

# Reverse Engineering

## 5. Compiler Stub Recognition

Ing. Tomáš Zahradnický, EUR ING, Ph.D.  
Ing. Martin Jirkal  
Ing. Josef Kokeš



**FACULTY  
OF INFORMATION  
TECHNOLOGY  
CTU IN PRAGUE**

Czech Technical University in Prague  
Faculty of Information Technology  
Department of Information Security

Version 2021-09-11

# Table of Contents

- 1 Motivation
- 2 Compiler Recognition
  - Microsoft Visual C++
  - Borland
  - Other Compilers
- 3 Startup, Runtime, and Library Code
  - Identifying Libraries
  - Library Signatures
  - Discovering Libraries

# Motivation

## Question

Why would we want to identify the compiler which was used to compile an executable?

## Answer

To reduce the amount of code we have to analyze. Known compiler stubs can be excluded from further analysis.

The same question can be asked for all 3<sup>rd</sup> party libraries the executable is linked with. If we can identify them, we can exclude them from the analysis.

Furthermore, the knowledge of the functions in these libraries can be used to extend our understanding of the application by giving types and meaning to variables which are passed to them.

# Introduction

- Every compiler uses its own style during the compilation (e.g. data and code locations are specific to the compiler).
- Every executable links to specific runtime libraries, either statically or dynamically. Identification of runtime library code can tell us a lot about the compiler.
- To identify which compiler was used to compile a binary, we need to know where, inside the binary, to look for important pieces of information.
- The binary is usually large so knowledge of what to skip speeds-up the analysis significantly.

# Compiler Recognition — Microsoft Visual C++

Marketing version	Internal version	CL.EXE version	Possible imported DLL	Release date
6	6.0	12.00	msvcrt.dll, msvcp60.dll	June 1998
.NET	7.0	13.00	msvcr70.dll, msvcp70.dll	February 13, 2002
.NET 2003	7.1	13.10	msvcr71.dll, msvcp71.dll	April 24, 2003
2005	8.0	14.00	msvcr80.dll, msvcp80.dll	November 7, 2005
2008	9.0	15.00	msvcr90.dll, msvcp90.dll	November 19, 2007
2010	10.0	16.00	msvcr100.dll, msvcp100.dll	April 12, 2010
2012	11.0	17.00	msvcr110.dll, msvcp110.dll	September 12, 2012
2013	12.0	18.00	msvcr120.dll, msvcp120.dll	October 17, 2013
2015	14.0	19.00	vcruntime140.dll	July 20, 2015
2017	14.1+	19.10+	vcruntime140.dll	March 7, 2017
2019	14.2+	19.20+	vcruntime140.dll	April 2, 2019

- MSVC compiler can be recognized by searching the import directory for the libraries above.
- MSVC-decorated symbol names usually start with a ? sign.

```
?doSomething@CMFCApplicationView@@QAEXXZ
```

# Compiler Recognition — Borland

- May import the BORLNDMM.DLL library.
- Mangled names start with the @ symbol.

```
@TModule@ValidWindow$qp14TWindowsObject
```

- Delphi contains data type names at the beginning of CODE segment. Strings like Boolean, Integer, TObject, Char etc. seen near the start of the file make Delphi identification quite fast and easy.

## Compiler Recognition — A Delphi Example

[illegible]

# Compiler Recognition — Other

- GCC

- `cygwin1.dll` is imported if Cygwin was used for compilation.
- `msvcrt.dll` is imported if MinGW was used for compilation.<sup>1</sup>
- Mangled names usually start with `_Z`.

`_Z1hv`

- Watcom

- Mangled names usually start with `W`.

`W?method$_class$n__v`

- FORTRAN

- Can import `libifcoremd.dll`, `libifportmd.dll`, `libiomp5md.dll`.

---

<sup>1</sup>Note that non-MinGW applications also frequently link against `msvcrt.dll`. 



# Startup, Runtime, and Library Code

- As explained in Lecture 2, an executable starts execution at the main entry point. That location is far away from the `main()` function.
- The startup code is heavily dependent on the compiler, compiler options and the runtime library.
- It is **mostly** uninteresting for analysis and analyzing it would be a waste of time.
- For this reason it is useful to mark the prologue code as “library code”, as it is of little interest to the reverse engineer.
- We can do the same with the libraries we find in the program.

# Identifying Libraries

- With compiler identified, we can try to find as many library routines as possible and exclude them from the analysis.
- This is done by performing pattern matching against signatures generated for each library.
- We search the executable for known signatures and notice all matched functions. These are:
  - renamed to the name of the matched function;
  - marked down as library code.
- IDA Pro uses an approach called F.L.I.R.T. (Fast Library Identification and Recognition Technology) [1], which does just this.

# Creating Signatures

- Each function from a library for which we have the source code can be described with a pattern.
- The pattern could be the first  $X$  bytes from the function start in the machine code, or the entire function code up to the `ret` instruction.
  - Note that the exact version of the library as well as its build variant, such as Debug or Release, is important here.
- Although we can set  $X$  to an arbitrary value or make a pattern out of the entire function, note that functions with the same body and different names (e.g. `htonl` and `ntohl`) cannot be distinguished one from the other. In that case we choose only one of the functions and discard the other.
- Patterns for the whole library are then combined into a signature file.
- IDA Pro provides a FLAIR SDK for this purpose.

# Libraries, how to find them? I

- How can we find that an executable uses a particular library?
- If the library is dynamically linked, it is easy:
  - Windows: `dumpbin`, CFF Explorer, Dependency Walker.
  - Linux: `objdump`, `readelf`, or just `ldd`.
  - OS X: `otool` or `dyldinfo`.
- If the library is statically linked, use `strings` to find:
  - copyright statements,
  - version statements,
  - error messages.

Then paste the discovered statements, quoted, into your favorite search engine!

- This will not, usually, find the exact version, but it will provide a good initial approximation.

# Libraries, how to find them? II

```
mail:MacOS admin$ strings -a MediaManager
```

```
...
```

```
Unknown Exif Version
```

```
Exif Version %d.%d
```

```
0100
```

```
FlashPix Version 1.0
```

```
0101
```

```
FlashPix Version 1.01
```

```
...
```

```
Copyright information. In this standard the tag is used to indicate both the...
```

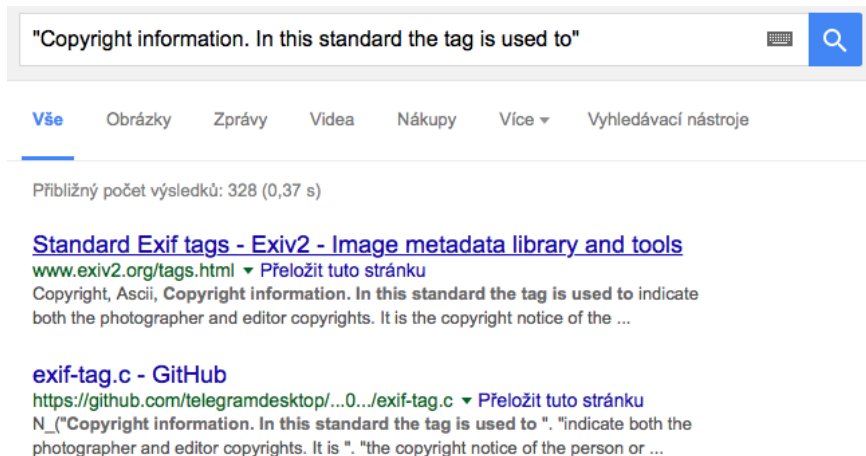
```
...
```

```
This tag is used to record the name of an audio file related to the image...
```

```
...
```

**Figure:** Running the strings tool on an executable to discover any valuable library identification strings.

# Libraries, how to find them? III



The screenshot shows a search engine interface. At the top is a search bar with the text "Copyright information. In this standard the tag is used to". To the right of the search bar is a keyboard icon and a magnifying glass icon. Below the search bar are several tabs: "Vše", "Obrázky", "Zprávy", "Videa", "Nákupy", "Více", and "Vyhledávací nástroje". Below the tabs, it says "Přibližný počet výsledků: 328 (0,37 s)". The first search result is titled "Standard Exif tags - Exiv2 - Image metadata library and tools" and has the URL "www.exiv2.org/tags.html". Below the title is a link "Přeložit tuto stránku". The snippet for this result is "Copyright, Ascii, Copyright information. In this standard the tag is used to indicate both the photographer and editor copyrights. It is the copyright notice of the ...". The second search result is titled "exif-tag.c - GitHub" and has the URL "https://github.com/telegramdesktop/...0.../exif-tag.c". Below the title is a link "Přeložit tuto stránku". The snippet for this result is "N\_('Copyright information. In this standard the tag is used to '. 'indicate both the photographer and editor copyrights. It is '. 'the copyright notice of the person or ...".

"Copyright information. In this standard the tag is used to"

Vše Obrázky Zprávy Videa Nákupy Více ▾ Vyhledávací nástroje

Přibližný počet výsledků: 328 (0,37 s)

[Standard Exif tags - Exiv2 - Image metadata library and tools](#)  
[www.exiv2.org/tags.html](http://www.exiv2.org/tags.html) ▾ Přeložit tuto stránku  
Copyright, Ascii, Copyright information. In this standard the tag is used to indicate both the photographer and editor copyrights. It is the copyright notice of the ...

[exif-tag.c - GitHub](#)  
<https://github.com/telegramdesktop/...0.../exif-tag.c> ▾ Přeložit tuto stránku  
N\_("Copyright information. In this standard the tag is used to ". "indicate both the photographer and editor copyrights. It is ". "the copyright notice of the person or ...

Figure: An identified library — libexif.

# Bibliography



Chris Eagle: *The Ida Pro Book 2nd Edition*, no starch press, 2011.



Eldad Eilam: *Reversing: Secrets of Reverse Engineering*, Wiley Publishing, Inc., 2005.



Reverend Bill Blunden: *The Rootkit Arsenal*, Wordware Publishing, Inc., 2009.