



ethical.blue Magazine



David Farbaniec

Document Metadata
Program Debug Information
File Artifacts and Anomalies
Clearing File Metadata

Oxide54

All materials presented here are protected by international copyright law. Unauthorized copying and distribution of the published materials is strictly prohibited. All materials published here are for informational purposes and are intended only for educational use. The author of this magazine is not liable for any illegal use or damage caused by these materials. The fonts Agency FB, Archivo, Open Sans, Papyrus, and Tangerine are used in accordance with the licenses provided by their respective distributors. The author of this magazine does not act on behalf of the companies whose technologies or products are described — except where explicitly noted. The products and solutions described in the text are randomly selected for illustrative purposes. The author did not receive any money for describing these products or solutions — except where explicitly noted. All trademarks and registered names are used for informational purposes only and belong to their respective owners. This magazine does not endorse any products, tools, or practices that could be used for unlawful purposes.

Index

23:59:59.9999999 UTC, December 31, 9999	4
Disable Geotags in Photos on Android	6
Redirect Unwelcome Visitors to Null (GPS Coordinates: 0° 0' 0" N, 0° 0' 0" E) . .	7
Document Metadata	8
Program Debug Information	9
Extract Embedded Text Strings	10
Obfuscated String Solver	12
Embedded Firmware Artifacts	14
Windows Shortcut File (*.LNK)	17

Bibliography	20
---------------------	-----------

23:59:59.9999999 UTC, December 31, 9999

Metadata is information about data. It has a wide definition. For example, a book in a library typically has metadata, including the author, title, and publication year.

It is similar to a file on a computer. Any file has attributes like file type, size, and timestamps. This information is related to a file system and can be obtained in various ways. For example, one can use the `Get-Item` command-let in PowerShell (Scriptum 1).

```
Get-Item "C:\de54\anomaly.exe" |
  Select-Object Name, CreationTime,
  LastWriteTime, LastAccessTime
  | Format-List
```

Scriptum 1. Get file creation, modification, and last access times (PowerShell)

Example file attributes can be as presented below (Scriptum 2).

```
Name: anomaly.exe
CreationTime: 15.10.2012 02:00:00
LastWriteTime: 15.10.2012 02:00:00
LastAccessTime: 17.03.2026 17:55:21
```

Scriptum 2. Example timestamps from `Get-Item` in PowerShell

Easy ways for timestamp manipulation can lead to anomalous values. Example from Scriptum 3 contains PowerShell code that sets the file creation time to a future value.

```
Add-Type -AssemblyName System.IO;
[System.IO.File]::SetCreationTime
("C:\de54\anomaly.exe",
[DateTime]::new(3050, 10, 15,
3, 3, 0));
```

Scriptum 3. Set file creation time in PowerShell using .NET methods

Now, file creation time can be read using PowerShell and .NET methods as in Scriptum 4.

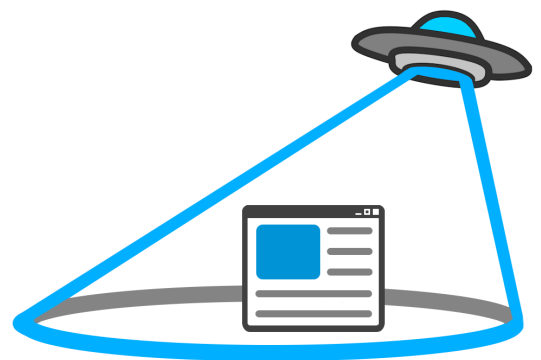
```
(Get-Item "C:\de54\anomaly.exe").
CreationTime.ToString(
"yyyy-MM-dd HH:mm:ss");
```

Scriptum 4. Read file creation time in PowerShell using .NET methods

The timestamp value will be printed to the console. See below.

```
3050-10-15 03:03:00
```

The limit is `DateTime.MaxValue`. That is 23:59:59.9999999 UTC, December 31, 9999.



LastAccessTime : 31.12.9999 23:59:59

PS C:\>

```

blue
Add-Type -AssemblyName System.IO; [System.IO.File]::SetLastAccessTime(
"C:\de54\anomaly.exe", [DateTime]::MaxValue);
Get-Item "C:\de54\anomaly.exe" | Select-Object Name, CreationTime,
LastWriteTime, LastAccessTime | Format-List

Name           : anomaly.exe
CreationTime    : 31.12.9999 23:59:59
LastWriteTime   : 31.12.9999 23:59:59
LastAccessTime : 31.12.9999 23:59:59

```

```

зона://ethical.blue
PS C:\> Add-Type -AssemblyName System.IO; [System.IO.File]::SetLastAccessTime(
"C:\de54\anomaly.exe", [DateTime]::MaxValue);
PS C:\> Get-Item "C:\de54\anomaly.exe" | Select-Object Name, CreationTime,
LastWriteTime, LastAccessTime | Format-List

Name           : anomaly.exe
CreationTime    : 31.12.9999 23:59:59
LastWriteTime   : 31.12.9999 23:59:59
LastAccessTime : 31.12.9999 23:59:59

PS C:\>

```

```

зона://ethical.blue
PS C:\> Add-Type -AssemblyName System.IO; [System.IO.File]::SetLastAccessTime(
"C:\de54\anomaly.exe", [DateTime]::MaxValue);
PS C:\> Get-Item "C:\de54\anomaly.exe" | Select-Object Name, CreationTime,
LastWriteTime, LastAccessTime | Format-List

Name           : anomaly.exe
CreationTime    : 31.12.9999 23:59:59
LastWriteTime   : 31.12.9999 23:59:59
LastAccessTime : 31.12.9999 23:59:59

PS C:\>

```

```

зона://ethical.blue
PS C:\> Add-Type -AssemblyName System.IO; [System.IO.File]::SetLastAccessTime(
"C:\de54\anomaly.exe", [DateTime]::MaxValue);
PS C:\> Get-Item "C:\de54\anomaly.exe" | Select-Object Name, CreationTime,
LastWriteTime, LastAccessTime | Format-List

Name           : anomaly.exe
CreationTime    : 31.12.9999 23:59:59
LastWriteTime   : 31.12.9999 23:59:59
LastAccessTime : 31.12.9999 23:59:59

PS C:\>

```

```

зона://ethical.blue
PS C:\> Add-Type -AssemblyName System.IO; [System.IO.File]::SetLastAccessTime(
"C:\de54\anomaly.exe", [DateTime]::MaxValue);
PS C:\> Get-Item "C:\de54\anomaly.exe" | Select-Object Name, CreationTime,
LastWriteTime, LastAccessTime | Format-List

Name           : anomaly.exe
CreationTime    : 31.12.9999 23:59:59
LastWriteTime   : 31.12.9999 23:59:59
LastAccessTime : 31.12.9999 23:59:59

PS C:\>

```

```

Add-Type -AssemblyName System.IO; [System.IO.File]::SetLastAccessTime(
[DateTime]::MaxValue);
anomaly.exe" | Select-Object Name, CreationTime,
LastWriteTime, LastAccessTime | Format-List

Name           : anomaly.exe
CreationTime    : 31.12.9999 23:59:59
LastWriteTime   : 31.12.9999 23:59:59
LastAccessTime : 31.12.9999 23:59:59

```

```

зона://ethical.blue
PS C:\> Add-Type -AssemblyName System.IO; [System.IO.File]::SetLastAccessTime(
"C:\de54\anomaly.exe", [DateTime]::MaxValue);
PS C:\> Get-Item "C:\de54\anomaly.exe" | Select-Object Name, CreationTime,
LastWriteTime, LastAccessTime | Format-List

Name           : anomaly.exe
CreationTime    : 31.12.9999 23:59:59
LastWriteTime   : 31.12.9999 23:59:59
LastAccessTime : 31.12.9999 23:59:59

```

Disable Geotags in Photos on Android

EXIF (Exchangeable Image File Format) is a metadata standard [1] supported by almost all digital camera manufacturers. If a digital camera has a built-in GPS module and the feature is enabled, photos can be geotagged.

Simply put, the EXIF metadata may contain GPS coordinates of the location where the photo was taken.

Geotags can be disabled by denying the Camera app access to the device's location, as shown in Figure 1.

EXIF metadata can be viewed in various ways. However, uploading a picture to a webpage may sometimes strip the metadata.

Detailed information on EXIF metadata can be obtained using ExifTool created by Phil Harvey, available at exiftool.org.

ExifTool.exe is a console program. An example command that clears EXIF metadata can be:
`exiftool.exe -all="" "C:\img*.jpg"`

Note that *.jpg is a wildcard expression that matches all files with the .jpg extension in the provided folder.

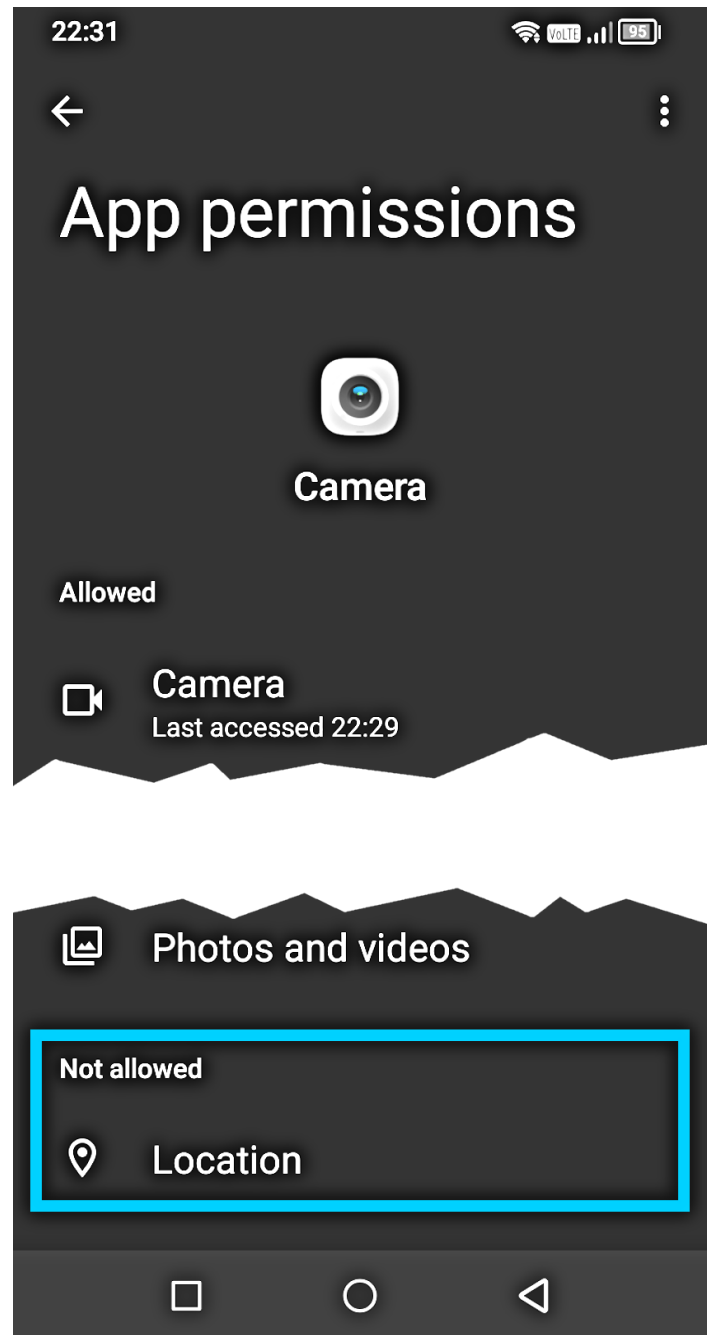


Figure 1. Disable adding geotags in photos on an Android device

Redirect Unwelcome Visitors to Null



exiftool_files



exiftool.exe



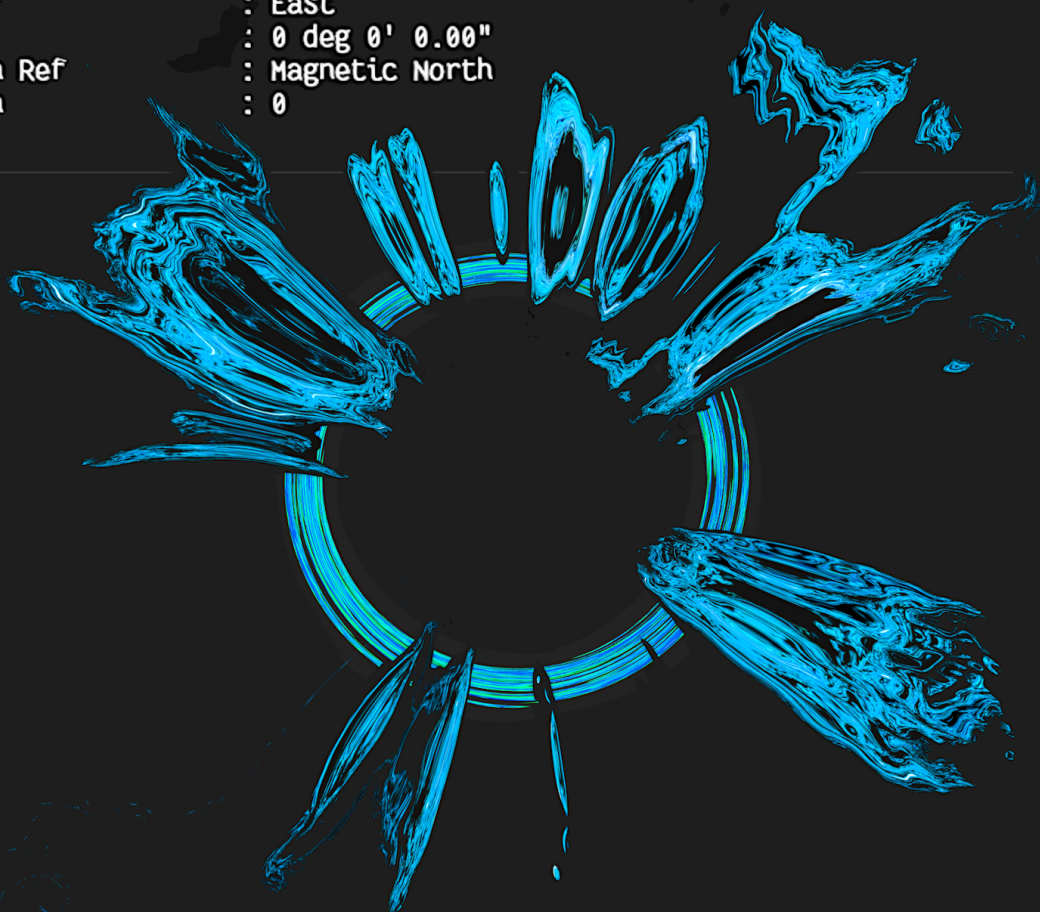
de54.jpg

GPS Coordinates:

0° 0' 0" N,

0° 0' 0" E

```
зона://ethical.blue x + v - □ ×  
PS C:\de54> exiftool.exe -a -gps:all de54.jpg  
GPS Latitude Ref : North  
GPS Latitude : 40 deg 42' 46.08"  
GPS Longitude Ref : West  
GPS Longitude : 74 deg 0' 21.60"  
GPS Img Direction Ref : Magnetic North  
GPS Img Direction : 0  
PS C:\de54> exiftool -gpsposition="0.0000, 0.0000" de54.jpg  
1 image files updated  
PS C:\de54> exiftool.exe -a -gps:all de54.jpg  
GPS Latitude Ref : North  
GPS Latitude : 0 deg 0' 0.00"  
GPS Longitude Ref : East  
GPS Longitude : 0 deg 0' 0.00"  
GPS Img Direction Ref : Magnetic North  
GPS Img Direction : 0  
PS C:\de54> █
```



Document Metadata

Embedded objects in complex formats may still contain metadata even after the main file cleanup (Figure 2).

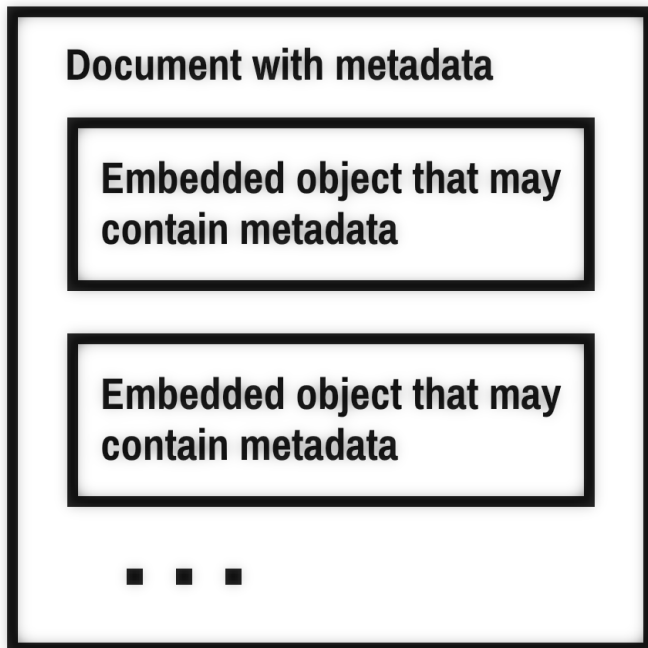


Figure 2. Embedded objects with metadata

Extracting metadata from .docx files is quite simple. See Figure 3. Embedded metadata may need detailed analysis to extract, but it is still possible.

Verify documents and embedded objects before publication. Check document details, remove unnecessary metadata, and export to PDF if possible.

```

zona://ethical.blue
PS C:\de54> exiftool.exe Document.docx
ExifTool Version      : 13.32
File Name             : Document.docx
Directory            : .
File Size             : 41 kB
Zone Identifier       : Exists
File Modification     : 2026:03:09 12:47:54+00
File Access Date     : 2026:03:24 10:58:10+00
File Creation Date   : 2026:03:24 10:58:10+00
File Permissions     : -rw-rw-rw-
File Type             : DOCX
File Type Extension  : docx
MIME Type            : application/vnd.openxmlformats-officedocument.wordprocessingml.document
Zip Required Version : 20
Zip Bit Flag         : 0x0006
Zip Compression      : Deflated
Zip Modify Date      : 1980:01:01 00:00:00
Zip CRC              : 0xa4b52315
Zip Compressed       : 478
Zip Uncompressed    : 3056
Zip File Name        : [Content_Types].xml
Title                :
Subject              :
Creator              : Dawid Farbaniec
Keywords             :
Last Modified Date   :
Revision Number      : 1
Create Date          : 2026:03:09 11:46:00Z
Modify Date          : 2026:03:09 11:46:00Z
Template             : Normal.dotm
Total Edit Time      : 0
Pages                : 1
Words                : 155
Characters           : 936
Application          : Microsoft Office Word
Doc Security         : None
Lines                : 7
Page Count           : 1
  
```

Figure 3. Document.docx file metadata

Program Database Path Embedded in Microsoft Visual C++ Executable



App.exe

```
зона://ethical.blue
PS C:\de54> Format-Hex App.exe -Offset 0x9734 -Count 0x2A
Label: C:\de54\App.exe
Offset Bytes Ascii
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----
00000000000009734 43 3A 5C 64 65 35 34 5C 65 74 68 69 63 61 6C 2E C:\de54\ethical.
00000000000009744 62 6C 75 65 5C 41 70 70 5C 78 36 34 5C 44 65 62 bLue\App\x64\Deb
00000000000009754 75 67 5C 41 70 70 2E 70 64 62 ug\App.pdb
PS C:\de54> █
```

*C:\de54\ethical.
blue\App\x64\Deb
ug\App.pdb*

Extract Embedded Text Strings

Executable files may contain plaintext strings as well as obfuscated ones.

Let's create a simple ELF file on Kali Linux.

The first command creates a C++ source file `main.cpp` with example code.

```
echo -e "#include <iostream> \n int
main() { std::cout << \"ethical.
blue\"; return 0; }" > main.cpp
```

The second command builds `main.cpp` to an executable named `ethical.elf`.

```
g++ -o ./ethical.elf ./main.cpp
```

The created file can be executed by typing file name `./ethical.elf` and will print:

```
ethical.blue
```

The `strings` command looks for printable strings in a file (Figure 4).

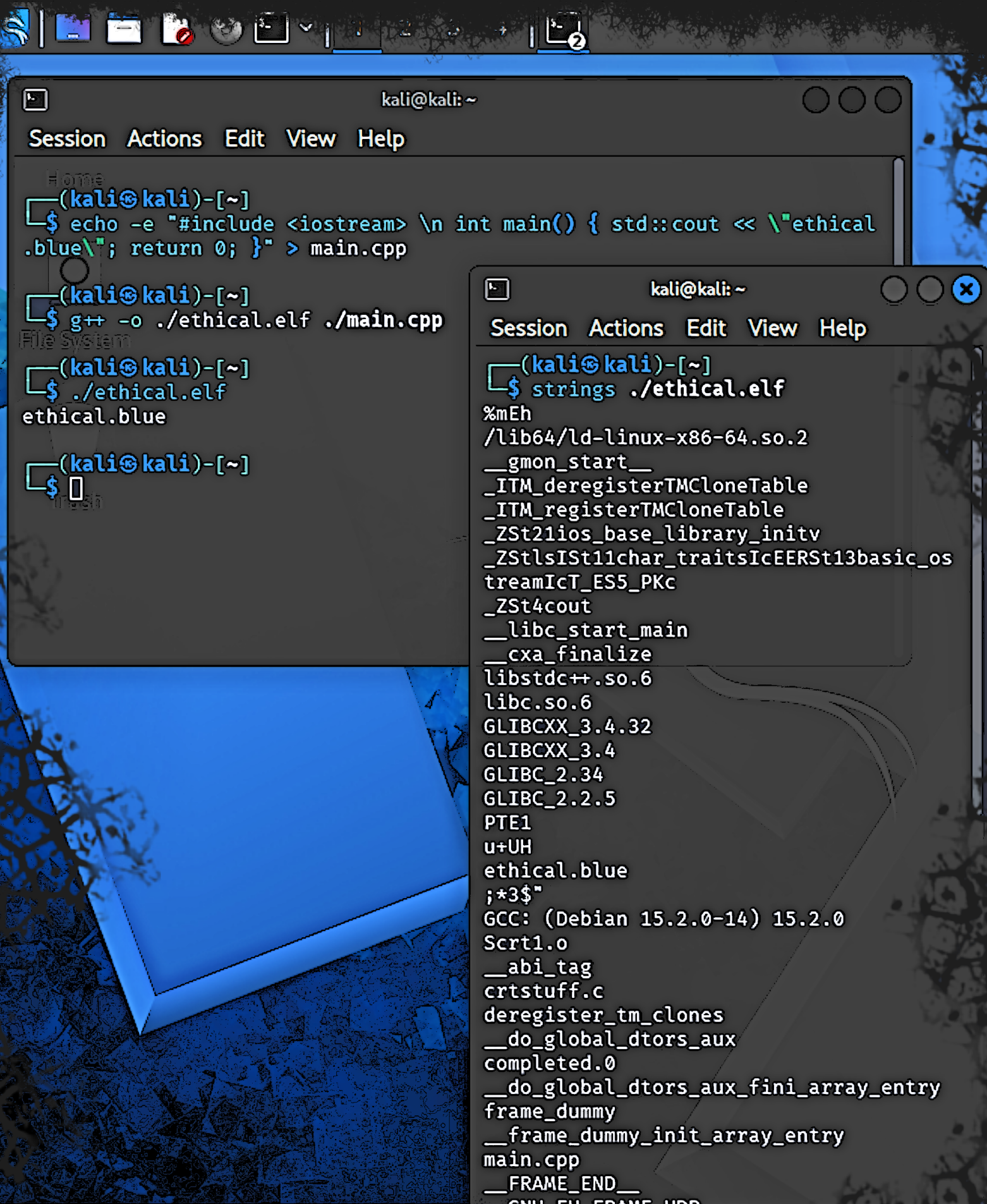
Here, in 3OHA laboratory, we use the `strings` command in the reconnaissance phase. It should be understood that the anomaly contains the specified string and nothing more.

Deeper analysis is needed to verify that embedded strings are not intended to mislead.

```
(kali@kali)-[~]
└─$ strings ./ethical.elf
%mEh
/lib64/ld-linux-x86-64.so.2
__gmon_start__
_ITM_deregisterTMCloneTable
_ITM_registerTMCloneTable
_ZSt21ios_base_library_initv
_ZStlsISt11char_traitsIcEERS
_ZSt4cout
__libc_start_main
__cxa_finalize
libstdc++.so.6
libc.so.6
GLIBCXX_3.4.32
GLIBCXX_3.4
GLIBC_2.34
GLIBC_2.2.5
PTE1
u+UH
ethical.blue
;*3$"
GCC: (Debian 15.2.0-14) 15.2
Scrt1.o
__abi_tag
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.0
__do_global_dtors_aux_fini_a
```

Figure 4. Look for printable strings in a file

Extract Embedded Text Strings



```
kali@kali: ~  
Session Actions Edit View Help  
# 0:00:00  
(kali@kali)-[~]  
└─$ echo -e "#include <iostream> \n int main() { std::cout << \"ethical  
.blue\"; return 0; }" > main.cpp  
(kali@kali)-[~]  
└─$ g++ -o ./ethical.elf ./main.cpp  
File System  
(kali@kali)-[~]  
└─$ ./ethical.elf  
ethical.blue  
(kali@kali)-[~]  
└─$ █  
kali@kali: ~  
Session Actions Edit View Help  
(kali@kali)-[~]  
└─$ strings ./ethical.elf  
%mEh  
/lib64/ld-linux-x86-64.so.2  
__gmon_start_  
_ITM_deregisterTMCloneTable  
_ITM_registerTMCloneTable  
_ZSt21ios_base_library_initv  
_ZSt1sISt11char_traitsIcEERSt13basic_os  
treamIcT_ES5_PKc  
_ZSt4cout  
__libc_start_main  
__cxa_finalize  
libstdc++.so.6  
libc.so.6  
GLIBCXX_3.4.32  
GLIBCXX_3.4  
GLIBC_2.34  
GLIBC_2.2.5  
PTE1  
u+UH  
ethical.blue  
;*3$"  
GCC: (Debian 15.2.0-14) 15.2.0  
Scrt1.o  
__abi_tag  
crtstuff.c  
deregister_tm_clones  
__do_global_dtors_aux  
completed.0  
__do_global_dtors_aux_fini_array_entry  
frame_dummy  
__frame_dummy_init_array_entry  
main.cpp  
__FRAME_END__  
GNU C++ FRAME UPD
```

Obfuscated String Solver

Anomalies found in 3OHA may contain obfuscated strings. Embedded text strings that are not plaintext may not catch the eye during quick reconnaissance.

FLARE Obfuscated String Solver [2] is a tool that can extract:

- static ASCII and UTF-16LE strings,
- strings constructed on the stack at program run-time,
- a special form of stack strings, decoded on the stack (tight strings),
- strings decoded in a function.

This solution can automate manual reverse engineering of the string decryption routine found in the anomaly.

The FLOSS tool can be used just like strings.exe to enhance basic static analysis.

For example:

```
.\floss.exe Anomaly.de54.Win64.exe
```

Use `.\floss.exe -h` to display help on available program parameters.



3OHA://fe80::ae54:3ccc:1c43:7a33
Connecting to laboratory... Please wait...

Obfuscated String Solver

зона://ethical.blue X + v - □ X

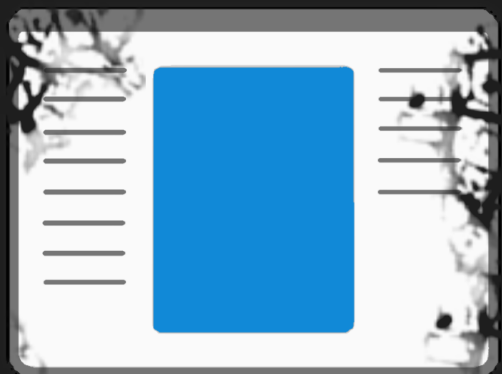
```
PS C:\de54> .\floss.exe Sample.MSVC++.Win64.A.exe
```

зона://ethical.blue X + v - □ X

```
zhchs  
zhcht  
zhcn  
zhhk  
zhmo  
zhsg  
zhtw  
zuza  
CONOUT$  
ethical.blue Magazine  
Sample.MSVC++.Win64.A (Microsoft Visual C++)
```



floss.exe



**Sample.MSVC++.
Win64.A.exe**

Embedded Firmware Artifacts

Firmware is a binary blob embedded in the device hardware. It can be stored in flash memory or other types of storage and often contains machine code and data that are crucial for the device to operate.

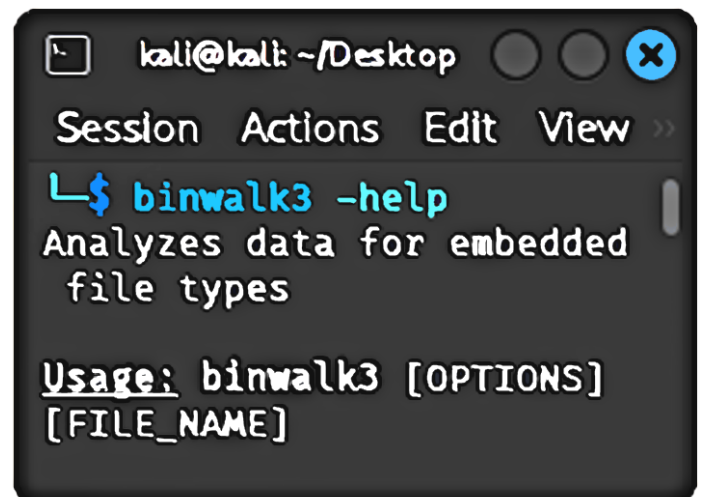
In many embedded systems, the firmware is packed, encrypted, and obfuscated. It may contain bootloaders, operating systems, and other components merged into a unified binary image. Acquiring firmware for analysis can be as straightforward as downloading a binary image from the manufacturer's website or as challenging as performing invasive hardware extraction.

The most common methods are:

- Download a firmware image from the manufacturer's website.
- Intercept network traffic during a device firmware update.
- Extract the firmware through exposed hardware debug interfaces.
- Identify the chip, desolder it, place it in a compatible reader, and dump the contents.

Now, let's skip to the moment when the firmware has already been dumped into a file.

Dumped firmware is typically a continuous byte blob. Merged components can be identified using binwalk, a firmware analysis tool. See Figure 5.



```
kali@kali: ~/Desktop
Session Actions Edit View >>
└─$ binwalk3 -help
Analyzes data for embedded
file types

Usage: binwalk3 [OPTIONS]
[FILE_NAME]
```

Figure 5. binwalk (Firmware Analysis Tool)

Use `binwalk3 -eM cctv-firmware` to automatically extract known file types and recursively scan extracted files (Figure 6).



```
kali@kali: ~/Desktop
Session Actions Edit View >>
└─(kali@kali)-[~/Desktop]
└─$ binwalk3 -eM cctv-firmware
```

The image shows a terminal window with the command `binwalk3 -eM cctv-firmware` executed. Below the terminal, there are two icons representing extracted files: a file icon labeled `cctv-firmw...` and a folder icon labeled `extractions`.

Figure 6. Extract files with binwalk

Dissected elements are located in the `extractions` folder and its subfolders. To navigate to a specific folder, use the `cd` command. For example:

```
cd ./cctv-firmware.extracted
```

The `ls` command is used to list all files and directories in the current working directory or a specified path. The folder names correspond to hexadecimal offsets.

```
400      30400    C00900    C2DB94
C45118  C5C7D4    C74508    C8BEE4
CA3588  ...
```

Use `binwalk --disasm` option to identify the CPU architecture of a file using the capstone disassembler. For example:

```
binwalk --disasm ./dump.bin
```

The result contains offset and description:

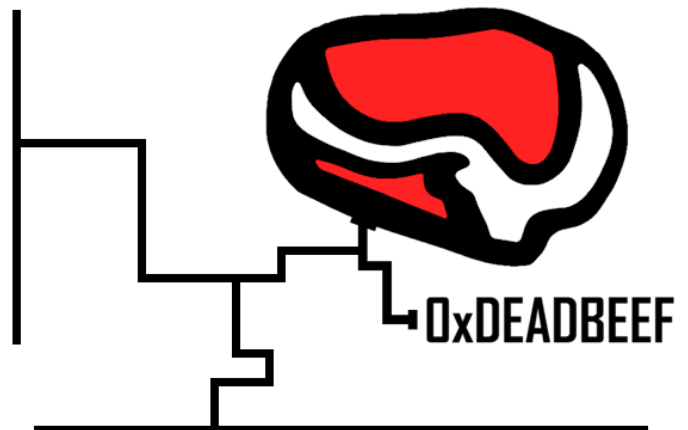
```
0xA48 ARM executable code, 32-bit,
little endian, at least 1250
valid instructions
```

Another option is `binwalk --signature`. It scans the target file(s) for common file signatures. For example:

```
binwalk --signature ./dump.bin
```

The result contains offset and description:

```
0x0 Linux kernel ARM boot
executable zImage (little-endian)
```



When working with firmware, it is often necessary to extract a specific fragment from a binary file. This can be achieved using the `dd` command in a terminal. For example:

```
dd if=decompressed.bin of=dump.bin
    bs=1 skip=4718712 count=14038
```

The command above extracts 14,038 bytes starting at offset 4,718,712 from `decompressed.bin` and writes them to `dump.bin`.

Once the binary fragments are extracted, the `strings` and `grep` commands can be used to search for specific text patterns. The command below uses the `-i` flag to search for the string "arm" in `dump.bin`, ignoring case.

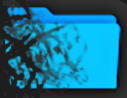
```
strings dump.bin | grep -i "arm"
```

After the reconnaissance phase, the dissected firmware components are ready for further analysis in reverse engineering tools, such as debuggers, disassemblers, and fuzzers.

Firmware Extraction

cctv-firmware.extracted

COABA8



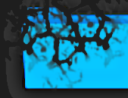
COABA8



C2DB94



C5C7D4



C8BEE4



C50CA0

```
kali@kali: ~/Desktop/extractions/cctv-firmware.extracted/30400
Session Actions Edit View Help

(kali@kali)-[~/Desktop/extractions/cctv-firmware.extracted/30400]
└─$ binwalk --disasm dump.bin

DECIMAL      HEXADECIMAL     DESCRIPTION
-----
2632      0xA48          ARM executable code, 32-bit, little endian,
at least 1250 valid instructions
```

```
kali@kali: ~/Desktop/extractions/cctv-firmware.extracted/30400/Linux-4.19.91.bin.extracted/60AE
Session Actions Edit View Help

(kali@kali)-[~/Desktop/extractions/cctv-firmware.extracted/30400/Linux-4.19.91.bin.extracted/60AE]
└─$ dd if=decompressed.bin of=dump.bin bs=1 skip=4718712 count=14038
14038+0 records in
14038+0 records out
14038 bytes (14 kB, 14 KiB) copied, 0.0327094 s, 429 kB/s

(kali@kali)-[~/Desktop/extractions/cctv-firmware.extracted/30400/Linux-4.19.91.bin.extracted/60AE]
└─$ binwalk --signature dump.bin

DECIMAL      HEXADECIMAL     DESCRIPTION
-----
0            0x0             YAFFS filesystem root entry, little endian, type file,
```

```
kali@kali: ~/Desktop/extractions/cctv-firmware.extracted/400/uimage_data.bin.extracted/0
Session Actions Edit View Help

(kali@kali)-[~/Desktop/extractions/cctv-firmware.extracted/400/uimage_data.bin.extracted/0]
└─$ strings decompressed.bin | grep -i "arm"
arch=arm
cpu=armv7
arm-linux-gcc.br_real (Buildroot 2020.02.9-10-g744f210) 8.4.0
arm-trusted-firmware
ARM Trusted Firmware
arm64
ARM CA9 global timer init successfully
ARM CA9 global timer had already been initiated
```

Windows Shortcut File (*.LNK)

A Microsoft Windows shortcut file has a .LNK extension, which is hidden by default. The .LNK file contains metadata that can be extracted.

When a shortcut file is opened by a program, two different things can happen. The application can resolve the .LNK file and open the target, or it may not resolve it and simply open the shortcut.

Unintended shortcut resolution can be avoided by changing the .LNK file extension. For example, with a simple PowerShell command:

```
Rename-Item -Path C:\m.LNK -NewName
C:\m.LNK.anomaly
```

The Shell Link (.LNK) binary file format in pseudo-C syntax can be:

```
struct WindowsShortcut {
    struct ShellLinkHeader sHeader;
    struct LinkTargetIDList sTarget;
    struct LinkInfo sInfo;
    struct StringData sStringData;
    struct ExtraData sExtraData;
};
```

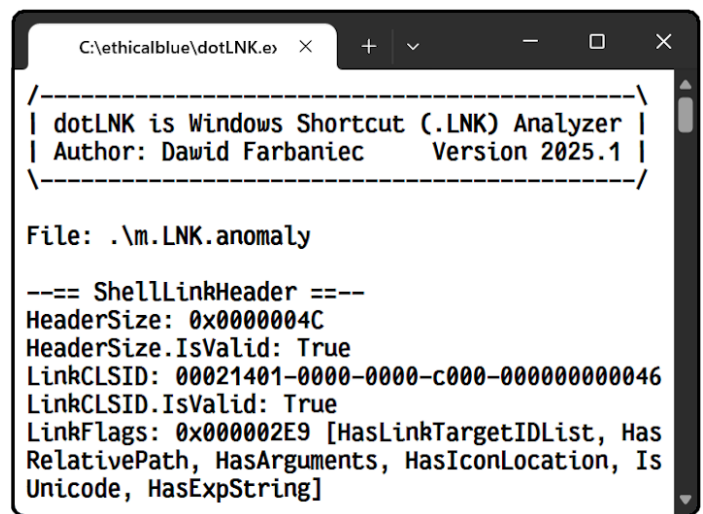
A detailed description of .LNK files can be found in the document [MS-SHLLINK] Shell Link (.LNK) Binary File Format.

The dotLNK tool, including C# source code, can be downloaded from:

```
//ethical.blue/n/dotLNK.zip
```

It is a simple console program that takes only one parameter — the .LNK file path.

See Figure 7.



```
C:\ethicalblue\dotLNK.e x + - □ ×
/-----\
| dotLNK is Windows Shortcut (.LNK) Analyzer |
| Author: Dawid Farbaniec      Version 2025.1 |
\-----/

File: .\m.LNK.anomaly

---== ShellLinkHeader ===
HeaderSize: 0x0000004C
HeaderSize.IsValid: True
LinkCLSID: 00021401-0000-0000-c000-000000000046
LinkCLSID.IsValid: True
LinkFlags: 0x000002E9 [HasLinkTargetIDList, Has
RelativePath, HasArguments, HasIconLocation, Is
Unicode, HasExpString]
```

Figure 7. dotLNK (.LNK Analysis Tool)

Analyze a Windows Shortcut (.LNK)

ANOMALY

```
зона://ethical.blue x + v - □ X
PS C:\de54> .\dotLNK.exe anomaly.LNK
----- ShellLinkHeader -----
HeaderSize: 0x0000004C
HeaderSize.IsValid: True
LinkCLSID: 00021401-0000-0000-c000-000000000046
LinkCLSID.IsValid: True
LinkFlags: 0x002040CF [HasLinkTargetIDList, HasLinkInfo, HasName, HasRelativePath, HasIconLocation, IsUnicode, HasExpIcon, DisableKnownFolderTracking]
```

with a dotLNK
Console App

```
зона://ethical.
ExtraData.Tracker
Length: 0x0000005
Version: 0x0000000
MachineID: ethical
Droid: 0502861a-5b4
Droid_2: 64e4963c-f
DroidBirth: 0502861a
DroidBirth_2: 64e496
MAC Address: a9:77:0h
```

c:\c43:7a55

please wait...



David Farber



x86-64 (x64) Assembly Lang
Мова Асемблера (Мова
x86-64 (x64) Assemblerspr
Lingua Assemblatoria

podstawy PowerShell
systemach Microsoft Windows
Assembler x64/arm64)
e w językach C#/IL
ucha bloków

Oxae58

blue Magazine



Bibliography

- [1] Japan Electronics and Information Technology Industries Association (JEITA), *Exchangeable Image File Format for Digital Still Cameras: EXIF Version 3.0*, 2024.
- [2] Mandiant, Inc., *FLARE Obfuscated String Solver Documentation*, 2026.
- [3] Microsoft Corporation, *[MS-SHLLINK]: Shell Link (.LNK) Binary File Format*, 2025.
- [4] Microsoft Corporation, *Windows PowerShell Language Specification Version 3.0*, 2012.
- [5] OffSec Services Limited, <https://www.kali.org/tools/binwalk/>, 2026.
- [6] OffSec Services Limited, <https://www.kali.org/tools/binwalk3/>, 2026.
- [7] Phil Harvey, <https://exiftool.org/>, 2026.