

Theories and methods of memory hacking

(By Faldo)

Intro:

The purpose of this tutorial is to awaken a manner of thinking in order to understand **basic memory hacking**. I believe that teaching people how to hack instead of giving them compiled hacks is the best way to keep as many as possible undetected from the bloodthirsty developers, admins and ban lists. A lot of tutorials tell you what to do if you want to make a hack, but never explains why, which may be the key to everything.

I will, in chronicle order go through theory and method of what to do and how to do it, including as much detail as possible so that even my grandma would be able to pull it off ;o)

At the end of this tutorial, you will have accomplished what many dream about and many more get jealous of: Your own private hack. The knowledge you are about to take part of will in theory be applicable on any offline or online game you encounter. Your way of thinking is the only obstacle to your success. There are some things you will not be able to hack, since they are out of your reach, like server side code. However, I'll be making a tutorial on theory and methods of packet editing right after this one so stay tuned.

Index:

-

1. In theory

1.1 The beginning

1.2 Everything resides inside the memory:

1.3 DrunkBusters

1.4 Conclusion

2. The methods

2.1 Tools:

2.2 Battlefield 1942 and mods:

2.3 Window mode

2.4 Finding the right address

2.4.1 Minimap-hack theory

2.4.2 Minimap-hack method

2.4.3 Tag-hack theory

2.4.4 Tag-hack method

2.4.5 Fog-hack theory

2.4.6 Fog-hack method

2.5 Making a trainer

2.5.1 Basics of making a trainer in

TMK

2.5.2 Debugging with Ollydbg

2.5.3 Converting ASM from

Ollydbg/T-Search to TMK

2.5.4 Creating POKEs for minimap

and tag-hack

[2.5.5 No need to convert fog-hack](#)

[address](#)

2.6 Creating a trainer in Tmk

[2.6.1 Making the minimap and tag-](#)

[button](#)

[2.6.2 Making the Remove fog-button](#)

2.7 Hiding treasures in code-caves

[2.7.1 Fool the game](#)

[2.7.2 First step, create the code-cave](#)

[2.7.3 Second step, create the jump-](#)

[gate](#)

3. F.A.Q.

1. In theory

[1.1 The beginning](#)

I'm going to take Battlefield 1942 as an example.

The first step is asking ourselves what we want to hack. Since only the game developer knows exactly how the game works, the best way to find loopholes in which we can hack, is to search and try. Many online games need to keep the Internet traffic at a very low rate. In order to keep the game floating and letting people with 56k modems play, the server only sends small triggers instead of many lines of code. These triggers tell the client what to do, and the client does the rest. Here is a hack opportunity. We need to alter the code triggered by the server in order to fool it.

Let's decide that we want to see the team members of the opposite side, since that would be a great advantage in a game like BF1942. In this particular example, we eliminate the code that defines and shows us what team we are on and the result being we see both teams' locations.

[1.2 Everything resides inside the memory](#)

The most common hacks use memory hacking. Others change the way the computer runs the game by altering the game itself, the latter involves pretty complex programming so no point in explaining that here.

Whenever a program (the game) runs on your computer, it uses RAM (Random Access Memory). Either your hardware memory modules or a page-file created by windows. In any case, the program "injects" code into the memory to be processed (read from and written to) since the program itself is already compiled and cannot change.

[1.3 DrunkBusters](#)

18 EvenBalance employees are doing what they can to stop cheaters, while many hundred more are doing what they can to get around their well know program: **PunkBuster**. To the disadvantage of EB, they can never predict what methods hackers are going to use. Because of this, I once again point out that making private hacks is the best way to stay undetected.

In this tutorial I'm going to explain how to make a code-cave. The basic function of a code-cave is to make the game read a copy of the original code, while PunkBuster scans that original code for hacks. The copy we made is located at a completely different place, where PB does not scan, so we can go ahead and hack whatever we want in that code.
Here is an example how this would look:

The memory is read by the game from top to bottom in chronicle order.

Address 676:Codeline
Address 677:Codeline
Address 678:Codeline
Address 679:Codeline scanned by PB
Address 680:Codeline scanned by PB
Address 681:Codeline scanned by PB → this is the code we need to change
Address 682:Codeline scanned by PB
Address 683:Codeline

If we change any code in the code-lines scanned by PB it will be detected.

To solve this we need to recreate the original code in a different location like this:

Address 676:Codeline
Address 677:Codeline telling the game to jump to address 1035
Address 678:Codeline
Address 679:Codeline scanned by PB
Address 680:Codeline scanned by PB
Address 681:Codeline scanned by PB → this is the code we need to change
Address 682:Codeline scanned by PB
Address 683:Codeline
Address 684:Codeline

This is our code-cave:

Address 1035:Copy of code-line in Address 678
Address 1036:Copy of code-line in Address 679
Address 1037:Copy of code-line in Address 680
Address 1038:NOP → we changed the code to our own
Address 1039:Copy of code-line in Address 682
Address 1040:Codeline telling the game to jump to address 683

As you can see, we told the game to jump from address 677 to 1035, execute the copied code, replace one code-line with a NOP and then tell the game to jump back, right after the scanned

addresses. This way, PB can scan those addresses all they want without detecting any changes.

1.4 Conclusion

We now know what we want to do and how our game uses the memory to run, so all there is left to do is change the memory so that the game acts the way we want.

The methods following, will explain, step by step how to search the memory for the code we need in order to hack what we want, then how to change it, how you stay undetected from PB scans, and finally how to compile your own hack. Best of all, you'll manage all this without any or very little knowledge of programming.

2. The methods

2.1 Tools

Since I'm going to presume that you don't have any programming skills, and to keep this tutorial as compatible as possible, I'm only going to explain and use very little program language in it.

Therefore we will need a couple of tools that will do all the programming for us:

- T-Search 1.6b (http://membres.lycos.fr/tsearch/tsearch_16b.zip)
- Ollydbg 1.10 (<http://home.t-online.de/home/Ollydbg/odbg110.zip>)
- Trainer Maker Kit 1.51
(http://membres.lycos.fr/tsearch/tmk_151.zip)

2.2 Battlefield 1942 and mods

I'm going to explain the method on how to hack the minimap, tags and fog since I think these are the most important hacks, the rest will just make the game too easy and take away the fun of it. But if you understood the theory above and when you have completed the methods below, you can hack pretty much anything you want (accuracy, zoom etc.).

2.3 Window-mode

To save some time, it's recommended that you set BF1942 in windows mode before you start hacking. This way you can work with T-search/Ollydbg in the foreground and have BF1942 in the background. To do this, find a file called "VideoDefault.con" located in "Battlefield 1942\Mods\bf1942\Settings"-folder.

Open this file with notepad and change the line:

renderer.setFullScreen 1

to

renderer.setFullScreen 0

2.4 Finding the right address

2.4.1 Minimap-hack theory

We are going to use T-search to find two addresses that contains code that checks what side we are on. Then by eliminating this check we fool the game in thinking we are on both sides. Since

these addresses we will first find is a dynamic addresses (*Dynamic Memory Allocation* = it changes place every time a map is loaded) we need to find the static addresses that writes and reads to and from their DMA. In other words we track down the address from which the DMAs originates; I call the static address the "anchor". In T-search this tracking function is called "auto-hack". In hacking terms, it's called "break-pointing". The two addresses, which we need to find, are: one for infantry and one for vehicles, so it's recommended that you use a map like Bocage where there are both.

2.4.2 Minimap-hack method

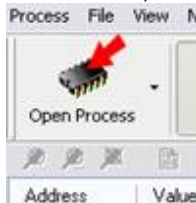
1. Start BF1942 (or mod).
2. Create a multiplayer LAN game (not internet), preferably Co-Op so you can see AI-players.
3. Join the game and check what team you are on. Red is axis and blue is allies.

If you are on the red/axis team you will have a value of 1 at a specific address, if you're on blue/allies team, that value will be 2. So it's better if you start of as allies since there are less 2s than 1s and will save some time when searching for values. This will be explained later.

4. Pause the game, alt+tab out to windows and start T-search.

If you "loose" your mouse cursor, click anywhere outside the BF1942 window.

5. In T-Search, click "Open Process" and choose BF1942.exe



6. Click the magnifying glass under the "Open process" button and you'll see the Search window.



Enter this:

Search: Exact Value

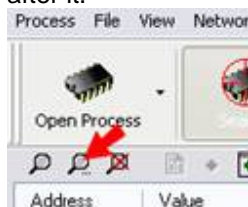
In the **Value**-field, enter: 2 as value if you are allies/blue or 1 as value if you are axis.

Type: 4 Bytes



Click Ok

7. You'll see t-search scanning through your memory (takes a few seconds).
8. When it's done, click Ok
9. Alt+tab back to the game, unpause it and change team by clicking the "suicide" button and pause the game once spawned as opposite team.
10. Alt+tab back to T-search and click the button next to the magnifying glass that looks the same but has three dots after it.



You will see the window "Search next". Enter this:

Search: Exact Value

In the **Value**-field, enter: 1 as value if you changed to axis, or 2 as value if you changed to allies.

Type: 4 Bytes

Click Ok

T-search will look through your memory again and search for values that have changed from 2 to 1 (or from 1 to 2 depending on what you searched for).

11. After clicking Ok you should end up with fewer addresses than you had in your first search.

But since you still have quite many addresses you need to narrow it down even more.

12. Redo everything from point 9 until you end up with less than 30 addresses (i.e. 1EF24D2=address) in the left upper window in T-search.
13. Click on the icon with the green "+" sign to move over all the addresses you found to the "Cheat list" window on the right hand side, this is where you can change the values.



This part is where you will save time by having the T-search window in the foreground and the BF1942 window in the background where you can see the minimap.

14. While watching BF1942 in the background, change the value of an address to 1 (if it is already a 2) or 2 (if it is already a 1).

What you are looking for is a change in the minimap where your arrow changes color to the opposite teams color. You will also notice that you are able to see the locations of the other team as well as if you look on the scoreboard, you moved to the other team's side.

15. Once you found the address that changes the teams (let's call it the Team address), change it to the original value and copy the address to your clipboard.

The address you found is a kind of trigger for other addresses that use it as reference to know what team you are on. Since the goal is not to change team but to change what you see, you need to break this address down into smaller parts/operations. In other words we need to find the addresses that use the team address as reference. The value you changed is only held in a DMA (explained in the theory) and will be useless to us when we start another map, so we need to find its static "anchor".

16. Choose **Enable debugger** from the **Auto-hack** menu. Then choose **Auto-hack window** from the same menu.

20. In the Auto-hack window, choose **Set breakpoint** from the **Edit** menu.

21. Paste the team address in the address field and select **Read/Write** from the dropdown field and hit **Set**.

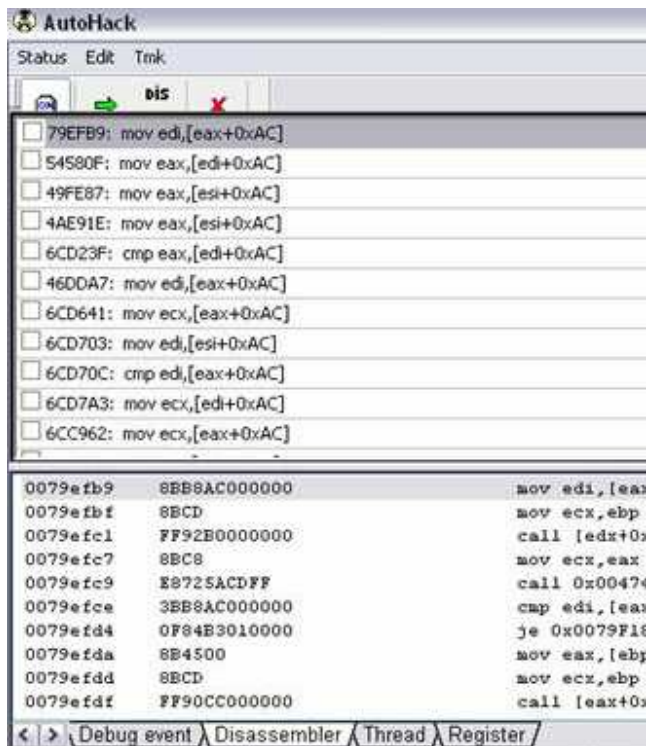
The address you see in the picture below is most likely not the address you have.



22. Alt+tab back to BF1942 window, unpause the game and change sides at least 2 times.

While you play the game, T-search looks for every address that reads or writes to or from our team address.

23. Pause the game and Alt+tab back to the auto-hack window. You should now have a list in the upper part displaying address followed by a code.



0x0079EFB9 BF1942.exe: dice::matchmaking::IHostService::IHostServ Address:18

This code (ASM) defines all the operations happening in the game (i.e. if you shoot someone, you die, you enter a vehicle etc.). Whatever you do, will affect one or many addresses containing the operation codes. Some codes change and some stay static.

This is our chance to change the behavior of the game; in this case we need to eliminate the code defining what team we see. Eliminating something in the code is commonly done with a NOP (No Operation).

A typical address line would look something like this:

66F2D3: cmp eax, [ecx+0x2] → lets pretend this is the line that checks what team we are see.

To eliminate this check, the same line would look like this:

66F2D3: NOP → When the game reads this line it, it will do nothing, in other words: It will not check what team we are on.

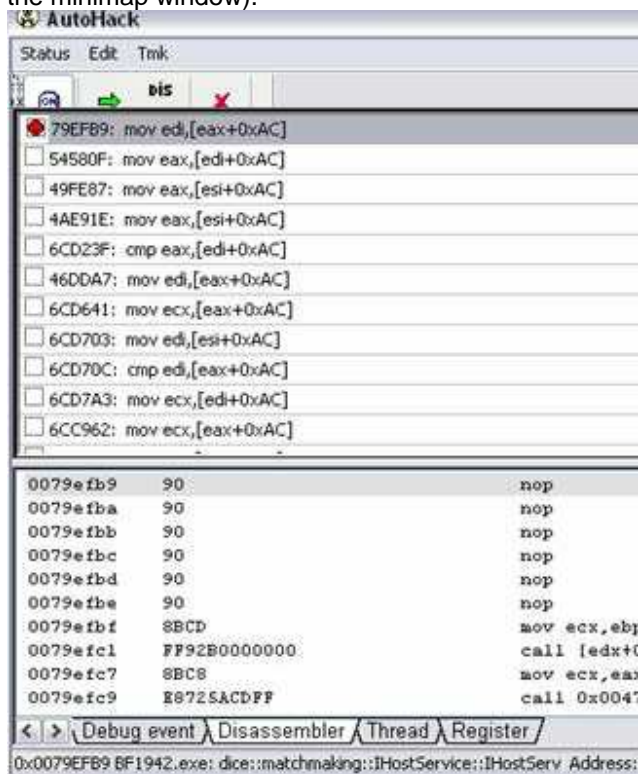
One of the addresses you see in the list holds the code you need to NOP.

Since I will not yet explain how to enter a NOP into the address we will let T-search do this for us.

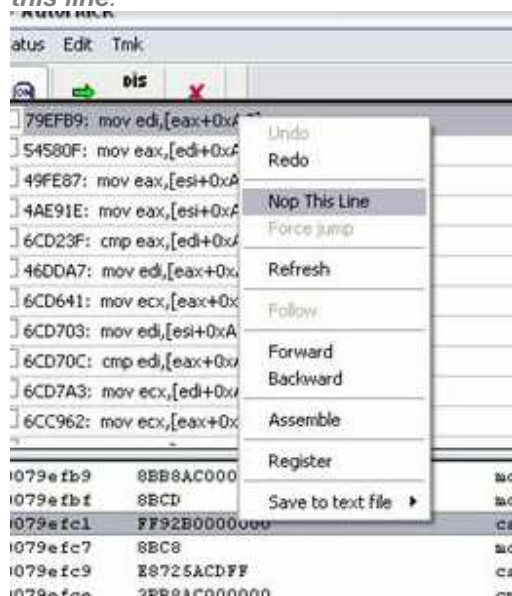
The list you see is not the entire list of addresses, only the “beginning address” for a specific portion of code. By clicking on one of the addresses you will notice another list appearing in the bottom half of the window, this list is the entire list of addresses following the “beginning address”. It’s like unfolding a directory in windows explorer if you click the “+” sign, letting you see its content.

24. By checking the checkbox left of the address you will NOP that address. So check the box for each address (one at a

time), while watching the BF1942 window in the background. Once you checked the right address you will get the desired effect (in this case you'll see both teams in the minimap window).



Since each “beginning address” might not produce the exact effect you need, you can also NOP a specific address inside the portion of code by right-clicking that address in the bottom half of the window and choose **NOP this line**.



This is the most time consuming part if you don't know what you are looking for. And to be honest, if you don't know ASM programming, you just won't find it as fast. The beauty of it is that you don't need to know any programming language; it will just take a little while longer than if you did. You know that all the addresses in that list has to do with your team address somehow, you just need to find the right one.

NOTE: There is one address for infantry and another for vehicles.

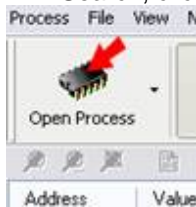
When you find the right addresses they will be a static addresses, meaning they will never change (unlike the DMA). Write these addresses down since you can produce a trainer using them later on in this tutorial.

2.4.3 Tag-hack theory

We are going to use T-search to find an address that contains code that checks if you see the enemy tags or not. Then by eliminating this check we fool the game in thinking we are on both sides. Since the address we will first find is a dynamic address (Dynamic Memory Allocation = it changes place every time a map is loaded) we need to find a static address that writes and reads to and from the DMA. In other words we track down the address from which the DMA originates; I call the static address the "anchor". In T-search this tracking function is called "auto-hack". In hacking terms, it's called "break-pointing".

2.4.4 Tag-hack method

1. Start BF1942 (or mod).
2. Create a multiplayer LAN game (not internet), preferably Co-Op so you can see AI-players.
3. Run around in the map until you find a player of the opposite team. Point your sight at the player to see his nametag and pause the game quickly before you get shot at.
4. In T-Search, click "Open Process" and choose BF1942.exe



5. Click the magnifying glass under the "Open process" button and you'll see the **Search** window.



Enter this:

Search: Range

In the **Value 1**-field, enter: 58 and in the **Value 2**-field, enter: 64.

The numbers 58 and 64 are values that exist in the opacity of the tag-color. Since the tags can contain different shades depending on how far you are from the player you need to enter Range values.

Type: 4 bytes



Click Ok

6. You'll see t-search scanning through your memory (takes a few seconds). When it's done, click Ok
7. Alt+tab back into the game, unpause it, take your sight off the player and pause it as soon as the enemy tag has faded completely.
8. Alt+tab back to t-search and hit the "search next"-button.



9. This time, change your **Search** field to **Exact value**, the **Value** to **0**, and the **Type**: to **4 bytes**:



Click Ok

10. T-search will once again search through your memory and hopefully end up with fewer addresses than last time. Click Ok.

11. Alt+tab back to the game and point your sight at the player of the opposite team again so that you see his tag reappear and pause the game quickly before you get shot at.
12. Alt+tab to T-search. To narrow down the number of addresses, repeat the process from point 5. but use **Search next** instead of **Search** as point 5 suggests.
13. Once you have less than 10 addresses you can start the hacking: Click on the icon with the green "+" sign to move over all the addresses you found to the "Cheat list" window on the right hand side, this is where you can change the values.



This part is where you will save time by having the T-search window in the foreground and the BF1942 window in the background where you can see the minimap.

14. Now, freeze the address, one by one, by checking the box left of the address and change its value. If it is **0** then change it to 64, if the value is anywhere between 58 and 64, just leave it.
15. Once you found the address that turns the enemy tag on (let's call it the "tag address"), unfreeze it and copy the address to your clipboard.

The address you found is a kind of trigger for other addresses that use it as reference to know if to keep the enemy tags on or off. Since the goal is not only to light the tag of the enemy you point your sight at, but to turn them all on, you need to break this address down into smaller parts/operations. In other words we need to find the addresses that use the tag address as reference. The value you changed is only held in a DMA (explained in the theory) and will be useless to us when we start another map, so we need to find its static "anchor".

16. Choose **Enable debugger** from the **Auto-hack** menu. Then choose **Auto-hack window** from the same menu.
20. In the Auto-hack window, choose **Set breakpoint** from the **Edit** menu.

21. Paste the tag address in the address field and select **Read/Write** from the dropdown field and hit **Set**.

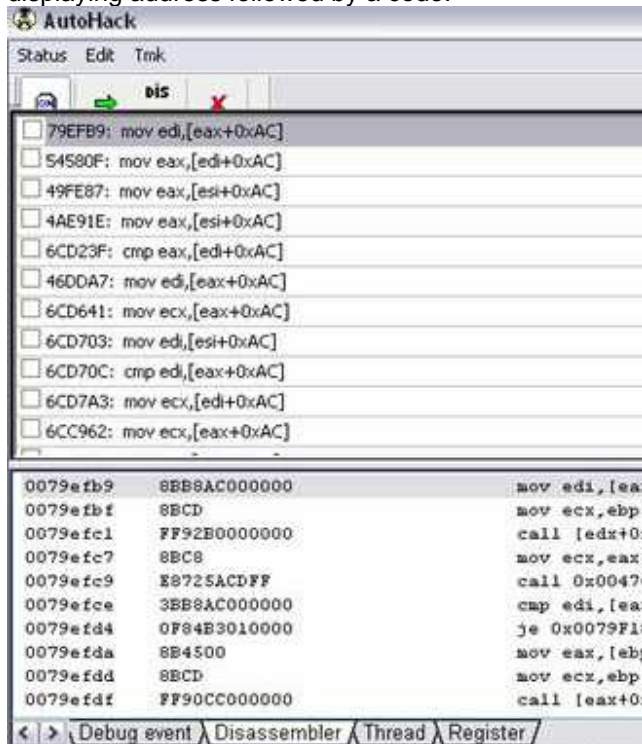
The address you see in the picture below is most likely not the address you have.



22. Alt+tab back to BF1942 window, unpause the game and play for a minute.

While you play the game, T-search looks for every address that reads or writes to or from our tag address.

23. Pause the game and Alt+tab back to the auto-hack window. You should now have a list in the upper part displaying address followed by a code.



0x0079EFB9 BF1942.exe: dice::matchmaking::IHostService::IHostServ Address:11
This code (ASM) defines all the operations happening in the game (i.e. if you shoot someone, you die, you enter a vehicle etc.). Whatever you do, will affect one or many addresses containing the operation codes. Some codes change and some stay static.

This is our chance to change the behavior of the game; in this case we need to eliminate the code defining what team we see. Eliminating something in the code is commonly done with a NOP (No OPeration).

A typical address line would look something like this:

66F2D3: cmp eax, [ecx+0x2] → lets pretend this is the line that checks if we see the enemy tag.

To eliminate this check, the same line would look like this:

66F2D3: NOP → When the game reads this line it, it will do nothing, in other words: It will not check if we see the enemy and will display it as default.

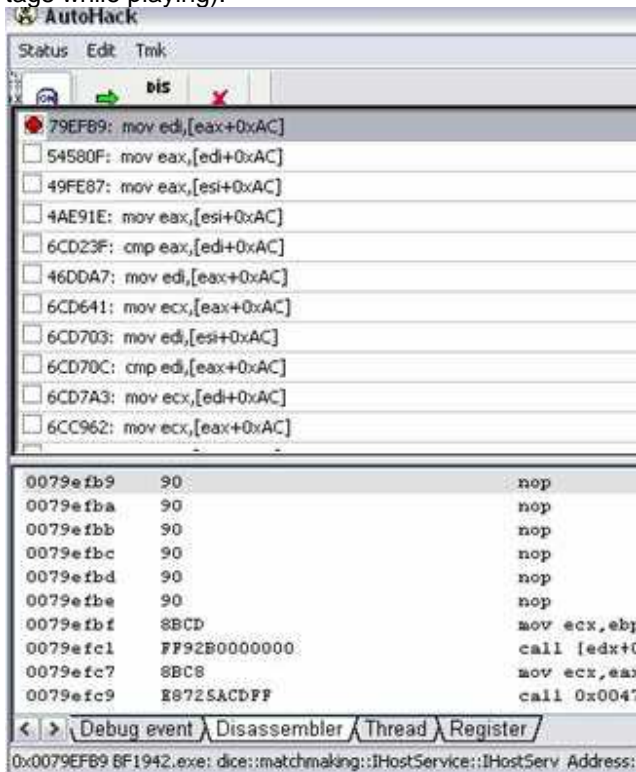
One of the addresses you see in the list holds the code you need to NOP.

Since I will not yet explain how to enter a NOP into the address we will let T-search do this for us.

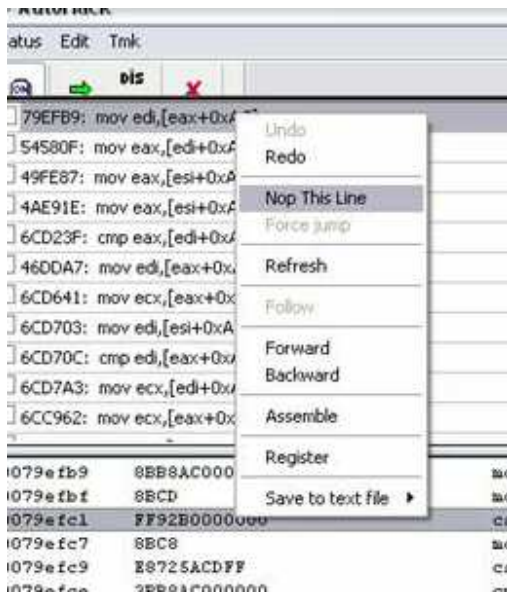
The list you see is not the entire list of addresses, only the "beginning address" for a specific portion of code. By clicking on one of the addresses you will notice another list appearing in the bottom half of the window, this list is the

entire list of addresses following the “beginning address”. It’s like unfolding a directory in windows explorer if you click the “+” sign, letting you see its content.

24. By checking the checkbox left of the address you will NOP that address. So check the box for each address (one at a time), while watching the BF1942 window in the background. Once you checked the right address you will get the desired effect (in this case you’ll see both team-tags while playing).



Since each “beginning address” might not produce the exact effect you need, you can also NOP a specific address inside the portion of code by right-clicking that address in the bottom half of the window and choose **NOP this line**.



This is the most time consuming part if you don't know what you are looking for. And to be honest, if you don't know ASM programming, you just won't find it as fast. The beauty of it is that you don't need to know any programming language; it will just take a little while longer than if you did. You know that all the addresses in that list has to do with your tag address somehow, you just need to find the right one.

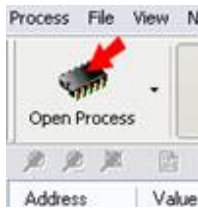
When you find the right address it will be a static address, meaning it will never change (unlike the DMA). Write this address down since you can produce a trainer using it later on in this tutorial.

2.4.5 Fog-hack theory

We are going to use T-search to find an address that contains the fog-value. The address will be DMA but only "semi-DMA", which means that it will stay static unless you reinstall the game. There are two ways of making this hack, one hard way and one easy way. The main advantage of the hard way is that any computer will be able to use it (public hack). The main advantage of the easy way is that it will be forever undetected by PB without having to create a code-cave (explained further down). Taking this notion in count, we will make the fog-hack the easy way.

2.4.6 Fog-hack method

1. Start BF1942 (or mod).
2. Start a single-player map.
3. Hit ESC and choose "Options" --> "Video"
4. Change your "View distance" to 97%
5. Alt+tab out to windows, and start T-search.
6. In T-search click "Open Process", choose BF1942.exe



7. Click the magnifying glass under the "Open process" button and you'll see the Search window.

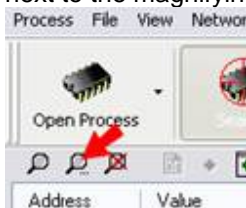


8. Enter this:
Search: Exact Value
Value: 97
Type: 4 Bytes



Click Ok

9. You'll see T-search looking through your memory (takes a few seconds).
10. When it's done, click Ok
11. Alt+tab back to the game, change the "View distance" to 37% and click "save"
12. Alt+tab back to T-search and click the "Search next" button next to the magnifying glass.



Enter this:
Search: Exact Value
Value: 37
Type: 4 Bytes
 Click Ok

13. T-search will look through your memory again and search for values that have changed from 97 to 37. After clicking Ok you should end up with very few addresses (i.e.: address=1EF24D2) in the left upper window.

14. Click on the icon with the green “+” sign to move over all the addresses you found to the “Cheat list” window on the right hand side, this is where you can change the values.



This part is where you will save time by having the T-search window in the foreground and the BF1942 window in the background where you can see the fog change.

15. While watching BF1942 in the background, change the value of an address to 200, and keep changing the addresses, one at a time, until you notice a change in the game.

What you are looking for is a change in the game where the fog disappears twice as far away as usual.

16. Once you found the address that changes fog (let's call it the Fog address), copy it to your clipboard.

The value you changed is only held in a DMA (explained in the theory) but will be very useful since it's a “semi-DMA” meaning it will be static on your computer, but will be useless to put in a public trainer, so no need to find its static “anchor”.

You now have the 4th and final address in order to continue this tutorial and make yourself a trainer.

So write this final address down and close both BF1942 and T-search.

2.5 Making a trainer

2.5.1 Basics of making a trainer in TMK

Since PB detects if you are running T-search while you play, you need to make a trainer that changes the addresses you have found for every hack. The method of this is slightly harder to understand than the above methods since you need to handle simple bits of ASM code.

2.5.2 Debugging with Ollydbg

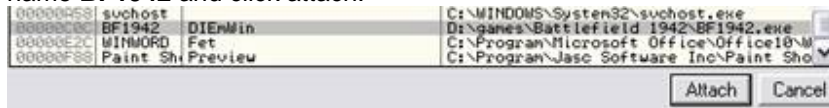
In order to make a good NOP hack you need to know 2 things:

1. The right addresses to hack
2. How many address-lines to hack, a.k.a. “balancing”.

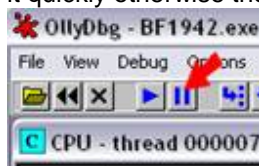
In the methods above, T-search NOPed the addresses necessary for you. Now you need to understand what T-Search did and port that knowledge over to Ollydbg for further analyze.

1. Start BF1942 (or mod).

2. Create a multiplayer LAN game (not internet), preferably Co-Op so you can see AI-players.
3. Pause the game once you spawned on either side.
4. Alt+tab out to windows and open Ollydbg.
5. Select **Attach** from the **File** menu, select the process-name **BF1942** and click attach.



WARNING: When you attach BF1942, Ollydbg will automatically pause the process once it scanned it (if you haven't set Ollydbg not to). Therefore you need to unpause it quickly otherwise the game might crash.



6. Now click on the window that says **CPU – thread...** (just to make it active) and press **Ctrl+G** to open the **Enter expression to follow**-window. Enter the one of the addresses you found in each method above and click Ok. You need to do this twice in order to get to the **Main thread**.



If the address you found in T-search only had 6 characters, add **00** in front of them just like in the picture above.

7. You should now be looking at a selected line where the address you found is displayed.

004C37B6	895424 14	MOV DWORD PTR SS:[ESP+14],EDX
004C37BA	75 04	JNZ SHORT BF1942.004C37C0
004C37BC	33FF	XOR EDI,EDI
004C37BE	EB 17	JMP SHORT BF1942.004C37D7

To the right of your address you will have a series of characters known as **Op-codes**. The

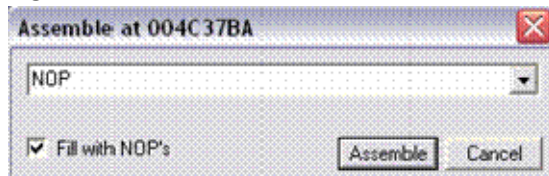
Op-codes represent the action the address makes when it's read. You don't need to know anything further about Op-codes at this point in order to continue.

The column right of the Op-code contains the translated ASM code for the address. This code is what you need to change.

8. By pressing spacebar while having a line selected you tell Ollydbg that you want to "Assemble" this line.

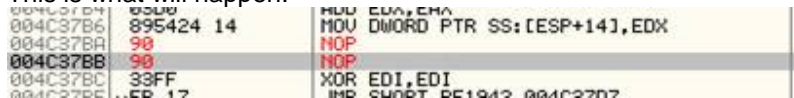


In this case, let's translate the word "Assemble" into "Change" so there is no confusion.
Where it says **JNZ SHORT 004C37C0** replace all that by **NOP**.



And click **Assemble**.

This is what will happen:



The address-line checking what side we were on:

004C37BA got an Op-code of **90** and the translated ASM code became **NOP**, just like we wanted it to. But hey, there is another line?

Now is where it gets a tad complicated: In order to NOP an address one has to know how many **bytes** the code consists of.

Our initial address looked like this:

004C37C0 75 04 (where **75 04** is the Op-code)

One byte consists of 2 hex characters. In the Op-code **75 04** there are 4 hex characters, therefore the entire Op-code uses 2 bytes.

Let's take another example. The address-line above our team address; **004C37B6** has the Op-code **895424 14**.

Since we know that 1 byte consists of 2 hex characters, then 8 characters (**89 54 24 14**) uses 4 bytes.

Here are a few more examples:

004C37D7	8B5424 18	MOV EDX,DWORD	
PTR SS:[ESP+18]		→ uses 4 bytes	
004C37DB	52	PUSH EDX	
		→ uses 1 byte	
004C37ED	8D04BF	LEA	
EAX,DWORD PTR DS:[EDI+EDI*4]		→ uses 3 bytes	
Get it?			

Now, to NOP an address using 2 bytes (like our example-address does), we need to NOP two address-lines, one for every byte.

Let's use 2 of the above examples and NOP them:

004C37D7 8B5424 18 MOV EDX,DWORD PTR
SS:[ESP+18]

Become:

004C37D7 90 NOP

```
004C37D8 90      NOP
004C37D9 90      NOP
004C37DA 90      NOP
```

```
004C37DB 52      PUSH EDX
Become:
004C37DB 90      NOP
```

Luckily, Ollydbg counts the bytes in the address you NOP and adds an appropriate amount of addresses for you. But it helps if you know how it works.

9. Select the lines that are now red, right-click them and copy them to a notepad document via the clipboard.



The pasted lines will look like this:

```
004C37BA 90      NOP
004C37BB 90      NOP
```

10. Close both BF1942 and Ollydbg since you now have the offsets you need to make your trainer.

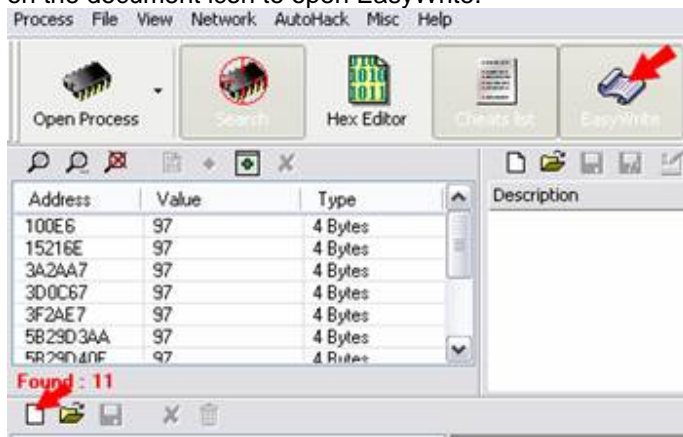
2.5.3 Converting ASM from Ollydbg/T-Search to TMK

We are approaching the final steps of this tutorial: the making of our trainer. You will need to convert the addresses and ASM codes you found in Ollydbg into a Tmk script that is later compiled into an .exe file.

2.5.4 Creating POKEs for minimap and tag-hack

The minimap and tag-hacks are typical NOP hacks; the following method explains how to create a **POKE** to be used in our TMK trainer. By poking an address we inject an offset of code that we created in order to achieve the change we need.

1. Open T-search and make sure EasyWrite is enabled. Click on the document icon to open EasyWrite.



2. In order to use the addresses we found we need to make a little change to them so that EasyWrite can convert it into a POKE.
3. We have the lines from Ollydbg:


```
004C37BA 90      NOP
004C37BB 90      NOP
```

The converted lines will look like this:

```
OFFSET 004C37BA
NOP
NOP
```

What I did was add the word **OFFSET** in front of the first address we want to **NOP**. The next line will contain the ASM code we want to put at that address-line (004C37BA). The line after that will contain the ASM code of the next address-line (004C37BB).

Here is another example:

```
006F1B81 90      NOP
006F1B82 90      NOP
006F1B83 90      NOP
006F1B84 90      NOP
```

I added two more addresses so that you will understand the conversion. This example uses 4 bytes, hence the 8 Op-code characters (90 90 90 90).

The converted lines will look like this:

```
OFFSET 006F1B81
NOP → ASM injection for address 006F1B81
NOP → ASM injection for address 006F1B82
NOP → ASM injection for address 006F1B83
NOP → ASM injection for address 006F1B84
```

See how easy that was? 😊

4. Now enter your converted lines into the upper part of the EasyWrite window.



5. Click the button **Tmk** and then **Check**.
If everything went as planned you should have a line in the right upper part that says:
Poke 4C37BA 90 90
Copy this poke to a notepad document for use with Tmk later.

2.5.5 No need to convert fog-hack address

When it comes to hacking the fog, we won't be making a NOP hack in this particular tutorial. We can enter an added value (to the original fog-value) directly into Tmk.

Note: It is very possible to make a fog-hack with NOPs, but since it's harder and can be detected by PB, I'll lead you through the easy way.

2.6 Creating a trainer in Tmk

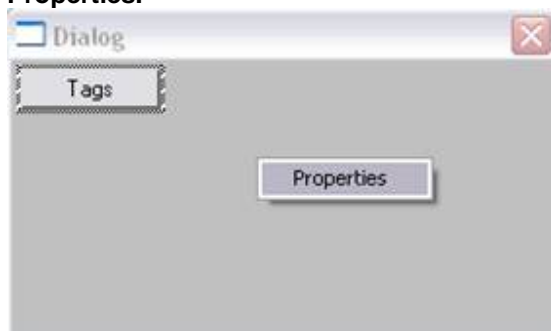
2.6.1 Making the minimap and tag-button

1. Open TMK and enter "BF 1942" as **project name** and hit "create". Click the splash screen to make it disappear.
2. Choose **Button** from the **Insert**-menu to add a button on the trainer.
3. Right click the button and choose **properties**.
4. In the **Styles**-tab, enter: **Tags** in the **Caption** field.
5. In the **Key**-tab and select **Shift** and enter **T** in the **hotkey** text field. Close the **Properties** Window.
6. Right Click the Trainer Interface and choose **Properties**.
7. Enter the poke-line you got from EasyWrite when you converted the Tag addresses.



Click **Apply**.

8. Right-click anywhere on the "trainer interface" and choose **Properties**.



9. In the **Caption**-text field enter a name for your trainer. Just to be sure, enter something harmless like: Notepad, Winmine etc.
Close the properties window.
10. Click the **Build Settings**-tab near the bottom left side of the Window.



Double-click the yellow fields and enter:

Process name: BF1942.exe

Exe name: Notepad (or whatever name you chose)

Exe type: Linked

11. Choose **Save project** from the **File**-menu.
12. Choose **Build your project** from the **Build**-menu. Close Trainer Make Kit.

Your trainer will now be in the Trainer Maker Toolkit folder (wherever you installed it).

You have now made your first button in you private hack, I suggest you try it in a created multiplayer LAN map to know if you managed. If it doesn't work, send your complaints to: support@ea.com ;o)

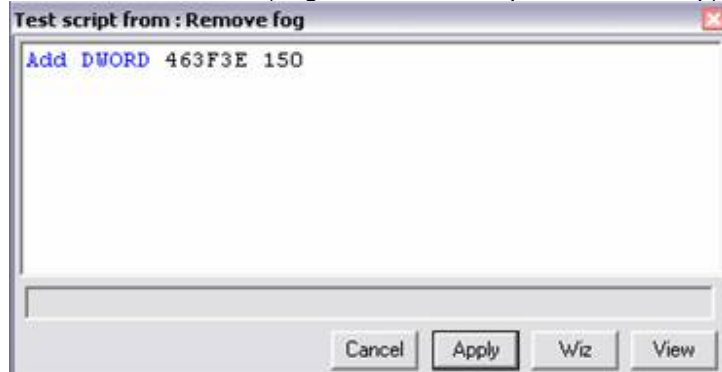
As for the minimap button, follow the instructions above to make another button. Remember to change the name of the button to Minimap and the hotkey to M (a good suggestion). When you come to step 7, you will have 2 Pokes, one for infantry and one for vehicles, enter the second poke on the next line.

2.6.2 Making the Remove fog-button

Now, the fog-hack is a little bit different. Follow the instructions in method 2.6.1 above to create a 3rd button. Changing its properties accordingly. When you get to step 7, enter the following, instead of a poke:

ADD WORD 463F3E 150

Of course 463F3E is not the correct address, you need to enter the one you found after having narrowed down addresses in T-Search, the DMA, remember? (Fog-hack method explained further up).



Instead of changing an address-line with a poke, we added to its value. So if you have set your fog to 100%, when your Remove fog-button is clicked, you will have clear sight up to 250% (100% then adding the 150 to that value equals 250%).

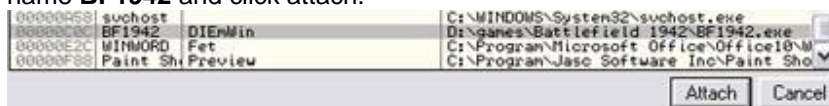
Note: If you click the Remove fog button more than once, the value will rise with 150 each time, so don't click on it too many times unless you have a very powerful computer or it will crash the computer.

2.7 Hiding treasures in code-caves

2.7.1 Fool the game

At the beginning of this tutorial I explained some theory of code-caves. I saved this part to last since it is the most difficult procedure unless you know ASM really well. Creating a code-cave is like taking a detour with your new sports car to avoid the police. Your only fear would be that the police would start checking the crossroad where the detour begins to see if you take it or not.

1. Start BF1942 (or mod).
2. Create a multiplayer LAN game (not internet), preferably Co-Op so you can see AI-players.
3. Pause the game once you spawned on either side.
4. Alt+tab out to windows and open Ollydbg.
5. Select **Attach** from the **File** menu, select the process-name **BF1942** and click attach.



WARNING: When you attach BF1942, Ollydbg will automatically pause the process once it scanned it (if you haven't set Ollydbg not to). Therefore you need to unpause it quickly otherwise the game might crash.



- Now click on the window that says **CPU – thread...** (just to make it active) and press **Ctrl+G** to open the **Enter expression to follow**-window. Enter the one of the addresses you found in each method above and click Ok. You need to do this twice in order to get to the **Main thread**.



If the address you found in T-search only had 6 characters, add **00** in front of them just like in the picture above.

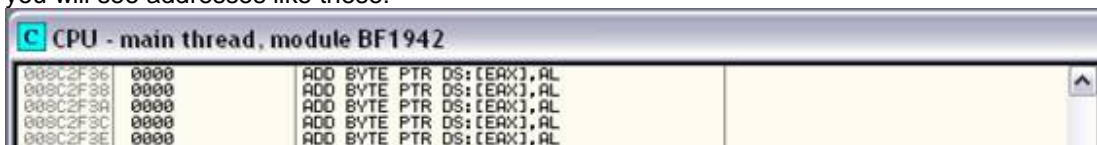
You should now be looking at a selected line where the address you found is displayed.

004C37B6	895424 14	MOV DWORD PTR SS:[ESP+14],EDX
004C37BA	75 04	JNZ SHORT BF1942.004C37C0
004C37BC	33FF	XOR EDI,EDI
004C37BE	EB 17	JMP SHORT BF1942.004C37D7

Now, let's presume that this address (004C37BA) is scanned by PB and if we change it we'll get kicked from the server we play at. So instead of changing it, we make a **JMP** (jump) from the line above it, change whatever we need and jump (JMP) back to the line below it.

2.7.2 First step, create the code-cave

- Locate a portion of the memory where there is no relevant ASM code. "Empty" addresses are often found at the end of each process. If you scroll down in your **Main thread** you will see addresses like these:



If the address has an Op-code of 0000, it means it isn't in use.

Let's take the first address we see in our process that is empty, **008C2F36** and create our code-cave here.

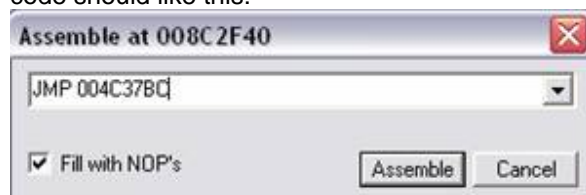
Hint: Always use **Ctrl+G** (go to expression) to navigate easier in Ollydbg.

- To recreate the address-line 004C37BA we need to copy it to our code-cave. Since we are also going to change the

address-line above it to a jump (JMP) we need to copy that once as well since we don't want to erase it's operation. This is what our code-cave would look like:



- The first line (008C2F36): Recreates the address we are going to change to a jump (JMP) in the original address (004C37BA).
- The second line (008C2F36): Is where we can change anything we like without PB noticing. So we could assemble/change this line to a NOP if we wanted to.
- The third line (008C2F36): Is where we tell the game to jump back to 004C37BC and continue all the following operations as normal. When you assemble this line, the code should like this:



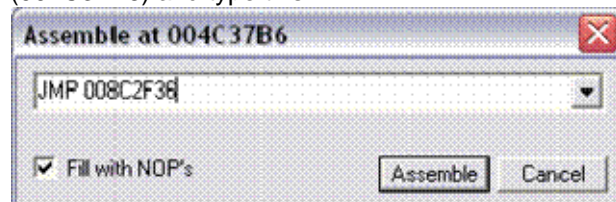
When we assemble this, Ollydbg will automatically create another address-line below it, NOPing that line. This is because we changed a 2byte address to a 1byte address (explained further up).

The **BF1942**. that you see before the address is like a bookmark for the game, no need to worry about it.

2.7.3 Second step, create the jump-gate

The jump-gate is the only address that we change in the original address-series so that it jumps to our code-cave. Of course, the jump-gate has to be located at an address not scanned by PB.

1. Now that we have our code-cave set up at **008C2F36** we need to jump to it from the original code. Assemble the line before the one scanned by PB (004C37B6) and type this:



Click **Assemble**.

This is what will happen:

CPU - main thread, module BF1942		
004C37B1	C1E8 1F	SHR EAX, 1F
004C37B4	03D0	ADD EDX, EAX
004C37B6	E9 7BF73F00	JMP BF1942.008C2F36
004C37BB	90	NOP
004C37BC	33FF	XOR EDI, EDI
004C37BE	EB 17	JMP SHORT BF1942.004C37D7
004C37C0	8BCB	MOV ECX, EBX
004C37C2	2BCF	SUB ECX, EDI

Ollydbg NOPed another address-line since we only had 1byte of code (90) to write on a 2byte address (explained further up).

If the game doesn't crash, you have successfully created your first code-cave in real-time.

If you have understood the methods 2.5 and 2.6 you will be able to extract these addresses and make pokes out of them to put in Tmk.

I will give you the brief method of doing this:

Copy the changes addresses to a notepad document. They should look like this:

Our jump-gate:

004C37B6	E9 7BF73F00	JMP BF1942.008C2F36
004C37BB	90	NOP

Our code-cave:

008C2F36	895424 14	MOV DWORD PTR
SS:[ESP+14],EDX		
008C2F3A	0F85 8008C0FF	JNZ BF1942.004C37C0
008C2F40	E9 7708C0FF	JMP BF1942.004C37BC
008C2F45	90	NOP

This is what you write in EasyWrite before you convert it:

Our jump-gate:

OFFSET 004C37B6
JMP 008C2F36
NOP

Our code-cave:

OFFSET 008C2F36
MOV DWORD PTR SS:[ESP+14],EDX
JNZ 004C37C0
JMP 004C37BC
NOP

Notice that you never write the "identifier/bookmark" (BF1942.) before an address when you convert it.

Our jump-gate converted to poke will be:

Poke 4C37B6 E9 7B F7 3F 00 90

And our code-cave becomes:

Poke 8C2F36 89 54 24 14 0F 85 80 08 C0 FF E9

Poke 8C2F41 77 08 C0 FF 90

Important: When you enter a code-cave into a Tmk script, always, and I repeat, always enter the code-cave before the jump-gate.

Since if the jump-gate gets pokes first, it will have nowhere to jump to and the game will crash.

So, finally, the code we need to enter into the Trk button is this:

Poke 8C2F36 89 54 24 14 0F 85 80 08 C0 FF E9

Poke 8C2F41 77 08 C0 FF 90

Poke 4C37B6 E9 7B F7 3F 00 90

3. F.A.Q.

Q: How come my friend can't use the fog-hack I created by following this tutorial?

A: Since we did the fog-hack the "easy way". Using a DMA address when creating the hack will have either of 2 effects:

1. The hack won't work at all since the DMA changed location.
2. The hack will work only on the computer that it was created on, since you managed to hack a "semi-dynamic" DMA that will stay static, but will hold different addresses on different computers.

Q: I copied the code on this tutorial but when I compiled the hack, it crashed the game.

A: None of the pictures or codes of this tutorial are real, the addresses and codes are manipulated and will not be the exact ones you're looking for.

Q: I was searching for addresses and suddenly; T-Search told me there were 0 addresses found.

A: If the map changes while you are searching/changing addresses you need to go back and do it all over again. Pausing it is a good way to prevent this from happening.

Q: I saw one address with 8 characters and another with 6, which one should I use?

A: In T-search addresses are displayed either with 6 characters or 8 characters (i.e. 66F2D3 or 0066F2D3) in Ollydbg the same address will always be displayed as: 0066F2D3.

Q: Why do I get the error message: **Unrecognized operand** when I try to assemble/change an address?



A: Remove the **BF1942**. from the ASM and the problem should be solved.

Q: Why do I get the error message: **Relative jump out of range** when I try to assemble/change inside the code-cave in Ollydbg (also applies to Easywrite)?



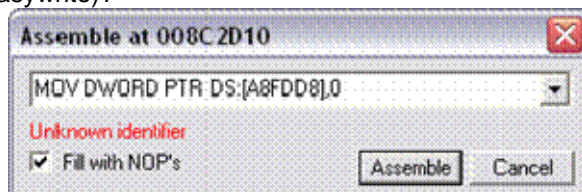
A: Since JNZ is a jump command (just like JE, JMP...) and a **SHORT** jump is within a few addresses from the jumps origin you need to change the **SHORT** to **LONG** since your code-cave is NOT within a few addresses from the jumps origin. This is how it should look:



Q: How will I know what addresses are scanned by PB?

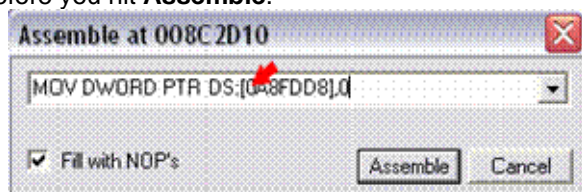
A: You could try sending e-mail to support@evenbalance.com and ask them (even if I doubt they will answer), or you can read Spontaneous tutorial on the subject (if/when it's released).

Q: Why do I get the error message: **Unknown identifier** when I try to copy original code to the code-cave in Ollydbg (also applies to Easywrite)?



A1: If you enter a **0** in front of the code in brackets ([A8FDD8]) the problem will be solved.

A2: If you don't have a long hex number in brackets like [A8FDD8], and if your number in brackets contain a "+"-sign ([ESP+AC]) you will need to enter that **0** right after the sign, like this: **[ESP+0AC]** before you hit **Assemble**.



I know all this probably doesn't make much sense, the **0** you enter in front of the hex number is like a prefix, telling Ollydbg/EasyWrite

that the number is a hexadecimal number and not a decimal number.

Outro:

I know this tutorial contains a lot of information; some may be harder to understand than other. A good trick is to always keep in mind what you are trying to accomplish. If you want to become a good hacker, my suggestion is you learn ASM and Visual Basic that are not very hard to learn.

Many thanx to:

Captain Cox, Max Powers and MPC.de