SYSTEMY LINUX
W KRYMINALISTYCE

PRAKTYCZNY PRZEWODNIK
DLA ANALITYKÓW ŚLEDCZYCH

BRUCE NIKKEL

Helion

no starch press

Tytuł oryginału: Practical Linux Forensics: A Guide for Digital Investigators

Tłumaczenie: Filip Kamiński

ISBN: 978-83-283-9404-9

Title of English-language original: Practical Linux Forensics: A Guide for Digital Investigators, ISBN 9781718501966, published by No Starch Press Inc.
245 8th Street, San Francisco, California United States 94103.

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/sylikr
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

3

Dowody znajdujące się na nośnikach pamięci i w systemach plików

W tym rozdziale skupię się na analizie kryminalistycznej pamięci masowej systemu Linux, w tym na tablicach partycji, zarządzaniu woluminami, RAID, systemach plików, partycjach wymiany, hibernacji oraz szyfrowaniu dysku. W każdym z tych obszarów możemy znaleźć specyficzne dla systemu Linux artefakty, które możemy przeanalizować. Choć większość przedstawionych w tym rozdziale czynności możesz wykonać za pomocą komercyjnych narzędzi kryminalistycznych, w przykładach zdecydowałem się zastosować narzędzia przeznaczone dla Linuksa.

Analizę kryminalistyczną pamięci masowej powinieneś rozpocząć od dokładnego ustalenia, co znajduje się na dysku. Musisz poznać układ, formaty, wersje i konfigurację dysków. Po ogólnym zorientowaniu się w zawartości dysku możesz przystąpić do poszukiwania innych interesujących Cię artefaktów kryminalistycznych oraz danych, które będziesz chciał poddać dalszym badaniom lub wyodrębnianiu.

W porównaniu z artykułami naukowymi i inną literaturą poświęconą informatyce śledczej analiza kryminalistyczna systemu plików przedstawiona w tym rozdziale jest opisana dość wysokopoziomowo. W tym rozdziale opiszę metadane i informacje dotyczące plików i systemów plików, które mogą być przydatne w trakcie dochodzenia. Pokażę, jak wyświetlać i wyodrębniać pliki, oraz wspomnę o szansach na odzyskanie usuniętych plików i o przestrzeniach luzu. Podczas analizy

zakłada się najczęściej, że systemy plików są w (stosunkowo) spójnym stanie, a narzędzia mogą analizować ich struktury danych. Naruszone, poważnie uszkodzone lub częściowo wymazane i nadpisane nośniki wymagają zastosowania innego podejścia, które obejmuje odzyskiwanie plików poprzez ponowne ręczne łączenie sektorów lub bloków oraz inne niskopoziomowe techniki. Zagadnienia te wykraczają poza zakres niniejszej książki. Świetne i dokładniejsze omówienie analizy systemów plików znajdziesz w książce *File System Forensic Analysis* Briana Carriera.

Podrozdział „Analiza kryminalistyczna systemów plików" rozpoczyna się od opisu struktur wspólnych dla wszystkich uniksopodobnych systemów. W dalszej części tego rozdziału przyjrzymy się najpopularniejszym systemom plików stosowanym w Linuksie: *ext4*, *xfs* oraz *btrfs*. Poświęcone im podrozdziały mają następującą strukturę:

- Historia, przegląd i funkcje.
- Jak znaleźć i zidentyfikować system plików.
- Artefakty kryminalistyczne w metadanych systemu plików (superbloki).
- Artefakty kryminalistyczne w metadanych plików (i-węzły).
- Wyświetlanie zawartości i wyodrębnianie plików.
- Inne unikalne cechy.

W przykładach zastosowałem The Sleuth Kit (TSK) oraz inne bezpłatne, otwarte i rozwijane przez społeczność narzędzia. TSK to rozwijany przez społeczność zestaw narzędzi do debugowania i rozwiązywania problemów. W kilku przykładach zastosowałem poprawioną wersję TSK, która obsługuje systemy *btrfs* i *xfs*.

W przykładach w tym rozdziale obrazy pełnych dysków nazywają się *image.raw*, a obrazy pojedynczych partycji (zawierających systemy plików) *partimage.raw*. Jeśli określisz odpowiednie przesunięcie, to polecenia z przykładów wykorzystujących obrazy partycji mogą zadziałać na pełnych obrazach dysków. Niektóre narzędzia działają tylko z urządzeniami i nie wspierają obrazów kryminalistycznych. W takich przypadkach tworzę urządzenie *loopback* skojarzone z takim obrazem.

Obecnie zbliżamy się do końca „złotego wieku" kryminalistyki systemów plików. W przypadku dysków magnetycznych po usunięciu pliku dane pozostają w fizycznych sektorach dysku. Usuwane są jedynie dowiązania do pliku, a zajmowane przez niego bloki oznaczane są jako niezaalokowane. Narzędzia kryminalistyczne potrafią „magicznie" odzyskać usunięte pliki i fragmenty częściowo nadpisanych plików. W odróżnieniu od dysków talerzowych dyski SSD wykonują polecenia TRIM i DISCARD wydawane przez system operacyjny. W ich efekcie (z przyczyn wydajnościowych) oprogramowanie układowe wymazuje nieużywane bloki. Ponadto warstwa translacji (ang. *Flash Translation Layer*, FTL) mapuje uszkodzone bloki pamięci w obszary, które nie są dostępne przez standardowe interfejsy sprzętowe (SATA, SAS lub NVMe). Z tego powodu niektóre tradycyjne techniki kryminalistyczne stają się mniej skuteczne. Techniki odzyskiwania, takie jak *chip off* (który polega na wylutowaniu układów pamięci), wymagają specjalnego sprzętu i szkolenia. W tym rozdziale opisałem odzyskiwanie usuniętych plików w sytuacjach, w których nadal możliwe jest użycie narzędzi programowych.

Analiza schematu pamięci i zarządzania woluminem

W tym podrozdziale opiszę, jak zidentyfikować partycje i woluminy systemu Linux na nośnikach pamięci. Pokażę też, jak zrekonstruować woluminy, które mogą zawierać systemy plików, oraz wyjaśnię, które informacje z nimi związane mogą okazać się przydatne w toku dochodzenia.

Analiza tablic partycji

Zazwyczaj podział pamięci na nośniku określa schemat partycjonowania (ang. *partition scheme*). Do najpopularniejszych schematów należą:

- DOS/MBR (oryginalny schemat partycjonowania w komputerach PC),
- GPT,
- BSD,
- Sun (*vtoc*),
- APM (Apple Partition Map),
- brak (brak schematu partycjonowania, systemy plików rozpoczyna się od zerowego sektora).

Przez wiele lat najpopularniejszym schematem partycjonowania był DOS, obecnie coraz powszechniej spotyka się schemat GPT.

Partycje definiuje się w tabeli partycji[1], która zawiera informacje o ich rodzaju, rozmiarze, przesunięciu itd. W systemie Linux dysk często dzieli się na partycje, aby stworzyć osobne systemy plików. Często spotykane partycje to:

- **/** — miejsce instalacji systemu operacyjnego, korzeń drzewa katalogów;
- ***ESP*** — partycja systemowa EFI (FAT) wykorzystywana do rozruchu z użyciem UEFI;
- ***swap*** — używana do stronicowania, wymiany i hibernacji;
- ***/boot/*** — informacje o bootloaderze, jądrach i początkowych ramdyskach;
- ***/usr/*** — czasami wykorzystywana do systemowych systemów plików tylko do odczytu;
- ***/var/*** — czasami wykorzystywana do przechowywania zmiennych lub zmieniających się danych systemowych;
- ***/home/*** — partycja zawierająca katalogi domowe użytkowników.

Domyślny układ partycji i systemu plików różni się w zależności od dystrybucji Linuksa, a użytkownik ma możliwość jego dostosowania podczas instalacji systemu.

[1] W systemach BSD/Solaris partycje określa się angielskim słowem *slices* (dosł. plastry).

W trakcie badania kryminalistycznego chcemy zidentyfikować schemat partycjonowania, przeanalizować tablice partycji i poszukać możliwych luk pomiędzy partycjami. Analiza tablic partycji DOS i GPT[2] jest niezależna od systemu operacyjnego. Wszystkie komercyjne narzędzia kryminalistyczne potrafią analizować linuksowe tablice partycji. Skoncentruję się tutaj na artefaktach specyficznych dla systemu Linux.

Wpis w tablicy partycji DOS zawiera jeden bajt określający typ partycji. Typy partycji DOS nie zostały określone autorytatywnie przez żaden organ, ale w internecie znajduje się utrzymywana przez społeczność lista znanych typów partycji (*https://www.win.tue.nl/~aeb/partitions/partition_types-1.html*, do listy tej odwołuje się nawet specyfikacja UEFI). Do najpopularniejszych typów partycji linuksowych zaliczamy:

- *0x83* — Linux,
- *0x85* — Linux extended,
- *0x82* — Linux swap,
- *0x8E* — Linux LV,
- *0xE8* — LUKS (Linux Unified Key Setup),
- *0xFD* — Linux RAID auto.

Prefiks `0x` oznacza, że typy partycji są zapisane w formacie szesnastkowym. Instalacje systemu Linux zawierają zazwyczaj jedną lub więcej **partycji podstawowych** (ang. *primary partitions*), które są reprezentowane przez tradycyjne wpisy w tablicy partycji. W tablicy może się też znajdować jedna **partycja rozszerzona** (ang. *extended partition*, bajt `0x05` lub `0x85`), która zawiera dodatkowe partycje logiczne[3].

Wpis w tablicy partycji GPT zawiera 16-bajtowy identyfikator GUID partycji. W specyfikacji UEFI napisano: „Dostawcy systemów operacyjnych muszą wygenerować własne identyfikatory GUID typu partycji, które pozwolą na identyfikację ich typów". Linux Discoverable Partitions Specification (*https://systemd.io/DISCOVERABLE_PARTITIONS/*) definiuje identyfikatory GUID kilku typów linuksowych partycji. Lista nie jest jednak pełna. Aby dowiedzieć się, jak wyświetlić identyfikator GUID, zajrzyj na stronę `systemd-id128(1)` manuala, którą możesz wyświetlić za pomocą polecenia `man 1 systemd-id128`. Poniżej pokazano identyfikatory GUID niektórych linuksowych typów partycji, które można znaleźć w tablicy partycji GPT:

- *Linux swap* — `0657FD6D-A4AB-43C4-84E5-0933C84B4F4F`.
- *System plików Linux* — `0FC63DAF-8483-4772-8E79-3D69D8477DE4`.
- *Linux root (x86-64)* — `4F68BCE3-E8CD-4DB1-96E7-FBCAF984B709`.

[2] Opublikowałem kiedyś artykuł szczegółowo opisujący tabele partycji GPT. Znajdziesz go pod adresem *https://digitalforensics.ch/nikkel09.pdf*.

[3] Partycje rozszerzone są rozwiązaniem, które pozwala na obejście ograniczenia liczby partycji w pierwotnym projekcie MBR do jedynie czterech.

- *Linux RAID* — `A19D880F-05FC-4D3B-A006-743F0F84911E`.
- *Linux LVM* — `E6D6D379-F507-44C2-A23C-238F2A3DF928`.
- *Linux LUKS* — `CA7D7CCB-63ED-4C53-861C-1742536059CC`.

Standardowy identyfikator GUID typu partycji to nie to samo co losowo wygenerowany, unikalny identyfikator GUID partycji lub systemu plików.

Typy partycji DOS lub GPT mogą wskazywać na zawartość partycji podczas badania kryminalistycznego. Musisz jednak uważać, ponieważ użytkownicy mogą zdefiniować dowolny typ partycji, a następnie stworzyć na niej zupełnie inny system plików. Typ partycji jest wykorzystywany jako wskaźnik dla różnych narzędzi, ale nie ma gwarancji co do jego poprawności. Jeżeli typ partycji różni się od oczekiwanego, to może być to próba ukrycia lub zaciemnienia danych (podobna do próby ukrycia typu pliku przez zmianę jego rozszerzania).

W systemie Linux wykryte partycje znajdują się w katalogu */dev/*. Jest to pseudokatalog zamontowany w działającym systemie. Podczas analizy kryminalistycznej *post mortem* katalog ten będzie pusty. W takim przypadku nazwy zamontowanych urządzeń będziesz mógł odnaleźć w logach, plikach konfiguracyjnych lub w innych plikach w systemie plików.

W systemach Linux najczęściej wykorzystuje się urządzenia SATA, SAS, NVMe i karty SD. Te urządzenia to urządzenia blokowe, które w działającym systemie są reprezentowane w katalogu */dev/* zgodnie z poniższą konwencją nazewnictwa:

- */dev/sda*, */dev/sdb*, */dev/sdc*, …
- */dev/nvme0n1*, */dev/nvme1n1*, …
- */dev/mmcblk0*, */dev/mmcblk1*, …

Na jeden dysk przypada jeden plik urządzenia. Dyski SATA i SAS są nazywane w porządku alfabetycznym (*sda*, *sdb*, *sdc*, …). Dyski NVMe są reprezentowane za pomocą liczb; pierwsza liczba oznacza numer dysku, a druga (po *n* na powyższej liście) przestrzeń nazw[4]. Karty SD również są reprezentowane w sposób numeryczny (*mmcblk0*, *mmcblk1*, …).

Jeśli Linux wykryje obecność partycji na określonym dysku, to w systemie utworzone zostaną dodatkowe pliki urządzeń, które będą je reprezentować. Nazwa partycji składa się najczęściej z nazwy urządzenia, do której dołączono dodatkowo numer lub literę *p* i dodatkową liczbę, np.:

- */dev/sda1*, */dev/sda2*, */dev/sda3*, …
- */dev/nvme0n1p1*, */dev/nvme0n1p2*, …
- */dev/mmcblk0p1*, */dev/mmcblk0p2*, …

[4] Napisałem kiedyś artykuł o analizie śledczej NVMe. Znajdziesz go pod adresem *https://digital-forensics.ch/nikkel16.pdf*.

Jeśli Twoje komercyjne narzędzie kryminalistyczne nie jest w stanie poprawnie przeanalizować tablicy partycji Linuksa lub jeśli potrzebujesz dodatkowych wyników, możesz skorzystać z kilku dostępnych w Linuksie narzędzi, w tym z `mmls` (z TSK) oraz `disktype`.

Oto przykład użycia polecenia `mmls` (z TSK) na tablicy partycji Manjaro Linux:

```
$ mmls image.raw
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  000:000   0000002048   0024188109   0024186062   Linux (0x83)
003:  000:001   0024188110   0041929649   0017741540   LinuxSwap/Solarisx86(0x82)
004:  -------   0041929650   0041943039   0000013390   Unallocated
```

Narzędzie `mmls` wyświetla listę różnych „slotów", którymi mogą być metadane partycji, nieprzydzielone obszary (w tym przerwy między partycjami) i rzeczywiste partycje. Początek (*Start*), koniec (*End*) i długość (*Length*) partycji są zapisane za pomocą liczby sektorów, których długość wynosi 512 bajtów. W tym przykładzie widoczny jest tradycyjny schemat partycjonowania DOS obejmujący partycję Linuksa (0x83) rozpoczynającą się w sektorze 2048 oraz partycję wymiany znajdującą się bezpośrednio po niej. Ostatnich 13 390 sektorów nie jest przydzielonych do żadnej partycji.

WSKAZÓWKA *Uważaj na jednostki. Niektóre narzędzia wykorzystują liczby sektorów, inne używają bajtów.*

Teraz spójrz na przykład wyjścia z `disktype` dla tablicy partycji systemu Linux Mint[5]:

```
# disktype /dev/sda

 --- /dev/sda
Block device, size 111.8 GiB (120034123776 bytes)
DOS/MBR partition map
Partition 1: 111.8 GiB (120034123264 bytes, 234441647 sectors from 1) ❶
  Type 0xEE (EFI GPT protective)
GPT partition map, 128 entries
  Disk size 111.8 GiB (120034123776 bytes, 234441648 sectors)
  Disk GUID 11549728-F37C-C943-9EA7-A3F9F9A8D071
```

[5] Znak # oznacza, że użytkownik jest zalogowany jako root — *przyp. tłum.*

```
Partition 1: 512 MiB (536870912 bytes, 1048576 sectors from 2048)
  Type EFI System (FAT) (GUID 28732AC1-1FF8-D211-BA4B-00A0C93EC93B) ❷
  Partition Name "EFI System Partition"
  Partition GUID EB66AA4C-4840-1E44-A777-78B47EC4936A
  FAT32 file system (hints score 5 of 5)
    Volume size 511.0 MiB (535805952 bytes, 130812 clusters of 4 KiB)
Partition 2: 111.3 GiB (119495720960 bytes, 233390080 sectors from 1050624)
  Type Unknown (GUID AF3DC60F-8384-7247-8E79-3D69D8477DE4)
  Partition Name "" ❸
  Partition GUID A6EC4415-231A-114F-9AAD-623C90548A03
  Ext4 file system
    UUID 9997B65C-FF58-4FDF-82A3-F057B6C17BB6 (DCE, v4)
    Last mounted at "/"
    Volume size 111.3 GiB (119495720960 bytes, 29173760 blocks of 4 KiB)
 Partition 3: unused
```

Na powyższym listingu symbolem ❶ zaznaczono partycję GPT pełniącą rolę chronionego MBR (typ `0xEE`). Partycja 1 to partycja EFI FAT (❷). Narzędzie ustaliło to na podstawie jej identyfikatora UUID (`GUID`). UUID partycji 2 (❸) nie został rozpoznany przez narzędzie `disktype`, ale narzędziu udało się zidentyfikować system plików i wyświetlić pewne informacje na jego temat.

Formatowanie identyfikatorów GPT UUID w różnych narzędziach może być różne. Identyfikator może mieć inny format niż wartość przechowywana na dysku. Poniżej pokazano sposób wyświetlania identyfikatora GPT partycji linuksowej (`0FC63DAF-8483-4772-8E79-3D69D8477DE4`) w kilku różnych narzędziach:

- `fdisk/gdisk`: `0FC63DAF-8483-4772-8E79-3D69D8477DE4`
- `disktype`: `AF3DC60F-8384-7247-8E79-3D69D8477DE4`
- `hexedit`: `AF 3D C6 0F 83 84 72 47 8E 79 3D 69 D8 47 7D E4`
- `xxd`: `af3d c60f 8384 7247 8e79 3d69 d847 7de4`

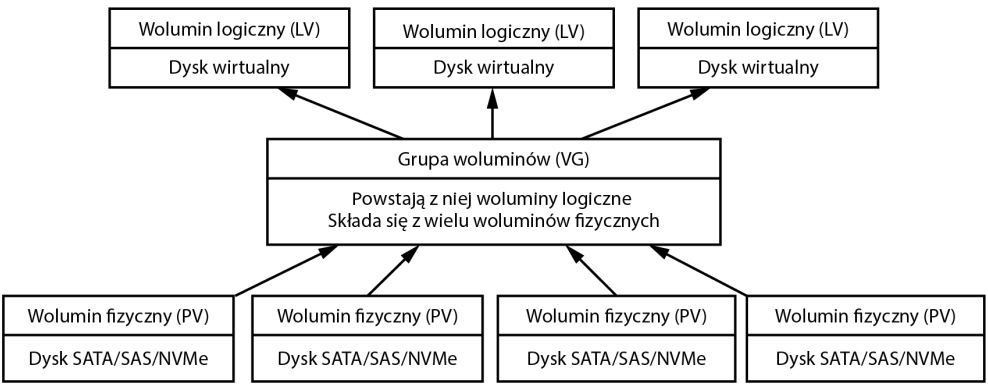
GPT UUID ma jasno określoną strukturę, a jego części są przechowywane na dysku w kodowaniu *little endian*. Specyfikacja UEFI (dodatek A) opisuje szczegóły formatu EFI GUID (*https://uefi.org/sites/default/files/resources/UEFI_Spec_2_8_final.pdf*). Niektóre narzędzia (na przykład `disktype` lub narzędzia do wyświetlania wartości szesnastkowych) zamiast interpretować bajty jako identyfikatory GPT UUID, mogą po prostu prezentować na ekranie nieprzetworzone bajty znajdujące się na dysku.

Zarządca woluminów logicznych

Nowoczesne systemy operacyjne zapewniają usługi zarządzania woluminami pozwalające tworzyć logiczne (wirtualne) dyski, których zawartość znajduje się na kilku urządzeniach fizycznych. Zarządzanie woluminami może być oddzielnym mechanizmem, takim jak linuksowy zarządca woluminów logicznych (ang. *Logical Volume Manager*, LVM), lub może być wbudowane bezpośrednio w system plików, tak jak ma to miejsce w systemach *btrfs* i *zfs*.

Przykłady pokazane w tym punkcie dotyczą uproszczonej konfiguracji LVM, w której występuje tylko jedno urządzenie fizyczne. Taka konfiguracja wystarczy do przeanalizowania sytuacji znanej z wielu dystrybucji Linuksa, które standardowo instalują LVM, nawet jeżeli w systemie znajduje się tylko jeden dysk twardy. W bardziej złożonych przypadkach obejmujących wiele dysków potrzebne będą narzędzia kryminalistyczne obsługujące woluminy LVM lub komputer kryminalistyczny pracujący pod kontrolą systemu Linux, który będzie w stanie uzyskać dostęp do woluminów LVM. Z narzędzi kryminalistycznych, które nie obsługują LVM, możesz skorzystać, jeśli znasz przesunięcie (offset) dla systemu plików oraz gdy system plików jest zapisany w postaci liniowej sekwencji sektorów na pojedynczym dysku.

Najpopularniejszym zarządcą woluminów w systemie Linux jest LVM. Jego wysokopoziomową architekturę pokazano na rysunku 3.1.



Rysunek 3.1. Zarządca woluminów logicznych

Z zarządcą LVM wiąże się kilka kluczowych pojęć:

- **Wolumin fizyczny** (ang. *Physical Volume*, *PV*) — fizyczne urządzenie przechowujące dane (dysk SATA, SAS lub NVMe).
- **Grupa woluminów** (ang. *Volume Group*, *VG*) — grupa składająca się z kilku woluminów fizycznych.
- **Wolumin logiczny** (ang. *Logical Volume*, *LV*) — wirtualne urządzenie magazynujące stworzone w grupie woluminów.
- **Ekstenty fizyczne** (ang. *Physical Extends*, *PEs*) — sekwencje kolejnych sektorów na woluminie fizycznym.
- **Ekstenty logiczne** (ang. *Logical Extends*, *LEs*) — sekwencje kolejnych sektorów na woluminie logicznym.

Ekstenty z LVM są podobne do tradycyjnych bloków znanych z systemów plików. Ekstenty mają stały rozmiar określony podczas ich tworzenia. Standardowy rozmiar ekstentu LVM to 8192 sektory (4 MB). Jest on stosowany w obu

rodzajach ekstentów. LVM może również zapewniać redundancję oraz stripping dla woluminów logicznych.

W przypadku użycia LVM nie ma konieczności tworzenia tablicy partycji, wolumin fizyczny może być utworzony bezpośrednio na niepodzielonym na partycje surowym dysku. Jeśli dysk jest podzielony na partycje, to LVM umieszcza w polu typu wpisu informację, że dysk fizyczny pełni rolę woluminu fizycznego. W przypadku schematu partycjonowania DOS kod partycji LVM to `0x8E`. W przypadku schematu GPT identyfikator UUID partycji LVM to `E6D6D379-F507-44C2-`
`↪A23C-238F2A3DF928` (niektóre narzędzia mogą wyświetlać tę wartość w postaci bajtów, które są przechowywane na dysku, tj. jako `D3 79 E6 D6 F5 07 44 C2 3C A2 8F 23 3D 2A 28 F9`). Poniżej pokazano przykładową tablicę partycji:

```
$ sudo mmls /dev/sdc
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors
      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000002047   0000002048   Unallocated
002:  000:000   0000002048   0002099199   0002097152   Linux (0x83)
003:000:001     0002099200   0117231407   0115132208   Linux Logical Volume Manager(0x8e)
```

W tym przykładzie `mmls` wyświetla tablicę partycji DOS. Na listingu widać, że partycja LVM rozpoczyna się w sektorze `2099200` i zajmuje większą część dysku.

Informacje o woluminie fizycznym są zapisywane w 32-bajtowej etykiecie nagłówka znajdującej się w drugim sektorze partycji LVM (sektor 1). Etykieta zawiera następujące informacje:

- identyfikator LVM, czyli ciąg znaków `LABELONE` (8 bajtów);
- sektor w partycji, w którym znajduje się etykieta (8 bajtów);
- sumę kontrolną CRC pozostałej części tego sektora (4 bajty);
- przesunięcie bajtowe (offset) początku treści (4 bajty);
- typ LVM, czyli ciąg znaków `LVM2 001` (8 bajtów),
- UUID woluminu fizycznego (16 bajtów).

Oto przykładowa zawartość etykiety znajdującej się na początku (drugi sektor) partycji LVM. Wartości są zapisane szesnastkowo:

```
40100200    4C 41 42 45 4C 4F 4E 45    01 00 00 00 00 00 00 00 LABELONE........
40100210    53 BF 78 2F 20 00 00 00    4C 56 4D 32 20 30 30 31 S.x/...LVM2001
40100220    55 77 37 73 73 53 4A 61    50 36 67 43 44 42 4D 61 Uw7ssSJaP6gCDBMa
40100230    51 32 4A 57 39 32 71 6F    66 71 59 47 56 57 6F 68 Q2JW92qofqYGVWoh
...
```

Do zarządzania woluminami LVM niezbędny jest pakiet `lvm2`. Zawiera on szereg narzędzi, które mogą pomóc w analizie kryminalistycznej dysków LVM. Bardziej szczegółowy opis systemu LVM znajdziesz na stronie `lvm(8)` manuala.

Narzędzia LVM działają na urządzeniach, a nie na zwykłych plikach. Aby przeanalizować konfigurację LVM na kryminalistycznej stacji roboczej pracującej pod kontrolą systemu Linux, musisz do niej podłączyć badane dyski za pomocą blokera zapisu lub zamontować je jako obrazy powiązane z urządzeniem typu *loop* w trybie tylko do odczytu (patrz punkt „Urządzenia" w rozdziale 2.). W poniższych przykładach badany dysk został zamontowany na maszynie kryminalistycznej jako */dev/sdc*.

Informacje o woluminach fizycznych można wyświetlić za pomocą narzędzia `pvdisplay`. Flaga `--foreign` pozwala uwzględnić w wyjściu również woluminy, które normalnie zostałyby pominięte, a opcja `--readonly` powoduje odczyt danych bezpośrednio z dysku (ignoruje sterownik mapowania urządzeń jądra):

```
$ sudo pvdisplay --maps --foreign --readonly
--- Physical volume ---
PV Name               /dev/sdc2
VG Name               mydisks
PV Size               <54.90 GiB / not usable <4.90 MiB
Allocatable           yes
PE Size               4.00 MiB
Total PE              14053
Free PE               1
Allocated PE          14052
PV UUID               Uw7ssS-JaP6-gCDB-MaQ2-JW92-qofq-YGVWoh

  --- Physical Segments ---
...
  Physical extent 1024 to 14051:
    Logical volume      /dev/mydisks/root
    Logical extents     0 to 13027
...
```

Powyższe dane wyjściowe zawierają informacje o pojedynczym woluminie fizycznym (*sdc2*), w tym rozmiar pojedynczego ekstentu fizycznego (*PE Size*), liczbę ekstentów fizycznych na woluminie (*Total PE*) oraz inne informacje o ekstentach. Identyfikatory UUID LVM nie mają standardowego szesnastkowego formatu. Są to raczej losowo wygenerowane ciągi składające się ze znaków 0 – 9, a – z i A – Z.

Za pomocą narzędzia `lvdisplay` możesz uzyskać informacje o woluminach logicznych. Flaga `--maps` pozwala wyświetlić dodatkowe szczegóły dotyczące segmentów i ekstentów:

```
$ sudo lvdisplay --maps --foreign --readonly
...
  --- Logical volume ---
```

```
LV Path                /dev/mydisks/root
LV Name                root
VG Name                mydisks
LV UUID                uecfOf-3EOx-ohgP-IHyh-QPac-IaKl-HU1FMn
LV Write               Access read/write
LV Creation host, time pc1, 2020-12-02 20:45:45 +0100 ❶
LV Size                50.89 GiB
Current LE             13028
Segments               1
Allocation             inherit
Read ahead sectors     auto

--- Segments ---
Logical extents 0 to 13027:
  Type               linear ❷
  Physical volume    /dev/sdc2
  Physical extents   1024 to 14051
```

W miejscu ❷ znajduje się informacja (*Type linear*) o tym, że wolumin jest zapisany na dysku w postaci ciągłej sekwencji sektorów (tak jak w przypadku logicznego adresowania blokowego). W przypadku konfiguracji z pojedynczym dyskiem wystarczy jedynie znaleźć przesunięcie początku systemu plików. Po jego ustaleniu możesz analizować wolumin za pomocą narzędzi kryminalistycznych, które nie obsługują LVM. Z punktu widzenia śledztwa interesująca jest również nazwa hosta, na którym utworzono wolumin logiczny, oraz znacznik czasu jego utworzenia (❶).

Informacje o ekstentach pomagają znaleźć (obliczyć) pierwszy sektor systemu plików. W pokazanej wcześniej tablicy partycji (wyjście z `mmls`) znajduje się informacja o tym, że partycja LVM zaczyna się w sektorze `2099200`. Pierwszy ekstent fizyczny znajduje się w odległości 2048 sektorów od początku partycji LVM[6]. W danych wyjściowych z `pvdisplay` widzimy, że rozmiar ekstentu to 8192 sektory (*PE Size 4.00 MiB*), a wyjście z `lvdisplay` pokazuje, że główny wolumin zaczyna się od ekstentu 1024. Na podstawie tego wszystkiego możemy określić przesunięcie (w sektorach) systemu plików od początku dysku:

$$2099200 + 2048 + (8192 \cdot 1024) = 10489856$$

W przypadku liniowego jednodyskowego systemu LVM, w którym system plików jest przechowywany w postaci ciągłej sekwencji sektorów, możesz zastosować standardowe narzędzia kryminalistyczne, którym należy dodatkowo przekazać przesunięcie względem początku dysku fizycznego. Poniżej pokazano przykład użycia narzędzia z pakietu TSK:

[6] Zgodnie z kodem źródłowym (*https://github.com/lvmteam/lvm2/blob/master/lib/config/defaults.h*) jest to 1 MiB nagłówka LVM.

```
$ sudo fsstat -o 10489856 /dev/sdc
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: Ext4
Volume Name:
Volume ID: 6d0edeac50c97b979148918692af1e0b
...
```

Polecenie `fsstat` z TSK dostarcza informacji o systemach plików. W tym przykładzie na partycji LVM w obliczonej powyżej odległości od początku partycji został znaleziony system plików *ext4*. Alternatywą do obliczania przesunięcia jest dokładne przeszukiwanie dysku w celu znalezienia początku systemu plików przy użyciu narzędzi takich jak np. `gpart`. Aby uzyskać dodatkowe, szczegółowe informacje o grupach woluminów i woluminach fizycznych, możesz zastosować polecenia `vgdisplay` i `pvs` wraz z jedną lub kilkoma flagami `-v`.

LVM oferuje również możliwość wykonywania migawek *copy-on-write* (CoW; kopiowanie przy zapisie). Są one interesujące z przyczyn śledczych, ponieważ w systemie mogą istnieć migawki woluminów z wcześniejszych punktów w czasie. W działających systemach istnieje możliwość „zamrożenia" stanu woluminu w migawce w celu jego analizy, a nawet akwizycji.

Linux Software RAID

Na wczesnych etapach rozwoju zastosowań informatyki w biznesie odkryto, że grupy dysków twardych da się ze sobą połączyć, tak aby pracowały równolegle. Takie połączenie zwiększa niezawodność oraz wydajność systemu pamięci masowej. Koncepcja ta jest znana jako RAID (ang. *redundant array of independent disks*[7], pol. nadmiarowa macierz niezależnych zysków). Do opisu konfiguracji RAID używa się kilku terminów. **Tryb lustrzany** (ang. *mirror*) odnosi się do sytuacji, w której na dwóch dyskach przechowywane są te same dane (dyski są swoimi lustrzanymi odbiciami). **Tryb striped** to rozłożenie danych na wielu dyskach w celu zwiększenia wydajności (można równocześnie odczytywać i zapisywać dane na wielu dyskach). **Bit parzystości** (ang. *parity bit*) to dodatkowy bit używany do wykrywania i/lub korygowania błędów.

Istnieją różne odmiany RAID nazywane poziomami (ang. *levels*). Poziomy opisują, w jaki sposób współpracują ze sobą poszczególne dyski:

- *RAID0* — tryb *striped* zwiększający wydajność. Brak redundancji.
- *RAID1* — tryb lustrzany zapewniający redundancję (połowa dysków może się zepsuć, a dane nie zostaną utracone). Wynikowa pojemność systemu to połowa pojemności oryginalnych dysków.
- *RAID 2, 3, 4, 5* — warianty wykorzystujące bity parzystości pozwalające na awarię pojedynczego dysku.

[7] Znane również jako *reduntant array of inexpesnive disks* (pol. nadmiarowa macierz niedrogich dysków).

- *RAID6* — podwójna parzystość pozwalająca na awarię maksymalnie dwóch dysków.
- *RAID10* — połączenie trybu lustrzanego i *striped* („1 + 0") w celu uzyskania zarówno redundancji, jak i zwiększenia wydajności.
- *JBOD* (*Just a Bunch Of Disks*) — połączenie kilku dysków, brak nadmiarowości i wydajności, maksymalna pojemność.

Wybór odpowiedniego poziomu to kwestia znalezienia kompromisu pomiędzy kosztami, wydajnością i niezawodnością.

Niektóre komercyjne narzędzia kryminalistyczne mogą oferować możliwość odtworzenia i analizy macierzy RAID. W przypadku braku takiej możliwości wystarczy przenieść obrazy kryminalistyczne na komputer z systemem Linux, na którym zostanie przeprowadzona dalsza ich analiza. W mojej poprzedniej książce (*Practical Forensic Imaging*, No Starch Press, 2016) wyjaśniam, jak tworzy się obraz kryminalistyczny różnych macierzy RAID, w tym tych utworzonych w systemie Linux. W tym punkcie zakładam, że zawartość poszczególnych dysków została pozyskana w sposób zgodny z zasadami kryminalistyki cyfrowej oraz że zawartość ta jest dostępna w postaci plików obrazów tylko do odczytu lub podłączona do systemu komputerowego za pośrednictwem blokera zapisu. Ważne jest, abyś upewnił się, że dyski lub obrazy są podłączone w trybie tylko do odczytu. W przeciwnym razie system zainstalowany na komputerze analitycznym może automatycznie wykryć partycje RAID i spróbować ponownie podłączyć, zsynchronizować lub odbudować macierz RAID.

Funkcjonalność RAID w systemie Linux zapewnia `md` (sterownik wielu urządzeń znany też jako Linux Software RAID) lub LVM. RAID może też być elementem systemu plików (na przykład *btrfs* i *zfs* posiadają zintegrowany RAID).

Najczęściej stosowaną metodą tworzenia RAID (i głównym tematem tego punktu) jest użycie `md`. Ten moduł jądra tworzy metaurządzenie reprezentujące zbiór dysków. Do konfiguracji i zarządzania macierzą można wykorzystać narzędzie `mdadm`. W pozostałej części tego punktu przedstawię artefakty występujące w typowym systemie RAID utworzonym za pomocą `md`. Więcej informacji o urządzeniach `md` znajdziesz na stronie `md(4)` manuala.

Dysk wykorzystywany w macierzy RAID może zawierać tablicę partycji ze standardowymi typami partycji RAID systemu Linux. W przypadku tablicy GPT identyfikatorem GUID linuksowego systemu RAID jest `A19D880F-05FC-4D3B-A006-743F0F84911E` (w postaci bajtów zapisanych na dysku będzie to `0F889DA1-FC05-3B4D-A006-743F0F84911E`).

W przypadku tablic DOS/MBR typ partycji Linux Software RAID to `0xFD`. Partycję tego typu znajdziesz za pomocą dowolnego narzędzia kryminalistycznego na każdym dysku będącym częścią macierzy RAID.

Każde urządzenie linuksowej macierzy RAID zawiera **superblok** (nie mylić z superblokami z systemów plików, które są czymś innym), w którym znajdują się informacje o urządzeniu i systemie. W nowoczesnych urządzeniach Linux RAID superblok znajduje się standardowo w odległości ośmiu sektorów od początku

partycji. Można go zidentyfikować za pomocą specjalnego ciągu `0xA92B4EFC`. Zawartość superbloku możesz sprawdzić za pomocą edytora szesnastkowego lub, jak pokazano poniżej, za pomocą polecenia `mdadm`:

```
# mdadm --examine /dev/sda1
/dev/sda1:
          Magic : a92b4efc
        Version : 1.2
    Feature Map : 0x0
  Array UUID : 1412eafa:0d1524a6:dc378ce0:8361e245 ❶
           Name : My Big Storage ❷
  Creation Time : Sun Nov 22 13:48:35 2020 ❸
     Raid Level : raid5
   Raid Devices : 3

 Avail Dev Size : 30270751 (14.43 GiB 15.50 GB)
     Array Size : 30270464 (28.87 GiB 31.00 GB)
  Used Dev Size : 30270464 (14.43 GiB 15.50 GB)
    Data Offset : 18432 sectors
   Super Offset : 8 sectors
   Unused Space : before=18280 sectors, after=287 sectors
          State : clean
    Device UUID : 79fde003:dbf203d5:521a3be5:6072caa6 ❹

    Update Time : Sun Nov 22 14:02:44 2020 ❺
  Bad Block Log : 512 entries available at offset 136 sectors
       Checksum : 8f6317ee - correct
         Events : 4

         Layout : left-symmetric
     Chunk Size : 512K

    Device Role : Active device 0
    Array State : AAA ('A' == active, '.' == missing, 'R' == replacing)
```

Powyższy listing zawiera kilka artefaktów, które mogą być interesujące podczas badania kryminalistycznego. *Array UUID* (❶) to identyfikator całego systemu RAID. Każdy dysk będący elementem tej macierzy (w tym dyski, które zastąpiono nowymi) będzie zawierał ten sam UUID w swoim superbloku. Nazwa (w tym przypadku *My Big Storage*; ❷) może być ustalona przez administratora lub wygenerowana w sposób automatyczny. Identyfikator UUID urządzenia (*Device UUID*; ❹) jednoznacznie identyfikuje poszczególne dyski. Znacznik czasu ❸ to data utworzenia macierzy (świeżo wymieniony dysk odziedziczy oryginalną datę utworzenia macierzy). Znacznik czasu ❺ odnosi się do ostatniej aktualizacji superbloku spowodowanej zajściem jakiegoś zdarzenia w systemie plików.

Dyski w macierzy mogą nie mieć identycznych rozmiarów. Może to mieć znaczenie w trakcie badania kryminalistycznego. W tym przykładzie badam

macierz RAID5 o pojemności 31 GB stworzoną z trzech dysków udostępniających po 15,5 GB swojej pojemności. Jak pokazano poniżej, jedno z urządzeń (*sdc*) ma rozmiar aż 123,6 GB:

```
# mdadm --examine /dev/sdc1
/dev/sdc1:
...
 Avail Dev Size : 241434463 (115.12 GiB 123.61 GB)
     Array Size : 30270464 (28.87 GiB 31.00 GB)
  Used Dev Size : 30270464 (14.43 GiB 15.50 GB)
    Data Offset : 18432 sectors
...
```

Urządzenie to jest znacznie większe od pozostałych dysków macierzy. Znajduje się na nim ponad 100 GB nietkniętych danych. Obszar ten można przebadać pod kątem przechowywanych w nim wcześniej danych.

Nazwa urządzenia reprezentującego macierz ma zazwyczaj postać */dev/md#*, */dev/md/#* lub */dev/md/NAZWA* (administrator systemu może określić wartość występującą w miejscu *# /NAZWA*). Te urządzenia jądra będą istniały tylko w działającym systemie, ale wzmianki na ich temat można również znaleźć w logach w trakcie analizy kryminalistycznej *post mortem*; na przykład:

```
Nov 22 11:48:08 pc1 kernel: md/raid:md0: Disk failure on sdc1, disabling device.
                            md/raid:md0: Operation continuing on 2 devices.
...
Nov 22 12:00:54 pc1 kernel: md: recovery of RAID array md0
```

W powyższych logach jeden z dysków w systemie RAID5 uległ awarii, a jądro wygenerowało komunikat, który został następnie zapisany w dzienniku. Po wymianie uszkodzonego dysku został wygenerowany kolejny komunikat jądra o odtworzeniu w pełni działającego systemu RAID5.

Jądro powinno automatycznie poszukiwać i rozpoznawać urządzenia Linux RAID w trakcie rozruchu systemu. Urządzenia można również zdefiniować w osobnych plikach konfiguracyjnych. Podczas badania systemów RAID sprawdź, czy w pliku */etc/mdadm.conf* (lub w pliku */etc/mdadm.conf.d/*) nie występują odkomentowane wiersze zawierające słowa *DEVICE* lub *ARRAY*. Więcej informacji znajdziesz na stronie `mdadm.conf(5)` manuala.

Jeśli uda Ci się odnaleźć uszkodzone dyski, które stanowiły wcześniej element systemu RAID, to nadal możesz odczytać ich zawartość. Uszkodzone lub wymienione dyski będą zawierały migawkę danych z określonego momentu i mogą mieć znaczenie w trakcie dochodzenia.

Na przyszłe wykorzystanie tradycyjnych macierzy RAID w środowisku korporacyjnym będzie miało wpływ wiele czynników. Standardowe dyski o dużej pojemności (w chwili pisania tego tekstu na rynku dostępne są dyski o pojemności 18 TB) wymagają więcej czasu na ponowną synchronizację i odbudowę systemu

RAID. W niektórych przypadkach, w zależności od rozmiarów i szybkości dysków, operacje te mogą trwać kilka dni. Obecnie widoczny jest trend zastępowania systemów RAID klastrami niedrogich komputerów (coś à la RAID zbudowany z pecetów), które wykorzystują replikację danych w celu zwiększenia wydajności i zapewnienia redundancji. Wykorzystanie dysków SSD zamiast talerzowych dysków magnetycznych również zmniejsza ryzyko awarii (brak ruchomych części mechanicznych).

Analiza kryminalistyczna systemu plików

W tym podrozdziale znajdziesz omówienie elementów, które są wspólne dla wszystkich uniksopodobnych systemów plików. W przykładach wykorzystałem zestaw narzędzi TSK, ale wszystkie opisane czynności powinno się też dać przeprowadzić za pomocą popularnych komercyjnych narzędzi kryminalistycznych. Linux obsługuje dziesiątki systemów plików, a pokazane w tym podrozdziale podejście analityczne można zastosować do badania większości z nich.

Koncepcja systemu plików w Linuksie

System plików jest centralnym i fundamentalnym elementem systemów Unix i Linux. W trakcie prac nad pierwszą wersją Uniksa Ken Thompson opracował pierwszy system plików i rozwinął w Uniksie koncepcję, zgodnie z którą „wszystko jest plikiem". Pomysł ten umożliwia dostęp do wszystkich elementów systemu, w tym urządzeń sprzętowych, procesów, struktur danych jądra, sieci, komunikacji międzyprocesowej i oczywiście zwykłych plików i katalogów, za pomocą plików znajdujących się w drzewie katalogów.

Podstawowe typy plików wymienione w standardzie POSIX omówię w następnym rozdziale. Zaliczamy do nich zwykłe pliki, katalogi, dowiązania symboliczne, nazwane potoki, urządzenia i gniazda. W tym rozdziale, pisząc „plik", mam na myśli typy plików ze standardu POSIX znajdujące się w uniksowym systemie plików, a nie typy plików aplikacji, takie jak obrazy, filmy lub dokumenty.

Dyski twarde i dyski SSD zawierają zintegrowaną elektronikę, która tworzy abstrakcję ciągłej sekwencji sektorów (logiczne adresowanie bloków, ang. *Logical Block Addressing*, LBA). Partycje znajdujące się na dysku mogą zawierać systemy plików, które znajdują się w znanej odległości (przesunięcie/offset) od sektora zerowego. System plików wykorzystuje ciągłą grupę sektorów do utworzenia bloku (zwykle o rozmiarze 4 KB). Zbiór jednego lub więcej (niekoniecznie ciągłych) bloków tworzy zawartość pliku.

Każdy plik ma przypisany (unikalny w systemie plików) numer zwany *i*-węzłem (ang. *inode/index node*). Bloki zajmowane przez każdy plik oraz inne metadane (uprawnienia, znaczniki czasu itd.) są przechowywane w **tablicy i-węzłów**. Nazwy plików nie są zapisane w *i*-węzłach, a znajdują się we wpisach w pliku reprezentującym katalog (ang. *directory file*). Wpisy w pliku katalogu łączą nazwę pliku z *i*-węzłem i tworzą iluzję drzewiastej struktury systemu plików. Znana Ci pełna

„ścieżka" do pliku (*/jakaś/ścieżka/plik.txt*) nie jest nigdzie przechowywana. Można ją wyznaczyć poprzez podążanie za powiązanymi ze sobą nazwami katalogów, począwszy od pliku, aż do katalogu głównego (/).

Stan alokacji bloków i *i*-węzłów jest zapisany w mapach bitowych. Stan ulega zmianie podczas tworzenia lub usuwania plików. Na rysunku 3.2 pokazano warstwy tworzące tę abstrakcję.

Tradycyjne systemy plików zostały zaprojektowane w czasach dysków magnetycznych z wirującymi talerzami i głowicami odczytu/zapisu przymocowanymi do mechanicznych ramion. W tamtych czasach niezbędna była optymalizacja wydajności i odporność na błędy, co udało się osiągnąć dzięki grupowaniu bloków i *i*-węzłów na dysku.

Niektóre z oryginalnych decyzji projektowych podjętych w trakcie opracowywania systemu plików (na przykład optymalizacja wydajności związana z obecnością obracających się talerzy i poszukiwaniem odpowiednich obszarów przez głowice dysków) są niepotrzebne w przypadku dysków SSD, ale nadal istnieją w systemach plików. Nowoczesne systemy plików oferują dodatkowe funkcjonalności, takie jak kronikowanie, które zapewnia spójność danych w przypadku awarii, lub wykorzystują *ekstenty* (ciągłe zakresy bloków) zamiast list bloków przydzielonych do poszczególnych plików. Ponadto każdy system plików może mieć własne unikalne cechy i atrybuty, które mogą być interesujące z kryminalistycznego punktu widzenia (na przykład system *ext4* zawiera znacznik czasu i ścieżkę do ostatniego punktu montowania).

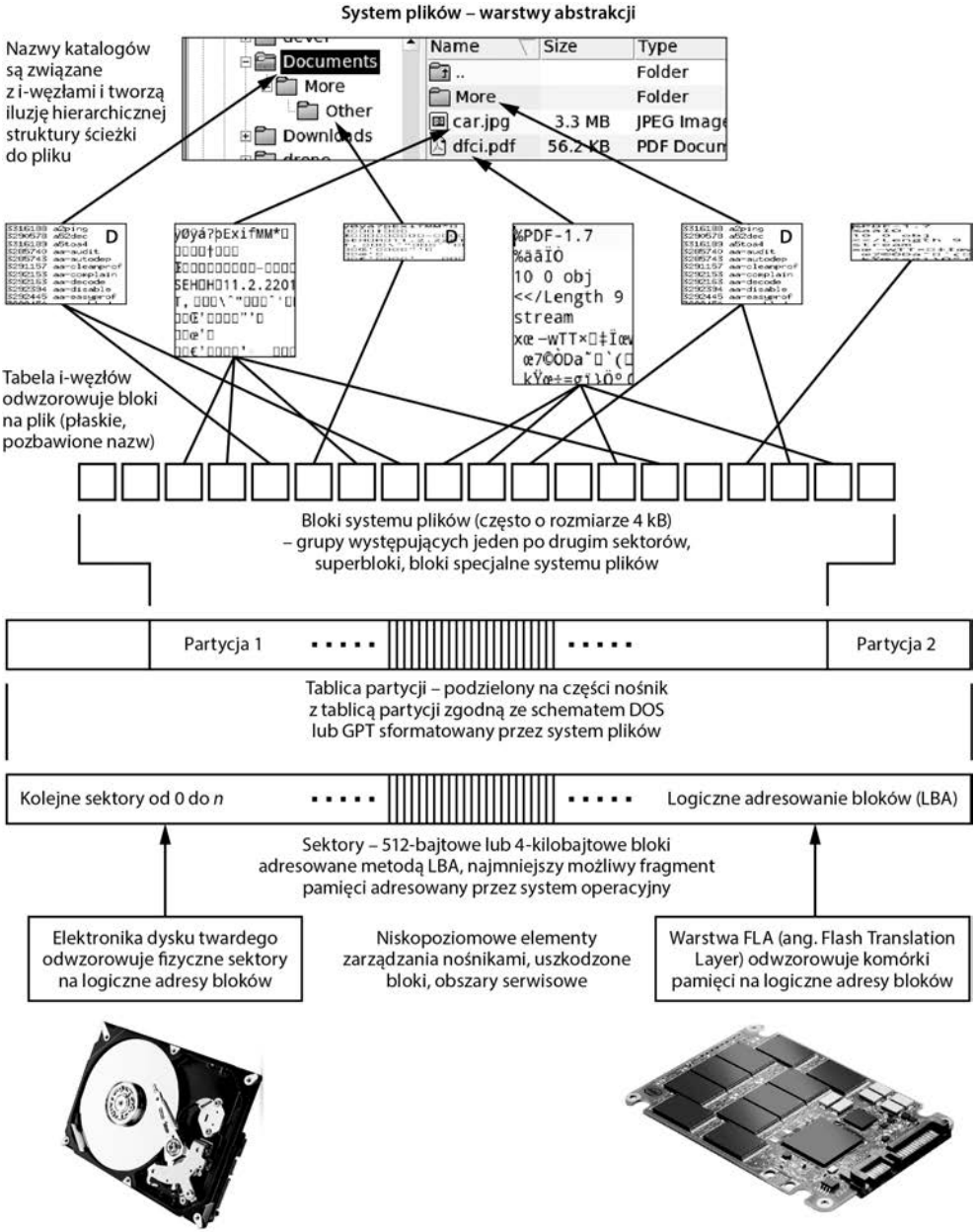
Sieciowe systemy plików (NFS, CIFS/Samba itd.), FUSE i pseudosystemy plików (*/proc/*, */sys/* itd.) wykorzystują podobną do innych systemów reprezentację drzewa katalogów oraz plików. Ich omówienie wykracza jednak poza zakres tej książki, ponieważ nie da się ich analizować *post mortem*, tak jak urządzeń fizycznych.

Większość systemów plików w świecie Uniksa i Linuksa opiera się na tych samych koncepcjach projektowych, co ułatwia zastosowanie tych samych metod cyfrowej analizy kryminalistycznej do wielu systemów plików.

Artefakty w systemach plików Linuksa

Pierwszym krokiem w analizie systemu plików jest ustalenie, jakie system plików badamy. Jak wspomniałem wcześniej, w ustaleniu rodzaju systemu mogą pomóc tablice partycje. Może się jednak okazać, że typ partycji zapisany w tablicy nie pokrywa się z rzeczywistością. Dlatego potrzebna nam będzie bardziej niezawodna metoda.

Większość systemów plików można zidentyfikować na podstawie kilku pierwszych bajtów rozpoczynających system plików na nośniku. Bajty te są nazywane **sygnaturą** (ang. *signature*) lub **magicznym stringiem** (ang. *magic string*). Jeśli wykorzystywane przez Ciebie narzędzia kryminalistyczne nie potrafią automatycznie ustalić nazwy systemu plików, to możesz spróbować ręcznie ustalić jego sygnaturę (na przykład za pomocą narzędzia `sigfind` z TSK). Specyfikacja systemu plików określa wartość jego sygnatury. Do identyfikacji systemu plików możesz



Rysunek 3.2. Abstrakcja systemu plików. (Jest to uproszczony schemat, który nie obejmuje grup bloków, redundancji, skalowalności i innych specjalnych funkcjonalności)

wykorzystać również inne narzędzia, takie jak `disktype` lub `fsstat` z TSK. Dobrym wskaźnikiem istnienia systemu plików jest obecność jego sygnatury w przewidywanej odległości od początku partycji.

Superblok to zbiór metadanych opisujących cały system plików. W zależności od systemu może on zawierać interesujące z punktu widzenia śledztwa informacje, w tym:

- nazwę lub etykietę woluminu ustaloną przez właściciela systemu,
- unikalny identyfikator (UUID/GUID),
- znaczniki czasu (utworzenie systemu plików, ostatnie montowanie, ostatni zapis i ostatnie sprawdzenie),
- rozmiar i liczbę bloków (przydatne podczas identyfikowania przestrzeni luzu),
- liczbę montowań i ostatni punkt montowania,
- informacje związane z innymi funkcjonalnościami systemu plików i jego konfiguracją.

Informacje te można wyświetlić za pomocą większości narzędzi kryminalistycznych, w tym za pomocą `fsstat`. Zazwyczaj systemy plików oferują również narzędzia do debugowania i rozwiązywania problemów, które mogą wyświetlać jeszcze więcej informacji.

System plików określa również strukturę *i*-węzła oraz to, jakie metadane będą związane z każdym plikiem. Wśród nich mogą się znajdować interesujące informatyków śledczych informacje, w tym:

- POSIX-owy typ pliku,
- uprawnienia i informacje o właścicielu pliku,
- wiele znaczników czasu (w tym powszechnie znane znaczniki MACB oraz być może inne znaczniki),
- rozmiar i liczba bloków zajmowanych przez plik (można je wykorzystać do sprawdzenia, czy istnieją niewykorzystane przestrzenie na końcu pliku),
- inne flagi i atrybuty.

Najbardziej wiarygodnym źródłem informacji o strukturze i-węzłów w konkretnym systemie jest dokumentacja lub kod źródłowy systemu plików.

Inne artefakty kryminalistyczne mają związek z zawartością pamięci. Dysk dzieli się na różne obszary, w których przechowywane są dane. Ich znajomość będzie pomocna podczas wyodrębniania i odzyskiwania danych. Do najbardziej interesujących z punktu widzenia analizy kryminalistycznej obszarów zaliczamy:

- **Sektor** — najmniejsza dostępna jednostka pamięci na dysku.
- **Blok** — grupa występujących jeden za drugim sektorów i najmniejsza jednostka pamięci dostępna w systemie plików.
- **Ekstent** — grupa kolejnych bloków w systemie plików (zmienny rozmiar).

- **Zaalokowane bloki** — bloki, w których przechowywana jest zawartość plików.
- **Bloki niezaalokowane** — bloki, które nie są powiązane z plikami (prawdopodobnie bloki te zawierają dane z usuniętych plików).

W systemie plików usunięcie pliku sprowadza się do usunięcia związanego z nim dowiązania oraz oznaczenia związanego z nim *i*-węzła i powiązanych z nim bloków danych jako nieprzydzielonych i możliwych do ponownego użycia. Na dyskach magnetycznych dane znajdujące się w usuniętym pliku pozostają na talerzach, dopóki zawartość bloków nie zostanie nadpisana. Dzięki temu możemy odzyskać usunięte dane za pomocą narzędzi kryminalistycznych. W przypadku dysków SSD w ramach przygotowań do następnego zapisu system operacyjny może wysłać do oprogramowania układowego dysku polecenie (`TRIM` lub `DISCARD`) nakazujące fizyczne usunięcie usuniętych danych[8]. Funkcjonalność ta zmniejsza prawdopodobieństwo odzyskania danych z niezaalokowanych obszarów dysków SSD.

W informatyce śledczej **luzem** (ang. *slack*) lub **przestrzeniami luzu** (ang. *slackspace*) nazywamy obszary dysków, w których teoretycznie mogą się znajdować jakieś dane:

- **Luz woluminu** (ang. *volume slack*) — obszar między końcem systemu plików a końcem partycji.
- **Luz pliku** (ang. *file slack*) — obszar między końcem pliku a końcem bloku.
- **Luz pamięci RAM** (ang. *RAM/memory slack*) — obszar między końcem pliku a końcem sektora.
- **Przestrzeń pomiędzy partycjami** — fragment dysku nienależący do żadnej partycji (prawdopodobnie usunięta partycja).

Obecnie systemy operacyjne zachowują większą ostrożność podczas obsługi usuniętych danych. Polecenia `TRIM` i `DISCARD` służą do wymazywania komórek pamięci na nośnikach SSD, a sektory 4 KB w systemie operacyjnym (najmniejsza adresowalna w systemie operacyjnym jednostka pamięci) mają taki sam rozmiar jak bloki w systemach plików. Czynniki te powodują, że znaczenie dowodowe przestrzeni luzu zmniejsza się.

Wyświetlanie i wyodrębnianie danych

Częścią analizy kryminalistycznej systemu plików jest odzyskiwanie plików (w tym tych usuniętych) oraz odzyskiwanie fragmentów plików (przestrzenie luzu lub niezaalokowane obszary). Obie te operacje można przeprowadzić za pomocą większości istniejących narzędzi kryminalistycznych. Spójrzmy na kilka gotowych instrukcji pozwalających wykonać te czynności za pomocą narzędzi z TSK.

[8] W przypadku dysków SSD komórki pamięci muszą być wymazane, zanim będzie w nich można coś ponownie zapisać.

Najpierw przeanalizujmy związki między sektorami, blokami, *i*-węzłami i nazwami plików. Poniżej wykorzystałem elementarną matematykę oraz narzędzia z TSK, aby odpowiedzieć na następujące pytania:

- Znam sektor na dysku. Do którego bloku w systemie plików należy ten sektor?

  ```
  (sektor – offset_partycji) * rozmiar_sektora / rozmiar_bloku
  ```

- Znam blok w systemie plików. W jakim sektorze się on znajduje?

  ```
  (blok * rozmiar_bloku/rozmiar_sektora) + offset_partycji
  ```

- Czy dany blok (123) jest zaalokowany?

  ```
  blkstat partimage.raw 123
  ```

- Z jakim *i*-węzłem jest związany konkretny (456) zaalokowany blok?

  ```
  ifind -d 456 partimage.raw
  ```

- Znam *i*-węzeł pliku. Jak wyświetlić metadane pliku (i użyte bloki)?

  ```
  istat partimage.raw 789
  ```

- Znam *i*-węzeł pliku. Jaka jest nazwa pliku?

  ```
  ffind partimage.raw 789
  ```

- Z jakim *i*-węzłem jest związany plik o znanej mi nazwie?

  ```
  ifind -n "hello.txt" partimage.raw
  ```

WSKAZÓWKA *Upewnij się, że używasz właściwych jednostek! W zależności od narzędzia jednostkami mogą być bajty, sektory lub bloki.*

Pakiet TSK zawiera narzędzia do analizy obrazów dysków i systemów plików. Podczas korzystania z narzędzia do analizy systemu plików potrzebna jest znajomość jego lokalizacji. Narzędzia śledcze przeznaczone do badania systemów plików potrafią odczytywać dane z plików urządzeń partycji (np. */dev/sda1*), z wyodrębnionych obrazów partycji (*partimage.raw*) oraz podłączonych dysków i ich obrazów. W przypadku dysków i obrazów konieczne jest wskazanie przesunięcia/offsetu dla wybranego systemu plików (zazwyczaj za pomocą opcji `-o`).

Listę wszystkich (w tym usuniętych) rozpoznanych plików znajdujących się w systemie plików można wyświetlić za pomocą narzędzia `fls` z pakietu TLS. W poniższym przykładzie zastosowałem opcję `-r`, która powoduje rekurencyjne wyświetlenie plików znajdujących się we wszystkich katalogach, oraz opcję `-p`, która skutkuje pokazaniem pełnych ścieżek do plików (użycie `-l` spowodowałoby wyświetlenie sygnatur czasowych, rozmiarów i informacji o właścicielach plików).

```
$ fls -r -p partimage.raw
...
r/r 262172:     etc/hosts
d/d 131074:     var/cache
...
r/r 1050321:    usr/share/zoneinfo/Europe/Vaduz
r/r 1050321:    usr/share/zoneinfo/Europe/Zurich
```

```
...
r/r * 136931(realloc):  var/cache/ldconfig/aux-cache~
r/r 136931:     var/cache/ldconfig/aux-cache
...
V/V 1179649:    $OrphanFiles
-/r*655694:     $OrphanFiles/OrphanFile-655694
...
```

Powyższe wywołanie znalazło w moim testowym systemie ponad 45 000 plików. Wybrałem kilka z nich, aby wyjaśnić prezentowane przez narzędzie wyniki. Więcej informacji na temat polecenia `fls` znajdziesz w wiki TSK (*https://github.com/sleuthkit/sleuthkit/wiki/fls/*). Pierwsza kolumna wyjścia (`r/r`, `d/d` itd.) reprezentuje typ pliku ustalony na podstawie wpisu w pliku katalogu oraz zawartości *i*-węzła. Na przykład plik */etc/hosts* jest zwykłym plikiem (`r`), co na powyższym wyjściu jest reprezentowanie przez `r/r`. Pierwsza wartość `r` została ustalona na podstawie wpisu reprezentującego katalog */etc/*. Druga na podstawie metadanych pliku */etc/hosts* (zawartości *i*-węzła). Typy plików występujące w Linuksie[9] opisano w wiki TSK. Są to:

- *r/r* — zwykły plik,
- *d/d* — katalog,
- *c/c* — urządzenie znakowe,
- *b/b* — urządzenie blokowe,
- *l/l* — dowiązanie symboliczne,
- *p/p* — nazwana kolejka FIFO,
- *h/h* — gniazdo.

Myślnik po obu stronach ukośnika (-/-) reprezentuje nieznany typ (brak informacji o typie we wpisie reprezentującym katalog oraz w *i*-węźle). Liczba występująca po typie pliku to numer *i*-węzła. Zauważ, że dwa pliki mogą współdzielić ten sam *i*-węzeł (*Vaduz* i *Zurich*). Jest tak w przypadku dowiązania twardego/sztywnego. Symbol * reprezentuje usunięty plik. Jeśli po usunięciu pliku system ponownie wykorzystał związany z nim *i*-węzłem, to fakt ten jest reprezentowany przez słowo *realloc* (może się ono również pojawić, gdy nazwa pliku została zmieniona). Jeśli plik został usunięty i poza danymi w *i*-węźle nie istnieje już żadna inna informacja na jego temat, to plik taki jest wyświetlany jako plik znajdujący się w wirtualnym katalogu TSK o nazwie *$OrphanFiles*. TSK może wyświetlać dodatkowe informacje o usuniętym pliku lub katalogu (typ *v/v* lub *V/V*), ale są to nazwy wirtualne, które nie istnieją w analizowanym systemie plików. Numer *i*-węzła związanego z wirtualnym katalogiem *$OrphanFiles* to maksymalna liczba *i*-węzłów w systemie plus jeden.

[9] Obsługiwane są też formaty Solaris Shadow (`s/s`) oraz OpenOBSD Whiteout (`w/w`).

Polecenia z pakietu TSK pozwalają również wyodrębnić zawartość z systemu plików. Oto kilka przykładów:

- Wyodrębnianie zawartości pliku na podstawie numeru *i*-węzła (użyj `-s`, aby uwzględnić luz):
  ```
  icat partimage.raw 1234
  ```
- Wyodrębnianie zawartości pliku na podstawie jego nazwy (użyj `-s`, aby uwzględnić luz):
  ```
  fcat hello.txt /dev/sda1
  ```
- Wyodrębnianie bloków z systemu plików (argumenty to przesunięcie oraz liczba bloków):
  ```
  blkcat partimage.raw 56789 1
  ```
- Wyodrębnianie wszystkich niezaalokowanych bloków w systemie plików:
  ```
  blkls partimage.raw
  ```
- Wyodrębnianie wszystkich luzów plikowych (z niezaalokowanych bloków):
  ```
  blkls -s partimage.raw
  ```
- Wyodrębnianie jednego sektor za pomocą `dd` (liczbę sektorów można ustalić za pomocą parametru `count`):
  ```
  dd if=image.raw skip=12345 count=1
  ```

Zawsze przesyłaj lub przekierowuj wyodrębnione dane do następnego programu lub pliku (za pomocą | lub >). W przeciwnym razie ryzykujesz wykonanie niechcianych poleceń lub powstanie bałaganu w terminalu/powłoce.

Dla ułatwienia poniżej pogrupowałem wszystkie polecenia z TSK według ich funkcji/zastosowania:

- Obrazy kryminalistyczne: `img_cat`, `img_stat`.
- Partycje: `mmcat`, `mmls`, `mmstat`.
- Informacje o systemie plików: `fsstat`, `pstat`.
- Bloki systemu plików: `blkcalc`, `blkcat`, `blkls`, `blkstat`.
- Nazwy plików: `fcat`, `ffind`, `fls`, `fiwalk`.
- *i*-węzły: `icat`, `ifind`, `ils`, `istat`.
- Osie czasu: `mactime`, `tsk_gettimes`.
- Wyszukiwanie i sortowanie: `sigfind`, `sorter`, `srch_strings`, `tsk_comparedir`, `tsk_loaddb`, `tsk_recover`, `hfind`.
- Dziennik systemu plików: `jcat`, `jls`, `usnjls`.

Więcej informacji na ich temat znajdziesz w manualu. (Manual Debiana zawiera kilka dodatkowych stron, które nie występują w dokumentacji TSK).

Większość komercyjnych narzędzi kryminalistycznych również potrafi wykonać te same czynności. Jak wspomniałem wcześniej, alternatywą w przypadku systemów plików, które nie są przez nie obsługiwane, jest debugowanie i narzędzia

do rozwiązywania problemów, które są zwykle dostarczane przez twórców systemu plików. W kolejnych podrozdziałach wykorzystam tego typu narzędzia dołączone do systemów *ext4*, *btrfs* i *xfs*.

Analiza ext4

Jednym z najstarszych i najpopularniejszych systemów plików Linuksa jest system *ext* (ang. *extended filesystem*, dosł. rozszerzony system plików). Każda nowoczesna dystrybucja Linuksa obsługuje *ext4*, a w wielu z nich jest to domyślny system plików. Ze względu na popularność systemu *ext* (wersje 2, 3 i 4) wiele komercyjnych narzędzi kryminalistycznych wspiera analizę tego formatu. Obsługują go TSK i Autopsy, a także wiele innych narzędzi do rozwiązywania problemów, debugowania i odzyskiwania danych.

Ext4 to oparty na ekstentach skalowalny system plików obsługujący kronikowanie oraz szyfrowanie na poziomie katalogów. Zajrzy na stronę `ext4(5)` manuala, aby uzyskać więcej informacji.

W porównaniu z innymi popularnymi w Linuksie systemami plików superblok *ext4* zawiera więcej interesujących z punktu widzenia śledztwa artefaktów. Z drugiej strony podczas usuwania plików system *ext4* kasuje więcej informacji o plikach, co utrudnia odzyskiwanie usuniętych danych.

Metadane systemu plików — superblok

Superblok rozpoczyna się w odległości 1024 bajtów (`0x400`) od początku systemu plików. Sygnaturą *ext2*, *ext3* i *ext4* jest `0xEF53` (identyczna sygnatura we wszystkich trzech wersjach systemu). Sygnatura znajduje się w odległości 56 bajtów (`0x38`) od początku superbloku, czyli 1080 bajtów (`0x438`) od początku systemu plików. Sygnatura jest zapisana w kodowaniu *little endian*:

```
00000438:  53ef  S.
```

Superblok *ext4* zawiera znaczniki czasu, unikalne identyfikatory, cechy i informacje opisowe, które mogą się przydać w trakcie badania kryminalistycznego, w tym:

- Znacznik czasu utworzenia systemu plików.
- Znacznik czasu ostatniego zamontowania systemu plików.
- Znacznik czasu ostatniego sprawdzenia systemu plików (`fsck`).
- Znacznik czasu ostatniej modyfikacji superbloku.
- Nazwę lub etykietę woluminu określoną przez użytkownika (maksymalnie 16 znaków).
- Unikalny UUID woluminu.

- Informację o system operacyjnym osoby, która utworzyła system. Jeśli nie jest to Linux, może to oznaczać, że w sprawę był zaangażowany inny system operacyjny (*0* = Linux, *3* = FreeBSD).
- Katalog, w którym ostatnio zamontowano system plików. Jeśli nie jest to standardowa lokalizacja, użytkownik mógł ręcznie utworzyć punkt montowania w systemie.
- Liczbę montowań od ostatniego sprawdzenia systemu przez `fsck`. W przypadku dysków zewnętrznych może ona wskazywać, jak często używany był ten system plików.
- Liczbę KiB zapisanych w trakcie istnienia systemu plików. Daje ona wyobrażenie o „aktywności" w tym systemie plików.

W niektórych przypadkach interesująca może być liczba KiB zapisanych w czasie życia systemu plików (na przykład w przypadku kradzieży danych na nośniki zewnętrzne kopiowane są duże ilości plików). Jeśli całkowita liczba zapisanych bajtów jest taka sama jak całkowity rozmiar wszystkich plików, to system plików nie był używany do innych celów. Jeśli dysk obsługuje funkcje SMART, to do porównania ilości danych na dysku z ilością danych zapisanych w okresie eksploatacji urządzenia można wykorzystać atrybut *Total LBAs Written* (podobną analizę można przeprowadzić z użyciem atrybutu *Total LBAs Read*).

Większość komercyjnych narzędzi kryminalistycznych powinna wspierać analizę superbloków z systemów *ext4*. W przeciwnym razie możesz skorzystać z `fsstat`. Narzędzie `dumpe2fs` (część pakietu `e2fsprogs`) również pozwala wyświetlić szczegółowe informacje o superbloku. W poniższym przykładzie korzystam z obrazu kryminalistycznego partycji (*partimage.raw*). Opcja `-h` pozwala wyświetlić informacje o nagłówku superbloku:

```
$ dumpe2fs -h partimage.raw
dumpe2fs 1.46.2 (28-Feb-2021)
Filesystem volume name:  TooManySecrets
Last mounted on:         /run/media/sam/TooManySecrets
Filesystem UUID:         7de10bcf-a377-4800-b6ad-2938bf0c08a7
Filesystem magic number: 0xEF53
...
Filesystem OS type:      Linux
Inode count:             483328
Block count:             1933312
...                      Sat Mar 13 07:42:13 2021
Filesystem created:      Sat Mar 13 08:33:42 2021
Last mount time:         Sat Mar 13 08:33:42 2021
Last write time:         16
Mount count:             -1
Maximum mount count:     Sat Mar 13 07:42:13 2021
Last checked:            1933312
...
```

Aby podkreślić istotne z punktu widzenia śledztwa informacje, z powyższego listingu usunąłem niektóre wartości. Jeśli nazwa woluminu (*TooManySecrets*) została ustalona przez użytkownika, to może ona zawierać opis jego zawartości (z perspektywy użytkownika). Rekord *Last mounted on:* to katalog, w którym system plików był ostatnio zamontowany. Ma on szczególne znaczenie w sprawach, w których występują nośniki zewnętrzne, ponieważ dzięki niemu możemy powiązać dysk z punktem montowania lub użytkownikiem w określonym systemie Linux. Punkt montowania może być utworzony przez użytkownika lub może to być tymczasowy punkt montowania stworzony przez menedżera dysków. W poprzednim przykładzie system plików był ostatnio zamontowany w */run/media/sam/TooManySecrets*, co wskazuje, że użytkownik *Sam* mógł zamontować go na swoim komputerze stacjonarnym za pomocą menedżera dysków[10]. Wiarygodnym źródłem informacji o strukturze superbloków jest strona *https://www.kernel.org/doc/html/latest/filesystems/ext4/globals.html*.

Informacje o superblokach możesz wyświetlić również za pomocą polecenia `fsstat` z pakietu TSK. Otrzymane wyniki będą mniej szczegółowe niż wyjście z `dumpe2fs`:

```
$ fsstat partimage.raw
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: Ext4
Volume Name: TooManySecrets
Volume ID: a7080cbf3829adb64877a3cf0be17d

Last Written at: 2021-03-13 08:33:42 (CET)
Last Checked at: 2021-03-13 07:42:13 (CET)

Last Mounted at: 2021-03-13 08:33:42 (CET)
Unmounted properly
Last mounted on: /run/media/sam/TooManySecrets

Source OS: Linux
...
```

Pełne dane wyjściowe zawierają informacje o grupach bloków i ich alokacji. W wielu badaniach kryminalistycznych informacje o alokacji bloków nie są potrzebne (ale nadal można je zamieścić w załączniku do raportu).

Zwróć uwagę, że identyfikatory UUID zwrócone przez `dumpe2fs` (`Filesystem UUID`) oraz `fsstat` (`Volume ID`) są różnymi reprezentacjami tego samego szesnastkowego łańcucha znaków.

[10] Jest to standardowy punkt montowania menedżera dysków `udisks`. Po więcej szczegółów zajrzyj do sekcji `udisks(8)` manuala.

Metadane pliku — *i*-węzły

Struktura *i*-węzłów w *ext4* jest dobrze udokumentowana i zawiera wiele interesujących z punktu widzenia informatyki śledczej informacji.

W *i*-węźle znajduje się rozmiar pliku i liczba zajmowanych przez niego bloków. Zwykle wartości te nie pokrywają się, chyba że rozmiar pliku jest wielokrotnością rozmiaru bloku. Wszelkie dane znajdujące się w ostatnim bloku po symbolu końca pliku nazywamy luzem plikowy.

W *i*-węźle znajdują się też dodatkowe flagi. Na przykład flaga `0x80` oznacza, że czas dostępu do pliku nie powinien być aktualizowany. Flaga `0x800` oznacza, że bloki *i*-węzłów są zaszyfrowane[11].

W rekordzie *file mode* zapisane są prawa dostępu (odczyt, zapis, wykonywanie przez właściciela, grupę i innych), bity specjalne (SetUID, SetGID i *sticky* bit), a także typ pliku (zwykły, katalog, dowiązanie symboliczne, FIFO, gniazdo oraz urządzenia znakowe i blokowe).

Rozszerzone atrybuty (na przykład listy ACL) nie są przechowywane w *i*-węźle, a w oddzielnym bloku danych, *i*-węzeł zawiera jedynie wskaźnik do takiego bloku.

Właściciel pliku jest określony przez UID użytkownika i jego grupę (GID). Pierwotnie informacje te zapisywano na 16 bitach, co pozwalało na reprezentowanie maksymalnie 65 535 użytkowników i grup. W późniejszym czasie na UID i GID przeznaczono dwa dodatkowe bajty (przechowywane w innych miejscach *i*-węzła), dzięki czemu UID i GID mają obecnie długość 32 bitów.

W *i*-węźle w systemie *ext4* przechowywanych jest pięć znaczników czasu (nazywanych znacznikami M, A, C, B i D):

- czas ostatniej modyfikacji danych (`mtime`),
- czas ostatniego dostępu (`atime`),
- czas ostatniej modyfikacji *i*-węzła (`ctime`),
- czas utworzenia (`crtime`),
- czas usunięcia.

Znacznik czasu usunięcia jest ustawiany tylko wtedy, gdy status *i*-węzła zmienia się z zaalokowanego na niezaalokowany.

W przeszłości znaczniki czasu miały długość 32 bitów i zawierały liczbę sekund, jakie upłynęły od 1 stycznia 1970. W takiej reprezentacji ostatnim możliwym do zapisania dniem jest 19 stycznia 2038. Nowoczesne systemy wymagają większej rozdzielczości czasowej (nanosekund) i muszą obsługiwać daty późniejsze niż rok 2038. Aby rozwiązać ten problem, w systemie *ext4* dodano do wszystkich znaczników czasu dodatkowe cztery bajty. Te dodatkowe 32 bity są rozdzielone w następujący sposób: 2 bity są przeznaczone na zapis czasu po 2038 roku, a 30 bitów zapewnia wyższą rozdzielczość (większą dokładność czasu).

[11] *https://www.kernel.org/doc/html/latest/filesystems/ext4/dynamic.html*.

Informacje o *i*-węźle z systemu *ext4* możesz wyświetlić za pomocą narzędzia `istat` z TSK:

```
$ istat partimage.raw 262172
inode: 262172
Allocated
Group: 32
Generation Id: 3186738182
uid/gid:0/0
mode: rrw-r--r--
Flags: Extents,
size: 139
num of links: 1

Inode Times:
Accessed:       2020-03-11 11:12:37.626666598 (CET)
File Modified:  2020-03-11 11:12:34.483333261 (CET)
Inode Modified: 2020-03-11 11:12:34.483333261 (CET)
FileCreated:    2020-03-1111:03:19.903333268(CET)

Direct Blocks:
1081899
```

Powyższe dane wyjściowe zawierają informacje o stanie *i*-węzła (*Allocated*, zaalokowany), prawa własności, uprawnienia, cztery znaczniki czasu i używane bloki.

Aby uzyskać więcej informacji, możesz skorzystać z narzędzia `debugfs` (z pakietu `e2fsprogs`). Poniżej pokazano przykład jego użycia dla usuniętego pliku. Flaga `-R` odnosi się do żądania (ang. *read only*), a nie użycia w trybie tylko do odczytu (polecenie standardowo działa w tym trybie). W parametrze `stat <136939>` określam, że żądam informacji statystycznych dla *i*-węzła *136939*. Polecenie wywołuję na obrazie kryminalistycznym *partimage.raw*:

```
$ debugfs -R "stat <136939>" partimage.raw
debugfs 1.45.6 (20-Mar-2020)
Inode: 136939 Type: regular Mode: 0000 Flags: 0x80000
Generation: 166965863 Version: 0x00000000:00000001
User:     0    Group: 0   Project: 0       Size: 0
File ACL: 0
Links:0 Blockcount:0
Fragment: Address: 0 Number: 0 Size: 0
 ctime: 0x5e68c4bb:04c4b400 -- Wed Mar 11 12:00:11 2020
 atime: 0x5e68c4ba:9a2d66ac -- Wed Mar 11 12:00:10 2020
 mtime: 0x5e68c4ba:9a2d66ac -- Wed Mar 11 12:00:10 2020
crtime: 0x5e68c4ba:9a2d66ac -- Wed Mar 11 12:00:10 2020
 dtime: 0x5e68c4bb:(04c4b400) -- Wed Mar 11 12:00:11 2020
Size of extra inode fields: 32
Inode checksum: 0x95521a7d
EXTENTS:
```

Na powyższym listingu widoczny jest *i*-węzeł usuniętego pliku, który zawiera pięć znaczników czasu, w tym znacznik czasu usunięcia. Zwróć uwagę na brak informacji o bloku po linii *EXTENTS:*. W systemie *ext4* po usunięciu pliku z *i*-węzła usuwane są informacje o poprzednio używanych blokach. W efekcie w tym systemie plików odzyskiwanie danych przy użyciu niektórych tradycyjnych technik kryminalistycznych może nie być możliwe.

Wyświetlanie listy plików i wyodrębnianie

Przykłady użycia narzędzi TSK do wyświetlenia listy plików i ekstrakcji z systemu *ext4* pokazałem w poprzednim punkcie. W tym zaprezentuję alternatywne podejście. Wiele czynności, które można wykonać za pomocą TSK, da się też zrobić za pomocą `debugfs`. Poniżej pokazałem, jak wykonać przykładowe czynności.

- Wyświetlanie zawartości katalogu, w tym usuniętych plików (bez wyświetlania zawartości podkatalogów):

```
debugfs -R "ls -drl" partimage.raw
```

- Wyodrębnianie zawartość pliku na podstawie *i*-węzła (działa podobnie do `icat`):

```
debugfs -R "cat <14>" partimage.raw
```

- Wyodrębnianie metadanych *i*-węzła (działa podobnie do `istat`):

```
debugfs -R "stat <14>" partimage.raw
```

- Wyodrębnianie metadanych *i*-węzła w postaci zrzutu szesnastkowego (działa podobnie do `istat`, ale w wyniku otrzymujemy surowe dane):

```
debugfs -R "inode_dump <14>" partimage.raw
```

<14> reprezentuje *i*-węzeł (w tym przykładzie *i*-węzeł o numerze 14). W poleceniu można również określić ścieżkę do pliku:

```
$ debugfs -R "ls -drl /Documents" partimage.raw
debugfs 1.45.6 (20-Mar-2020)
     12   40750 (2)      0     0        4096 30-Nov-2020 22:35 .
      2   40755 (2)      0     0        4096 30-Nov-2020 22:39 ..
     13  100640 (1)      0     0          91 30-Nov-2020 22:35 evilplan.txt
```

W danych wyjściowych widoczna jest lista plików z *i*-węzłami, rozmiarami, znacznikami czasu i nazwami plików.

Dane wyjściowe z `debugfs` mogą być wyświetlone w terminalu lub przekierowane do pliku, który zostanie zapisany na maszynie do analizy kryminalistycznej. Poniżej za pomocą `debugfs` wyświetlam zawartość pliku z poprzedniego przykładu (*evilplan.txt*):

```
$ debugfs -R "cat <13>" partimage.raw
debugfs 1.45.6 (20-Mar-2020)
this is the master plan to destroy all copies of powerpoint.exe across the entire
company.
```

Zawartość pliku jest wysyłana do terminala (`stdout`) i może być przekierowana do pliku lub do programu. Łańcuch znaków zawierający informacje o wersji `debugfs` jest widoczny w terminalu, ale nie jest dodawany do pliku lub przesyłany na wejście innego programu (informacja ta jest częścią `stderr`).

Innym interesującym śledczych elementem *ext4* są zaszyfrowane podkatalogi. Identyfikacji i odszyfrowaniu podkatalogów w *ext4* przyjrzymy się pod koniec tego rozdziału.

Specyfikacja *ext4* jest dostępna w dokumentacji jądra Linuksa pod adresem *https://www.kernel.org/doc/html/latest/filesystems/ext4/index.html*.

Więcej informacji o analizie kryminalistycznej systemu *ext4* znajdziesz w następujących artykułach naukowych (materiały w j. angielskim):

- Kevin D. Fairbanks, *An Analysis of Ext4 for Digital Forensics*, *https://www.sciencedirect.com/science/article/pii/S1742287612000357/*.
- Thomas Göbel i Harald Baier, *Anti-Forensics in Ext4: On Secrecy and Usability of Timestamp-Based Data Hiding*, *https://www.sciencedirect.com/science/article/pii/S174228761830046X/*.
- Andreas Dewald i Sabine Seufert, *AFEIC: Advanced Forensic Ext4 Inode Carving*, *https://dfrws.org/presentation/afeic-advanced-forensic-ext4-inode-carving/*.

Analiza btrfs

Twórcą systemu *btrfs* jest Chris Mason, który opracował go w trakcie pracy w Oracle. W 2007 roku system został umieszczony na liście dyskusyjnej Linux Kernel Mailing List (LKML). Społeczność Linuksa potrzebowała czegoś lepszego od starzejącego się *ext3*, a z różnych powodów systemy ReiserFS i *zfs* nie stanowiły w tamtym czasie realnej alternatywy. Od tego czasu system *btrfs* stał się częścią głównej linii rozwojowej jądra Linuksa i zyskał popularność. Obecnie jest standardowym systemem plików w SUSE i Fedorze, a do przechowywania wewnętrznych danych wykorzystuje go Facebook. Na systemie tym polegają również firmy zajmujące się przechowywaniem danych, takie jak Synology.

Wśród wielu nowoczesnych funkcjonalności oferowanych przez *btrfs* są: zarządzanie wieloma urządzeniami, podwoluminy i migawki CoW. Ze względu na ich obecność *btrfs* nie wymaga oddzielnej warstwy zarządzania woluminami, takiej jak LVM. System *btrfs* jest aktywnie rozwijany, a nowo wprowadzone funkcjonalności są prezentowane na stronie głównej projektu pod adresem *https://btrfs.wiki.kernel.org/index.php/Main_Page*.

W chwili pisania tego tekstu obsługa *btrfs* w oprogramowaniu kryminalistycznym jest kiepska. Większość podstawowych pakietów do prowadzenia analiz kryminalistycznych nie obsługuje tego systemu plików. Nie wspiera go nawet pakiet TSK. Kilka eksperymentalnych implementacji obsługi *btrfs* w TSK jest dostępnych w serwisie GitHub. Znajdziesz w nim też starszy *pull request* z prośbą o dodanie obsługi *btrfs* do TSK (*https://github.com/basicmaster/sleuthkit/*) oraz narzędzie

obsługujące ten system, które używa bibliotek TSK i naśladuje działanie jego poleceń (*https://github.com/shujianyang/btrForensics/*). Te narzędzia mogą, ale nie muszą zadziałać, możesz z nich skorzystać na własne ryzyko.

W tym podrozdziale wykorzystam kombinację narzędzi stworzonych przez zespół *btrfs* (pakiet `btrfs-progs`) oraz badań przeprowadzonych w Fraunhofer FKIE, których wyniki przedstawiono na konferencji DFRWS USA w 2018 roku (*https://www.sciencedirect.com/science/article/pii/S1742287618301993/*). Odmianę TSK oferującą wsparcie dla *btrfs* możesz pobrać z serwisu GitHub (*https://github.com/fkie-cad/sleuthkit/*).

Przykłady pokazane w tym podrozdziale wykorzystują różnorodne narzędzia i techniki. Każde narzędzie może wymagać innej formy dostępu do systemu plików *btrfs*. Aby uniknąć nieporozumień, zamieszczam poniżej listę nazw urządzeń, plików i katalogów, które wykorzystałem w kolejnych przykładach:

- ***image.raw*** — surowy obraz dysku pozyskany zgodnie z zasadami kryminalistyki (należy podać przesunięcie dla systemu plików);
- ***partimage(X).raw*** — oddzielnie wyodrębnione obrazy partycji zawierające tylko system plików;
- ***/dev/loopX*** — urządzenie blokowe (w */dev/*) fizycznie podłączone do komputera lub urządzenie typu *loop*;
- ***/evidence/*** — ścieżka do zamontowanego systemu plików *btrfs*;
- ***pool/*** lub ***poolm/*** — katalog zawierający pulę (jeden lub więcej) obrazów partycji *btrfs*.

W tym podrozdziale zakładam, że ścieżki odnoszą się do bieżącego katalogu roboczego.

Metadane systemu plików — superblok

System plików *btrfs* można zidentyfikować na podstawie sygnatury z superbloku. Główny superblok systemu *btrfs* znajduje się w odległości 65 536 bajtów (`0x10000`) od początku systemu plików. Na dysku z 512-bajtowymi sektorami odpowiada mu sektor 128 (licząc od początku partycji). Ośmiobajtowa sygnatura, która identyfikuje system plików *btrfs*, to `_BHRfS_M`. Pokazano ją poniżej wraz z odpowiadającą jej reprezentacją szesnastkową:

```
5F 42 48 52 66 53 5F 4D   _BHRfS_M
```

Sygnatura znajduje się 64 bajty (`0x40`) od początku superbloku, czyli 65 600 bajtów (`0x10040`) od początku partycji zawierającej system plików. Wyszukiwanie tej sygnatury we wszystkich sektorach dysku może ujawnić kopie superbloku lub inne systemy plików *btrfs* do dalszej analizy.

W odmianie TSK rozwijanej przez Fraunhofer FKIE wprowadzono kilka nowych flag w poleceniach służących do analizy systemu plików. W tym wariancie

TSK zakłada się, że obrazy kryminalistyczne partycji *btrfs* znajdują się w katalogu „puli" (którym w poniższych przykładach jest *pool/*). Położenie katalogu określa się za pomocą opcji `-P`. W poniższym przykładzie wykorzystałem `fsstat` do wyświetlenia zawartości superbloku, który zawiera kilka interesujących z kryminalistycznego punktu widzenia informacji:

```
$ fsstat -P pool/
Label: My Stuff ❶
File system UUID: EA920473-EC49-4F1A-A037-90258D453DB6 ❷
Root tree root address:  5406720
Chunk tree root address: 1048576
Log treerootaddress:     0
Generation:              20 ❸
Chunk root generation:   11
Total bytes:             3293898240
Number of devices:       1

Device UUID:             22D40FDB-C768-4623-BCBB-338AC0744EC7 ❹
Device ID:               1
Device total bytes:      4293898240 ❺
Device total bytes used: 457179136 ❻

Total size: 3GB
Used size: 38MB

The following subvolumes or snapshots are found: ❼
256       Documents
257       Videos
259       .snapshot
260       Confidential
```

Użytkownik ma możliwość nadania etykiety (❶; maksymalnie 256 znaków), której nazwa może być pomocnym artefaktem w trakcie dochodzenia. Pierwszy UUID (❷) jest unikalnym identyfikatorem systemu plików *btrfs*. Drugi UUID (❹) to unikalny identyfikator urządzenia *btrfs*. ❺ to całkowita pojemność dysku, a ❻ to liczba użytych bajtów. Wartości te powinny pokrywać się z innymi danymi dotyczącymi pojemności znalezionymi w trakcie badania (na przykład w tablicy partycji). Wartość w polu *Generation* (❸) jest aktualizowana po wprowadzeniu zmian, dzięki czemu system plików „wie", która kopia (spośród wszystkich nadmiarowych kopii) superbloku jest najnowsza. Na końcu listingu widoczna jest lista podwoluminów i migawek (❼; są one opisane w jednym z kolejnych punktów).

Te same informacje wraz z kilkoma dodatkowymi statystykami i flagami (które są mniej przydatne w kontekście dochodzenia) wyświetli też polecenie `btrfs inspect-internal dump-super partimage.raw`. Polecenie `btrfs inspect-internal` pozwala analizować różne niskopoziomowe, techniczne artefakty dotyczące systemu plików i sposobu przechowywania struktur na dysku. Więcej informacji na jego temat znajdziesz w sekcji `btrfs-inspect-internal(8)` manuala. W odróżnieniu od *ext4* superblok *btrfs* nie zawiera żadnych znaczników czasu.

Metadane pliku — *i*-węzły

Struktura *i*-węzła w systemie *btrfs* jest opisana na stronie *kernel.org* (*https://btrfs.wiki.kernel.org/index.php/Data_Structures#btrfs_inode_ref*). W przeciwieństwie do *ext4* i *xfs* *i*-węzeł w *btrfs* zawiera niewiele informacji i przechowuje pewne informacje o plikach w różnych strukturach drzewiastych. *i*-węzeł w *btrfs* zawiera następujące informacje:

- `generation` — licznik zmian (wartość rośnie),
- `transid` — ID transakcji,
- `size` — rozmiar pliku w bajtach,
- `nbytes` — rozmiar zaalokowanych bloków w bajtach (w przypadku katalogów *0*),
- `nlink` — liczba dowiązań,
- `uid` — identyfikator właściciela pliku,
- `gid` — identyfikator grupy pliku,
- `mode` — uprawnienia,
- `rdev` — jeśli *i*-węzeł jest urządzeniem, to w tym polu przechowywane są związane z nim numery główny i poboczny (ang. *minor/major number*),
- `flags` — flagi *i*-węzła (opisane w następnym akapicie),
- `sequence` — pole zapewniające zgodności z NFS (początkowo ma wartość *0*, wartość jest zwiększana za każdym razem, gdy zmienia się wartość `mtime`),
- `atime` — znacznik czasu ostatniego dostępu,
- `ctime` — znacznik czasu ostatniej zmiany *i*-węzła,
- `mtime` — znacznik czasu ostatniej zmiany zawartości pliku,
- `otime` — znacznik czasu utworzenia *i*-węzła (utworzenie pliku).

Większość z tych elementów jest Ci już znana i można je znaleźć również w innych systemach plików. Wartość `sequence` zapewniająca zgodność z NFS jest zwiększana za każdym razem, gdy zmienia się zawartość pliku (`mtime`). W trakcie dochodzenia wiedza o tym, ile razy (lub czy) plik był modyfikowany, może być interesująca. Możesz ją również wykorzystać do sprawdzenia, jak często wprowadzano zmiany w pliku lub katalogu w porównaniu z innymi plikami.

Flagi *i*-węzła[12] opisują dodatkowe atrybuty związane z plikiem. W dokumentacji *btrfs* zdefiniowano następujące flagi:

- `NODATASUM` — sumy kontrolne tego *i*-węzła nie będą obliczane;
- `NODATACOW` — nie wykonuj kopiowania przy zapisie dla ekstentów w tym *i*-węźle, jeżeli liczba referencji wynosi 1;
- `READONLY` — *i*-węzeł tylko do odczytu niezależnie od uprawnień Uniksa lub własności (zastąpiona przez `IMMUTABLE`);

[12] W zależności od wersji jądra część z tych flag może nie być zaimplementowana lub używana.

- NOCOMPRESS — brak kompresji *i*-węzła;
- PREALLOC — *i*-węzeł zawiera wstępnie zaalokowane ekstenty;
- SYNC — operacje na tym *i*-węźle będą wykonywane synchronicznie;
- IMMUTABLE — *i*-węzeł tylko do odczytu, niezależnie od uprawnień Uniksa lub własności;
- APPEND — *i*-węzeł działa w trybie dołączania;
- NODUMP — *i*-węzeł nie jest kandydatem do wykonania zrzutu za pomocą dump (patrz dump(8));
- NOATIME — brak aktualizacji atime (znacznika czasu ostatniego dostępu);
- DIRSYNC — operacje na katalogach będą wykonywane synchronicznie;
- COMPRESS — kompresja zawartości *i*-węzła.

Atrybut NOATIME może wpływać na analizę kryminalistyczną, ponieważ powoduje brak aktualizacji znaczników czasu ostatniego dostępu do pliku przez jądro systemu.

Możliwość utworzenia zrzutów całej zawartości *i*-węzłów w systemie plików *btrfs* będzie zależna od możliwości konkretnego narzędzia kryminalistycznego. Na przykład narzędzie istat opracowane w Fraunhofer FKIE wyświetla niewiele informacji (opcja -P jest opisana w kolejnym punkcie):

```
$ istat -P pool/ 257
Inode number: 257
Size: 29
Name: secret.txt

Directory Entry Times(local);
Created time:  Sun Nov 29 16:55:34 2020
Accesstime:    Sun Nov 29 16:56:412020
Modified time: Sun Nov 29 16:55:25 2020
```

W przypadku niektórych dochodzeń ten poziom szczegółowości może być niewystarczający. Aby uzyskać więcej szczegółów, zastosuj polecenie btrfs inspect-
↪internal:

```
$ btrfs inspect-internal dump-tree pool/partimage.raw
...
    item 8 key (257 INODE_ITEM 0) itemoff 15721 itemsize 160
            generation 10 transid 12 size 29 nbytes 29
            block group 0 mode 100640 links 1 uid 1000 gid 1000 rdev 0
            sequence 15 flags 0x0(none)
            atime 1606665401.870699900 (2020-11-29 16:56:41)
            ctime 1606665334.900190664 (2020-11-29 16:55:34)
            mtime 1606665325.786787936 (2020-11-29 16:55:25)
            otime 1606665325.786787936 (2020-11-29 16:55:25)
```

```
    item 9 key (257 INODE_REF 256) itemoff 15701 itemsize 20
            index 4 namelen 10 name: secret.txt
...
```

Powyższe polecenie wykonuje zrzut metadanych z całego systemu plików. Jeśli znasz numer *i*-węzła, to możesz przeszukać dane wyjściowe w celu znalezienia interesujących Cię informacji. Na powyższym listingu pokazałem *i*-węzeł o numerze 257 wraz z jego pełną strukturą.

W zależności od rozmiaru pliku i liczby obiektów wykonanie zrzutu wszystkich metadanych za pomocą polecenia `btrfs inspect-internal` może dać w wyniku ogromną ilość danych. Jeśli planujesz przeprowadzić bardziej złożoną analizę lub wielokrotne wyszukiwania, łatwiej będzie Ci zapisać wyniki w osobnym pliku.

Wiele urządzeń i podwoluminów

Btrfs szeroko wykorzystuje identyfikatory UUID dla różnych obiektów tworzących ten system plików. Podobnie czyni też schemat partycjonowania GPT, który używa identyfikatorów UUID dla różnych składników pamięci. Aby rozjaśnić sytuację i ułatwić identyfikację różnych elementów, poniżej wymieniłem niektóre zastosowania identyfikatorów UUID:

- Identyfikatory UUID dla każdego urządzenia GPT (dysk z partycją GPT).
- Identyfikatory UUID dla każdej partycji GPT (*PARTUUID*).
- Identyfikatory UUID dla każdego systemu plików *btrfs*.
- Identyfikatory UUID dla każdego urządzenia *btrfs* (dysku będącego częścią systemu plików *btrfs*; *UUID_SUB*).
- Identyfikatory UUID dla każdego podwoluminu *btrfs* lub migawki.

Identyfikatory UUID mogą pełnić w raporcie kryminalistycznym funkcję unikalnych identyfikatorów. Można je też wykorzystać do powiązania ze sobą dowodów z wielu źródeł. Identyfikatory UUID odgrywają ważną rolę podczas analizowania systemów *btrfs* składających się z wielu urządzeń.

Jednym z elementów *btrfs* jest mechanizm zarządzania woluminami. Pojedynczy system plików *btrfs* może obejmować wiele urządzeń fizycznych. Sposób, w jaki dane i metadane są replikowane na urządzeniach (poziomy RAID itd.), określa „profil" systemu. Więcej informacji na temat tworzenia systemów plików *btrfs* znajdziesz w sekcji `mkfs.btrfs(8)` manuala.

Twórcy *zfs* do opisu wielu urządzeń stosują określenie **pula** (ang. *pool*). Wersja TSK opracowana na Fraunhofer FKIE korzysta z tej samej terminologii i oferuje polecenie `pls` wyświetlające informacje o kolekcji obrazów zapisanych w katalogu, w którym przechowujemy naszą pulę. Inne polecenia tej odmiany TSK zawierają opcje pozwalające określić lokalizację puli (`-P`), numer transakcji/generacji (`-T`) i podwolumin, na którym ma zostać uruchomione polecenie (`-S`). W tym przykładzie w katalogu *poolm/* umieściłem kilka obrazów partycji, które zostały zabezpieczone na trzech dyskach:

```
$ ls poolm/
partimage1.raw partimage2.raw partimage3.raw
$ pls poolm/
FSID:              CB9EC8A5-8A79-40E8-9DDB-2A54D9CB67A9 ❶
System chunks:     RAID1 (1/1) ❷
Metadata chunks:   RAID1 (1/1)
Data chunks:       Single (1/1)
Number of devices: 3 (3 detected) ❸
------------------------------------------------
ID:                 1 ❹
GUID:               2179D1FD-F94B-4CB7-873D-26CE05B41662

ID:                 2
GUID:               0F784A29-B752-46C4-8DBC-C8E2455C7A13

ID:                 3
GUID:               31C19872-9707-490D-9267-07B499C5BD06
...
```

W danych wyjściowych widoczny jest UUID systemu plików (❶), liczba urządzeń wchodzących w skład systemu (❸), użyte profile (takie jak *RAID1*; ❷) oraz UUID (lub GUID) każdego urządzenia *btrfs* (❹). Pokazane powyżej identyfikatory UUID urządzeń są częścią systemu plików *btrfs*. Identyfikatory te różnią się od identyfikatorów znajdujących się w tablicy partycji GPT.

Podwoluminy to funkcjonalność *btrfs*, która pozwala podzielić system plików na oddzielne części logiczne, które mogą mieć swoje własne cechy. Podział na podwoluminy nie odbywa się w warstwie bloków/ekstentów, dzięki czemu bloki/ekstenty danych mogą być współdzielone między podwoluminami. W ten sposób zaimplementowana jest też funkcjonalność migawek. W poprzednim punkcie pokazałem przykład użycia `fsstat` do wyświetlenia informacji o superbloku. Polecenie to wyświetla również podwoluminy znajdujące się w badanym systemie plików:

```
$ fsstat -P pool/
...
The following subvolumes or snapshots are found:
256       Documents
257       Videos
259       .snapshot
260       Confidential
```

Podwoluminy posiadają identyfikatory i własne UUID. Na poziomie plików i katalogów możemy analizować podwoluminy tak, jakby były one oddzielnymi systemami plików (w obrębie każdego podwoluminu pliki mają nawet unikatowe *i*-węzły). Na niższym poziomie pliki znajdujące się w różnych podwoluminach mogą współdzielić bloki/ekstenty.

W niektórych przypadkach możesz chcieć zamontować system plików *btrfs* na komputerze kryminalistycznym, np. aby przejrzeć jego zawartość za pomocą

narzędzi do zarządzania plikami lub jakichś aplikacji (przeglądarek i programów biurowych) lub aby uruchomić na nim polecenia analizujące system plików *btrfs*, które działają jedynie na zamontowanym systemie plików. Pokażę teraz, jak zamontować obraz pojedynczej partycji (*pool/partimage.raw*) w katalogu */evidence/*. Proces montowania składa się z dwóch etapów:

```
$ sudo losetup -f --show -r pool/partimage.raw
/dev/loop0
$ sudo mount -o ro,subvol=/ /dev/loop0 /evidence/
```

Pierwsze polecenie tworzy urządzenie typu *loop* tylko do odczytu skojarzone z obrazem partycji. Drugie polecenie montuje urządzenie *loop0* w trybie tylko do odczytu w katalogu */evidence/*. W drugim poleceniu jawnie określam główny podwolumin *btrfs*, tak aby polecenie nie użyło żadnego domyślnego podwoluminu. Teraz mogę bezpiecznie korzystać z zamontowanego w katalogu */evidence/* podwoluminu.

Polecenie `btrfs subvolume` pozwala również wyświetlić listę podwoluminów i migawek znalezionych w systemie plików. Polecenie to działa na zamontowanym systemie plików:

```
$ sudo btrfs subvolume list /evidence/
ID 256 gen 19 top level 5 path Documents
ID 257 gen 12 top level 5 path Videos
ID 259 gen 13 top level 5 path .snapshot
ID 260 gen 19 top level 256 path Documents/Confidential
```

Każdemu podwoluminowi nadawany jest identyfikator (znajdziesz go również w wyjściu z polecenia `stat` lub `ls -i`). Widoczny jest też inkrementowany numer generacji. Wartość *top level* to identyfikator nadrzędnego podwoluminu, a *path* to ścieżka do głównego katalogu zamontowanego systemu plików (w tym przypadku */evidence/*).

Polecenie `btrfs subvolume` pozwala wyświetlić więcej informacji o wybranym podwoluminie. Poniżej pokazano metadane dla podwoluminu *Documents*:

```
$ sudo btrfs subvolume show /evidence/Documents/
Documents
        Name:                   Documents
        UUID:                   77e546f8-9864-c844-9edb-733da662cb6c
        Parent UUID:            -
        Received UUID:          -
        Creation time:          2020-11-29 16:53:56 +0100
        Subvolume ID:           256
        Generation:             19
        Gen at creation:        7
        Parent ID:              5
```

```
        Top level ID:               5
        Flags:                      -
        Snapshot(s):
```

Na powyższym listingu widoczny jest identyfikator UUID podwoluminu wraz ze znacznikiem czasu jego utworzenia i innymi flagami. Jeśli podwolumin zawiera migawki, to polecenie również je wyświetli.

Migawki (ang. *snapshots*) są jednym z najważniejszych elementów *btrfs*. Migawki implementują koncepcję kopiowana przy zapisie, aby stworzyć migawkę podwoluminu z określonego punktu w czasie. Oryginalny podwolumin pozostaje dostępny dla użytkownika i jest dalej wykorzystywany, a migawka zostaje zapisana w nowo utworzonym podwoluminie. Migawki mogą być elementami tylko do odczytu i są zazwyczaj wykorzystywane do tworzenia kopii zapasowych lub przywracania systemu do poprzedniego stanu. Można je też wykorzystać do „zamrożenia" systemu plików w trakcie analizy kryminalistycznej typu *live* (w przypadku *btrfs* proces zachodzi na poziomie pliku, a nie na poziomie bloków/sektorów). Migawki są interesujące z punktu widzenia informatyki śledczej, ponieważ mogą zawierać poprzednie wersje plików. Analiza plików w migawce odbywa się w ten sam sposób co w każdym innym podwoluminie. Na przykład znacznik czasu utworzenia migawki można znaleźć (w identyczny jak poprzednio sposób) za pomocą polecenia `btrfs subvolume`:

```
$ sudo btrfs subvolume show /evidence/.snapshot/
.snapshot
        Name:                       .snapshot
        UUID:                       57912eb8-30f9-1948-b68e-742f15d9408a
...
        Creation time:              2020-11-29 16:58:28 +0100
...
```

Pliki z migawki, które nie uległy zmianie, oraz odpowiadające im pliki znajdujące się w oryginalnym podwoluminie współdzielą te same bloki.

Wyświetlanie listy plików i wyodrębnianie

Narzędzie kryminalistyczne zapewniające pełną obsługę *btrfs* powinno umożliwiać przeglądanie, badanie i wyodrębnianie plików w standardowy sposób. Główną różnicą w stosunku do innych systemów plików jest obecność podwoluminów. Podczas badania plików i katalogów każdy podwolumin musi być traktowany jak osobny system plików (z uwzględnieniem możliwości współdzielenia bloków).

W chwili pisania tego tekstu podstawowa wersja TSK ciągle nie oferuje wsparcia dla *btrfs*. Podstawową (eksperymentalną) obsługę tego systemu plików oferuje wariant TSK opracowany na Fraunhofer FKIE. Oto kilka przykładów:

```
$ fls -P pool/
r/r 257:      secret.txt
$ fls -P pool/ -S .snapshot
```

```
r/r 257:      secret.txt
$ fls -P pool/ -S Documents
r/r 257:      report.pdf
$ fls -P pool/ -S Videos
r/r 257:      phiberoptik.mkv
```

Polecenie `fls` uruchomione z opcją `-P` pozwala wyświetlić listę plików znajdujących się w obrazach umieszczonych w katalogu *pool/*. Opcja `-S` pozwala odwołać się do konkretnego podwoluminu/migawki. Z powodu okoliczności numery *i*-węzłów w tym przykładzie są takie same w różnych podwoluminach. Jest to możliwe, ponieważ każdy podwolumin posiada własną tablicę *i*-węzłów.

Pliki można wyodrębnić za pomocą polecenia `icat`. Polecenie to należy wywołać z tymi samymi opcjami `-P` i `-S`. Konieczne jest również przekazanie numeru i-węzła:

```
$ icat -P pool/ 257
The new password is "canada101"
$ icat -P pool/ -S .snapshot 257
The password is "canada99"
$ icat -P pool/ -S Documents 257 > report.pdf
$ icat -P pool/ -S Videos 257 > phiberoptik.mkv
```

Zawartość pliku wyekstrahowanego za pomocą narzędzia `icat` może być wyświetlona na ekranie lub zapisana do pliku. Zawartość pliku można następnie zbadać na lokalnej maszynie kryminalistycznej.

Za pomocą narzędzia `undelete-btrfs` (*https://github.com/danthem/undelete-btrfs/*) możesz spróbować odzyskać usunięte pliki w systemie plików *btrfs*. Narzędzie to jest skryptem, który wykorzystuje polecenia `btrfs restore` oraz `btrfs-find-root` w celu wyszukania i wyodrębnienia usuniętych plików. Korzystasz na własne ryzyko.

Teoretycznie analiza kryminalistyczna systemów plików *btrfs* pozwala na odzyskanie usuniętych lub wcześniej zapisanych danych z dużym prawdopodobieństwem. Filozofia kopiowania przy zapisie pozwala uniknąć nadpisywania starych danych i preferuje tworzenie nowych bloków/ekstentów zamiast aktualizacji tych już użytych. Jawnie utworzone migawki tworzą historię wersji plików i katalogów z poprzednią zawartością i metadanymi. Z czasem na rynku komercyjnym oraz w społeczności wolnego i otwartego oprogramowania pojawią się narzędzia kryminalistyczne do prowadzenia takich analiz. Zanim to nastąpi, potrzebnych jest więcej badań naukowych dotyczących analizy kryminalistycznej systemu *btrfs*.

Analiza xfs

System plików *xfs* został opracowany na początku lat 90. XX wieku przez firmę Silicon Graphics (SGI). Pierwotnie był on przeznaczony dla systemu SGI IRIX UNIX. W 2000 roku firma SGI opublikowała *xfs* na licencji GPL, co doprowadziło do jego przeportowania na Linuksa. Później system plików *xfs* stał się częścią głównej linii rozwojowej jądra Linuksa i jest dziś obsługiwany przez każdą z najważniejszych dystrybucji. *Xfs* jest domyślnym systemem plików w Red Hat Enterprise Linux. Najbardziej wiarygodnym źródłem informacji o tym systemie jest związana z nim strona wiki w serwisie *kernel.org* (*https://xfs.wiki.kernel.org/*).

Niewiele narzędzi kryminalistycznych obsługuje system plików *xfs* — w odróżnieniu od *ext4*. W chwili pisania tego tekstu obsługę systemu plików *xfs* oferuje AccessData Imager w wersjach 4.3 i nowszych oraz XWays Forensics, które wydaje się oferować pełne wsparcie dla tego systemu plików. Gdy piszę te słowa, systemu *xfs* nie obsługuje nawet TSK, chociaż w serwisie GitHub istnieje kilka próśb o dołączenie do głównej linii rozwojowej implementacji *xfs* stworzonej przez społeczność. W niektórych przykładach tego podrozdziału wykorzystam odmianę TSK obsługującą *xfs*, której twórcą jest Andrey Labunets (patrz *https://github.com/isciurus/sleuthkit.git/*).

Twórcy systemu *xfs* udostępniają narzędzia, takie jak `xfs_db` i `xfs_info`, przeznaczone do debugowania i rozwiązywania problemów z tym systemem plików. Narzędzia te oferują wiele funkcjonalności, które mogą być przydatne w trakcie analizy kryminalistycznej tego systemu. Więcej informacji na ich temat znajdziesz w sekcjach `xfs_info(8)` i `xfs_db(8)` manuala.

Metadane systemu plików — superblok

Xfs *to dobrze udokumentowany system plików. Powiązane z nim struktury da*nych można analizować pod kątem artefaktów kryminalistycznych. Sekcja `xfs(5)` manuala zawiera dobre wprowadzenie do szczegółów montowania systemu, jego projektu oraz różnych atrybutów. Struktury danych wykorzystywane w *xfs* są szczegółowo opisane w dokumencie *XFS Algorithms & Data Structures* (*https://mirrors.edge.kernel.org/pub/linux/utils/fs/xfs/docs/xfs_filesystem_structure.pdf*).

Systemy plików *xfs* można rozpoznać po następującej sygnaturze znajdującej się w superbloku:

```
0x58465342 XFSB
```

Sygnatura znajduje się na początku pierwszego sektora systemu plików. Istnieje ponad 50 różnych sygnatur pozwalających zidentyfikować różne obszary systemu plików *xfs* (patrz rozdział 7. dokumentu *XFS Algorithms & Data Structures*).

Aby wyświetlić metadane zapisane w superbloku, możesz skorzystać z narzędzia `xfs_db`. W poniższym przykładzie flaga `-r` pozwala zagwarantować, że operacja będzie operacją w trybie tylko do odczytu, dwie flagi `-c` służą do przekazania nazw poleceń potrzebnych do wyświetlenia superbloku, a *partimage.raw* to plik obrazu kryminalistycznego:

```
$ xfs_db -r -c sb -c print partimage.raw
magicnum = 0x58465342
blocksize = 4096
dblocks = 524288
...
uuid = 75493c5d-3ceb-441b-bdee-205e5548c8c3
logstart = 262150
...
fname = "Super Secret"
...
```

Większość superbloku *xfs* zajmują flagi, statystyki, liczniki bloków itd. Niektórej z nich są interesujące z punktu widzenia analizy kryminalistycznej. Rozmiar bloku (*blocksize*) i całkowitą liczbę bloków (*dblocks*) warto porównać z rozmiarem partycji, na której znajduje się system plików. Pole *uuid* zawiera unikalny ciąg identyfikujący. 12-znakowa etykieta lub nazwa systemu plików (*fname*) może być określona przez właściciela systemu i, jeżeli istnieje, może być przedmiotem zainteresowania śledczych. Więcej informacji o różnych parametrach, które można ustawić podczas tworzenia systemów plików *xfs*, znajdziesz w sekcji `mkfs.xfs(8)` manuala.

Podsumowanie informacji o systemie plików z superbloku możesz też wyświetlić za pomocą polecenia `fsstat` z odmiany TSK obsługującej *xfs*:

```
$ fsstat partimage.raw
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: XFS
Volume Name: Super Secret

Volume ID: 75493c5d-3ceb-441b-bdee-205e5548c8c3
Version: V5,NLINK,ALIGN,DIRV2,LOGV2,EXTFLG,MOREBITS,ATTR2,LAZYSBCOUNT,
PROJID32BIT,CRC,FTYPE
Features Compat: 0
Features Read-Only Compat: 5
Read Only Compat Features: Free inode B+tree, Reference count B+tree,
Features Incompat: 3
InCompat Features: Directory file type, Sparse inodes,
CRC: 3543349244
...
```

Dane wyjściowe z `fsstat` mają bardziej opisowy charakter niż te z `xfs_db`, ale są to te same informacje.

Superblok *xfs* jest krótki (jeden sektor) i nie zawiera żadnych dodatkowych informacji znanych z innych systemów plików (takich jak znaczniki czasu, ostatni punkt montowania itd.).

Metadane pliku — *i*-węzły

System plików *xfs* wykorzystuje tę samą koncepcję *i*-węzłów, co inne uniksopodobne systemy plików. *i*-węzeł zawiera metadane i informacje o blokach (lub ekstentach) dysku związanych z plikiem. (Struktura *i*-węzłów jest zdefiniowana w rozdziale 7. dokumentu *XFS Algorithms & Data Structures*).

Metadane interesującego Cię *i*-węzła możesz wyświetlić za pomocą polecenia `xfs_db`, któremu musisz przekazać numer węzła. W poniższym przykładzie ze względu na spację oddzielającą polecenie od numeru *i*-węzła parametr `"i-node 133"` jest umieszczony w cudzysłowie. Parametry `print` i nazwa obrazu partycji są takie same jak w poprzednim przykładzie:

```
$ xfs_db -r -c "inode 133" -c print partimage.raw
core.magic = 0x494e
core.mode = 0100640 (1)
core.version = 3
core.format = 2 (extents)
core.nlinkv2 = 1
core.onlink = 0
core.projid_lo = 0
core.projid_hi = 0
core.uid = 0 (2)
core.gid = 0
core.flushiter = 0
core.atime.sec = Mon Nov 30 19:57:54 2020 (3)
core.atime.nsec = 894778100
core.mtime.sec = Mon Nov 30 19:57:54 2020 (4)
core.mtime.nsec = 898113100
core.ctime.sec = Mon Nov 30 19:57:54 2020 (5)
core.ctime.nsec = 898113100
core.size = 1363426
core.nblocks = 333
...
core.immutable = 0
core.append = 0
core.sync = 0
core.noatime = 0
core.nodump = 0
...
core.gen = 1845361178
...
v3.crtime.sec = Mon Nov 30 19:57:54 2020 (6)
v3.crtime.nsec = 894778100
v3.inumber = 133
v3.uuid = 75493c5d-3ceb-441b-bdee-205e5548c8c3 (7)
...
```

Na powyższym listingu pokazano dane wyjściowe zawierające metadane pliku, którego *i*-węzeł ma numer 133. Są w nich widoczne cztery znaczniki czasu: czas ostatniego dostępu (❸; `atime`), czas ostatniej modyfikacji zawartości pliku (❹;

mtime), czas ostatniej modyfikacji metadanych (❺; ctime) oraz znacznik czasu utworzenia pliku (❻; crtime, jest on dostępny w trzeciej wersji *xfs*). Wyświetlane są również informacje o właścicielu pliku (❷; *uid/gid*), prawa dostępu (❶; *mode*) oraz inne atrybuty. UUID z punktu ❼ jest identyfikatorem superbloku. Nie jest to unikalna wartość identyfikująca plik lub *i*-węzeł.

Polecenie istat z odmiany TSK obsługującej *xfs* wyświetla podobne informacje, ale są one zapisane w innym formacie:

```
$ istat partimage.raw 133
Inode: 133
Allocated
uid/gid:0/0
mode: rrw-r-----
Flags:
size: 1363426
num of links: 1

Inode Times:
Accessed:       2020-11-30 19:57:54.894778100 (CET)
File Modified:  2020-11-30 19:57:54.898113100 (CET)
Inode Modified: 2020-11-30 19:57:54.898113100 (CET)
FileCreated:    2020-11-30 19:57:54.894778100 (CET)

Direct Blocks:
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
...
```

W tym sformatowanym wyjściu znajduje się również lista zaalokowanych bloków używanych przez plik.

Wyświetlanie listy plików i wyodrębnianie

Przedstawione poniżej przykłady pokrywają się z poprzednimi przykładami użycia TSK. Umieściłem je jedynie w celu zapewnienia kompletności opisu. Polecenie fls z odmiany TSK obsługującej *xfs* pozwala wyświetlić listę plików z systemu w taki sam sposób jak standardowa wersja fls:

```
$ fls -pr partimage.raw
d/d 131:        Documents
r/r 132:        Documents/passwords.txt
r/r 133:        report.pdf
d/d 1048704:    Other Stuff
```

Za pomocą opcji -l możesz wyświetlić również rozmiar pliku, informacje o właścicielu oraz znaczniki czasu. Na powyższym listingu widoczne są też numery *i*-węzłów każdego pliku i katalogu.

Do wyodrębnienia plików z obrazu kryminalistycznego możesz wykorzystać numery *i*-węzłów:

```
$ icat partimage.raw 132
The new password is "Supercalifragilisticexpialidocious"
$ icat partimage.raw 133 > report.pdf
```

W pierwszym przykładzie dane wyjściowe są wyświetlane w terminalu. Drugie wywołanie pokazuje, jak przekierować wyodrębnione dane do pliku na maszynie kryminalistycznej.

Xfs posiada również system rejestrowania zdarzeń (dziennik). Omówienie analizy dziennika i innych niskopoziomowych technik wykracza poza zakres tej książki. Dodatkowe informacje na temat analizy kryminalistycznej systemu *xfs* znajdziesz w pięcioczęściowej serii wpisów na blogu Hala Pomeranza (*https://righteousit.wordpress.com/2018/05/21/xfs-part-1-superblock/*).

W serwisie GitHub znajdziesz też inne projekty związane z analizą kryminalistyczną systemu *xfs*, na przykład *https://github.com/ianka/xfs_undelete/* i *https://github.com/aivanoffff/xfs_untruncate/*. Rozwijane w ich ramach narzędzia mogą zadziałać na Twoim obrazie kryminalistycznym, ale nie muszą. Korzystasz na własne ryzyko.

Analiza wymiany systemu Linux

Analiza kryminalistyczna wymiany i hibernacji należy do dziedziny kryminalistyki pamięci. Postanowiłem opisać ją w tym rozdziale, ponieważ oba pojęcia wiążą się z danymi, które zostały zapisane w pamięci trwałej i są dostępne w trakcie analizy kryminalistycznej *post mortem*. W tym podrozdziale dowiesz się, jak działają pliki wymiany i jak ustalić ich lokalizację na dysku twardym, oraz zrozumiesz potencjalne artefakty śledcze, które się w nich znajdują.

Identyfikacja i analiza wymiany

Zarządzanie pamięcią stanowi wyzwanie od początków informatyki. Komputery dysponują ograniczoną ilością szybkiej pamięci ulotnej (RAM), po której zapełnieniu system ulega awarii lub wykorzystuje techniki jej oczyszczania. Jedną z technik oczyszczania pamięci jest tymczasowe zapisywanie części zawartości pamięci na dysku (który ma znacznie większą pojemność) i odczytywanie jej w razie potrzeby. Czynnością tą kieruje jądro, a proces jest znany pod nazwą *swapping* (pol. wymiana). Po zapełnieniu pamięci RAM poszczególne strony pamięci działającego systemu są zapisywane w specjalnych obszarach dysku, z których można je później odczytać. Jeśli zarówno pamięć, jak i przestrzeń wymiany są pełne, to do oczyszczenia pamięci wykorzystywany jest mechanizm Out-of-Memory Killer (OOM Killer), który na podstawie heurystyk decyduje, które procesy działające w systemie należy zakończyć. O ile w konfiguracji jądra nie wybrano opcji związanej

z wykonywaniem zrzutów zakończonych procesów (`sysctl vm.oom_dump_tasks`), to na dysku nie zostanie zapisane nic, co mogłoby zostać poddane analizie kryminalistycznej.

Obszar wymiany w systemie Linux może mieć formę dedykowanej partycji na dysku lub pliku w systemie plików. Większość dystrybucji Linuksa korzysta z oddzielnej, dedykowanej do tego celu partycji wymiany. W schemacie partycjonowania DOS/MBR typ partycji wymiany systemu Linux to `0x82`. W schemacie GPT identyfikatorem GUID takiej partycji jest `0657FD6D-A4AB-43C4-84E5-`
`↪0933C84B4F4F`. Rozmiary partycji wymiany są zwykle większe lub równe ilości pamięci RAM w systemie.

Jądro systemu musi wiedzieć, których obszarów wymiany ma używać. Informacja ta jest zwykle odczytywania podczas startu systemu z pliku */etc/fstab* lub pliku jednostek *.swap* `systemd`. Plik *fstab* będzie zawierał po jednej linii dla każdej partycji wymiany (zwykle jest tylko jedna, ale może ich być więcej). Poniższe trzy linie z pliku *fstab* demonstrują sposób konfiguracji wymiany.

```
UUID=3f054075-6bd4-41c2-a03d-adc75dfcd26d none swap defaults 0 0
/dev/nvme0n1p3 none swap defaults 0 0
/swapfile none swap sw 0 0
```

Pierwsze dwa wiersze reprezentują partycje wymiany identyfikowane przez UUID i plik urządzenia. W trzeciej widoczne jest użycie w roli obszaru wymiany zwykłego pliku. Aby przeanalizować partycje, możesz wyodrębnić ich zawartość lub przeanalizować je „w miejscu" przy użyciu offsetu uzyskanego z tablicy partycji. W przypadku pliku wymiany możesz go skopiować lub wyodrębnić z obrazu, a następnie przeanalizować.

Partycje wymiany można również skonfigurować za pomocą `systemd`. Plik jednostek `systemd` z rozszerzeniem **.swap* zawiera informacje potrzebne do skonfigurowania urządzenia lub pliku wymiany:

```
# cat /etc/systemd/system/swapfile.swap
[Swap]
What=/swapfile
# ls -lh /swapfile
-rw------- 1 root root 1.0G 23. Nov 06:24 /swapfile
```

Ten prosty, dwuwierszowy plik konfiguracyjny wskazuje na plik *swapfile* o rozmiarze 1 GB znajdujący się w głównym katalogu. Po uruchomieniu systemu plik ten stanie się plikiem wymiany. Więcej informacji znajdziesz w sekcji `systemd.swap(5)` manuala.

Administrator systemu może utworzyć plik o żądanym rozmiarze i oznaczyć go jako plik wymiany (np. jeśli potrzebna jest dodatkowa przestrzeń wymiany lub gdy administrator preferuje użycie pliku zamiast partycji). Nie ma żadnej zestandaryzowanej konwencji nazewnictwa plików wymiany, chociaż w niektórych

dystrybucjach i w wielu poradnikach stosowana jest nazwa *swapfile*. Nie istnieje też żadna standardowa lokalizacja plików wymiany, ale dość często są one umieszczane w katalogu głównym (/).

Partycję (lub plik) wymiany można zidentyfikować za pomocą 10-znakowej sygnatury, która znajduje się 4086 bajtów od początku pliku/partycji (`0xFF6`):

```
00000ff6: 5357 4150 5350 4143 4532 SWAPSPACE2
```

Łańcuch znaków sygnatury to `SWAPSPACE2` lub `SWAP-SPACE`. Sygnatura wskazuje, że partycja lub plik zostały skonfigurowane do użycia jako obszar wymiany (za pomocą polecenia `mkswap`).

Do identyfikacji plików wymiany i wyświetlenia podstawowych informacji z nimi związanych możesz wykorzystać polecenie `file`[13]:

```
# file swapfile
swapfile: Linux swap file, 4k page size, little endian, version 1, size 359674
pages, 0 bad pages, no label, UUID=7ed18640-0569-43af-998b-aabf4446d71d
```

Administrator systemu może wygenerować 16-znakową etykietę pliku wymiany. UUID jest generowany losowo i powinien być unikalny.

Aby przeanalizować wymianę, na maszynie kryminalistycznej możesz zapisać partycję wymiany z dysku podejrzanego (za pomocą polecenia `dd` lub innego podobnego) w postaci obrazu kryminalistycznego. W przypadku pliku wymiany wystarczy po prostu skopiować plik. Partycja lub plik wymiany mogą zawierać fragmenty pamięci procesów, która została tymczasowo zapisana na dysku.

W tej książce omówienie analizy pamięci jest ograniczone do identyfikacji, poszukiwania i wydobywania z pamięci danych, które mogą zawierać wiele interesujących artefaktów. Na przykład analiza ciągów znaków za pomocą narzędzia `bulk_extractor` (*https://forensicswiki.xyz/wiki/index.php?title=Bulk_extractor*) pozwala wyodrębnić z nich następujące elementy:

- numery kart kredytowych i informacje z *track* 2,
- nazwy domen,
- adresy e-mail,
- adresy IP,
- adresy MAC kart sieciowych,
- adresy URL,
- numery telefonów,
- dane EXIF z plików multimedialnych (zdjęcia i filmy),
- ciągi znaków dopasowane do wprowadzonych przez użytkownika wyrażeń regularnych.

[13] Ponieważ plik wymiany może zawierać prywatne informacje dotyczące wszystkich użytkowników i procesów w systemie, jest on dostępny tylko dla użytkownika *root*.

Poza wyszukiwaniem ciągów znaków możemy też poszukiwać plików. Do wyodrębnienia plików lub fragmentów plików z obszarów wymiany można wykorzystać standardowe narzędzia przeznaczone do tego celu (na przykład `foremost`).

Hibernacja

Większość współczesnych komputerów PC ma możliwość zawieszenia działania różnych komponentów sprzętowych lub całego systemu w celu oszczędzania energii. Przejście w tryb oszczędzania energii odbywa się zazwyczaj za pomocą interfejsu ACPI i jest kontrolowane przez różne narzędzia działające w przestrzeni użytkownika.

Jeśli rozmiar partycji lub pliku wymiany jest większy lub równy rozmiarowi pamięci RAM zainstalowanej w systemie, to jej zawartość może być w trakcie hibernacji zapisana na dysku. Po zapisaniu całej zawartości pamięci RAM na dysku (w partycji wymiany) system operacyjny może zostać zatrzymany, a maszyna wyłączona. Po ponownym włączeniu komputera uruchamiany jest program rozruchowy (*bootloader*) oraz jądro. Jeśli jądro ustali, że system znajduje się w stanie hibernacji (zawieszenia), to rozpocznie się proces wznawiania systemu, który zakończy się przywróceniem stanu sprzed zawieszenia systemu. Istnieją również inne tryby oszczędzania energii, ale ten jest szczególnie interesujący, ponieważ cała zawartość pamięci jest w nim zapisywana na dysku i może być przeanalizowana.

Za pomocą parametru `resume=` program rozruchowy może przekazać do jądra informacje o partycji (np. ścieżkę */dev/sdaX* lub UUID), w której należy szukać zahibernowanego obrazu. Na przykład:

```
resume=UUID=327edf54-00e6-46fb-b08d-00250972d02a
```

Na powyższym listingu po przekazaniu parametru `resume=` jądro rozpoczyna poszukiwanie urządzenia blokowego o identyfikatorze UUID równym `327edf54-`
`↪00e6-46fb-b08d-00250972d02a` i sprawdza, czy system nie powinien wznowić działania ze stanu hibernacji. Jeśli zamiast partycji używany jest plik, parametr `resume_offset=` zawiera informacje o przesunięciu (w blokach) pliku wymiany względem początku systemu plików.

Partycja (lub plik) wymiany zawiera obraz pamięci hibernacji, jeśli w odległości 4086 bajtów (`0xFF6`) od początku obszaru znajduje się łańcuch znaków `S1SUSPEND`:

```
00000ff6: 5331 5355 5350 454e 4400 S1SUSPEND
```

To przesunięcie to ten sam offset co wartość wspomniana w poprzednim punkcie w trakcie omawiania partycji wymiany. Gdy system przechodzi w stan hibernacji, ciąg `SWAPSPACE2` (lub `SWAP-SPACE`) jest zastępowany przez `S1SUSPEND`. Po wznowieniu systemu wartość ulega ponownej zmianie. Do sprawdzenia istnienia tego

ciągu na pozyskanym obrazie można użyć podstawowych narzędzi kryminalistycznych lub edytora szesnastkowego.

Do sprawdzenia, czy system znajdujący się w obrazie kryminalistycznym partycji wymiany jest w stanie hibernacji, można też wykorzystać polecenie `file`:

```
$ file swapfile
swapfile: Linux swap file, 4k page size, little endian, version 1, size 359674 pages,
0 bad pages, no label, UUID=7ed18640-0569-43af-998b-aabf4446d71d, with SWSUSP1 image
```

Ciąg znaków *with SWSUSP1 image* na końcu danych wyjściowych wskazuje, że plik zawiera obraz partycji wymiany zahibernowanego systemu.

Partycja wymiany systemu w stanie hibernacji z pełnym zrzutem pamięci RAM zawiera wiele informacji, w tym wrażliwe dane (hasła, klucze itd.). W 2005 roku zaproponowano łatkę jądra Linuksa, która wprowadzała szyfrowanie hibernowanych danych (zawierała ona flagę kompilacji `SWSUSP_ENCRYPT`). Wkrótce po opublikowaniu łatka została usunięta ze względu na sprzeciw części programistów tworzących jądro, związany z tym, że klucz deszyfrujący był przechowywany na dysku w niezaszyfrowanej postaci[14]. W ramach rozwiązania społeczność zaproponowała użycie szyfrowania opartego na `dm-crypt`, takiego jak Linux Unified Key Setup (LUKS). Niektóre instalacje Linuksa mogą wykorzystywać LUKS do szyfrowania obszarów wymiany. W takim przypadku przed przeprowadzeniem analizy musisz je odszyfrować. W przypadku LUKS partycja jest zaszyfrowana w warstwie bloków, a jej odszyfrowanie (przy założeniu, że znany jest klucz) za pomocą `cryptsetup` ujawni zawartość zahibernowanej pamięci. (Odszyfrowywanie LUKS jest opisane w następnym podrozdziale).

Do pozyskiwania informacji z obrazów hibernacji możesz wykorzystać techniki opisane w poprzednim punkcie. Interesujące wyniki może również przynieść wyszukiwanie kluczy kryptograficznych.

Przeprowadzono badania nad wykorzystaniem kompresji obrazów wymiany i hibernacji. Kompresja może ograniczać liczbę informacji, które można łatwo wydobyć z pliku lub partycji. Więcej informacji znajdziesz w artykule *https://www.sciencedirect.com/science/article/pii/S1742287614000541*.

Analiza szyfrowania systemu plików

Tradycyjnie największym wyzwaniem stojącym przed informatykami śledczymi jest użycie szyfrowania. Podczas gdy informatyka śledcza skupia się na uzyskaniu dostępu do danych, szyfrowanie polega na ograniczaniu ich dostępności. Konflikt pomiędzy tymi dziedzinami nadal pozostaje nierozwiązany i stanowi przedmiot dyskusji.

[14] Szczegóły sprawy znajdziesz w liście dyskusyjnej LKML z 2005 roku.

Szyfrowanie informacji stało się powszechne i może się odbywać w wielu warstwach:

- Szyfrowanie plików aplikacji: chronione pliki PDF, dokumenty biurowe itd.
- Indywidualne kontenery plików: GPG, zaszyfrowane pliki *.zip*.
- Katalogi: eCryptfs, *fscrypt*.
- Woluminy: TrueCrypt/Veracrypt.
- Urządzenia blokowe: Linux LUKS, Microsoft Bitlocker, Apple FileVault.
- Dyski fizyczne: OPAL/SED (dyski samoszyfrujące).

W tym podrozdziale skoncentruję się na następujących technologiach szyfrowania: LUKS, eCryptfs i *fscrypt* (wcześniej znanym jako *ext4 directory encryption*). Istnieją też inne systemy szyfrowania plików i systemów plików w Linuksie, ale nie będę ich omawiał w tej książce, ponieważ albo nie są one opracowane tylko dla Linuksa, albo są zbyt skomplikowane i rzadko używane.

Odszyfrowanie danych wymaga znajomości hasła lub posiadania kopii klucza kryptograficznego (łańcuch znaków lub plik). Wyzwaniem stojącym przed śledczymi jest znalezienie klucza deszyfrującego. Niektóre metody/środki wykorzystywane do odzyskiwania haseł/kluczy (oczywiście część z nich nie jest wykorzystywana przez śledczych) to:

- Wykorzystanie metody siłowej w celu złamania prostych haseł.
- Wykorzystanie metody siłowej i klastra GPU do przeprowadzenia szybkiego, wyczerpującego wyszukiwania haseł.
- Kryptoanaliza (podatności matematyczne, zmniejszenie przestrzeni kluczy).
- Wyszukiwanie wcześniej zapisanych lub przesłanych haseł.
- Ponowne użycie hasła na wielu kontach lub urządzeniach.
- Prawny wymóg przedstawienia haseł w sądzie.
- Informacje od wściela systemu/administratora (korporacje) lub współpracowników.
- Kopia zapasowa kluczy w środowiskach korporacyjnych.
- Podatności urządzenia, luki w zabezpieczeniach lub backdoory.
- Keyloggery lub podgląd klawiatury (za pomocą kamery wideo lub lornetki).
- Tęczowe tablice (tabele wstępnie obliczonych skrótów kryptograficznych).
- Wyodrębnianie kluczy z pamięci: ataki typu bezpośredni dostęp do szyny PCI, hibernacja.
- Ataki typu *man in the middle*.
- Inżynieria społeczna.
- Kradzież tożsamości biometrycznej (poszkodowany może zostać zmuszony do przekazania np. odcisków palców lub może nie być świadomy, że padł ofiarą ataku).
- Tortury, szantaż, wymuszenia i inne pokrewne działania (patrz rysunek 3.3).

Rysunek 3.3. Rysunek z serwisu XKCD będący odpowiedzią na standard ISO 8601 (https://xkcd.com/538/)

W Linuksie do odzyskania haseł/kluczy możesz wykorzystać narzędzia takie jak John the Ripper, Hashcat i Bulk_Extractor.

Poniżej wyjaśnię, jak działa szyfrowanie, jak zidentyfikować użyty algorytm oraz jak wyodrębnić metadane z zaszyfrowanego woluminu lub katalogu. Wyjaśnię też deszyfrowanie w przypadku, gdy klucz jest znany.

Szyfrowanie całego dysku za pomocą LUKS

LUKS[15] to standardowy format do zaszyfrowanego przechowywania danych. Jego specyfikacja znajduje się pod adresem *https://gitlab.com/cryptsetup/cryptsetup/*, a referencyjną implementacją tego standardu jest pakiet `cryptsetup`. Więcej informacji na jego temat znajdziesz w sekcji `cryptsetup(8)` manuala. Jeśli Twoje komercyjne oprogramowanie kryminalistyczne nie obsługuje analizy i odszyfrowywania woluminów LUKS, możesz zbadać obraz kryminalistyczny na komputerze analitycznym z systemem Linux.

Woluminy LUKS mogą być tworzone na dyskach zawierających tablicę partycji oraz na nośnikach jej pozbawionych. W schemacie DOS[16] typem woluminu LUKS jest `0xE8`. W schemacie GPT woluminy LUKS mają GUID[17] równy `CA7D7CCB-63ED-4C53-861C-1742536059CC`. Jeśli są używane, te typy partycji mogą wskazywać na istnienie woluminu LUKS. Pamiętaj, że nie wszystkie narzędzia rozpoznają te typy partycji (nie robi tego np. `fdisk`), a partycje LUKS są czasami reprezentowane przez standardowe (ogólne) typy partycji systemu Linux.

[15] Przykłady użycia LUKS w tej książce wykorzystują LUKS2, czyli najnowszą wersję tego formatu.
[16] *https://www.win.tue.nl/~aeb/partitions/partition_types-1.html*.
[17] *https://en.wikipedia.org/wiki/GUID_Partition_Table*.

W trakcie rozruchu systemy Linux odczytują plik */etc/crypttab*, aby skonfigurować zaszyfrowane systemy plików. Warto go przeanalizować, ponieważ pokazuje on, co jest zaszyfrowane, skąd pochodzi hasło i jakie opcje zastosowano. Plik *crypttab* zawiera cztery pola:

- ***name*** — nazwa urządzenia blokowego, która pojawi się w */dev/mapper/*.
- ***device*** — UUID lub urządzenie reprezentujące zaszyfrowany wolumin.
- ***password*** — źródło hasła. Może to być plik klucza lub hasło wpisywane ręcznie (drugą opcję reprezentuje wartość *none* lub -).
- ***options*** — informacje o algorytmach kryptograficznych, konfiguracji i innych zarachowaniach.

Poniżej pokazano kilka przykładowych wierszy z pliku */etc/crypttab*, które reprezentują szyfrowanie głównego katalogu i partycji wymiany:

```
#   <name>    <device>    <password>    <options>
root-crypt UUID=2505567a-9e27-4efe-a4d5-15ad146c258b none luks,discard
swap-crypt /dev/sda7 /dev/urandom swap
```

W tym przypadku *swap-crypt* i *root-crypt* będą odszyfrowanymi urządzeniami znajdującymi się w */dev/mapper/*. Dostęp do głównego katalogu wymaga podania hasła (wartość *none*), a klucz do partycji wymiany jest generowany losowo. Plik *crypttab* może się również znajdować w *initramfs*. Niektórzy administratorzy chcą mieć możliwość restartu serwera bez konieczności wprowadzania hasła. W takich przypadkach ukrywają oni gdzieś w systemie plik klucza. Plik ten może się również znajdować w kopii zapasowej.

Wolumin LUKS można zidentyfikować za pomocą sześciobajtowej sygnatury i dwubajtowego ciągu reprezentującego wersje LUKS (1 lub 2):

```
4C55 4B53 BABE 0001 LUKS....
4C55 4B53 BABE 0002 LUKS....
```

Jeśli podejrzewasz, że w systemie znajduje się partycja LUKS, ale nie możesz jej znaleźć w tablicy partycji, spróbuj poszukać na dysku tej sygnatury. Prawidłowe dopasowanie powinno się znajdować na początku sektora.

Moduł LUKS w jądrze szyfruje dane w warstwie bloków „poniżej" systemu plików. Nagłówek zaszyfrowanej partycji zawiera opis użytych algorytmów, informacje o slotach, unikalny identyfikator (UUID), etykietę woluminu wybraną przez użytkownika oraz inne dane. Nagłówek woluminu LUKS możesz wyodrębnić za pomocą polecenia `cryptsetup luksDump`. Możesz je wywołać na podłączonym (za pomocą blokera zapisu) urządzeniu lub na surowym obrazie kryminalistycznym:

```
# cryptsetup luksDump /dev/sdb1
LUKS header information
Version:        2
Epoch:          5
Metadata area:  16384 [bytes]
Keyslots area:  16744448 [bytes]
UUID:           246143fb-a3ec-4f2e-b865-c3a3affab880
Label:          My secret docs
Subsystem:      (no subsystem)
Flags:          (no flags)

Data segments:
  0: crypt
        offset: 16777216 [bytes]
        length: (whole device)
        cipher: aes-xts-plain64
        sector: 512 [bytes]

Keyslots:
  1: luks2
        Key:        512 bits
        Priority:   normal
        Cipher:     aes-xts-plain64
        Cipher key: 512 bits
        PBKDF:      argon2i
        Time cost:  4
        Memory:     964454
        Threads:    4
        Salt:       8a 96 06 13 38 5b 61 80 c3 59 75 87 f7 31 43 87
                    54 dd 32 8c ea c0 b2 8b e5 bc 77 23 11 fb e9 34
        AF stripes: 4000
        AF hash:    sha256
        Area offset:290816 [bytes]
        Area length:258048 [bytes]
        Digest ID:  0
Tokens:
Digests:
  0: pbkdf2
        Hash:       sha256
        Iterations: 110890
        Salt:       74 a3 81 df d7 f0 f5 0d d9 c6 3d d8 98 5a 16 11
                    7c c2 ea cb 06 7f e9 b1 37 0b 66 24 3c 69 e1 ce
        Digest:     17 ad cb 13 16 f2 cd e5 d8 ea 49 d7 a4 89 bc e0
                    00 a0 60 e8 95 6b e1 e2 19 4b e7 07 24 f4 73 cb
```

Nagłówek LUKS nie zawiera żadnych znaczników czasu wskazujących na datę utworzenia lub ostatniego użycia. Interesująca może być etykieta woluminu (o ile użytkownik ją ustalił). Etykieta (*Label*) to pole tekstowe zdefiniowane przez użytkownika i może zawierać opis zaszyfrowanej zawartości. Śledczego mogą też

zainteresować sloty (*Keyslots*). Wolumen LUKS może mieć do ośmiu kluczy, co potencjalnie pozwala odszyfrować go za pomocą ośmiu różnych haseł.

Tworzenie kopii zapasowych nagłówka LUKS jest zalecane, w systemie mogą więc istnieć jakieś jego kopie. Jeśli podczas tworzenia kopii zapasowej stosowane były inne (być może znane Ci) hasła, to mogą one zapewniać dostęp do zaszyfrowanych danych LUKS. Narzędzie `cryptsetup` zawiera podkomendy `luksHeaderBackup` i `luksHeaderRestore`, które służą do tworzenia i przywracania kopii zapasowych nagłówka LUKS. Kopię zapasową nagłówka można też wykonać za pomocą `dd`, ponieważ jest to po prostu kopia ciągu surowych bajtów (w tym przykładzie jest to 16 777 216 bajtów lub 32 768 sektorów).

Aby można było odszyfrować wolumin LUKS na maszynie kryminalistycznej z systemem Linux, obraz śledczy musi być zamontowany w postaci urządzenia blokowego (`cryptsetup` nie potrafi odblokowywać zwykłych plików). Podkomenda `luksOpen` tworzy nowe urządzenie, które pozwala na dostęp do zawartości odszyfrowanego woluminu:

```
# cryptsetup luksOpen --readonly /dev/sdb1 evidence
Enter passphrase for /dev/sdb1:
# fsstat /dev/mapper/evidence
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: Ext4
Volume Name:
Volume ID: 6c7ed3581ee94d952d4d120dd29718d2

Last Written at: 2020-11-20 07:14:14 (CET)
Last Checked at: 2020-11-20 07:13:52 (CET)
...
```

Po wywołaniu tworzone jest nowe urządzenie blokowe */dev/mapper/evidence*, które zawiera odszyfrowaną zawartość woluminu LUKS. W tym przykładzie widoczny jest system plików *ext4*. Mimo że urządzenie powinno być chronione blokadą zapisu, dla pewności możesz dodatkowo zastosować opcję `--readonly`. Urządzenie można usunąć za pomocą podkomendy `luksClose` (`cryptsetup luksClose evidence`).

Narzędzie do łamania haseł John the Ripper umożliwia podjęcie próby odzyskania haseł do LUKS w wersji 1 (aby sprawdzić, czy pojawiła się obsługa wersji 2, zajrzyj do repozytorium projektu: *https://github.com/openwall/john/*). Niektóre systemy mogą nadal korzystać z LUKS w wersji 1.

Nowy `systemd-homed` standardowo wykorzystuje LUKS do szyfrowania katalogów domowych. W chwili pisania tego tekstu `systemd-homed` to nowo proponowane rozwiązanie, które nie jest powszechnie stosowane. Techniki przedstawione w tym punkcie powinny zadziałać z dowolnym zaszyfrowanym woluminem LUKS.

Szyfrowanie katalogów za pomocą eCryptfs

Podczas instalacji niektóre dystrybucje Linuksa oferują możliwość zaszyfrowania katalogu domowego lub podkatalogu użytkownika (zamiast szyfrowania pełnego dysku, tak jak w przypadku LUKS).

Do niedawna eCryptfs był najpopularniejszym systemem szyfrowania katalogów wykorzystującym warstwową budowę systemu plików. Inne systemy szyfrowania katalogów to EncFS i *cryptfs* (oparty na wbudowanej w *ext4* funkcji szyfrowania katalogów). W tym punkcie omówię eCryptfs. Przyszłość eCryptfs nie jest jasna. Niektóre dystrybucje uznały go za przestarzałe rozwiązanie, a twórcy Debiana podjęli decyzję o jego usunięciu z powodu niezgodności z `systemd`.

System eCryptfs składa się trzech głównych katalogów: zaszyfrowanego drzewa katalogów (często jest to ukryty katalog o nazwie *.Private/*), punktu montowania dla odszyfrowanego drzewa katalogów oraz ukrytego katalogu zawierającego hasło i różne pliki stanu (często nazywa się on *.ecryptfs/* i znajduje się w tym samym katalogu co *.Private/*).

W przypadku szyfrowania całych katalogów domowych niektóre dystrybucje Linuksa umieszczają katalogi *.Private/* i *.ecryptfs/* każdego użytkownika w osobnym katalogu */home/.ecryptfs/*. Odszyfrowane katalogi są następnie montowane w katalogach domowych użytkowników. W poniższym przykładzie z Linux Mint do użytkownika *Sam* należą trzy katalogi:

```
/home/.ecryptfs/sam/.ecryptfs/
/home/.ecryptfs/sam/.Private/
/home/sam/
```

Pierwszy katalog zawiera plik z hasłem użytkownika *Sam* i inne informacje. W drugim przechowywane są zaszyfrowane pliki i katalogi użytkownika *Sam*. Ostatni katalog to punkt montowania używany przez system eCryptfs. Punkt montowania zapewnia dostęp do odszyfrowanego katalogu domowego użytkownika.

W niektórych przypadkach użytkownik może chcieć zaszyfrować jedynie podkatalog znajdujący się w jego katalogu domowym. W takim przypadku najczęściej spotyka się następującą strukturę katalogów:

```
/home/sam/.ecryptfs/
/home/sam/.Private/
/home/sam/Private/
```

Tak jak poprzednio: katalog *.ecryptfs/* zawiera hasło i pliki pomocnicze, *.Private/* to ukryty katalog zawierający zaszyfrowane pliki, a *Private/* to punkt montowania, w którym znajdą się odszyfrowane pliki. W trakcie badania kryminalistycznego o użyciu eCryptfs świadczy znalezienie w systemie dowolnego katalogu *.ecryptfs*. Lokalizacja odszyfrowanego punktu montowania jest zapisana w pliku *Private.mnt*.

Aby ukryć informacje o typie pliku lub zawartości katalogów, ich nazwy również są szyfrowane. Poniżej pokazano przykład zaszyfrowanej nazwy pliku (*secrets.txt*):

```
ECRYPTFS_FNEK_ENCRYPTED.FWb.MkIpyP2LoUSd698zVj.LP4tIzB6lyLWDy1vKIhPz8WBMAYFCpelfHU--
```

W trakcie badania kryminalistycznego możesz również przeprowadzić wyszukiwanie plików z przedrostkiem `ECRYPTFS_FNEK_ENCRYPTED.*`, które ujawni, czy w systemie użyto eCryptfs.

Choć zawartość i nazwy plików są zaszyfrowane, istnieją pewne metadane, które mogą być przydatne w trakcie dochodzenia. Poniżej porównałem dane wyjściowe (informacje z *i*-węzła) dla zaszyfrowanego i odszyfrowanego pliku:

```
$ stat Private/secrets.txt
  File: Private/secrets.txt
  Size: 18         Blocks:  24         IOBlock: 4096  regularfile ❶
Device: 47h/71d  Inode: 33866440    Links: 1
Access: (0640/-rw-r-----)   Uid: ( 1000/    sam)    Gid: ( 1000/ sam) ❷
Access: 2020-11-21 10:14:56.092400513 +0100 ❸
Modify: 2020-11-21 09:14:45.430398866 +0100
Change: 2020-11-21 14:27:43.233570339 +0100
  Birth: -
...
$ stat .Private/ECRYPTFS_FNEK_ENCRYPTED.FWb.MkIpyP2LoUSd698zVj.
➥LP4tIzB6lyLWDy1vKIhPz8WBMAYFCpelfHU--
File:
.Private/ECRYPTFS_FNEK_ENCRYPTED.FWb.MkIpyP2LoUSd698zVj.LP4tIzB6lyLWDy1vKIhPz
➥8WBMAYFCpelfHU--
  Size: 12288         Blocks:24          IOBlock:4096   regularfile ❶
Device: 1bh/27d Inode: 33866440    Links: 1
Access: (0640/-rw-r-----)   Uid: ( 1000/    sam)    Gid: ( 1000/ sam) ❷
Access: 2020-11-21 10:14:56.092400513 +0100 ❸
Modify: 2020-11-21 09:14:45.430398866 +0100
Change: 2020-11-21 14:27:43.233570339 +0100
 Birth: 2020-11-21 09:14:45.430398866 +0100
```

Zaszyfrowane pliki mają te same znaczniki czasu (❸), prawa dostępu i właściciela (❷), co ich odszyfrowane odpowiedniki. Rozmiary plików (❶) są różne, a zaszyfrowane pliki zawsze mają rozmiar co najmniej 12 288 bajtów. Po zamontowaniu zaszyfrowane i odszyfrowane pliki mają ten sam numer *i*-węzła (nawet jeśli znajdują się w różnych systemach plików).

Odszyfrowane pliki są dostępne jedynie po zamontowaniu w działającym systemie. Aby uzyskać dostęp do odszyfrowanej zawartości (zakładając, że znasz hasło), możesz skopiować zaszyfrowany katalog do swojego systemu i go odszyfrować. Aby to zrobić, zainstaluj pakiet oprogramowania `ecryptfs-utils`, skopiuj trzy katalogi (*.ecryptfs/*, *.Private/* i *Private/*) i uruchom polecenie `ecryptfs-mount-➥private`. Na ekranie pojawi się prośba o podanie hasła, a katalog *Private/*

zostanie zamontowany. Do dopasowania do siebie zaszyfrowanych i odszyfrowanych plików możesz wykorzystać numer *i*-węzła (w tym celu możesz wykorzystać narzędzie `ecryptfs-find`).

Aby odmontować zaszyfrowane pliki (uczynić je niedostępnymi), uruchom polecenie `ecryptfs-umount-private`. Alternatywne lokalizacje i sposoby odszyfrowywanie opisano w sekcji `mount.ecryptfs_private(1)` manuala.

Z katalogiem eCryptfs są powiązane dwa hasła: **hasło montowania** (ang. *mount passphrase*) i **hasło kontenera** (ang. *wrapping passphrase*). Standardowo hasło montowania to losowy 32-znakowym ciąg wartości szesnastkowych, który powinno się zapisać na wszelki wypadek (przyda się, jeśli użytkownik zapomni hasła do kontenera). Hasło montowania jest przekazywane do jądra w celu zamontowania i odszyfrowania plików. Hasło kontenera chroni hasło montowania i jest ustalane przez użytkownika, który może je zmienić bez wpływu na zaszyfrowane pliki. Hasło kontenera jest często takie samo jak hasło wykorzystywane do logowania w systemie.

Znalezienie w trakcie badania kryminalistycznego hasła montowania może umożliwić dostęp do zaszyfrowanych plików. Jeśli uda Ci się znaleźć hasło montowania, to za pomocą polecenia `ecryptfs-wrap-passphrase` będziesz mógł ustalić nowe hasło do kontenera. Nowo ustawione hasło może być następnie użyte do zamontowania katalogu eCryptfs.

W ostateczności możesz spróbować złamać hasło eCryptfs za pomocą narzędzia John the Ripper. W poniższym przykładzie najpierw wyodrębniam informacje z pliku eCryptfs, który przechowuje informacje o haśle do kontenera oraz zapisuje je w formacie akceptowanym przez narzędzie John The Ripper. Następnie próbuję złamać hasło za pomocą polecenia `john`:

```
$ ecryptfs2john.py .ecryptfs/wrapped-passphrase > ecryptfs.john
$ john ecryptfs.john
Using default input encoding: UTF-8
Loaded 1 password hash (eCryptfs [SHA512 128/128 AVX 2x])
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
canada           (wrapped-passphrase)
1g 0:00:01:35 DONE 2/3 (2020-11-20 15:57) 0.01049g/s 128.9p/s 128.9c/s
128.9C/s 123456..maggie
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Po przeprowadzeniu wielu operacji i siłowej próbie złamania hasła za pomocą listy słów John the Ripper odkrywa, że hasłem jest *canada*.

Szyfrowanie katalogów za pomocą fscrypt (ext4 directory encryption)

Jądro Linuksa zapewnia możliwość szyfrowania plików i katalogów na poziomie systemu plików (w przeciwieństwie do szyfrowania LUKS, które działa na poziomie bloków) za pomocą biblioteki *fscrypt*. Pierwotnie była ona częścią *ext4*. Obecnie stanowi osobne narzędzie i obsługuje też inne systemy plików (na przykład F2FS). Opis jądrowego interfejsu API znajdziesz pod adresem *https://www.kernel.org/doc/html/latest/filesystems/fscrypt.html*. Do konfiguracji jądra i uruchomienia lub dezaktywacji szyfrowania niektórych katalogów możesz wykorzystać narzędzia z przestrzeni użytkownika, takie jak `fscrypt` lub `fscryptctl`.

Dowody użycia *fscrypt* można znaleźć w kilku miejscach. System plików *ext4* zawiera artefakty wskazujące, że użyto szyfrowania *fscrypt*:

```
$ dumpe2fs -h partimage.raw
...
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype
↪needs_recovery extent 64bit flex_bg encrypt sparse_super large_file huge_file
↪dir_nlink extra_isize metadata_csum
...
```

Zwróć uwagę na obecność słowa *encrypt* w danych wyjściowych z superbloku. Obsługa *fscrypt* zazwyczaj jest domyślnie wyłączona (głównie ze względu na kompatybilność wsteczną). Obecność słowa *encrypt* nie oznacza, że używane jest szyfrowanie *fscrypt*, ale wskazuje, że szyfrowanie zostało włączone przez użytkownika, co oznacza, że należy przeprowadzić dalsze badania.

Niektóre narzędzia *fscrypt* dostępne w przestrzeni użytkownika mogą zostawiać ślady w systemie. Na przykład *fscrypt* od Google (*https://github.com/google/fscrypt/*) tworzy plik konfiguracyjny */etc/fscrypt.conf* i ukryty katalog */.fscrypt/* w głównym katalogu systemu plików. Obecność tych plików wskazuje na użycie *fscrypt*. Innym wskaźnikiem potencjalnego użycia *fscrypt* jest obecność niemożliwych do skopiowania plików o długich i zagadkowych nazwach. Na poniższym listingu pokazano zawartość katalogu *fscrypt* kolejno w zablokowanym i odblokowanym stanie:

```
$ ls KEEPOUT/
GpJCNtGVcwD7bkNVer7dWV8aTlb8gt2PP3,pG23vDQtRTldW1zpS7D
OWmj3cUXuNmIMZN6VP+qiE8DgROZZAXwVynF5ftvSaBBmayI9dq3HA
$ ls KEEPOUT/
report.doc video.mpeg
```

W przeciwieństwie do danych zaszyfrowanych za pomocą eCryptfs plików zaszyfrowanych za pomocą *fscrypt* nie da się skopiować na maszynę kryminalistyczną, ponieważ system plików nie może uzyskać dostępu do danych bez znajomości klucza:

```
$ cp KEEPOUT/* /evidence/
cp: cannot open 'KEEPOUT/GpJCNtGVcwD7bkNVer7dWV8aTlb8gt2PP3,pG23vDQtRTldW1zpS7D'
for reading: Required key not available
cp: cannot open 'KEEPOUT/OWmj3cUXuNmIMZN6VP+qiE8DgROZZAXwVynF5ftvSaBBmayI9dq3HA'
for reading: Required key not available
```

Dostęp do odszyfrowanego katalogu jest możliwy tylko wtedy, gdy cały system plików jest dostępny na maszynie kryminalistycznej z odpowiednio skonfigurowanym szyfrowaniem w jądrze. Na maszynie muszą być również zainstalowane narzędzia przestrzeni użytkownika użyte do zaszyfrowania katalogu. Jeśli hasło jest znane, można uzyskać dostęp do zaszyfrowanego katalogu. W tym celu należy porównać zawartość pliku */etc/fscrypt.conf* na maszynie kryminalistycznej i dysku podejrzanego. W przypadku różnic konieczne może być zastąpienie pliku na maszynie kryminalistycznej plikiem z dysku podejrzanego (plik zawiera dane konfiguracyjne).

Poniżej pokazałem, jak wykorzystać narzędzie `fscrypt` do uzyskiwania dostępu do dowodów w zaszyfrowanym katalogu w systemu plików *ext4*:

```
# mount /dev/sdb /evidence/
# fscrypt unlock /evidence/KEEPOUT/
Enter custom passphrase for protector "sam":
"/evidence/KEEPOUT/" is now unlocked and ready for use.
```

W pierwszym wierszu montuję partycję *ext4* w */evidence/* (wciąż jest to normalny system plików; na tym poziomie nie jest to nic niezwykłego). W drugim wierszu za pomocą polecenia `fscrypt unlock` próbuję odszyfrować katalog. Na ekranie pojawia się prośba o podanie hasła. Informacje o kluczu są przechowywane w katalogu *.fscrypt/* w głównym katalogu dysku, ale do ich odszyfrowania wymagane jest hasło.

Metadane nie podlegają szyfrowaniu. Informacje o *i*-węźle (wyświetlone za pomocą `stat` lub `istat`) będą takie same, niezależnie od tego, czy katalog jest zablokowany, czy odblokowany. Sygnatury czasowe, prawa własności, prawa dostępu itd. są widoczne, nawet jeśli katalog jest zaszyfrowany (zablokowany).

Podsumowanie

W tym rozdziale opisałem analizę kryminalistyczną nośników pamięci. Dowiedziałeś się, jak analizować strukturę dysków i tablice partycji, RAID i LVM. Przedstawiłem też trzy najpopularniejsze systemy plików wykorzystywane w Linuksie. W rozdziale skupiłem się na ich analizie oraz odzyskiwaniu artefaktów kryminalistycznych. Niestety w niektórych obszarach kryminalistycznej analizy nośników pamięci brakuje narzędzi społecznościowych. Ponieważ jest to ciągle rozwijający się obszar badań, z czasem braki te zostaną uzupełnione.

Skorowidz

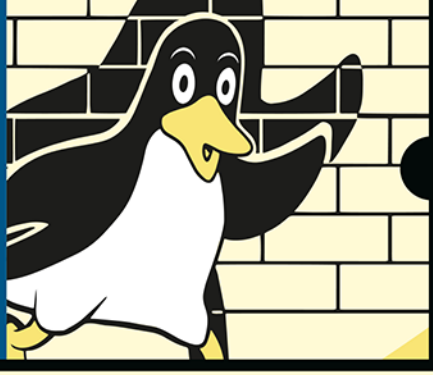
R

S

Ś


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

ANALIZA LINUKSA: ZACZNIJ PRZYGODĘ Z INFORMATYKĄ ŚLEDCZĄ!



Rozwój technologii służy również przestępcom. Wykrywanie śladów niewłaściwego użycia dotyczy maszyn, które zarówno posłużyły do przeprowadzenia ataków, jak i były ich przedmiotem. Obecnie dostępnych jest wiele opracowań poświęconych sposobom działania na miejscu zdarzenia i analizie działających systemów Linux za pomocą poleceń dostępnych po zalogowaniu się na pracującym urządzeniu. Równie ważną metodą pracy śledczej jest badanie obrazu dysku, tworzonego zgodnie z regułami kryminalistyki. Można też podłączyć badany dysk do maszyny badawczej — w bezpieczny sposób, za pośrednictwem kryminalistycznego blokera zapisu. I właśnie o tych technikach mowa w tej książce.

Dokładnie opisano w niej, jak lokalizować i interpretować dowody elektroniczne znajdujące się na komputerach stacjonarnych, serwerach i urządzeniach IoT pracujących pod kontrolą systemu Linux, a także jak odtwarzać ciąg zdarzeń, które nastąpiły po popełnieniu przestępstwa lub wystąpieniu incydentu związanego z bezpieczeństwem. Przedstawiono zasady analizy pamięci masowej, systemu plików i zainstalowanego oprogramowania. Wyjaśniono sposób badania dziennika systemd, dzienników jądra i jego systemu audytu, jak również dzienników demonów i aplikacji. Ponadto znajdziesz tu omówienie metod analizy konfiguracji sieciowej, w tym interfejsów, adresów, menedżerów sieci i artefaktów związanych z sieciami bezprzewodowymi, sieciami VPN czy zaporami.

Dzięki książce dowiesz się, jak:

- sprawdzać istotne ustawienia
- zrekonstruować proces uruchamiania Linuksa
- analizować tabele partycji, zarządzanie woluminami, systemy plików, układ katalogów, zainstalowane oprogramowanie i konfigurację sieci
- badać historię środowiska fizycznego, restartów i awarii systemu
- analizować sesje logowania użytkowników
- identyfikować ślady podłączonych urządzeń peryferyjnych

Bruce Nikkel jest profesorem Bern University of Applied Sciences w Szwajcarii, specjalizuje się w zagadnieniach informatyki śledczej i cyberprzestępczości. Od 1997 roku pracuje również w działach ryzyka i bezpieczeństwa jednej z globalnych instytucji finansowych. Jest redaktorem czasopisma „Forensic Science International: Digital Investigation”.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-283-9404-9

9 788328 394049

Cena: 129,00 zł

no starch press