

Analiza Malware Laboratorium nr 5

Raport – Nikodem Jakubowski

Laboratorium 5.1

Przeprowadź analizę pliku Lab06-01.exe za pomocą programu OllyDbg i odpowiedz na poniższe pytania.

Na początek analiza pliku.

Wykonując kilka jumps od maina dochodzimy do poniższego elementu.

```
00402410 55      PUSH EBP
00402411 8BEC    MOV EBP,ESP
00402413 81EC    SUB ESP,208
00402419 53      PUSH EBX
0040241A 56      PUSH ESI
0040241B 57      PUSH EDI
0040241C 68 04010000  PUSH 104
00402421 8D85    LEA EAX,DWORD PTR SS:[EBP-208]
00402427 50      PUSH EAX
00402428 6A 00    PUSH 0
0040242A FF15    CALL DWORD PTR DS:[<&KERNEL32.GetModuleFileNameA>]
00402430 68 04010000  PUSH 104
00402435 8D8D    LEA ECX,DWORD PTR SS:[EBP-208]
00402438 51      PUSH ECX
0040243C 8D95    LEA EDX,DWORD PTR SS:[EBP-208]
00402442 52      PUSH EDX
00402443 FF15    CALL DWORD PTR DS:[<&KERNEL32.GetShortPathNameA>]
00402449 BF DCC04000  MOV EDI,Lab06-01.0040C0DC
0040244E 8D95    LEA EDX,DWORD PTR SS:[EBP-104]
00402454 83C9    OR ECX,FFFFFFFF
00402457 33C0    XOR EAX,EAX
00402459 F2:RE   REPNE SCAS BYTE PTR ES:[EDI]
0040245B F7D1    NOT ECX
0040245D 2BF9    SUB EDI,ECX
0040245F 8BF7    MOV ESI,EDI
00402461 8BC1    MOV EAX,ECX
00402463 8BFA    MOV EDI,EDX
00402465 C1E9    SHR ECX,2
00402468 F3:A5   REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0040246A 8BC8    MOV ECX,EAX
0040246C 83E1    AND ECX,3
0040246F F3:A4   REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00402471 8D8D    LEA EDI,DWORD PTR SS:[EBP-208]
00402477 8D95    LEA EDX,DWORD PTR SS:[EBP-104]
0040247D 83C9    OR ECX,FFFFFFFF
00402480 33C0    XOR EAX,EAX
00402482 F2:RE   REPNE SCAS BYTE PTR ES:[EDI]
00402484 F7D1    NOT ECX
00402486 2BF9    SUB EDI,ECX
00402488 8BF7    MOV ESI,EDI
0040248A 8BD9    MOV EBX,ECX
0040248C 8BFA    MOV EDI,EDX
0040248E 83C9    OR ECX,FFFFFFFF
00402491 33C0    XOR EAX,EAX
00402493 F2:RE   REPNE SCAS BYTE PTR ES:[EDI]
00402495 83C7    ADD EDI,-1
00402498 8BCB    MOV ECX,EBX
0040249A C1E9    SHR ECX,2
0040249D F3:A5   REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0040249F 8BCB    MOV ECX,EBX
004024A1 83E1    AND ECX,3
004024A4 F3:A4   REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004024A6 BF D4C04000  MOV EDI,Lab06-01.0040C0D4
004024AB 8D95    LEA EDX,DWORD PTR SS:[EBP-104]
004024B1 83C9    OR ECX,FFFFFFFF
004024B4 33C0    XOR EAX,EAX
004024B6 F2:RE   REPNE SCAS BYTE PTR ES:[EDI]
004024B8 F7D1    NOT ECX
Local calls from 00402B13, 00402B3A, 00402BBD, 00402C45, 00402CCF, 00402D6A, 00402D71
```

Później program wykonuje coś w shellu za pomocą cmd.

```
004024D8 F3:A4   REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004024DA 6A 00    PUSH 0
004024DC 6A 00    PUSH 0
004024DE 8D85    LEA EAX,DWORD PTR SS:[EBP-104]
004024E4 50      PUSH EAX
004024E5 68 CCC04000  PUSH Lab06-01.0040C0CC
004024EA 6A 00    PUSH 0
004024EC 6A 00    PUSH 0
004024EE FF15    CALL DWORD PTR DS:[<&SHELL32.ShellExecuteA>]
004024F4 6A 00    PUSH 0
```

```
IsShown = 0
DefDir = NULL
Parameters
FileName = "cmd.exe"
Operation = NULL
hWnd = NULL
ShellExecuteA
```

Następnie dokonuje własnej terminacji.

00402DD5	75 11	JNZ SHORT Lab06-01.00402DE8	
00402DD7	FF7424 08	PUSH DWORD PTR SS:[ESP+8]	
00402DD8	FF15 A0B04000	CALL DWORD PTR DS:[<&KERNEL32.GetCurrentProcess>]	ExitCode
00402DE1	50	PUSH EAX	GetCurrentProcess
00402DE2	FF15 9CB04000	CALL DWORD PTR DS:[<&KERNEL32.TerminateProcess>]	hProcess
00402DE8	837C24 0C 00	CMP DWORD PTR SS:[ESP+C],0	TerminateProcess
00402DED	53	PUSH EBX	

Podając Isoowany argument i podążając ścieżką funkcji 402B1D dochodzimy do podrutyny 402510 gdzie mamy interesującą rzecz. Dochodzi do skanowania argumentu i sprawdzenia czy jest długości 4.

00402510	55	PUSH EBP	
00402511	8BEC	MOV EBP,ESP	
00402513	51	PUSH ECX	
00402514	57	PUSH EDI	
00402515	8B7D 08	MOV EDI,DWORD PTR SS:[EBP+8]	
00402518	83C9 FF	OR ECX,FFFFFFFF	
0040251B	33C0	XOR EAX,EAX	
0040251D	F2AE	REPNE SCAS BYTE PTR ES:[EDI]	
0040251F	F7D1	NOT ECX	
00402521	83C1 FF	ADD ECX,-1	
00402524	83F9 04	CMP ECX,4	

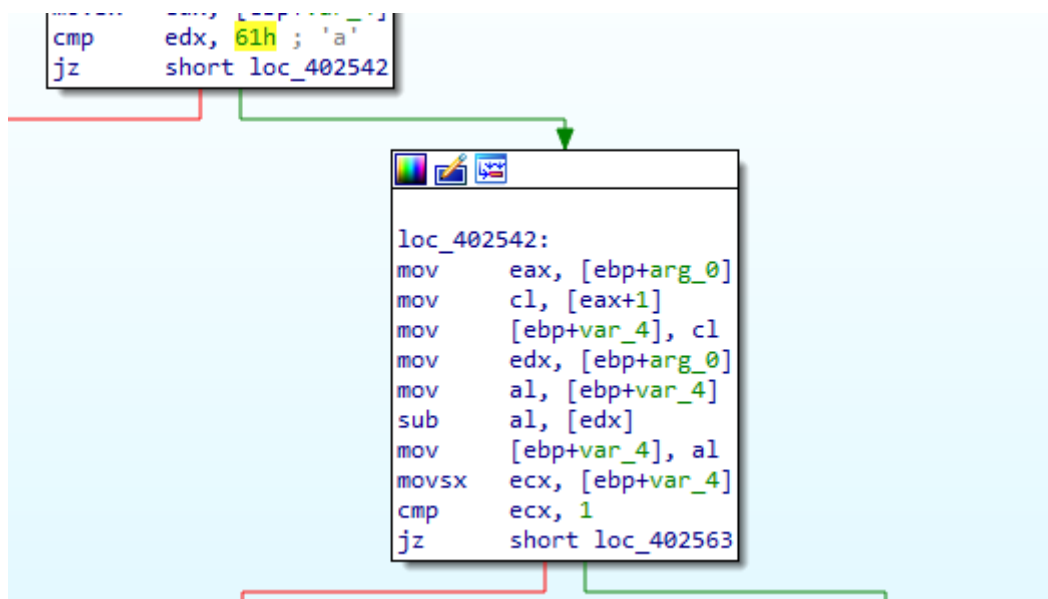
Po włączeniu debuggera i podaniu argumentu widzimy, że on trafia do tej funkcji (np. „aaaa”).

00402510	55	PUSH EBP	
00402511	8BEC	MOV EBP,ESP	
00402513	51	PUSH ECX	
00402514	57	PUSH EDI	
00402515	8B7D 08	MOV EDI,DWORD PTR SS:[EBP+8]	
00402518	83C9 FF	OR ECX,FFFFFFFF	
0040251B	33C0	XOR EAX,EAX	
0040251D	F2AE	REPNE SCAS BYTE PTR ES:[EDI]	
0040251F	F7D1	NOT ECX	
00402521	83C1 FF	ADD ECX,-1	
00402524	83F9 04	CMP ECX,4	
00402527	74 04	JE SHORT Lab06-01.0040252D	
00402529	33C0	XOR EAX,EAX	
0040252B	EB 73	JMP SHORT Lab06-01.004025A0	
0040252D	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
00402530	8A08	MOV CL, BYTE PTR DS:[EAX]	
00402532	8B40 FC	MOV BYTE PTR SS:[EBP-4],CL	
00402535	0FB555 FC	MOVSX EDX, BYTE PTR SS:[EBP-4]	
00402539	83FA 61	CMP EDX,61	
0040253C	74 04	JE SHORT Lab06-01.00402542	
0040253E	33C0	XOR EAX,EAX	
00402540	EB 5E	JMP SHORT Lab06-01.004025A0	
00402542	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
00402545	8A48 01	MOV CL, BYTE PTR DS:[EAX+1]	
00402548	8B40 FC	MOV BYTE PTR SS:[EBP-4],CL	
0040254B	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]	
0040254E	8A45 FC	MOV AL, BYTE PTR SS:[EBP-4]	
00402551	2A02	SUB AL, BYTE PTR DS:[EDX]	
00402553	8A45 FC	MOV BYTE PTR SS:[EBP-4],AL	
00402556	0FB540 FC	MOVSX ECX, BYTE PTR SS:[EBP-4]	
0040255A	83F9 01	CMP ECX,1	
0040255D	74 04	JE SHORT Lab06-01.00402563	
0040255F	33C0	XOR EAX,EAX	
00402561	EB 3D	JMP SHORT Lab06-01.004025A0	

Registers (FPU)
EAX 00890B59 ASCII "aaaa"
ECX 00890B10
EDX 00890B59 ASCII "aaaa"
EBX 7FFD5000
ESP 0012E74C
EBP 0012FF80
ESI FFFFFFFF
EDI 7C910208 ntdll.7C910208
EIP 00402510 Lab06-01.00402510
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
I 0
O 0 LastErr ERROR_MOD_NOT_FOUND (0000007E)
EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty -UNORM BDEC 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.000000000000000000000000
ST7 empty 1.000000000000000000000000
FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (EQ)
FCW 027F Prec NEAR,S3 Mask 1 1 1 1 1 1

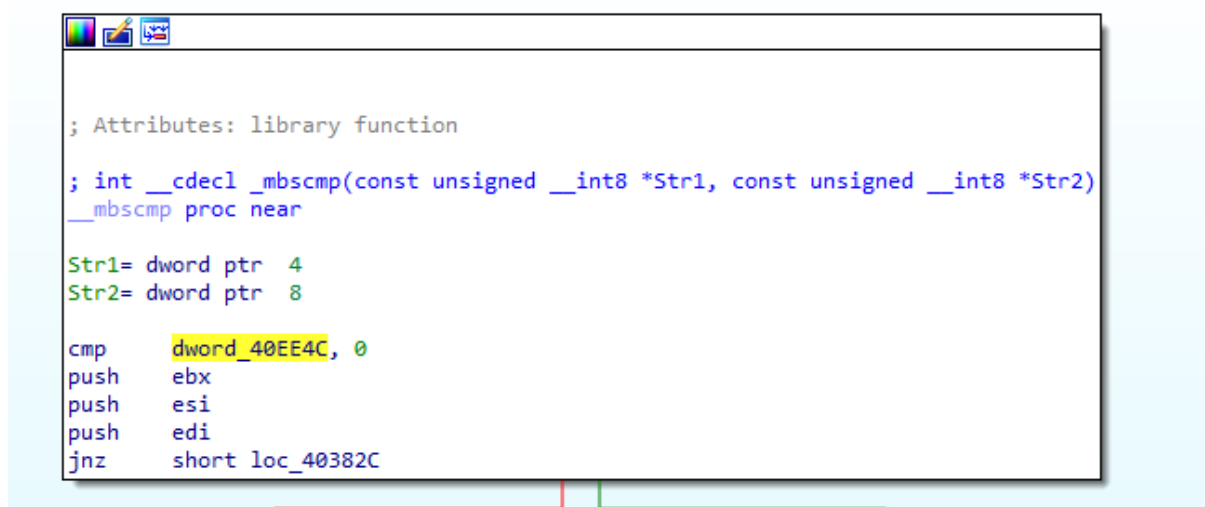
Dzięki temu, że mamy odpowiednią długość to przenosimy się do kolejnej rutyny. Widzimy, że pierwszym argumentem jest a. Następnie sprawdzamy kolejne.

loc_40252D:
mov eax, [ebp+arg_0]
mov cl, [eax]
mov [ebp+var_4], cl
movsx edx, [ebp+var_4]
cmp edx, 61h ; 'a'
jz short loc_402542



Zauważamy odejmowanie i okazuje się, że różnica może wynosić maksymalnie 1, czyli następnym znakiem powinno być b. Później c i d, co widać też w IDA.

Przechodząc do następnego kroku mamy podrutynę 402B3F, gdzie mamy wywołanie porównania argumentu ze stringiem.



Widać dokładnie, że porównuje ona, czy wprowadzono którąś z opcji opisanych offsetami?

```

loc_402B3F:
mov     ecx, [ebp+argv]
mov     edx, [ecx+4]
mov     [ebp+Str1], edx
push    offset byte_40C170 ; Str2
mov     eax, [ebp+Str1]
push    eax ; Str1
call    __mbscmp
add     esp, 8
test    eax, eax
jnz     short loc_402BC7

```

W IDA widzimy Str2 z takim offsetem i w OllyDbg to jest zdekodowane jako „-in”.

00402B38	. 75 05	JNZ SHORT Lab06-01.00402B3F	
00402B3A	. E8 D1F8FFFF	CALL Lab06-01.00402410	
00402B3F	> 8B40 0C	MOV ECX,DWORD PTR SS:[EBP+C]	
00402B42	. 8B51 04	MOV EDX,DWORD PTR DS:[ECX+4]	
00402B45	. 8995 E0E7FFFF	MOV DWORD PTR SS:[EBP-1820],EDX	
00402B48	. 68 70C14000	PUSH Lab06-01.0040C170	ASCII "-in"
00402B50	. 8B85 E0E7FFFF	MOV EAX,DWORD PTR SS:[EBP-1820]	
00402B56	. 50	PUSH EAX	
00402B57	. E8 B30C0000	CALL Lab06-01.0040380F	
00402B5C	. 83C4 08	ADD ESP,8	
00402B5F	. 85C0	TEST EAX,EAX	
00402B64	. 75 04	JNZ SHORT Lab06-01.00402B67	

Poza tym mamy kilka innych opcji:

- „-in”
- „-re”
- „-c”
- „-cc”.

1. W jaki sposób można zmusić malware do instalacji?

Trzeba dodać opcję -in do argumentów.

2. Podaj argumenty wiersza poleceń dla tego programu. Jakie są wymagania dotyczące hasła?

Potencjalne opcje to:

- „-in”
- „-re”
- „-c”
- „-cc”.

Znalezione hasło to „abcd” – jest 4 literowe.

3. Jak można wykorzystać OllyDbg do wprowadzenia zmian w tym malware, aby nie wymagał podawania hasła w wierszu poleceń?

Można spatchować kod w miejscu gdzie jest sprawdzane hasło. Tak jak poniżej.

0040250D	CC	INT3
0040250E	CC	INT3
0040250F	CC	INT3
00402510	B8 01000000	MOV EAX,1
00402515	C3	RETN
00402516	7D 08	JGE SHORT Lab06-01.00402520
00402518	83C9 FF	OR ECX,FFFFFFFF
0040251B	33C0	XOR EAX,EAX
0040251D	F2:AE	REPNE SCAS BYTE PTR ES:[EDI]
0040251F	F7D1	NOT ECX
00402521	83C1 FF	ADD ECX,-1
00402524	83F9 04	CMP ECX,4
00402527	74 04	JE SHORT Lab06-01.0040252D

4. Podaj indykatory hostowe związane z tym malware.

Przykłady:

- „%SYSTEMROOT%\system32”
- SOFTWARE\Microsoft\XPS

5. Opisz działania umożliwiające wykorzystanie złośliwego pliku przy pomocy sieci Internet.

Możliwe działania:

- pobranie szkodliwego pliku,
- wysłanie czegoś do zdalnego hosta,
- DOS,
- przekształcenie w stan SLEEP
- wykonanie kodu i zwrócenie wyniku do innego hosta.

6. Wymień przydatne wskaźniki sieciowe tego malware.

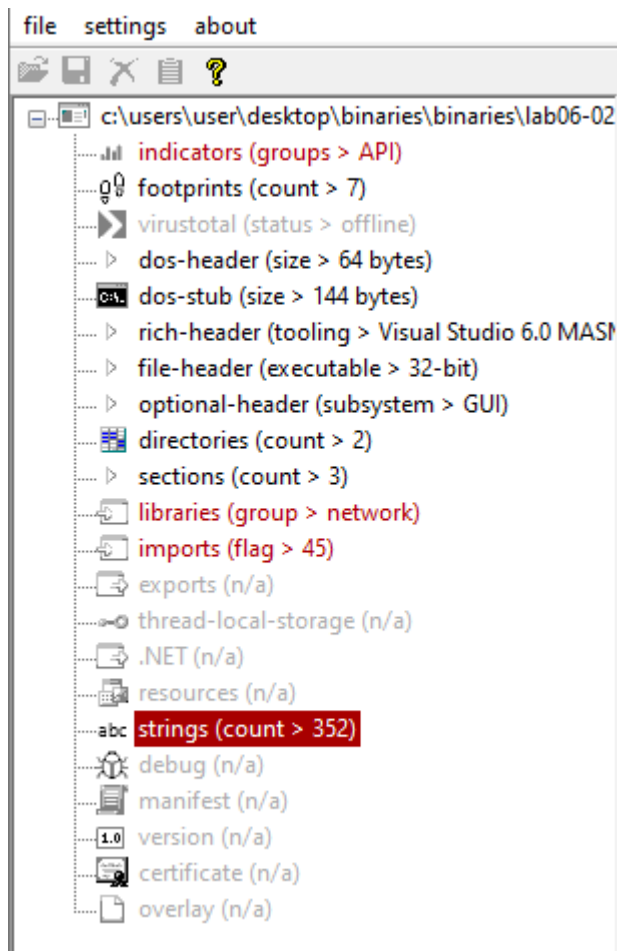
Wskaźniki:

- <http://www.practicalmalwareanalysis.com>
- HTTP 1.0
- GET
- Port 80
- command.com.

Laboratorium 5.2

Przeprowadź analizę pliku Lab06-02.exe za pomocą programu OllyDbg i odpowiedz na poniższe pytania:

1. Przeanalizuj i wypisz łańcuchy znaków, które jesteśmy w stanie odszukać w pliku.



Jest ich około 300, przeważnie śmieci albo coś związane z API.

2. Opisz wynik działania z uruchomienia tego pliku.

Na początku nic się nie dzieje gdy po prostu chcemy uruchomić.

3. W jaki sposób zmusić analizowany plik do uruchomienia swojej szkodliwej zawartości?

```

mov [ebp+var_1AB], 77h ; 'w'
mov [ebp+var_1AA], 73h ; 's'
mov [ebp+var_1A9], 78h ; 'x'
mov [ebp+var_1A8], 33h ; '3'
mov [ebp+var_1A7], 65h ; 'e'
mov [ebp+var_1A6], 64h ; 'd'
mov [ebp+var_1A5], 63h ; 'c'
mov [ebp+var_1A4], 0
mov [ebp+Str1], 6Fh ; 'o'
mov [ebp+var_19F], 63h ; 'c'
mov [ebp+var_19E], 6Ch ; 'l'
mov [ebp+var_19D], 2Eh ; '.'
mov [ebp+var_19C], 65h ; 'e'
mov [ebp+var_19B], 78h ; 'x'
mov [ebp+var_19A], 65h ; 'e'
mov [ebp+var_199], 0
mov ecx, 8
mov esi, offset unk_405034
lea edi, [ebp+var_1F0]
rep movsd

```

Spróbujemy zmienić nazwę na ocl.exe, ponieważ program oczekuje w strcmp takiej właśnie nazwy pliku (widać to po debugowaniu).

Address	Hex dump	ASCII	Registers (FPU)
00401236	CALL ocl.004014C0		EAX 0012FCB0 ASCII "ocl.exe"
0040123B	ADD ESP, 8		ECX 0012FDE0 ASCII "ocl.exe"
0040123E	TEST EAX, EAX		EDX 0012FCB0 ASCII "ocl.exe"
00401240	JE SHORT ocl.0040124C		EBX 7FFD4000
00401242	MOV EAX, 1		ESP 0012FC6C
00401247	JMP ocl.004013D6		EBP 0012FFB0
0040124C	MOV EDI, 1		ESI 00405055 ocl.00405055
00401251	TEST EDI, EDI		EDI 0012FDE0
00401253	JE ocl.004013D4		EIP 00401236 ocl.00401236
00401259	LEA EAX, DWORD PTR SS:[EBP-198]		C 0 ES 0023 32bit 0 (FFFFFFFF)
0040125F	PUSH EAX		F 0 CS 001B 32bit 0 (FFFFFFFF)
00401260	PUSH 202		A 1 SS 0023 32bit 0 (FFFFFFFF)
00401265	CALL DWORD PTR DS:[<MS2_32.115>]		Z 0 DS 0023 32bit 0 (FFFFFFFF)
00401268	MOV DWORD PTR SS:[EBP-1B4], EAX		S 0 FS 0038 32bit 7FFDF000 (FFF)
00401271	CMOV DWORD PTR SS:[EBP-1B4], 0		T 0 GS 0000 NULL
00401278	JE SHORT ocl.00401284		O 0 LastErr ERROR_INVALID_HANDLE (0)
0040127A	MOV EAX, 1		EFL 00000212 (NO, NB, NE, A, NS, PO, GE, G)
0040127F	JMP ocl.004013D6		ST0 empty -UNORM BDEC 01050104 000000
00401286	PUSH 0		ST1 empty 0.0
00401288	PUSH 0		ST2 empty 0.0
0040128A	PUSH 0		ST3 empty 0.0
0040128C	PUSH 6		ST4 empty 0.0
0040128E	PUSH 1		ST5 empty 0.0
0040128E	PUSH 2		ST6 empty 0.0
00401290	CALL DWORD PTR DS:[<MS2_32.WSASocketA		ST7 empty 0.0
00401296	MOV DWORD PTR SS:[EBP-304], EAX		FST 0000 Cond 3 2 1 0 Err 0 0 0 0
0040129C	CMOV DWORD PTR SS:[EBP-304], -1		FCW 027F Prec NEAR, S3 Mask 1 1 1
004012A3	JNZ SHORT ocl.004012AF		
004012A5	MOV EAX, 1		
004012A8	JMP ocl.004013D6		
004012AF	LEA ECX, DWORD PTR SS:[EBP-1F0]		
004012B5	PUSH ECX		
004012B6	LEA EDI, DWORD PTR SS:[EBP-1B0]		
004012BC	PUSH EDI		
004012BD	CALL ocl.00401089		
004012C2	ADD ESP, 8		
004012C5	MOV DWORD PTR SS:[EBP-8], EAX		
004012C8	MOV EAX, DWORD PTR SS:[EBP-8]		
004012CB	PUSH EAX		
004012CC	CALL DWORD PTR DS:[<MS2_32.#52>]		
004012D2	MOV DWORD PTR SS:[EBP-1BC], EAX		
004012D8	CMOV DWORD PTR SS:[EBP-1BC], 0		
004012DF	JNZ SHORT ocl.004013D4		
004012E1	MOV ECX, DWORD PTR SS:[EBP-304]		
004012E7	PUSH ECX		
004012E8	CALL DWORD PTR DS:[<MS2_32.#3>]		
004012EE	CALL DWORD PTR DS:[<MS2_32.116>]		
004012F4	PUSH 7530		
004012F9	CALL DWORD PTR DS:[<KERNEL32.Sleep>]		
004012FF	JMP ocl.0040124C		
00401304	MOV EDI, DWORD PTR SS:[EBP-1BC]		
0040130A	MOV EAX, DWORD PTR DS:[EDI+C]		
0040130D	MOV ECX, DWORD PTR DS:[EAX]		
0040130F	MOV EDI, DWORD PTR DS:[ECX]		
00401311	MOV DWORD PTR SS:[EBP-1C8], EDI		
00401317	PUSH 270F		
0040131C	CALL DWORD PTR DS:[<MS2_32.#9>]		
00401322	MOV DWORD PTR SS:[EBP-1C8], 0		
004014C0	CALL ocl.004014C0		

00401259	8085 68FEFFFF	LEA EAX,DWORD PTR SS:[EBP-198]	
0040125F	50	PUSH EAX	
00401260	68 02020000	PUSH 202	
00401265	FF15 9C404000	CALL DWORD PTR DS:[&WS2_32.#115>]	pWSAData RequestedVersion = 202 (2.2.)
0040126B	8985 4CFEFFFF	MOV DWORD PTR SS:[EBP-1B4],EAX	WSAStartup
00401271	83BD 4CFEFFFF	CMP DWORD PTR SS:[EBP-1B4],0	
00401278	74 0A	JE SHORT ocl_exe.00401284	
0040127A	B8 01000000	MOV EAX,1	
0040127F	E9 52010000	JMP ocl_exe.004013D6	
00401284	6A 00	PUSH 0	Flags = 0
00401286	6A 00	PUSH 0	Group = 0
00401288	6A 00	PUSH 0	pWSAProtocol = NULL
0040128A	6A 06	PUSH 6	Protocol = IPPROTO_TCP
0040128C	6A 01	PUSH 1	Type = SOCK_STREAM
0040128E	6A 02	PUSH 2	Family = AF_INET
00401290	FF15 A0404000	CALL DWORD PTR DS:[&WS2_32.WSASocketA]	WSASocketA
00401296	8985 FCFCFFFF	MOV DWORD PTR SS:[EBP-304],EAX	
0040129C	83BD FCFCFFFF	CMP DWORD PTR SS:[EBP-304],-1	
004012A3	75 0A	JNZ SHORT ocl_exe.004012AF	
004012A5	B8 01000000	MOV EAX,1	
004012AA	E9 27010000	JMP ocl_exe.004013D6	
004012AF	80BD 10FEFFFF	LEA ECX,DWORD PTR SS:[EBP-1F0]	
004012B5	51	PUSH ECX	Arg2
004012B6	8095 50FEFFFF	LEA EDI,DWORD PTR SS:[EBP-1B0]	Arg1
004012BC	52	PUSH EDI	ocl_exe.00401089
004012BD	E8 C7FDFFFF	CALL ocl_exe.00401089	
004012C2	83C4 08	ADD ESP,8	
004012C5	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX	
004012C8	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	
004012CB	50	PUSH EAX	Name
004012CC	FF15 A4404000	CALL DWORD PTR DS:[&WS2_32.#52>]	gethostbyname
004012D2	8985 44FEFFFF	MOV DWORD PTR SS:[EBP-1BC],EAX	
004012D8	83BD 44FEFFFF	CMP DWORD PTR SS:[EBP-1BC],0	
004012DF	75 23	JNZ SHORT ocl_exe.00401304	
004012E1	8B8D FCFCFFFF	MOV ECX,DWORD PTR SS:[EBP-304]	
004012E7	51	PUSH ECX	Socket
004012E8	FF15 A8404000	CALL DWORD PTR DS:[&WS2_32.#3>]	closesocket
004012EE	FF15 AC404000	CALL DWORD PTR DS:[&WS2_32.#116>]	WSACleanup
004012F4	68 30750000	PUSH 7530	Timeout = 30000. ms
004012F9	FF15 08404000	CALL DWORD PTR DS:[&KERNEL32.Sleep>]	Sleep
004012FF	E9 48FEFFFF	JMP ocl_exe.0040124C	
00401304	8B95 44FEFFFF	MOV EDI,DWORD PTR SS:[EBP-1BC]	
0040130A	8B42 0C	MOV EAX,DWORD PTR DS:[EDI+C]	
0040130D	8B08	MOV ECX,DWORD PTR DS:[EAX]	

Malware otwiera połączenie. Następnie po pobraniu komendy wykonuje ją w cmd.exe.

00401028	804D F0	LEA ECX,DWORD PTR SS:[EBP-10]	
0040102B	51	PUSH ECX	
0040102C	E8 AF030000	CALL ocl_exe.004013E0	
00401031	83C4 0C	ADD ESP,0C	
00401034	C745 D4 01010000	MOV DWORD PTR SS:[EBP-2C],101	
0040103B	66:C745 D8 00000000	MOV WORD PTR SS:[EBP-28],0	
00401041	8B55 18	MOV EDX,DWORD PTR SS:[EBP+18]	
00401044	8955 E0	MOV DWORD PTR SS:[EBP-20],EDX	
00401047	8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]	
0040104A	8945 E8	MOV DWORD PTR SS:[EBP-18],EAX	
0040104D	8B4D E8	MOV ECX,DWORD PTR SS:[EBP-18]	
00401050	894D E4	MOV DWORD PTR SS:[EBP-1C],ECX	
00401053	8D55 F0	LEA EDI,DWORD PTR SS:[EBP-10]	
00401056	52	PUSH EDI	pProcessInfo
00401057	8D45 A8	LEA EAX,DWORD PTR SS:[EBP-58]	
0040105A	50	PUSH EAX	pStartupInfo
0040105B	6A 00	PUSH 0	CurrentDir = NULL
0040105D	6A 00	PUSH 0	pEnvironment = NULL
0040105F	6A 00	PUSH 0	CreationFlags = 0
00401061	6A 01	PUSH 1	InheritHandles = TRUE
00401063	6A 00	PUSH 0	pThreadSecurity = NULL
00401065	6A 00	PUSH 0	pProcessSecurity = NULL
00401067	68 30504000	PUSH ocl_exe.00405030	CommandLine = "cmd"
0040106C	6A 00	PUSH 0	ModuleFileName = NULL
0040106E	FF15 04404000	CALL DWORD PTR DS:[&KERNEL32.CreateProcessA]	CreateProcessA
00401074	8945 EC	MOV DWORD PTR SS:[EBP-14],EAX	
00401077	6A FF	PUSH -1	Timeout = INFINITE
00401079	8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	
0040107C	51	PUSH ECX	hObject
0040107D	FF15 00404000	CALL DWORD PTR DS:[&KERNEL32.WaitForSingleObject]	WaitForSingleObject
00401083	33C0	XOR EAX,EAX	
00401085	8BE5	MOV ESP,EBP	
00401087	5D	POP EBP	
00401088	C3	RETN	
00401089	55	PUSH EBP	
0040108A	8BEC	MOV EBP,ESP	
0040108C	81EC 00010000	SUB ESP,108	
00401092	57	PUSH EDI	
00401093	C785 F8FEFFFF	MOV DWORD PTR SS:[EBP-108],0	

4. Opisz działania znajdujące się pod adresem 0x00401133.

Dochodzi do zapisania danych. W dumpie to jest widoczne jako słabo czytelne znaki.

```
00401133  C6 85 50 FE FF FF 31 C6 1F 1F 1F 1F 1F 1F 1F 1F
00401134  8E E4 EE EE EE 74 C6 8E 50 74 C6 8E 50 74 C6 8E 50 74
```

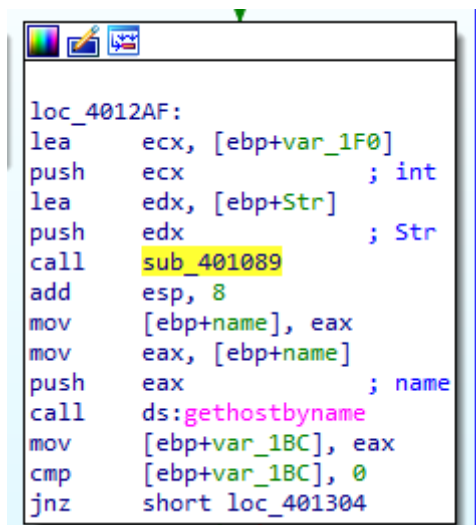
W IDA to wygląda tak.


```

mov     [ebp+Str], 31h ; '1'
mov     [ebp+var_1AF], 71h ; 'q'
mov     [ebp+var_1AE], 61h ; 'a'
mov     [ebp+var_1AD], 7Ah ; 'z'
mov     [ebp+var_1AC], 32h ; '2'
mov     [ebp+var_1AB], 77h ; 'w'
mov     [ebp+var_1AA], 73h ; 's'
mov     [ebp+var_1A9], 78h ; 'x'
mov     [ebp+var_1A8], 33h ; '3'
mov     [ebp+var_1A7], 65h ; 'e'
mov     [ebp+var_1A6], 64h ; 'd'
mov     [ebp+var_1A5], 63h ; 'c'
mov     [ebp+var_1A4], 0
mov     [ebp+Str1], 6Fh ; 'o'
mov     [ebp+var_19F], 63h ; 'c'
mov     [ebp+var_19E], 6Ch ; 'l'
mov     [ebp+var_19D], 2Eh ; '.'
mov     [ebp+var_19C], 65h ; 'e'
mov     [ebp+var_19B], 78h ; 'x'
mov     [ebp+var_19A], 65h ; 'e'
mov     [ebp+var_199], 0
mov     ecx, 8

```

5. Podaj argumenty, które są przekazywane do podprogramu pod adresem 0x00401089?



```

loc_4012AF:
lea     ecx, [ebp+var_1F0]
push    ecx                ; int
lea     edx, [ebp+Str]
push    edx                ; Str
call    sub_401089
add     esp, 8
mov     [ebp+name], eax
mov     eax, [ebp+name]
push    eax                ; name
call    ds:gethostbyname
mov     [ebp+var_1BC], eax
cmp     [ebp+var_1BC], 0
jnz     short loc_401304

```

A ten String to jak poniżej „1qaz2wsx3edc”

```

mov     [ebp+Str], 31h ; '1'
mov     [ebp+var_1AF], 71h ; 'q'
mov     [ebp+var_1AE], 61h ; 'a'
mov     [ebp+var_1AD], 7Ah ; 'z'
mov     [ebp+var_1AC], 32h ; '2'
mov     [ebp+var_1AB], 77h ; 'w'
mov     [ebp+var_1AA], 73h ; 's'
mov     [ebp+var_1A9], 78h ; 'x'
mov     [ebp+var_1A8], 33h ; '3'
mov     [ebp+var_1A7], 65h ; 'e'
mov     [ebp+var_1A6], 64h ; 'd'
mov     [ebp+var_1A5], 63h ; 'c'
mov     [ebp+var_1A4], 0

```

Do funkcji podany jest również integer, ale nie do końca wiem jak go znaleźć.

Teraz widzimy to w OllyDbg.

The screenshot shows the OllyDbg interface. The main window displays assembly code starting at address 00401089. The code includes instructions like `PUSH EBP`, `MOV EBP, ESP`, `SUB ESP, 108`, `PUSH EDI`, `MOV DWORD PTR SS:[EBP-108], 0`, `MOV ECX, 3F`, `XOR EAX, EAX`, `LEA EDI, DWORD PTR SS:[EBP-FF]`, `REP STOS DWORD PTR ES:[EDI]`, `STOS WORD PTR ES:[EDI]`, `STOS BYTE PTR ES:[EDI]`, `MOV EAX, DWORD PTR SS:[EBP+8]`, `PUSH EAX`, `CALL ocl.00401440`, `ADD ESP, 4`, `MOV DWORD PTR SS:[EBP-104], EAX`, `MOV DWORD PTR SS:[EBP-108], 0`, `JMP SHORT ocl.004010E3`, `MOV ECX, DWORD PTR SS:[EBP-108]`, `ADD ECX, 1`, `MOV DWORD PTR SS:[EBP-108], ECX`, `MOV WORD PTR SS:[EBP-108], 20`, `JGE SHORT ocl.0040111D`, `MOV EDX, DWORD PTR SS:[EBP+C]`, `ADD EDX, DWORD PTR SS:[EBP-108]`, `MOVSX ECX, BYTE PTR DS:[EDX]`, `MOV EAX, DWORD PTR SS:[EBP-108]`, `CDQ`, `IDIV DWORD PTR SS:[EBP-104]`, `MOV EAX, DWORD PTR SS:[EBP+C]`, `MOVSX EDX, BYTE PTR DS:[EAX+EDX]`, `XOR ECX, EDX`, `MOV EAX, DWORD PTR SS:[EBP-108]`, `MOV BYTE PTR SS:[EBP+EAX-100], CL`, `JMP SHORT ocl.004010D4`, `LEA EAX, DWORD PTR SS:[EBP-100]`, `POP EDI`, `MOV ESP, EBP`, `POP EBP`, `RETN`. The registers window on the right shows the state of the CPU registers, including EAX, ECX, EDI, ESI, EDI, EIP, etc. The stack window at the bottom shows the current stack frame, including the return address 0012FC68 and the ASCII string "1qaz2wsx3edc".

6. Podaj nazwę domeny, która wykorzystuje ten malware.

Tutaj widzimy domenę w kolejnym breakpoint.

The screenshot shows the OllyDbg interface. The main window displays assembly code starting at address 00401110. The code includes instructions like `LEA EAX, DWORD PTR SS:[EBP-100]`, `POP EDI`, `MOV ESP, EBP`, `POP EBP`, `RETN`, `PUSH EBP`, `MOV EBP, ESP`, `SUB ESP, 304`, `PUSH ESI`, `PUSH EDI`, and a series of `MOV BYTE PTR SS:[EBP-100], 31` instructions. The stack window at the bottom shows the current stack frame, including the return address 0012FC68 and the ASCII string "1qaz2wsx3edc".

7. Jaka procedura kodowania została zastosowana przez ten program do zaciemnienia nazwy domeny?

Użyto XORa w tym celu.

8. Opisz znaczenie wywołania `CreateProcessA` znajdującego się pod adresem 0x0040106E w nawiązaniu do tego malware?

Ten proces wykonuje cmd.exe i przekierowuje uzyskane stdout i stderr do stworzonego gniazdka (socket).

Laboratorium 5.3

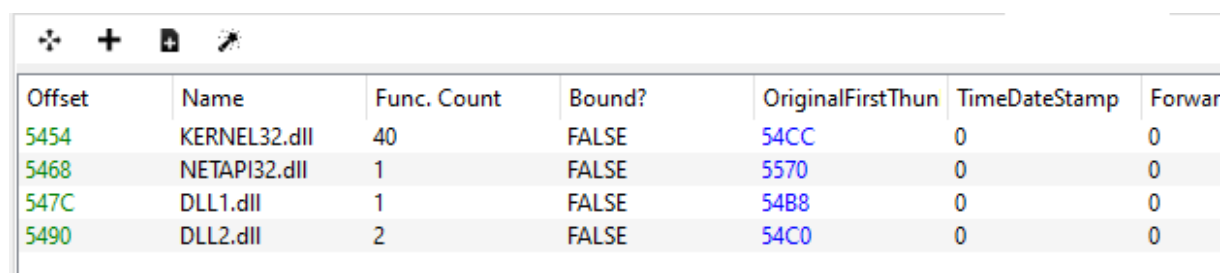
Przeprowadź analizę pliku Lab06-03.exe za pomocą programu OllyDbg i IDA. Ten malware ładuje dodatkowe 3 biblioteki DLL (DLL1.dll, DLL2.dll i DLL3.dll), które muszą znajdować się w tej samej lokalizacji podczas ładowania do pamięci. Podczas przeglądania tych bibliotek DLL w OllyDbg, w porównaniu do IDA, mogą pojawiać się różnice w lokalizacji w pamięci.

Zadanie to ma na celu ułatwienie znalezienia poprawnej lokalizacji kodu w OllyDbg w porównaniu do programu IDA. Odpowiedz na poniższe pytania:

1. Które biblioteki DLL są importowane przez Lab06-03.exe (np. PE-bear). Podaj te, które ładują się dynamicznie (IDA: funkcja LoadLibraryA).

Importowane biblioteki to:

- KERNEL32.dll,
- NETAPI32.dll,
- DLL1.dll,
- DLL2.dll.



Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwar
5454	KERNEL32.dll	40	FALSE	54CC	0	0
5468	NETAPI32.dll	1	FALSE	5570	0	0
547C	DLL1.dll	1	FALSE	54B8	0	0
5490	DLL2.dll	2	FALSE	54C0	0	0

W IDA widzimy dodatkowo dynamicznie wykorzystaną bibliotekę DLL3.dll.

```
mov     edx, [ebp+hFile]
push    edx                ; hObject
call    ds:CloseHandle
push    offset LibFileName ; "DLL3.dll"
call    ds:LoadLibraryA
mov     [ebp+hModule], eax
push    offset ProcName    ; "DLL3Print"
mov     eax, [ebp+hModule]
push    eax                ; hModule
call    ds:GetProcAddress
mov     [ebp+var_8], eax
call    [ebp+var_8]
push    offset aDll3getstructu ; "DLL3GetStructure"
mov     ecx, [ebp+hModule]
push    ecx                ; hModule
```

2. Podaj adres bazowy wymagany przez DLL1.dll, DLL2.dll i DLL3.dll (np. w PEvent).

Na zdjęciu widzimy adresy bazowe poszczególnych bibliotek.

library (4)	duplicate (0)	flag (1)	first-thunk-original (INT)	first-thunk (IAT)	type (1)	imports (4)
KERNEL32.dll	-	-	0x000054CC	0x00005014	implicit	<u>40</u>
NETAPI32.dll	-	x	0x00005570	0x000050B8	implicit	<u>1</u>
DLL1.dll	-	-	0x000054B8	0x00005000	implicit	<u>1</u>
DLL2.dll	-	-	0x000054C0	0x00005008	implicit	<u>2</u>

3. Wykorzystując OllyDbg do debugowania Lab06-03.exe podaj przypisany adres bazowy dla DLL1.dll, DLL2.dll i DLL3.dll.

Gdy załadujemy sobie plik w OllyDbg i użyjemy skrótu „ALT+M” to widzimy mapę pamięci.

00330000	00001000	DLL2		PE header	Inag R	RWE	
00331000	00006000	DLL2	.text	code	Inag R	RWE	
00337000	00001000	DLL2	.rdata	imports,exp	Inag R	RWE	
00338000	00005000	DLL2	.data	data	Inag R	RWE	
0033D000	00001000	DLL2	.reloc	relocations	Inag R	RWE	
00340000	00004000				Priv RW	RW	
00350000	00003000				Map R	RW	
00360000	00006000				Priv RW	RW	
00370000	00006000				Priv RW	RW	
00380000	00002000				Priv RW	RW	
00390000	00001000	DLL3		PE header	Inag R	RWE	
00391000	00006000	DLL3	.text	code	Inag R	RWE	
00397000	00001000	DLL3	.rdata	imports,exp	Inag R	RWE	
00398000	00005000	DLL3	.data	data	Inag R	RWE	
0039D000	00001000	DLL3	.reloc	relocations	Inag R	RWE	
003A0000	00006000				Priv RW	RW	
00400000	00001000	Lab06-03		PE header	Inag R	RWE	
00401000	00004000	Lab06-03	.text	code	Inag R	RWE	
00405000	00001000	Lab06-03	.rdata	imports	Inag R	RWE	
00406000	00003000	Lab06-03	.data	data	Inag R	RWE	
10000000	00001000	DLL1		PE header	Inag R	RWE	
10001000	00006000	DLL1	.text	code	Inag R	RWE	
10007000	00001000	DLL1	.rdata	imports,exp	Inag R	RWE	
10008000	00005000	DLL1	.data	data	Inag R	RWE	
1000D000	00001000	DLL1	.reloc	relocations	Inag R	RWE	

DLL1 – 10000000

DLL2 – 00330000

DLL3 – 00390000.

4. Opisz działanie importowanej funkcji z DLL1.dll wywoływanej przez Lab06-03.exe.

Zajrzyjmy co widać w IDA.

```
; Exported entry 1. DLL1Print

; Attributes: bp-based frame

public DLL1Print
DLL1Print proc near
push    ebp
mov     ebp, esp
mov     eax, dword_10008030
push    eax
push    offset aDll1MysteryDat ; "DLL 1 mystery data %d\n"
call    sub_10001038
add     esp, 8
pop     ebp
retn
DLL1Print endp
```

Widzimy, że jako argument do printf podany jest dword_10008030. Jeśli sobie popatrzymy, gdzie go użyto, to się okaże, że to **PID procesu**.

```
push    ebp
mov     ebp, esp
call    ds:GetCurrentProcessId
mov     dword_10008030, eax
mov     al, 1
pop     ebp
retn    0Ch
```

10001020	55	PUSH EBP	
10001021	8BEC	MOV ESP, EBP	
10001023	A1 30300010	MOV EBX, DWORD PTR DS:[10008030]	
10001028	50	PUSH EBX	
10001029	68 34300010	PUSH DLL1.10008034	ASCII "DLL 1 mystery data %c"
1000102E	E8 55000000	CALL DLL1.10001038	
10001033	83C4 08	ADD ESP, 8	
10001036	5D	POP EBP	
10001037	C3	RETN	
10001038	53	PUSH EBX	
10001039	56	PUSH ESI	
1000103A	BE 70300010	MOV ESI, DLL1.10008070	
1000103F	57	PUSH EDI	
10001040	56	PUSH ESI	
10001041	6A 01	PUSH 1	
10001043	E8 C5020000	CALL DLL1.10001300	
10001048	56	PUSH ESI	
10001049	E8 34030000	CALL DLL1.10001382	
1000104E	8BF8	MOV EDI, EAX	
10001050	8D4424 20	LEA EAX, DWORD PTR SS:[ESP+20]	
10001054	50	PUSH EAX	
10001055	FF7424 20	PUSH DWORD PTR SS:[ESP+20]	
10001059	56	PUSH ESI	
1000105A	E8 DA030000	CALL DLL1.10001439	
1000105F	56	PUSH ESI	
10001060	57	PUSH EDI	
10001061	8BD8	MOV EBX, EAX	
10001063	E8 A7030000	CALL DLL1.1000140F	
10001068	56	PUSH ESI	
10001069	6A 01	PUSH 1	
1000106B	E8 EF020000	CALL DLL1.1000135F	
10001070	83C4 28	ADD ESP, 28	
10001073	8BC3	MOV EAX, EBX	
10001075	5F	POP EDI	
10001076	5E	POP ESI	
10001077	5B	POP EBX	
10001078	C3	RETN	
10001079	8B4424 08	MOV EAX, DWORD PTR SS:[ESP+8]	
1000107D	83F8 01	CMP EAX, 1	
10001080	75 0F85 88000000	JNZ DLL1.1000110E	
10001086	FF15 88700010	CALL DWORD PTR DS:[<&KERNEL32.GetVersion kernel32.GetVersion	
1000108C	6A 01	PUSH 1	
1000108E	A3 3C000010	MOV DWORD PTR DS:[1000AD3C], EAX	
10001093	E8 6F160000	CALL DLL1.10002707	
10001098	85C9	TEST EAX, EAX	
1000109A	59	POP ECX	
1000109B	74 3C	JE SHORT DLL1.100010D9	
1000109D	A1 3C000010	MOV EAX, DWORD PTR DS:[1000AD3C]	
100010A2	33C9	XOR ECX, ECX	
100010A4	8A00 3D000010	MOV CL, BYTE PTR DS:[1000AD3D]	
100010AA	25 FF000000	AND EAX, 0FF	
100010AF	C12D 3C000010	SHR DWORD PTR DS:[1000AD3C], 10	
100010B5	A3 44000010	MOV DWORD PTR DS:[1000AD44], EAX	
100010BB	890D 48000010	MOV DWORD PTR DS:[1000AD48], ECX	
100010C1	C1E0 08	SHL EAX, 8	
100010C4	03C1	ADD EAX, ECX	
100010C6	A3 48000010	MOV DWORD PTR DS:[1000AD48], EAX	
100010CB	E8 F8030000	CALL DLL1.10001DBE	
10008034=DLL1.10008034 (ASCII "DLL 1 mystery data %c")			

Po zdebugowaniu również okazuje się, że zwraca stringa "DLL1 mystery data %d\n".

5. Jaką nazwę pliku wykorzystuje funkcja WriteFile podczas zapisu (Lab06-03.exe w związku z plikiem DLL2.dll)?

Tutaj mamy kilka kroków, trzeba się zagłębić. Zaczniemy od WriteFile. Widzimy, że wykorzystuje zmienną hFile.

```
mov     ecx, [ebp+hFile]
push    ecx                ; hFile
call    ds:WriteFile
mov     edx, [ebp+hFile]
push    edx                ; hObject
call    ds:CloseHandle
```

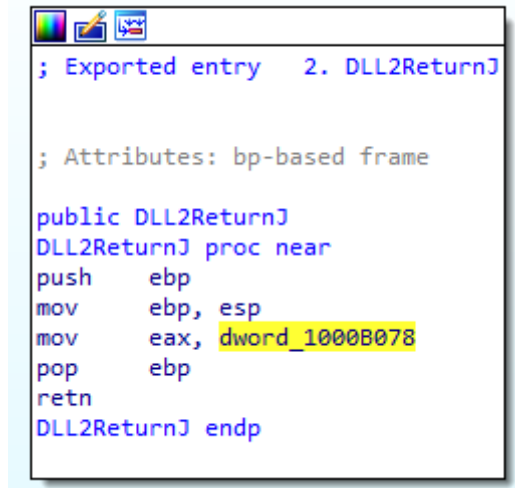
Prześledźmy biblioteki, które w IDA są w pobliżu tej zmiennej. Weźmy DLL2, a konkretnie DLL2ReturnJ.

```

call    ds:DLL1Print
call    ds:DLL2Print
call    ds:DLL2ReturnJ
mov     [ebp+hFile], eax

```

Samo ReturnJ wygląda tak.



```

; Exported entry 2. DLL2ReturnJ

; Attributes: bp-based frame

public DLL2ReturnJ
DLL2ReturnJ proc near
push    ebp
mov     ebp, esp
mov     eax, dword_1000B078
pop     ebp
retn
DLL2ReturnJ endp

```

Teraz śledząc tego dword_1000B078 znajdujemy ciekawą rzecz.

```

push    ebp
mov     ebp, esp
push    0                ; hTemplateFile
push    80h              ; dwFlagsAndAttributes
push    2                ; dwCreationDisposition
push    0                ; lpSecurityAttributes
push    0                ; dwShareMode
push    40000000h        ; dwDesiredAccess
push    offset FileName ; "temp.txt"
call    ds:CreateFileA
mov     dword_1000B078, eax
mov     al, 1
pop     ebp
retn    0Ch
_DllMain@12 endp

```

Można wywnioskować, że została wykorzystana nazwa „temp.txt”.

6. Skąd pobierane są dane dla drugiego parametru z funkcji NetScheduleJobAdd?

Drugim parametrem jest Buffer.

```

; Attributes: thunk

; DWORD __stdcall NetScheduleJobAdd(LPCWSTR Servername, LPBYTE Buffer, LPDWORD JobId)
NetScheduleJobAdd proc near

Servername= dword ptr 4
Buffer= dword ptr 8
JobId= dword ptr 0Ch

jmp ds:imp_NetScheduleJobAdd
NetScheduleJobAdd endp

```

Bierze się on z DLL3GetStructure.

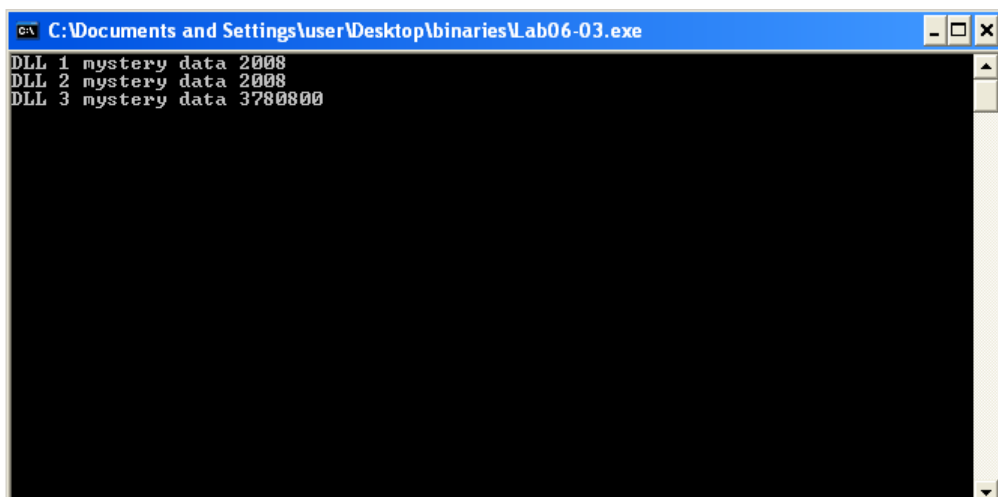
```

call [ebp+var_8]
push offset aDll3getstructu ; "DLL3GetStructure"
mov ecx, [ebp+hModule]
push ecx ; hModule
call ds:GetProcAddress
mov [ebp+var_10], eax
lea edx, [ebp+Buffer]
push edx
call [ebp+var_10]
add esp, 4
lea eax, [ebp+JobId]
push eax ; JobId
mov ecx, [ebp+Buffer]
push ecx ; Buffer
push 0 ; Servername
call NetScheduleJobAdd

```

7. Podczas uruchamiania lub debugowania programu można zobaczyć, że wyświetla on trzy fragmenty tajemniczych danych. Z czym są powiązane: DLL 1 mystery data, DLL 2 mystery data i DLL 3 mystery data?

W poprzednich punktach już analizowałem część z nich.



```

C:\Documents and Settings\user\Desktop\binaries\Lab06-03.exe
DLL 1 mystery data 2008
DLL 2 mystery data 2008
DLL 3 mystery data 3780800

```


Wiemy, że każdy DLL odnosi się do czego innego:

- DLL1 -> PID programu,
- DLL2 -> handle do temp.exe,
- DLL3 -> adres Buffer w pamięci.

8. W jaki sposób można załadować DLL2.dll do IDA, aby było to zgodne z adresem ładowania zastosowanym przez OllyDbg?

W OllyDbg był to adres: 00330000. Wybieramy w IDA opcję Manual Load i wpisujemy odpowiedni adres.

