[35] Rosenblum, N.E., Zhu, X., Miller, B.P., Hunt, K.: Learning to analyze binary computer code. In: AAAI. pp. 798–804 (2008)

[36] Shirani, P., Wang, L., Debbabi, M.: Binshape: Scalable and robust binary library function identification using function shape. In: Detection of Intrusions and Malware, and Vulnerability Assessment (2017)

[37] Tabish, S.M., Shafiq, M.Z., Farooq, M.: Malware detection using statistical analysis of byte-level file content. In: Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics. p. 23–31. CSI-KDD '09, New York, NY, USA (2009). https://doi.org/10.1145/1599272.1599278

[38] Tian, Z., Huang, Y., Xie, B., Chen, Y., Chen, L., Wu, D.: Fine-grained compiler identification with sequence-oriented neural modeling. IEEE Access (2021)

[39] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)

[40] Wang, R., Shoshitaishvili, Y., Bianchi, A., Machiry, A., Grosen, J., Grosen, P., Kruegel, C., Vigna, G.: Ramblr: Making reassembly great again. In: NDSS (2017)

[41] Wang, S., Wang, P., Wu, D.: Reassembleable disassembling. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 627–642. USENIX Association, Washington, D.C. (Aug 2015), https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/wang-shuai

[42] Xu, X., Liu, C., Feng, Q., Yin, H., Song, L., Song, D.: Neural network-based graph embedding for cross-platform binary code similarity detection. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 363–376 (2017)

[43] Yang, S., Shi, Z., Zhang, G., Li, M., Ma, Y., Sun, L.: Understand code style: Efficient cnn-based compiler optimization recognition system. In: ICC 2019 - 2019 IEEE International Conference on Communications (ICC) (2019)

9 Types of Alignment Padding

The code generation and placement of alignment padding in the binary is an indication of different strategies and policies followed by the compilers. We present eight scenarios in which the strategies followed by compilers affect the amount of alignment padding emitted in the binary.

(1) *Data interleaved in code.* Compilers interleave data within the code in the `.text` section. The data interleaved is called inline data. Therefore, the compilers interleave more inline data at higher optimization levels for increased performance. The inline data is represented by the `DB` instruction in x86–64 code.

(2) *Import functions.* Compilers reduce the number of import functions at higher optimization levels (i.e. O2 and Ox) for enhanced performance. This omits the

alignment padding emitted before the branch instruction, respectively.

(3) *Exception Handling Functions.* Clang-cl and MSVC compilers at higher optimization levels prefer to reduce the exception handling functions as these functions are only called at runtime. On the other hand, ICC prefers to reduce the size of the binary at O1 optimization level. Therefore, ICC emits less code for EH function information in order to reduce the size of the jump table-data generated [17].

(4) *Intrinsic functions.* Intrinsic functions are built-in by the compilers and emitted to achieve performance. These functions are inline, they neither allocate stack space, nor do they call other functions [24]. Therefore, the optimizing compilers like MSVC, ICC and Clang-cl expand the intrinsic functions differently depending on the context and arguments of the call. Hence, the padding inserted by the compilers to align intrinsic function vary along with the level of optimization [28].

(5) *Function-entry alignment.* In the `.text` section of the binary, before a function executes (i.e. subroutine call or an EH function), the data and code of the function is aligned for efficient execution by the processor [14].

(6) *Common Runtime (CRT) Routines.* CRT routines like `strncpy`, `memcpy`, `memset`, `memmove` etc. are hand-coded assembly routines particular to the compiler architecture. They are being added by the optimizing compilers like MSVC, ICC, Clang-cl and MinGW to align the source and destination addresses. This is to add efficiency for large move operations so the cost of unaligned accesses are minimized [28].

(7) *End-of-segment alignment.* The compilers on Microsoft platform pads the end of the section with zero bytes or data bytes to fill the remaining space in order to align the next section starting at 512 byte (aligned) address. The end-of-section alignment is only specific to Microsoft Windows platform because every section of the Windows executable is aligned to the page size (i.e., 512 bytes), when loaded in memory. The end-of-section alignment is, however, not applicable to Linux platform binaries [28]. We observed this kind of alignment in all of the four compilers under study on Windows.

(8) *Vector operations alignment.* The ICC and Clang-cl compiler mostly utilizes 128-bit multimedia operands with MMX and SSE (XMM) instructions. XMMWORD is one such instruction that is used for high-speed numerical and vector operations. Therefore, all the vector data and instructions are forced to be aligned at 16-byte address. Intel embeds increased alignment padding at higher optimization levels [17]. We illustrate the differences emitted by the compilers against each compiler and optimization level as shown in Figure 3. We selected four binaries from our dataset, namely, *valgrind*, *curl*, *zlib* and *tftpd*. We compiled these binaries with three different compilers and at four optimization levels. Moreover, we extracted the proportion of alignment padding in each compiled binary at different optimization levels. In Figure 3, along the x-axis are the three compilers (i.e. MSVC, ICC and Clang-cl) with four optimization levels settings (i.e., Od/O0, O1, O2 and Ox/O3). Along y-axis is the proportion of the alignment padding in each of the four binaries.

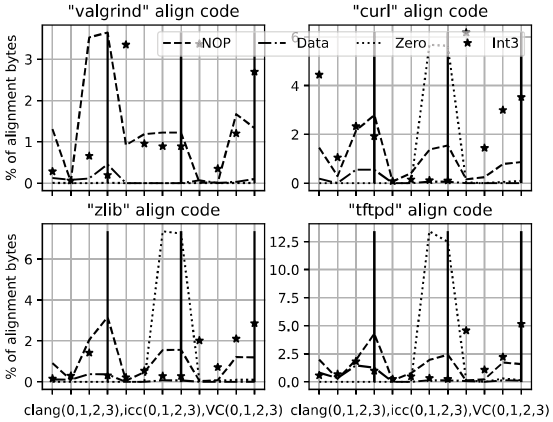


Fig. 3 – Differences of Alignment Padding in fours applications (i.e., *valgrind*, *curl*, *zlib* and *tftpd*)

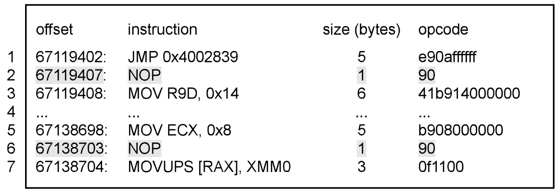
We observed a consistent pattern of alignment padding for the compilers and optimization levels except for the NOP in Clang-cl compiler for the *valgrind* application. The increase in NOP instructions is due to the increased number of functions in the binary at Od level. Moreover, the consistent increase in INT 3 in Clang-cl refers to the increasing number of returning functions at increasing optimization levels. Similarly, before the beginning of a function, the compilers emit alignment paddings to align the stack for function call.

As illustrated in Figure 3, ICC shows maximum number of Zero padding at O2 and O3/Ox optimization levels. This is because ICC performs function inlining, code inlining and loop unrolling at higher optimization levels. Therefore, it achieves maximum alignment padding at higher optimization levels. On the other hand, ICC prefers reduced code size at O1 optimization level and does not optimize instructions at Od, respectively.

MSVC, on the other hand, emits the highest number of INT 3 alignment padding in the `.text` section. This makes the compiler prevents the erroneous execution of the program by transferring control to debug exception handler, in case of an exception.

We illustrate some examples to show the use of alignment padding in different compilers. Listings 9 to 12 shows the enforcement of NOP padding alignment in four compilers (i.e., MSVC, ICC, Clang-cl and MinGW) at O2 optimization level.

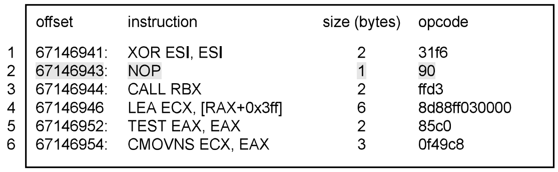
In Listing 9 at line 2, MSVC uses NOP instruction before passing arguments to the function. It prepares the arguments before the function call by moving the required value 14 in the register R9D at line 3. Similarly, at line 6, MSVC uses NOP to align MOVUPS instruction that moves 512 bits of packed single-precision floating-point value from XMM register [18] to memory location pointed to by RAX register.



```
     offset      instruction               size (bytes)   opcode

 1   67119402:   JMP 0x4002839                  5          e90affffff
 2   67119407:   NOP                            1          90
 3   67119408:   MOV R9D, 0x14                  6          41b914000000
 4   ...         ...                           ...         ...
 5   67138698:   MOV ECX, 0x8                   5          b908000000
 6   67138703:   NOP                            1          90
 7   67138704:   MOVUPS [RAX], XMM0             3          0f1100
```

Listing 9 – Align `MOV` operation by MSVC compiler at O2 optimization level

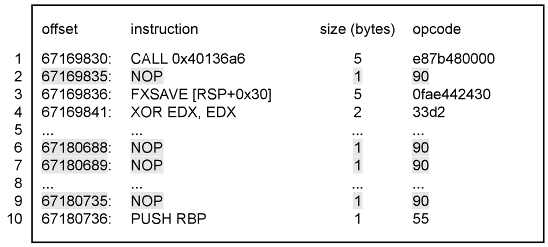


```
     offset      instruction               size (bytes)   opcode

 1   67146941:   XOR ESI, ESI                   2          31f6
 2   67146943:   NOP                            1          90
 3   67146944:   CALL RBX                       2          ffd3
 4   67146946    LEA ECX, [RAX+0x3ff]           6          8d88ff030000
 5   67146952:   TEST EAX, EAX                  2          85c0
 6   67146954:   CMOVNS ECX, EAX                3          0f49c8
```

Listing 10 – Align `CALL` operation by Clang-cl compiler at O2 optimization level

Likewise, in Listing 10, at line 2, we found that Clang-cl compiler uses NOP instruction to align the indirect call pointed to by the address stored in the register RBX. Moreover, at line 6, the padding is used to align the CMOVNS instruction. CMOVNS moves source operand to destination operand, if the sign flag (SF) is not set (i.e. SF=0) [17].

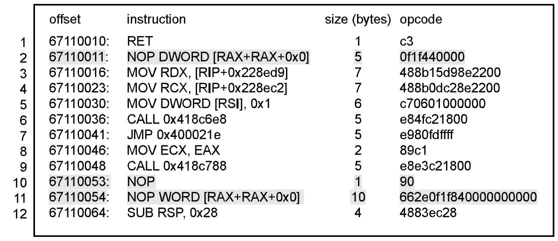


```
      offset      instruction              size (bytes)   opcode

  1   67169830:   CALL 0x40136a6                 5         e87b480000
  2   67169835:   NOP                            1         90
  3   67169836:   FXSAVE [RSP+0x30]              5         0fae442430
  4   67169841:   XOR EDX, EDX                   2         33d2
  5   ...         ...                           ...        ...
  6   67180688:   NOP                            1         90
  7   67180689:   NOP                            1         90
  8   ...         ...                           ...        ...
  9   67180735:   NOP                            1         90
 10   67180736:   PUSH RBP                       1         55
```

Listing 11 – Align function start in ICC compiler at O2 optimization level

On the other hand, ICC utilizes a number of 48 single-byte NOP instructions prior to a function call to align the stack at 16-byte memory address as shown in Listing 11, line 6 – 9. Line 10 is the starts the function prolog and PUSH instruction moves the value of the old base pointer in EBP on to the stack. Furthermore, in Listing 11, at line 2, the ICC compiler emits NOP instructions to align the FXSAVE operation at O2 optimization level. FXSAVE [17] saves the current state of floating point unit (FPU) register to a 512-byte memory location.



```
    offset      instruction                    size (bytes)  opcode

1   67110010:   RET                                 1        c3
2   67110011:   NOP DWORD [RAX+RAX+0x0]             5        0f1f440000
3   67110016:   MOV RDX, [RIP+0x228ed9]             7        488b15d98e2200
4   67110023:   MOV RCX, [RIP+0x228ec2]             7        488b0dc28e2200
5   67110030:   MOV DWORD [RSI], 0x1                6        c70601000000
6   67110036:   CALL 0x418c6e8                      5        e84fc21800
7   67110041:   JMP 0x400021e                       5        e980fdffff
8   67110046:   MOV ECX, EAX                        2        89c1
9   67110048    CALL 0x418c788                      5        e8e3c21800
10  67110053:   NOP                                 1        90
11  67110054:   NOP WORD [RAX+RAX+0x0]             10        662e0f1f840000000000
12  67110064:   SUB RSP, 0x28                       4        4883ec28
```

Listing 12 – Align function entry in MinGW compiler at O2 optimization level

Similarly, the fourth compiler in our experiments which is MinGW (Minimalist GNU for Windows Subsystem) also emits alignment paddings at the entry of function to align for the function code and data. In Listing 12, MinGW compiler emits 5 bytes of multi-byte NOP at line 2 to align the data before preparing arguments at the subsequent line for the function call at line 9. Similarly, at line 10–11, the compiler emits NOP bytes to align the data for the next function stack frame setup at line 12.

Hence, the above examples show us that the alignment padding differs based on the compilers and compiler optimization levels.

10 Dataset

Table 15 shows the description of some of the projects used in our dataset. As we presented in Section 5.1, we selected our dataset from GitHub based on their popularity (i.e., the number of stars).

We included all the popular and real life programs in the our collection of dataset for experimentation of compiler provenance.

Table 15 – The selected C/C++ projects from Github in our dataset

Project	Description
abseil-cpp	a C++ standard library (compliant to C++11) designed to augment the C++ code.
audioFile	a C++ library for reading and writing audio files.
ogg-vorbis	an audio coding format and software reference encoder/decoder for lossy audio compression by xiph.org foundation.
bamboo	an elixir family of projects for popular mail delivery services like Mandrill, Mailgun, and SendGrid.
bear	a tool to generate a compilation database for clang tooling.
benchmark	a micro-benchmark support library.
c-ares	an asynchronous resolver library to perform multiple DNS queries in parallel.
json	json library for modern cpp.
cmake	a cross-platform, open-source build system generator.
curl	a powerful command-line tool and library for transferring data with url syntax, supporting dict, file, ftp, ftps, gopher, gophers, http, https, imap, imaps, ldap, ldaps, mqtt, pop3, pop3s, rtmp, rtmps, rtsp, scp, sftp, smb, smbs, smtp, smtps, telnet, and tftp.
directx	a collection of application programming interfaces for handling multimedia and game programming tasks.
dlib	a modern C++ toolkit containing machine learning algorithms and tools for creating complex real-world data analysis software.
doxygen	a standard tool for generating documentation from annotated C++ sources, it also supports C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, and D programming language.
eigen	a C++ template library for linear algebra, matrices, vectors, numerical solvers, and related algorithms.
gflags	C++ library that implements command-line flags processing for standard types such as strings.
glm	openGL Mathematics (GLM) is a C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specifications.
glog	google logging (glog) is the C++ implementation of the Google logging module.
gperftools	a collection of a high-performance multi-threaded malloc implementation, including performance analysis tools.
hdf5	high-performance data management and storage suite. A suite of library to manage, process, and store heterogeneous data for fast I/O processing and storage.
juce	cross-platform C++ application framework for rapidly developing high-quality desktop and mobile applications, including VST, AU (and AUv3), RTAS and AAX audio plugins.
libevent	a library that provides asynchronous event notifications.
llvm	a set of compiler and toolchain technologies, used to develop a front end for any programming language and a back end for any instruction set architecture (ISA).
onednn	oneAPI deep neural network library (oneDNN) is used to measure the performance of deep learning applications. The library is optimized for Intel Architecture Processors, Intel Processor Graphics, and Xe architecture-based Graphics.
opencl	opencl (Open Computing Language) is a framework for writing programs that execute across heterogeneous platforms CPUs, GPUs, DSPs, FPGAs) or hardware accelerators.
openssl	a software library for applications to secure communications over computer networks against eavesdropping.
Microsoft C++ standard library	Microsoft's implementation of the C++ Standard Library (also known as the STL), which ships as part of the MSVC tool-set and the Visual Studio IDE.
sqlite	a relational database management system contained in a C library. Unlike, client-server database engine, it is embedded into the end program.
vtk	an image processing, 3D graphics, volume rendering, and visualization toolkit.
zlib	data compression library and an abstraction of the deflate compression algorithm used in gzip file compression program.