Introducing the Dragon to the Greek

Towards Ghidra Reverse Engineering Tools for Delphi Executables

Lukas Wenz

August 18, 2025

Rheinische Friedrich-Wilhelms-Universität Bonn
MA-INF 0401 - Master Thesis
Institute for Computer Science IV
Work group for IT-Security

Table of contents - Introducing the 🐉 to the 🏛️

C:\intermediate_talk\motivation

C:\intermediate_talk\motivation\context

- Rapid advancement of technology $\rightarrow$ *novel challenges* for cyber security:
  - Cross-platform attacks [3, 8]
  - Usage of LLMs for cybercrimes [11]
- **Problematic trend**: *Sole focus* on emerging threats
- **Consequence**: Neglecting older yet still relevant issues

WormGPT: A Large Language Model Chatbot for Criminals

Publisher: IEEE Cite This PDF

Mohamed Fazil Mohamed Firdhous ; Walid Elbreiki ; Ibrahim Abdullahi ; B.H. Sudantha ; Rahmat Budiarto

All Authors

1 Cites in Paper | 440 Full Text Views

Abstract

Document Sections

I. Introduction
II. Related Work
III. Background Information
IV. WormGPT
V. Conclusions

Authors
Figures
References
Citations

Abstract:
In recent years, AI has made many advances in multiple fronts. One such recent advance is the development of Large Language Models(LLMs), a deep learning model trained with a large set of data on billions of parameters. Many Generative Pretrained Transformer (GPT) models have been developed on these LLMs. ChatGPT has become the most popular GPT used by many people. Recently, a blackhat GPT named WormGPT has been launched for cybercriminals. WormGPT is capable of launching many types of attacks resulting in irreparable damage. Hence, Users are required to be familiar with these tools, if they are to protect themselves. This article is possibly the very first attempt to initiate a discussion on this topic within the formal research circles. This article starts with an introduction to LLMs and GPTs and then presents the details of WormGPT, its capabilities and the damage it can cause. Finally, a set of safeguards has been presented that can help to protect users from possible attacks.

Published in: 2023 24th International Arab Conference on Information Technology (ACIT)

Date of Conference: 06-08 December 2023 **DOI:** 10.1109/ACIT58888.2023.10453752

Date Added to IEEE *Xplore*: 18 March 2024 **Publisher:** IEEE

Figure 1: Screenshot of an IEEE published paper on WormGPT, an LLM focussed on cybercrime. [9]

C:\intermediate_talk\delphi

- **Overlooked issue**: lacking understanding of programs written in the Delphi programming language.
- **Delphi**:
  - Originally developed by Borland Software Corporation in 1995
  - Dialect of Object Pascal
  - Currently maintained by *Embarcadero Technologies*
  - Produces executables primarily for the Windows OS

Figure 2: Embarcadero Technologies logo. [4]

Figure 3: Delphi logo. [4]

C:\intermediate_talk\delphi\general_relevance (1)

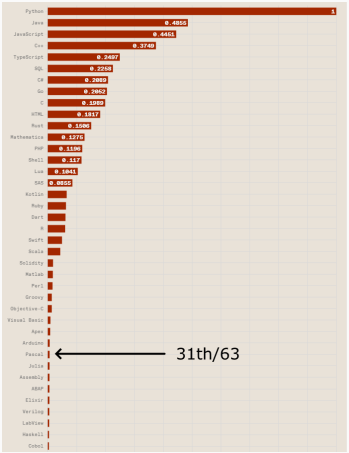


Figure 4: Screenshot of IEEE Spectrum report about *The Top Programming Languages 2024*, **sorted by scientific usage**. [1]

C:\intermediate_talk\delphi\general_relevance (1)

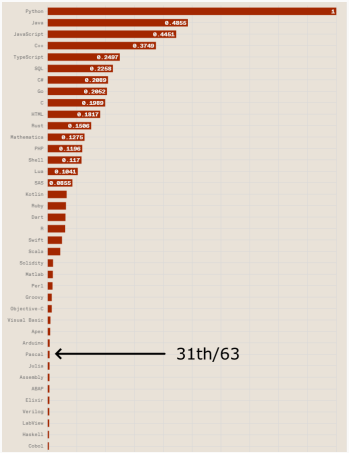


Figure 4: Screenshot of IEEE Spectrum report about *The Top Programming Languages 2024*, sorted by scientific usage. [1]

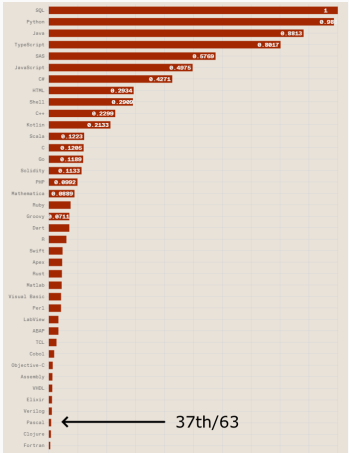


Figure 5: Screenshot of IEEE Spectrum report about *The Top Programming Languages 2024*, sorted by employer demand. [1]

C:\intermediate_talk\delphi\relevance_for_malware (1)

Figure 6: Screenshot of Crowdstrike Holdings, Inc. blog post reporting Delphi-based malware in Latin America. [12]

C:\intermediate_talk\delphi\relevance_for_malware (2)

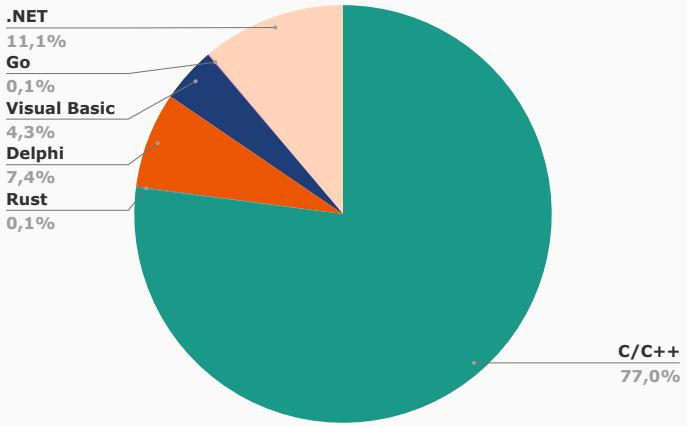


Figure 7: Compiler language distribution of user-submitted files to the automated malware unpacking service *UnpacMe* (data from [13]).

C:\intermediate_talk\ghidra

C:\intermediate_talk\ghidra\related_work (1)

What tools exist for reversing Delphi executables?

C:\intermediate_talk\ghidra\related_work (1)

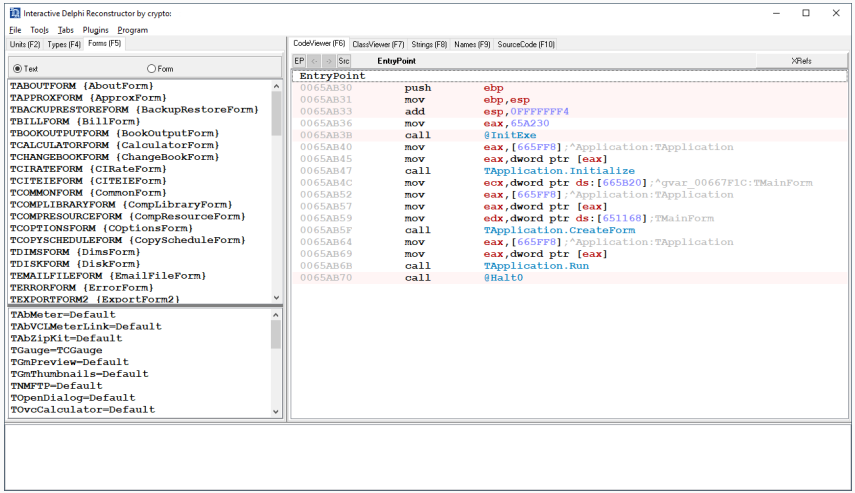


Figure 8: Interactive Delphi Reconstructor (IDR) is the only well-known available tool (32bit).

C:\intermediate_talk\ghidra\related_work (1)

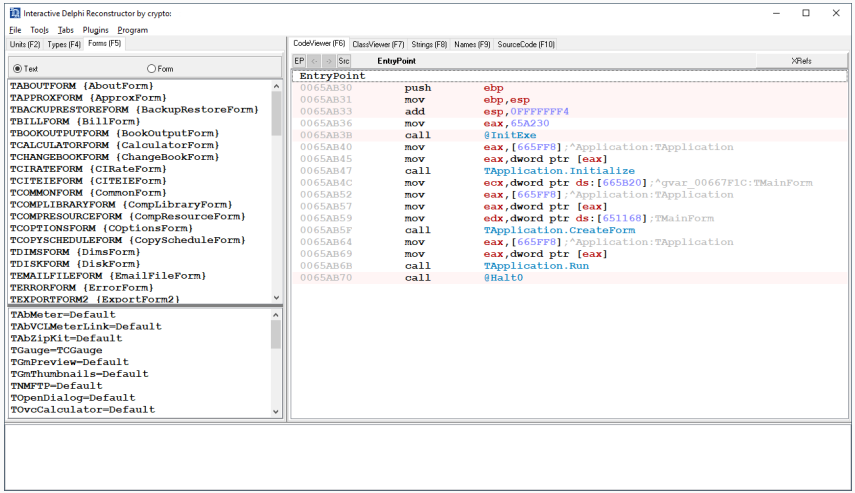
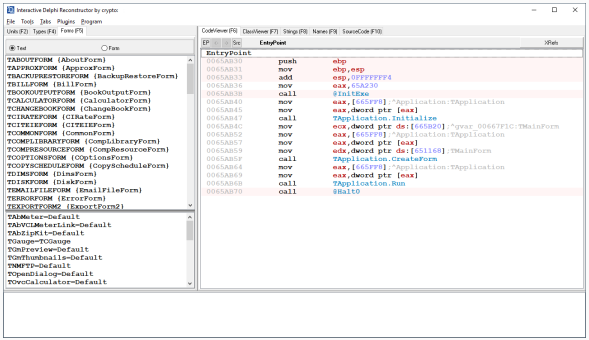


Figure 8: Interactive Delphi Reconstructor (IDR) is the only well-known available tool (32bit).

C:\intermediate_talk\ghidra\related_work (2)



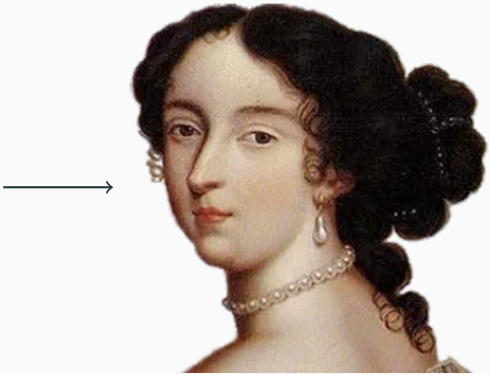


Figure 9: IDR provides Delphi decompilation support for IDA Pro only.

Ghidra:

- Reverse engineering tool
- Seen by many security researchers as a competitor to IDA Pro
- Developed by NSA
- Free and open source
- Free-to-use plugin API
  - As opposed to Binary Ninja

Figure 10: Ghidra logo. [10]

C:\intermediate_talk\ghidra\related_work (3) (continued)

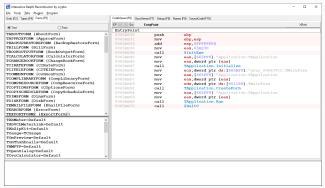
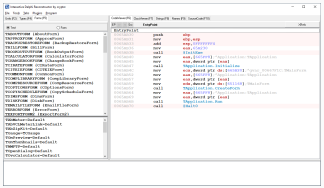


Figure 11: Dataflow of related work programs for reversing Delphi executables.



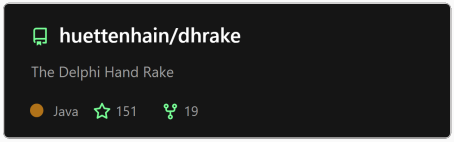


Figure 11: Dataflow of related work programs for reversing Delphi executables.

```
void entry(void)

{
  FUN_00411798(&DAT_005e2cec);
  FUN_005d6be8(*(undefined4 *)PTR_DAT_005f2894);
  FUN_005d8bd4(*(undefined4 *)PTR_DAT_005f2894,1)
       ;
  FUN_005d6c00(*(undefined4 *)PTR_DAT_005f2894,&
       PTR_LAB_005e2af4,PTR_DAT_005f275c);
  FUN_005d6d60(*(undefined4 *)PTR_DAT_005f2894);
                    /* WARNING: Subroutine does
                         not return */
  FUN_00409f8c();
}
```

```
void EntryPoint(void)

{
  FUN_00411798(&DAT_005e2cec);
  TApplication.Initialize(*(undefined4 *)
       Application);
  FUN_005d8bd4(*(undefined4 *)Application,1)
       ;
  Vcl::Forms::TApplication::CreateForm(*(
       undefined4 *)Application,&
       PTR_LAB_005e2af4,gvar_005F275C);
  TApplication.Run(*(undefined4 *)
       Application);
                    /* WARNING: Subroutine
                         does not return */
  @Halt0();
}
```

Figure 12: Ghidra decompilation of a typical Delphi application's entry point.
Left: Before IDR+dhrake; Right: After IDR+dhrake.

C:\intermediate_talk\objectives

C:\intermediate_talk\objectives

Problem 1:

Little to no resources for the Delphi file format available.

Problem 2:

Little tooling for reversing Delphi executables available: IDR+Dhrake

Also: 64bit is not fully supported.

C:\intermediate_talk\objectives

Problem 1:

Little to no resources for the Delphi file format available.

Problem 2:

Little tooling for reversing Delphi executables available: IDR+Dhrake

Also: 64bit is not fully supported.

Objective 1: 🏛️

Gain a deep understanding about the Delphi file format and document it.

C:\intermediate_talk\objectives

Problem 1:

Little to no resources for the Delphi file format available.

Problem 2:

Little tooling for reversing Delphi executables available: IDR+Dhrake

Also: 64bit is not fully supported.

Objective 1: 🏛️

Gain a deep understanding about the Delphi file format and document it.

Objective 2: 🐉

Create a tool that makes reversing Delphi executables more accessible. Exemplary functionality: *Symbol recovery*.

For 32bit & 64bit.

C:\intermediate_talk\format_analysis

```
$ strings delphi.exe | head -n 100
This program must be run under Win32
.text
`.itext
`.data
.bss
.idata
[...]
InheritsFrom
Self
AClass
MethodAddress
Self
Name
QualifiedClassName
Self
FieldAddress
```

```
__classmethod bool __fastcall
    InheritsFrom(TClass AClass);

__classmethod void* __fastcall
    MethodAddress(const UnicodeString
    Name)/* overload */;

__classmethod UnicodeString __fastcall
    QualifiedClassName();
```

Figure 13: Corresponding method signatures in C++ style, derived from official documentation [5, 6, 7].

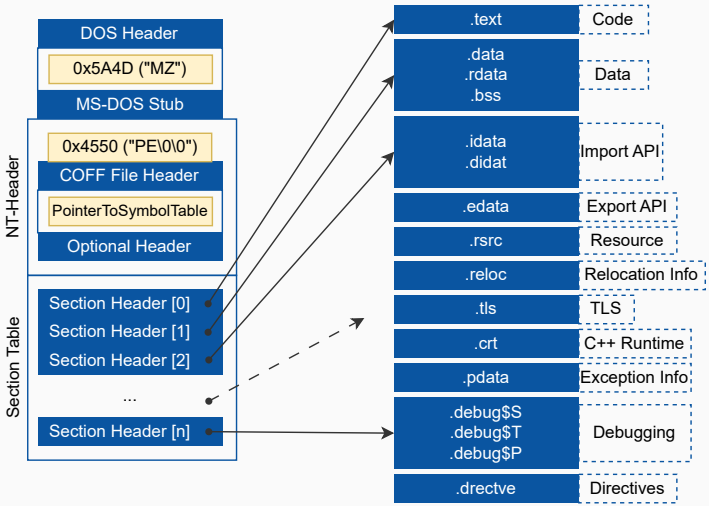


Figure 14: Overview of PE format (Image only), derived from [2].

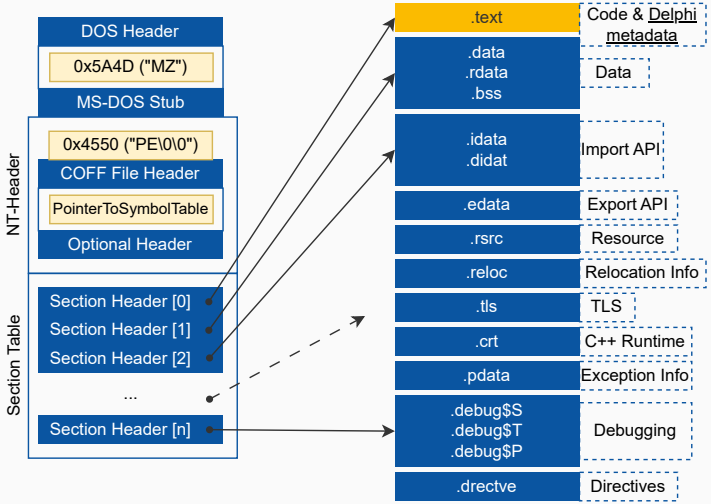


Figure 14: Overview of PE format (Image only), derived from [2].

C:\intermediate_talk\format_analysis\methodology

- Trial and error approach, grounded in...
  - *manual analysis* of the *.text* section
  - iterative *hypothesis refinement* based on counter-evidence
  - *comparative* evaluation (e.g. to IDR) & *convention-based* reasoning
- Assigned confidence levels to every field

C:\intermediate_talk\format_analysis\tables

Table 2: *Layout of* **VMT***s for versions Delphi 2010 and onwards for 32/64-bit architectures. A field's offset (O) and size (S) information can be seen in the columns of the corresponding letter and architecture number. The column CL depicts the Confidence Level as introduced in Table 1.*

O'32	O'64	S'32	S'64	Fieldname	Description	CL
0d	0d	4B	8B	`NextStruct`	Optional pointer to the structure following the corresponding VMT (same as VMT's `NextStruct` field).	2
4d	8d	4B	8B	`DirectAncestorClass`	Optional pointer to the parent class in the RTTI object's inheritance chain (indirect reference).	2
8d	16d	2B	2B	unknown	unknown; likely flags	0
12d	24d	2B	2B	`NumOfMethodEntryRefs`	Number of `MethodEntryRef` records following	2
16d	32d	4B	8B	`MethodEntryRef 1`	`MethodEntryRef` record 1	2
20d	40d	4B	8B	...	...	...
24d	48d	4B	8B	`MethodEntryRef`$_n$	`MethodEntryRef` record n	2
28d	56d	4B	8B	unknown	unknown	0
32d	64d	4B	8B	`NamedFunction 1`	Pointer to the function definition of named function 1	2
36d	72d	4B	8B	`NamedFunction`$_n$	Pointer to the function definition of named function n	2
40d	80d	4B	8B	`ParentEntry`	Optional pointer to the `ParentEntry` structure	2

Table 3: *Layout of* **MDT***s (top level view) for versions Delphi 2010 and onwards for 32/64-bit architectures. A field's offset (O) and size (S) information can be seen in the columns of the corresponding letter and architecture number. The column CL depicts the Confidence Level as introduced in Table 1.*

O'32	O'64	S'32	S'64	Fieldname	Description	CL
0d	0d	2B	2B	unknown	unknown; likely flags	0
2d	2d	2B	2B	`NumOfMethodEntryRefs`	Number of `MethodEntryRef` records following	2
4d	4d	8B	12B	`MethodEntryRef 1`	`MethodEntryRef` record 1	2
...	...	...	...	...	...	...
n.a.$_1$	n.a.$_2$	8B	12B	`MethodEntryRef`$_n$	`MethodEntryRef` record n	2
n.a.$_3$	n.a.$_4$	2B	2B	unknown	unknown	0
n.a.	n.a.	len(S)	len(S)	`NamedFunction 1`	Pointer to the function definition of named function 1	2
n.a.	n.a.	dyn	dyn	`NamedFunction`$_n$	Pointer to the function definition of named function n	2
n.a.	n.a.	dyn	dyn	`ParentEntry`	Optional pointer to the `ParentEntry` structure	2

Table 4: *Layout of a* **MethodEntry** *substructure for versions Delphi 2010 and onwards for 32/64-bit architectures. A field's offset (O) and size (S) information can be seen in the columns of the corresponding letter and architecture number. The column CL depicts the Confidence Level as introduced in Table 1.*

O'32	O'64	S'32	S'64	Fieldname	Description	CL
0d	0d	2B	2B	`LenOfMethodEntry`	Cardinal size of this `MethodEntry` structure	2
2d	2d	4B	8B	**`FunctionDefinition`**	2B long field followed by a variable length string field	2
6d	10d	len(S)	len(S)	**`NameOfFunction`**	Pointer to the function definition	2
n.a.	n.a.	2B	2B	unknown	Pascal String of the name represented by the previous field.	2
n.a.	n.a.	4B	8B	**`ReturnType`**	Byte sequence 03h 08h 00h 00h	2
n.a.	n.a.	2B	2B	unknown	Optional pointer to class object representing the function's return type;	2
n.a.	n.a.	1B	1B	`NumOfParamEntries`	unknown	0
n.a.	n.a.	1B	1B	unknown	Number of `ParamEntry` substructures $k \geq 1$ following this structure	2
n.a.	n.a.	dyn	dyn	**`ParamEntry`**$_1$	unknown	0
n.a.	n.a.	dyn	dyn	**`ParamEntry`**$_{2,\ldots,k}$	1st `ParamEntry` substructure. For class methods, it is a pointer to the VMT's RTTI class type; its fields are 'Self' (indirect reference).	2
n.a.	n.a.	4B	4B	Separator	Optional additional `ParamEntry` substructures. Only exist for $k > 1$. In this case, each `ParamEntry`$_i$ for $i \in \{2,\ldots,k\}$ is preceded by 3 bytes of unknown data!	2
					Separating byte: 00h	

Table 5: *Layout of the* **ParamEntry** *substructure for versions Delphi 2010 and onwards for 32/64-bit architectures. A field's offset (O) and size (S) information can be seen in the columns of the corresponding letter and architecture number. The column CL depicts the Confidence Level as introduced in Table 1.*

O'32	O'64	S'32	S'64	Fieldname	Description	CL
0d	0d	4B	8B	**`AddrOfRtti`**	Optional pointer to the corresponding RTTI object (indirect reference).	1
4d	8d	2B	2B	unknown	unknown	0
6d	10d	len(S)	len(S)	**`NameOfParam`**	A Pascal String stating the name of the parameter.	1
n.a.	n.a.	8B	8B	`DestroyDestructor`	Optional null byte padding.	2
n.a.	n.a.	176d	176d	...	...	...

Table 6: *Layout of the* **ParamEntry** *substructure for versions Delphi 2010 and onwards for 32/64-bit architectures. A field's offset (O) and size (S) information can be seen in the columns of the corresponding letter and architecture number. The column CL depicts the Confidence Level as introduced in Table 1.*

Table 8: *Layout of the* **RTTI Class** *structure for versions Delphi 2010 and onwards for 32/64-bit architectures. A field's offset (O) and size (S) information can be seen in the columns of the corresponding letter and architecture number. The column CL depicts the Confidence Level as introduced in Table 1.*

O'32	O'64	S'32	S'64	Fieldname	Description	CL
0d	0d	1B	1B	`RttiObjectType`	Magic byte declaring the type of RTTI object (`0x07` for RTTI_Class objects).	2
1d	1d	len(S)	len(S)	**`RttiObjectName`**	Name of the RTTI_Class object.	0
n.a.	n.a.	4B	8B	`VmtsNextStruct`	Optional pointer to the structure following the corresponding VMT (same as VMT's `NextStruct` field).	2
n.a.	n.a.	4B	8B	`DirectAncestorClass`	Optional pointer to the parent class in the RTTI object's inheritance chain (indirect reference).	2
n.a.	n.a.	2B	2B	unknown	unknown; likely flags	0
n.a.	n.a.	len(S)	len(S)	**`RttiNamespace`**	The *namespace* of the RTTI class as a Pascal S...	2
n.a.	n.a.	2B	2B	`NumOfPropertyEntries`	...	2

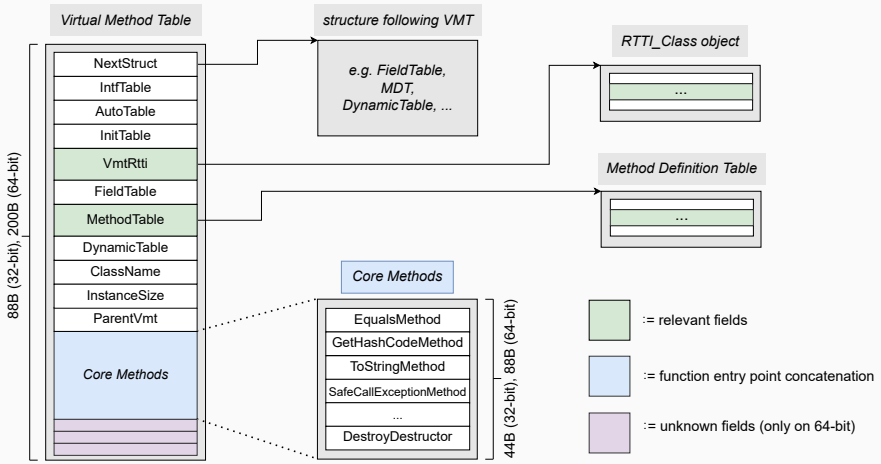


Figure 15: Schematic illustration of the VMT layout. Parts of substructures coloured in green are relevant fields for navigation in the symbol recovery task.

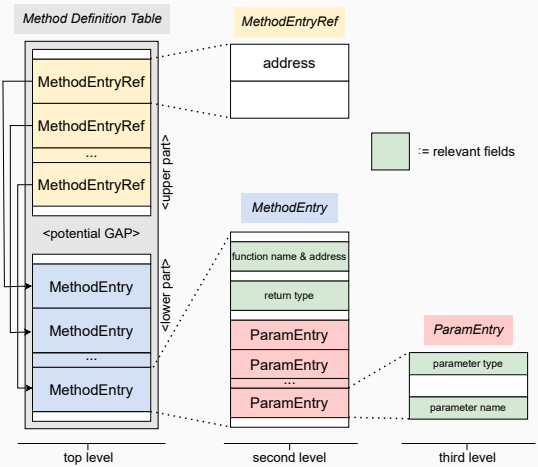


Figure 16: Abstract illustration of the MDT layout, including its repeating and nested substructures. Parts of substructures coloured in green are relevant fields for the symbol recovery task.

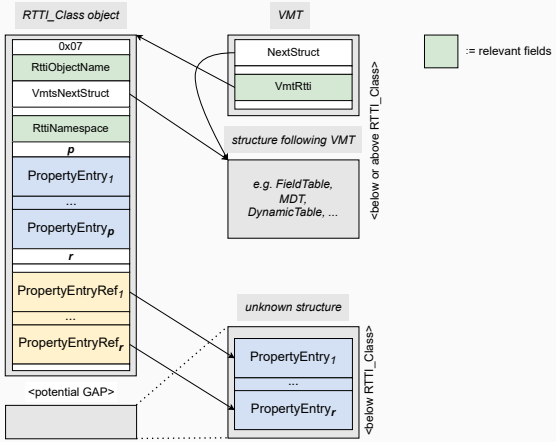


Figure 17: Abstract illustration of the RTTI_Class layout, including its repeating `PropertyEntry` structures as well as its relationship to its corresponding VMT. Parts of substructures coloured in green are relevant fields for the symbol recovery task.

C:\intermediate_talk\tooling

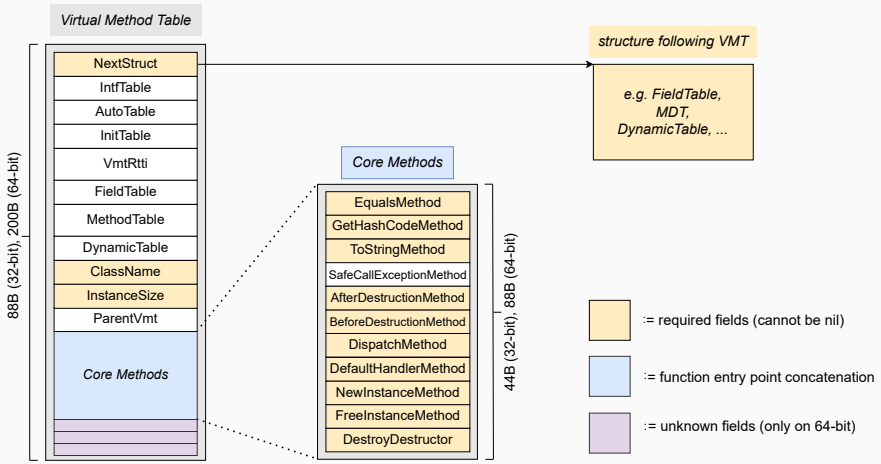


Figure 18: Schematic illustration of the VMT layout. Parts of substructures coloured in yellow are required fields potentially usable for VMT detection.

C:\intermediate_talk\evaluation

C:\intermediate_talk\evaluation\evaluation_set

Executable	Type	Architecture	Delphi Version	Result
VclMinimum-32	crafted	32-bit	Delphi 12 Athens	success
VclMinimum-64	crafted	64-bit	Delphi 12 Athens	success
Concat-32	crafted	32-bit	Delphi 12 Athens	success
Concat-64	crafted	64-bit	Delphi 12 Athens	success
VclEnhanced-32	crafted	32-bit	Delphi 12 Athens	success
VclEnhanced-64	crafted	64-bit	Delphi 12 Athens	success
RADStudio	legitimate	32-bit	Delphi 10.3 Rio	success
Acrylic DNS Proxy	legitimate	32-bit	Delphi 7	fail, old format
Kiron	malware	64-bit	Delphi 11 Alexandria	success
CosmicBanker	malware	64-bit	Delphi 10.3 Rio	success
DanaBot	malware	32-bit	Delphi 10.2 Tokyo	success
DanaBot	malware	32-bit	Delphi 10.2 Tokyo	success
DanaBot	malware	32-bit	Delphi 10.2 Tokyo	success
DanaBot	malware	32-bit	Delphi 10.2 Tokyo	success
Culebra	malware	32-bit	Delphi XE2-XE4	success
Culebra, packed	malware	32-bit	unknown	success, when unpacked
Culebra, packed	malware	64-bit	unknown	fail, packed

C:\intermediate_talk\evaluation\format_change

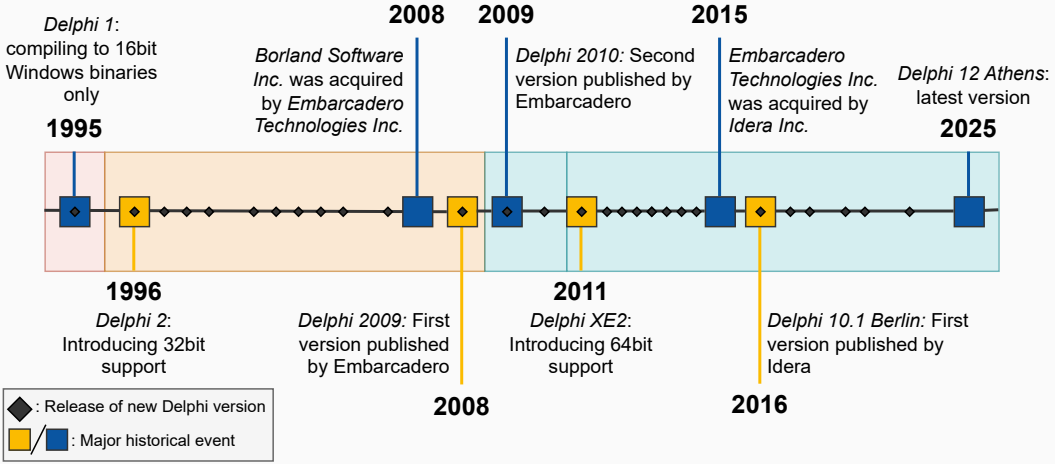


Figure 19: Overview of the various changes in Delphi's file format aligned to its historical timeline.

C:\intermediate_talk\evaluation\accuracy_results

Table 1: Results for **successful** runs after a clean import and auto-analysis of the executables.

Executable	#funcs	#rec_func / #rec_ret	#rec_param set	Acc_func / Acc_ret	Acc_param set
VclMinimum-32	9796	4846	4373	49,47 %	44,64%
VclMinimum-64	10956	5007	4547	45,70 %	41,50%
Concat-32	733	66	53	9,00 %	7,23%
Concat-64	831	70	57	8,42 %	6,86%
VclEnhanced-32	9699	4847	4374	49,97 %	45,10%
VclEnhanced-64	10966	5007	4547	45,66 %	41,46%
RADStudio	2873	896	703	31,19 %	24,47%
Kiron	24533	13214	11727	53,86 %	47,80%
CosmicBanker	16033	8695	7513	54,23 %	46,86%
DanaBot	5295	2803	2481	52,94 %	46,86%
DanaBot	5738	3055	2679	53,24 %	46,69%
DanaBot	13737	5341	4717	38,88 %	34,34%
DanaBot	10763	5576	4926	51,81 %	45,77%
Culebra	12924	5105	4575	39,50 %	35,40%
Culebra, packed	9053	2857	2472	31,56 %	27,31%

```
void entry(void)

{
  FUN_00411798(&DAT_005e2cec);
  Vcl::Forms::TApplication::Initialize(*(
       TApplication **)PTR_DAT_005f2894);
  FUN_005d8bd4(*(undefined4 *)PTR_DAT_005f2894
       ,1);
  Vcl::Forms::TApplication::CreateForm(*(
       TApplication **)PTR_DAT_005f2894,(
       TComponentClass *)&PTR_LAB_005e2af4,(
       TApplication *)PTR_DAT_005f275c);
  Vcl::Forms::TApplication::Run(*(TApplication
       **)PTR_DAT_005f2894);
                    /* WARNING: Subroutine does
                         not return */
  FUN_00409f8c();
}
```

```
void EntryPoint(void)

{
  FUN_00411798(&DAT_005e2cec);
  TApplication.Initialize(*(undefined4 *)
       Application);
  FUN_005d8bd4(*(undefined4 *)Application,1);
  Vcl::Forms::TApplication::CreateForm(*(
       undefined4 *)Application,&
       PTR_LAB_005e2af4,gvar_005F275C);
  TApplication.Run(*(undefined4 *)Application)
       ;
                    /* WARNING: Subroutine
                         does not return */
  @Halt0();
}
```

Figure 20: Ghidra decompilation of a typical Delphi application's entry point.
Left: This thesis' tool; Right: After IDR+dhrake.

C:\intermediate_talk\evaluation\IDR_comparison (2)

```
bool System::TObject::InheritsFrom(TObject
     *Self, TClass *AClass)

{
  ...
}
```

```
undefined4 TObject.InheritsFrom(int param_1,
     int param_2)

{
  ...
}
```

Figure 21: Ghidra decompilation of a typical Delphi application's function definition.
Left: This thesis' tool; Right: After IDR+dhrake.

```
__classmethod bool __fastcall InheritsFrom(TClass AClass);
```

C:\intermediate_talk\summary

C:\intermediate_talk\summary

- **File format analysis**: Identification of VMTs, MDTs, RTTI objects and their layouts
- **Reverse engineering tool**: Forward references of 88B/200B in *.text* & traversal of structures to retrieve and apply meta data via the Ghidra API
  - Recovers 47% *Acc_func* / *Acc_ret* and 39% *Acc_param* & supports 32-bit and 64-bit
  - Limitation: weak recovery rates on non-graphical applications
- **Future work**: Increase coverage & investigate yet unknown fields

Thank you for listening.
Questions? Suggestions?

The tool is available on GitHub! I'm grateful to everyone who's interested in the project. (and for every ⭐).

References i

S. Cass.
The top programming languages 2024, 8 2024.

S. Daulaguphu.
Mastering pe structure for malware analysis: A layman's guide, 8 2022.

S. Eckels.
Ready, set, go — golang internals and symbol recovery, 2 2022.

Embarcadero Technologies Inc.
Embarcadero logo.

Embarcadero Technologies Inc.
System.tobject.qualifiedclassname, 10 2011.

Embarcadero Technologies Inc.
System.tobject.inheritsfrom, 7 2020.

References ii

Embarcadero Technologies Inc.
System.tobject.methodaddress, 2 2023.

R. Lakshmanan.
Cybercriminals exploit popular game engine godot to distribute cross-platform malware, 11 2024.

M. F. Mohamed Firdhous, W. Elbreiki, I. Abdullahi, B. Sudantha, and R. Budiarto.
Wormgpt: A large language model chatbot for criminals.
In *2023 24th International Arab Conference on Information Technology (ACIT)*, pages 1–6, 2023.

National Security Agency.
Ghidra logo.

References iii


K. Poireault.
The dark side of generative ai: Five malicious llms found on the dark web, 8 2023.


K. Ratto.
Still alive: Updates for well-known latin america ecrime malware identified in 2023, 2 2024.


S. Wilson.
Malware trends: Yearly 2023, 1 2024.

Table 2: Confidence Levels used in format analysis tables including their used abbreviations.

Confidence Level	Description
Unknown (0)	The purpose of the field is unknown or cannot be reasonably deduced. The field might have an even more fine-granular structure.
Provisionally Hypothesised (1)	A tentative interpretation based on a consistently recurring pattern across all observations, though the exact purpose remains unclear.
Strongly Hypothesised (2)	A robust interpretation with a deducible purpose and clear structural coherence, consistent across all observations.
Trusted Reference (3)	The structure and purpose of the field is either confirmed by an official source, or derived using the de facto standard tool for Delphi reverse engineering (IDR).

C:\intermediate_talk\evaluation\stats

Executable	Mode	#VMTs	#pre funcs	#post funcs	#recov. func'names	#recov. FQNs	#recov. ret.	#recov. param-sets
VclMinimum-32	!a	817	52	4903	4851	4851	4851	4378
	a	817	6878	9796	4846	4846	4846	4373
	a+t	817	9799	9799	4846	4846	4846	4373
VclMinimum-64	!a	840	52	5059	5007	5007	5007	4547
	a	840	8762	10956	5007	5007	5007	4547
	a+t	840	10970	10970	5007	5007	5007	4547
Kiron	!a	2589	66	13280	13214	13214	13214	11727
	a	2589	18112	24533	13214	13214	13214	11727
	a+t	2589	24622	24622	13214	13214	13214	11727
Culebra	!a	1116	85	5205	5121	5121	5121	4585
	a	1116	10415	12924	5105	5105	5105	4575
	a+t	1116	12939	12939	5105	5105	5105	4575