ZFS Raiders of the Lost File

Nikolaos Tsapakis, leHACK 2023

Intro

- a hot summer day back in 2019
- an old hard drive
- a file that I forgot to backup
- a format process that got interrupted in panic
- a poem that was brought back to life
- this is a forensics story

The Hard Disk

- FreeBSD image with ZFS filesystem
- LZ4 compression on
- compresses files on the fly
- tried many free recovery tools but in vain
- recover the file by directly accessing the disk ?

Creating Hard Disk Image

- ddrescue /dev/ada0 /media/ada0.backup
- 25 gb of size
- can now start working on the image

The File

- I remember it was an .rtf file
- possibly < 64 kb in size
- ZFS may allocate multiple data blocks for bigger files
- notice smallest data block as a file grows is close to 65 kb
- 65 kb of compressed space for storing continuous file data
- only a single block for our file if we are lucky

The Signature

- compress an .rtf sample file using LZ4
- check header after compression
- create signature based on compressed header
- run signature against image
- verify that signature triggers on .rtf files

The Signature

```
first bytes of rtf sample file
# hexdump -C -n 32 test.rtf
00000000 7b 5c 72 74 66 31 5c 61 6e 73 69 5c 61 6e 73 69 {\rtf1\ansi\ansi
00000010 63 70 67 31 32 35 32 5c 64 65 66 66 30 5c 6e 6f cpg1252\deff0\no

compress the file
# lz4 test.rtf
Compressed filename will be : test.rtf.lz4
Compressed 53644 bytes into 6802 bytes ==> 12.68%

first bytes of compressed rtf sample file
# hexdump -C -n 32 test.rtf.lz4
00000000 04 22 4d 18 64 40 a7 7f 1a 00 00 b1 7b 5c 72 74 ."M.d@......{\rt
00000010 66 31 5c 61 6e 73 69 05 00 f0 75 63 70 67 31 32 f1\ansi...ucpg12
```

The Signature

```
rule detect_rtf
{
strings:
$rtf_header = { 7B 5C 72 74 66 31 5C }
condition:
$rtf_header
}
```

The Hunt

```
yara64.exe -s detect_rtf.yara ada0.backup
```

```
Results
0x1456b6005:$rtf_header: 7B 5C 72 74 66 31 5C
0x1666b1005:$rtf_header: 7B 5C 72 74 66 31 5C
0x1666c3005:$rtf_header: 7B 5C 72 74 66 31 5C
0x18991a28e:$rtf_header: 7B 5C 72 74 66 31 5C
0x2c20348fa:$rtf_header: 7B 5C 72 74 66 31 5C
0x5ca004115:$rtf_header: 7B 5C 72 74 66 31 5C
```

File Data Block

```
1666c2ff2 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1666c3002 1a 85 b1 7b 5c 72 74 66 31 5c 61 6e 73 69 05 00 ...{\rtf1\ansi..
1666c3012 f0 75 63 70 67 31 32 35 32 5c 64 65 66 66 30 5c .ucpg1252\deff0\
[...]
1666c4a72 69 6e 93 00 22 20 2a 3b 00 4f 7d 0d 0a 00 01 00 in.." *;.O}.....
1666c4a82 5c 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 \P..............
1666c4a92 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

🟥 XX = missing header

🟦 XX = size of compressed data

🟨 XX = signature (yara) part of compressed data

🟧 XX = compressed data

🟩 XX = trailing nulls

Header

LZ4 header is missing before the file data block so let’s construct minimal header using RTF specification.

```
LZ4_Header
{
DWORD MagicNb       Magic Number
BYTE FLG            various properties like version, block checksum etc.
BYTE BD             block maximum size
BYTE HC             checksum, can be calculated using xxhash on FLG, BD
}
```

Header.(dword)MagicNb

MagicNb = 0x04 0x22 0x4D 0x18

Header.(byte)FLG

FLG = 0x40 = 01 0 0 0 0 0 0

01 : version

0 : each block depends on previous ones for decoding

0 : no Block checksum in data blocks

0 : uncompressed size of data not included within the header

0 : no content checksum will be appended after the EoS mark

0 : reserved and set to 0

0 : no DictID

Header.(byte)BD

BD = 0x40 = 0 100 0000

100 = 4 = 64 KB of block maximum size

bits 7 and last 4 bits are reserved thus set to 0

Header.(byte)HC

The following python code will generate the HC descriptor checksum :

```
import xxhash
FLG = 0x40
BD = 0x40
payload = chr(FLG) + chr(BD)
out = xxhash.xxh32(payload, seed=0).intdigest()
out = hex((out>>8) & 0xFF)
print out
```

HC = 0xC0

Header + File Data Block

[0x04 0x22 0x4D 0x18]	MagicNb
[0x40]	FLG
[0x40]	BD
[0xC0]	HC
[0x85 0x1a 0x00 0x00]	compressed data size
[0xb1 0x7b 0x5c 0x72 0x74 ...]	compressed data
[0x00 ...]	null(s) trailer

The Automation

- python script
- scan the image for files using signature
- if found then prepend minimal LZ4 header
- try to decompress (header + file data block)
- if success then write file out

References

- Original paper and scripts :
- https://github.com/nitsa/zfs

Thank you !