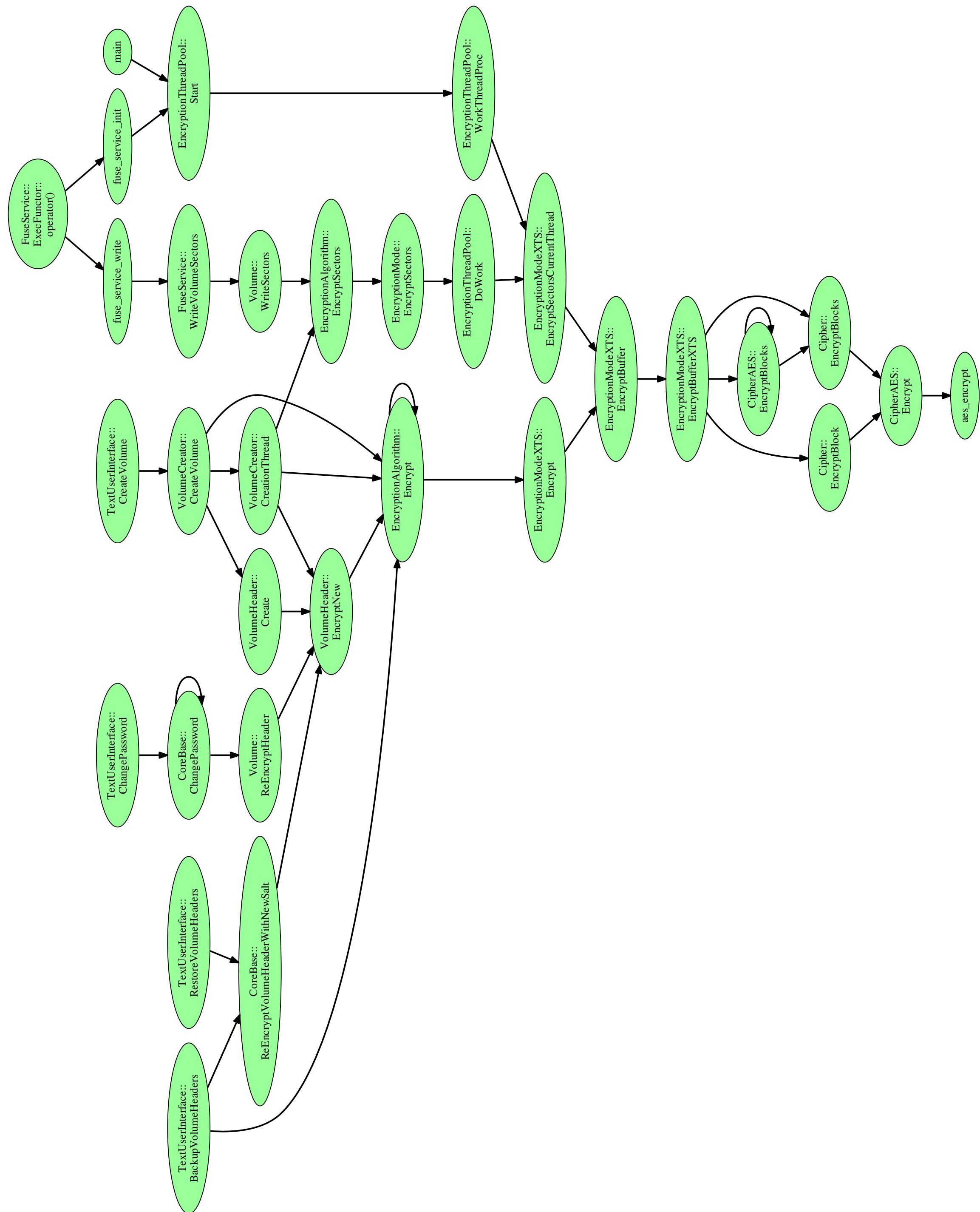

Abbildung 4.2: Call Graph zur Funktion `aes_encrypt`

Die grünen Knoten in Abbildung 4.3 zeigen einen Ausschnitt des Call Graphen, der für die AES-Verschlüsselung im XTS Mode verwendet wird. Dieser Teil enthielt keine Schnittstelle, die für einen direkten Vergleich mit tcplay/Libcrypt verwendet werden konnte. Hierfür wurde die Funktion *EncryptionAlgorithm::EncryptSectors* aus dem generischen Teil der TrueCrypt-Implementierung gewählt (blaue Knoten). Die für den Vergleich implementierten Tests wurden

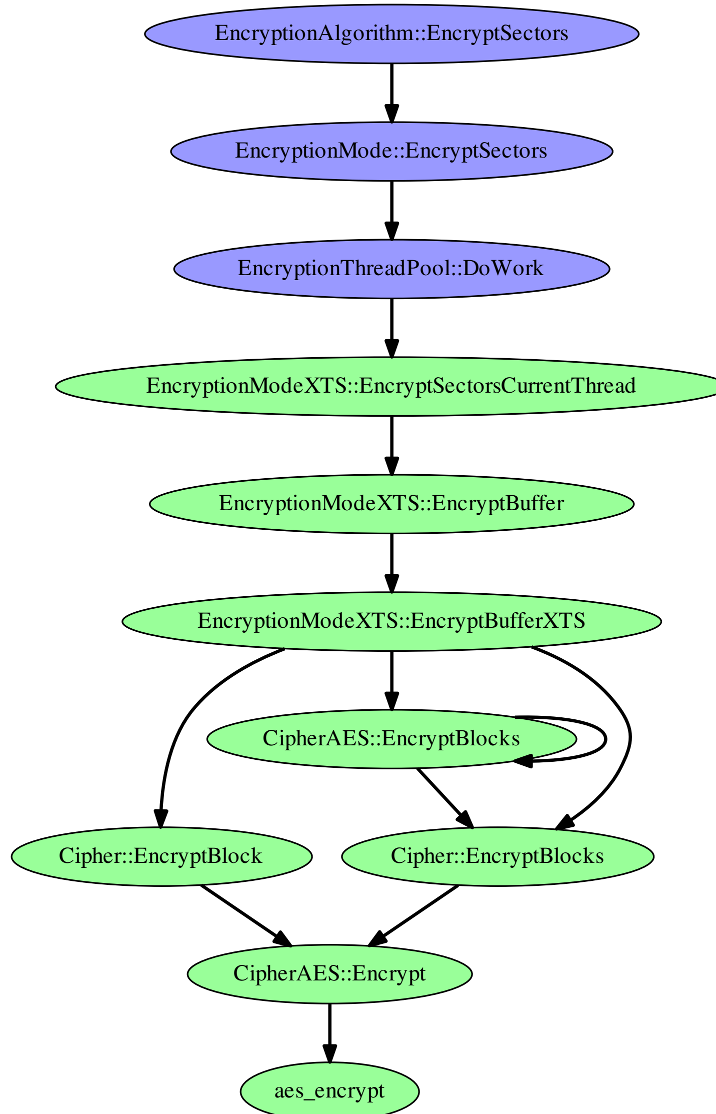

Abbildung 4.3: Call Graph Test AES Verschlüsselung

zunächst mit den im IEEE P1619TM/D16 Standard [13] definierten Testvektoren durchgeführt, um die Korrektheit des Ansatzes zu überprüfen. Dieser Test wurde mit der modifizierten tcplay Bibliothek erfolgreich durchgeführt. Die Daten für die weiteren Tests wurden zufallsgesteuert erzeugt:

- 256 Bit AES Schlüssel für die Datenverschlüsselung
- 256 bit AES “Tweak Key” für das Einbeziehen der Datenposition in die Verschlüsselung.
- 512 Byte Datenblöcke

Da auch Serpent und Twofish die gleichen Schlüssellängen wie AES verwenden, konnten die Tests für alle Algorithmen parallel mit den gleichen Testdaten durchgeführt werden. Die generierten Testdaten werden entsprechend dem in Tabelle 4.2 definierten Format für jeden Algorithmus in eine CSV Datei (Delimiter = ,) geschrieben. Die Daten werden hierbei im Hexformat (z. B. 4 Byte: CAFFEE02) ausgegeben.

Type	Algorithmus	Länge
Data Key	Random	32 Byte
Tweak Key	Random	32 Byte
Daten	Random	512 Byte
Cipher	AES, Serpent oder Twofish	512 Byte

Tabelle 4.2: Format Testdaten symetrische Verschlüsselung

Es wurden 10 Millionen Tests durchgeführt. Bei keinem der Tests wurde ein Unterschied im berechneten Cipher festgestellt.

4.2 Schlüsselableitung

In allen getesteten Verschlüsselungsverfahren werden 256 Bit Schlüssel verwendet. Die beiden für den XTS Mode benötigten Keys werden mit der PBKDF2-Methode, die in PKCS5 v2.0 spezifiziert ist, erzeugt. Abbildung 4.4 zeigt den Ausschnitt aus dem TrueCrypt Call Graph mit den Funktionen zur Schlüsselableitung als Endpunkte.

Für die Überprüfung der verwendeten Schlüsselableitungsfunktionen wurde OpenSSL gemäß Tabelle 4.3 verwendet.

TrueCrypt	OpenSSL
pkcs5HmacRipemd160.DeriveKey	PKCS5_PBKDF2_HMAC digest: EVP_ripemd160
pkcs5HmacSha1.DeriveKey	PKCS5_PBKDF2_HMAC_SHA1
pkcs5HmacSha512.DeriveKey	PKCS5_PBKDF2_HMAC digest: EVP_sha512
pkcs5HmacWhirlpool.DeriveKey	PKCS5_PBKDF2_HMAC digest: EVP_whirlpool

Tabelle 4.3: Schlüsselableitungsfunktionen TrueCrypt / OpenSSL

Die generierten Testdaten werden entsprechend dem in Tabelle 4.4 definierten Format in eine CSV Datei (Delimiter = ,) geschrieben. Bis auf die Längenangaben werden die Daten im Hexformat (z. B. 4 Byte: CAFFEE02) ausgegeben. Die Längenangaben werden als normale Zahl in Textform angegeben. Für jeden Algorithmus wird eine CSV Datei erzeugt.

Typ	Algorithmus	Länge
Passwort Länge	Random	variabel
Passwort	Random	Passwort Länge (Bytes)
Salt Länge	-	4
Salt	Random	4 Bytes
Schlüssellänge	Ripemd160	240 Byte
Schlüssel	Sha1, Sha512, Whirlpool oder Ripemd 160	Schlüssellänge in Byte

Tabelle 4.4: Format Testdaten Schlüsselableitung

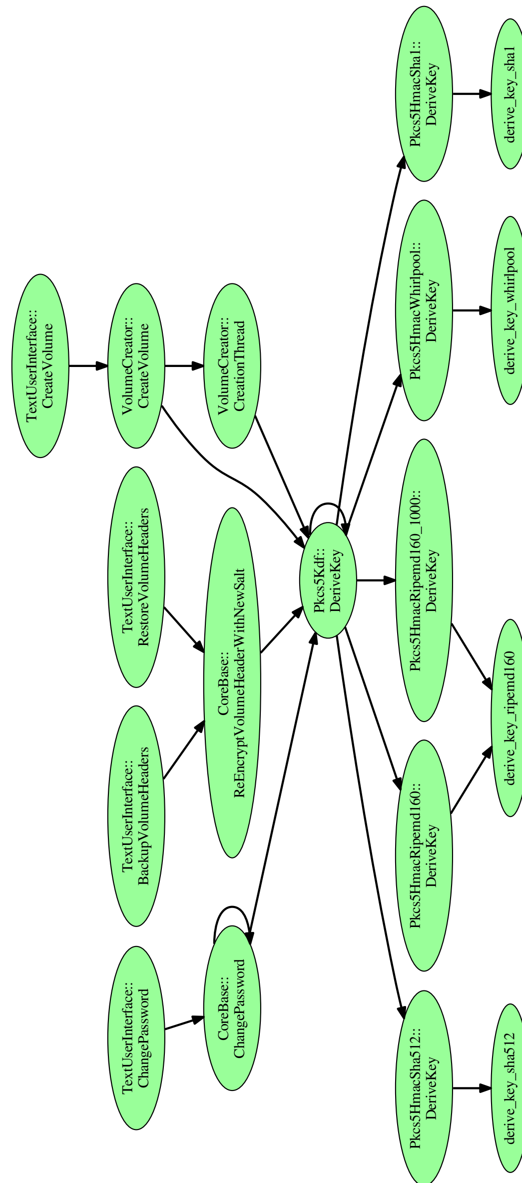

Abbildung 4.4: Call Graph Schlüsselableitungsfunktionen

Die Längenangaben wurden entsprechend der vorliegenden TrueCrypt Implementation gewählt. Die Schlüssellänge 240 entspricht der maximal möglichen Schlüssellänge multipliziert mit Zwei, da für einige Modes zwei Schlüssel benötigt werden. Die beiden Schlüssel werden jeweils aus dem längeren abgeleiteten Schlüssel ohne Überschneidung entnommen, was, da in der Ableitung immer der Maximalfall angenommen wird, kein Problem darstellt. An den berechneten Call Graphen wurde untersucht, ob in allen Fällen dieser Maximalwert verwendet wird. Auch hier wurden 10 Millionen Tests durchgeführt, bei denen keine Unterschiede zwischen den berechneten Schlüsseln festgestellt wurde.

4.3 Zufallszahlengenerierung

Die Zufallszahlengenerierung in der Datei Core/RandomNumberGenerator.cpp erscheint verbesserungswürdig. Die Funktion `RandomNumberGenerator::AddSystemDataToPool` sorgt für das Hinzufügen starker Entropie in den Zufallszahlenpool des Zufallszahlengenerators. Dieses kann scheinbar in zwei Modi erfolgen; `fast` und nicht-`fast` – gekapselt durch die Aufrufe `RandomNumberGenerator::GetDataFast` sowie `RandomNumberGenerator::GetData` in der Datei Core/RandomNumberGenerator.h. Diese Zufallszahlengenerierung wird nur von der Linux- oder MacOS-Variante von TrueCrypt verwendet. Die Zufallszahlengenerierung under Windows in der Datei Common/Random.c wird anhand der Windows-Crypto-Funktionen durchgeführt. Auch hier wurden im Rahmen des OCAP-2 Schwächen gefunden, die in Kapitel 9.2 diskutiert werden. Die `fast`-Variante findet lediglich in den Funktionen `CoreBase::ChangePassword`[1] sowie `FatFormatter::Format` Anwendung. Diese Anwendungen sind nicht so kritisch, könnten jedoch auch mit einem besseren Schema behandelt werden (siehe im Folgenden). Alle weiteren Aufrufe erfolgen in der non-`fast` Variante.

Jedoch ist die non-`fast` Variante so implementiert, dass sie in bestimmten Fällen auf die `fast`-Variante zurückfällt. In beiden Fällen wird zuerst der Entropiepool über die gesamte Länge mit Daten aus `/dev/urandom` aufgefüllt. Danach wird der Pool mit so vielen Daten aus `/dev/random` angereichert, wie aktuell zur Verfügung stehen. Hierin liegt das Problem: Sollten aktuell keine Daten in `/dev/random` zur Verfügung stehen, so wird keine weitere Entropie gegenüber der `fast`-Variante hinzugefügt. Dies gilt insbesondere auch für die erste Initialisierung des Entropiepools. Des Weiteren wird sämtliche Entropie, die seitens des Kernels zu `/dev/random` hinzugefügt wird auch `/dev/urandom` hinzugefügt. Damit gilt, dass der Output von `/dev/urandom` eine mindestens genauso große Entropie besitzt, wie die in diesem Schema per `/dev/random` dem Pool hinzugefügte, und diese insbesondere die gleiche Entropie ist.

Der Hintergrund, `/dev/random` überhaupt zu verwenden, liegt darin, dass der Linux-Kernel Anfragen erst dann beantwortet, wenn er hinreichend Entropie gesammelt hat. Allerdings dauert ein entsprechender Aufruf typischerweise sehr lang, da in jedes Byte eine entsprechende Mindestentropie eingegangen sein muss. Per `/dev/urandom` werden Anfragen jedoch bereits davor beantwortet. Für den Anwendungsfall von automatisierten Deployment, Eingebetteten Systemen und Virtualisierung (die üblicherweise Entropiemangel aufweisen) hilft die verwendete Implementierung nicht, da keine Mindestentropieanforderung gegen `/dev/random` durchgesetzt wird. Außerdem könnten nach der Sicherstellung dieser Mindestentropie im Linux-Kernel die Daten aus `/dev/urandom` direkt Verwendung finden und weitere Aufrufe an `/dev/random` wären unnötig[2].

Eine korrekte Implementierung würde stattdessen vorsehen, dass RandomNumberGenerator bei der Initialisierung ein Mindestmaß an Entropie aus `/dev/random` ausliest und erst dann die ersten Zufallszahlen liefert. Danach wären keine weiteren Aufrufe zu `/dev/random` nötig. Ab diesem Zeitpunkt können entweder die Daten von `/dev/urandom` direkt genutzt werden, oder einem Pool

[1] für die ersten n Runden; gefolgt von einem non-`fast`-Aufruf
[2] Anmerkung: Zu viel Entropie schadet nicht

hinzugefügt werden, oder diese Urentropie würde einen Deterministischen Zufallszahlengenerator mit einem Seed versorgen und gar keine weiteren Zugriffe auf `/dev/random` oder `/dev/urandom` wären nötig. Desweiteren existiert seit Linux 3.17 der SysCall `getrandom`[3]. Dieser greift das Problem auf, die hinreichende Initialisierung des Linux-Kernels sicherstellen zu können, ohne auf Ergebnisse von `/dev/random` warten zu müssen.

Diese Aussage auf Basis des Code Reviews sollte in einem praktischen Versuch beispielsweise anhand eines QEmu-Snapshot-Aufbaus oder entsprechender Debug-Points im SourceCode verifiziert werden.

4.4 Zusammenfassung

Zusammenfassung der Ergebnisse aus Kapitel 4

- Zur Identifikation geeigneter Punkte, die eine Untersuchung der in TrueCrypt verwendeten kryptographischen Funktionen ermöglichen, wurden automatisiert Call Graphen erstellt.
- Es wurden Tests für den Vergleich aktuell verwendeter kryptographischer Funktionen zur symmetrischen Verschlüsselung sowie zur Schlüsselableitung auf der Basis von gängigen Open Source-Bibliotheken implementiert. Die Testdaten wurden zufallsgesteuert generiert. Langlaufende Test (8×10 Millionen Testfälle) ergaben keine Unterschiede zur TrueCrypt Implementation.
- Eine Untersuchung des Zufallszahlengenerators für Linux ergab, dass in entropieschwachen Szenarien keine Garantie für ausreichende Entropie vorhanden ist, da die Nutzung des entropiestarken `/dev/random` stark verbesserungswürdig ist.

[3] http://man7.org/linux/man-pages/man2/getrandom.2.html

5 Untersuchung mittels automatisierter Code-Analyse

Automatisierte statische Code-Analyse einer Software hat sich als wirksames Hilfsmittel in der Softwareentwicklung etabliert und erlaubt eine umfangreiche sowie objektive Einschätzung eines Softwareprodukts. Die Forschung der letzten Jahre ermöglicht es Projekte großen Umfangs mit hoher Präzision unter anderem in Bezug auf Codequalität und Sicherheit zu analysieren. Dabei wird durch den Codescanner zunächst ein statisches Modell des Quelltextes abgeleitet, welches anschließend auf bekannte Muster, so genannten *Checkers*, untersucht werden kann. Die Software wird dabei nicht explizit ausgeführt; manche Analyse-Tools benötigen nicht mal kompilierbaren Code.

Starke Verbesserungsmöglichkeiten besitzen die Tools jedoch im Bereich der *Falsch Positiven*. Dies sind Warnung der Scanner, die nicht mit tatsächlich auftreten Mängeln übereinstimmen, sondern oftmals durch zu breite Annahmen der Scanner entstehen. Ein Großteil der wissenschaftlichen Forschung beschäftigt sich derweil damit, den Anteil der Falsch Positiven drastisch zu reduzieren.

In den letzten Jahren sind einige automatisierte Werkzeuge – unter kommerzieller Lizenz sowie Open Source – für verschiedene Programmiersprachen auf dem Markt gekommen. Innerhalb des Projektes haben wir drei der bekanntesten und am weitesten entwickelten statischen Code-Analyse-Werkzeuge für C und C++ benutzt, den Clang Static Analyzer, Coverity und Cppcheck. Im Folgenden beschreiben wir die Eigenschaften der Werkzeuge und gehen dabei insbesondere auf deren Stärken, Schwächen und Unterschiede ein.

Clang

Der Clang Static Analyzer ist ein Analyse-Tool für C, C++ und Objective-C, das den Quelltext der Software benötigt. Das Tool ist Teil des Clang-Projekts, ein Frontend der LLVM Compiler-Infrastrukur, die für alle gängigen Plattformen verfügbar ist. Spezialisiert ist das Analyse-Tool auf hohe Präzision bei der Analyse von Softwarefehlern im Allgemeinen, dies bedeutet eine geringe Falsch Positiv-Rate. Um die Falsch-Positive gering zu halten, wird dabei die *Symbolic Execution*-Technik verwendet. Bei dieser Technik wird das Programm systematisch entlang aller zulässigen Pfade gescannt und von konkreten Werten des Programms auf symbolische Werte abstrahiert. Clang integriert einige spezialisierte Checkers und verifiziert diese entlang der entdeckten Programm-Pfade. Innerhalb unserer Analyse haben wir alle – stabile und experimentelle – Standard-Checkers und zusätzlich die sicherheitsrelevanten Checkers aktiviert. Der Befehl, den wir zur Analyse benutzt haben ist daher:

```
scan-build -enable-checker core -enable-checker alpha.core \
           -enable-checker security -enable-checker alpha.security
```

Viele der Analysen, die auf Symbolic Execution beruhen, haben das Problem der *Path Explosion*. Die Anzahl aller möglichen Pfade in einem Programm wächst exponentiell mit seiner Größe. Um eine durchführbare Analyse zu erhalten, muss allgemein die Tiefe der Analyse auf realistischen Programmen beschränkt werden. Dies kann zu unvollständigen Ergebnissen führen.

Coverity

Coverity ist ein kommerzielles Tool zur statischen Codeanalyse und bietet Softwareentwicklern die Möglichkeit ihre Programme auf Softwarefehler zu untersuchen. Interessant ist das Tool, da alle möglichen Pfade – insbesondere inter-prozedural[1] – des Programms untersucht werden können. Coverity setzt voraus, dass die zu untersuchende Software kompiliert werden kann. Das Tool klinkt sich dann in den Buildprozess ein und transformiert den Bytecode in ein statisches Modell, das anschließend von Coverity nach typischen Mustern, den Checkers, untersucht werden kann. Die Muster decken verschiedene Kategorien von Softwarefehlern ab, beispielsweise berichtet Coverity über mangelnde Codequalität aber auch Sicherheitslücken werden aufgezeigt.

Die Analyse mit Coverity erfordert drei Schritte:

1. Konfigurieren des Compilers für Coverity. (Einmalig pro Compiler)

2. Kompilieren des Projekts mittels vorgeschaltetem Coverity. Hierbei wird der Code in eine Coverity-interne Zwischensprache transformiert.

3. Analyse der Zwischensprache, hierbei können die gewünschten Checker aktiviert werden.

Wir haben die statische Code-Analyse auf den Linux und den Windows-Build angewendet. Im Folgenden beschreiben wir die konkrete Befehlsreihenfolge zum Scannen mit Coverity.

Details für Linux:

1. Unter Linux kann TrueCrypt mittels des Compilers `gcc`[2] kompiliert werden.

   ```
   cov-configure gcc
   ```

2. Coverity erwartet hier einen Build-Befehl. Als Argument muss zusätzlich ein Verzeichnis angegeben werden in dem die Zwischensprache gespeichert wird. Als weiteres Argument folgt der komplette Build-Befehl von TrueCrypt. Es ist hierbei wichtig, darauf zu achten, dass zuvor ein `make clean` aufgerufen wird, so dass alle vorherigen bereits kompilierten Dateien verworfen und neu kompiliert werden.

   ```
   cov-build --dir ... make NOGUI=1 WXSTATIC=1
   ```

3. Analyse der Zwischensprache mit den spezifischen Checkern.

   Dies erfolgt mittels: `cov-analyze --dir ... --all --aggressiveness-level medium`

   Dabei werden die Dateien im angegebenen Verzeichnis des `cov-build` Befehls analysiert.

Der Parameter `aggressiveness-level` gibt an, wie intensiv gesucht werden soll. Dies erhöht zum einen die Laufzeit der Analyse, zum anderen werden allerdings auch mehr Falsch Positive berichtet.

Beispielsweise wurden für die Konfiguration `aggressiveness-level high` insgesamt 1382 Resultate gefunden, im Vergleich hierzu wurde bei der Einstellung `medium` insgesamt 58 Fehler gefunden. Für die weitere manuelle Inspektion der Ergebnisse haben wir uns auf die Resultate mit `aggressiveness-level medium` beschränkt.

[1] Der Effekt von Methodenaufrufen innerhalb einer anderen Methode wird beachtet. Dies ermöglicht ein komplettes Model der Software.

[2] https://gcc.gnu.org/

Details für Windows: Ein Build von TrueCrypt unter Windows benutzt insgesamt drei verschiedene Compiler, dies führt bei der Analyse mit Coverity zu Problemen. Das TrueCrypt-Projekt besteht aus den Projekten `Boot`, `Driver`, `Crypto`, `Mount`, `Format` und `Setup`. Das Teilprojekt `Boot` benutzt den Microsoft Visual C++ 1.52c Compiler aus dem Jahr 1994. Coverity unterstützt diesen Compiler nicht: Es ist für diesen nicht konfigurierbar (Schritt 1). Für die Coverity-Analyse kann daher nicht der übliche TrueCrypt Build-Prozess des Visual Studio-Projekts verwendet werden. Daher war es notwendig den Build-Prozess auf die einzelnen Teilprojekte und deren Buildvorgänge aufzubrechen.

Außer `Crypto` und `Driver`, hängen alle anderen Projekte direkt von `Boot` ab. Da Coverity einen funktionierenden Build-Befehl benötigt, muss dafür gesorgt werden, dass `Boot` zuvor bereits kompiliert wird. Im Detail (siehe Schritt 2) muss folgendes erfolgen:

1. Konfigurieren des Visual Studio Compilers:

   ```
   cov-configure --msvc .../cl.exe
   ```

2. Kompilieren des Projektes `Boot` ohne Coverity:

   ```
   msbuild  /p:Configuration=Boot .../TrueCrypt.sln
   ```

   Analyse aller anderen Projektes mit Coverity:

   ```
   cov-build --dir  ... msbuild /p:Configuration={project} ../TrueCrypt.sln
   ```

   Dabei sollten die Projekte nach ihren Abhängigkeiten kompiliert werden, am besten in folgender Reihenfolge `Driver`, `Mount`, `Format` und `Setup`. Das Projekt `Crypto` lässt sich nicht einzeln kompilieren, die Teilprojekte referenzieren es direkt.

3. Die Coverity-Analyse starten:

   ```
   cov-analyze -- dir ... --all
   ```

Die Analyse von Coverity benutzt durch Schritt 2 den zuvor erstellten Build von `Boot`. Daher analysiert Coverity dieses Projekt nicht. Folglich können keine Ergebnisse innerhalb `Boot` erwartet werden. Die Analyse liefert mit dem Default-Wert `aggressiveness-level normal` bereits 158 Funde, auf die wir uns im Folgenden konzentrieren.

Cppcheck

Cppcheck ist ein Tool zum Finden von Softwarefehlern in C und C++. Das Werkzeug analysiert direkt den Quelltext des Programms. Bei der Analyse wird nach Mustern zur Erkennung von Speicherlecks, nicht-initialisierten Variablen oder Null-De-Referenzierungen zurückgegriffen. Ein Vorteil des Scans auf Quelltextebene ist, dass die Semantik der Sprache mit in Betracht gezogen werden kann.

Cppcheck verspricht eine 0-Prozent Falsch Positive Rate. Dies geht allerdings damit einher, dass einige potenziell interessante Funde bei der Analyse verworfen oder nicht berichtet werden. Im Fall von TrueCrypt 7.1a sind die meisten Resultate von Cppcheck schlechten Programmierpraktiken zuzuschreiben, die zu Sicherheitslücken führen könnten aber dies nicht zwangsläufig tun.

Da die einzelnen Resultate stark von der zugrunde liegenden Technologie abhängen ist es eine eine Notwendigkeit bei der automatisierten Sicherheitsanalyse auf ein breites Spektrum an Werkzeugen zurückzugreifen. Dies bestätigt ebenfalls ein Vergleich der (Linux-)Funde untereinander. Von den 58 Funden von Coverity konnten wir 8 den 209 Resultaten von Cppcheck zuordnen. (Coverity gruppiert Fehlermeldungen des gleichen Typen in der selben Datei als ein Resultat. Ohne die Gruppierung lassen sich insgesamt 17 Warnungen zuordnen.) Im Gegensatz hierzu konnten wir keine Überschneidung der 71 Resultate von Clang mit den anderen Tools feststellen.

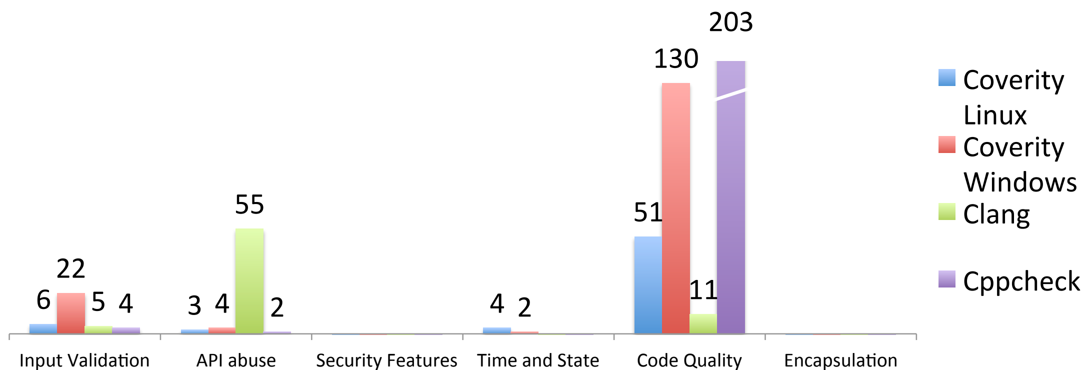

Abbildung 5.1: Klassifizierung nach Seven Pernicious Kingdoms [21]. Die Softwarefehler in Input Validation entsprechen den sicherheitsrelevanten Softwarefehlern.

5.1 Übersicht über die Analyse-Ergebnisse

In diesem Abschnitt geben wir einen Überblick über die Warnungen, die von den verschiedenen Werkzeugen berichtet wurden. Um die Ergebnisse vergleichbar zu machen, haben wir sie anhand der Taxonomie *Seven (plus one) Pernicious Kingdoms*, die von der Fortify Software Security Forschungsgruppe entwickelt wurde, klassifiziert. Die Klassifizierung ist als CWE-700 festgehalten und zielt darauf ab, Softwarefehler eher als »konkrete spezifische« Probleme anstelle von »abstrakten theoretischen« zu kategorisieren.

Die Softwarefehler, die von den verschiedenen Analyse-Werkzeugen berichtet werden, können auf die acht »Kingdoms« abgebildet werden. Die acht »Kingdoms« sind (nach absteigender Relevanz sortiert)

- Input Validation and Representation
- API Abuse
- Security Features
- Time and State
- Errors.
- Code Quality
- Encapsulation
- Environment

In diesem Bericht haben wir die Kategorien Errors und Environment in die Kategorie Code Quality integriert, da in den ersten beiden Kategorien lediglich eine geringe Anzahl an Softwarefehlern berichtet wurde und wir diese schlechten Programmierpraktiken zuschreiben konnten.

Die Klassifizierung der Resultate der Scanner Clang, Coverity und Coverity nach CWE-700 befindet sich in der Abbildung 5.1. Die meisten gefundenen Fehler von Coverity und Cppcheck sind Mängeln in der Code-Qualität zuzuschreiben. Hieran erkennt man auch, dass Cppcheck auf Qualitätsmängel im Code spezialisiert ist. Der Clang Static Analyzer hingegen berichtet hauptsächlich Fehler, die wir als falsche API-Benutzung einstufen.

ID	Dateiname	Methode	Typ	Angegebenes Risikolevel
CLANG-1	`/Crypto/Sha2.c`	sha_end1	Out-of-bounds access	hoch
CLANG-2	`/Crypto/Sha2.c`	sha_end2	Out-of-bounds access	hoch

Tabelle 5.1: Risikoeinstufung der Clang-Ergebnisse

5.2 Details zu den Fehlerberichten und Empfehlungen

In diesem Abschnitt berichten wir über die Resultate der drei Analyse-Tools bzgl. TrueCrypt.

Dabei gehen wir auf jedes Tool einzeln ein und diskutieren die sicherheitsrelevanten Klassen von Warnungen. Die sicherheitsrelevanten Softwarefehler fallen hauptsächlich in eine der ersten vier »Kingdoms« (Abbildung 5.1).

Mittels einer ersten Aufbereitung der Warnungen konnten wir potenzielle sicherheitsrelevante Softwarefehler ausfindig machen. Jeden dieser Softwarefehler haben wir anschließend detailliert analysiert und sind zum Ergebnis gekommen, dass es sich um eine falsch positive Warnung handelt. In diesem Abschnitt haben wir daher die interessanten, nicht-trivialen Beispiele aufbereitet.

5.2.1 Clang Static Analyzer

Die Analyse von TrueCrypt mittels des Clang Static Analyzer lieferte insgesamt 71 Meldungen. Nach manueller Analyse der Resultate sind wir der festen Überzeugung, dass alle Meldungen Falsch-Positive sind. Ein Großteil der Ergebnisse (55) berichtet über Datentypenumwandlungen von Pointern, die zwar prinzipiell zu korrupten Daten führen können, aber tatsächlich im vorliegenden Fall unproblematisch sind. Die hohe Anzahl an Falsch Positiven lässt sich dadurch erklären, dass solche Operationen extrem schwer durch automatische statische Code-Analyse erkannt werden können, da beispielsweise keine präzise semantische Behandlung von Integer-Arithmetik (konkret: Semantik auf Bit-Level-Ebene) durchgeführt wird, wie es bei Symbolic Execution sonst üblich ist.

Die 11 Ergebnisse, die Clang bezüglich Code-Qualität liefert sind nicht sicherheitsrelevant, daher beziehen wir sie hier nicht mit ein und verweisen auf Kapitel 6. Dennoch möchten wir darauf hinweisen, dass es möglich ist die meisten der Warnungen durch kleine Verbesserungen am Quelltext zu beheben. Dadurch kann die Anzahl der Resultate der drei Werkzeuge drastisch reduziert werden.

Die verbleibenden sechs Fundstellen betreffen potenzielle Out-of-bound Zugriffe. In Tabelle 5.1 sind zwei interessante Fälle gelistet, die eine weitere Analyse benötigten, um letztendlich als Falsch-Positive kategorisiert werden zu können. Die beiden Warnungen werden durch zwei ähnliche Funktionen hervorgerufen. Beide führen sogenannte »digest« Berechnung für die kryptographischen Hashfunktionen SHA256 und SHA512 durch.

In Abbildung 5.2 ist ein möglicher Out-of-Bound Zugriff des Arrays `ctx->wbuf` innerhalb der Funktion `sha_end2` abgebildet. Ein Out-of-Bound Zugriff kann eine ausnutzbare Sicherheitslücke darstellen. Daher haben wir weiterhin untersucht, unter welchen Umständen es zu dem Array-Zugriff kommen kann.

Die rote Box in Abbildung 5.2 zeigt den komplexen Ausdruck, der den Index berechnet mittels dem auf das Array zugegriffen wird. In diesem Beispiel werden speziell die Indizes durch aufwändige binäre Operationen wie Shifts, OR und AND Ausdrücke berechnet. Wie wir bereits erwähnt haben, sind genau solche Ausdrücke extrem schwer – automatisch und manuell – zu analysieren.

In diesem Fall jedoch hilft die Tatsache, dass das betrachtete Array eine feste Größe hat. Daher kann erwartet werden, dass nur eine begrenzte Anzahl an Ausführungspfaden im Programm existiert, die auf das Array zugreift. Daher ist es sinnvoll, ein durch Symbolic Execution gestütztes

```
563  static void sha_end2(unsigned char hval[], sha512_ctx ctx[1], const unsigned int hlen)
564  {   uint_32t    i = (uint_32t)(ctx->count[0] & SHA512_MASK);
565
566      /* put bytes in the buffer in an order in which references to   */
567      /* 32-bit words will put bytes with lower addresses into the    */
568      /* top of 32 bit words on BOTH big and little endian machines   */
569      bsw_64(ctx->wbuf, (i + 7) >> 3);

         2 ← Within the expansion of the macro 'bsw_64':
           a  Access out-of-bound array element (buffer
              overflow)

570
571      /* we now need to mask valid bytes
572      /* a single 1 bit and as many zero
573      /* we can always add the first padd
574      /* buffer always has at least one e
575      ctx->wbuf[i >> 3] &= li_64(fffffff
576      ctx->wbuf[i >> 3] |= li_64(0000000
```

```
{ int _i = ((i + 7) >> 3); while(_i--) ((uint_64t*)ctx->
wbuf)[_i] = (((uint_64t)(((((((uint_32t)(((uint_64t*)ctx->
wbuf)[_i]))) >> 24) | ((((uint_32t)(((uint_64t*)ctx->
wbuf)[_i]))) << (32 - 24))) & 0x00ff00ff) | (((((uint_32t
)(((uint_64t*)ctx->wbuf)[_i]))) >> 8) | ((((uint_32t
)(((uint_64t*)ctx->wbuf)[_i]))) << (32 - 8))) & 0xff00ff00
)))) << 32 | ((((((uint_32t)((((uint_64t*)ctx->wbuf
)[_i]) >> 32))) >> 24) | ((((uint_32t)((((uint_64t
*)ctx->wbuf)[_i]) >> 32))) << (32 - 24))) &
 0x00ff00ff) | (((((uint_32t)((((uint_64t*)ctx->wbuf)[_i]
) >> 32))) >> 8) | ((((uint_32t)((((uint_64t*)ctx
->wbuf)[_i]) >> 32))) << (32 - 8))) & 0xff00ff00
))); }
```

Abbildung 5.2: SHA512 out of bound array access.

ID	Dateiname	Methode	Typ	Angegebenes Risikolevel
COV-1	`/Main/TextUserInterface.cpp`	AskPassword	Out-of-bounds access	hoch
COV-2	`/Main/CommandLineInterface.cpp`	ToKeyfileList	Various	hoch
COV-3	`/Platform/Unix/File.cpp`	GetPartition-DeviceStartOffset	Overflowed return value	mittel

Tabelle 5.2: Risikoeinstufung der Coverity-Ergebnisse für Linux

Tool zu benutzen, das sich auf Test-Case Generierung fokussiert.

KLEE ist ein solches Werkzeug: Es kann C und C++ analysieren und symbolisch Test-Cases generieren. Mittels des integrierten STP Theorem Prover können Ausdrücke mit Bitvektor-Arithmetik präzise behandeln werden [4].

KLEE konnte eine vollständige symbolische Analyse innerhalb von weniger als einer Minute erfolgreich abschließen und dabei 17 verschiedene Ausführungspfade im analysierten Teilprogramm ausfindig machen. Zu jedem einzelnen Programmpfad wurde der Index des Array-Zugriffs generiert. Keiner dieser Indizes führt zu einem Out-of-Bound-Zugriff auf das Array.

5.2.2 Coverity

Linux-Scan Insgesamt liefert der Scan der Linux-Version von TrueCrypt mit Coverity 58 Resultate. Auffallend hoch ist der Anteil an Ergebnissen im Bereich der Code Qualität, dies betrifft insbesondere einige nicht-initialisierte Variablen in Konstruktoren. Wir konnten diese Softwarefehler durch manuelle Analyse als nicht sicherheitsrelevant einstufen, dennoch ist es sinnvoll diese Fehler in TrueCrypt zu beheben. Hierfür sind die Ergebnisse von Coverity äußert hilfreich.

Dieser Abschnitt konzentriert sich auf drei durch Coverity als sicherheitsrelevant (Kategorie Input Validation) eingestuften Ergebnisse (siehe Tabelle 5.2). Unsere Analyse ergab, dass diese drei Falsch-Positive sind. Im Folgenden werden wir dies darstellen.

COV-1 In Abbildung 5.3 sind die Details des ersten Ergebnis von Coverity aufgeführt. Die Methode `Set` wird aufgerufen, wobei der zweite Parameter 0 ist. Innerhalb der Methode `Set`

```
110                         if (!verPhase && length < 1)
111                         {
    CID 10309 (#1 of 1): Out-of-bounds access (OVERRUN)
    5. underrun-buffer: Underrunning passwordBuf at -1 by passing argument 0UL [show details]
112                                 password->Set (passwordBuf, 0);
113                                 return password;
114                         }
115
```

(a) Die Methode wird mit Parametern aufgerufen, die laut Coverity zu einem Buffer-Underrun führt.

```
        7. loop_bounded_by_parm: charCount bounds loop condition i < charCount .
133             for (size_t i = 0; i < charCount; ++i)
134             {
        8. index_parm_via_loop_bound: Pointer password is accessed by i , whose upper bound is charCount in loop conditional i < charCount .
135                     conv.c = password[i];
136                     passwordBuf[i] = conv.b[lsbPos];
137                     for (int j = 0; j < (int) sizeof (wchar_t); ++j)
138                     {
139                             if (j != lsbPos && conv.b[j] != 0)
140                                     unportable = true;
141                     }
142             }
```

(b) Details innerhalb Methode `Set`, die in der oberen Abbildung aufgerufen wird.

Abbildung 5.3: Details zum Ergebnis COV-1.

ist der zweite Parameter die Variable `charCount`. In Zeile 135 gibt Coverity eine Warnung aus, da es annimmt das Array `passwordBuf` wird mit Index `i` de-referenziert. Es ist klar zu sehen, dass die Variable `i` jedoch nach unten durch 0 und nach oben durch `charCount` beschränkt ist. Außerdem ist in diesem Aufrufkontext die Variable `charCount` sogar 0 und die Schleife wird direkt übersprungen. Daher kann kein Buffer-Underrun auftreten.

COV-2 Das zweite Ergebnis (Abb. 5.4) resultiert aus einer mangelnden Initialisierung einer Variable innerhalb des Konstruktors einer eingebundenen Bibliothek (wxWidgets). Später wird auf die nicht initialisierte Variable zugegriffen. Die Variable ist jedoch ein Feld eines Objektes innerhalb der Bibliothek. Die Initialisierung obliegt daher dem Entwickler der Bibliothek und nicht TrueCrypt.

COV-3 Abbildung 5.5 zeigt einen Integerüberlauf. Es wird eine Zeile aus einer Datei gelesen und anschließend in eine vorzeichenlose Integervariable mit 64 Bits transformiert. Die anschließende Multiplikation mit einer anderen Integervariable kann dazu führen, dass der maximale Bereich einer Variable vom Typ `uint64` von $-2^{63}+1$ bis $2^{63}-1$ überschritten wird.

Die Variable `line` ist in jeden Fall durch die Speicherkapazität eines `long` (von $-2^{31}+1$ bis $2^{31}-1$) begrenzt (siehe Kommentar). Das Resultat von `GetDeviceSectorSite()` ist ebenfalls von Typ `long`. Eine Multiplikation von zwei Variablen des Typs `long` kann jedoch den maximalen Bereich von `uint64` nicht überschreiten.

Windows-Scan Der Scan der Windows-Variante lieferte insgesamt 158 Warnungen. Auch hier ist anzumerken, dass ein Großteil der Warnungen ebenfalls wieder Code-Qualität betrifft. Insbesondere sind die 22 Befunde aus der Kategorie Input Validation von Interesse. Wir zeigen anhand von drei (siehe Tabelle 5.3) ausgewählten, sicherheitsrelevanten Beispielen das Vorgehen der manuellen Analyse. Mit den restlichen Funden sind wir analog vorgegangen.

```
499                 // Handle escaped separator
      1. var_decl: Declaring variable arr .
500                 wxArrayString arr;
501                 bool prevEmpty = false;
```

(a) Die Variable `arr` wird mit dem Default-Konstruktor initialisiert.

```
522                         if (arr.Count() < 1)
523                         {
524                                 arr.Add (L"");
525                                 continue;
526                         }
    CID 10307 (#2 of 2): Uninitialized pointer read (UNINIT)
    9. uninit_use_in_call: Using uninitialized value arr.m_pItems when calling Last . [show details]
527                         arr.Last() += token.empty() ? L',' : token;
```

(b) Später wird das letzte Element des Arrays gelesen.

Abbildung 5.4: Details zum Ergebnis COV-1.

```
129             // HDIO_GETGEO ioctl is limited by the size of long
130             TextReader tr ("/sys/block/" + string (Path.ToHostDriveOfPartition().ToBaseName()) + "/" + string (Path.ToBaseName()) +
131
132             string line;
133             tr.ReadLine (line);
    1. overflow: Multiply operation overflows on operands TrueCrypt::StringConverter::ToUInt64(line) and this->GetDeviceSectorSize() . Example values for
    operands: this->GetDeviceSectorSize() = 2147483648, TrueCrypt::StringConverter::ToUInt64(line) = 18397239859749060607.
    CID 10284 (#1 of 1): Overflowed return value (INTEGER_OVERFLOW)
    2. overflow_sink: Overflowed or truncated value (or a value computed from an overflowed or truncated value)
    TrueCrypt::StringConverter::ToUInt64(line) * this->GetDeviceSectorSize() used as return value.
134             return StringConverter::ToUInt64 (line) * GetDeviceSectorSize();
```

Abbildung 5.5: Details zum Ergebnis COV-3.

ID	Dateiname	Methode	Typ	Angegebenes Risikolevel
COV-4	`/Common/Keyfiles.c`	KeyFilesApply	Out-of-bounds write	hoch
COV-5	`/Mount/Mount.c`	DismountIdleVolumes	Insecure data handling	mittel
COV-6	`/Format/InPlace.c`	FastVolume-HeaderUpdate	Overflowed return value	mittel

Tabelle 5.3: Risikoeinstufung der Coverity-Ergebnisse für Windows

```
261                     for (size_t i = 0; i < keyfileData.size(); i++)
262                     {
263                             crc = UPDC32 (keyfileData[i], crc);
264
    23. incr: Incrementing writePos. The value of writePos may now be up to 64.
265                             keyPool[writePos++] += (unsigned __int8) (crc >> 24);
    CID 10927 (#1 of 1): Out-of-bounds write (OVERRUN)
    24. overrun-local: Overrunning array keyPool of 64 bytes at byte offset 64 using index writePos++ (which evaluates to 64).
266                             keyPool[writePos++] += (unsigned __int8) (crc >> 16);
267                             keyPool[writePos++] += (unsigned __int8) (crc >> 8);
268                             keyPool[writePos++] += (unsigned __int8) crc;
269
    10. Condition writePos >= 64, taking false branch
    14. Condition writePos >= 64, taking true branch
    18. Condition writePos >= 64, taking false branch
    19. cond_at_most: Checking writePos >= 64 implies that writePos has the value which may be up to 63 on the false branch.
270                             if (writePos >= KEYFILE_POOL_SIZE)
271                                     writePos = 0;
272
```

Abbildung 5.6: Details zum Ergebnis COV-4.

COV-4 Dieser Fund (siehe Abb. 5.6) von Coverity ist falsch positiv: Coverity findet hier einen Out-of-bounds Schreibzugriff. Das Array `keyPool` hat die Länge 64 und die statische Analyse ergibt, dass der Index 64 beschrieben wird. Dies würde zu einem Schreibzugriff außerhalb des Arrays führen. Die Analyse erkennt jedoch nicht den Fakt, dass die Operation `writePos++` genau viermal innerhalb des Loops aufgerufen wird. Da `writePos = 0` initialisiert wird, gilt nach der Schleife die Invariante `writePos % 4 == 0`. Im `if`-Statement in Zeile 270 hat `writePos` somit einen Wert, der ein Vielfaches von `4` ist. Entweder ist der Wert genau `KEYFILE_POOL_SIZE = 64`, oder er ist kleiner und damit höchstens `60`. Ein weiterer Durchlauf der Schleife führt daher zu keinem Schreibzugriff außerhalb der Länge des Arrays.

COV-5 Das Resultat in Abbildung 5.7 zeigt einen sogenannten Taint-Fluss. Der gezeigt Code ist innerhalb einer Schleife. Die Funktion `DeviceIoControl` sendet eine Kontrollsequenz an einen Treiber, der die entsprechende Operation durchführt. Diese Operationen können unter anderem Lese- und Schreibzugriffe auf einem Datenvolumen sein. Die statische Analyse vermutet hier, dass nach dem ersten Durchlauf der Schleife das Argument `prop` sensible Daten z.B. durch einen Lesezugriff erhalten könnte. Im zweiten Durchlauf wird dann erneut die Funktion `DeviceIoControl` aufgerufen. Diese könnte nun die Daten auf einen anderen Datenträger schreiben. Im konkreten vorliegenden Fall ist zu sehen, das in jedem Schleifendurchlauf lediglich Daten vom Treiber geladen werden, außerdem werden die Speicherwerte zu `prop` durch den Aufruf von `memset` (Zeile 7098) zusätzlich gelöscht. Somit ist gesichert, dass `prop` keine Daten des letzten Schleifendurchlaufs enthält.

COV-6 Dieses Resultat (siehe Abb. 5.8) ist interessant, da Coverity einen Taint-Fluss findet, der eine Kryptofunktion benutzt. Die Variable `header` enthält Informationen des Headers einer Datei. Diese Informationen fließen anschließen in die Funktion `EncryptBuffer(...)` über mehrere Funktionsaufrufe wird letztendlich (abhängig von den Einstellungen) der Verschlüsselungsalgorithmus BlowFish benutzt, um die Header-Informationen zu verschlüsseln. Ein Teil des Algorithmus BlowFish ist in Abbildung 5.8 (b) zu sehen. Die Variable `left` enthält Teilinformationen von der Variable `header`. Selbstverständlich muss ein Verschlüsselungsalgorithmus die (vertraulichen) eingehenden Daten verwenden, um die verschlüsselten Daten zu generieren. Dennoch wird die Variable `left` lediglich als Index-Zugriff auf dem Array `s` benutzt. Daher sehen wir dieses Resultat nicht als sicherheitskritisch.

```
         4. Condition LastKnownMountList.ulMountedDrives & (1 << i), taking true branch
         17. Condition LastKnownMountList.ulMountedDrives & (1 << i), taking true branch
7096                 if (LastKnownMountList.ulMountedDrives & (1 << i))
7097                 {
7098                         memset (&prop, 0, sizeof(prop));
7099                         prop.driveNo = i;
7100
         5. tainted_data_argument: Calling function DeviceIoControl taints argument prop.
         CID 11150 (#1 of 1): Untrusted value as argument (TAINTED_SCALAR)
         18. tainted_data: Passing tainted variable prop to a tainted sink.
7101                         bResult = DeviceIoControl (hDriver, TC_IOCTL_GET_VOLUME_PROPERTIES, &prop,
7102                                 sizeof (prop), &prop, sizeof (prop), &dwResult, NULL);
7103
```

Abbildung 5.7: Details zum Ergebnis COV-5.

```
1150
        CID 11110 (#1 of 1): Use of untrusted scalar value (TAINTED_SCALAR)
        7. tainted_data: Passing tainted variable header + 64 to a tainted sink. [show details]
1151        EncryptBuffer (header + HEADER_ENCRYPTED_DATA_OFFSET, HEADER_ENCRYPTED_DATA_SIZE, headerCryptoInfo);
```

(a) Die Variable `header` enthält Informationen über den Header einer Datei.

```
        3. data_index: Using tainted variable (unsigned int)(unsigned char)(left >> 24) as an index to pointer s.
369                 right ^= (((s[GETBYTE(left,3)] + s[256+GETBYTE(left,2)])
370                         ^ s[2*256+GETBYTE(left,1)]) + s[3*256+GETBYTE(left,0)])
371                         ^ p[2*i+1];
372
373                 left ^= (((s[GETBYTE(right,3)] + s[256+GETBYTE(right,2)])
374                         ^ s[2*256+GETBYTE(right,1)]) + s[3*256+GETBYTE(right,0)])
375                         ^ p[2*i+2];
376         }
377
```

(b) Später wird eine Integer-Repräsentation von `header` zum Verschlüsseln benutzt.

Abbildung 5.8: Details zum Ergebnis COV-6.

5.2.3 Cppcheck

Dieser statische Code-Scanner lieferte hauptsächlich Resultate im Bereich der Code-Qualität. Das Tool überprüft den Quelltext lediglich syntaktisch und modelliert dabei keine Methodenaufrufe. Die Ergebnisse konnten daher leicht eingestuft werden. Interessant waren vier weitere Array-Out-of-Bounds-Zugriffe, jedoch konnten diese schnell durch manuelle Analyse als Falsch-Positive deklariert werden.

5.3 Zusammenfassung

Im Rahmen dieses Arbeitspakets wurde TrueCrypt 7.1a mit Hilfe von statischer Code-Analyse näher auf Sicherheitslücken inspiziert. Der Einsatz von drei unterschiedlichen Tools hat gezeigt, dass verschiedene Warnungen gefunden wurden.

Die genauere Analyse der sicherheitsrelevanten Funde der Tools hat gezeigt, dass diese jedoch Falsch-Positive sind, die zur Laufzeit nicht auftreten können oder somit nicht problematisch sind.

Wir haben die Resultate der Scanner exportiert und liefern sie zusätzlich zu dem Bericht aus.

Zusammenfassung der Ergebnisse aus Kapitel 5

- Es ist sinnvoll ein breites Spektrum an Tools anzuwenden: Clang, Coverity und Cppcheck liefern jeweils andere Resultate.
- Mittels automatisierter Code-Analyse konnten keine Sicherheitslücken in TrueCrypt 7.1a gefunden werden, jedoch einige potenzielle Lücken ausgeschlossen werden.
- Einige Resultate konnten erst nach umfangreicher Analyse mit zusätzlichen Tools als falsch positiv eingestuft werden.
- Anhand der Resultate können Qualitätsmängel in der Implementierung leicht und effizient behoben werden.

6 Bewertung der Code-Qualität und Dokumentation

6.1 Bewertung der Code-Qualität

Die aussagekräftigste Einschätzung der Qualität von Software-Artefakten würden formale Korrektheitsbeweise liefern. Für große und komplexe Software wie TrueCrypt sind solche formalen Beweise jedoch noch immer unpraktikabel. Die gängigste Methode zur Verifikation von Software ist daher das Funktionale Testen. Die Gründlichkeit Funktionaler Tests wird dabei meist auf Basis der Testabdeckung des Quelltextes bewertet. Für TrueCrypt existieren leider kaum geeignete Testfälle, so dass eine aussagekräftige Bewertung der Softwarequalität auf dieser Basis nicht möglich ist.

Die Bewertung der Code-Qualität auf Basis der inneren Struktur und der Quelltexte einer Software ist ein indirekter aber dennoch hilfreicher Indikator für die Qualität der Software insgesamt. Insbesondere hat sich gezeigt, dass die Code-Qualität einen großen Einfluss auf den Wartungsaufwand eines Softwareprojekts hat. Gerade wegen der Einstellung des offiziellen TrueCrypt-Projekts halten wir eine Bewertung der Wartbarkeit der Codebasis für besonders wichtig. Denn Institutionen, die sich zur Weiterverwendung von TrueCrypt entscheiden, müssen im Zweifelsfall selbst für seine Wartung aufkommen.

6.1.1 Programmierrichtlinien und Best-Practices

Zunächst ist auffällig, dass die Dokumentation zu TrueCrypt keinerlei Hinweise zu Programmierrichtlinien enthält. Schriftlich fixierte Programmierrichtlinien sind in größeren Softwareprojekten sehr üblich, da Sprachen wie C und C++ ein extremes Spektrum unterschiedlicher Programmierstile erlauben. Durch die Festlegung von Programmierrichtlinien kann die Projektleitung dafür sorgen, dass die Beiträge mehrerer Programmierer zu einem gemeinsamen Quelltext mit einheitlichem Programmierstil führen. Der Grund für das Fehlen schriftlicher Programmierrichtlinien beim TrueCrypt-Projekt ist vermutlich die ursprünglich kleine Zahl von Entwicklern. Nichtsdestotrotz zeigt sich im Quelltext von TrueCrypt eine Vermischung unterschiedlicher Konventionen. Beispielsweise wird zur Darstellung zusammengesetzter Variablenbezeichner sowohl die Form mit Unterstrich (»`file_name`«, als auch mit Binnenmajuskel (»`fileName`«) parallel verwendet. Ebenso werden komplexe Datenstrukturen wechselweise durch die C++-Konstrukte `class` und `struct` definiert.

Verletzung allgemein anerkannter Regeln

Die Verwendung der `goto`-Anweisung wurde bereits in den 60ern und 70ern von Verfechtern der strukturierten Programmierung wie E. W. Dijkstra [7] kritisiert. Im Quelltext von TrueCrypt konnten wir 388 Vorkommen von `goto`-Anweisungen zählen. Die Verwendung von `goto` wird jedoch im Allgemeinen zur Umsetzung einer Ausnahmebehandlung akzeptiert, da die Sprache C kein eigenes Konstrukt hierfür kennt. Neuere Untersuchungen haben ergeben, dass Programmierer mittlerweile die `goto`-Anweisung überwiegend nur noch in sinnvoller Weise verwenden [20]. Dies trifft auch für TrueCrypt zu. Die Verwendung von `goto`-Anweisungen beschränkt sich auf Sonderfälle. Im normalen Kontrollfluss wurde `goto` nicht benutzt.

Listing 6.1: GetSystemPartitions

```
1  static bool GetSystemPartitions (byte drive) {
2      size_t partCount;
3      if (!GetActivePartition(drive))
4          return false;
5      // Find partition following the active one
6      [...]
7      return true;
8  }
```

Listing 6.2: File::Read

```
1  DWORD File::Read (byte *buffer, DWORD size) {
2      DWORD bytesRead;
3      if (Elevated) {
4          DWORD bytesRead;
5          Elevator::ReadWriteFile(false, IsDevice, Path,
6                                  buffer, FilePointerPosition,
7                                  size, &bytesRead);
8          FilePointerPosition += bytesRead;
9          return bytesRead;
10     }
11     throw_sys_if(!ReadFile(Handle, buffer, size, &bytesRead, NULL));
12     return bytesRead;
13 }
```

Mehrere return-Anweisungen

Ähnlich wie die Verwendung von `goto` werden mehrere `return`-Anweisungen innerhalb einer Funktion allgemein als schlechter Stil angesehen, wobei es auch hier Ausnahmen von der Regel gibt. In seinem einflussreichen Buch »Code Complete« [16] schreibt Steve McConnell dazu:

> »*Minimize the number of returns in each routine.* It's harder to understand a routine if, reading it at the bottom, you're unaware of the possibility that it returned somewhere above.
> *Use a return when it enhances readability.* In certain routines, once you know the answer, you want to return it to the calling routine immediately. If the routine is defined in such a way that it doesn't require any cleanup, not returning immediately means that you have to write more code.«

TrueCrypt enthält eine große Anzahl von Funktionen mit mehreren Ausstiegspunkten. Listing 6.1 zeigt ein Beispiel eines akzeptablen frühzeitigen Funktionsabbruchs (gekürzt). Die Abbruchbedingung in Zeile 3 kann direkt aus den Funktionsparametern berechnet werden. Ein sofortiger Abbruch hat keine Seiteneffekte auf den restlichen Code der Funktion.

Listing 6.2 zeigt dagegen den Fall einer fragwürdigen Verwendung einer `return`-Anweisung (Zeile 9). Die Berechnung der Abbruchbedingung umfasst hier den größten Teil des Funktionsrumpfs und kann leicht übersehen werden.

Anwendungslogik innerhalb von Header-Dateien

Eine weitere Auffälligkeit ist der Verstoß gegen die übliche Regel, dass Header-Dateien mit der Endung `.h` nur Definitionen von Schnittstellen und Datenstrukturen enthalten sollten. In der Sprache C ist diese Aufteilung zwischen Schnittstellendefinition und Implementierung üblich, um eine separates Übersetzung zu erlauben. Konstrukte wie Template-Klassen im moderneren

Listing 6.3: WasHiddenFilePresentInKeyfilePath

```
1 static bool WasHiddenFilePresentInKeyfilePath() {
2     bool r = HiddenFileWasPresentInKeyfilePath;
3     HiddenFileWasPresentInKeyfilePath = false;
4     return r;
5 }
```

C++ werden dagegen erst zum Übersetzungszeitpunkt erzeugt. Aus diesem Grund muss die Implementierung der Klassen-Templates bei der Übersetzung verfügbar sein. Auf einfache Weise kann dies erreicht werden, indem die Implementierung des Klassen-Templates in der Header-Datei enthalten ist. Sauberere Lösungen für dieses Problem existieren und sind vorzuziehen, sie sind jedoch oft Compiler-abhängig und nicht portabel.

Im Quelltext von TrueCrypt finden sich mehrere Header-Dateien, die Anwendungslogik enthalten. Ein Beispiel ist die Definition der Funktion `MountVolume` in der Datei `Core/Unix/CoreServiceProxy.h`, die über 70 Zeilen komplexen Code enthält. Allerdings handelt es sich bei `MountVolume` um ein Funktions-Template, was die Einbettung in die Header-Datei rechtfertigen kann.

Ein klarer Verstoß gegen die übliche Modularisierung sind mehrere kurze Code-Fragmente in Header-Dateien, die nicht zur Definition von Templats dienen. Listing 6.3 aus der Datei `Volume/Keyfile.h` zeigt ein typisches Beispiel einer solchen kurzen Funktionsdefinition innerhalb einer Header-Datei.

6.1.2 Komplexität des Quelltextes

Es ist intuitiv einleuchtend und durch eine große Zahl wissenschaftlicher Arbeiten untermauert, dass es einen engen Zusammenhang zwischen der Quelltextkomplexität und der Wartbarkeit von Software gibt. Zu den einfachsten und verbreitetsten Metriken zur Messung der Quelltextkomplexität zählen die Länge von Funktionen und Komplexität des Kontrollflusses. Als Maß für die Kontrollflusskomplexität wird insbesondere die zyklomatische Komplexität verwendet. Werte größer 15 sind ein Indiz dafür, dass Refaktorierung sinnvoll ist. Werte über 30 gehen oft mit fehlerhaftem Code einher [15].

Um die Wartbarkeit von TrueCrypt zu bewerten, haben wir das Analysewerkzeug Lizard[1] verwendet. Lizard misst die Länge und die zyklomatische Komplexität von C, C++ und Java-Funktionen und gibt ab verschiedenen Schwellwerten Warnungen aus. Die Ergebnisse für TrueCrypt sind bedenklich. Es gibt 170 Funktionen deren Messwert für die zyklomatische Komplexität über 15 liegt. Für 75 davon ist der Wert über 30, für 9 sogar über 100. Die 75 Funktionen mit dem höchsten Wert sind in Anhang A aufgelistet.

In Bezug auf ihre Länge stechen drei Funktionen heraus, die mehr als 1500 Zeilen lang sind. Alle drei Funktionen dienen der Umsetzung der Benutzerschnittstelle von TrueCrypt und sind daher vermutlich nicht kritisch für die Kernfunktion von TrueCrypt. Dagegen sind die beiden Funktionen *ProcessMainDeviceControlIrp* und *TCOpenVolume* mit einer Länge von jeweils über 500 Zeilen Teil des Gerätetreibers, der TrueCrypt in das Windows Betriebssytem einbindet.

6.1.3 Codeduplikate

Ein weiterer typischer Indikator für Refaktorisierungspotenzial sind Codeduplikate, identische Codesequenzen, die sich in verschiedenen Codeteilen wiederholen. Es ist leicht ersichtlich, dass solche Codewiederholungen zu einer Funktion zusammengefasst werden sollten, um die Modula-

[1] https://github.com/terryyin/lizard

Listing 6.4: Codeduplikat in Mount.c

```
1 if(!VolumeSelected(hwndDlg)) {
2     Warning("NO_VOLUME_SELECTED");
3 } else {
4     GetWindowText(GetDlgItem(hwndDlg, IDC_VOLUME),
5                   volPath,  sizeof (volPath));
6     WaitCursor();
7     if(!IsAdmin()&& IsUacSupported()&& IsVolumeDeviceHosted(volPath))
8         ...
```

rität zu verbessern und ein Auseinanderdriften der Duplikate durch nachlässige Codepflege zu Vermeiden.

Das Analysewerkzeug Duplo[2] findet Codeduplikate in C und C++ Quelltexten. In der Standardeinstellung werden Codeduplikate ab einer Länge von drei Zeilen erfasst. In der Codebasis von TrueCrypt ermittelte Duplo 7091 duplizierte Codezeilen in 1155 duplizierten Codeblöcken (siehe Anhang B).

Ein Beispiel für ein von Duplo gefundenes Codeduplikat zeigt Listing 6.4. Die Codesequenz bestehend aus sieben Zeilen ist identisch an vier Stellen in der Datei `Mount.c` enthalten.

6.1.4 Fazit

Zusammenfassend möchten wir betonen, dass die hier aufgeführten Qualitätsprobleme des Quelltextes nicht unmittelbar auf Schwachstellen oder gar Sicherheitslücken in TrueCrypt hinweisen. Sie rechtfertigen es aber, die Zuverlässigkeit von TrueCrypt etwas in Frage zu stellen. Insbesondere die hohe Komplexität des Codes bei gleichzeitig fehlenden umfassenden Testfällen lässt erwarten, dass Wartungsaufwand und -kosten sehr hoch sind. Da das Projekt nicht mehr von den ursprünglichen Entwicklern weiter gepflegt wird, muss dieser Wartungsaufwand von einzelnen Anwendern oder einer größeren Community übernommen werden.

6.2 Bewertung der Dokumentation

Die Sicherheitseigenschaften von IT-Systemen sind meistens davon abhängig, dass bestimmte Annahmen über die Umgebung oder die Verwendungsweise zutreffen. Werden diese Annahmen nicht erfüllt, oder ist sich der Anwender nicht über Einschränkungen bewusst, so können Sicherheitsmechanismen durch Konfigurations- oder Bedienungsfehler wirkungslos werden, oder sogar einen gegenteiligen Effekt erzielen. So kann ein Anwender etwa durch falsche Nutzung der »plausible deniability« Funktion von TrueCrypt erst recht Verdacht auf sich ziehen.

Auch Entwickler, die möglicherweise zu einem späteren Zeitpunkt zum Projekt hinzustoßen, und Systemadministratoren, die Software für andere beschaffen und einrichten, benötigen eine vollständige Kenntnis der zugrundeliegenden Annahmen und Einschränkungen von Sicherheitsfunktionen. Ansonsten besteht die Gefahr, dass Sicherheitseigenschaften aus Unkenntnis durch Programmier- oder Konfigurationsfehler zunichte gemacht werden.

Bei einem Sicherheitsprodukt wie TrueCrypt ist eine vollständige, aktuelle und zielgruppengerechte Dokumentation aller sicherheitsrelevanten Eigenschaften daher ein unverzichtbarer Bestandteil.

[2] http://duplo.sourceforge.net/

Verfügbare Dokumentation

Für TrueCrypt war bis zur Einstellung des Projekts ein englischsprachiges Handbuch für Endanwender (»User's Guide«) verfügbar [9], sowie die Projekt-Webseite [10] mit größten Teils identischem Inhalt. Die vor der Abkündigung verfügbaren Inhalte der Webseite wurden archiviert und können unter [24] abgerufen werden.

Der Quelltext für TrueCrypt 7.1a enthält eine Datei »Readme.txt« mit Anleitungen zur Übersetzung des Quelltextes auf den Plattformen Windows, Linux und MacOS. Ein Absatz in dieser Datei richtet sich an Software-Entwickler, die zum Quelltext beitragen möchten. Hier wird im Wesentlichen empfohlen, Kontakt mit den TrueCrypt-Entwicklern aufzunehmen.

Weitere öffentliche Dokumentation ist nicht bekannt. Insbesondere scheint es keine öffentlichen Informationen für Software-Entwickler zu geben, die sich an der Weiterentwicklung von TrueCrypt beteiligen, oder Teile davon in eigenen Projekten verwenden möchten.

Inhaltliche Qualität und Vollständigkeit der Dokumentation

Das »User's Guide« Dokument [9] beschreibt die Sicherheitseigenschaften von TrueCrypt auf konzeptioneller Ebene. Beim Leser wird jedoch bereits ein Verständnis der Grundlegenden Funktion vorausgesetzt. Was Verschlüsselung ist und wozu sie dient wird beispielsweise nicht erklärt.

Die Funktion von TrueCrypt wird sehr detailliert beschrieben, etwa die Schlüsselberechnung aus Keyfiles oder der Aufbau von Volume-Headern im Kapitel »Technical details«. Der Detailgrad der Beschreibung ermöglicht Sicherheitsexperten eine sinnvolle Bewertung des Sicherheitskonzepts von TrueCrypt. Das Dokument liefert somit auch Entwicklern einen ersten Einstiegspunkt zum Verständnis des Software-Designs.

Die Beschreibung erscheint prinzipiell ausreichend genau, um alle Kernfunktionen von TrueCrypt nachimplementieren zu können. Allerdings enthält der Text einige Fehler, die vom Entwickler des formatkompatiblen »tcplay« entdeckt wurden [12, README.md]:

> »The TrueCrypt documentation is pretty bad and does not really represent the actual on-disk format nor the encryption/decryption process.
>
> Some notable differences between actual implementation and documentation:
>
> - PBKDF using RIPEMD160 only uses 2000 iterations if the volume isn't a system volume.
> - The keyfile pool is not XOR'ed with the passphrase but modulo-256 summed.
> - Every field except the minimum version field of the volume header are in big endian.
> - Some volume header fields (creation time of volume and header) are missing in the documentation.
> - All two-way cipher cascades are the wrong way round in the documentation, but all three-way cipher cascades are correct.«

Für Endanwender bietet der »User's Guide« keine ausreichende Information. Beispielsweise wird die Wahl der zu verwendenden Hash-Funktionen und Verschlüsselungsalgorithmen dem Anwender überlassen. Es gibt jedoch keinerlei Empfehlung, welche Auswahl oder Auswahlkriterien sinnvoll sind.

Eine Auflistung von zugrunde liegenden Annahmen und Einschränkungen findet sich in den Kapiteln »Security model« und »Security requirements and precautions«. Viele weitere Hinweise zu möglichen Bedrohungen und unsicheren Anwendungsweisen werden über alle Kapitel im jeweiligen Kontext erwähnt. Um alle Hinweise zur sicheren Anwendung berücksichtigen zu können, ist es daher nötig, das gesamte Dokument zu lesen.

Kommentierung im Quelltext

Der Quelltext von TrueCrypt ist nur vereinzelt mit Kommentaren versehen. Hierbei handelt es sich in den meisten Fällen um stichwortartige Anmerkungen am Zeilenende, beispielsweise zur Beschreibung der Funktion einer Variablen.

An ca. 40 Stellen finden sich mit »WARNING« oder »IMPORTANT« gekennzeichnete Warnhinweise für Entwickler, die auf mögliche Fehlerquellen durch Code-Änderungen aufmerksam machen, etwa:

> »IMPORTANT: Modifying this value can introduce incompatibility with previous versions«
>
> »WARNING: ADD ANY NEW CODES AT THE END (DO NOT INSERT THEM BETWEEN EXISTING). DO *NOT* DELETE ANY«

Einige wenige Kommentare sind ausführlicher und liefern auch Begründungen für Warnungen. Diese Art von Information ist sehr wichtig für spätere Entwicklungsarbeiten.

Insgesamt bezieht sich die Kommentierung auf lokale Implementierungsdetails. Eine Beschreibung von übergreifenden Design- oder Architekturaspekten gibt es nicht. Selbst zusammenfassende Beschreibungen der in den einzelnen Quelltextdateien implementierten Funktionen sind nicht vorhanden. Eine Ausnahme hiervon bildet nur solche Quelltextteile, die aus dritten Quellen übernommen wurden.

6.3 Zusammenfassung

Zusammenfassung der Ergebnisse aus Kapitel 6

Bewertung der Code-Qualität:

- Fehlen umfassender Testfälle.
- Keine schriftlich festgehaltenen Programmierrichtlinien. Der Stil der Quelltexte ist inkonsistent. Missachtung von Best-Practices.
- Schlechte Wartbarkeit und damit hohe Wartungskosten (teilweise hohe Komplexität, Codeduplikate)

Bewertung der Dokumentation:

- Die Dokumentation beschränkt sich im Wesentlichen auf den »Users' Guide«. Dokumentation für Entwickler gibt es nicht.
- Der »Users' Guide« enthält viele Informationen zur Funktionsweise und Verwendung von TrueCrypt, ist jedoch schlecht strukturiert und enthält Fehler.
- Quelltexte sind sporadisch mit knappen Kommentaren versehen.

7 Konzeptionelle Bewertung der Architektur

7.1 Einleitung

Dieses Kapitel widmet sich der Bedrohungsanalyse und einer darauf aufbauenden konzeptionellen Bewertung der TrueCrypt-Architektur. Die Grundlage dieser Betrachtungen bilden eine Reihe von Annahmen, die den vorgesehenen Anwendungskontext und sich daraus ergebende Anforderungen umfassen. Dies ist notwendig, da die Bewertung von Angriffsszenarien und der Angemessenheit einer Systemarchitektur nie absolut erfolgen kann, sondern stets an konkrete, situationsspezifische Faktoren gebunden ist. Im Folgenden werden wir zunächst diese Annahmen dokumentieren und eine Menge von angestrebten Schutzzielen festlegen. Anschließend stellen wir verschiedene potentielle Angriffsstrategien vor und bewerten deren Relevanz mit Hinblick auf die TrueCrypt-Architektur.

7.2 Kontext und Anwendungsfälle

Der Anwendungskontext von TrueCrypt liegt im privaten, behördlichen, oder im Unternehmenseinsatz, um sensible Daten vor unautorisiertem Fremdzugriff zu schützen. Diese drei Einsatzfelder sind im Rahmen dieser Betrachtung als weitgehend ähnlich anzusehen, bei Bedarf wird der Report jedoch an gegebener Stelle auf etwaige Besonderheiten eingehen. Wir nehmen an, dass durch TrueCrypt geschützte Systeme im behördlichen und Unternehmenseinsatz zumindest logisch in ein Computernetzwerk eingebunden sind und durch Administratoren betreut werden. Darüber hinaus gehen wir davon aus, dass zumindest stationäre Systeme (PCs) in der Regel in Räumlichkeiten des jeweiligen Arbeitgebers betrieben werden.

Die für die weiteren Betrachtungen relevanten Anwendungsfälle für TrueCrypt leiten wir aus der Beschreibung der Funktionalität ab, die wir dem TrueCrypt-Handbuch entnehmen. Daraus ergeben sich die folgenden drei Anwendungsfälle:

ID: UC1

Titel: Informationsschutz durch Systemverschlüsselung (»System encryption«)

Beschreibung: Der Benutzer verwendet einen Computer, um sensible Daten zu verarbeiten. Der Benutzer schaltet den Computer bei Nichtbenutzung stets aus. Der Benutzer verwendet TrueCrypt, um die sensiblen Daten auf dem permanenten Speicher (i.d.R. Festplatte) vor unautorisiertem Fremdzugriff zu schützen. TrueCrypt fordert den Benutzer beim Systemstart dazu auf, sich zu authentifizieren. TrueCrypt verweigert den Start des Betriebssystems und somit den Zugriff auf die sensiblen Daten, wenn die Authentifizierung scheitert und gestattet den Zugriff nur, wenn die Authentifizierung erfolgreich war. Nach erfolgreicher Authentifizierung kann der Benutzer in gewohnter Weise mit den Daten auf dem Computer arbeiten. Die Funktionalität zur Systemverschlüsselung wird nur für bestimmte Windows-Versionen angeboten.

Varianten:

a) **Verschlüsselung der Systemplatte**
TrueCrypt verschlüsselt die gesamte Festplatte, auf dem sich die Systempartition befindet. TrueCrypt fragt den Benutzer unmittelbar beim Start des Computers nach einem Passwort um sich zu authentifizieren.

b) **Verschlüsselung der Systempartition**
TrueCrypt verschlüsselt lediglich die Systempartition und fragt den Benutzer unmittelbar beim Start des Computers nach einem Passwort um sich zu authentifizieren. Andere Partitionen als die Systempartition können unverschlüsselt bleiben.

ID: UC2

Titel: Informationsschutz durch verschlüsselte Volumes

Beschreibung: Der Benutzer verarbeitet sensible Daten auf einem Computer, oder möchte eine Kopie der Daten auf ein externes Persistenzmedium übertragen (z. B. für eine Sicherheitskopie, oder um die Daten für den dienstlichen Gebrauch mit Projektpartnern zu teilen). Der Benutzer möchte die sensiblen Daten auf dem Transportmedium und/oder dem Persistenzmedium vor unautorisiertem Fremdzugriff schützen. Hierfür verwendet der Benutzer TrueCrypt, um die zu schützenden Daten in einem verschlüsselten Volume abzulegen. Anstelle der unverschlüsselten lesbaren Daten, überträgt/speichert der Benutzer das Volume. Damit der Benutzer die verschlüsselten Inhalte des Volumes wieder lesen, schreiben und bearbeiten kann, legt TrueCrypt nach erfolgreicher Authentifizierung des Nutzers ein virtuelles Laufwerk an, das die Inhalte des Volumes lesbar darstellt (»mounten«). Wie für reguläre Laufwerke üblich, kann der Benutzer in dem virtuellen TrueCrypt-Laufwerk Dateien anlegen, löschen und bearbeiten. Die Änderungen werden automatisch von TrueCrypt in dem Volume abgebildet.

Varianten:

a) **Dateibasiert**
TrueCrypt bietet Funktionalität an, ein verschlüsseltes Volume in einer Datei abzulegen (auch »file-hosted (container)« genannt). Bei diesem Container handelt es sich um eine reguläre Datei, die ebenso wie andere Dateien gespeichert, gelöscht, und anderweitig verarbeitet werden kann. Im Kontext dieses Anwendungsfalls könnte der Benutzer diese Datei also beispielsweise auch als geschützte Sicherheitskopie auf einer externen Festplatte ablegen, oder auch per E-Mail an Projektpartner versenden.

b) **Partitions-/Gerätebasiert**
Neben den dateibasierten Volumes bietet TrueCrypt auch sogenannte partitions-/gerätebasierte Volumes (»partition/device-hosted (non-system)«) an. Hierbei wird das Volume direkt in einer Partition eines Datenträgers gespeichert, oder es nimmt den gesamten Speicher eines Datenträgers ein. Soll das Volume innerhalb einer Partition eines Datenträgers abgelegt werden, darf es sich hierbei jedoch nicht um die Systempartition des laufenden Systems handeln.

ID: UC3

Titel: Geheimhaltung und Informationsschutz durch versteckte Volumes

Beschreibung: Der Benutzer möchte sensible Daten, die er auf einem Computer verarbeitet, in einer solchen Weise ablegen, dass sie einerseits vor unautorisiertem Fremdzugriff geschützt sind, und andererseits deren Existenz ohne entsprechende Vorkenntnisse nicht

herleitbar ist. Der Benutzer kann daher zur Herausgabe der Daten nicht gezwungen werden, da er glaubhaft abstreiten kann, dass es sie überhaupt gibt. Für diesen Zweck verwendet der Nutzer TrueCrypt, um ein verstecktes Volume innerhalb eines anderen TrueCrypt-Volumes zu erstellen. Der Zugriff auf Daten innerhalb des versteckten Volumes erfolgt aus Benutzersicht ebenso wie der Zugriff auf Daten innerhalb eines regulären Volumes. Der Benutzer muss sich hierfür zunächst bei TrueCrypt authentifizieren und erhält dann Zugriff auf ein virtuelles Laufwerk, welches die versteckten Daten lesbar darstellt.

Varianten:

a) **Verstecktes Volume**
Mit TrueCrypt kann man innerhalb eines anderen TrueCrypt-Volumes ein verstecktes Volume (»hidden volume«) anlegen, auf das genauso wie auf reguläre Volumes zugegriffen werden kann. Als »äußeres« Volume eignen sich sowohl dateibasierte, wie auch partitions-/gerätebasierte Volumes.

b) **Verstecktes Betriebssystem**
TrueCrypt kann auch dazu benutzt werden, eine gesamte Systempartition zu verstecken (»hidden operating system«). Hierfür werden insgesamt mindestens drei TrueCrypt-Volumes benötigt. Das erste TrueCrypt-Volume beinhaltet eine vollständige Systempartition, deren Existenz nicht glaubhaft abgestritten werden kann (das »Locksystem«, oder auch »decoy operating system«). Der Benutzer kann also zur Herausgabe des Passworts gezwungen werden und es sollte keine wirklich sensiblen Daten beinhalten. Daneben gibt es ein zweites TrueCrypt-Volume, dessen Existenz ebenfalls nicht glaubhaft abstreitbar ist. Dieses zweite Volume dient jedoch als »äußeres« Volume für das dritte, versteckte Volume, welches das zu versteckende Betriebssystem beinhaltet. Beim Start des Computers fordert TrueCrypt zur Eingabe eines Passworts auf. Wenn der Benutzer das Passwort des Locksystems eingibt, startet lediglich das Betriebssystem der ersten Partition. Gibt der Benutzer jedoch das Passwort des versteckten Betriebssystems ein, dann startet dieses. Dies ist eine Form der Systemverschlüsselung und wird ebenfalls nur für bestimmte Windows-Versionen angeboten.

7.3 Schutzziele und Anforderungen

Im Folgenden beschreiben wir die Schutzziele, die wir im Kontext der oben skizzierten Anwendungsfälle als relevant betrachten. Sie bilden die Grundlage für eine Menge von Anforderungen (R1-R7), die für die Bewertung der Architektur herangezogen werden können.

Primäres Schutzziel

Ausgehend von den Einsatzszenarien und Verwendungszweck von TrueCrypt ist das primäre Schutzziel der Schutz der vertraulichen Daten in einem TrueCrypt-Volume oder TrueCrypt-Container vor dem Zugriff oder Kenntnisnahme durch Unberechtigte (Vertraulichkeit der Daten im Container oder Volume).

R1: Der Schutz der Vertraulichkeit von Daten muss durch den Einsatz geeigneter kryptographischer Verfahren erfolgen. Der alleinige Einsatz eines simplen Authentifizierungsmechanismus zur Zugriffsbeschränkung, wie zum Beispiel die typischerweise von Betriebssystemen für die Benutzeranmeldung verwendete Passwortabfrage, ist völlig unzureichend, da dies mit einfachsten Mitteln umgangen werden kann.

R2: Daten, deren Vertraulichkeit mit TrueCrypt geschützt werden soll, dürfen nicht von TrueCrypt im Klartext persistiert oder an Dritte übertragen werden. Insbesondere geht damit

einher, dass die Anwendung der kryptographischen Verfahren zum Schutz der Daten lokal auf dem Computer des Benutzers erfolgen muss, und beispielsweise nicht etwa zu einem externen Dienst ausgelagert werden darf.

Sekundäre Schutzziele

Neben der Vertraulichkeit der gespeicherten Daten gibt es weitere Schutzziele, die TrueCrypt im begrenzten Maße zugeordnet werden können. So ist ein eingeschränkter Schutz der Integrität der TrueCrypt-verschlüsselten Daten erwünscht: Es sollten keine Veränderungen des Chiffrats möglich sein, die gezielt bestimmte Klartexte bei der Entschlüsselung erzwingen könnten. Für das Erkennen von Veränderungen finden sich hingegen in der Beschreibung von TrueCrypt keine Anhaltspunkte dafür, dass dies ein relevantes Schutzziel sei.

Eine Besonderheit ist der Modus der »versteckten Datenspeicher« oder des »versteckten Betriebssystems«. Bei der Nutzung dieser Betriebsmodi ist das Schutzziel der Abstreitbarkeit der Existenz von verschlüsselten Daten von Relevanz. Dies ist eine Besonderheit von TrueCrypt gegenüber anderen Verschlüsselungssystemen.

R3: Daten, deren Vertraulichkeit durch TrueCrypt geschützt werden soll, sollten integritätsgesichert sein. Ein Benutzer sollte also erkennen können, ob geschützte Daten seit der letzten autorisierten Bearbeitung unbefugt verändert wurden.

R4: TrueCrypt selbst sollte integritätsgesichert sein. Der autorisierte Benutzer sollte bei der Verwendung von TrueCrypt erkennen können, ob es seit der letzten Verwendung unbefugt verändert wurde.

R5: TrueCrypt darf auf dem Persistenzmedium keinen Hinweis darauf speichern, ob ein TrueCrypt-Volume ein weiteres, verstecktes Volume beinhaltet oder nicht. Dies gilt sowohl für ein verstecktes Volume, wie auch für ein verstecktes Betriebssystem.

Abgeleitete Schutzziele

Aus den Eigenschaften einer Verschlüsselungssoftware ergibt sich der Zusammenhang, dass die Vertraulichkeit der Daten in einem TrueCrypt-Volume oder TrueCrypt-Container direkt von der Vertraulichkeit der zur Entschlüsselung benötigten Schlüssel, Schlüsseldateien oder Passwörter abhängt. Somit ist die Vertraulichkeit dieser Metadaten ebenfalls ein Schutzziel von TrueCrypt, zumindest dort wo TrueCrypt den Schutz dieser Daten beeinflussen kann.

R6: TrueCrypt darf zu keinem Zeitpunkt geheime Schlüssel oder Passwörter im Klartext auf einem Persistenzmedium ablegen.

R7: TrueCrypt darf zu keinem Zeitpunkt geheime Schlüssel oder Passwörter an Dritte übertragen.

7.4 Angriffsstrategien

7.4.1 Vorbetrachtungen

TrueCrypt-geschützte IT-Systeme sehen sich, bezogen auf den durch TrueCrypt gelieferten Schutz, im Wesentlichen durch potentielle Angreifer bedroht, welche Kenntnis vertraulicher im System gespeicherter Informationen erlangen möchten und zu diesem Zweck versuchen Daten aus diesem IT-System zu extrahieren. Prinzipiell sind als Bedrohung auch Angreifer denkbar, welche eine Motivation haben, Daten in dem System zu verändern, hinzuzufügen, oder zu löschen. Diese haben aber eine geringere Bedeutung, siehe Abschnitt 7.3.

Da ein Angreifer in irgendeiner Form Zugang zu dem TrueCrypt-geschützten IT-System oder Datenträger benötigt, lassen sich zu Beginn der Analyse generell zwei verschiedenen Ausgangssituationen unterscheiden (vgl. Tabelle 7.1). Ein Angreifer kann für einen Angriff sowohl

physischen Zugang zu einem TrueCrypt-geschützten IT-System erlangen, als auch

logischen Zugang zu einem TrueCrypt-geschützten IT-System.

Der Zugang kann vorübergehend, aber auch permanent sein. Physischer Zugang ist möglich in Situationen, in denen sich ein Angreifer unberechtigten Zugang zu Räumlichkeiten, oder Orten verschafft in denen ein geschütztes IT-System betrieben oder gelagert wird (Büros, Serverräume), oder ein geschütztes IT-System in einer Umgebung exponiert wird, zu der ein Angreifer berechtigten Zugang hat (öffentliche Räume, nicht-öffentliche Räume mit Publikumsverkehr). Logischer Zugang ist dann möglich, wenn ein Angreifer die Datenverarbeitungsprozesse in dem IT-System beeinflussen kann, ohne physisch auf das System zugreifen zu müssen, z. B. durch unberechtigte Nutzung der Fernzugänge legitimer Nutzer oder mithilfe eingeschleuster Schadsoftware. Physischer Zugang erlaubt häufig auch den logischen Zugang, umgekehrt gilt dies zumeist nicht.

Eine weitere wichtige Unterscheidung ist die nach der Möglichkeit des Angreifers nur einmalig oder wiederholt Zugang zu dem IT-System zu erlangen (vgl. Tabelle 7.1). Entsprechend unterscheidet man den

Einzelzugriff bei welchem der Angreifer einmalig Zugriff zu dem geschützten IT-System oder dem Datenträger mit den geschützten Daten erhält, und den

Mehrfachzugriff bei welchem der Angreifer mehrfach Zugang zu dem IT-System oder dem Datenträger erhält, wobei die legitimen Anwender das IT-System oder den Datenträger zwischen den Angriffen regulär nutzen.

Einzelzugriffe stehen zumeist nicht im direkten Zusammenhang mit gezielten Angriffen speziell auf die Vertraulichkeit bestimmter im IT-System gespeicherter Daten. Die Bedrohung der Vertraulichkeit ist vielmehr oft ein Nebeneffekt, wohingegen der Einzelzugriff auf die Aneignung physischer Werte, speziell der Hardware des IT-Systems abzielt. Das häufigste Beispiel ist der Zugriff auf Hardware im öffentlichen Raum oder in Bereichen mit Publikumsverkehr, oder der Einbruch in geschlossene Räumlichkeiten mit dem Ziel kostspielige Hardware wie Laptops oder Server zu stehlen.

Im Gegensatz dazu ist damit zu rechnen, dass gezielte Angriffe häufig Möglichkeiten des Mehrfachzugriffs auf das IT-System ausnutzen. Diese Möglichkeit besitzen typischerweise Innentäter (reguläre Angestellte und Dienstpersonal) durch berechtigten Zugang zu zeitweise unbeobachtet betriebenen oder aufbewahrten IT-Systemen oder Datenträgern. Bei verhältnismäßig schlecht gesicherten Räumlichkeiten (z.B. Wohnungen) besteht ebenfalls ein Potential für Mehrfachzugriffe. Die Möglichkeit des Mehrfachzugriffs schließt immer die Möglichkeit des Einzelzugriffs ein; logischer Zugang ermöglicht in vielen Fällen den mehrfachen, logischen Zugriff auf das IT-System.

7.4.2 Überblick über Angriffe mit physischen Zugriff

Entlang der Schutzziele aus 7.3 und unter Berücksichtigung der Anwendungsfälle aus 7.2 haben insbesondere jene Angriffe eine herausragende Bedeutung, bei denen ein Angreifer zumindest punktuell für einen kurzen Zeitraum physischen Zugang zu dem TrueCrypt-geschützten System hat oder hatte. Es bestehen unter der Voraussetzung eines solchen Zugriffs mehrere Möglichkeiten für Angriffe auf die Vertraulichkeit der TrueCrypt-geschützten Daten in solchen Szenarien.

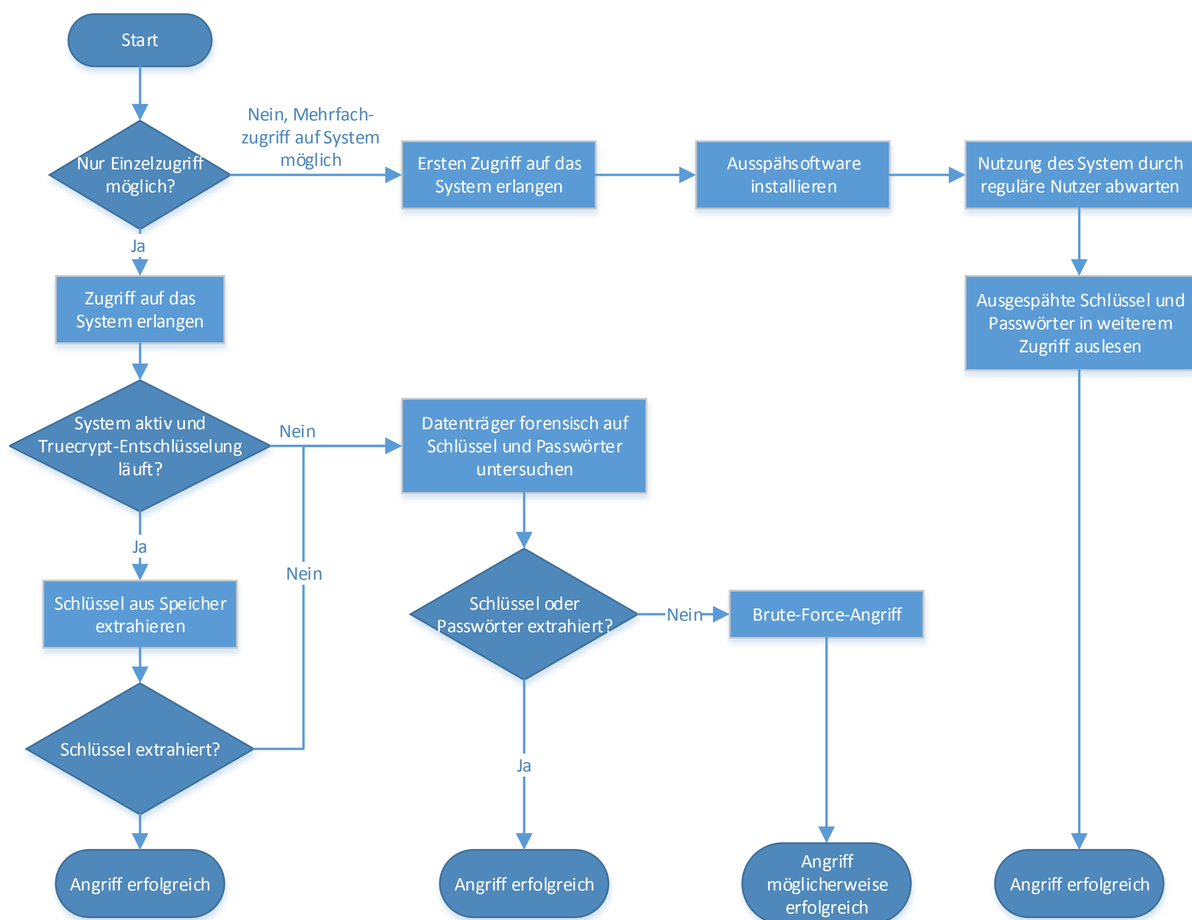

Abbildung 7.1: Optionen für Angriffe am TrueCrypt bei physischem Zugang

Angriffsszenario	Beschreibung	Beispiele
Einzelzugriff	Angreifer erlangt einmalig Zugang zum zum Datenträger oder geschützten IT-System	Diebstahl
Mehrfachzugriff	Angreifer erlangt mehrfach Zugang zum zum Datenträger oder geschützten IT-System	»Dienstmädchen-Attacke«, gezielte Angriffe auf Einzelpersonen
physischer Zugang	Angreifer erlangt physisch Zugang zum Datenträger oder geschützten IT-System	Diebstahl, Zugang zu Räumlichkeiten mit IT-System
logischer Zugang	Angreifer erlangt logisch Zugang zum geschützten IT-System	Malware, Umgehen von logischen Zugriffskontrollen

Tabelle 7.1: Arten von Angriffsszenarien und Beispiele

Abbildung 7.1 zeigt ein grobes Vorgehensschema, in welches sich prinzipiell alle im Folgenden im Detail erläuterten Angriffsstrategien einordnen lassen.

Wie in Abbildung 7.1 gezeigt, ist immer zunächst entscheidend, ob prinzipiell nur ein Einzelzugriff, oder aber auch Mehrfachzugriffe möglich sind. Ist der Mehrfachzugriff möglich, ist je nach technischen Fähigkeiten des Angreifers das Ausspähen von Passwörtern und Schlüsseln das Mittel der Wahl, da diese Angriffsstrategie eine hohe Erfolgswahrscheinlichkeit verspricht. Gleichzeitig sind auch die für Einzelzugriffe geeigneten Angriffswege möglich. Ist nur der Einzelzugriff möglich, dann hängen Angriffswege und damit verbundene Erfolgswahrscheinlichkeiten von dem Zustand des kompromittierten Systems ab. Ist dieses aktiv, und läuft die TrueCrypt-Entschlüsselung, dann kann ein Angreifer versuchen aus dem flüchtigen Speicher verfügbare Verschlüsselungsschlüssel auszulesen. Dies erfordert je nach angegriffenem System erhöhte bis hohe technische Expertise; ist aber prinzipiell erfolgversprechend. Ist das System inaktiv, oder erlangt der Angreifer Zugang ausschließlich zum Datenträger, dann bleibt nur die forensische Suche nach Passwörtern und Schlüsseln, oder Brute-Force-Angriffe.

7.4.3 Angriffsstrategien im Detail

Brute-Force-Angriff

Bei einem Brute-Force-Angriff versucht ein Angreifer ein TrueCrypt-Passwort oder einen kryptografischen Schlüssel zu erraten, indem er versucht den TrueCrypt-Header mit zufällig gewählten Schlüsseln oder Passwörtern zu entschlüsseln. Diese Strategie kann zum Erfolg führen, wenn der Nutzer des Systems seine Passwörter oder Schlüssel aus einem kleinen Raum aller möglichen Passwörter oder Schlüssel ausgewählt hat. Ist dies nicht der Fall, dann ist ein Erfolg für den Angreifer sehr unwahrscheinlich.

Erschwert werden Brute-Force-Angriffe durch »Key stretching« in der Schlüsselableitungsfunktion, also durch das schleifenweise Anwenden einer Hashfunktion auf einen Ausgangsschlüssel, bzw. die abgeleiteten Zwischenschlüssel . Ein Angreifer muss beim Test von möglichen Passwörtern diese Iterationen ebenfalls jeweils für jedes Passwort durchlaufen, einschließlich des damit verbundenen Datenverarbeitungsaufwandes.

Brute-Force-Angriffe kommen üblicherweise dann in Frage, wenn der Angreifer durch

- Diebstahl als dauerhafte Inbesitznahme des IT-Systems, oder über

- Inbesitznahme des Datenträgers mit verschlüsselten Daten (z.B. auch bei der Ausmusterung von Datenträgern), oder durch
- Kopieren von verschlüsselten Daten

in den Besitz der verschlüsselten Daten (Chiffrat) gelangt, er aber den Schlüssel auf keinen anderen Weg mehr aus dem System oder Datenträger extrahieren kann, keine weitere Interaktionsmöglichkeit mit dem ursprünglichen Besitzer des Systems mehr besteht (z.B. die Durchführung von Social Engineering), oder die Inbesitznahme des Systems oder Datenträgers durch das Opfer bereits bemerkt wurde. Brute-Force-Angriffe sind damit eine typische Strategie wenn nur ein physischer Einzelzugriff auf das IT-System oder den Datenträger möglich ist.

Forensische Analyse unverschlüsselter Daten im IT-System

Anstatt zu versuchen, die TrueCrypt-Verschlüsselung durch Testen potentieller Schlüssel oder Passwörter zu brechen, kann ein Angreifer auch versuchen in dem IT-System gespeicherte unverschlüsselte Daten forensisch auf Hinweise für die eingesetzten Schlüssel oder Passwörter zu untersuchen. Als potentielle Ziele für solche forensischen Analysen eignen sich besonders die persistenten Datenspeicher, aber auch, sofern ein Angreifer Zugriff auf ein laufendes System erlangt, flüchtige Speicher.

Bei der forensischen Analyse persistenter Datenspeicher steht vor allem die Untersuchung von Dateien im Vordergrund, in welchem das Betriebssystem oder TrueCrypt selbst Schlüssel oder Passwörter zur Zwischenspeicherung oder Protokollierung abgelegt haben könnte. Beispiele sind Swap-Dateien, temporäre Dateien, Log-Dateien oder Eingabepuffer. Die Analyse muss sich dabei nicht nur auf die von bestehenden Dateisystemen referenzierten Dateien beziehen. Sie kann auch Speicherbereiche auf dem Datenträger umfassen, welche nicht mehr referenziert werden, aber noch nicht anderweitig überschrieben wurden. Verschlüsselt TrueCrypt nicht nur Datenpartitionen, sondern auch die Betriebssystem-Partition, dann ist die Erfolgswahrscheinlichkeit für einen solchen Angriff gering.

Sofern das IT-System beim Zugriff des Angreifers noch in einem aktiven Zustand ist, sind prinzipiell auch forensische Analysen der flüchtigen Speicher möglich, wie z. B. sog. Cold-Boot-Angriffe, bei denen ein Angreifer versucht Hauptspeicherinhalte mit eigener Software auszulesen, entweder durch Entnahme von zu diesem Zweck gekühlten Speicherbausteinen oder Booten eines eigenen Systems. Unter bestimmten Umständen ist auch ein Angriff über externe Schnittstellen z. B. per DMA möglich.

Ausspähen von Passwörtern und Schlüsseln

Hat ein Angreifer die Möglichkeit des Mehrfachzugriffs auf das IT-System zwischen Nutzungen der legitimen Anwender, so ist das Ausspähen von Passwörtern und Schlüsseln erste Wahl für einen Angriff. Wie der Angriff konkret durchgeführt werden könnte, hängt von den technischen Fähigkeiten des Angreifers, der erwünschten Unauffälligkeit des Angriffs, und der zur Verfügung stehenden Zeit beim Zugriff auf das IT-System ab. Insgesamt sind die denkbaren Angriffswege sehr vielfältig, so dass an dieser Stelle nur einige ausgewählte skizziert werden können:

Eine relativ schlichte, aber durchaus wirksame Maßnahme ist das Anbringen eines Hardware-Keyloggers, welcher zwischen Tastatur und Rechner gesteckt jegliche Tastatureingaben, inklusive TrueCrypt-Passwörter mitschneidet. Dies ist jedoch mit wenig Aufwand nur bei Desktop-Computern oder an Laptop-Dockingstationen möglich. Ist die Betriebssystem-Partition nicht verschlüsselt, dann kann ein Angreifer anstelle des Hardware-Keyloggers auch einen Software-Keylogger installieren. Eventuell müsste der Angreifer dazu aber den Datenträger zeitweise ausbauen und in einem anderen System die Installation der Ausspähsoftware vornehmen, was mehr Zeit in Anspruch nehmen würde. Technisch mehr Wissen setzt eine Softwaremanipulation

bei verschlüsseltem Betriebssystem voraus, da hier Veränderungen am Bootloader erforderlich sind. Ist beim Angreifer entsprechende technische Expertise vorhanden, kann er die Manipulation selbst allerdings innerhalb weniger Minuten durchführen, wie Türpe et al. am Beispiel von Microsoft BitLocker bereits demonstrierten [23].

Etwas auffälliger, aber prinzipiell genauso möglich ist der Austausch des Rechners durch einen identisch aussehenden. Das »Duplikat« würde nur den Trojaner ausführen und danach den Bootvorgang mit einem realistisch wirkenden Fehler abbrechen, z.B. einem vermeintlichen Datenträgerfehler, welcher das System unbrauchbar gemacht habe. Da der Datenträger komplett verschlüsselt ist, ist es praktisch zunächst schwierig herauszufinden, dass es sich um ein anderes Gerät handelt. Eine genauere Untersuchung von Gebrauchsspuren, Seriennummern, oder eine versuchte Entschlüsselung mit einer Sicherung des Masterschlüssels könnte zu einer Aufdeckung des Angriffs führen. Allerdings könnte der Angreifer dann bereits im Besitz des ausgespähten Passworts sein.

Denkbar wäre auch der Versuch durch von TrueCrypt verarbeitete Daten auf dem Datenträger direkt Schadcode einzuspielen und zur Ausführung zu bringen, z.B. durch einen Pufferüberlauf. Dazu könnte ein Angreifer entweder Header-Daten oder Chiffrat-Daten verändern. Allerdings setzt ein solcher Angriff im Unterschied zu den anderen Varianten eine Schwachstelle in TrueCrypt voraus. Abgesehen von der eventuellen Einfachheit des Angriffs hat ein Angreifer aber keine Vorteile gegenüber den anderen beschriebenen Spähangriffen.

Ein Angreifer muss ausgespähte Schlüssel und Passwörter nicht zwingend durch einen zweiten physischen Zugriff auf den Rechner auslesen. Prinzipiell ist es auch möglich, drahtgebundene oder drahtlose Netzwerk- oder Breitbandverbindungen zu nutzen. Hat ein Angreifer den verschlüsselten Datenträger vorher vor Ort bereits kopiert, so kann er nach Empfang ausgespähter Schlüssel oder Passwörter diesen Datenträger unmittelbar entschlüsseln.

Zusammenfassend ist festzuhalten, dass das »Repertoire« an Spähangriffen nahezu unerschöpflich ist. Insofern ein Angreifer mehrfach auf das TrueCrypt-verschlüsselte System zugreifen kann - wobei unter Umständen nur der erste Zugriff physischer Natur sein muss - sind eine Vielzahl von verschiedenen Spähangriffen möglich. Der Angreifer kann sich nahezu beliebig verschiedenen Faktoren anpassen: Risikoakzeptanz bezüglich Entdeckens des Angriffs, logische Zugänglichkeit des Systems durch Netzwerkschnittstellen, zur Verfügung stehende Zeit bei physischem Zugriff, Leichtgläubigkeit des legitimen Nutzers. Damit ist es plausibel anzunehmen, dass bei einem technisch versierten Angreifer mit entsprechender krimineller Energie ein Spähangriff mit hoher Wahrscheinlichkeit erfolgreich ist.

Seitenkanalangriffe

Seitenkanalangriffe auf TrueCrypt könnten dann eine Rolle spielen, wenn ein Angreifer physischen oder logischen Zugriff auf ein IT-System erlangt, in welchem eine Entschlüsselung eines TrueCrypt-verschlüsselten Datenvolumes bereits läuft. In einer solchen Situation ist es denkbar, dass ein Angreifer physikalische Parameter des Systems aufzeichnet, z. B. die Leistungsaufnahme bestimmter Komponenten, um damit möglicherweise Rückschlüsse auf bei der Datenverarbeitung verwendete Schlüssel ziehen zu können. Ist ein logischer Zugriff möglich dann können andere Informationen diesen Zweck erfüllen, wie z.B. Programmlaufzeiten oder Speichernutzung.

Insgesamt ist jedoch zu berücksichtigen, das Seitenkanalangriffe eine hohe technische Expertise und materielle Ausstattung eines Angreifers voraussetzen, zumindest dann, wenn dieser über das Beobachten physikalischer Parameter angreifen möchte. Dennoch sind Seitenkanalangriffe ein realistisches Szenario, wenngleich ein Angreifer allein schon aus ökonomischen Gründen - insofern es die Situation zulässt - andere Angriffe bevorzugen wird.

Angriffe bei logischem Zugang zum IT-System

Selbst wenn ein Angreifer ausschließlich begrenzten logischen Zugang zu dem IT-System hat, in welchem eine TrueCrypt-Entschlüsselung abläuft, dann wären trotzdem Angriffsszenarien denkbar, die Sicherheitsschwächen in TrueCrypt ausnutzen könnten. Diese könnten dann greifen, wenn der logische Zugang durch Zugriffsregeln auf Datenträger, Partitionen oder Teile des Dateisystems in dem IT-System beschränkt ist, so dass ein Angreifer nicht alle in dem verschlüsselten TrueCrypt-Volume oder TrueCrypt-Container lesen darf. Hier könnte eine Schwachstelle im TrueCrypt-Treiber eine Rechteeskalation ermöglichen, über die der Angreifer weitreichende zusätzliche Zugriffsrechte erlangen könnte; ein Beispiel für eine solche Schwachstelle in einem FAT-Treiber findet sich unter [17]; erst kürzlich wurden zwei ähnliche Schwachstellen speziell in Truecrypt gefunden [18, 19]. Im Rahmen der Rechteeskalation könnte dann auch der Zugang zu verschlüsselten, aber durch Zugriffskontrollen gesicherte Daten möglich sein.

Ein solcher Angriff ist jedoch ein Randszenario: erstens kann prinzipiell jeder beliebige andere Treiber, oder auch der Betriebssystemkern selbst, eine Schwachstelle besitzen, welche eine Rechteeskalation erlauben könnte. TrueCrypt ist also generell als eine im Betriebssystemkern laufende Software von diesem Sicherheitsrisiko betroffen, und nicht in seiner Eigenschaft als Verschlüsselungssoftware. Zweitens, muss der Angreifer bei diesem Angriffsszenario bereits in der Lage sein, mit dem TrueCrypt-Treiber Daten auszutauschen, um die Schwachstelle ausnutzen zu können. Dies kann im Grunde nur der Fall sein, wenn er bereits mit den administrativen Werkzeugen von TrueCrypt interagieren kann, oder wenn er bereits (eingeschränkten) Zugriff auf einen Ver- oder Entschlüsselungsprozess von TrueCrypt hat. In beiden Fällen besitzt er dann aber bereits weitreichende Rechte in dem angegriffenen System, z. B. einen Nutzerzugang mit bestimmten Programmausführungs- und Datenträger-Leserechten.

Indirekte Angriffe ohne physischen Zugang zum System

Neben dem Ziel die eigenen Rechte zu erweitern, wenn ein Angreifer bereits direkten, logischen Zugang zum IT-System hat, ist es auch denkbar, dass ein Angreifer dem rechtmäßigen Nutzer mit Schadcode kompromittierte Daten zur Verarbeitung in TrueCrypt »unterschiebt«. Hierfür kommen z. B. Schlüsseldateien, TrueCrypt-Container, oder Klartextdaten bzw. -dateien zur späteren Verschlüsselung durch TrueCrypt in Frage. Ein Angreifer könnte gezielt eine Schwäche in der Prüfung und Verarbeitung von Eingabedaten in TrueCrypt ausnutzen, z. B. einen Pufferüberlauf mit Schadcode-Ausführung provozieren.

Durch unberechtigte Ausführung von Schadcode auf dem IT-System mit TrueCrypt, könnte ein Angreifer versuchen Sicherheitsmechanismen von TrueCrypt zu schwächen oder aushebeln. Allerdings stellt sich auch hier die Frage, wie ein Szenario konkret aussehen und welchen Kosten-Nutzen-Effekt ein Angreifer auch in Vergleich zu anderen Angriffsstrategien erreichen könnte. Um das IT-System mit Schadcode zu kompromittieren sind andere Methoden der Schwachstellenausnutzung, z. B. in den Rendering-Algorithmen von Webbrowsern, in Adobe-Flash-Software, in PDF-Software etc. besser geeignet. Wenn ein Angreifer mit einem solchen Angriff weitreichenden logischen Systemzugang erreichen würde, dann kann er üblicherweise bereits vertrauliche Daten aus entsperrten TrueCrypt-Containern oder -Partitionen lesen. Interessant wäre aus praktischen Gesichtspunkten möglicherweise die Vorbereitung von physischen Angriffen; so könnte ein Angreifer beispielsweise mit Schadcode erreichen, dass Schlüssel im Klartext im persistenten Speicher abgelegt werden. Dann könnte er bei einem folgenden physischen Zugriff einfach den Datenträger entwenden; die zur Entschlüsselung notwendigen Schlüssel könnte er direkt von diesem Auslesen.

Angriffe über Hintertüren (Backdoors)

Eine weitere spezielle Angriffsstrategie ist das Ausnutzen sogenannter »Hintertüren« (»Backdoors«) in TrueCrypt, welche nur dem Angreifer bekannt sind. Ein Angreifer muss zur Einbettung einer solchen Hintertür in TrueCrypt unbemerkt Zugriff auf den Quellcode der Software gehabt, oder selbst als Entwickler an der Software mitgewirkt haben. Eine Hintertür kann aber auch auf andere Weg in die Software gelangt sein, so z. B. durch eine von den Entwicklern unbemerkte Manipulation von Quellcode in einem Quellcode-Repository. Alternativ könnte ein Angreifer den Quellcode oder kompilierte Programmteile vor einer Übertragung von den Entwicklern zum Endnutzer manipuliert haben, möglichweise inklusive notwendiger öffentlicher Schlüssel zum Signieren des Codes.

Die Hintertür kann verschiedene Funktionen implementieren, wobei häufig entscheidend für den Angreifer ist, dass seine Manipulationen nicht entdeckt werden. Das Verstecken der Hintertür ist insbesondere dann wichtig, wenn der Angreifer direkt den Originalquellcode der Entwickler kompromittiert. Veränderungen können und sollten hier sehr subtil sein, z. B. ein bewusstes Schwächen der Implementierung eines kryptografischen Algorithmus, um kryptografische Analysen zu vereinfachen. Großflächige Manipulationen bieten sich eher bei bereits kompilierten Programmcode an, beispielsweise eine gezielte Veränderung bevor oder während ein Endnutzer das TrueCrypt-Programmpaket herunterlädt.

Verändern verschlüsselter Daten

Im Gegensatz zu den zuvor beschriebenen Angriffen auf die *Vertraulichkeit* TrueCrypt-verschlüsselter Daten, könnte ein Angreifer prinzipiell auch die Integrität der geschützten Daten kompromittieren wollen. Dazu könnte er das Chiffrat TrueCrypt-verschlüsselter Daten verändern, wobei das naheliegende Ziel ein veränderter Programmablauf in einem TrueCrypt-verschlüsselten System ist. Beispielweise könnte ein Angreifer durch eine (unkontrollierte) Veränderung einer verschlüsselten Konfigurationsdatei erreichen, dass bestimmte Zugriffskontrollen in dem IT-System ineffektiv werden.

Allerdings bleibt festzuhalten, dass wenngleich diese Angriffsstrategie aus theoretischer Perspektive umsetzbar erscheint, das mit ihr einhergehende Verhältnis von Kosten zu Erfolgswahrscheinlichkeit, sowie ihr prinzipieller Nutzen fraglich sind. Zunächst müsste ein Angreifer wissen, wo in dem verschlüsselten TrueCrypt-Volume oder -Container die zu kompromittierenden Daten bzw. Dateien liegen. Diese Abschätzung ist mit einer nicht zu vernachlässigenden Unsicherheit verbunden, was breitere Veränderungen am Chiffrat erforderlich und damit auch eine Erkennbarkeit von Veränderung wahrscheinlich macht. Weiterhin geht dieses Angriffsszenario davon aus, dass der Angreifer physischen Zugang zum System hat, oder einen umfangreichen logischen Zugang. Dann stellt sich die Frage, warum ein Angreifer eine solche unsichere, teilweise unkontrollierbare, sowie auffällige Angriffsstrategie anwenden sollte. Einfacher sind unter solchen Voraussetzungen Angriffe auf die unverschlüsselten Daten, die leichter und zielsicherer zu gewollten Fehlfunktionen in dem System führen können.

7.4.4 Zwischenfazit

Aus der Analyse der denkbaren Angriffsstrategien heraus stellt sich die Frage, gegen welche der beschriebenen Angriffe TrueCrypt Gegenmaßnahmen vorsehen könnte, welche anwendbare Angriffstechniken ausschalten oder behindern können. Als Fazit lässt sich zunächst festhalten, dass TrueCrypt prinzipbedingt keine effektiven Maßnahmen gegen Angriffe in Mehrfachzugriffsszenarien implementieren kann. Wie Türpe et al. bereits zeigten, ist dieser Schutz selbst für Systeme mit TPM-basierten Secure-Boot-Mechanismen kaum zu leisten [23]; im Falle der Microsoft BitLocker Drive Encryption ist beispielsweise kein effektiv wirksamer Schutz gegeben. Technisch bedingt kann TrueCrypt hier kein höheres Sicherheitsniveau liefern.

Die zweite wichtige Feststellung ist, dass in dem Fall, dass ein technisch überdurchschnittlich versierter Angreifer Zugang zu einem *aktiven* IT-System mit laufender TrueCrypt-Entschlüsselung erlangt, das Risiko einer erfolgreichen Extraktion von Schlüssel- oder Klartextmaterial immer als hoch einzuschätzen ist. Dieses Risiko ist system- und situationsimmanent – die notwendigen Schlüssel sind zumindest im flüchtigen Speicher hinterlegt, um eine Entschlüsselung zu ermöglichen. Die Softwarearchitektur von TrueCrypt kann zum Schutz dieser Schlüssel bestenfalls flankierende Schutzmaßnahmen leisten, wie z. B. einen Schutz vor Seitenkanalangriffen, oder »sparsame« Ablage von Passwörtern und Schlüsselmaterial im flüchtigen Speicher. Dem gegenüber steht jedoch eine breite Palette von Angriffstechniken, wie Cold-Boot-Attacken und sonstige Eingriffe in die Hardware, oder auch das Ausnutzen eventueller Schwachstellen im Betriebssystem, Netzwerkdiensten und anderer auf dem System laufender Software. Die Frage, ob solche Attacken erfolgreich sind hängt nicht maßgeblich von den Sicherheitsmechanismen in TrueCrypt, sondern vom Gesamtzustand des IT-Systems, und der Expertise und den materiellen Fähigkeiten des Angreifers ab. Konservativ betrachtet kann man festhalten, dass ein aktives IT-System mit laufender TrueCrypt-Entschlüsselung per se als leicht kompromittierbar angesehen werden muss.

Als relevant in der Diskussion verbleiben die Angriffe mit Einzelzugriff auf inaktive IT-Systeme mit TrueCrypt. Hier können vor allem solche Schwachstellen in TrueCrypt Angriffe erlauben, bei denen Informationen zu verwendeten Schlüsseln und Passwörtern im Klartext auf persistente Datenträger geschrieben werden, oder die Verschlüsselung selbst algorithmische Schwächen aufweist. Es ist dabei sogar möglich, dass Angreifer, welche schreibenden Zugang zum TrueCrypt-Quellcode hatten, solche Schwachstellen als »Hintertüren« gezielt zum TrueCrypt-Quellcode hinzugefügt haben könnten. Ist die Schwachstelle einmal vorhanden, dann kann der Angriff selbst durch (zeitweise) Entwendung des Datenträgers und anschließender forensischer Analyse erfolgen.

7.4.5 Gegenüberstellung mit Sicherheitsmodell von TrueCrypt

Das Sicherheitsmodell der TrueCrypt-Entwickler sieht zunächst einen sehr engen Anwendungsbereich für die Software vor [9, Seite 83ff.]: TrueCrypt verschlüsselt Daten vor dem Schreiben auf einen persistenten Speicher, und entschlüsslt diese nach dem Lesen wieder.

Daneben weisen die Entwickler explizit darauf hin, dass TrueCrypt im Wesentlichen folgende Eigenschaften und Funktionen *nicht* besitzen würde:

- Sichern der Daten in einem IT-System, welches ein Angreifer in irgendeiner Form manipuliert oder kontrolliert, oder welches ein Angreifer beobachten kann,
- Sichern von Daten auf einem IT-System, zu welchem ein Angreifer bevor, während, oder unmittelbar nach der Ausführung von TrueCrypt-Zugang erlangt,
- Schutz der Integrität oder Authentizität von Daten,
- Verschlüsseln von Daten im flüchtigen Speicher,
- Schutz von Informationen zur Veränderung von Daten auf verschlüsselten Datenträgern oder in verschlüsselten Volumes,
- Schutz von Datenabflüssen außerhalb der TrueCrypt-Verschlüsselung, z.B. bei Übertragung über Computernetzwerke.

Wenn man dieses Sicherheitsmodell zugrundelegt, dann schließen die Entwickler von sich aus schon einen Schutz gegen viele der in 7.4.3 genannten Angriffsstrategien kategorisch aus. Dies betrifft insbesondere auch Mehrfach-Angriffe, Einzelangriffe auf Systeme mit aktiver TrueCrypt-Entschlüsselung, Angriffe mit logischem Zugang zum IT-System, sowie Spähangriffe. Diese Analyse deckt sich mit den im Zwischenfazit in 7.4.4 getroffenen Feststellungen, dass die Angriffe insgesamt eine große Herausforderung für die Systemsicherheit sind, denen mit Verschlüsselungssoftware

kaum begegnet werden kann. Insgesamt wirkt das Sicherheitsmodell konservativ konstruiert mit wenigen Zusicherungen und vielen Hinweisen auf verbleibende Sicherheitsrisiken.

Im besonderen Fokus des Sicherheitsmodells liegt das ungewollte Speichern von Schlüsseln, Passwörtern und Klartextdaten in unverschlüsselten Bereichen des Datenträgers. Diese ungewollten »Abflüsse« von Klartextdaten sind insbesondere dann problematisch, wenn ein Angreifer Gelegenheit bekommt, einen Datenträger mit TrueCrypt-verschlüsselten Daten forensisch zu untersuchen. Die Entwickler weisen hier auf sicherheitsrelevante Besonderheiten der Betriebssysteme (Page Files, Ruhemodus, etc.) und spezieller Hardware (Wear-Leveling, physisch verschobene Sektoren, Löschung flüchtiger Speicher usw.) hin.

7.5 Bewertung der Architektur

In diesem Unterkapitel diskutieren wir, wie geeignet die TrueCrypt-Architektur ist, die beschriebenen Angriffsszenarien grundsätzlich zu adressieren. Hierfür geben wir zunächst einen Überblick über die Laufzeitkomponenten von TrueCrypt. Anschließend erläutern wir, wie diese Komponenten zusammenspielen, um die Funktionalität für die eingangs dargelegten Anwendungsfälle umzusetzen. Im Zuge dessen bewerten wir, wie die Anforderungen R1-R7 erfüllt wurden, und gehen auf die relevanten Angriffsszenarien ein. Abschließend betrachten wir die Themen Wartbarkeit und Testbarkeit und geben Verbesserungsempfehlungen für eine teilweise Neuimplementierung.

7.5.1 Die Komponenten im Überblick

Nachfolgend erläutern wir, aus welchen Laufzeitkomponenten TrueCrypt besteht und welche grundlegende Verantwortung sie im Gesamtsystem übernehmen. Diese Architekturperspektive, die das System zur Laufzeit beschreibt, ist im Kontext der Bewertung von Angriffsszenarien und der Erfüllung von Schutzzielen von besonderem Interesse, denn alle hier betrachteten Angriffe erfolgen auf das laufende (oder falls ausgeschaltet, zumindest lauffähige) System.

Davon abgegrenzt ist eine Architekturbeschreibung, die die Struktur des Quelltextes abbildet. Grundsätzlich gibt es auch Angriffsszenarien, die auf die Integrität des Quelltextes abzielen, wie z.B. das Einbauen einer Hintertür, die möglichst schwer auffindbar sein soll. Diese werden wir jedoch nicht näher betrachten, daher ist auch die Struktur des Quelltextes an dieser Stelle nicht von zentralem Interesse.

Abbildung 7.2 zeigt die Laufzeitkomponenten von TrueCrypt im Überblick. Je nach Zielplattform besteht TrueCrypt aus unterschiedlichen Komponenten. Dies ist einerseits darin begründet, dass sich die angebotene Funktionalität von TrueCrypt auf den unterschiedlichen Plattformen unterscheidet, wobei auf Windows-Systemen der größte Funktionsumfang zur Verfügung steht. Andererseits können auch plattformspezifische Besonderheiten für eine andere Organisation von Laufzeitkomponenten sprechen.

Auf Linux-basierten Systemen besteht TrueCrypt aus lediglich einer Kernkomponente, die mit anderen Systemkomponenten interagiert:

Main Wir bezeichnen die Hauptkomponente auf Linux-basierten Systemen als »Main«, da sie auch im Quelltext so genannt wird. Hierbei handelt es sich um eine ausführbare Datei, die sowohl eine Kommandozeilenschnittstelle, wie auch eine graphische Benutzeroberfläche implementiert. Jede direkte Benutzerinteraktion mit TrueCrypt erfolgt über eine dieser Schnittstellen. Jegliche Funktionalität, die von TrueCrypt auf dieser Plattform angeboten wird, wird in dieser Komponente umgesetzt. Hierfür bedient sich Main einerseits Betriebssystemfunktionen (z.B. für die Darstellung der Benutzerschnittstelle), andererseits auch FUSE (»Filesystem in Userspace«). FUSE ist ein Kernelmodul und wird verwendet, um verschlüsselte Volumes als virtuelle Laufwerke anbinden zu können. Bei FUSE handelt es

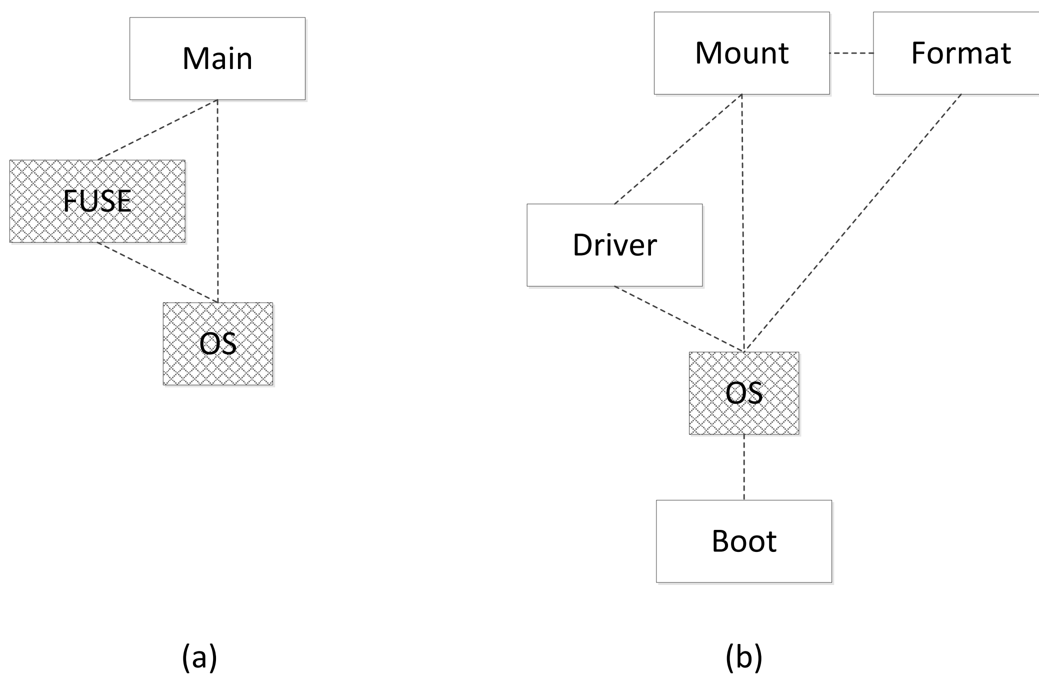

Abbildung 7.2: Vereinfachte Darstellung der Laufzeitkomponenten von TrueCrypt; (a) zeigt die Komponenten für Linux-basierte Systeme, (b) für Windows-Systeme

sich um eine vom TrueCrypt-Projekt unabhängige Software, die auch von einer Vielzahl anderer Projekte auf Linux-basierten Systemen für ähnliche Zwecke verwendet wird.

Auf Windows-Systemen besteht TrueCrypt aus den folgenden vier Komponenten:

Format Format ist eine ausführbare Datei, die per Kommandozeile, oder graphischer Benutzeroberfläche direkt vom Benutzer gesteuert werden kann. Das Programm stellt im Wesentlichen einen Assistenten dar, mit dessen Hilfe der Benutzer Volumes anlegen und/oder eine Systemverschlüsselung anstoßen kann.

Mount Neben Format ist Mount die zweite ausführbare Datei, die dem Benutzer eine textbasierte und graphische Schnittstelle anbietet. Wie der Name andeutet, wird dieses Programm dafür verwendet, bereits angelegte Volumes als virtuelle Laufwerke anzubinden. Um diese Funktionalität umzusetzen bedient sich Mount des Drivers. Mount bietet in der graphischen Benutzeroberfläche auch die Möglichkeit, Volumes zu erstellen und eine Systemverschlüsselung zu starten, allerdings wird hierfür direkt Format gestartet, die Funktionalität wurde also nicht nochmals in Mount implementiert.

Driver Der Driver ist ein Kerneltreiber und stellt eine zentrale Komponente von TrueCrypt auf Windows-Systemen dar. Einerseits wird er verwendet, um verschlüsselte Volumes als virtuelle Laufwerke anzubinden, womit er im Wesentlichen die Funktionalität von FUSE in diesem Anwendungskontext übernimmt. Darüber hinaus kann er auch als »vermittelnde« Komponente derart in die Funktionen des Betriebssystems eingebunden werden, dass alle schreibenden und lesenden Zugriffe auf einen Datenträger zunächst durch ihn gehen. Im Zuge dessen kann der Treiber dann die »on-the-fly« Ver- und Entschlüsselung vornehmen.

Boot Die Boot-Komponente umfasst einen Bootloader und implementiert das, was man allgemein als »Pre-Boot Authentication« bezeichnet. Boot findet nur dann Verwendung, wenn eine Systemverschlüsselung durchgeführt wurde, da eine verschlüsselte Systempartition nicht ohne Weiteres bootfähig ist. Zum Start des Computers erfragt die Komponente den Benutzer nach dem Passwort, das benötigt wird, um auf die verschlüsselten Inhalte zugreifen zu können. Während dieser frühen Phase des Startprozesses übernimmt Boot dann die »on-the-fly« Ver- und Entschlüsselung, sodass der ursprüngliche Bootloader geladen und die Kontrolle übergeben werden kann. Sobald das Betriebssystem gestartet wurde, übernimmt dann der Driver die Aufgabe der transparenten Ver- und Entschlüsselung und Boot hat keine Aufgabe mehr.

7.5.2 Die Kernfunktionalität von TrueCrypt

Nachfolgend erläutern wir, wie die drei Kernfunktionalitäten grundsätzlich technisch von TrueCrypt umgesetzt werden. Anschließend bewerten wir die Umsetzung mit Hinblick auf die Anforderungen R1-R7. Abschließend diskutieren wir die Angemessenheit der Architektur mit Hinblick auf relevante Angriffsszenarien.

Verschlüsselte Volumes Der erste Schritt für die Verwendung eines verschlüsselten Volumes ist dessen Erstellung. Auf Windows-Systemen verwendet der Benutzer hierfür TrueCrypt Format. Format ist wie ein Assistent aufgebaut und erfragt zunächst vom Benutzer Schritt für Schritt alle Informationen, die benötigt werden, um ein Volume anzulegen. Abschließend erstellt und speichert Format das gewünschte Volume. Wie in Sektion 7.2 bereits erläutert, kann es sich hierbei um datei- oder partitions-/gerätebasierte Volumes handeln. Die Datenstruktur, in der Volumes abgelegt werden, ist im TrueCrypt User Guide dokumentiert und ist grundlegend gleich, sowohl für dateibasierte, wie auch für andere Volumes.

Um auf die Inhalte eines bereits angelegten Volumes zugreifen zu können, verwendet der Benutzer TrueCrypt Mount. Mount erfragt die notwendigen Daten vom Nutzer (wie z.B. Pfad zum Volume, Passwort, Passwortdateien, etc.) und übergibt diese Informationen mit einem Steuerbefehl an den Driver. Der Driver erstellt ein virtuelles Laufwerk und leitet alle Schreib- und Lesezugriffe auf dieses Laufwerk zu dem Volume um. Während des »Umleitens« führt der Driver die on-the-fly Ver- und Entschlüsselung durch.

Das Umleiten geschieht wie folgt: Im regulären Betrieb verwenden Applikationen das sogenannte Win32-Subsystem. Das Subsystem bietet eine umfangreiche API (unter anderem Kernel32.dll, User32.dll, etc.) für gängige Operationen, wie beispielsweise für das Lesen und Schreiben von Daten auf einem Datenträger. Eine derartige Ein-/Ausgabe-Operation gelangt vom Windows-Subsystem zu einer weiteren Windows-Systemkomponente, dem I/O Manager, der die Anfrage weiter bearbeitet. Hierfür erstellt und initialisiert der I/O Manager zunächst ein sogenanntes Input/Output Request Packet, oder kurz IRP, welches alle Informationen über die gewünschte Schreib/Leseoperation beinhaltet. Dieses IRP wird dann vom I/O Manager an den Device-Stack gesendet, der für den angesprochenen Datenträger verantwortlich ist. Der Device-Stack umfasst alle Treiber (wie z.B. Bustreiber, Dateisystemtreiber, etc.) die nötig sind, um ein Gerät anzusprechen. Im Falle des von TrueCrypt erstellten virtuellen Laufwerks ist der TrueCrypt-Kerneltreiber (Driver) ein Teil dieses Stacks und erhält daher auch die IRPs, die einen Schreib- und Lesevorgang beschreiben. Dies ist genau die Stelle, an der der Driver »einspringt«, die Ver- und Entschlüsselung durchführt, und im Falle eines Schreibvorgangs die neuen Daten im Volume persistiert. Für Anwendungen ist es daher unerheblich, ob sie Daten von einem TrueCrypt-Volume, oder von einem unverschlüsselten Datenträger lesen/schreiben, denn die API des Subsystems wird stets auf die gleiche Weise angesprochen.

Systemverschlüsselung Das Verschlüsseln einer Systemplatte, oder einer Systempartition, geht mit einer Besonderheit einher: Das verschlüsselte System ist nicht ohne Weiteres bootfähig. Damit das von TrueCrypt verschlüsselte System also überhaupt gestartet werden kann, wird die TrueCrypt Boot-Komponente benötigt, die im Zuge der initialen Systemverschlüsselung von TrueCrypt installiert wird. Der TrueCrypt Bootloader startet anstelle des ursprünglichen Bootloaders und erfragt zunächst vom Benutzer das geheime Passwort. Mithilfe des Passworts kann die Boot-Komponente aus dem Volumeheader die geheimen Schlüssel auslesen, die benötigt werden, um auf Daten des verschlüsselten Systems zugreifen zu können. Anschließend führt die Komponente einen sogenannten Interrupt-Hook durch, um nachfolgende Lesezugriffe auf die verschlüsselten Daten durch eine Entschlüsselungsroutine umzuleiten. Im Detail geschieht dies wie folgt: Kurz nachdem das System startet, befindet es sich im sogenannten Real Mode. Im Real Mode stehen dem Bootloader und eventuell nachgeladenem Code sogenannte BIOS-Interrupts zur Verfügung, um grundlegende Ein- und Ausgabeoperationen vornehmen zu können. Interrupt 13h (kurz: INT 13h) ist an dieser Stelle für unsere Betrachtungen von besonderem Interesse, denn dieser Interrupt wird für Lese- und Schreiboperationen auf der Festplatte verwendet. Die TrueCrypt Boot-Komponente manipuliert nun gezielt mittels eines Interrupt-Hooks die Funktionsweise von INT 13h, sodass Lese- und Schreiboperation durch eine Ver- bzw. Entschlüsselungsroutine laufen. Nachdem dies erfolgt ist, übergibt TrueCrypt die Kontrolle an den ursprünglichen Bootloader. Der ursprüngliche Bootloader kann nun, unter Verwendung von INT 13h, die eigentlich verschlüsselten Kernkomponenten des Systems laden und die Kontrolle übergeben. Sobald das Betriebssystem die Kontrolle übernommen hat, werden Lese- und Schreibzugriffe nicht mehr durch INT 13h abgearbeitet, sondern stattdessen durch Kerneltreiber umgesetzt. An diesem Punkt übernimmt die TrueCrypt Driver-Komponente die on-the-fly Ver- und Entschlüsselung, genauso, wie dies für verschlüsselte Volumes bereits erläutert wurde.

Versteckte Volumes Jedes Volume besteht im Wesentlichen aus einem verschlüsselten Header

und aus verschlüsselten Nutzdaten. Ein Header reserviert in jedem Falle dedizierten Platz für einen zweiten Header, der zu einem versteckten Volume gehören kann. Falls ein TrueCrypt Volume nicht als äußeres Volume für ein zweites, verstecktes Volume dient, dann ist der dedizierte Platz für den zweiten Header mit Zufallszahlen befüllt. Des Weiteren ist in diesem Falle der ungenutzte Speicher diese Volumes ebenso mit Zufallszahlen befüllt. Dies ist anders, wenn ein Volume tatsächlich als äußeres Volume dient. In diesem Falle beinhaltet der dedizierte Platz für den zweiten Header tatsächlich den verschlüsselten Header für das versteckte Volume. Die verschlüsselten Nutzdaten des versteckten Volumes befinden sich dann im Bereich des ungenutzten Speichers des äußeren Volumes. Ohne das Passwort für das versteckte Volume kann weder dessen Header, noch dessen Nutzdaten gelesen werden. Tatsächlich kann ein Angreifer in dieser Situation die verschlüsselten Daten nicht von Zufallsdaten unterscheiden, die Existenz des versteckten Volumes ist also glaubhaft abstreitbar.

Grundsätzlich verhält sich TrueCrypt wie folgt: Wenn ein Benutzer versucht, auf ein Volume zuzugreifen, erfragt TrueCrypt zunächst das geheime Passwort. Dies erfolgt entweder durch TrueCrypt Boot, oder Mount, je nachdem, ob eine Systemverschlüsselung durchgeführt wurde, oder nicht. Mit dem eingegebenen Passwort versucht die jeweilige TrueCrypt-Komponente (Boot oder Driver) dann zunächst den äußeren Header zu lesen. Wenn dies erfolgreich ist, dann werden die Inhalte dieses äußeren Volumes zugreifbar gemacht. Wenn dies jedoch scheitert, versucht TrueCrypt im nächsten Schritt mit dem Passwort den möglicherweise vorhandenen zweiten Header zu lesen. Wenn dies ebenso scheitert, meldet TrueCrypt, dass das Passwort falsch ist. Ist dies jedoch erfolgreich, dann ist nun sicher, dass es sich hierbei um einen Zugriffsversuch auf ein tatsächlich vorhandenes, verstecktes Volume handelt, und die Nutzdaten dieses versteckten Volumes werden zugreifbar gemacht. Aus technischer Sicht erfolgt der Zugriff auf die Nutzdaten des versteckten Volumes dann in nahezu genau der gleichen Weise, wie auf ein äußeres Volume. Im Falle eines versteckten Betriebssystems geschieht dies also ganz ähnlich wie im Abschnitt »Systemverschlüsselung« beschrieben, im Falle eines versteckten Volumes wie im Abschnitt »Verschlüsselte Volumes« beschrieben.

Nachdem wir die grundlegende technische Umsetzung erläutert haben, betrachten wir nun nachfolgend, in wie fern die Anforderungen R1-R7 erfüllt wurden.

R1 - »Einsatz geeigneter kryptographischer Verfahren«: TrueCrypt verwendet für den Schutz der Daten in einem Volume anerkannte kryptographische Verfahren. So stehen als Blockchiffre die Verfahren AES, Serpent und Twofish zur Verfügung. Als Betriebsmodus wird XTS verwendet. Wir betrachten die Anforderung somit als erfüllt.

R2 - »Keine Speicherung vertraulicher Daten im Klartext«: Beim Erstellen eines verschlüsselten Volumes ist dieses zunächst leer. Alle Daten, die anschließend in das Volume geschrieben werden, werden im Zuge der Ausgabeoperation vom Driver verschlüsselt. Im Zuge der Initialisierung einer Systemverschlüsselung wird das gesamte System (je nach Auswahl also die Systempartition, bzw. die Systemplatte) auf dem Datenträger verschlüsselt. In jedem Falle werden im Zuge einer Leseoperation die verschlüsselten Daten vom Driver lediglich im Arbeitsspeicher entschlüsselt und an die anfragende Applikation weitergegeben, TrueCrypt selbst persistiert also im regulären Betrieb keine Klartexte. Der Driver wird auf dem System des Benutzers ausgeführt und bedient sich keiner systemexternen Dienste. Somit betrachten wir diese Anforderung als erfüllt.

R3 - »Integritätssicherung zu schützender Daten«: Durch die Verschlüsselung ist sichergestellt, dass Daten von einem Angreifer nicht beliebig so manipuliert werden können, dass nach einer Entschlüsselung bestimmte Klartexte entstehen. Darüber hinaus gibt es jedoch

keine Integritätssicherung, die auf eine unerlaubte Veränderung eines Volumes hindeuten könnte. Damit betrachten wir die Anforderung als teilweise erfüllt. Der Integritätsschutz nimmt jedoch im vorliegenden Kontext einen geringeren Stellenwert ein als der Schutz der Vertraulichkeit.

R4 - »Integritätssicherung von TrueCrypt selbst«: Die offiziell veröffentlichten Binärdateien für Mount, Format, und Driver sind digital signiert, somit kann deren Integrität auf einfache Weise vom Benutzer geprüft werden. Einige Windows-Versionen erfordern zudem per Default, dass Kerneltreiber signiert sind, um überhaupt benutzt werden zu können. Uns ist jedoch nicht bekannt, dass es eine Veröffentlichung eines TrueCrypt Bootloaders gibt, der sich zusammen mit Secure Boot verwenden lässt. Secure Boot ist eine zunehmend verbreitete technische Maßnahme, die gewährleistet, dass nur signierte Bootloader zum Systemstart verwendet werden. Dies erhöht den Aufwand für einen Angreifer, der eine Manipulation des Bootloaders durchführen möchte. Diese Anforderung ist damit nicht vollständig erfüllt.

R5 - »Keine Speicherung von Hinweisen auf versteckte Volumes«: Gemäß der Spezifikation des TrueCrypt User Guides beinhaltet jedes Volume einen Header mit Metainformationen. In diesem Header muss dediziert Platz für einen eventuell vorhandenen, zweiten verschlüsselten Header reserviert werden. Ohne das entsprechende Passwort kann ein Angreifer jedoch nicht zuverlässig entscheiden, ob der reservierte Platz lediglich mit Zufallszahlen, oder aber tatsächlich mit einem Header befüllt ist. Das gleiche gilt für die verschlüsselten Nutzdaten des versteckten Volumes: Entweder befinden sich diese im ungenutzten Speicher des äußeren Volumes, oder aber hier liegen nur Zufallszahlen. Auch dies kann ein Angreifer nicht unterscheiden. Da es keine weiteren Hinweise auf ein verstecktes Volume gibt, sehen wir die Anforderung als erfüllt an.

R6 - »Keine Speicherung von Schlüsseln oder Passwörtern im Klartext«: Es gibt keine Funktionalität, die es erfordern würde, Schlüssel oder Passwörter im Klartext abzulegen. Wir konnten keinen Hinweis darauf finden, dass TrueCrypt etwas Ähnliches durchführt. Die Anforderung ist erfüllt.

R7 - »Keine Übertragung von Schlüsseln oder Passwörtern an Dritte«: Aus den Anwendungsfällen und der Funktionsbeschreibung ergibt sich nicht die Notwendigkeit, Schlüssel oder Passwörter an Dritte zu übertragen. Wie konnten keinen Hinweis darauf finden, dass TrueCrypt etwas derartiges durchführen würde. Die Anforderung ist erfüllt.

An dieser Stelle betrachten wir abschließend, wie angemessen die grundlegende technische Umsetzung von TrueCrypt mit Hinblick auf die vorgestellten Angriffsszenarien ist. Hierbei ist an erster Stelle festzuhalten, dass ein mit TrueCrypt geöffnetes verschlüsseltes Volume im laufenden System grundlegend ungeschützt ist. Dies ist für alle Anwendungsfälle und Funktionalitäten gültig. In der Regel erscheint einem Angreifer mit logischem Zugang das geöffnete Volume wie ein reguläres Laufwerk. Daher kann es leicht ausgelesen oder manipuliert werden. Ebenso für alle Funktionalitäten gültig ist die Tatsache, dass TrueCrypt die verschlüsselten Inhalte eines Volumes nicht im Klartext persistiert. Selbst bei einem plötzlichen Systemausfall liegen die Daten also nicht unverschlüsselt vor, auch dann nicht, wenn zum Zeitpunkt des Ausfalls ein Volume geöffnet war.

Ist das System also ausgeschaltet, oder das Volume geschlossen, bietet TrueCrypt tatsächlich effektiven Schutz. Allerdings sind die von TrueCrypt angebotenen Funktionalitäten, und die Anwendungsfälle, im Rahmen derer sie verwendet werden, grundsätzlich mehr oder weniger anfällig für bestimmte Angriffsszenarien.

Die größte Auswahl an Möglichkeiten, die ein Angreifer haben kann, liegt im Falle der datei- oder partitions-/gerätebasierten Volumes vor. Zu den Besonderheiten zählt:

- Das System des Benutzers ist auf einfachste Weise über einen einmaligen logischen oder physischen Zugang analysier- und manipulierbar. Beispielsweise kann ein Angreifer Schadsoftware (wie einen Keylogger, oder eine Fernzugriffssoftware) installieren, um zukünftige Passworteingaben zu überwachen, oder direkt aus der Ferne auf die Daten eines geöffneten Volumes zuzugreifen.
- Durch die Verwendung von Keyfiles können Brute-Force-Angriffe deutlich erschwert werden. Hierfür bietet TrueCrypt Benutzern die Möglichkeit, eine Menge von Keyfiles mit einem Volume zu verknüpfen. Dies kann entweder gleich beim Erstellen des Volumes erfolgen, oder aber auch für bereits existierende Volumes durchgeführt werden. Beim Mounten des entsprechenden Volumes ist es dann nicht mehr ausreichend, das korrekte Passwort anzugeben, sondern der Benutzer muss auch auf die mit dem Volume verknüpften Keyfiles verweisen. TrueCrypt verwendet dann sowohl die Inhalte der Keyfiles, als auch das eingegebene Passwort, um den geheimen Schlüssel für den Volumeheader zu errechnen. Wenn entweder das eingegebene Passwort, oder die angegebenen Keyfiles nicht dem entsprechen, was der autorisierte Benutzer mit dem Volume verknüpft hat, scheitert die Schlüsselberechnung und der Zugriff auf das Volume bleibt verwehrt. Für den Angreifer erhöht dies den Aufwand für einen Brute-Force-Angriff, da nicht nur das Passwort korrekt erraten werden muss, sondern auch die Inhalte der jeweils verwendeten Keyfiles. Für diese Betrachtung ist jedoch zu berücksichtigen, dass TrueCrypt aus jeder Keyfile jeweils nur maximal die ersten 1.048.576 Bytes ($1024 \cdot 1024$ Bytes, entspricht 1MB) für die Schlüsselberechnung heranzieht. Falls eine Keyfile größer sein sollte, werden die zusätzlichen Daten ignoriert. Keyfiles können auf externen Datenträgern, wie etwa USB-Sticks oder aber auch SmartCards gespeichert werden. Die Anzahl der Keyfiles, die mit einem einzelnen Volume verknüpft werden können, ist praktisch nicht begrenzt.
- Die Integritätssicherung von TrueCrypt selbst kann in einigen Situationen vorteilhaft sein. Beispiel: TrueCrypt wird verwendet, um ein verschlüsseltes Backup auf einem externen Datenträger abzulegen. Um den Zugriff zu erleichtern, wird auf dem Backupmedium TrueCrypt in einer portablen Version abgelegt. Vor dem Zugriff auf das Volume kann die Integrität der TrueCrypt-Binärdateien leicht geprüft werden. Für einen Angreifer wäre es aufwendig, die Signatur zu fälschen.
- Die Verwendung versteckter Volumes geht zusätzlich mit glaubbarer Abstreitbarkeit einher. Es wird für einen Angreifer dadurch schwerer, den Zugang zu verschlüsselten Daten durch Bedrohung des Benutzers zu erzwingen.

Die Verschlüsselung der Systempartition ist aus der Sicht des Angreifers ein wenig anders zu betrachten als reguläre Volumes.

- In der Regel ist die Analyse und Manipulation des Systems aufwendiger, wenn die Systempartition vollständig verschlüsselt ist. So wird dadurch etwa die Möglichkeit Schadsoftware installieren zu können eingeschränkt.
- Eventuell vorhandene unverschlüsselte Partitionen sind dem Angreifer jedoch weiterhin zugänglich und können für eine forensische Analyse herangezogen werden. Gegebenenfalls legen Applikationen oder das Betriebssystem hier auswertbare Daten ab, die andere Angriffe erleichtern können. Womöglich legt der Benutzer gar selbst aus Versehen Daten in unverschlüsselten Bereichen ab.
- Da Secure Boot nicht unterstützt wird, stellt die Boot-Komponente eine Schwachstelle dar. Wie bereits praktisch demonstriert wurde (z. B. Stoned), kann ein Angreifer diese unverschlüsselte und nicht integritätsgesicherte Komponente infizieren und somit die Systemverschlüsselung angreifen.

- Es können keine Keyfiles verwendet werden, Brute-Force-Angriffe werden dadurch also nicht erschwert.

Die Verschlüsselung der gesamten Systemplatte schränkt den Angreifer im Kontext der betrachteten Funktionalität am meisten ein. Im Unterschied zur Verschlüsselung der Systempartition gibt es hier keine unverschlüsselten Partitionen mehr, die für einen Angreifer hilfreiche Daten beinhalten können. Im Sonderfall des versteckten Betriebssystems kommt zusätzlich die Abstreitbarkeit der Existenz hinzu.

Einige Angriffsszenarien sind jedoch in jedem Falle möglich. Beispielsweise ist die Integrität der Hardware nicht von TrueCrypt geschützt. Ein Angreifer könnte also etwa einen Hardwarekeylogger, oder andere Überwachungswerkzeuge am System anbringen, um geheime Informationen auszuspähen. Im Extremfall könnte ein Angreifer sogar die gesamte Hardware des Nutzers durch eine manipulierte Kopie ersetzen.

Zusammenfassend lässt sich sagen, dass die angebotene Funktionalität eine ganze Reihe von Angriffsszenarien zulässt. Die grundlegende technische Umsetzung von TrueCrypt ist dennoch angemessen, dann sie führt an dieser Stelle keine gravierenden, vermeidbaren Angreifbarkeiten ein.

7.5.3 Wartbarkeit und Testbarkeit

Unit-Tests sollten eine wesentliche Rolle im Test- und Entwicklungsprozess von TrueCrypt, sowie darauf basierender Software, einnehmen. Es gibt eine Reihe von Frameworks, wie beispielsweise CppUnit[1] oder Google Test[2], mit deren Hilfe entsprechende Testfälle für C++-Code entwickelt werden können. Eine umfangreiche Testsuite stellt sicher, dass sich bestimmte Teile des Codes erwartungskonform verhalten. Im Verlauf der Weiterentwicklung einer Software dienen sie als Regressionstests, um zu verhindern, dass durch Änderungen am Quelltext neue Fehler in bereits existierenden Code eingeführt werden.

Speziell für das Testen von Treibern auf Windows gibt es von Microsoft zur Verfügung gestellte Dokumentationen und Werkzeuge[3]. Neben umfangreichen Unterstützungen in aktuellen Visual Studio Editionen, die die Treiberentwicklung vereinfachen sollen, gibt es beispielsweise auch den Static Driver Verifier (SDV)[4]. Der SDV analysiert den Quelltext des Treibers um Fehler und Designprobleme zu finden, die eine fehlerfreie Interaktion mit dem Windows-Kernel gefährden können. Auf diese Weise können schwerwiegende Probleme bereits früh im Entwicklungsprozess gefunden werden. Neben dieser auf Treiber spezialisierten Lösung gibt es auch eine Reihe anderer, in der Regel kommerzieller Werkzeuge, die mittels statischer Programmanalyse Softwarefehler finden können. Die Fähigkeiten dieser Produkte sind mitunter recht unterschiedlich, das Einführen einer derartigen Analysesoftware in den Entwicklungsprozess sollte daher auf Grundlage sorgfältiger Prüfung erfolgen.

Als Ergänzung zu den auf Quelltext basierenden Testverfahren kann Fuzzing eingesetzt werden. Beim Fuzzing werden die Schnittstellen der Software zur Laufzeit mit einer sehr hohen Anzahl zufälliger Eingaben getestet, um eine Menge solcher Eingabewerte zu finden, die einen möglicherweise sicherheitskritischen Fehler in der Software verursachen. Auf diese Weise können Sicherheitsprobleme identifiziert werden, die nicht durch statische Analyse gefunden wurden, und die beim Erstellen der Unit-Tests keine Berücksichtigung fanden. Insbesondere bietet es sich an, die Schnittstellen der TrueCrypt-Treiberkomponente zu analysieren, da sie im Wesentlichen die Kernfunktionalität implementiert. Die Autoren des OCAP Phase 1 Prüfberichts geben an, einige Schnittstellen des Treibers und des Bootloaders mit Fuzzing getestet zu haben.

[1] http://sourceforge.net/projects/cppunit/
[2] https://code.google.com/p/googletest/
[3] https://msdn.microsoft.com/en-us/library/windows/hardware/ff554651%28v=vs.85%29.aspx
[4] https://msdn.microsoft.com/en-us/library/windows/hardware/ff552808%28v=vs.85%29.aspx

Ein gegebenenfalls recht aufwendiges Mittel um bestimmte Eigenschaften der zu testenden Software nachweisen zu können ist formale Verifikation. Durch den Einsatz mathematischer Beweisverfahren kann nachgewiesen werden, dass ein Teil der Software konform zu einer gegebenen formalen Spezifikation ist. Da dieser Nachweis in der Regel mit hohem Aufwand verbunden ist, kann das Verfahren im Falle von TrueCrypt nicht auf die gesamte Software angewendet werden, sondern muss sich auf besonders kritische Teile des Systems beschränken. Die Autoren des OCAP Phase 2 Prüfberichts geben die Empfehlung, die Funktionen `EncryptBufferXTSNonParallel` und `DecryptBufferXTSParallel` entsprechend zu verifizieren, da sie hier das Potential für fehlerhafte Zeigerarithmetik (engl. »pointer arithmetic«) und Feldgrößenprüfung (engl. »bounds checking«) sehen.

Um die Test- und Wartbarkeit von TrueCrypt und darauf basierender Software zu erhöhen, sollten zunächst die Codequalitätsmängel behoben und die Buildinfrastruktur aktualisiert werden. Nicht zuletzt ist die Wartbarkeit einer Software eng mit einem klaren und gut dokumentierten Softwaredesign verknüpft. Wohldefinierte Schnittstellen und eine modulare Softwarestruktur erleichtern zudem auch das Testen. Eine moderne Buildinfrastruktur ist erforderlich, um aktuelle Analysesoftware und Entwicklungswerzeuge einsetzen zu können. Einige dieser Produkte fügen sich für Analysezwecke in den Buildprozess ein, die Verwendung sehr alter Software könnte diesbezüglich problematisch sein.

7.5.4 Empfehlungen zur Verbesserung

Die Notwendigkeit Codequalitätsmängel zu beseitigen und die Buildinfrastruktur zu aktualisieren wurde bereits mehrfach betont. Neben dieser klaren Empfehlung gibt es eine Reihe weiterer Verbesserungsvorschläge, die die Sicherheit und Benutzbarkeit der Software erhöhen sollen.

Alternativen zu XTS-AES An erster Stelle steht die grundlegende Kritik am Betriebsmodus XTS-AES, der im Zuge der Volumeverschlüsselung verwendet wird. Wie in Kapitel 9 erläutert wird, weisen die Autoren des OCAP Phase 2 Prüfberichts auf mögliche Angriffsszenarien hin, die mit unzureichender Integritätssicherung und der geringen Blockgröße (16 Bytes) von AES in Zusammenhang stehen. Demzufolge kann ein Angreifer einen einzelnen Chiffreblock auf einem Datenträger manipulieren, um gezielt eine bestimmte Speicherstelle zu seinen Gunsten zu verändern, ohne dass dies dem Benutzer auffallen muss. Ohne Kenntnis des Schlüssels kann der Angreifer zwar nicht vorhersehen, welchen Effekt die Manipulation des Chiffrats auf den Klartext hat, allerdings ist in jedem Falle gewährleistet, dass lediglich der tatsächlich manipulierte Block verändert wird. Ein Angreifer kann davon profitieren, indem er bestimmte Teile einer ausführbaren Datei überschreibt, um ihren Kontrollfluss zu seinem Vorteil zu verändern. Auch möglich ist die Manipulation von Konfigurationsdateien und der Windows-Registry. Um einen derartigen Angriff durchführen zu können, muss der Angreifer Kenntnis über eine Speicherstelle haben, deren Manipulation seinem Vorteil dient, ohne aber die Funktionsweise des Zielsystems auf auffällige Weise einzuschränken. Um dem Problem zu begegnen verweisen die Autoren des Prüfberichts auf die Möglichkeit einer alternativen Lösung, die ursprünglich von Microsoft für Bitlocker entwickelt wurde und in [8] beschrieben wird. Die Publikation erläutert im Detail den Einsatz von AES-CBC im Zusammenhang mit einem sogenannten Diffusor. Der Diffusor fasst den Klartext mehrerer AES-Blöcke zu einem größeren Block zusammen. Vor der Verschlüsselung, bzw. nach der Entschlüsselung, wendet der Diffusor eine mathematische Operation auf den Klartext an, um lokale Änderungen über dem gesamten zusammengefassten Block zu verteilen. Die Blockgröße des Diffusors ist deutlich größer als die von AES und kann im Bereich von 512-8192 Bytes variiert werden. Durch den Einsatz des Diffusors ist sichergestellt, dass eine Manipulation des Chiffrats deutlich größere Auswirkungen auf den Klartext hat, als dies beim Einsatz von XTS-AES der Fall wäre. Selbst die minimale Blockgröße von 512 Bytes liegt deutlich oberhalb der 16 Bytes bei XTS-AES und erschwert Angriffe dadurch signifikant. Die Autoren

weisen in [8] jedoch deutlich darauf hin, dass der Einsatz des beschriebenen Verfahrens zunächst sorgfältig geprüft werden muss und keinesfalls für alle Anwendungsfälle geeignet sein muss. Bevor im Zuge der Weiterentwicklung von TrueCrypt ein ähnliches Verfahren angewendet wird, ist eine detaillierte Analyse der Auswirkungen dringend notwendig.

Ein möglicher alternativer Ansatz um die Integrität des Klartextes zu gewährleisten könnte der Einsatz eines speziellen Dateisystems sein, das bereits über eine entsprechende Funktionalität verfügt. Btrfs ist ein Beispiel für ein Dateisystem, das alle Daten und Metadaten mit einer Prüfsumme versieht, um die Integrität der gespeicherten Daten zu sichern. Im Kontext von TrueCrypt würde, selbst beim Einsatz von XTS-AES, die unerlaubte Manipulation des Chiffrats dadurch auffallen, dass die vom Dateisystem berechnete Prüfsumme nicht mehr korrekt ist. Ein Angreifer hätte ohne Kenntnis des Schlüssels keine Möglichkeit, die Prüfsumme, die ebenso wie die Daten selbst verschlüsselt ist, entsprechend an die Manipulation anzupassen.

Mehrbenutzerunterstützung Besonders im behördlichen und im Unternehmenseinsatz kann die Unterstützung mehrerer Benutzer pro Volume sinnvoll sein. TrueCrypt bietet derzeit keine direkte Möglichkeit, mehrere individuelle Benutzer mit einem einzelnen Volume zu verknüpfen. Eine derartige Funktionalität könnte beispielsweise dadurch umgesetzt werden, dass das Volume-Header-Format so angepasst wird, dass es den geheimen Schlüssel des Volumes nicht nur einmal ablegt, sondern gleich mehrmals. Dabei wäre jede Kopie des Schlüssels mit dem Passwort eines anderen Benutzers verschlüsselt, sodass jeder Nutzer mit einem individuellen Passwort darauf zugreifen kann. Die konkreten Anforderungen an eine Mehrbenutzerunterstützung sind stark vom Anwendungsfall abhängig, daher gibt es keine allgemeine Lösung, die in jedem Falle die Anforderungen erfüllt. Wie im TrueCrypt-User-Guide erläutert, kann durch die Verwendung von SmartCards im Zusammenhang mit Keyfiles eine ähnliche Funktionalität umgesetzt werden.

Entfernung der Hidden-Volume-Funktionalität Im professionellen Einsatz kann es für den Arbeitgeber wünschenswert sein, die Möglichkeit zu entfernen, versteckte Volumes anlegen zu können. Dadurch wird es Benutzern erschwert, geheime Daten auf dem Arbeitsgerät abzulegen, die selbst dem Arbeitgeber verborgen bleiben. Selbst wenn der Arbeitgeber dem Benutzer vollständig vertraut, kann die Entfernung dieser Funktionalität dazu beitragen, die Codebasis der Software zu verkleinern. Eine kleinere Codebasis ist gegebenenfalls mit geringerem Test- und Wartungsaufwand verbunden, und bringt möglicherweise eine kleinere Angriffsfläche mit sich. Das Entfernen von nicht benötigter Funktionalität kann daher durchaus nützlich sein und sollte im konkreten Anwendungsszenario geprüft werden.

7.6 Zusammenfassung

Zusammenfassung der Ergebnisse aus Kapitel 7

- Es gibt drei wesentliche Anwendungsfälle für TrueCrypt:
  - Informationsschutz durch Systemverschlüsselung
  - Informationsschutz durch verschlüsselte Volumes
  - Geheimhaltung und Informationsschutz durch versteckte Volumes
- Das primäre Ziel ist der Schutz der Vertraulichkeit von Daten in einem TrueCrypt-Volume.
- Neben dem primären Schutzziel gibt es noch weitere Ziele, wie etwa Integritätsschutz, oder Abstreitbarkeit.
- Die Zugriffsmöglichkeiten des Angreifers können in zwei Dimensionen unterschieden werden
  - physischer/logischer Zugang
  - Einzel-/Mehrfachzugriff
- Es gibt keine effektiven Maßnahmen gegen Angreifer mit Mehrfachzugriff.
- Ein geöffnetes Volume in einem laufenden System gilt als leicht kompromittierbar.
- Die grundlegende technische Umsetzung von TrueCrypt ist angemessen, da sie keine gravierenden, vermeidbaren Angreifbarkeiten einführt.

8 Identifikation entbehrlicher Code-Teile

8.1 Zielsetzung

Im Rahmen dieses Arbeitspaketes wurde die Codebasis auf Komponenten, Dateien und Funktionen untersucht, die für die gewünschte Funktionalität nicht erforderlich sind und somit gegebenenfalls zur Verkleinerung der Angriffsfläche entfernt werden könnten. Hierbei sollte, nach Absprache mit dem Auftraggeber, von allen möglichen Nutzungsszenarien ausgegangen werden, die TrueCrypt in der Praxis bietet. Insofern ist eine Funktion nur dann als entbehrlich zu betrachten, wenn sie in keiner möglichen Konfiguration der Software zum Einsatz kommt.

8.2 Vorgehensweise

Komponentenebene Um entbehrliche Teile auf Komponentenebene zu bestimmen, ist es hilfreich auf die Beschreibung der Software-Architektur im Abschnitt 7.5.1 zurückzugreifen. Sie liefert eine detaillierte Darstellung der Laufzeitkomponenten und ihrer Abhängigkeiten.

Dateiebene Auf dem Level von Dateien wurden im Rahmen des Projektes die Log-Dateien und Verzeichnisse der Windows und Linux-Builds verglichen, um abzuleiten ob jede `.c` und `.cpp`-Datei beim Kompilieren tatsächlich benutzt wird. Beim Build wird der Source-Code auf Linux in Dateien mit der Endung `.o` übersetzt, auf Windows werden sie in `.obj`-Dateien kompiliert. Existiert weder eine `.o` noch eine `.obj`-Datei zu einer Source-Code Datei, lässt sich daher folgern, dass die entsprechende Source-Code Datei im Build auf den entsprechenden Plattformen nicht benötigt wird.

Funktionsebene Um zu bestimmen, welche Funktionen in der Praxis zum Einsatz kommen, wurden zum einen die Call Graphen aus Abschnitt 4 herangezogen, zum anderen wurde der Quelltext auch von Hand auf Aufrufbeziehungen hin untersucht. Weitere Hinweise liefern die Resultate der statischen Codescanner aus Kapitel 5, die zusätzlich Checkers für nicht benutzte Funktionen, Branches und Variablen besitzen.

8.3 Ergebnis

Komponentenebene Die Architektur-Beschreibung aus Abschnitt 7.5.1 legt nahe, dass aus Sicht der Komponenten in einem allgemeinen Nutzungsszenario alle Komponenten benötigt werden (siehe Abb. 7.2). Schränkt man das Nutzungsszenario weiter ein, können komplette Komponenten weggelassen werden. Unter Windows ist es zum Beispiel möglich die Komponenten `Boot` wegzulassen, falls kein Interesse an einer kompletten Systemverschlüsselung besteht.

Dateiebene Lediglich die Dateien des User-Interfaces unter der Linux-Variante (hauptsächlich im Ordner `Main/Forms`) wurden nicht zum Kompilieren benötigt. Dies lässt sich allerdings darauf zurückführen, dass wir beim Build-Prozess das User-Interface explizit deaktiviert haben und sich die Analyse sich auf die Commandline-Variante TrueCrypts bezieht. Außerdem wurde zusätzlich noch die Datei `Crypto/Aescrypt.c` nicht übersetzt. Diese wird auf nicht-x86-Architekturen verwendet, auf denen sich diese Analyse nicht bezieht. Insgesamt lässt sich daher schließen, dass

ebenfalls auf Dateiebene ohne weitere Einschränkung der Nutzungsszenarien alle Dateien benötigt werden.

Funktionsebene Generell konnten keine größeren Codefragmente gefunden werden, die in TrueCrypt nicht verwendet werden, jedoch wurden auch nicht alle nicht verwendeten Codeteile konsequent eliminiert. Beispielsweise können die Funktionen `aes_encrypt_key`, `aes_encrypt_key128`, `aes_encrypt_key192` über das Setzen einer Preprozessor-Variablen aktiviert werden, es existieren aber keine Fälle in denen sie aufgerufen werden.

Die in Kapitel 5 beschriebenen Werkzeuge Clang, Coverity Scan und Cppcheck zur statischen Codeanalyse lieferten nur wenige Hinweise auf nicht-verwendete Codefragmente. Coverity und Clang berichten insgesamt über drei unbenutzte Variablen im Code, diese sollten definitiv gelöscht werden, führen aber im Weiteren nicht zu größeren entbehrlichen Codeteilen.

Weitere Empfehlungen Für konkrete Nutzungsstrategien von TrueCrypt ist es möglich eine detaillierte Aussage über entbehrliche Codeteile zu liefern. Hierzu kann beispielsweise unter Linux das Tool `gcov` hinzugezogen werden. Dies ist allerdings ein Werkzeug dynamischer Natur, es führt Statistik über benutzte und unbenutzte Aufrufe im Code zur Laufzeit des Programms. Hierzu müssen die gewünschten Operationen, z.B. ver- und entschlüsseln einer Partition mit TrueCrypt, ausgeführt werden. Anschließend, lässt sich dann anhand der Statistik argumentieren, welche Teile für die ausgeführte Funktionalität nicht von Bedeutung sind.

Bei einem solchen Vorgehen, muss jedoch darauf geachtet werden, dass das Tool dynamisch, also zur Laufzeit, protokolliert. Damit sind die Ausführungspfade, die nicht berichtet werden, nicht direkt entbehrlich. Zum Beispiel werden Pfade, die aufgrund von Fehlern auftreten nicht berücksichtigt, da das Programm zur Laufzeit möglicherweise einem anderen Ausführungspfad gefolgt ist.

Zusammenfassung der Ergebnisse aus Kapitel 8

- Unter einem allgemeinen Anwendungsszenario von TrueCrypt werden alle Komponenten und Dateien zum Ausführen von TrueCrypt benötigt.
- Durch weiteres Einschränken der Nutzungsszenarien kann unter Umständen auf komplette Komponenten verzichtet werden.
- Auf Funktionsebene konnten im Teilprojekt `Crypto` einige Funktionen ausfindig gemacht werden, die nicht ausgeführt werden.

9 Bewertung des OCAP Phase 2 Prüfberichts

9.1 Kommentare zum OCAP-2

Im Rahmen des Open Crypto Audit Project[1] wurde die zweite Phase des Audits von TrueCrypt in Version 7.1a abgeschlossen. Der Bericht von Alex Balducci, Sean Devlin und Tom Ritter erschien im März 2015 [2]. Innerhalb dieses Kapitels evaluieren wir den Rahmen des Berichts, diskutieren die Erkenntnisse und geben Handlungsempfehlungen für TrueCrypt basierte Software ab.

Umfang des Berichts

Die zweite Phase des OCAP bezog sich auf die Kryptographischen Services der TrueCrypt Software. Hierfür wurde ein umfangreiches source code review sowie gezieltes Debugging der Software auf einer Windows Plattform durchgeführt. Reverse Engineering des Assemblercodes oder der Abgleich zwischen den TrueCrypt Binärdateien und dem Source-Code wurden nicht durchgeführt. Der Fokus des Source Code Reviews lag auf den AES Implementierungen im XTS mode, sowie der Zufallszahlengeneratoren und der SHA-512 Hashfunktion. Des Weiteren wurde das *header volume format* untersucht. Insbesondere fällt auf, dass keine Analyse bezüglich nicht-sicheren Löschens von Speicher durchgeführt wurde.

In dieser Analyse ergaben sich 4 Schwachstellen, von denen jedoch keine eine generelle Attacke für beliebige TrueCrypt Installationen zulässt. Zwei Schwachstellen wurden als sehr kritisch, eine als minder kritisch und eine als unkategorisierbar eingestuft. Die Angriffsschwierigkeit wurde in allen Fällen als unkategorisiert oder hoch eingeschätzt.

9.2 Kommentare zu den Ergebnissen

Im Folgenden kommentieren wir die Resultate des OCAP-Berichts.

Resultat 1 – `CryptAcquireContext` may silently fail in unusual scenarios Die erste Schwachstelle bezieht sich darauf, dass der Random Number Generator von TrueCrypt unter Windows in manchen Szenarien auf schwache Entropiequellen beschränkt wird, ohne dass dies zu einem Abbruch oder zumindest einer Warnmeldung führt. [2] Der zugehörige Quellcode findet sich in der Datei `Common/Random.c` in den Zeilen 101-105, 645 sowie 756.

Die Schwachstelle beruht darauf, dass die Aufrufe zu `CryptAcquireContext`[3] fehlschlagen können, falls beispielsweise eine Windows Gruppenrichtlinie gesetzt ist, die den Zugriff auf die Schlüsselspeicher des Cryptographic Service Provider verbietet. TrueCrypt verwendet diesen Kontext ausschließlich zur Zufallszahlengenerierung via `CryptGenRandom` und würde somit eigentlich gar keinen Zugriff benötigen.

Sofern kein Cryptographic Service Provider Kontext zur Verfügung steht, nutzt TrueCrypt stattdessen eigene Entropiequellen, wie beispielsweise die Prozessnummer oder die aktuelle

[1] https://opencryptoaudit.org
[2] Diesbezüglich ist der Titel der Schwachstelle leider sehr technisch fokussiert und nicht zielgerichtet
[3] siehe https://msdn.microsoft.com/en-us/library/windows/desktop/aa379886.aspx

Laufzeit des Systems. Eine Auflistung findet sich in Annex A des OCAP-2-Berichts, sowie in der Datei `Common/Random.c` in den Zeilen 614 und 639 beziehungsweise 667-752 über die Funktionen `RandaddBuf` und `RandaddInt32`.

Der OCAP-2-Bericht empfiehlt, `CRYPT_VERIFYCONTEXT`[4] als 5. Parameter zu setzen. Dies bewirkt, dass bei der Initialisierung des Cryptographic Service Provider Kontexts kein Zugriff auf den Schlüsselspeicher erfragt wird. Somit würde `CryptAcquireContext` auch im Falle eines Verbots für den Zugriff auf die Schlüsselspeicher nicht fehlschlagen.

Darüber hinaus solle im Falle eines Fehlers bei dem kein Kontext zurückgegeben wird, ein Fehler ausgegeben und die Ausführung abgebrochen werden.

Dieses ist ähnlich dem Zustand beim Debian-OpenSSL-Bug[5], wobei jedoch mehr Entropiequellen genutzt werden. Dennoch können insbesondere in Szenarien von Virtuellen Maschinen, Eingebetteten Systemen und automatisierter Inbetriebnahme von Geräten auch diese (von TrueCrypt) verwendeten Entropiequellen ähnlich schwach zu Debians werden. Dementsprechend sollte darauf aufbauende Software die sinnvollen Empfehlungen des OCAP-2-Berichts dringend umsetzen und alte Schlüssel aus solchen Szenarien ersetzt werden.

Auch die Implementierung des Random Number Generators unter Linux aus der Datei `Core/RandomNumberGenerator.cpp` kann zu Schwächen in der Entropie führen, wie in Kapitel 4.3 beschrieben wird.

Resultat 2 – AES implementation susceptible to cache timing attacks Die zweite Schwachstelle beschreibt eine Möglichkeit, dass die verwendeten AES Schlüssel während der Verwendung ausgespäht werden können. Die Implementierungen in der Datei `Crypto/AesSmall.c` soll eine Implementierung verwenden, die per Cache-Timing Attacken angreifbar ist.

Bei einer Cache-Timing Attacke auf einen Schlüssel wird sich zunutze gemacht, wenn eine AES Implementierung Lookup-Tables verwendet und den Zugriff auf diese in naiver Weise optimiert. Der Angriff funktioniert dann beispielsweise so, dass ein Angreiferprogramm eingesetzt wird, dass sich in der CPU die Cache-Lines mit dem anzugreifenden Programm teilt. Das Angreiferprogramm nutzt dann seinen Schedulerslot um den Cache mit seinen Daten zu fluten. Im nächsten Slot (nachdem das anzugreifende Programm ausgeführt wurde) bemüht sich das Angreiferprogramm erneut um Zugriff auf seine Daten. Anhand der Latenz beim Datenzugriff kann es nun feststellen, welche Cache-Lines geflusht wurden. Diese Informationen ergeben ein Zugriffsmuster von allen parallel arbeitenden Programmen. Aus diesen Zugriffsmustern lassen sich dann Rückschlüsse auf mögliche Schlüssel in der AES Implementierung ziehen, die zu einer Einschränkung des möglichen Suchraumes und der Findung des geheimen Schlüssels führen.

Um einen solchen Angriff durchführen zu können benötigt der Angreifer jedoch Zugriff auf den gleichen Rechner und die Möglichkeit, Code auszuführen. Der OCAP-2-Bericht verweist in diesem Zusammenhang auf Fortschritte bei dieser Angriffstechnik, beispielsweise von JavaScript aus. Sie empfehlen die Nutzung alternativer Implementierungen, wie der von Käsper und der Nutzung von weiteren Mitigationsstrategien.

Diesen Empfehlungen sollte im Allgemeinen definitiv Folge geleistet werden. Insbesondere kann damit Szenarien vorgesorgt werden, bei denen aufbauend auf TrueCrypts Source Code neue Projekte geschaffen, und der vorhandene Code in anderer Weise weiter verwendet werden könnte. Für die Nutzung innerhalb von TrueCrypt sollte es im Allgemeinen kein Problem darstellen, da AesSmall.c nur für das Boot-Modul verwendet wird (siehe Boot/Windows/Boot.vcproj sowie Boot/Windows/Makefile). Eine Problematik stelle dies nur in dem Szenario dar, wenn TrueCrypt zur Full-Disk-Encryption in einer Virtuellen Maschine genutzt wird, und der Angreifer Zugriff auf eine andere Virtuelle Maschine hat. Für solche Szenarien steigt die Schwierigkeit dann nochmal stärker an, jedoch wären dann Cache-Timing Angriffe auf die AES Schlüssel von TrueCrypt möglich.

[4] https://msdn.microsoft.com/en-us/library/windows/desktop/aa379886.aspx#crypt_verifycontext
[5] siehe https://www.debian.org/security/2008/dsa-1571

Resultat 3 – Keyfile mixing is not cryptographically sound Der Algorithmus zum Ableiten eines Schlüssels aus Keyfiles (auch in Kombination mit einem Passwort) ist kryptographisch nicht sicher. Die Problematik liegt darin, dass das verwendete Verfahren auf Basis von CRC und Ring-Addition nicht Kollisionssicherheit im Sinne einer Kryptographisch Sicheren Hashfunktion ist. Das heißt, dass es möglich ist, bei einer Menge gegebener Keyfiles (und/oder Passworts) eine weitere KeyFile (und/oder Passwort) so zu errechnen, dass der Keypool für die Schlüsselberechnung einen vorher gewünschten Wert annimmt. Insbesondere ist es auch möglich, unter Kenntnis eines Keypools, der aus mehreren Keyfiles (und/oder Passwort) zusammengestellt ist, eine alternative Keyfile (teilweise auch Passwort) zu erstellen, die den gleichen Schlüssel erzeugen würde.

Der OCAP-2-Bericht empfiehlt die Nutzung von kryptographisch sicheren Hashfunktionen und stuft diese Lücke als gering ein.

Jedoch besteht der Zweck dieser Funktion in der Möglichkeit Mehrfaktorauthentifikation sowie Vier-Augen- oder Sechs-Augen-Prinzipien zu erlauben. Die Möglichkeit, diese Eigenschaft der Mehrfaktorheit sowohl während der Erstellung als auch im Nachhinein zu negieren, und dabei die Nachweisbarkeit einer solchen Manipulation nicht gegeben ist, sollte zumindest eine mittlere Bewertung sowie eine mittlere Schwierigkeit nach sich ziehen.

Die Szenarien können dabei wie folgt lauten: Zur Sicherung eines Schlüssels werden ein Passwort sowie eine Keyfile (auf einem USB-Stick) verwendet. Der Angreifer stellt dabei den PC, der die Keyfile für das Passwort erstellt. Im Zuge der Erstellung wird die Keyfile nicht zufällig, sondern so gewählt, dass es ein zweites Passwort gibt, mit dem ohne Keyfile der gleiche Schlüssel erzeugt wird. Ein anderes Szenario wäre, dass drei KeyFiles für ein Sechs-Augen-Prinzip erstellt wird, wobei zwei der KeyFiles so gewählt werden, dass sie sich gegenseitig aufheben. Dann ist es im Nachgang sowohl möglich, nach dem Sechs-Augen-Prinzip, als auch für den Eigentümer der nicht aufgehobenen Keyfile alleine, den AES Schlüssel zu erzeugen.

Entsprechend des OCAP-2-Berichts sollte ein kryptographisch sicherer Hash verwendet werden. Um die Eigenschaft von TrueCrypt zu erhalten, dass die KeyFiles in beliebiger Reihenfolge eingegeben werden können, könnten die KeyFiles entsprechend ihrer Werte sortiert werden, bevor sie gehasht werden. Dann würde auch kein Verlust dieses Features durch die Änderung auftreten.

In Fällen, wo nicht sichergestellt werden kann, dass der Ersteller der Keyfiles keine Manipulationen dieser vorgenommen hat, oder ein Angreifer zumindest zeitweisen Zugriff auf den Keypool zu Erstellung einer »MasterKeyfile« hatte, sollten alle Schlüssel entsprechend dieser sichereren Methode neue erstellt werden. Insbesondere trifft dies zu, da diese Manipulationen nicht nachweisbar sind. Selbst für den Fall der sich aufhebenden Keyfiles, ließen sich stattdessen auch solche erzeugen, die ein Passwort in Kombination mit der dritten Keyfile den Zugriff ermöglichen.

Resultat 4 – Unauthenticated ciphertext in volume headers Eine häufige Anforderung in der Kryptographie ist es, anhand eines symmetrischen Schlüssels sowohl die Vertraulichkeit als auch die Integrität eines Datums zu schützen. Hierbei ist es eine häufige Praxis, innerhalb des verschlüsselten Speicherblocks eine Checksumme in Form eines CRC oder Hashes abzulegen und nach der Entschlüsselung auf Plausibilität – Korrektheit bezüglich der eigentlichen Daten – zu überprüfen. In der Theorie sollte es einem Angreifer aufgrund der Verschlüsselung des Speicherblocks nicht möglich sein, sowohl die eigentlichen Daten als auch die Checksumme (sowie auch den Magic String) so zu verändern, dass die Ergebnisse wieder plausibel erscheinen.

Praktisch fällt ein entsprechender Nachweis jedoch schwer. Anstelle der gewählten Vorgehensweise ist es gängige Praxis, die Integrität der Daten kryptographisch ohne Umwege zu sichern. Der OCAP-2-Bericht geht darauf ein und sprich die gute Empfehlung aus, einen zweiten Schlüssel – neben dem Verschlüsselungsschlüssel – vom Hauptschlüssel abzuleiten und mit diesem einen Message Authentication Code (MAC) – dies könnte sowohl ein CMAC sowie ein HMAC sein – zu nutzen. Auf diese Weise ließe sich die Integrität der Daten entsprechend kryptographischer

Best-Practices gewährleisten.

Über diese Empfehlung hinaus gibt es die Möglichkeit, kryptographische Verschlüsselung in Modi für authenticated encryption zu betreiben. Die Modi CCM und GCM bieten dem Nutzer die Möglichkeit in einem Durchgang sowohl die Vertraulichkeit als auch die Integrität der Daten sicherzustellen.

9.3 Weitergehende Ergebnisse

XTS Mode Der OCAP Phase 2 Prüfbericht übt allgemein Kritik an XTS als Betriebsmodus für eine Festplattenverschüsselung. Demzufolge sei der Schutz der Integrität der Daten nur unzureichend gegeben. Wie bei einigen anderen Betriebsmodi beruht im Fall von XTS der Schutz vor unbefugten Änderungen lediglich darauf, dass ein Angreifer ohne Kenntnis des Schlüssels keine Änderung am Chiffrat vornehmen kann, die eine bestimmte und vorhersehbare Manipulation des Klartextes zur Folge hat. Da hierdurch unerlaubte Manipulationen aber nicht direkt erkennbar sind, wie dies beispielsweise beim Einsatz von sogenannten Message Authentication Codes (kurz MACs) der Fall wäre, wird diese Form des Schutzes in der Literatur auch als »poor man's authentication« bezeichnet. Als Problem wird in dem Prüfbericht beschrieben, dass bei XTS-AES die Blockgröße (16 Bytes) sehr klein ist und daher Angreifer durch Manipulation des Chiffrats zielgerichtet bestimmte Bereiche der Festplatte manipulieren können. Zwar sind ohne Kenntnis des Schlüssels die konkreten Auswirkungen einer Manipulation des Chiffrats auf den Klartext nicht vorhersehbar, allerdings ist gewährleistet, dass die Änderung lediglich den tatsächlich manipulierten Block betrifft und die anderen Blöcke davon unbeeinflusst bleiben. Ein Angreifer könnte sich dies etwa zunutze machen, indem er bestimmte Teile einer ausführbaren Datei überschreibt, um ihren Kontrollfluss zu seinen Gunsten zu ändern. Ebenso möglich ist die Manipulation von Konfigurationsdateien oder der Windows-Registry. Als Voraussetzung für einen solchen Angriff muss der Angreifer eine Speicherstelle kennen, deren Manipulation profitabel ist, aber nicht die Funktionsweise des Systems auf auffällige Weise stört. In Kapitel 7.5.4 des vorliegenden Berichts werden Vorschläge diskutiert, dem Problem zu begegnen.

Neben dieser allgemeinen Kritik an XTS empfehlen die Autoren des Prüfberichts eine formale Verifikation ausgewählter Funktionen in TrueCrypt (`EncryptBufferXTSNonParallel`, `DecryptBufferXTSParallel`), die im Zuge der Ver- und Entschlüsselung verwendet werden. Insbesondere sollen hierdurch Fehler in der Zeigerarithmetik (engl. »pointer arithmetic«) und der Feldgrößenprüfung (engl. »bounds checking«) ausgeschlossen werden.

Weitere Empfehlungen Weiter Empfehlungen des Berichts gehen darauf ein, das Code-Review fortzuführen und Code-Verbesserungen einzupflegen. Insbesondere die Pointerarithmetik und Boundchecks in der XTS-Implementierung auf verschiedenen Plattformen mit unterschiedlicher Endianess scheint hier bedeutsam. Auch die Parameter für die verschlüsselten Bereiche sowie Gesamtgröße erscheinen als mögliches Einfallstor und sollten durch den gesamten Source-Code hindurch analysiert werden – in der OCAP-2-Phase wurde nur `DriveFilter.c` dahingehend untersucht. Letztlich sollte insbesondere der Programmfluss mit besonderem Fokus auf dem Umgang mit Fehler-Codes und den daraus resultierenden Ausschlüssen bestimmter Funktionsaufrufe jenseits der eigentlichen kryptographischen Funktionen.

Darüber hinaus erscheint die Programmlogik sehr komplex und sollte vereinfacht werden. Insbesondere die Vielfalt an kryptographischen Funktionen und redundanten Implementierungen (siehe auch Kapitel 4 Abbildung 4.1) resultieren in einer hohen Komplexität. Der Bericht hebt die Implementierung von Hardwarebeschleunigung jedoch ausdrücklich als Ausnahme hervor. Ein weiteres Beispiel erhöhter Komplexität findet sich in Appendix D des Berichts, der auf die Nutzung von Devinsive-Coding-Praktiken hinweist. Fälle wie die präsentierten Fall-Through- und Default-lose Switch-Statements erhöhen die Komplexität für Programmierer und Reviewer deutlich.

Des Weiteren empfiehlt der Bericht bessere Fehlerbehandlung und Logging. Insbesondere der Fall der schlechten Fehlerbehandlung für Zufallsquellen gibt hier ein sehr gutes Beispiel. Ein Programmabbruch mit entsprechenden Fehlerausgaben hätte den Programmierer vermutlich dazu geführt, die unter Resultat 1 beschriebenen Verbesserungen bereits einzubauen. Das Logging hätte dazu vermutlich die nötigen Hinweise in der aktuellen Implementierung gegeben und der Programmabbruch die Notwendigkeit einer korrekten Lösung motiviert.

9.4 Zusammenfassung

Zusammenfassung der Ergebnisse aus Kapitel 9

- Der Random Number Generator unter Windows ist bei der Verwendung bestimmter Gruppenpolicies stark geschwächt. Dies kann bei vielerlei Systemen unbemerkt zu schwach verschlüsselten Volumes geführt haben. Dies sollte behoben und entsprechende Schlüssel neu generiert werden.
- Die Implementierung des Bootloaders für Full-Disk-Encryption ist anfällig für Cache-Timing Attacken. Dies ist jedoch nur im Falle der Nutzung in virtuellen Maschinen relevant und auch dann sehr aufwändig.
- Das Mixen von mehreren Keyfiles und/oder Passwort ist kryptographisch nicht sicher gegen Kollisionsattacken. Dies bricht das Erzwingen von Mehrfaktorauthentifikation und Mehr-Augen-Prinzipien.
- Die Volume-Header sollten statt per CRC innerhalb des Ciphertexts wahlweise per MAC oder CCM/GCM gesichert werden.
- Der Einsatz von XTS-AES ist mit Risiken verbunden, da ein Angreifer zielgerichtet bestimmte Speicherstellen überschreiben kann, ohne dass dies dem Benutzer auffallen muss.
- Vereinfachung der Programmlogik, Defensive Coding und bessere Fehlerbehandlung und -ausgabe würden es Programmieren und Reviewern erleichtern, die Sicherheit zu gewährleisten.
- Weitere Reviews insbesondere zur Pointerarithmetik der XTS-Implementierung, der Handhabung der Header-Volume-Parameter sowie des Programmflusses jenseits der kryptographischen Funktionen sollten folgen.

Anhang A

Funktionen mit zyklomatischer Komplexität über 15

```
========================================
  NLOC    CCN   token  PARAM  location
----------------------------------------
    1832    440   8579      4 MainDialogProc@5368-7770@./Format/Tcformat.c
    1536    416   8087      4 MainDialogProc@4637-6565@./Mount/Mount.c
    1542    368  12019      4 PageDialogProc@3292-5364@./Format/Tcformat.c
     604    138   3990      3 ProcessMainDeviceControlIrp@805-1553@./Driver/Ntdriver.c
     522    113   2956      5 TCOpenVolume@36-732@./Driver/Ntvol.c
     345    107   1760      1 TestSectorBufEncryption@637-1030@./Common/Tests.c
     306    103   1495      1 TestLegacySectorBufEncryption@1033-1363@./Common/Tests.c
     344    102   1898      0 EncryptionTest::TestXts@473-861@./Volume/EncryptionTest.cpp
     443    102   2374      1 TCFormatVolume@73-675@./Common/Format.c
     377     98   2751      2 CommandLineInterface::CommandLineInterface@20-483@./Main/CommandLineInterface.cpp
     368     87   2064      1 VolumeCreationWizard::ProcessPageChangeRequest@514-972@./Main/Forms/VolumeCreationWizard.cpp
     393     78   2491      4 PasswordChangeDlgProc@1381-1867@./Mount/Mount.c
     282     78   2128      1 TextUserInterface::CreateVolume@477-846@./Main/TextUserInterface.cpp
     332     74   1112      1 AfterWMInitTasks@8423-8914@./Format/Tcformat.c
     232     72   1399      9 MountVolume@5963-6263@./Common/Dlgcode.c
     245     70   1232      1 volTransformThreadFunction@2261-2576@./Format/Tcformat.c
     306     68   1478      2 RestoreVolumeHeader@7668-8074@./Mount/Mount.c
     291     67   1722      5 ReadVolumeHeader@163-575@./Common/Volumes.c
     324     63   1753      4 EncryptPartitionInPlaceResume@638-1102@./Format/InPlace.c
     241     59   1496      4 RawDevicesDlgProc@2869-3180@./Common/Dlgcode.c
     229     59   1123      5 ChangePwd@117-421@./Common/Password.c
     245     58   1550      1 SetupThreadProc@1150-1466@./Driver/DriveFilter.c
     266     57   1878      1 AnalyzeKernelMiniDump@8609-8940@./Mount/Mount.c
     165     56    927      2 MountAllDevices@3687-3915@./Mount/Mount.c
     212     55   1244      2 LoadPage@2578-2850@./Format/Tcformat.c
     120     55    706      2 TCDispatchQueueIRP@199-349@./Driver/Ntdriver.c
     155     54   1067      1 TCTranslateCode@1864-2024@./Driver/Ntdriver.c
     209     53   1638      1 MainThreadProc@482-756@./Driver/EncryptedIoQueue.c
     150     53   1183      4 PreferencesDlgProc@2190-2383@./Mount/Mount.c
     226     53   1300      2 ExtractCommandLine@6567-6845@./Mount/Mount.c
     225     51   1322      4 FavoriteVolumesDlgProc@101-407@./Mount/Favorites.cpp
     258     49   2002      4 VolumePropertiesDlgProc@2624-2968@./Mount/Mount.c
     268     49   2112      4 CipherTestDialogProc@5044-5400@./Common/Dlgcode.c
     287     48   1748      2 LoadDriveLetters@978-1334@./Mount/Mount.c
     229     47   1458      4 PasswordDlgProc@1876-2167@./Mount/Mount.c
      61     46    740      0 EncryptionTest::TestLegacyModes@42-113@./Volume/EncryptionTest.cpp
     161     45   1059      9 Volume::Open@100-308@./Volume/Volume.cpp
     162     45   1010      3 BackupVolumeHeader@7446-7665@./Mount/Mount.c
     115     44    697      3 Mount@3364-3530@./Mount/Mount.c
     415     44   1308      0 UserInterface::ProcessCommandLine@867-1323@./Main/UserInterface.cpp
     207     43   1220      4 HotkeysDlgProc@270-523@./Mount/Hotkeys.c
      54     43    393      1 Hotkey::GetVirtualKeyCodeString@62-127@./Main/Hotkey.cpp
      72     42    605      2 GetKeyName@42-120@./Mount/Hotkeys.c
     162     42   1230      1 ExceptionHandlerThread@1657-1855@./Common/Dlgcode.c
     157     42   1130      0 LoadLanguageFile@104-313@./Common/Language.c
     249     41   1815      3 ProcessVolumeDeviceControlIrp@493-802@./Driver/Ntdriver.c
     190     41   1200     16 CreateVolumeHeaderInMemory@685-983@./Common/Volumes.c
      62     40    549     11 StringFormatter::StringFormatter@15-83@./Main/StringFormatter.cpp
     181     40   1177      1 GraphicUserInterface::RestoreVolumeHeaders@1112-1359@./Main/GraphicUserInterface.cpp
     140     39    693      0 CheckMountList@4445-4632@./Mount/Mount.c
     145     38   1182      1 IoThreadProc@298-479@./Driver/EncryptedIoQueue.c
     164     37    602      1 SwitchWizardToSysEncMode@749-947@./Format/Tcformat.c
     191     37   1353      4 TravelerDlgProc@2971-3216@./Mount/Mount.c
     139     37   1160      3 CoreLinux::MountVolumeNative@290-473@./Core/Unix/Linux/CoreLinux.cpp
     108     37    575      1 TextUserInterface::MountVolume@1061-1194@./Main/TextUserInterface.cpp
     218     36   1051      3 EncryptPartitionInPlaceBegin@302-635@./Format/InPlace.c
      92     36    521      1 GraphicUserInterface::MountVolume@647-755@./Main/GraphicUserInterface.cpp
     205     35   1448      1 VolumeCreationWizard::GetPage@75-335@./Main/Forms/VolumeCreationWizard.cpp
     111     34    622      4 MountFavoriteVolumes@7143-7282@./Mount/Mount.c
      47     34    350      0 InitOSVersionInfo@2223-2276@./Common/Dlgcode.c
     161     34   1219      1 PerformBenchmark@4261-4504@./Common/Dlgcode.c
     134     34   1141      1 DoAutoTestAlgorithms@1366-1563@./Common/Tests.c
     152     34    745      1 CoreUnix::MountVolume@393-575@./Core/Unix/CoreUnix.cpp
     128     33    877      4 LanguageDlgProc@317-480@./Common/Language.c
      93     33    630      4 BootEncryption::CreateBootLoaderInMemory@961-1082@./Common/BootEncryption.cpp
     141     32   1067      4 MountOptionsDlgProc@2386-2573@./Mount/Mount.c
     136     32    752      4 PerformanceSettingsDlgProc@8083-8254@./Mount/Mount.c
     134     32    755      6 OpenVolume@8457-8637@./Common/Dlgcode.c
     118     32    753      4 GetAvailableHostDevices@9333-9483@./Common/Dlgcode.c
     156     32    986      1 TextUserInterface::RestoreVolumeHeaders@1263-1476@./Main/TextUserInterface.cpp
     138     31    643      2 CheckRequirementsForNonSysInPlaceEnc@88-299@./Format/InPlace.c
     141     31    685      0 RepairMenu@842-1025@./Boot/Windows/BootMain.cpp
     145     31   1074      5 FormatFat@256-445@./Common/Fat.c
     187     31   1068      4 SecurityTokenKeyfileDlgProc@9056-9286@./Common/Dlgcode.c
```

```
     188     30    780      2 ExtractCommandLine@7772-8015@./Format/Tcformat.c
     149     30   1060      5 Process::Execute@26-201@./Platform/Unix/Process.cpp
      89     29    468      2 handleError@3842-3945@./Common/Dlgcode.c
     128     29    827      1 GraphicUserInterface::BackupVolumeHeaders@109-283@./Main/GraphicUserInterface.cpp
     153     28   1139      1 CoreService::StartElevated@339-524@./Core/Unix/CoreService.cpp
      93     28    623      0 PlatformTest::TestAll@234-347@./Platform/PlatformTest.cpp
     137     27    861      4 KeyfileGeneratorDlgProc@4862-5036@./Common/Dlgcode.c
      81     27    429      0 GetWindowsEdition@8223-8321@./Common/Dlgcode.c
      86     26    515      2 SaveFavoriteVolumes@650-761@./Mount/Favorites.cpp
     165     26   1249      4 MultiChoiceDialogProc@5497-5716@./Common/Dlgcode.c
      93     25    563      7 GetDrivePartitions@359-472@./Boot/Windows/BootDiskIo.cpp
     105     25    688      4 RandomPoolEnrichementDlgProc@4715-4844@./Common/Dlgcode.c
      67     25    421      1 MainFrame::OnHotkey@884-965@./Main/Forms/MainFrame.cpp
      85     25    432      0 MainFrame::OnTimer@1215-1317@./Main/Forms/MainFrame.cpp
      88     25   1061      1 aes_init@274-421@./Crypto/Aestab.c
      66     24    341      2 VerifySizeAndUpdate@1255-1334@./Format/Tcformat.c
     105     24    631      1 DecryptDrive@699-839@./Boot/Windows/BootMain.cpp
      98     24    593      5 DismountAll@3562-3685@./Mount/Mount.c
      76     24    454      2 main@26-127@./Main/Unix/Main.cpp
      73     24    415      6 TextUserInterface::ChangePassword@360-452@./Main/TextUserInterface.cpp
      50     23    261      1 UpdateNonSysInPlaceEncControls@1835-1896@./Format/Tcformat.c
     105     23    659      5 AnalyzeHiddenVolumeHost@8066-8213@./Format/Tcformat.c
     145     23   1078      3 MountDrive@220-404@./Driver/DriveFilter.c
      76     23    480      0 BootMenu@481-572@./Boot/Windows/BootMain.cpp
     115     23    601      4 SecurityTokenPreferencesDlgProc@8257-8401@./Mount/Mount.c
     133     23    785      2 KeyFilesApply@217-387@./Common/Keyfiles.c
     137     23    842      2 InitApp@2281-2472@./Common/Dlgcode.c
     103     23    595      4 TextInfoDialogBoxDlgProc@2717-2843@./Common/Dlgcode.c
     108     23    832      0 PlatformTest::SerializerTest@26-160@./Platform/PlatformTest.cpp
      61     22    308      3 PrintFreeSpace@2853-2919@./Format/Tcformat.c
      75     22    314      0 main@1035-1151@./Boot/Windows/BootMain.cpp
     103     22    679      0 Int13Filter@26-177@./Boot/Windows/IntFilter.cpp
      76     22    602      3 LoadFavoriteVolumes@501-597@./Mount/Favorites.cpp
     152     22    979      4 KeyFilesDlgProc@418-607@./Common/Keyfiles.c
      58     22    579      1 UpdateProgressBarProc@61-130@./Common/Progress.c
      60     22    382      1 MountVolume@65-133@./Core/Unix/CoreServiceProxy.h
     112     22    700      2 CoreService::ProcessRequests@84-225@./Core/Unix/CoreService.cpp
      75     22    415      4 File::Open@185-279@./Platform/Unix/File.cpp
      90     22    550      0 VolumeCreationWizard::OnVolumeCreatorFinished@397-512@./Main/Forms/VolumeCreationWizard.cpp
      78     22    376      3 UserInterface::DismountVolumes@144-233@./Main/UserInterface.cpp
      82     22    469      1 UserInterface::MountAllDeviceHostedVolumes@572-671@./Main/UserInterface.cpp
      77     21    552      1 FinalPreTransformPrompts@3148-3250@./Format/Tcformat.c
      46     21    324      3 LoadImageNotifyRoutine@1035-1091@./Driver/DriveFilter.c
      87     21    508      2 CopySystemPartitionToHiddenVolume@577-693@./Boot/Windows/BootMain.cpp
      72     21    282      2 ChangeSysEncPassword@3960-4050@./Mount/Mount.c
      60     21    301      2 HandleHotKey@7353-7429@./Mount/Mount.c
     100     21    504      4 BootLoaderPreferencesDlgProc@8410-8531@./Mount/Mount.c
      83     21    791      1 GetFatParams@27-132@./Common/Fat.c
     148     21   1227      4 BenchmarkDlgProc@4507-4712@./Common/Dlgcode.c
     110     21    829      1 VolumeCreator::CreateVolume@177-330@./Core/VolumeCreator.cpp
      83     21    791      1 GetFatParams@43-148@./Core/FatFormatter.cpp
      75     21    474      1 ChangePasswordDialog::OnOKButtonClick@75-165@./Main/Forms/ChangePasswordDialog.cpp
      92     21    601      1 TextUserInterface::BackupVolumeHeaders@233-358@./Main/TextUserInterface.cpp
      63     20    524      3 VolumeHeader::Deserialize@134-216@./Volume/VolumeHeader.cpp
      65     20    495      3 MoveClustersBeforeThresholdInDir@1588-1676@./Format/InPlace.c
      83     20    379      3 QueryFreeSpace@3038-3145@./Format/Tcformat.c
      77     20    374      1 DriverAttach@3423-3543@./Common/Dlgcode.c
      83     20    466      5 WriteRandomDataToReservedHeaderAreas@1088-1196@./Common/Volumes.c
      79     20    435      2 TextUserInterface::AskPassword@84-181@./Main/TextUserInterface.cpp
     110     20    598      0 GraphicUserInterface::OnInit@768-913@./Main/GraphicUserInterface.cpp
      55     19    477      1 dynamic@329-398@./Boot/Windows/Decompressor.c
      78     19    368      0 DriverUnload@3319-3420@./Common/Dlgcode.c
      60     19    372      1 EncryptionThreadPoolStart@218-307@./Common/EncryptionThreadPool.c
      82     19    531      2 CoreMacOSX::MountAuxVolumeImage@109-211@./Core/Unix/MacOSX/CoreMacOSX.cpp
      21     19    189      0 ChangePasswordDialog::OnPasswordPanelUpdate@167-194@./Main/Forms/ChangePasswordDialog.cpp
      48     19    267      1 MainFrame::OnClose@726-785@./Main/Forms/MainFrame.cpp
      99     18    586      1 DecoySystemWipeThreadProc@1692-1821@./Driver/DriveFilter.c
      80     18    435      1 SlowPoll@535-651@./Common/Random.c
      59     18    255      0 cleanup@233-308@./Common/Dlgcode.c
      64     18    452      0 BootEncryption::GetSystemDriveConfiguration@847-927@./Common/BootEncryption.cpp
      98     18    703      0 VolumeCreator::CreationThread@44-175@./Core/VolumeCreator.cpp
      60     17    235      6 OpenPartitionVolume@1177-1247@./Format/InPlace.c
      84     17    457      2 DumpFilterEntry@20-137@./Driver/DumpFilter.c
     103     17    509      2 MountDevice@2511-2641@./Driver/Ntdriver.c
      52     17    335      3 BroadcastDeviceChange@5884-5949@./Common/Dlgcode.c
      83     17    468      0 BootEncryption::GetPartitionForHiddenOS@409-518@./Common/BootEncryption.cpp
     103     17    580      3 BootEncryption::ChangePassword@2050-2193@./Common/BootEncryption.cpp
      69     17    508      1 EncryptionThreadProc@123-215@./Common/EncryptionThreadPool.c
      84     16    471      6 EncryptionThreadPool::DoWork@25-133@./Volume/EncryptionThreadPool.cpp
      62     16    383      3 UnmountDevice@2643-2727@./Driver/Ntdriver.c
      45     16    271      3 HiberDriverEntryFilter@957-1014@./Driver/DriveFilter.c
      55     16    259      2 GetPoolBuffer@34-103@./Driver/EncryptedIoQueue.c
      82     16    609      1 EncryptedIoQueueStart@868-979@./Driver/EncryptedIoQueue.c
      81     16    463      1 DisplayHotkeyList@165-266@./Mount/Hotkeys.c
      62     16    508      1 InitMainDialog@220-304@./Mount/Mount.c
      62     16    225      1 DecryptSystemDevice@4100-4170@./Mount/Mount.c
      71     16    294      1 CreateRescueDisk@4196-4277@./Mount/Mount.c
      53     16    416      3 AddMountedVolumeToFavorites@31-98@./Mount/Favorites.cpp
     105     16    779      1 InitDialog@1106-1239@./Common/Dlgcode.c
      38     16    207      0 BootEncryption::CheckRequirements@1862-1912@./Common/BootEncryption.cpp
      68     16    462      3 GetArgumentID@130-210@./Common/Cmdline.c
      48     16    222      3 Wipe35Gutmann@83-139@./Common/Wipe.c
      59     16    300      3 CoreMacOSX::DismountVolume@32-100@./Core/Unix/MacOSX/CoreMacOSX.cpp
      83     16    510      0 MainFrame::UpdateVolumeList@1455-1561@./Main/Forms/MainFrame.cpp
      48     16    362      1 UserInterface::ExceptionToMessage@298-365@./Main/UserInterface.cpp
      46     16    385      1 UserInterface::ExceptionToString@367-436@./Main/UserInterface.cpp
==========================================================================================
Total nloc  Avg.nloc  Avg CCN  Avg token  Fun Cnt  Warning cnt   Fun Rt   nloc Rt
```

```
-----------------------------------------------------------------------------------------
      79853         20      4.68      135.88      3272          170       0.05     0.43
```

Anhang B

Codeduplikate (Exzerpt)

```
Boot/Windows/BootConsoleIo.cpp(50)
Boot/Windows/BootConsoleIo.cpp(35)
if (ScreenOutputDisabled)
return;
__asm
mov bx, 7
mov al, c

Boot/Windows/BootDiskIo.cpp(220)
Boot/Windows/BootDiskIo.cpp(143)
if (result == BiosResultEccCorrected)
result = BiosResultSuccess;
} while (result != BiosResultSuccess && --tryCount != 0);
if (!silent && result != BiosResultSuccess)

Boot/Windows/BootDiskIo.cpp(214)
Boot/Windows/BootDiskIo.cpp(136)
int 0x13
jnc ok
mov result, ah
ok:

Boot/Windows/BootDiskIo.cpp(200)
Boot/Windows/BootDiskIo.cpp(116)
BiosResult result;
byte tryCount = TC_MAX_BIOS_DISK_IO_RETRIES;
result = BiosResultSuccess;
__asm

Boot/Windows/BootMain.cpp(293)
Boot/Windows/BootConsoleIo.cpp(288)
__asm
push es
xor ax, ax
mov es, ax

Boot/Windows/Decompressor.c(142)
Boot/Windows/Decompressor.c(114)
local int decode(struct state *s, struct huffman *h)
    int len;
    int code;
    int first;
    int count;
    int index;

Boot/Windows/Decompressor.c(166)
Boot/Windows/Decompressor.c(127)
                return h->symbol[index + (code - first)];
            index += count;
            first += count;
            first <<= 1;
            code <<= 1;

Boot/Windows/IntFilter.cpp(525)
Boot/Windows/IntFilter.cpp(518)
popad
popf
leave
add sp, 2

Boot/Windows/IntFilter.cpp(616)
Boot/Windows/IntFilter.cpp(602)
mov ax, es:[si]
mov [di], ax
mov ax, es:[si + 2]
mov [di + 2], ax

Boot/Windows/IntFilter.cpp(154)
Boot/Windows/IntFilter.cpp(110)
IntRegisters.Flags &= ~TC_X86_CARRY_FLAG;
else
IntRegisters.Flags |= TC_X86_CARRY_FLAG;
passOriginalRequest = false;
break;

Boot/Windows/IntFilter.cpp(384)
Boot/Windows/IntFilter.cpp(318)
Print ("EAX:"); PrintHex (IntRegisters.EAX);
Print (" EBX:"); PrintHex (IntRegisters.EBX);
Print (" ECX:"); PrintHex (IntRegisters.ECX);
Print (" EDX:"); PrintHex (IntRegisters.EDX);
Print (" DI:"); PrintHex (IntRegisters.DI);

Boot/Windows/IntFilter.cpp(312)
Boot/Windows/IntFilter.cpp(46)
uint16 spdbg;
__asm mov spdbg, sp
PrintChar (' ');
PrintHex (spdbg);
PrintChar ('<'); PrintHex (TC_BOOT_LOADER_STACK_TOP);

Boot/Windows/Platform.cpp(49)
Boot/Windows/Platform.cpp(18)
__asm
jnc nocarry
mov carry, 1
nocarry:

Common/BaseCom.cpp(155)
Common/BaseCom.cpp(130)
catch (SystemException &)
return GetLastError();
catch (Exception &e)
e.Show (NULL);
return ERROR_EXCEPTION_IN_SERVICE;
catch (...)
return ERROR_EXCEPTION_IN_SERVICE;
return ERROR_SUCCESS;

Common/BaseCom.cpp(180)
Common/BaseCom.cpp(155)
catch (SystemException &)
return GetLastError();
catch (Exception &e)
e.Show (NULL);
return ERROR_EXCEPTION_IN_SERVICE;
catch (...)
return ERROR_EXCEPTION_IN_SERVICE;
return ERROR_SUCCESS;

Common/BaseCom.cpp(205)
Common/BaseCom.cpp(180)
catch (SystemException &)
return GetLastError();
catch (Exception &e)
e.Show (NULL);
return ERROR_EXCEPTION_IN_SERVICE;
catch (...)
return ERROR_EXCEPTION_IN_SERVICE;
return ERROR_SUCCESS;

Common/BaseCom.cpp(180)
Common/BaseCom.cpp(130)
catch (SystemException &)
return GetLastError();
catch (Exception &e)
e.Show (NULL);
return ERROR_EXCEPTION_IN_SERVICE;
catch (...)
return ERROR_EXCEPTION_IN_SERVICE;
return ERROR_SUCCESS;

Common/BaseCom.cpp(205)
Common/BaseCom.cpp(155)
catch (SystemException &)
return GetLastError();
catch (Exception &e)
e.Show (NULL);
return ERROR_EXCEPTION_IN_SERVICE;
catch (...)
return ERROR_EXCEPTION_IN_SERVICE;
return ERROR_SUCCESS;

Common/BaseCom.cpp(130)
Common/BaseCom.cpp(65)
catch (SystemException &)
```

```
return GetLastError();
catch (Exception &e)
e.Show (NULL);
return ERROR_EXCEPTION_IN_SERVICE;
catch (...)
return ERROR_EXCEPTION_IN_SERVICE;
return ERROR_SUCCESS;

Common/BaseCom.cpp(205)
Common/BaseCom.cpp(130)
catch (SystemException &)
return GetLastError();
catch (Exception &e)
e.Show (NULL);
return ERROR_EXCEPTION_IN_SERVICE;
catch (...)
return ERROR_EXCEPTION_IN_SERVICE;
return ERROR_SUCCESS;

Common/BaseCom.cpp(155)
Common/BaseCom.cpp(65)
catch (SystemException &)
return GetLastError();
catch (Exception &e)
e.Show (NULL);
return ERROR_EXCEPTION_IN_SERVICE;
catch (...)
return ERROR_EXCEPTION_IN_SERVICE;
return ERROR_SUCCESS;

Common/BaseCom.cpp(180)
Common/BaseCom.cpp(65)
catch (SystemException &)
return GetLastError();
catch (Exception &e)
e.Show (NULL);
return ERROR_EXCEPTION_IN_SERVICE;
catch (...)
return ERROR_EXCEPTION_IN_SERVICE;
return ERROR_SUCCESS;

Common/BaseCom.cpp(205)
Common/BaseCom.cpp(65)
catch (SystemException &)
return GetLastError();
catch (Exception &e)
e.Show (NULL);
return ERROR_EXCEPTION_IN_SERVICE;
catch (...)
return ERROR_EXCEPTION_IN_SERVICE;
return ERROR_SUCCESS;

Common/BootEncryption.cpp(2247)
Common/BootEncryption.cpp(2234)
BootEncryptionStatus encStatus = GetStatus();
if (encStatus.DriveMounted)
throw ParameterIncorrect (SRC_POS);
CheckRequirements ();

Common/BootEncryption.cpp(2360)
Common/BootEncryption.cpp(2344)
BootEncryptionStatus encStatus = GetStatus();
if (!encStatus.DeviceFilterActive || !encStatus.DriveMounted
|| encStatus.SetupInProgress)
throw ParameterIncorrect (SRC_POS);
BootEncryptionSetupRequest request;
ZeroMemory (&request, sizeof (request));

Common/BootEncryption.cpp(1151)
Common/BootEncryption.cpp(1129)
Device device (GetSystemDriveConfiguration().DevicePath);
byte mbr[TC_SECTOR_SIZE_BIOS];
device.SeekAt (0);
device.Read (mbr, sizeof (mbr));

Common/BootEncryption.cpp(1175)
Common/BootEncryption.cpp(1137)
device.SeekAt (0);
device.Write (mbr, sizeof (mbr));
byte mbrVerificationBuf[TC_SECTOR_SIZE_BIOS];
device.SeekAt (0);
device.Read (mbrVerificationBuf, sizeof (mbr));
if (memcmp (mbr, mbrVerificationBuf, sizeof (mbr)) != 0)
throw ErrorException ("ERROR_MBR_PROTECTED");

Common/BootEncryption.cpp(1323)
Common/BootEncryption.cpp(1265)
Device device (GetSystemDriveConfiguration().DevicePath);
byte mbr[TC_SECTOR_SIZE_BIOS];
device.SeekAt (0);
device.Read (mbr, sizeof (mbr));

Common/BootEncryption.cpp(1265)
Common/BootEncryption.cpp(1151)
Device device (GetSystemDriveConfiguration().DevicePath);
byte mbr[TC_SECTOR_SIZE_BIOS];
device.SeekAt (0);
device.Read (mbr, sizeof (mbr));

Common/BootEncryption.cpp(347)
Common/BootEncryption.cpp(238)
FILE_FLAG_RANDOM_ACCESS | FILE_FLAG_WRITE_THROUGH, NULL);
try
throw_sys_if (Handle == INVALID_HANDLE_VALUE);
catch (SystemException &)
if (GetLastError() == ERROR_ACCESS_DENIED && IsUacSupported())
Elevated = true;
else
throw;
FileOpen = true;
FilePointerPosition = 0;

[...]

Results:
  Lines of code: 48774
  Duplicate lines of code: 7091
  Total 1155 duplicate block(s) found.

  Time: 8.04078 seconds
```

Anhang C

Detail of the static analysis tools' warnings

Clang			
Kingdom	Warning type	N. of warnings	Security relevant
Input validation	malloc() size overflow	2	X
	Out of bound array access	4	X
API Abuse	Cast from non-struct type to struct type	55	
Code quality	Assigned value is garbage or undefined	1	
	Dead assignment	3	
	Dead initialisation	3	
	Dereference of null pointer	1	
	Uninitialized argument value	2	
	Use fixed address	1	

Cppcheck			
Kingdom	Warning type	N. of warnings	Security relevant
Input validation	Buffer access out of bounds	4	X
API Abuse	Memset with POD	2	
Code quality	C style cast	4	
	Redundant condition	1	
	Use Initialisation List	1	
	Uninitialised variable	33	
	Variable scope	80	
	Unassigned variable	4	
	Uninitialised member variable	26	
	Unused structure member	1	
	No copy constructor	1	
	Unread variable	4	
	Unused variable	3	
	Invalid printf argument type string	1	
	Invalid printf argument type int	2	
	Invalid scanf	1	
	Invalid scanf specific to libc	11	
	Wrong printf/scanf arguments	1	
	Clarify condition	2	
	Clarify calculation	3	
	Uninitialised data	4	
	Wrong copy of pointer	4	
	Use strcmp()	1	
	Redundant assignment	12	
	Bad use of c_str	2	
	Prefix operators for non primitive types	2	

Coverity Unix			
Kingdom	Warning type	N. of warnings	Security relevant
Input validation	Insecure data handling	2	X
	Integer handling issues	2	X
	Memory corruptions	1	X
API Abuse	API usage errors	1	
	Build system issues	2	
Time and state	Performance inefficiencies	1	
	Concurrent data access violations	1	
	Program hangs	3	
Code quality	Error handling issues	2	
	Null pointer dereferences	11	
	Parse warnings	4	
	Resource leaks	1	
	Uninitialised members	26	
	Various	1	

Coverity Windows			
Kingdom	Warning type	N. of warnings	Security relevant
Input validation	Insecure data handling	18	X
	Integer handling issues	1	X
	Memory corruptions	5	X
	Memory illegal accesses	7	X
	Various	1	X
API Abuse	API usage errors	1	
	Integer handling issues	3	
	Various	1	
Time and state	Program hangs	1	
	Security best practices	1	
Code quality	Code maintainability issues	1	
	Control flow issues	16	
	Error handling issues	19	
	Incorrect expression	6	
	Integer handling issues	1	
	Null pointer dereference	3	
	Performance inefficiencies	23	
	Parse warnings	3	
	Possible control flow issues	1	
	Resource leaks	13	
	Security best practices	33	
	Uninitialised variables	11	

Literatur

[1] Anders Bakken. *rtags Webseite (GIT-Hub)*. URL: https://github.com/Andersbakken/rtags.

[2] Alex Balducci, Sean Devlin und Tom Ritter. *Cryptographic Review*. Version 1.0. Open Crypto Audit Project, März 2015. URL: https://opencryptoaudit.org/reports/TrueCrypt_Phase_II_NCC_OCAP_final.pdf.

[3] Milan Broz und Vashek Matyas. »The TrueCrypt On-Disk Format-An Independent View«. In: *IEEE Security & Privacy* 12.3 (2014), S. 74–77. DOI: 10.1109/MSP.2014.60. URL: http://dx.doi.org/10.1109/MSP.2014.60.

[4] Cristian Cadar, Daniel Dunbar und Dawson Engler. »KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs«. In: *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*. OSDI'08. San Diego, California: USENIX Association, 2008, S. 209–224. URL: http://dl.acm.org/citation.cfm?id=1855741.1855756.

[5] Software Engineering Institute Carnegie Mellon University. *MSC06-CPP. Be aware of compiler optimization when dealing with sensitive data*. URL: https://www.securecoding.cert.org/confluence/display/cplusplus/MSC06-CPP.+Be+aware+of+compiler+optimization+when+dealing+with+sensitive+data.

[6] *Cscope (version 15.8a) Webseite (Sourceforge)*. URL: http://cscope.sourceforge.net/.

[7] Edsger W. Dijkstra. »Go To statement considered harmful«. In: *Comm. ACM* 11.3 (1968). letter to the Editor, S. 147–148.

[8] Niels Ferguson. *AES-CBC+ Elephant diffuser: A disk encryption algorithm for Windows Vista*. 2006.

[9] TrueCrypt Foundation. *TrueCrypt User's Guide. version 7.1a*. Englisch. Feb. 2012.

[10] TrueCrypt Foundation. *TrueCrypt Web-Seite*. URL: http://truecrypt.sourceforge.net/.

[11] Geoffrey K. Gill und Chris F. Kemerer. »Cyclomatic Complexity Density and Software Maintenance Productivity«. In: *IEEE Trans. Softw. Eng.* 17.12 (Dez. 1991), S. 1284–1288. ISSN: 0098-5589. DOI: 10.1109/32.106988. URL: http://dx.doi.org/10.1109/32.106988.

[12] Alex Hornung. *tcplay Webseite (GIT-Hub)*. URL: https://github.com/bwalex/tc-play.

[13] »IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices«. In: *IEEE Std 1619-2007* (2008), S. c1–32. DOI: 10.1109/IEEESTD.2008.4493450.

[14] Andreas Junestam und Nicolas Guigo. *TrueCrypt Security Assessment*. Version 1.1. Open Crypto Audit Project, März 2014. URL: https://opencryptoaudit.org/reports/iSec_Final_Open_Crypto_Audit_Project_TrueCrypt_Security_Assessment.pdf.

[15] Chris F. Kemerer. »Software complexity and software maintenance: A survey of empirical research«. English. In: *Annals of Software Engineering* 1.1 (1995), S. 1–22. ISSN: 1022-7091. DOI: 10.1007/BF02249043. URL: http://dx.doi.org/10.1007/BF02249043.

[16] Steve McConnell. *Code Complete, Second Edition*. Redmond, WA, USA: Microsoft Press, 2004. ISBN: 0735619670, 9780735619678.

[17] MITRE. *CVE-2014-4115*. Englisch. Juni 2014.

[18] MITRE. *CVE-2015-7358*. Englisch. Sep. 2015.

[19] MITRE. *CVE-2015-7359*. Englisch. Sep. 2015.

[20] Meiyappan Nagappan u. a. »An empirical study of goto in C code.« In: *PeerJ PrePrints* (2015). URL: https://dx.doi.org/10.7287/peerj.preprints.826v1.

[21] Katrina Tsipenyuk, Brian Chess und Gary McGraw. »Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors«. In: *IEEE Security & Privacy* 3.6 (2005), S. 81–84. DOI: 10.1109/MSP.2005.159. URL: http://doi.ieeecomputersociety.org/10.1109/MSP.2005.159.

[22] Meltem Sönmez Turan u. a. *Recommendation for Password-Based Key Derivation Part 1: Storage Applications*. Version Special Publication 800-132. NIST, 2010.

[23] Sven Türpe u. a. »Attacking the BitLocker Boot Process«. In: *Trusted Computing, 2nd International Conference, Trust 2009*. Hrsg. von Liqun Chen, Chris Mitchell und Andrew Martin. Bd. 5471. LNCS. Oxford, UK, April 6-8, 2009. Berlin / Heidelberg: Springer, 2009, S. 183–196. DOI: 10.1007/978-3-642-00587-9_12.

[24] Andrew Y. *Unofficial reference site of truecrypt.org*. 2014. URL: http://andryou.com/truecrypt/.