potassium / ARTeam

A Security Aspect Of USB Flash Drives

Version 1.1
September 2007

Table of Contents

1. Forewords

Have you ever asked yourself these questions?

- *How safe is the password function of my USB flash drive (UFD)?*
- *What if I loose or forget my password, then what?*
- *If my UFD is lost or stolen, can someone else access my data?*

If so, continue reading this document and you will hopefully get some answers!

This paper will focus on *four* different USB flash drives and the different software that is distributed with the UFD in question. It also includes a brief analysis of how safe they are. Or should I say "how unsafe they are?!". As the software that I have tested does not use encryption a simple patch may sometimes do the trick and provide us with the real password.

Sometimes those handy devices, which we rely on so much to keep our work portable and safe, are NOT always as safe as you would wish them to be. Using a ring3 debugger (OllyDbg) the communication between the protection software and the flash drive is easily intercepted. If the data, sent between the UFD and the computer, is just plain text, security could be totally compromised when monitoring the data via the debugger. This is both positive AND negative; the upside is that if you really have lost/forgotten your password, it MIGHT be retrievable (if you have to knowledge). On the downside, if someone wants to snoop around on your "protected" section of your UFD you could be, depending on choice of software, VERY poorly protected. As a special bonus I have decided to bundle this paper with my password recovery tool, the "UFD Password Revealer v1.2".

Enjoy your read,
potassium / ARTeam

Disclaimers

All code included with this tutorial is free to use and modify; we only ask that you mention where you found it. This tutorial is also free to distribute in its current unaltered form, with all the included supplements.

All the commercial programs used within this document have been used only for the purpose of demonstrating the theories and methods described. No distribution of patched applications has been done under any media or host. The applications used were most of the times already been patched, and cracked versions were available since a lot of time. ARTeam or the authors of the paper cannot be considered responsible damages the companies holding rights on those programs. The scope of this tutorial as well as any other ARTeam tutorial is of sharing knowledge and teaching how to patch applications, how to bypass protections and generally speaking how to improve the RCE art. We are not releasing any cracked application.

Verification

ARTeam.esfv can be opened in the ARTeamESFVChecker to verify all files have been released by ARTeam and are unaltered. The ARTeamESFVChecker can be obtained in the release section of the ARTeam site: http://arteam.accessroot.com/releases/

Table of Contents

1. TakeMS – Protection? Where?

1.1. Hardware

First victim is a USB flash drive (UFD) from TakeMS (1 Gb, fig 1.1), which supports a public and a "secure" partition that was setup with the software that came with the device.

Fig 1.1 TakeMS 1 Gb stick

1.2. Methodology

The first UFD to be examined was the TakeMS stick. So load up the protection software included with the stick (CarryItEasy from cososys.com) and assign a password (ARTeam) and a password reminder (Who owns?) to the protected partition. Unplug and re-plug the UFD and re-run CarryItEasy. This time you will be asked to enter a password. See figure 1.2.

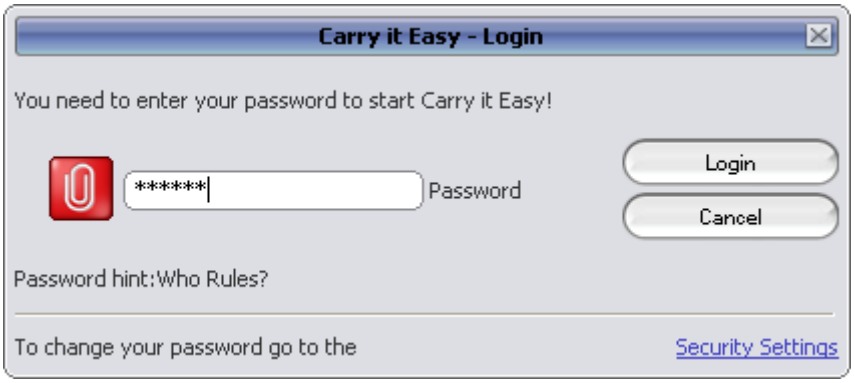

Figure 1.2 Password dialog of CarryItEasy. Yeah, Who Rules? BGates? Nah.

Since this application launches a copy of itself in a temporary folder and re-launches with CreateProcess, we need to attach OllyDbg to the newly created process. So launch our good friend Olly!

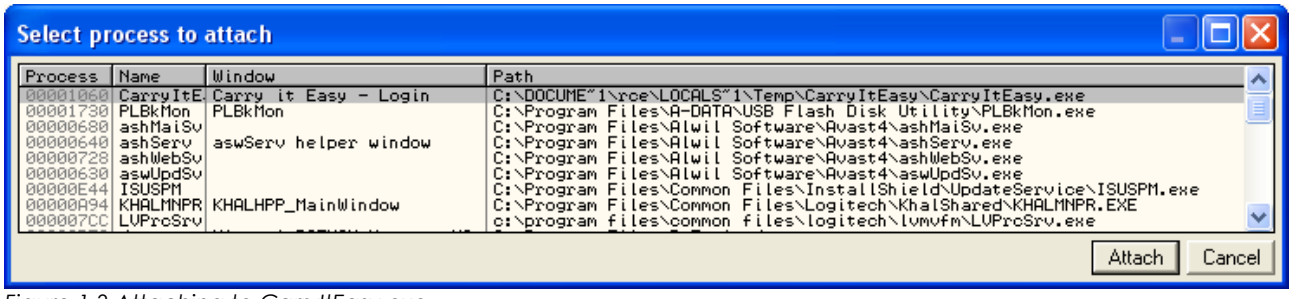

Figure 1.3 Attaching to CarryItEasy.exe

There it is. Press the "Attach" button and then press "F9" to continue running the application. Now, set a break-on-access bp on the code section of CarryItEasy and press the login dialog, you will now, hopefully, end up somewhere in the running code. Search for "All intermodular calls" and find the calls to DeviceIoControl (for more info, consult MSDN) and set breakpoints on all of them.

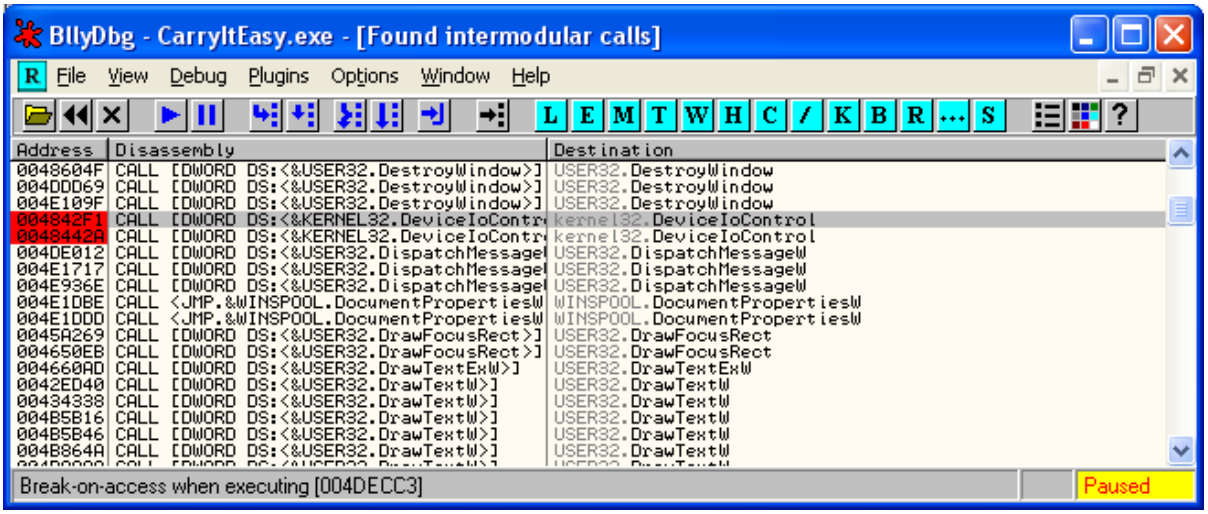

Figure 1.4 Setting breakpoints on DeviceIoControl

With this done, return to the login dialog and enter any password e.g BGates :) and press the "login"-button. Now OllyDbg will break here:

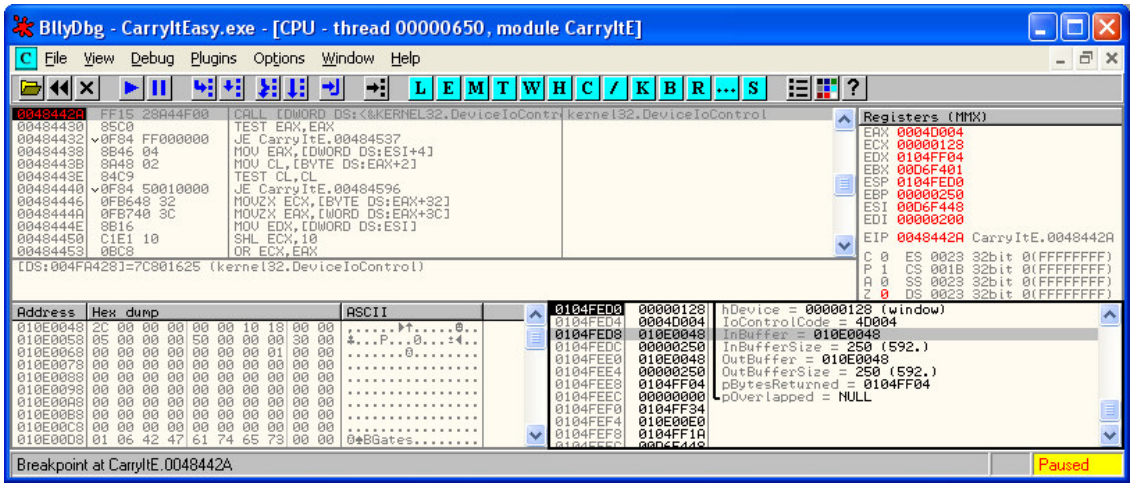

Figure 1.5 Break at DeviceIoControl with fake password

Follow the "InBuffer" in dump and you will see our input password "BGates". Press "F9" one time and break again on DeviceIoControl. Now press "F8" and check the place where the text "BGates" was before! Now it displays your real password! ARTeam (of course)

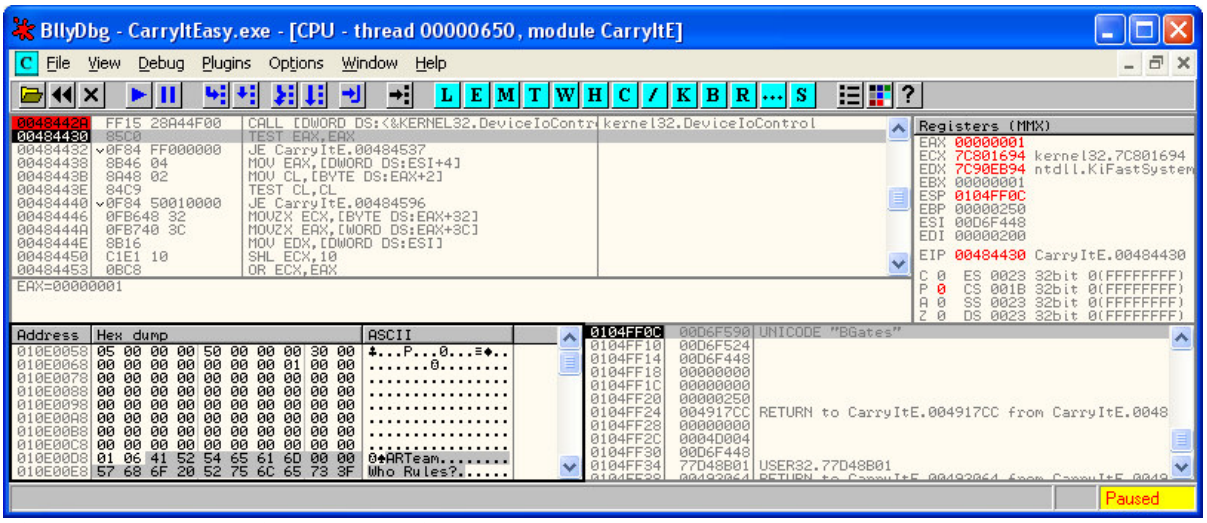

Figure 1.6 Break at DeviceIoControl with the real password

Now we need not to know more. Remove the bp's and let the application run freely. Enter "ARTeam" as password and.. Voilá! You now have complete access to everything that resides inside the so-called "protected" partition. Now this will of course satisfy the needs for some reverse engineers. But I want to take it a step further. How about modifying CarryItEasy to show the real password instead of the password hint?!

1.3. Patching protection application to reveal the password

My thought was to modify the application CarryItEasy. When you press the "Password Reminder" button, the reminder, which is read from UFD during launch, is shown in the dialog window. Problem is that if we are going to reverse engineer this application we will certainly need more than one try at the reminder button and since it becomes hidden directly after pressing.. Ummm. It needs some improvements :) The following section of code hides the reminder button.

```
004DFD13   FF7424 04        PUSH [DWORD SS:ESP+4]
004DFD17   FF71 1C          PUSH [DWORD DS:ECX+1C]
004DFD1A   FF15 28A74F00    CALL [DWORD DS:<&USER32.ShowWindow>]
004DFD20   C2 0400          RETN 4
```

Exchanging PUSH [DWORD SS:ESP+4] (0 = SW_HIDE) with PUSH 1 (1 = SW_SHOWNORMAL) will let us investigate further ^_^

Figure 1.7 Yeah, who rules? Button is still there :)

Now restart the CarryItEasy.exe and reattach OllyDbg, the breakpoints set earlier will still be there. Now, follow the "InBuffer" in dump just as before. When you see the "Who Rules?" text set a hardware breakpoint on write on the first char of the text. Continue to execute with "F9". Then you will break in kernel32.dll for a while and then return to CarryItEasy.exe, here:

```
00491C82   51              PUSH ECX
00491C83   56              PUSH ESI
00491C84   57              PUSH EDI
00491C85   50              PUSH EAX
00491C86   E8 D53CFFFF     CALL CarryItE.00485960
00491C8B   8B55 FC         MOV EDX,[DWORD SS:EBP-4]
00491C8E   8B7D 08         MOV EDI,[DWORD SS:EBP+8]
00491C91   03D2            ADD EDX,EDX
00491C93   8BCA            MOV ECX,EDX
00491C95   8BF0            MOV ESI,EAX
00491C97   8BC1            MOV EAX,ECX
00491C99   C1E9 02         SHR ECX,2
00491C9C   F3:A5           REP MOVS [DWORD ES:EDI],[DWORD DS:ESI]
00491C9E   8BC8            MOV ECX,EAX
00491CA0   83E1 03         AND ECX,3
00491CA3   F3:A4           REP MOVS [BYTE ES:EDI],[BYTE DS:ESI]
00491CA5   8B7B 04         MOV EDI,[DWORD DS:EBX+4]
00491CA8   81C7 A0000000   ADD EDI,0A0
00491CAE   8955 FC         MOV [DWORD SS:EBP-4],EDX
00491CB1   75 04           JNZ SHORT CarryItE.00491CB7
00491CB3   33C0            XOR EAX,EAX
00491CB5   EB 29           JMP SHORT CarryItE.00491CE0
00491CB7   57              PUSH EDI
00491CB8   FF15 24A44F00   CALL [DWORD DS:<&KERNEL32.lstrlenA>
00491CBE   8BF0            MOV ESI,EAX


Registers:

EDX 010E14F9 ASCII "ho Rules?"
ESP 0104FEF0 UNICODE "ARTeam"
EDI 010E14F8 ASCII "Who Rules?"
```

As you can plainly see the correct password is currently stored as pointer to a UNICODE string in ESP. Setting a breakpoint @ 00491C82 reveals something interesting. The call at 00491C86 converts the password in ASCII format to UNICODE format, which suits us just fine :). At 00491CA8 something of interest caught my eye. EDI is a pointer to the "InBuffer" (read from UFD), adding 0xA0h to the starting point of the buffer will point to the string "Who Rules?" and then the code goes on in similar fashion, convert ASCII to UNICODE etc. Changing the ADD EDI, 0A0 to ADD EDI,092, just like previous procedure above, will then point to the ASCII string "ARTeam" and convert it to a UNICODE string. Now pressing of the reminder button is so much nicer. :D Make things easy on yourself now, write the one-byte patch to disk to make things permanent and you're all done!

Figure 1.8 The real password is now pwnd!!