SSDs und Verschlüsselung: Datenremanenz als Problem

Carl-Daniel Hailfinger · Kerstin Lemke-Rust

Hochschule Bonn-Rhein-Sieg
{Carl-Daniel.Hailfinger,Kerstin.Lemke-Rust}@h-brs.de

Zusammenfassung

SSDs (Solid State Disks) werden zunehmend anstelle von klassischen Festplatten als Datenspeicher in Desktops und Laptops eingesetzt. Der in SSDs eingesetzte Flash Translation Layer (FTL) setzt Lese- und Schreibzugriffe des Betriebssystems in Zugriffe auf die Flashbausteine um, die in einer SSD als Datenspeicher dienen. Generell impliziert das Löschen oder Überschreiben von Daten der SSD durch das Betriebssystem keine entsprechende Operation auf den Flashbausteinen, sondern das FTL entscheidet nach intern vorgegebenen Kriterien über die Verweildauer alter Daten und Metadaten in den einzelnen Flashbausteinen. Dies führt zu einer vom Betriebssystem nicht sichtbaren Datenremanenz in den Flashbausteinen. Einige Sicherheitsaspekte generischer Datenträgerverschlüsselungslösungen beruhen auf der impliziten Annahme, dass der Datenträger keinerlei Datenremanenz zeigt. Die vorliegende Arbeit beschäftigt sich mit Angriffen, die auf SSDs durch die Verletzung dieser impliziten Annahme erst möglich werden. Konkret wird ein Angriff auf das Schlüsselmanagement von mit Linux LUKS/dm-crypt vollverschlüsselten Datenträgern gezeigt, bei dem ein längst nicht mehr gültiges Passwort zur Entschlüsselung der aktuellen Daten verwendet werden kann. Dieser Angriff ist nach aktuellem Stand der Technik nur mit Änderungen an SSDs selbst abzuwehren.

1 Grundlagen

1.1 SSDs

SSDs sind auf Flashspeicher basierende Datenträger, die sich im Vergleich zu klassischen magnetischen Festplatten (Hard Disk Drives, HDDs) durch höhere Lese- und Schreibgeschwindigkeiten, stark reduzierte Lese- und Schreiblatenz sowie Unempfindlichkeit gegenüber mechanischen Beanspruchungen auszeichnen. SSDs leiden allerdings im Gegensatz zu HDDs an Materialermüdung durch Schreibvorgänge, die sukzessive zu geringerer Zuverlässigkeit und schlussendlich Datenverlust führen kann. Es gibt SSDs in verschiedenen Bauformen, z.B. als USB-Sticks, 2,5-Zoll-Laufwerke sowie M.2-Steckkarten. Als Schnittstellen werden USB für USB-Sticks sowie SATA (Serial ATA) und NVMe (NVM Express) für 2,5-Zoll-Laufwerke und M.2-Steckkarten verwendet. Aus Kompatibilitätsgründen werden via USB und SATA angeschlossene SSDs wie klassische magnetische Datenträger angesteuert. Dies ermöglicht es dem Betriebssystem, ohne Änderung von magnetischen Festplatten (HDD, Hard Disk Drive) auf SSDs umzusteigen.

Aus Sicht der Endanwender und Administratoren ist eine SSD im Betrieb oftmals nichts anderes als eine schnellere Festplatte. Erst bei der Entsorgung werden bei entsprechendem Hintergrundwissen SSD-spezifische Löschmethoden angewandt, die im Allgemeinen eine vollständige Löschung der Daten einer SSD erreichen. Weitere bekannte notwendige Änderungen des Nut-

P. Schartner · A. Baumann (Hrsg.) · D•A•CH Security 2017 · syssec (2017) 338-349.

zungsprofils eines Datenträgers beim Wechsel von HDDs auf SSDs ist z.B. das Vermeiden sonstiger exzessiver Schreibzugriffe, da diese für eine beschleunigte Alterung der SSD sorgen [Sams].

Gegenüber einem Betriebssystem präsentieren sich SSDs im Fall von USB und SATA wie klassische Datenträger mit einem erweiterten optionalen Befehlssatz. Genau wie HDDs bieten sie dem Betriebssystem die Möglichkeit, logische Sektoren à 512 Byte oder 4096 Byte wahlfrei zu lesen und zu schreiben. Die logischen Sektoren werden mit einer Logical Block Address (LBA) in einem Bereich von 0 bis n-1 adressiert, wobei n die Anzahl der logischen Sektoren des Datenträgers ist. Im Allgemeinen unterstützen moderne SSDs noch weitere Befehle, wie z.B. einen Befehl zum vollständigen Löschen der SSD [Sams].

Intern setzten SSDs die LBA mittels eines Flash Translation Layer (FTL) auf physikalische Adressen in den einzelnen Flashbausteinen um. Die Firmware einer SSD, speziell der FTL, verwaltet das Mapping zwischen LBAs und physikalischen Adressen, wobei die physikalischen Adressen unabhängig von der LBA sind. Insbesondere ist das interne Mapping des FTL für das Betriebssystem nicht sichtbar [Webe16].

Der FTL muss dabei berücksichtigen, dass Schreibvorgänge gleichmäßig über den gesamten Flashspeicher verteilt werden (*Wear Leveling*) und dass Speicherbereiche mit veralteten Daten invalidiert und einer Wiederverwertung zugeführt werden (Garbage Collection). Wear Leveling ist notwendig, da Flashspeicher für nur eine begrenzte Anzahl von Schreibzyklen zuverlässig bleiben und sich das Risiko für Datenverlust durch jeden Schreibvorgang erhöht [APWD+08]. Garbage Collection wird durch die interne Organisation und die physikalischen Eigenschaften von Flashspeicher bedingt: Daten können nur in zuvor gelöschte Bereiche geschrieben werden, d.h. mehrfaches Schreiben in den gleichen Bereich ist nur nach zwischenzeitlichem Löschen möglich, und Löschvorgänge erfolgen auf *Eraseblocks*, die ein Vielfaches der Größe einer Page haben, welche die kleinste Schreib- und Leseeinheit ist. Die Anzahl der Lösch- und Schreibvorgänge wird daher durch den FTL soweit als möglich minimiert.

Je nach FTL befinden sich wenige bis erhebliche Mengen an veralteten Daten in den einzelnen Flashbausteinen, die mit entsprechender Software oder Hardware ausgelesen werden kann. Man spricht hier allgemein von Datenremanenz, die bei SSDs prinzipbedingt ist. Diese permanente (d.h. ohne äußere Einflüsse dauerhaft vorhandene) Datenremanenz mit dem exakten Erhalt ganzer Sektoren des Datenträgers ist zu unterscheiden von transienter Datenremanenz (z.B. bei Cold-Boot-Angriffen auf das RAM), bei der die Inhalte im Lauf der Zeit und abhängig von Umgebungseinflüssen schrittweise degenerieren, bis kein einziges Bit mehr wiederhergestellt werden kann.

Auch klassische magnetische HDDs zeigen in Ausnahmefällen Datenremanenz. Zwar gibt es keine Alterung durch Lese- und Schreibvorgänge, aber es ist durch Umgebungseinflüsse und/oder Fertigungstoleranzen möglich, dass einzelne Sektoren einer HDD Daten nicht zuverlässig speichern. Sofern die HDD einen solchen Zustand detektiert, werden die im betroffenen Sektor gespeicherten Daten sofern möglich gelesen und in einen der für diesen Zweck vorgehaltenen Reservesektoren geschrieben. Eine Löschung der alten Daten im betroffenen physikalischen Sektor erfolgt nicht. Weitere Lese- und Schreibzugriffe auf den betroffenen logischen Sektor erfolgen nicht mehr auf den inzwischen defekten physikalischen Sektor, sondern werden auf den neu zugeordneten physikalischen Reservesektor umgeleitet (sogenannte *Sector Remapping*). Beim ersten Auftreten von Sector Remapping sollte die HDD getauscht werden, da ein Totalausfall zu erwarten ist [PiWB07].

1.2 Datenträgerverschlüsselung

Verschlüsselung auf Datenträgern existiert in verschiedenen Formen: Es ist möglich, einzelne Dateien manuell zu verschlüsseln, die Verschlüsselung für einzelne Dateien transparent durch das Betriebssystem erledigen zu lassen, verschlüsselte Container mit ganzen Dateisystemen als Datei zu speichern, dem Datenträger intern die Verschlüsselung zu überlassen oder ganze Datenträger durch das Betriebssystem verschlüsseln zu lassen. Es gibt zwar Unterstützung für selbstverschlüsselnde Datenträger (Self-Encrypting Drive, SED) nach dem TCG Opal Standard in aktuellen Windows- und Linux-Versionen, diese ist aber nicht SSD-spezifisch.

Eine Datenträgerverschlüsselung durch das Betriebssystem ist z.B. unter Windows mit VeraCrypt/TrueCrypt und unter Linux mit LUKS/dm-crypt möglich. Diesen Verfahren ist gemein, dass sie den Datenträger als abstraktes Medium betrachten und insbesondere keine spezifischen Anpassungen für SSDs aufweisen. Des weiteren unterstützen die o.g. Produkte ein Schlüsselmanagement auf dem Datenträger selbst. Dabei wird für das Schlüsselmanagement ein Bereich auf dem Datenträger reserviert, der z.B. den mit einem aus dem Passwort abgeleiteten Schlüssel verschlüsselten Datenträgerschlüssel enthält. Insbesondere wird der Schlüssel nicht direkt aus dem Passwort generiert. Dies ermöglicht Passwortwechsel ohne Schlüsselwechsel. Die o.g. Produkte haben in ihrem Design die Annahme verankert, dass überschriebene Daten nicht wiederherzustellen sind. Auf diese Annahme wird im Abschnitt 3 eingegangen.

Exemplarisch gilt für Linux mit LUKS/dm-crypt, dass AES-256 in der Betriebsart XTS (Xor-Encrypt-Xor-based tweaked-codebook mode with ciphertext stealing) [AESXTS] verwendet wird. LUKS/dm-crypt ermöglicht einen Passwortwechsel ohne Schlüsselwechsel, d.h. der eigentliche Datenträgerschlüssel ändert sich dabei nicht. Für das Schlüsselmanagement ist ein Bereich am Anfang des Datenträgers (FDE Superblock) reserviert, der 8 Keyslots für jeweils mit einem z.B. aus einem Passwort abgeleiteten Schlüssel verschlüsselten Datenträgerschlüssel enthält. Aus einem Passwort abgeleitete Schlüssel werden mit dem PBKDF2-Algorithmus (Password-Based Key Derivation Function 2) [Kali00] berechnet. Es können bis zu 8 verschiedene Passwörter festgelegt werden, die unabhängig voneinander eine Entschlüsselung ermöglichen.

2 Verwandte Arbeiten

2.1 Datenremanenz auf SSDs

Es gibt zwei Arten von Datenremanenz auf SSDs: Bitweise Remanenz basiert auf minimalen Spuren alter Daten nach Löschvorgängen in Flashbausteinen, wobei einzelne Bits probabilistisch rekonstruiert werden konnten [Gutm01]. Auch mit fortschreitender Entwicklung von Flashspeicher blieb die bitweise Remanenz ein Problem [Skor05]. Als Seiteneffekt der Verbesserung der Löschverfahren und der internen Konditionierung der einzelnen Speicherzellen während eines Löschvorgangs ist bitweise Remanenz seit ca. 2015 auf gängigen SSDs kaum noch relevant [Parn16]. [Skor05] bezieht sich auf Designs, die Single-Level-Cell (SLC) Flashspeicher verwenden, wo nur zwei verschiedene Pegel in einer Speicherzelle gespeichert werden. Moderne SSDs verwenden aus Kostengründen aber Multi-Level-Cell (MLC) oder Triple-Level-Cell (TLC) Flashspeicher, bei dem respektive vier oder acht verschiedene Pegel gespeichert werden. Die Streuung der Eigenschaften der einzelnen Speicherzellen bei der Herstellung ist bei MLC und TLC wesentlich größer als jegliche mögliche Remanenz von Daten über einen Löschvorgang [YGSS+12], sodass auch dieser Umstand bitweise Remanenz als Angriff massiv erschwert.

Ansätze zur Lösung des Problems der seitenweisen (Page) Remanenz auf SSDs wurden u.a. von Grupp et al. [WGSS11] entwickelt, die Modifikationen des FTL vorschlagen, um Remanenz entweder komplett zu eliminieren oder zumindest auf ein kleines Zeitfenster zu beschränken. [WGSS11] weist auch auf den Nachteil hin, dass durch die vorgeschlagenen Ansätze die Schreiblast auf die SSD signifikant erhöht wird und Schreibdurchsatz sowie Latenz verschlechtert werden. Aus diesem Grund ist momentan nach Wissen der Autoren keine SSD auf dem Markt, die einen dieser Ansätze in der Praxis verwendet. Ferner existieren Befehle (SECURITY ERASE UNIT) in den ATA- und SCSI-Standards, die einen kompletten Datenträger löschen und dadurch das Problem der Datenremanenz zumindest für den Fall der Komplettlöschung beheben. Die Lage kurz nach der Einführung der entsprechenden Befehle wird in [SwWe10] beschrieben, wo ein erheblicher Teil der Hersteller Fehler in der Implementierung hatte. Wenige Jahre später war die korrekte Implementierung der Löschbefehle besser [WGSS11], aber noch nicht für alle auf dem Markt befindlichen SSDs umgesetzt.

Es gibt bei aktuellen SSDs Befehle, mit denen das Betriebssystem der SSD mitteilen kann, dass die Daten bestimmter logischer Sektoren nicht mehr vom Betriebssystem benötigt werden und die entsprechenden Bereiche für Wear Leveling bzw. Garbage Collection freigegeben sind. Bei NVMe ist dies das Deallocate Kommando, bei SATA ist es das in ACS-2 beschriebene ATA DATA SET MANAGEMENT TRIM Feature [ACS2], und für SCSI wird das in SBC-3 beschriebene UNMAP Kommando verwendet. Die durch diese Befehle ausgelöste Aktion ist nur ein Hinweis an die SSD-Firmware und hat nicht notwendigerweise eine Löschung der betreffenden Daten zur Folge. Zwar ist bei einem Großteil der SSDs das betreffende Feature so implementiert, dass die entsprechenden Daten nicht mehr durch das Betriebssystem ausgelesen werden können, eine Löschung der Daten im Flash erfolgt jedoch nicht zwingend [NiLR13]. Somit sind diese Befehle nicht wirksam gegen einen Angreifer, der Zugriff auf die Hardware hat bzw. am FTL vorbei den Flashspeicher auslesen kann.

Zugriffe am FTL vorbei sind bei USB-Sticks, SATA-SSDs sowie NVMe-SSDs nicht im Befehlssatz vorgesehen und erfordern entweder den physischen Zugriff auf den Flashspeicher oder den Einsatz alternativer SSD-Firmware. Neuere SATA- und NVMe-SSDs akzeptieren nur noch vom Hersteller kryptographisch signierte Firmware und erschweren damit den direkten Zugriff auf den Flashspeicher.

Eine neue Entwicklung stellt die LightNVM-Erweiterung [BjGB17] des NVMe-Standard dar, die explizit dem Betriebssystem direkten Zugriff auf den Flashspeicher ohne die oberen Ebenen des FTL ermöglicht. Das Betriebssystem ist hier zuständig für Wear Leveling, Garbage Collection und Mapping und somit werden Schreib- und Löschbefehle des Betriebssystems direkt an den Flashspeicher durchgereicht. Die Ansteuerung des Flashspeichers inklusive Fehlerkorrektur wird von der SSD-Firmware erledigt, da diese Umsetzung hochgradig hardwarespezifisch ist. SSDs, die klassisch im NVMe-Modus betrieben werden und LightNVM unterstützen, geben somit über den LightNVM-Befehlssatz einem Angreifer die Möglichkeit, am FTL vorbei alle noch physikalisch vorhandenen Daten auszulesen, selbst wenn diese normalerweise nicht mehr für das Betriebssystem sichtbar sind. LightNVM stellt nach Wissen der Autoren den einzigen öffentlich verfügbaren und standardisierten Befehlssatz zur Umgehung von FTL auf SSDs dar und ermöglicht daher auch ohne Eingriffe in die Hardware den in diesem Paper beschriebenen Angriff.

2.2 Angriffe auf Datenträgerverschlüsselung

2.2.1 Aktive und passive Angriffe auf Klartext und Chiffretext

Praktisch durchführbare Chosen Ciphertext Attacks (CCA) im Kontext von Datenträgerverschlüsselung lassen sich grob in drei Gruppen einteilen: Kopie eines Chiffretextsektors in einen Sektor einer anderen LBA, Änderung des Chiffretexts eines Sektors sowie Wiederherstellung des vorherigen Chiffretexts eines Sektors.

Das Vertauschen bzw. Kopieren von verschlüsselten Sektoren mit unterschiedlicher LBA kann z.B. durch die Verwendung des AES-XTS erschwert werden, indem die LBA als Tweak für AES-XTS verwendet wird [AESXTS]. Dadurch werden identische Klartextsektoren unterschiedlicher LBAs zu unterschiedlichen Chiffretexten verschlüsselt. Ein Vertauschen des Chiffretexts zweier Sektoren mit unterschiedlicher LBA ist damit möglich, aber der Angriff ist nicht besser als das Überschreiben des Zielsektors mit zufälligen Inhalten. Die Verwendung von Authenticated Encryption (AE) als Integritätsschutz ist alleine nicht hinreichend, da die Authentisierungsinformationen nicht notwendigerweise die LBA mit einbeziehen und auch nicht unbedingt anderweitig gegen Austausch gesichert sind. Sofern AE auf eine Art verwendet wird, die entweder die Authentisierungsinformationen abhängig von der LBA macht oder anderweitig absichert (Hashtree o.ä.), wird der Angriff zuverlässig erkannt.

Die (teilweise) Änderung des Chiffretexts eines Sektors zu einem Chiffretext, für den der Angreifer keine passenden Authentisierungsinformationen hat, wird bei Nutzung von AE erkannt. AES-XTS erreicht zumindest, dass der Klartext nicht bitweise verändert werden kann, sondern bei Veränderung des Chiffretextes blockweise nicht mehr sinnvoll entschlüsselbar ist. Dies kann bei ausreichend Informationen über den Klartext (z.B. bekannte Dateisystemstrukturen o.ä.) für eine Korrumpierung sicherheitskritischer verschlüsselter Komponenten des Systems verwendet werden, sodass das System hinterher zur Laufzeit Schwachstellen aufweist.

Die Wiederherstellung des alten Chiffretextes eines Sektors kann weder durch AES-XTS noch durch AE verhindert bzw. detektiert werden, es sei denn, AE wird zusätzlich abgesichert, und auch dann nur, wenn nicht der komplette Datenträger in einen alten Zustand (d.h. alte Chiffretexte eines bestimmten Zeitpunktes) versetzt wird.

Für Linux mit LUKS/dm-crypt gilt: Praktikable kryptoanalytische Angriffe auf AES sind nach aktuellem Kenntnisstand nicht bekannt und Brute-Force-Angriffe gegen den bei LUKS für das Passwort verwendeten PBKDF2-Algorithmus mit üblicherweise 10^5 - 10^6 Iterationen sind bei einem Passwort mit ausreichend Entropie ebenfalls nicht praktikabel [WaBr].

Für Datenträgerverschlüsselung relevante Known Ciphertext Attacks nutzen aus, dass frühere Lösungen zur Datenträgerverschlüsselung jeden Sektor mit einem identischen Schlüssel verschlüsselt haben. Effektiv handelt es sich bei solchen Lösungen um Blockchiffren im ECB-Modus (Electronic Code Book) mit einer Blockgröße von einem Sektor (512 oder 4096 Byte), unabhängig davon, welcher Verschlüsselungsalgorithmus intern verwendet wird. Entsprechend sind auch die bekannten Angriffe auf ECB anwendbar.

2.2.2 Selbstverschlüsselnde Datenträger

Angriffe auf fehlerhafte Implementierungen, ungeeignete Verschlüsselungsalgorithmen (XOR) und Designfehler selbstverschlüsselnder Datenträger sind sattsam bekannt. Exemplarisch seien eine Verschlüsselung mit konstantem XOR-Schlüssel sowie per verschlüsselter PIN gesi-

cherte Stromzufuhr ohne Datenverschlüsselung genannt. Eine gute Übersicht verschiedener Implementierungs- und Designfehler bietet [Rütt10]. Selbst bei selbstverschlüsselnden Datenträgern nach TCG Opal Standard sind mit physikalischem Zugriff bei einem Gerät, das sich im Ruhezustand (Suspend-to-RAM) oder im Betrieb befindet, verschiedene Angriffe möglich, die z.B. im laufenden Betrieb die Stromversorgung bestehen lassen und nur das SATA-Kabel in den Rechner des Angreifers umstöpseln (sogenannte Replugging-Angriff) oder ausnutzen, dass die Firmware des Rechners für das Aufwachen aus dem Ruhezustand den Freischaltcode bzw. das Passwort des SED intern vorhalten muss und dies auch vom Angreifer für die Freischaltung des SED genutzt werden kann [MüLF12].

2.2.3 Betriebssystembasierte Datenträgerverschlüsselung (FDE)

Es gibt mehrere Möglichkeiten, Angriffe zu klassifizieren: Entweder erfolgt eine Einteilung in Klassen nach der zeitlichen Vorgehensweise des Angreifers, oder nach dem Ziel des Angreifers. Zeitlich wird unterschieden zwischen Angriffen, die eine Manipulation des Systems durch den Angreifer mit anschließender Rückgabe an den legitimen Nutzer vorsehen (Evil Maid Angriffe [RuTe09]), und Angriffen, die einem einfachen Diebstahl des Systems entsprechen. Nach Zielen des Angreifers wird unterschieden zwischen unverschlüsseltem Teil des Datenträgers (Metadaten, Passwort, Schlüsselmaterial, Bootloader, Teile des Betriebssystems) und dem verschlüsselten Teil des Datenträgers. Eine harte Einteilung in Klassen ist allerdings nicht zielführend, da einige Angriffe mehrere Aspekte kombinieren.

Aktive Angriffe auf Hard- und Softwarekomponenten der FDE

Es gibt verschiedene Varianten des Evil Maid Angriffs, die bei einer nachfolgenden Nutzung durch den rechtmäßigen Benutzer das Schlüsselmaterial oder Passwort speichern und/oder direkt exfiltrieren. Im Allgemeinen wird dabei der Teil des Datenträgers manipuliert, der nicht verschlüsselt ist und z.B. Teile des Bootloaders oder Betriebssystems enthält. Der bekannteste Angriff dieser Art wurde von Rutkowska et al. 2009 umgesetzt [RuTe09] und sieht als ersten Schritt eine Manipulation, als zweiten Schritt eine Rückgabe, sowie als dritten Schritt eine Entwendung des Systems oder zumindest eine Kopie des abgefangenen Schlüsselmaterials vor. Köhler et al. [KoDo16] haben diesen Ansatz erweitert und eine Exfiltration durch das Schreiben des Schlüssels in einen unverschlüsselten Bereich vorgeschlagen, der im Normalbetrieb ebenfalls lesend zur Verfügung steht und daher auch anderweitig zugänglich sein könnte. Allen Angriffen dieser Klasse ist aber gemeinsam, dass eine Manipulation erfolgen muss, der eine reguläre Nutzung folgt. Erweiterte Varianten des Evil Maid Angriffs nutzen die Kooperation eines legitimen unprivilegierten Benutzers vor und/oder während der eigentlichen Manipulation, um die Manipulation bzw. die Exfiltration zu erleichtern.

Da der Schlüssel für den Datenträger durch das Betriebssystem vorgehalten werden muss, kann auch an diesem Punkt angesetzt werden. Im laufenden System kann der Schlüssel i.A. per DMA-Zugriff (Direct Memory Access) über geeignete Schnittstellen extrahiert werden, z.B. Firewire [BeDK05] [Böck09], PCI Express und Thunderbolt. Zwar ist es mit Hilfe einer IOMMU (IO Memory Management Unit) möglich, sich gegen solche Angriffe zu schützen, dies wird jedoch nur von einem Teil der aktuell verfügbaren Hardware unterstützt und muss ferner noch durch das Betriebssystem unterstützt und in der richtigen Konfiguration aktiviert werden, was ebenfalls nicht die Regel ist. Aus einem in den RAM suspendierten System können Schlüssel ebenfalls extrahiert werden und es gibt verschiedene Ansätze, dieses zu verhindern [HuHW17]. Datenremanenz im RAM, bei der die Inhalte im RAM auch nach Unterbrechung

der Stromversorgung (z.B. Ausschalten des Systems) noch eine gewisse Zeit verfügbar sind, wird in verschiedenen Varianten der Cold Boot Attack [HSHC+09] als Bypass genutzt.

Einen anderen Ansatz geht [BoVi15] mit der Suche nach Schwachstellen im Schlüsselmanagement von LUKS ohne Manipulation durch einen Angreifer. Die Verschlüsselungsparameter von LUKS können durch den Administrator so ungünstig gewählt werden, dass bekannte Eigenschaften des Klartextes in gewissen Szenarien Brute-Force-Angriffe erleichtern.

Passive Angriffe auf Hard- und Softwarekomponenten der FDE

Für klassische HDDs gilt, dass Angreifer, die den alten Zustand eines Datenträgers (teilweise) wiederherstellen wollen, den Datenträger bereits im Voraus manipulieren müssen, da ansonsten das Überschreiben eines logischen Sektors auch das Überschreiben des zugeordneten physikalischen Sektors zur Folge hat. Somit fällt für klassische HDDs eine Wiederherstellung alter Daten unter das Evil-Maid-Modell. Datenwiederherstellende Angriffe werden dementsprechend von den Entwicklern von LUKS/dm-crypt nicht als relevant angesehen, da mit gleichem Aufwand auch im klassischen Evil-Maid-Angriff Schlüsselmaterial extrahiert werden kann [Wagn10]. Die Besonderheit bei SSDs ist, dass für die Wiederherstellung alter Daten keine vorherige Manipulation durch eine Evil Maid wegen der prinzipbedingten Datenremanenz mehr notwendig ist, und ein einfacher Diebstahl des Datenträgers hinreichend ist. Dies hat zur Folge, dass das Angreifermodell von dm-crypt die aktuellen Möglichkeiten nicht widerspiegelt.

2.3 Verbesserung der Sicherheit von Verschlüsselung durch SSDs

Da bei den gängigen Lösungen für Festplattenverschlüsselung kein zusätzlicher Platz für Authentisierungsinformationen oder LBA-unabhängige Initialisierungsvektoren oder Tweaks vorhanden ist, gibt es nur einen möglichen Chiffretext je Klartext eines bestimmten Sektors. Ein neuer, nur bei SSDs umsetzbarer Ansatz wird von [KhMV17] vorgeschlagen: Ein sogenannter Diversifier mit wenigen Bits wird vom Betriebssystem als Teil des Initialisierungsvektors oder des Tweaks verwendet und für jeden Schreibvorgang neu gewählt. Um zu vermeiden, dass der Diversifier zusätzlichen Platz im Flashspeicher verbraucht, wird er als Metadaten des Schreibvorgangs vom Betriebssystem an die SSD übermittelt und in der SSD als zusätzlicher Input für der FTL verwendet, wo er einige Bits der physikalischen Speicheradresse festlegt. Damit kann der Diversifier bei einem Lesevorgang aus der physikalischen Adresse zurückgewonnen und dem Betriebssystem zurückübermittelt werden. Der Diversifier ist allerdings auch durch einen Angreifer auslesbar und kann insbesondere nicht die Wiederherstellung alter Daten verhindern.

3 Analyse und Angriff

Ziel ist, einen Angriff zu entwickeln, der darauf basiert, dass die Annahme der irrelevanten oder unmöglichen Wiederherstellung überschriebener Daten verletzt wird. Es handelt sich somit um eine Ausnutzung der prinzipbedingten Datenremanenz in SSDs. Im Gegensatz zu Evil Maid ist beim hier beschriebenen Angriff hinreichend, die SSD ohne Rückgabe zu entwenden, eine vorherige Manipulation ist nicht notwendig.

Als Plattform dient eine Standardinstallation eines aktuellen Ubuntu 16.04 LTS x86_64 mit LUKS/dm-crypt Datenträgervollverschlüsselung auf einer SSD. Ubuntu ist als verbreitetste generische Linux-Distribution ein realistisches Ziel.

Für eine bessere Reproduzierbarkeit wird ein SSD-Simulator verwendet, der die Inspektion der geschriebenen Daten zu jedem Zeitpunkt ermöglicht und über umfangreiche Protokollierungsmöglichkeiten verfügt.

Ein perfekter forensischer Datenträger hält Zeitpunkt und Inhalt jedes Schreibvorgangs fest und ermöglicht damit eine Rekonstruktion des Datenträgerinhalts zu jedem beliebigen Zeitpunkt. Eine klassische HDD hält weder den Zeitpunkt irgendeines Zugriffs fest noch verfügt sie über Daten, die zwischenzeitlich überschrieben wurden. Nutzungsmuster sind allerdings insofern sichtbar, als dass die Festplatte im Auslieferungszustand i.A. alle Sektoren nur mit Nullbytes gefüllt hat und somit z.B. von Nullbytes verschiedene Sektoren auf einen erfolgten Schreibzugriff schließen lassen. Bei SSDs hängt sowohl die Verfügbarkeit zwischenzeitlich überschriebener Daten und zugehöriger Metadaten als auch die Existenz von Schreibstatistiken vom FTL ab.

Anhand zweier in der SSD-Forschung häufiger verwendeten FTL-Implementierungen (FAST, DFTL) wird der konkrete Einfluss des FTL und die Umsetzbarkeit des Auslesens alter Daten und Metadaten untersucht.

3.1 Schlüsselmanagement von LUKS/dm-crypt

In großen Organisationen mit zentraler IT und Datenträgerverschlüsselung ist es nicht unüblich, ein für die Organisation konstantes und einfaches initiales Passwort für die Verschlüsselung der Datenträger aller Geräte zu wählen und den Benutzer zur sofortigen Änderung auf ein vom Benutzer gewähltes individuelles Passwort zu verpflichten. Die Wahl eines einfachen initialen Passworts wird aus Gründen der einfacheren telefonischen Übermittlung bzw. zur Vermeidung von Rückfragen der Benutzer beim Abtippen umgesetzt. Da eine sofortige Änderung nach der Auslieferung erfolgt, wird das Risiko als gering eingeschätzt. Ziel dieser Vorgehensweise ist, zu keinem Zeitpunkt unverschlüsselte Daten auf dem Datenträger gespeichert zu haben, dem Benutzer ohne Warten auf einen langwierigen Erstverschlüsselungsvorgang die Nutzung eines verschlüsselten Datenträgers zu ermöglichen, und zu verhindern, dass Administratoren und eventuell weitere Dritte Kenntnis des Datenträgerpassworts des Benutzers erlangen.

Da sich der Datenträgerschlüssel bei einem Passwortwechsel nicht ändert, sondern nur mit einem neuen vom Passwort abgeleiteten Schlüssel verschlüsselt und im entsprechenden Keyslot gespeichert wird, ist ein Angreifer bei Kenntnis des alten Keyslotinhalts und des alten Passworts in der Lage, die aktuellen Daten auf dem Datenträger zu entschlüsseln [WaBr].

Bei einem Datenträger ohne Remanenz (HDD) sorgt der Passwortwechsel für den unwiederbringlichen Verlust des alten Inhalts des Keyslots mit dem geänderten Passwort. Bei einem perfekten forensischen Datenträger sind die alten Inhalte des Slots ewig verfügbar und ermöglichen eine Entschlüsselung der aktuellen Daten mit Hilfe des alten organisationsweit einheitlichen Passworts, welches im Gegensatz zum vom Benutzer gewählten individuellen Passwort einem Angreifer eher bekannt sein könnte. Auf einer realen SSD hängt die Effektivität des Angriffs davon ab, wie das FTL agiert und wie das Neuschreiben des Schlüsselslots der LUKS-Metadaten implementiert ist. Da LUKS den AFsplit-Algorithmus zur Verteilung des Schlüssels auf mehrere Sektoren verwendet, ist ein Slot mehrere Sektoren lang und bereits der Verlust eines einzigen Sektors sorgt für einen kompletten Verlust des Schlüsselmaterials.

Der hier beschriebene Angriff setzt voraus, dass das initiale einfache FDE-Passwort dem Angreifer bekannt wird, entweder durch einen Brute-Force-Angriff oder durch Social Engineering.

3.2 Simulator

Um reproduzierbare Ergebnisse für verschiedene FTLs zu erzielen, wurde der SSD-Simulator flashsim [Bjør11] verwendet. flashsim ist weitgehend konfigurierbar bezüglich der Größe der simulierten SSD, der internen Organisation der SSD sowie dem verwendeten FTL. Alle Lese- und Schreibvorgänge des Experiments wurden in einem Tracefile aufgezeichnet. Dieses Tracefile wurde danach vom SSD-Simulator jeweils mit den beiden getesteten FTL ausgeführt. Der virtuelle Datenträger hatte eine logische Sektorgröße von 4096 Bytes, um eine direkte Umsetzung der logischen Sektoren in die effektive physikalische Seitengröße des Flashspeichers von 4096 Bytes möglich zu machen. Damit sind Seiteneffekte durch die Emulation kleinerer Sektoren ausgeschlossen und die Ergebnisse sind nur von der Größe der SSD, dem verwendeten FTL und dem Schreibmuster abhängig. Der Einfachheit halber wurde nur ein einziges Tracefile generiert, in dem die verschiedenen Phasen des Experiments annotiert waren. Dadurch war es möglich, aus einem Tracefile verschiedene Schreibmuster zu generieren, die sich in der Menge der vor dem bzw. nach dem Passwortwechsel geschriebenen Daten unterschieden.

3.2.1 Ablauf des Experiments

Ubuntu 16.04.2 LTS x86_64 wurde mit FDE in einer Minimalversion installiert. Die bis zum Ende der Installation geschriebene Datenmenge war 2622 MiB.

Nach der abgeschlossenen Installation wurde eine Datei mit der Größe des freien Platzes (6018 MB) auf dem Wurzel-Dateisystem geschrieben, gelöscht, nochmals geschrieben und wieder gelöscht, um vielfältige Schreibaktivitäten vor dem Wechsel des Datenträgerverschlüsselungspassworts zu simulieren. Diese Schreibvorgänge wurden als *pre-write* annotiert, um bei der späteren Ausführung des Tracefiles im SSD-Simulator die Menge der hier geschriebenen Daten begrenzen zu können, ohne mehrere Tracefiles zu benötigen.

Darauf erfolgte eine Änderung des Passworts. Für die hierbei überschriebenen logischen Sektoren (FDE Superblock mit Keyslots) wurde später im SSD-Simulator verfolgt, wie lange die ursprünglichen Superblock-Daten noch komplett auf der SSD verfügbar waren, d.h. wie lange eine Entschlüsselung des Datenträgers unter Umgehung des FTL möglich war.

Nach der Änderung des Passworts wurde analog zur Phase *pre-write* der freie Platz mehrmals vollgeschrieben und die Schreibvorgänge als *post-write* annotiert, um die Menge der geschriebenen Daten im SSD-Simulator begrenzen zu können.

Das so erhaltene Tracefile wurde parametrisiert mit der Menge an Schreibzugriffen (in MB) vor bzw. nach dem Passwortwechsel in flashsim ausgeführt. Ermittelt wurde, ob der alte Keyslot durch der FTL überschrieben oder gelöscht wurde und damit eine Entschlüsselung des Datenträgers mit dem alten Passwort unmöglich wurde.

3.2.2 FTL-Algorithmus FAST

FAST: A log buffer-based flash translation layer using fully-associative sector translation [LPCL+07] von Lee et al. ist einer der frühen FTL-Algorithmen, die als Basis für weitere Forschung gedient haben. Hier wurde er mit statischem Wear-Leveling genutzt.

Aufgrund der Art der Schreibzugriffe (lineares Schreiben des gesamten LUKS-Headers) gibt das Wear Leveling die veralteten Daten sehr zügig für eine Wiederverwendung frei. In der Grafik 1 ist schwarz gefärbt aufgetragen, bis zu welcher Schreibmenge jeweils vor und nach dem Passwortwechsel eine Wiederherstellung des alten Schlüsselmaterials möglich ist.

3.2.3 FTL-Algorithmus DFTL

DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings [GuKU09] von Gupta et al. versucht, durch selektives Caching die Performance vorheriger FTL-Algorithmen zu verbessern. Hier wird kein statisches Wear Leveling genutzt. Die Ergebnisse sind in Grafik 2 zu finden.

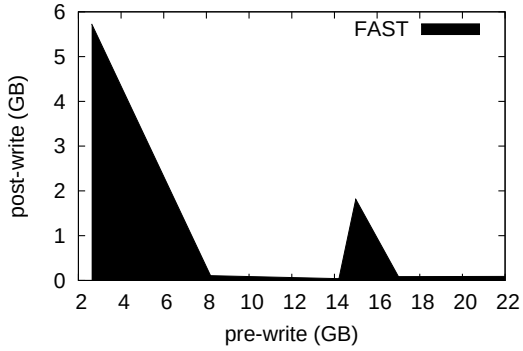

Abb. 1: Schlüsselverfügbarkeit bei FAST

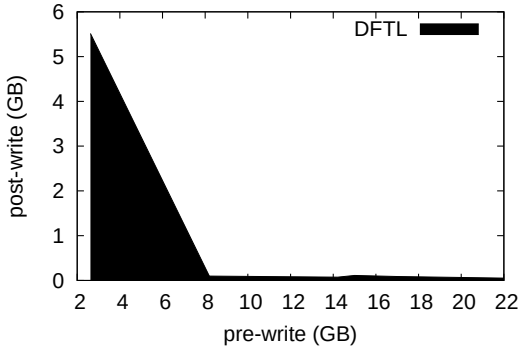

Abb. 2: Schlüsselverfügbarkeit bei DFTL

3.2.4 Ergebnisse

Für beide betrachteten FTLs gilt: Sobald die Gesamtmenge der Schreibzugriffe die Größe der SSD erreicht hat, startet die Garbage Collection und gibt alte Daten zum Überschreiben frei. Sofern die Menge der vor dem Passwortwechsel geschriebenen Daten geringer als die Datenträgergröße ist, erfolgt ein Überschreiben des LUKS-Headers frühestens nach dem Erreichen der Schreibmenge der Datenträgergröße. Da die FTLs auch die Anzahl der Schreibzyklen berücksichtigen und der ursprüngliche LUKS-Header durch die im Laufe der Installation frühe Erstellung an eine Stelle im Flashspeicher geschrieben wird, die vorher ungenutzt war, sind die Daten des alten LUKS-Headers spätestens dann ein Ziel der Garbage Collection, wenn ansonsten mehrfach beschriebener Flashspeicher der Garbage Collection zugeführt werden müsste. Statisches Wear Leveling kann diesen Mechanismus durch die Erhöhung der Schreibzyklen selten geschriebener Bereiche teilweise aushebeln. Bei dem hier betrachteten Ablauf der Operationen mit mehrfachen Vollschreiben des Dateisystems ist spätestens beim zweiten Vollschreiben des freien Platzes der Inhalt des ursprünglichen LUKS-Headers verloren.

Exemplarisch wurde für jedes FTL einmal geprüft, ob bei noch im Flashspeicher vorhandenen Daten des alten LUKS-Headers eine korrekte Entschlüsselung des LUKS/dm-crypt-Volume möglich war. Dies war jeweils erfolgreich.

4 Zusammenfassung und Ausblick

Auf einer simulierten SSD wurde gezeigt, dass abhängig von der geschriebenen Datenmenge vor und nach dem Passwortwechsel eine Wiederherstellung alten Schlüsselmaterials von LUKS/dm-crypt nach einem Passwortwechsel möglich ist und dies für die Entschlüsselung der aktuellen Daten ausreicht. Hier ist aber durch die Art des Schreibens beim Passwortwechsel als Seiteneffekt ein schnelles Recycling des Flashspeichers mit dem alten Schlüsselmaterial gegeben. Eine vollständige Sicherheit gegen diesen Angriff kann momentan nur durch einen Wechsel des Datenträgerschlüssels des LUKS/dm-crypt-Containers erreicht werden.

Für zukünftige Forschung wäre es interessant, dem Betriebssystem durch spezielle Kommandos die Möglichkeit zu geben, einem Datenträger mitzuteilen, welche Sektoren keinesfalls beim Überschreiben irgendwelche Remanenzen zeigen dürfen, sowie solche Mechanismen innerhalb einer SSD ohne negativen Einfluss auf Durchsatz und Lebensdauer zu entwickeln.

Literatur

[ACS2] Section 7.10.3.2: TRIM Command. In: *ATA/ATAPI Command Set 2* (2015).

[AESXTS] Standard for Cryptographic Protection of Data on Block-oriented Storage Devices, The XTS-AES Tweakable Block Cipher. In: *IEEE Std. 1619-2007* (2008).

[APWD+08] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, R. Panigrahy: Design Tradeoffs for SSD Performance. *In: 2008 USENIX Annual Technical Conference. Proceedings*, USENIX Association (2008), 57–70.

[BeDK05] M. Becher, M. Dornseif, C. N. Klein: FireWire: all your memory are belong to us. In: *Proceedings of CanSecWest* (2005).

[BjGB17] M. Bjørling, J. Gonzalez, P. Bonnet: LightNVM: The Linux Open-Channel SSD Subsystem. *In: 15th USENIX Conference on File and Storage Technologies, FAST 2017*, USENIX Association (2017), 359–374.

[Bjør11] M. Bjørling: Ext.FlashSim. In: *http://github.com/MatiasBjorling/flashsim* (2011).

[BoVi15] S. Bossi, A. Visconti: What Users Should Know About Full Disk Encryption Based on LUKS. *In: Cryptology and Network Security - 14th International Conference, CANS 2015*, Springer (2015), *LNCS*, Bd. 9476, 225–237.

[Böck09] B. Böck: Firewire-based Physical Security Attacks on Windows 7, EFS and BitLocker. In: *Secure Business Austria Research Lab* (2009).

[GuKU09] A. Gupta, Y. Kim, B. Urgaonkar: DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. *In: 14th Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2009*, ACM (2009), 229–240.

[Gutm01] P. Gutmann: Data Remanence in Semiconductor Devices. *In: Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, USENIX Association, Berkeley, CA, USA (2001).

[HSHC+09] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, E. W. Felten: Lest We Remember: Cold-boot Attacks on Encryption Keys. In: *Commun. ACM*, 52, 5 (2009), 91–98.

[HuHW17] M. Huber, J. Horsch, S. Wessel: Protecting Suspended Devices from Memory Attacks. *In: Proceedings of the 10th European Workshop on Systems Security, EUROSEC 2017* (2017), 10:1–10:6.

[Kali00] B. Kaliski: PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898, RFC Editor (2000).

[KhMV17] L. Khati, N. Mouha, D. Vergnaud: Full Disk Encryption: Bridging Theory and Practice. *In: The Cryptographers Track at the RSA Conference 2017, San Francisco, CA, USA, February 14–17, 2017, Proceedings* (2017), 241–257.

[KoDo16] T. Koehler, A. Doersam: Angriff auf verschluesselte Linux System-Partitionen. *In: P. Schartner (Hrsg.), DACH Security 2016*, Syssec (2016), 54–62.

[LPCL+07] S. Lee, D. Park, T. Chung, D. Lee, S. Park, H. Song: A log buffer-based flash translation layer using fully-associative sector translation. In: *ACM Trans. Embedded Comput. Syst.*, 6, 3 (2007), 18, .

[MüLF12] T. Müller, T. Latzo, F. Freiling: Self-encrypting disks pose self-decrypting risks. *In: Proceedings of the 29th Chaos Communication Congress* (2012), 27–30.

[NiLR13] A. Nisbet, S. Lawrence, M. Ruff: A forensic analysis and comparison of solid state drive data retention with trim enabled file systems. *In: The Proceedings of 11 th Australian Digital Forensics Conference* (2013), 103–111.

[Parn16] T. Parnell: NAND Flash Basics & Error Characteristics (2016), flash Memory Summit 2016, Santa Clara, CA, USA.

[PiWB07] E. Pinheiro, W. Weber, L. A. Barroso: Failure Trends in a Large Disk Drive Population. *In: 5th USENIX Conference on File and Storage Technologies, FAST 2007, February 13-16, 2007, San Jose, CA, USA*, USENIX (2007), 17–28.

[RuTe09] J. Rutkowska, A. Tereshkin: Evil maid goes after TrueCrypt. In: *Invisible Things Labs Blog, http://theinvisiblethings.blogspot.de/2009/10/evil-maid-goes-after-truecrypt.html* (2009).

[Rütt10] C. Rütten: Panzerknacker: Sicherheit von USB-Safes. In: *c't Magazin für Computer Technik*, 22 (2010), 158–162.

[Sams] Samsung: SSD FAQ. In: *http://www.samsung.com/global/business/semiconductor/minisite/SSD/M2M/html/support/faqs_03.html*, 2017-04-10.

[Skor05] S. P. Skorobogatov: Data Remanence in Flash Memory Devices. *In: J. R. Rao, B. Sunar (Hrsg.), Cryptographic Hardware and Embedded Systems - CHES 2005*, Springer (2005), *LNCS*, Bd. 3659, 339–353.

[SwWe10] S. Swanson, M. Wei: SAFE: Fast, Verifiable Sanitization for SSDs Or: Why encryption alone is not a solution for sanitizing SSDs (2010).

[WaBr] A. Wagner, M. Broz: cryptsetup Frequently Asked Questions. In: *https://gitlab.com/cryptsetup/cryptsetup/wikis/FrequentlyAskedQuestions*, 2017-04-03.

[Wagn10] A. Wagner: dm-crypt attacker model. In: *http://www.saout.de/pipermail/dm-crypt/2010-June/000857.html* (2010).

[Webe16] K. Weber: Über den Einfluss von SSD-Technologie auf die Datenträger-Forensik. Technische Berichte in Digitaler Forensik Nr. 9, Lehrstuhl für Informatik 1 der Friedrich-Alexander-Universität Erlangen-Nürnberg (2016).

[WGSS11] M. Wei, L. M. Grupp, F. E. Spada, S. Swanson: Reliably Erasing Data from Flash-based Solid State Drives. *In: Proceedings of the 9th USENIX Conference on File and Storage Technologies*, FAST'11, USENIX Association, Berkeley, CA, USA (2011), 105–117.

[YGSS+12] E. Yaakobi, L. M. Grupp, P. H. Siegel, S. Swanson, J. K. Wolf: Characterization and error-correcting codes for TLC flash memories. *In: ICNC*, IEEE Computer Society (2012), 486–491.