Westfälische Hochschule

Padding Matters – Exploring Function Detection in PE Files

04.06.2025 – CODASPY '25

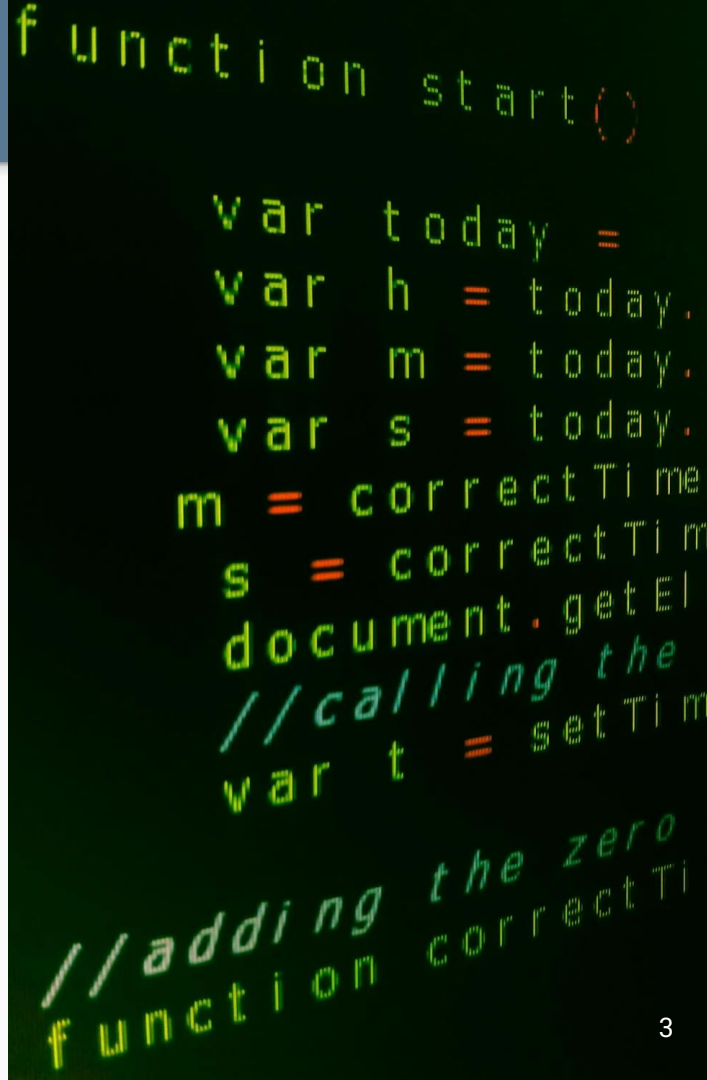
Raphael Springer, Alexander Schmitz, Artur Leinweber, Tobias Urban, Christian Dietrich
Institute for Internet Security – Westphalian University of Applied Sciences

Motivation

Motivation

- 300k+ new hash-unique malware samples daily
- automatic binary code analysis is essential
- 30 out of 61 binary similarity methods operate at the function level according to Haq and Caballero [1]

Detecting functions in binaries is a critical component of cyber defence.

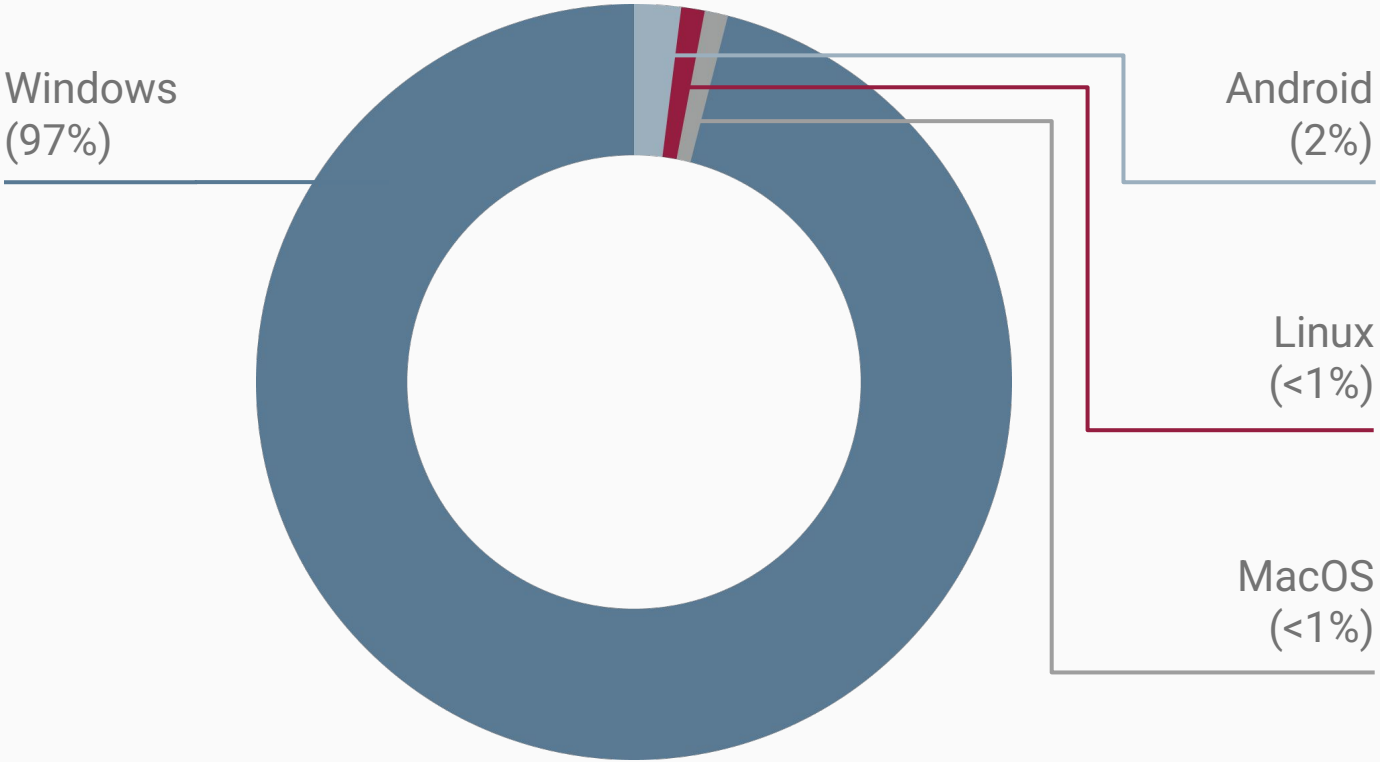


Tool/Paper	Samples		Functions	
	ELF	PE	ELF	PE
FunProbe [2]	19.09k	0	3.06m	0
DeepDi [3]	1.4k	688	n/a	n/a
Koo et al. [4]	152	0	769k	0
FETCH [5]	43	0	1.11m	0
XDA [6]	2.6k	528	n/a	n/a
Jima [7]	3.8k	0	4.91m	0
LEMNA [8]	2.1k	0	n/a	n/a
Nucleus [9]	324	152	n/a	379k
rev.ng [10]	1.9k	0	n/a	0
Andriesse et al. [11]	829	152	1.53m	379k
Shin et al. [12]	2.1k	136	589k	188k
BAP/ByteWeight [13]	2.1k	136	589k	188k
Rosenblum et al. [14]	728	443	284k	100k

Tool/Paper	Samples		Functions	
	ELF	PE	ELF	PE
FunProbe [2]	**19.09k**	0	3.06m	0
DeepDi [3]	1.4k	688	n/a	n/a
Koo et al. [4]	152	0	769k	0
FETCH [5]	43	0	1.11m	0
XDA [6]	2.6k	528	n/a	n/a
Jima [7]	3.8k	0	**4.91m**	0
LEMNA [8]	2.1k	0	n/a	n/a
Nucleus [9]	324	152	n/a	379k
rev.ng [10]	1.9k	0	n/a	0
Andriesse et al. [11]	829	152	1.53m	379k
Shin et al. [12]	2.1k	136	589k	188k
BAP/ByteWeight [13]	2.1k	136	589k	188k
Rosenblum et al. [14]	728	443	284k	100k

Tool/Paper	Samples		Functions	
	ELF	PE	ELF	PE
FunProbe [2]	19.09k	**0**	3.06m	**0**
DeepDi [3]	1.4k	688	n/a	n/a
Koo et al. [4]	152	**0**	769k	**0**
FETCH [5]	43	**0**	1.11m	**0**
XDA [6]	2.6k	528	n/a	n/a
Jima [7]	3.8k	**0**	4.91m	**0**
LEMNA [8]	2.1k	**0**	n/a	n/a
Nucleus [9]	324	152	n/a	379k
rev.ng [10]	1.9k	**0**	n/a	**0**
Andriesse et al. [11]	829	152	1.53m	379k
Shin et al. [12]	2.1k	136	589k	188k
BAP/ByteWeight [13]	2.1k	136	589k	188k
Rosenblum et al. [14]	728	443	284k	100k

Malware distribution according to AV-TEST

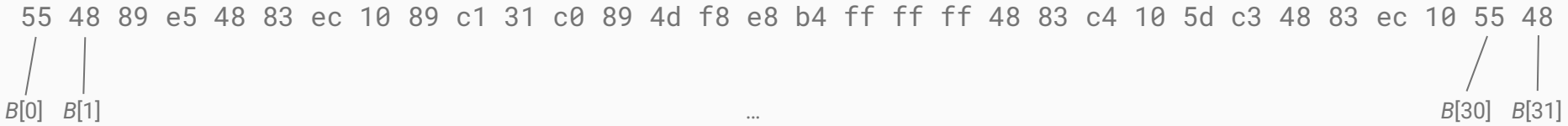


Malware distribution across operating systems; https://portal.av-atlas.org/malware

Function Detection

Identifying sequences of bytes in a compiled binary that correspond to individual functions in the original source code.

Byte Sequence *B*



Byte Sequence *B*

```
55 48 89 e5 48 83 ec 10 89 c1 31 c0 89 4d f8 e8 b4 ff ff ff 48 83 c4 10 5d c3 48 83 ec 10 55 48
```

F_1 = [*B*[0], *B*[1], *B*[2], *B*[3], *B*[4], *B*[5], *B*[6], *B*[7], *B*[11], *B*[12], *B*[13], *B*[14], *B*[15], *B*[16], *B*[17]]

Bytes belonging to function F_1

Byte Sequence *B*

```
55 48 89 e5 48 83 ec 10 89 c1 31 c0 89 4d f8 e8 b4 ff ff ff 48 83 c4 10 5d c3 48 83 ec 10 55 48
```

F_1 = [*B*[0], *B*[1], *B*[2], *B*[3], *B*[4], *B*[5], *B*[6], *B*[7], *B*[11], *B*[12], *B*[13], *B*[14], *B*[15], *B*[16], *B*[17]]

F_2 = [*B*[8], *B*[9], *B*[10], *B*[20], *B*[21], *B*[22], *B*[23], *B*[24], *B*[25], *B*[26]]

Bytes belonging to function F_2

Byte Sequence *B*

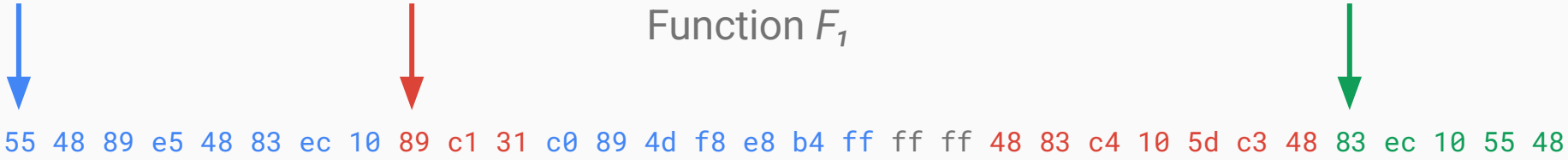
```
55 48 89 e5 48 83 ec 10 89 c1 31 c0 89 4d f8 e8 b4 ff ff ff 48 83 c4 10 5d c3 48 83 ec 10 55 48
```

F_1 = [*B*[0], *B*[1], *B*[2], *B*[3], *B*[4], *B*[5], *B*[6], *B*[7], *B*[11], *B*[12], *B*[13], *B*[14], *B*[15], *B*[16], *B*[17]]

F_2 = [*B*[8], *B*[9], *B*[10], *B*[20], *B*[21], *B*[22], *B*[23], *B*[24], *B*[25], *B*[26]]

F_3 = [*B*[27], *B*[28], *B*[29], *B*[30], *B*[31]]

Bytes belonging to function F_3



F_1 = [B[0], B[1], B[2], B[3], B[4], B[5], B[6], B[7], B[11], B[12], B[13], B[14], B[15], B[16], B[17]]

Function start:	B[0]
Function end:	B[17]
Function boundary:	(B[0], B[17])

Pattern Recognition

- characteristic byte sequences form signatures
- prologue patterns for function starts
- epilogue patterns for function ends

Prologue

```
55              PUSH RBP
4889e5          MOV RBP,RSP
4883ec10        SUB RSP,0x10
```

554889e54883ec10

Pattern Recognition

- recognizing prologue and epilogue
- prologue pattern for function starts
- epilogue pattern for function ends
- normalization of immediate values

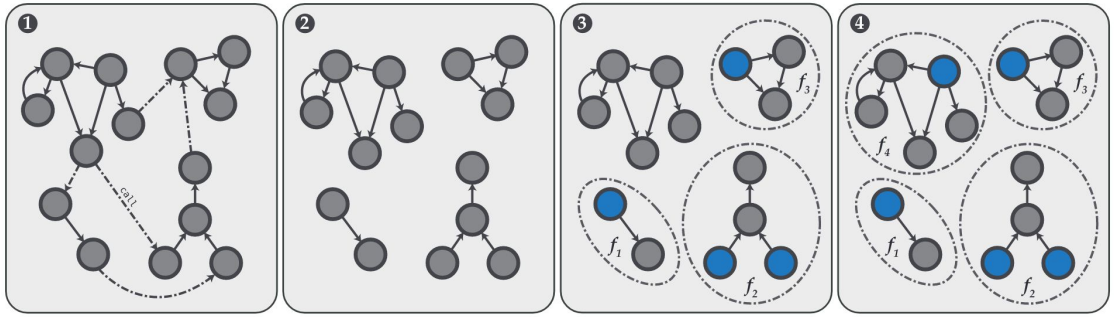
Prologue

```
55              PUSH RBP
4889e5          MOV RBP,RSP
4883ec10        SUB RSP,0x10
```

554889e54883ec??

Static Analysis

- disassemble
- construct control-flow graph (CFG)
- find call targets
- recognize and analyze jump tables
- some tools incorporate machine learning



Function Detection of Nucleus [9]

FuncPEval

FuncPEval

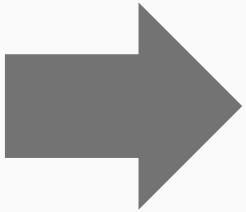
Four samples

Chromium

- x86
- x64

Conti Ransomware

- x86
- x64



1,092,820 functions

FuncPEval – Chromium

- open source browser Chromium version 109
- compiled and linked with LLVM Clang
  - Ox, Os, Oy, O1, Ot, and O2 optimization levels
- public snapshot published by Google
  - `chrome.dll`
  - `chrome.dll.pdb`

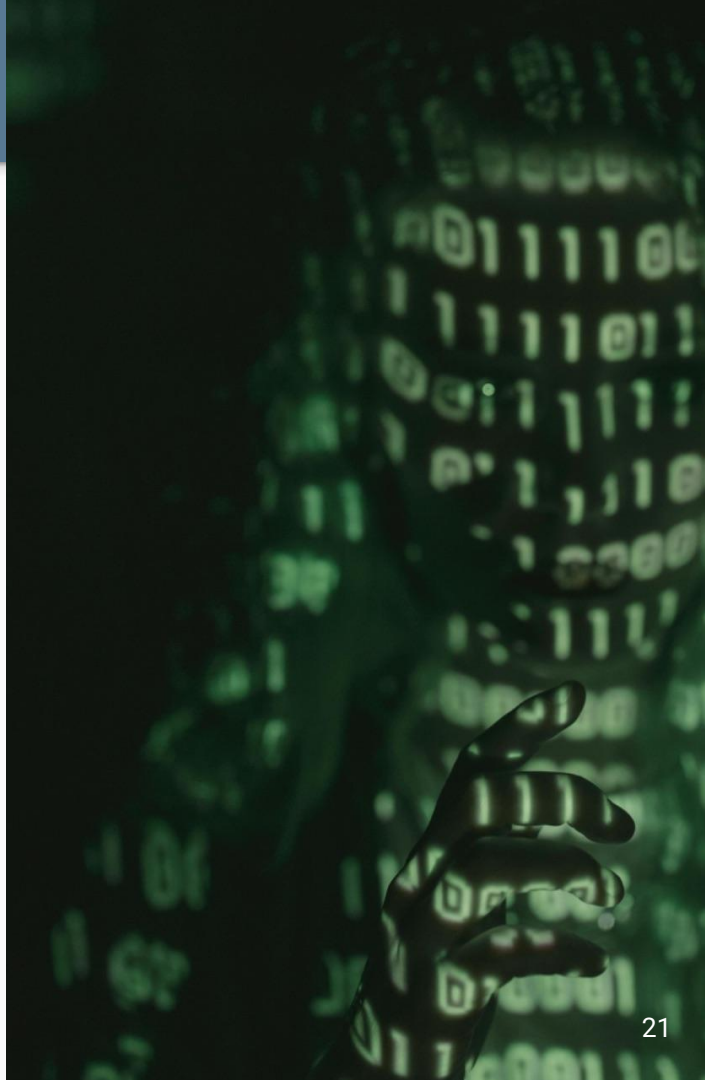
http://commondatastorage.googleapis.com/chromium-browser-snapshots/index.html?prefix=Win_x64/1069922/

http://commondatastorage.googleapis.com/chromium-browser-snapshots/index.html?prefix=Win/1069956/

chromium

FuncPEval – Conti

- Conti Ransomware version 3
- source code leaked in early 2022
- `cryptor.exe`
  - component that encrypts machine
- compiled and linked using Visual Studio 2022



Dataset	Functions		
	total	unique	normalized
Chromium x64	542,902	536,182 (99%)	315,745 (58%)
Conti x64	662	659 (99%)	450 (68%)
BAP PE [13]	94,548	65,733 (70%)	18,169 (19%)

Evaluation

Evaluation – Tools

Industry Standards

- IDA Pro 7.7
- Ghidra 10.0.4

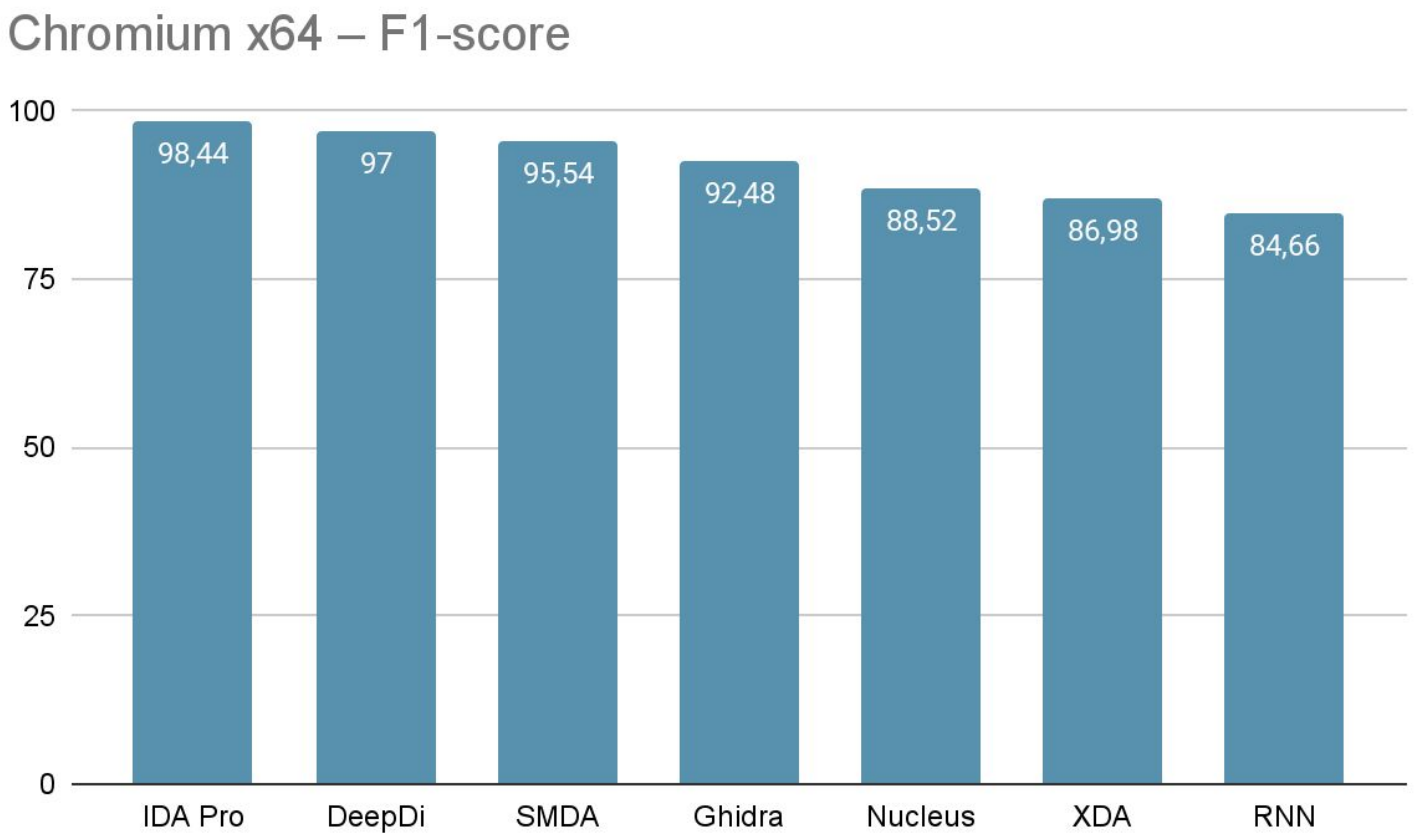
Scientific Publications

Non-ML-based

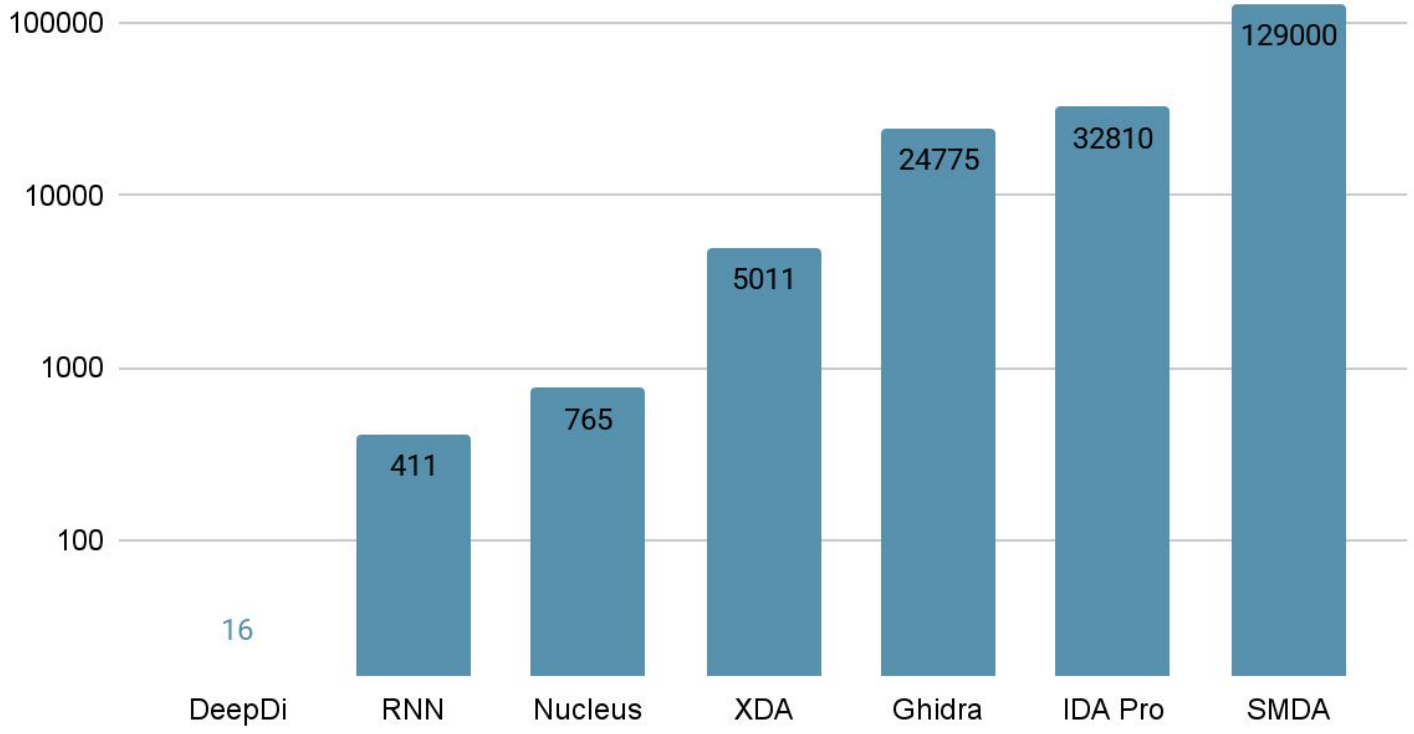
- Nucleus [9]
- SMDA [15]

ML-based

- RNN [12]
- XDA [6]
- DeepDi [3]


Chromium x64 – F1-score
100
75
50
25
0
98,44
97
95,54
92,48
88,52
86,98
84,66
IDA Pro
DeepDi
SMDA
Ghidra
Nucleus
XDA
RNN

Chromium x64 – Execution time in seconds



Padding

Padding – Motivation

What features do ML-based tools learn?

How do they generalize over different compiler versions?

Padding

- compiler align functions at multiples of 16
- padding fills bytes in between
  - `0xCC int3`
  - `0x90 NOP`
- padding not part of control flow
  - can be changed arbitrarily

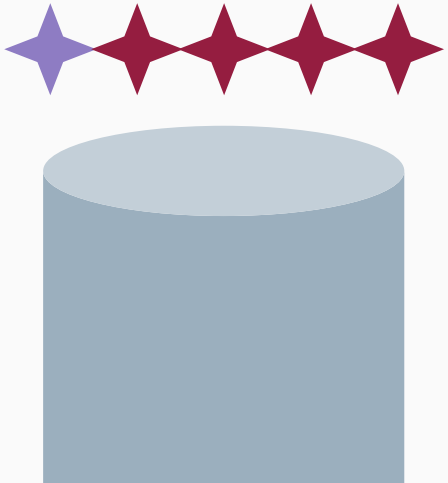
```
140018fd7 49 8b c1        MOV         RAX,R9
140018fda c3              RET
140018fdb cc              ??          CCh
140018fdc cc              ??          CCh
140018fdd cc              ??          CCh
140018fde cc              ??          CCh
140018fdf cc              ??          CCh

                          ****************************
                          *                          FU
                          ****************************
                          byte * __fastcall FUN_140018f
                             assume GS_OFFSET = 0xff000
     byte *                  RAX:8          <RETURN>
     char *                  RCX:8          param_1
                          FUN_140018fe0
140018fe0 0f b6 01          MOVZX       EAX,byte ptr
140018fe3 4c 8d 41 01       LEA         R8,[param_1 +
140018fe7 84 c0             TEST        AL,AL
140018fe9 75 64             JNZ         LAB_14001904f
```

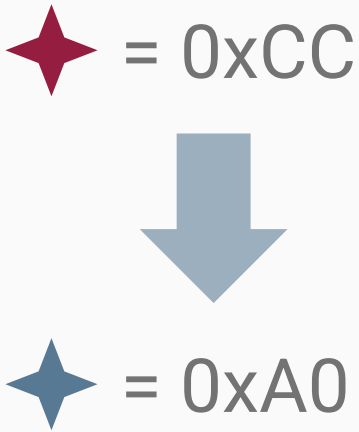
Padding – Randomization

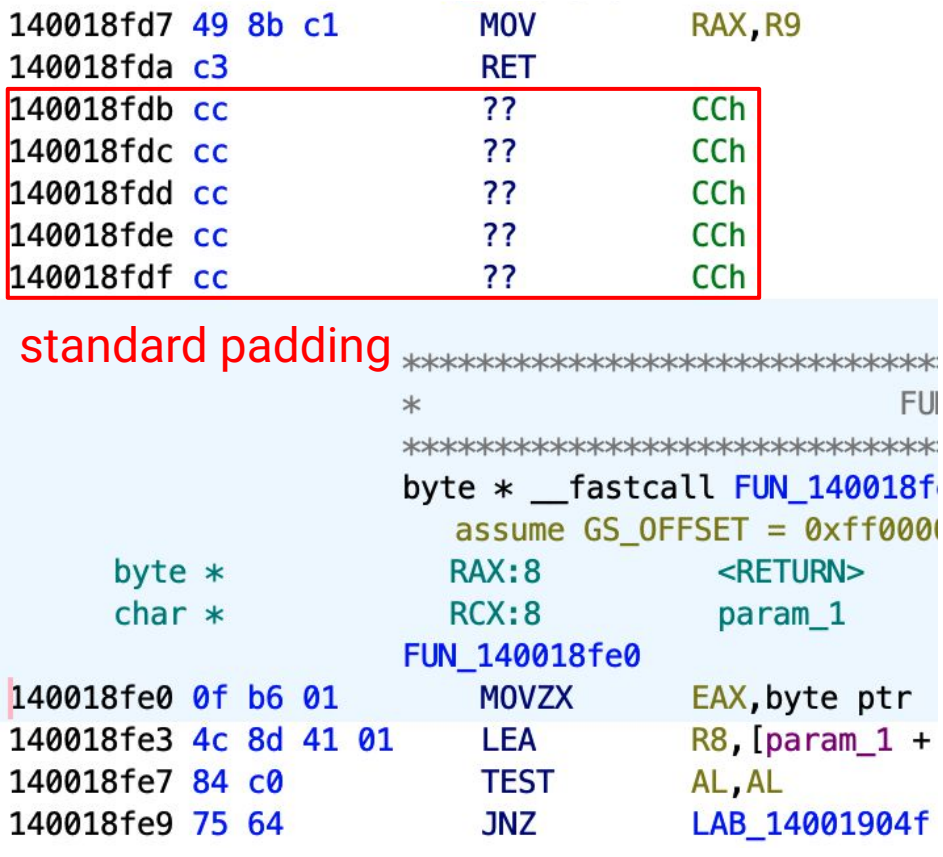
1 Collect up to 20 bytes before the beginning of the function.

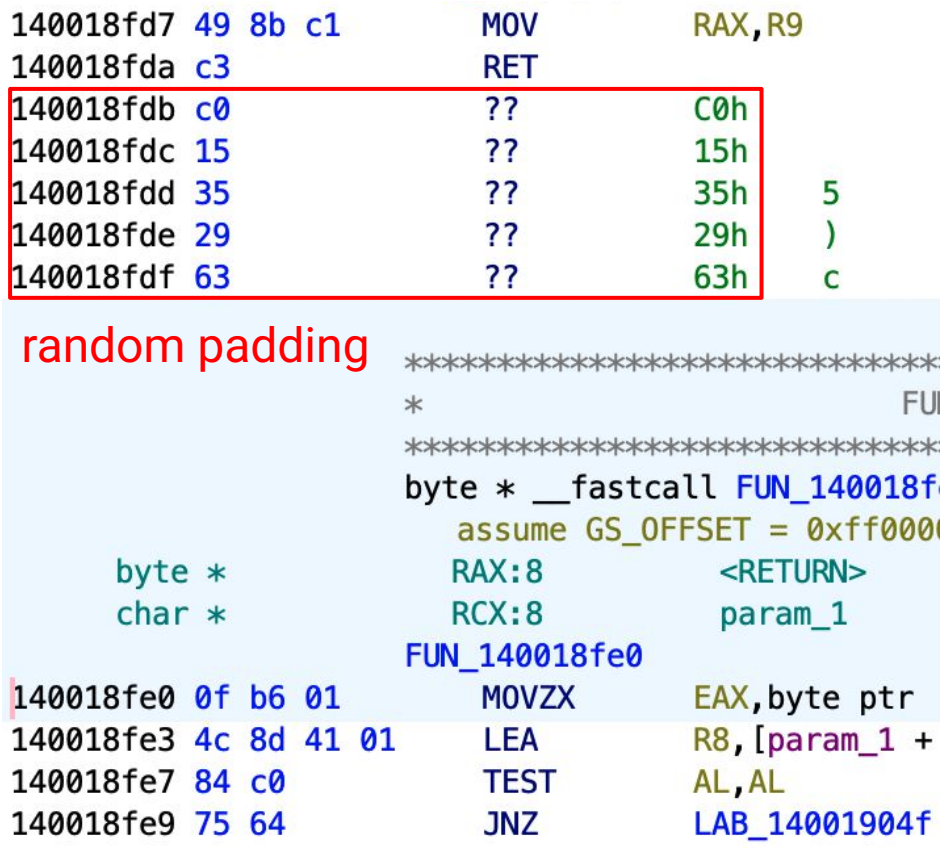
2 Consider only those bytes that do not belong to a preceding function.

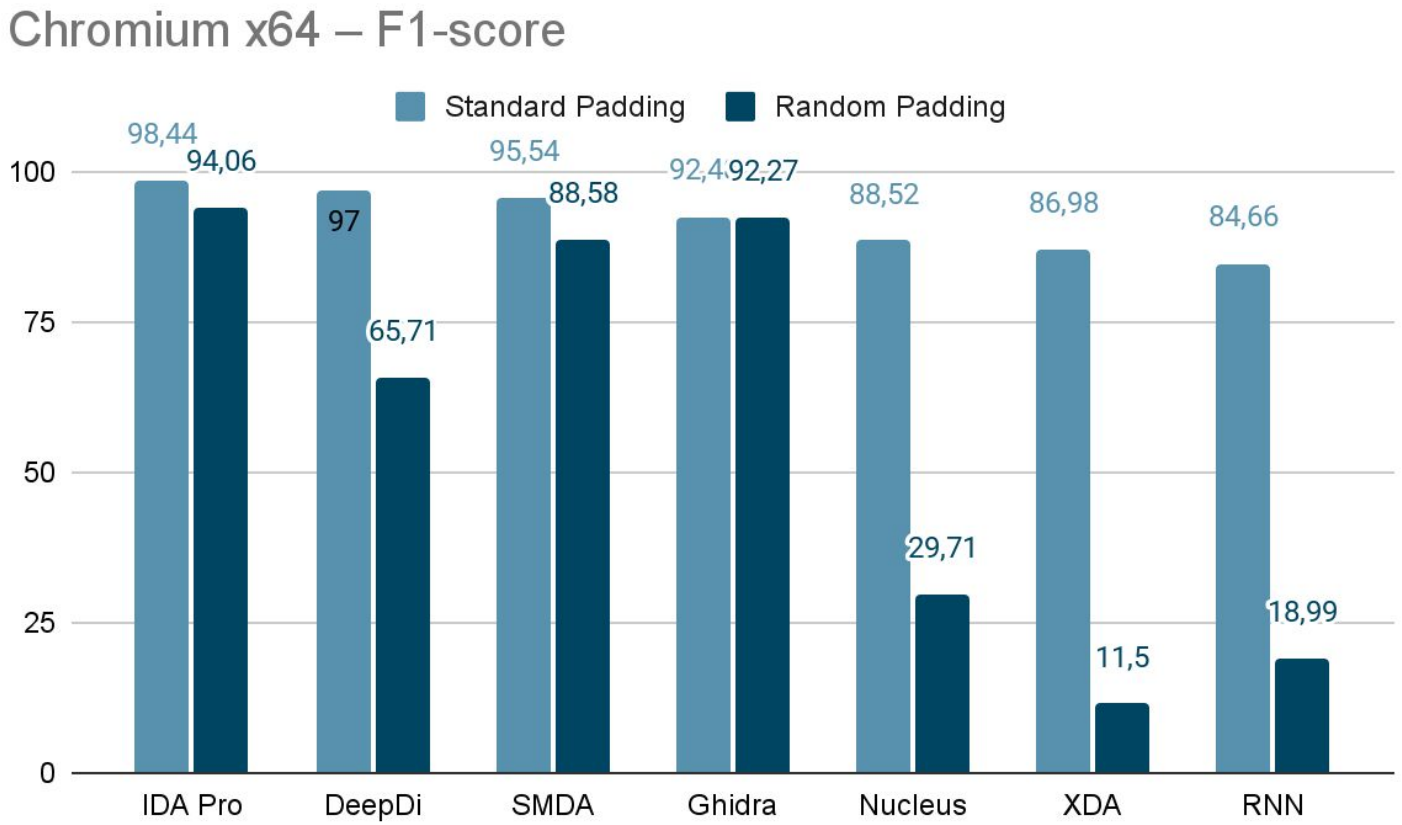
3 Replace each padding byte of value `0xCC` with a random arbitrary byte value.






140018fd7 49 8b c1 MOV RAX,R9
140018fda c3 RET
140018fdb cc ?? CCh
140018fdc cc ?? CCh
140018fdd cc ?? CCh
140018fde cc ?? CCh
140018fdf cc ?? CCh
standard padding
byte * __fastcall FUN_140018f
assume GS_OFFSET = 0xff000
byte * RAX:8 <RETURN>
char * RCX:8 param_1
FUN_140018fe0
140018fe0 0f b6 01 MOVZX EAX,byte ptr
140018fe3 4c 8d 41 01 LEA R8,[param_1 +
140018fe7 84 c0 TEST AL,AL
140018fe9 75 64 JNZ LAB_14001904f


140018fd7 49 8b c1 MOV RAX,R9
140018fda c3 RET
140018fdb c0 ?? C0h
140018fdc 15 ?? 15h
140018fdd 35 ?? 35h 5
140018fde 29 ?? 29h)
140018fdf 63 ?? 63h c
random padding
byte * __fastcall FUN_140018f
assume GS_OFFSET = 0xff000
byte * RAX:8 <RETURN>
char * RCX:8 param_1
FUN_140018fe0
140018fe0 0f b6 01 MOVZX EAX,byte ptr
140018fe3 4c 8d 41 01 LEA R8,[param_1 +
140018fe7 84 c0 TEST AL,AL
140018fe9 75 64 JNZ LAB_14001904f


Chromium x64 – F1-score
Standard Padding
Random Padding
100
75
50
25
0
98,44
94,06
97
65,71
95,54
88,58
92,4
92,27
88,52
29,71
86,98
11,5
84,66
18,99
IDA Pro
DeepDi
SMDA
Ghidra
Nucleus
XDA
RNN

Conclusion

Conclusion

- new PE dataset spanning 1,092,820 functions
- evaluated seven tools
- IDA Pro highest F1-score
- DeepDi fastest execution time
- ML-based tools sensitive to padding modification

- Artifacts
  - FuncPEval
  - tooling for ground truth extraction
  - all trained models and training data
  - https://github.com/internet-sicherheit/Padding-Matters---Exploring-Function-Detection-in-PE-Files

References

[1] Haq, Irfan Ul, and Juan Caballero. "A survey of binary code similarity." Acm computing surveys (csur) 54.3 (2021): 1-38.
[2] Kim, Soomin, Hyungseok Kim, and Sang Kil Cha. "FunProbe: probing functions from binary code through probabilistic analysis." Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2023.
[3] Yu, Sheng, et al. "{DeepDi}: Learning a relational graph convolutional network model on instructions for fast and accurate disassembly." 31st USENIX Security Symposium (USENIX Security 22). 2022.
[4] Koo, Hyungjoon, Soyeon Park, and Taesoo Kim. "A look back on a function identification problem." Proceedings of the 37th Annual Computer Security Applications Conference. 2021.
[5] Pang, Chengbin, et al. "Towards optimal use of exception handling information for function detection." 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2021.
[6] Pei, Kexin, et al. "Xda: Accurate, robust disassembly with transfer learning." arXiv preprint arXiv:2010.00770 (2020).
[7] Alves-Foss, Jim, and Jia Song. "Function boundary detection in stripped binaries." Proceedings of the 35th Annual Computer Security Applications Conference. 2019.
[8] Guo, Wenbo, et al. "Lemna: Explaining deep learning based security applications." proceedings of the 2018 ACM SIGSAC conference on computer and communications security. 2018.

References

[9] Andriesse, Dennis, Asia Slowinska, and Herbert Bos. "Compiler-agnostic function detection in binaries." 2017 IEEE European symposium on security and privacy (EuroS&P). IEEE, 2017.
[10] Di Federico, Alessandro, Mathias Payer, and Giovanni Agosta. "rev. ng: a unified binary analysis framework to recover CFGs and function boundaries." Proceedings of the 26th International Conference on Compiler Construction. 2017.
[11] Andriesse, Dennis, et al. "An {In-Depth} analysis of disassembly on {Full-Scale} x86/x64 binaries." 25th USENIX security symposium (USENIX security 16). 2016.
[12] Shin, Eui Chul Richard, Dawn Song, and Reza Moazzezi. "Recognizing functions in binaries with neural networks." 24th USENIX security symposium (USENIX Security 15). 2015.
[13] Bao, Tiffany, et al. "{BYTEWEIGHT}: Learning to recognize functions in binary code." 23rd USENIX Security Symposium (USENIX Security 14). 2014.
[14] Rosenblum, Nathan E., et al. "Learning to Analyze Binary Computer Code." AAAI. 2008.
[15] Daniel Plohmann. SMDA. https://github.com/danielplohmann/smda, 2024.

Thank you.

Raphael Springer, Alexander Schmitz,
Artur Leinweber, Tobias Urban, Christian Dietrich

https://threatlab.if-is.net

With funding from the:

Federal Ministry
of Research, Technology
and Space

grants 13FH101KB1
and 16KIS1746

nicos
xantaro group

if(is)
internet-sicherheit.

Westfälische
Hochschule