Lab 06-1.malware

1. **What is the address of the _main function?**

The main function is located at 0x403420.

   a. **What imported function does main call? What do these functions do?**
      I. *GetCommandLineA* Retrieves the command-line string for the current process.
      II. *GetStartupInfoA* Retrieves the contents of the STARTUPINFO structure that was specified when the calling process was created.
      III. *GetModuleHandleA* Retrieves a module handle for the specified module. The module must have been loaded by the calling process.

2. **Looking at the subroutine at 0x00402C6E**
   a. **Is there an encoding/decoding function? If so:**

      Yes.

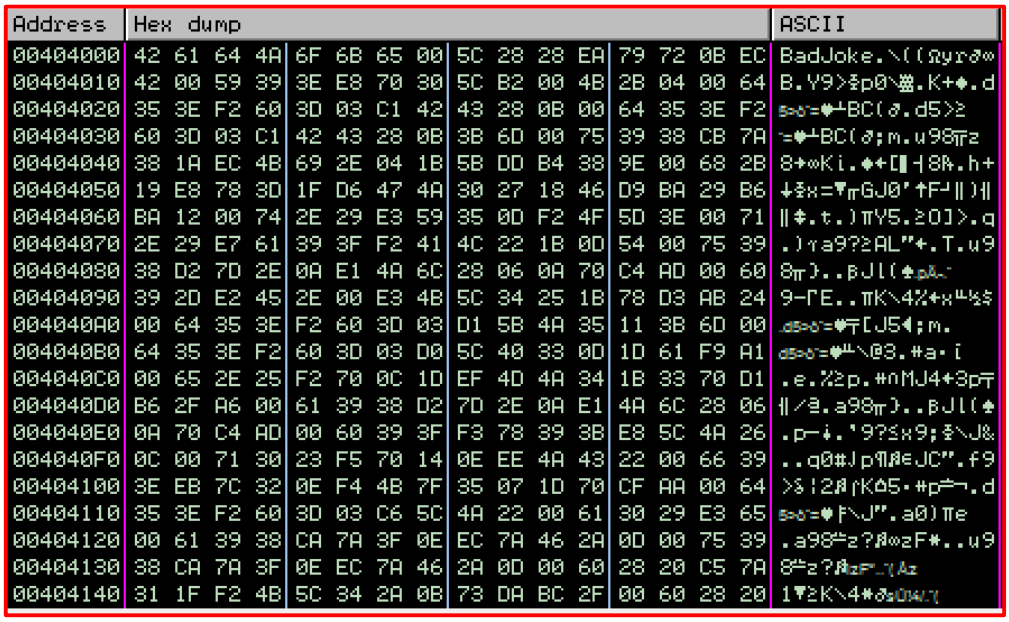
```
00402C71 sub     esp, 148h       ; char *
00402C77 lea     edx, [ebp+var_118] ; Load Effective Address
00402C7D mov     eax, 104h
00402C82 mov     [esp+148h+var_140], eax
00402C86 mov     [esp+148h+var_144], 0
00402C8E mov     [esp+148h+var_148], edx
00402C91 call    memset          ; Call Procedure
00402C96 mov     [ebp+var_11C], 0
00402CA0 mov     [esp+148h+var_140], 9
00402CA8 mov     [esp+148h+var_144], offset aYrB ; "\\((Oyr\v8B"
00402CB0 mov     [esp+148h+var_148], offset aBadjoke ; "BadJoke"
00402CB7 call    decryption_function ; Call Procedure
00402CBC mov     [esp+148h+var_144], eax
00402CC0 mov     [esp+148h+var_148], offset aYrB ; "\\((Oyr\v8B"
00402CC7 call    strcpy          ; Call Procedure
00402CCC mov     [esp+148h+var_140], 0Ch
00402CD4 mov     [esp+148h+var_144], offset aY9Sp0 ; "Y9>Fp0\\¦"
00402CDC mov     [esp+148h+var_148], offset aBadjoke ; "BadJoke"
00402CE3 call    decryption_function ; Call Procedure
00402CE8 mov     [esp+148h+var_144], eax
00402CEC mov     [esp+148h+var_148], offset aY9Sp0 ; "Y9>Fp0\\¦"
00402CF3 call    strcpy          ; Call Procedure
00402CF8 mov     [esp+148h+var_140], 0Ch
00402D00 mov     [esp+148h+var_144], offset aS8Ce5 ; "S8:te5\\¦"
00402D08 mov     [esp+148h+var_148], offset aBadjoke ; "BadJoke"
00402D0F call    decryption_function ; Call Procedure
00402D14 mov     [esp+148h+var_144], eax
```

      I. **What is the address of the function?**

         0x4012EC (renamed to decryption_function in above screenshot)

II. What is being encoded/decoded?

A series of encoded strings is being decrypted using the key “BadJoke”. Before and after photos are shown below.



Address	Hex dump												ASCII
00404000	42	61	64	4A	6F	6B	65	00	5C	28	28	EA	BadJoke.\((Ωyrð∞
00404010	42	00	59	39	3E	E8	70	30	5C	B2	00	4B	B.Y9>èp0\▒.K+♦.d
00404020	35	3E	F2	60	3D	03	C1	42	43	28	0B	00	5>≥`=♥┴BC(♂.d5>≥
00404030	60	3D	03	C1	42	43	28	0B	3B	6D	00	75	`=♥┴BC(♂;m.u98╥z
00404040	38	1A	EC	4B	69	2E	04	1B	5B	DD	B4	38	8→∞Ki.♦+[▌┤8Ä.h+
00404050	19	E8	78	3D	1F	D6	47	4A	30	27	18	46	↓₹x=▼╥GJ0'↑F╛‖)‖
00404060	BA	12	00	74	2E	29	E3	59	35	0D	F2	4F	5D	3E	00	71	‖↕.t.)πY5.≥O]>.q
00404070	2E	29	E7	61	39	3F	F2	41	4C	22	1B	0D	54	00	75	39	.)τa9?≥AL"←.T.u9
00404080	38	D2	7D	2E	0A	E1	4A	6C	28	06	0A	70	C4	AD	00	60	8╥}..ßJl(♠◘◙p─¨`
00404090	39	2D	E2	45	2E	00	E3	4B	5C	34	25	1B	78	D3	AB	24	9-ΓE..πK\4%←x╙½$
004040A0	00	64	35	3E	F2	60	3D	03	D1	5B	4A	35	11	3B	6D	00	◘d5>≥`=♥╤[J5◄;m.
004040B0	64	35	3E	F2	60	3D	03	D0	5C	40	33	0D	1D	61	F9	A1	d5>≥`=♥╨\@3.↔a·¡
004040C0	00	65	2E	25	F2	70	0C	1D	EF	4D	4A	34	1B	33	70	D1	.e.%≥p.↔∩MJ4←3p╤
004040D0	B6	2F	A6	00	61	39	38	D2	7D	2E	0A	E1	4A	6C	28	06	╢/ª.a98╥}..ßJl(♠
004040E0	0A	70	C4	AD	00	60	39	3F	F3	78	39	3B	E8	5C	4A	26	◙p─¡.`9?≤x9;è\J&
004040F0	0C	00	71	30	23	F5	70	14	0E	EE	4A	43	22	00	66	39	♀.q0#Jp¶♫ɛJC".f9
00404100	3E	EB	7C	32	0E	F4	4B	7F	35	07	1D	70	CF	AA	00	64	>δ¦2♫⌠K⌂5•↔p⌐¬.d
00404110	35	3E	F2	60	3D	03	C6	5C	4A	22	00	61	30	29	E3	65	5>≥`=♥╞\J".a0)πe
00404120	00	61	39	38	CA	7A	3F	0E	EC	7A	46	2A	0D	00	75	39	.a98╩z?♫ìzF*♪.u9
00404130	38	CA	7A	3F	0E	EC	7A	46	2A	0D	00	60	28	20	C5	7A	8╩z?♫ìzF*♪.`(┼z
00404140	31	1F	F2	4B	5C	34	2A	0B	73	DA	BC	2F	00	60	28	20	1▼≥K\4*♂s┌╝/.`(

Address	Hex dump												ASCII
00404000	42	61	64	4A	6F	6B	65	00	6E	74	64	6C	6C	2E	64	6C	BadJoke.ntdll.dl
00404010	6C	00	6B	65	72	6E	65	6C	33	32	2E	64	6C	6C	00	56	l.kernel32.dll.V
00404020	69	72	74	75	61	6C	41	6C	6C	6F	63	00	56	69	72	74	irtualAlloc.Virt
00404030	75	61	6C	41	6C	6C	6F	63	45	78	00	47	65	74	4D	6F	ualAllocEx.GetMo
00404040	64	75	6C	65	46	69	6C	65	4E	61	6D	65	41	00	5A	77	duleFileNameA.Zw
00404050	55	6E	6D	61	70	56	69	65	77	4F	66	53	65	63	74	69	UnmapViewOfSecti
00404060	6F	6E	00	46	72	65	65	4C	69	62	72	61	72	79	00	43	on.FreeLibrary.C
00404070	72	65	61	74	65	50	72	6F	63	65	73	73	41	00	47	65	reateProcessA.Ge
00404080	74	54	68	72	65	61	64	43	6F	6E	74	65	78	74	00	52	tThreadContext.R
00404090	65	61	64	50	72	6F	63	65	73	73	4D	65	6D	6F	72	79	eadProcessMemory
004040A0	00	56	69	72	74	75	61	6C	51	75	65	72	79	45	78	00	.VirtualQueryEx.
004040B0	56	69	72	74	75	61	6C	50	72	6F	74	65	63	74	45	78	VirtualProtectEx
004040C0	00	57	72	69	74	65	50	72	6F	63	65	73	73	4D	65	6D	.WriteProcessMem
004040D0	6F	72	79	00	53	65	74	54	68	72	65	61	64	43	6F	6E	ory.SetThreadCon
004040E0	74	65	78	74	00	52	65	73	75	6D	65	54	68	72	65	61	text.ResumeThrea
004040F0	64	00	43	6C	6F	73	65	48	61	6E	64	6C	65	00	54	65	d.CloseHandle.Te
00404100	72	6D	69	6E	61	74	65	50	72	6F	63	65	73	73	00	56	rminateProcess.V
00404110	69	72	74	75	61	6C	46	72	65	65	00	61	30	29	E3	65	irtualFree.a0)πe
00404120	00	53	65	74	4C	6F	63	61	6C	54	69	6D	65	00	47	65	.SetLocalTime.Ge
00404130	74	4C	6F	63	61	6C	54	69	6D	65	00	60	28	20	C5	7A	tLocalTime.`(┼z
00404140	31	1F	F2	4B	5C	34	2A	0B	73	DA	BC	2F	00	60	28	20	1▼≥K\4*♂s┌╝/.`(

b. **What is the very large basic block doing?**

The basic block is doing all of the decoding. It seems to be decoding names of libraries to be used later on in 0x4023D0, where these strings are used in function calls to *LoadLibraryA*. The block is large because of the lack of control-

flow logic.

3. **Looking at the subroutine at 0x004023D0:**
   a. **What are all of the *GetProcAddress* calls doing?**

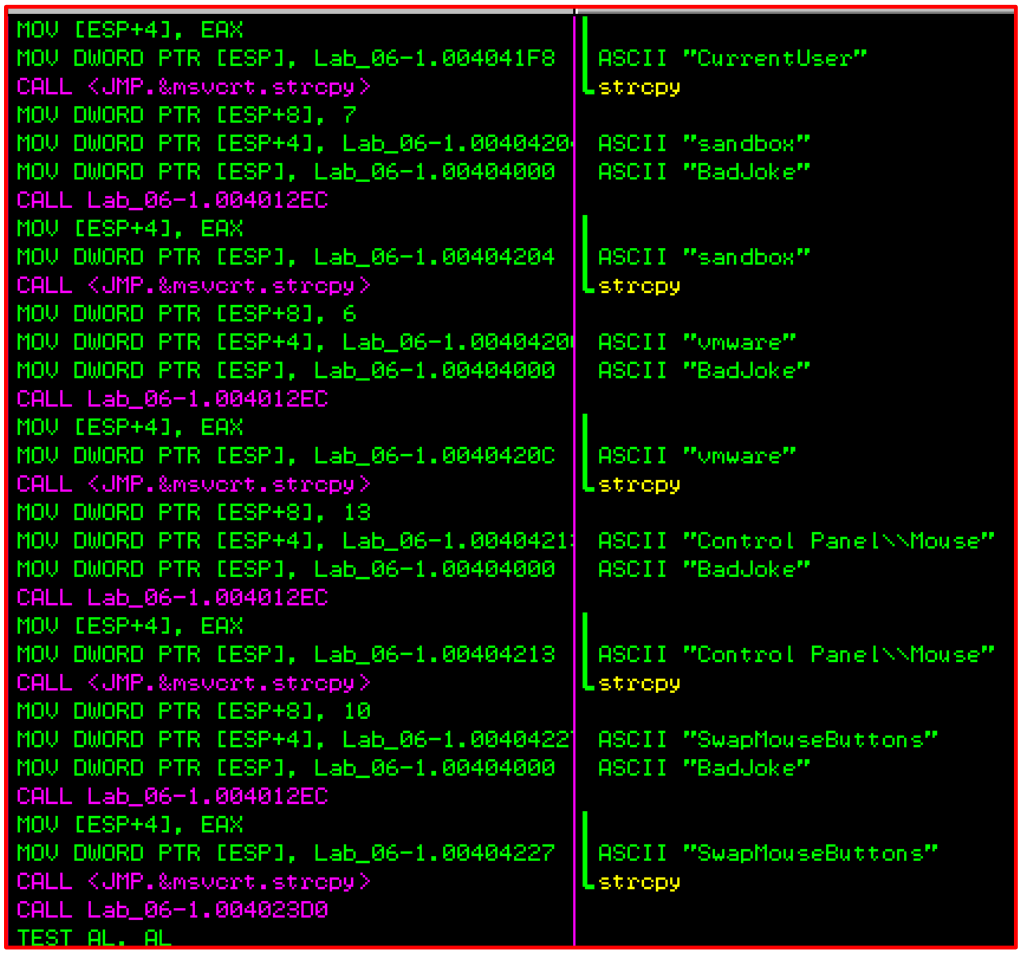
      The *GetProcAddress* calls are loading the function strings decoded by the decoding routine described in question 2.

   b. **What does this function do?**

      If the functions load successfully, it returns 1. If it fails at any point, it returns 0 and calls *FreeLibrary*.

4. **What does this sample do?**

After decoding the necessary strings in the .data section, this malware checks to see if it is in a sandbox using the strings ‘sandbox’ and ‘vmware’ and comparing these to the username of the current user. If it does detect a sandbox, it switches the left and right mouse buttons using the registry key *Control Panel\Mouse\SwapMouseButtons*.


MOV [ESP+4], EAX
MOV DWORD PTR [ESP], Lab_06-1.004041F8
CALL <JMP.&msvcrt.strcpy>
MOV DWORD PTR [ESP+8], 7
MOV DWORD PTR [ESP+4], Lab_06-1.0040420
MOV DWORD PTR [ESP], Lab_06-1.00404000
CALL Lab_06-1.004012EC
MOV [ESP+4], EAX
MOV DWORD PTR [ESP], Lab_06-1.00404204
CALL <JMP.&msvcrt.strcpy>
MOV DWORD PTR [ESP+8], 6
MOV DWORD PTR [ESP+4], Lab_06-1.0040420
MOV DWORD PTR [ESP], Lab_06-1.00404000
CALL Lab_06-1.004012EC
MOV [ESP+4], EAX
MOV DWORD PTR [ESP], Lab_06-1.0040420C
CALL <JMP.&msvcrt.strcpy>
MOV DWORD PTR [ESP+8], 13
MOV DWORD PTR [ESP+4], Lab_06-1.0040421
MOV DWORD PTR [ESP], Lab_06-1.00404000
CALL Lab_06-1.004012EC
MOV [ESP+4], EAX
MOV DWORD PTR [ESP], Lab_06-1.00404213
CALL <JMP.&msvcrt.strcpy>
MOV DWORD PTR [ESP+8], 10
MOV DWORD PTR [ESP+4], Lab_06-1.0040422
MOV DWORD PTR [ESP], Lab_06-1.00404000
CALL Lab_06-1.004012EC
MOV [ESP+4], EAX
MOV DWORD PTR [ESP], Lab_06-1.00404227
CALL <JMP.&msvcrt.strcpy>
CALL Lab_06-1.004023D0
TEST AL, AL
ASCII "CurrentUser"
strcpy
ASCII "sandbox"
ASCII "BadJoke"
ASCII "sandbox"
strcpy
ASCII "vmware"
ASCII "BadJoke"
ASCII "vmware"
strcpy
ASCII "Control Panel\\Mouse"
ASCII "BadJoke"
ASCII "Control Panel\\Mouse"
strcpy
ASCII "SwapMouseButtons"
ASCII "BadJoke"
ASCII "SwapMouseButtons"
strcpy