DOWNLOAD PDF OLLYDBG TUTORIAL

Chapter 1 : Ollydbg Basics for Beginners ~ Learn | HackingSkillz

OllyDbg Tutorial, Reverse Engineering, Reverse Engineering Malware OllyDbg is a bit disassembler/debugger for Microsoft Windows binary files. It is shareware and it is available here.

It is shareware and it is available here. The goal today is to provide a tour of OllyDbg and how the tool can be used in reverse engineering software or malware. Prerequisites You should have a good understanding of Intel x86 assembly opcodes; not how to program but at the very least, know how to it. You will also need the following tools: And if you want to follow along, I created a simple little CrackMe program that you can download from here. The Window with the disassembly and byte-code instructions is called the CPU window, there is a window that shows the current register settings and the EFLAGs register settings, the hints pane will display useful information such as register or address values while single-stepping through the code, you can always view the memory contents of data and registers in the memory view window, and the stack window shows the current stack setup during your debugging session. Note that many of the view menu items have hot-key commands. You may also call up many of the view menu options by clicking on the corresponding blue buttons L, E, M, T, etc. This window displays all debugging events such as module loads, thread creations, breakpoint hits, and errors. If you need to do some trouble-shooting during your debugging session, the Log Window may be useful in tracking down unusual or unexpected behaviors while stepping through mal-code. The Log window is also useful in checking to ensure any plug-ins you installed were loaded correctly. Executable Modules Window Figure 5: Red text means that the module was loaded dynamically. While in this window, right-clicking on a module opens a context menu. The Names Window shows the list of imported and exported functions for a given module. For example, if you see functions opening an internet connection and downloading files from an URL, the sample may be a downloader. This allows you to set breakpoints on API functions. For example, many malware will use the API IsDebuggerPresent to check if they are being debugged and attempted to kill the debugger. The Handles window shows the object type, reference count, access flags, and the object name for each handle owned by the process. The Stack window shows the virtual address of stack frame for each function call, the stack contents at that virtual address, the procedure and its arguments as pushed on the stack, as well as who called the procedure. The Call Stack Window And finally, the Breakpoints window shows all the user-set software breakpoints in the process. Breakpoints Window The window shows the virtual address of all software breakpoints currently set, the active status always, disabled , and the disassembly instruction of the breakpoint. You can right-click on this window to disable or delete the breakpoints that have been set. OllyDbg Hotkeys There are several hotkeys that you will find useful during your debugging session. F7 â€” the Step Into command. This key single-step traces one instruction at a time F8 â€” the Step Over command. This key single-step traces one instruction except for CALL instructions. This is handy for stepping over C-runtime libraries, such as printf, scanf, etc. F9 â€” Run runs the debuggee F2 â€” Set Breakpoint sets a software breakpoint on the currently selected instruction. OllyDbg has many context menus. You can right-click on almost anything in OllyDbg to get a context menu to examine your many debugging options. Reversing with Olly With our tour of Olly behind us, we are now ready to start doing some real work: First, it is usually a good idea to configure OllyDbg to ignore exceptions and to show loops. In the Exceptions tab, make sure your settings look like Figure Next click on the CPU tab and make sure the boxes highlighted in Figure Debugging Options to Show Loops Meet the Debuggee To demonstrate the power and functionalities of OllyDbg, we will use a sample that has some copy protections. Say we downloaded a trial piece of software that expires after a certain date or after 30 days. All we get is an error message when we attempt to execute it. Software Trial has Expired The first thing we should do is assess the software with CFF explorer to identify the development language used and some other particulars. We will need to rely on OllyDbg. In the menu bar, select File then open to navigate to the location of CrackMeDemo. After disassembly, Olly will take us to the entry point, which for this sample is at virtual address 0xE. At this point, the question we are now faced with is where to begin? Instead, we will use the power of the debugger to help us locate the error message. By hitting F9 to run the debugger, we should

encounter the error message as seen in Figure Now we will attempt to find the time limit checking code. Next press F12 to pause the debugging execution. With the execution paused, we now can search for the code that causes the error message. CrackMeDemo at Entry Point One way to look for our error message is to examine the current call stack since the error message is currently displayed at this point. From this vantage point you can easily see that the error message string is a parameter of the MessageBoxA function call see Figure MessageBoxA near the bottom of the call stack. MessageBoxA is made Figure Running the debuggee Figure Select Show Call Figure The parameters start with the PUSH 10 instruction at 0x Since we are at the PUSH 10 instruction indicated by the grey line , we can examine the Hints pane to see the parts of code that references this call: The Hints pane shows two places that jump to this error message box Select the text area of the hints pane and right-click to open a context menu. It allows you to easily navigate to the code where those jumps are made: Now we can modify those parts of the code. Select the JNZ from 0x from the context menu. This will take you to the jump command at 0x In order to prevent the program from hitting this error code path, we can change the jump instruction to a NOP no operation instruction. The CrackMeDemo file window Figure Saving our changes I renamed our fixed CrackMeDemo software and saved it to the desk top. Double clicking on our new, patched binary should result in: I also consult with several University Computer Science Departments on how to develop a reverse engineering course. If you liked this post, feel free to share it with others who may be interested in learning the art of reverse engineering.

DOWNLOAD PDF OLLYDBG TUTORIAL

Chapter 2 : [Tutorial] OllyDBG - The very basics

OllyDbg is a bit assembler level analyzing debugger for MicrosoftÂ® WindowsÂ® developed by Oleh Yuschuk. It is used in cases where the source is unavailable.

One of the tests consisted of getting the serial key of a simple program. You can download this awesome tool from here: You can see its details in the picture below. The first thing I usually do in these cases is to check if the executable is compressed or not. Some programs pack some of their codes in order to limit our attempt to statically analyze it. To achieve this purpose we are going to use PeID. If the file were compressed with UPX for example, the program would advise us about it and we could uncompress it with this tool. We can see the R. Size Raw Size "" and the V. Size Virtual Size "" are similar in ". If some day you detect that the R. Size is "0" and the V. Size is "" for example, it would be an indicator that the executable is compressed because in the disk it does not have any size it is packed and in the memory it has a size it is unpacked itself. Now we have the assurance that the file has not been compressed. This is one of the first steps in a static analysis. We are going to make a dynamic analysis with OllyDbg but I want to know if the developer has made an effort in order to try to hide some code. Notice if the executable is packed then we are not going to be able to read a lot of strings within the file. It is possible I will talk about that in future posts The next step would be to run the program by double clicking on the executable. After that, we can see that a MS-DOS window is launched and the program requires us to type the serial number. We have not figured out the serial number Now, we are going to run OllyDbg. It does not need installation, just download it and uncompress it. Now we can see the binary code. We are going to click on the play button in order to run the executable just loaded in our debugger and check the file behaviour. Please, click on the picture to see the entire details But If we reload the file again on OllyDbg, one line of the code draws our attention If we seek this API on Microsoft we can see that "This function allows an application to determine whether or not it is being debugged, so that it can modify its behavior". Ok, the program is closed when it is open within a debugger. There are many options to avoid being detected by this technique To achieve this purpose we are going to use the " Hide Debugger 1. It is necessary to restart OllyDbg in order to work with this plugin. If you click on Plugins tab you can see Hide Debugger plugin. We have just installed the plugin to avoid being detected and now, we are going to load and play the executable again. Now the program requires typing the serial number. We are going to type a sentence which will be easily recognizable. If we come back to OllyDbg we can see our sentence in the Arg1. Please, click on the picture to see the entire details If we continue looking for this sentence through the code we can locate the code below. We can figure out that the executable is comparing these strings to each other in order for you to check if both have the same value. We can suppose that the string "" is serial number. Please, click on the picture to see the entire details OllyDbg offers us to copy the value of this line by left clicking on the line we are interested in. In the end, we just need to try paste the value just copied in our program and We have obtained the serial number of our program!!! This post could be applied to many of the simple programs which have a keygen integrated but it is needed to have more knowledge if you want to crack more complex programs. This post is focus on show you some techniques using OllyDbg. It is only a game to get more reversing engineer skills to research malware. I recommend you use to use free software!!!!

Chapter 3 : How To Reverse Engineer Using OllyDbg

Ollydbg Tutorial #1 Autosofted. Loading Unsubscribe from Autosofted? Software license restrictions versus Ollydbg - The Basics.[Part 1 of 2] - Duration:

I was storing some files on my webserver and my WinRAR license was past its due date like a really long time and the nag screen annoyed me as fuck. WinRAR has a 32 and 64 bit installer, whereas the previous target Internet Downloader Manager only has a 32 bit installer. Now 32 bit applications runs fine on 64 bit but not vice versa. This tells us the target hash multiple license formats, which we can and will exploit. Attach your debugger to the WinRAR process and make sure you are in the winrar. This will pop up a list of all occuring strings in the WinRAR. Lets check them all. This looks like the function is trying to find any of the allowed file formats. Nothing we can use. Lets take a look at the next one: That looks alright, it seems like call 13FD20 is the method to parse the license key, meaning that the JE 13FD3B afterwards will decide whether the license is valid or not, interesting. Note that the JE instruction jumps over a call, which could be the function where the nagscreen is located: Although we have 2 more strings to check, so lets continue. You might already have seen due the bytes between the last 2 addresses are minimal that they are really close to eachother: Okay, lets get back to our only useful find: If you put a breakpoint on there and resume F9 the program, you notice that the breakpoint is actually getting hit within a second. This usually means the function we are in is inside a thread, which matches the behaviour of the nag screen. Lets follow the code step over, f8: Make sure you stand on the call instruction like in the screenshot, the address on the left side is black at the call instruction, this means that is where we are currently. Then, press F7 to step inside the call function. The second JE jumps all the way down to the function. If you put a breakpoint on the first JE and run the program again F9 you will notice it keeps getting hit. Whenever it hits the breakpoint and you hit F9 again, it will instantly hit the breakpoint again. It means we are still in the thread. If you have used WinRAR for awhile you know that the nag screen appears randomly, so my guess would be this is the right method. To test out whether we have the right method, you can change the 2nd instruction its opcode from JE to JNE. Now whenever you resume the program and get past the breakpoint a nag screen will appear! If you want it to stop it will keep opening a nag screen every second change it back to JE. If its JE it will only show the nag screen sometimes, but still enough for it to annoy the fuck out of us. JMP jump means that it will always take the jump, no matter what and that means we will never have to see the nag again: If you inspected the first JE you would notice that it jumps past the second JE. Again, the solution to this is also very simple:

Chapter 4 : OllyDbg Tutorial | Eric Hokanson

Beginner Olly Tutorial Part 03 Sep Shub Nigurrath kB

The doSomething function will probably do something with these variables and then return control back to the calling program. The stack is one of the main ways that variables can be passed to a function: The stack is not the only way to do this, it is just the most often used. These variables could also have been put into registers and accessed through these registers inside the called function, but in this case, the compiler of our program chose to put them on the stack. We will also go over this several times in the future. This is because we stepped-over the call that actually has most of the program in it. Inside this call is code that enters a loop waiting for us to do something, so we never get control back to the line after the call. Olly will immediately pause on the next line after the call: You will also notice that our program has disappeared. This is because, somewhere in that call, the dialog window was closed. Also, if you look down one line you will see that we are just about to call kernel This is the Windows API that stops an application, so basically Olly has paused our program after it has closed the window but before it has actually been terminated! Address will now turn red: What you have done is set a breakpoint on address Breakpoints force Olly to pause execution when it reaches it. There are different types of breakpoints in order to stop execution on different events: Software Breakpoints A software breakpoint replaces the byte at your breakpoint address with a 0xCC opcode, which is an int 3. This is a special interrupt that tells the operating system that a debugger wishes to pause here and to give control over to the debugger before executing the instruction. Now that we have a BP breakpoint set at address and our program is paused at the first instruction, hit F9 run. Our program will run but will pause at the line with our BP on it. I also want to point out something very helpful. You will see an entry in the breakpoint window that shows our currently set BP: This gives you a quick overview of all of your breakpoints set. You can double-click on one of them and the disassembly window will jump to that breakpoint though the EIP will stay the same as you are not actually changing the flow of control of the program. If you highlight a breakpoint and click the space bar, the breakpoint will toggle between enabled and disabled. Lastly, restart the program, go into the breakpoints window, highlight the breakpoint we set at address , and hit the space bar. Now run the program F9. You will notice that Olly did not stop at our BP because it was disabled. Even though there are 8 built into the chip, we can only use four of them. These can be used for breaking on reading, writing or executing a memory section. You set a hardware breakpoint by right-clicking on the desired line, selecting Breakpoint, and then choosing Hardware, on Execution: Using a memory breakpoint tells Olly that you want to pause whenever ANY instruction in the program reads or writes to that memory address or groups of addresses. There are three ways to select a memory breakpoint: To set a BP on an address in the memory dump, highlight one or more bytes in the dump window, right-click on them and choose the same option as above. You can also set a BP for an entire section of memory. Make sure the FirstProgram is loaded and paused at the beginning. If you look at this line, you will notice that this call is going to jump us down to address c, which happens to be 3 lines down from where we currently are. Hit F7 to step in to the jump and we will then be at that address at c. Remember that this is a CALL instruction, so we will eventually come back to or at least the instruction after this. Now, step the code F8 until we get to address You can also just set a breakpoint on this line and hit F9 to run to it. Remember how to set a breakpoint? You can also just highlight the line and hit F2 to toggle the BP on and off. Now we are paused at address You will notice that we have a couple options here: This loads whatever address is being affected in the instruction. We would basically just be looking at a dump of what we were looking at in the disassembly window. This would in effect show the memory for the local variables we were working with on the stack. As you can see, the dump now shows memory starting at address , which is the address the instruction Olly was loading the ASCII string from. On the left, you can see the actual hex for each character. These strings will be used in other parts of the program. You will notice that the first hex digit is also highlighted on the left. This allows us to change the memory contents of our program: We should then have a screen that looks like this: You may overwrite other strings the program need, or worse, code that the program needs!!! Now click OK and run the app click

inside Olly and hit F9. Now look at our dialog box!!! Notice anything different about the title of the dialog box.

Chapter 5 : Tutorial #3: Using OllyDBG, Part 1 Â« The Legend Of Random

Abstract. The objective of writing this paper is to explain how to crack an executable without peeping at its source code by using the OllyDbg calendrierdelascience.comgh, there are many tools that can achieve the same objective, the beauty behind OllyDbg is that it is simple to operate and freely available.

The objective of writing this paper is to manifest, how to crack an executable without peeping its source code by exercising OllyDbg tool. Abstract The objective of writing this paper is to manifest, how to crack an executable without peeping its source code by exercising OllyDbg tool. Although, there are much of tools that can achieve the same objective but the beauty behind OllyDbg is that, it is simple to operate and freely available. We have already done much of reversing of. This time, we are confronting with an application which origin is unknown altogether. We just have only the executable version of a particular application which is in fact, a tedious task in context of Reverse Engineering. Essentials The security researcher must have rigorous knowledge of Assembly Programming language. It is expected that their machine must be configured with the following tools; OllyDbg CFF explorer Patching Native Binaries When the source code is not provided, it is still possible to patch the corresponding software binaries in order to remove various security restrictions imposed by vendor as well as fix the inherent bugs in the source code. A familiar type of restriction built into software is copy protection which is normally forced by software vendor in order to test the robustness of software copy protection sachem. Typically in copy protection, the user is obligatory to register first for the product before use. The vendor stipulates a time restrictions condition over the beta software in order to be license misuse and permitting the product to run only in a reduced-functionality mode until the user registers. Executable Software The following sample shoes a way, how to bypass or remove the copy protection sachem in order to use the product without extending the trail duration or in fact, without purchasing the full version. The copy protection mechanism often involves a process in which the software checks whether it should run and, if should, which functionality should be exposed. One type of copy protection common in trail or beta software, allows a program to run only until a certain date. In order to explain reverse engineering, we have downloaded the beta version of software from internet which is operative till 30 days. As you can see, the following trail software application is expired and not working further and showing an error message when we try to execute it. So, the first task is to identify its origin. So, we can easily conclude that this is a native executable and it is not executing under CLR. This time, we have to elect some different approach to crack the native executable. How can we use this software regardless of the expiration of the trail period? The following section illustrates the steps in the context of removing the copy protection restriction as; The Roadmap Load the expired program in order to understand what is happening behind the scene. Debug this program into OllyDbg. Trace the code backward to identify the code path. Modify the binary to force all code paths to succeed and to never hit the trail expiration code path again. Such tasks however, can also be accomplished by one of the powerful tool IDA pro but it is commercialized and not available freely. OllyDbg is not as powerful like as IDA pro but useful in some scenario. First thing first, download OllyDbg from its official website and configure it properly onto your machine. It looks like as following; Now open the SoftwareExpiration. Here, the red box showing, the entry point instructions of the program referred to as The CPU main thread window displaying the software code in form of assembly instructions which are executed top to down fashion. That is why as we stated earlier, assembly programming knowledge is necessary when reversing with native executable. While the error dialog box is still displayed, start debugging by pressing F9 or from Debug menu. Now, you can find the time limit code. Next press F12 in order to pause the code execution so that we could find the code that causes the error message to be displayed. MessageBoxA near the bottom of the call stack and right click and choose show call as following; It shows the starting point in which the assembly call to MessageBoxA is selected. Directly before the call to MessageBoxA in red color right-pane , four parameters are pushed onto the stack. Select the PUSH 10 instruction located at C0 address, the line of code that reference the selected line are displayed in the text area below the top pane in the CPU windows as following; Select the text area code in the above figure and right

click to open the shortcut menu. It allow you to easily navigate to code that refers to a selected line of code as shown; Up till now, we have identified the actual line code that is responsible for producing the error message. Now it is time to do some modification in the binary code. The context menu in the previous figure shows that both and contains JA jump above to the PUSH 10 used for message box. So, first select the Go to JA from the context menu. You should now be on code at location 0x Your ultimate objective is to prevent the program from hitting the error code path. So this can be accomplished by changing the JA instruction to NOP no operation which actually do nothing. Right click the 0x instruction inside the CPU window and select binary where click over Fill with NOPs as following; This operation fills all the corresponding instruction for 0x with NOPs as following. Clicking Copy to Executable, and then clicking All Modifications. Then hit the Copy all button in the next occur dialog box as shown below; Right after hitting the Copy all button, a new window will appear named SoftwareExpiration. Here, right-click in this window and choose Save File as following. Finally, save the modified or patched binary with new name. Now load the modified program, you can that no expiration error message is shown. We successfully defeated the expiration trail period restriction. Final Note This article demonstrates one way to challenge the strength of the copy protection measure using OllyDbg and identify ways to make your software more secure against unauthorized consumption. By attempting to defeat the copy protection of your application, we can learn a great deal about how roust the protection mechanism. By doing this testing before the product become publically available, we can modify the code to make circumvent of copy protection more sophisticated before its release.

Chapter 6 : Reversing Tutorial - Cracking Registration of A-One Video Converter | calendrierdelascience.co

Reddit gives you the best of the internet in one place. Get a constantly updating feed of breaking news, fun stories, pics, memes, and videos just for you. Passionate about something niche?

August 15, A-PDF Screen Tutorial Maker is powerful but easy-to-use software to create live and clear tutorial from screenshot with ease. The friendly user interface guides you operation details in a clear way. You can make a complete tutorial just by capturing screen, inputting instructed text, and then publishing with wanted format PDF, Word, Html. No more professional tutorial editing skills needed. May 30, The GIF Construction Set Professional animation software is easy to use, and this tutorial will make it even easier to learn. The tutorial installs on your hard drive to make it quick and easy to access. Freeware Guide to Webmaster Affiliate Programs 1. June 21, This tutorial will show webmasters how to make money with webmaster affiliate programs. After years of trying to make money on the Internet I finally discovered what works. I currently receive residual income with minimal effort. This tutorial explains in plain English how to get your online career started. Freeware Meeting Manager tutor 3. June 21, This tutorial teaches you how to conduct effective meetings using Smartworks-Meeting manager enterprise software. Meeting Manager is a Smartworks Plug in, which can be used to schedule and keep track of meetings and resources in an organization. This tool will help you to organize meetings, inform attendees and to track post meeting Action points till closure. Meeting rooms and resources can be scheduled to avoid clashes. Freeware Time management tutorial 1. May 24, Intuitive daily planning tutorial what shows basics of collecting to-do tasks, planning a daily work and self-control principles. Freeware LSP tutorial 1. Freeware How to Play the Guitar Vol1 5. August 09, How to Play the Guitar - Volume I, is a guitar tutorial that makes it easy for absolute beginners to learn to master the instrument. Developed by a professional guitar player and trainer, this multimedia tutorial takes you step-by-step through the fundamentals of playing blues, rock, folk, arpeggios, and even classical music on the electric or acoustic guitar. The lessons are designed for guitar enthusiasts with no prior musical knowledge. All lessons are written in both tablature and Developed by a professional guitar player and trainer, this multimedia tutorial takes you step-by-step through the fundamentals of playing blues, rock, folk, finger picking, on the electric or acoustic guitar. Volume II is for beginning and intermediate musicians. Its 23 lessons cover guitar skills, technique, and musical theory. It uses both tab and standard musical notation to teach students to master the pick, June 21, L-Basic is a Basic language tutorial program. L-Basic emphasizes teaching proper Basic programming concepts rather than how to use visual controls. L-Basic will help you understand how to program in the most popular versions of the Basic language, Visual Basic and Microsoft Access Basic. L-nix includes a simple UNIX shell simulator allowing users to try out commands as they are learned. The L-Nix demo contains 10 modules: The Registered version of L-nix provides nine additional modules: March 09, FlashDemo Screen Recorder captures all screen activity from your Windows Desktop in real time and publishes as a Flash movie. Easily reach the widest possible audience by placing lectures, presentations and demonstrations online. Insert callout, shape,arrow, text, image, sound etc. June 18, Visharad Process Mentor VPM simplifies and speeds up the generation of documentation of a business process and the content showing simulation of business process. It has the features to captures business processes, author content, and launch content. It captures a process running on PC and then generates contents. HTML file documents process flow. It is best for any type of Business Documentation and Brochure making. October 07, Tutorial Bridge is a program for learning the game of contract bridge while at the same time playing and enjoying the game. This self-playing computer program gives the reasoning behind each non-trivial bid that is made, or card that is played. The user can interact with the program at a rate that is consistent with his or her current knowledge of the game. The ability to undo any bid or play permits the student to compare his or her action with what the program would do in the same It demonstrates the most-commonly used PinYin input method, plus advanced Chinese input techniques. December 27, FlashDemo Screen Recorder captures all screen activity from your Windows Desktop in real time and publishes as a Flash movie. Insert callout, shape, arrow, text, image, sound etc March 05, FlashDemo Studio screen recording tool helps you to record

everything happening in your desktop. April 03, A Musical Tutorial - An excellent way to encourage musical study. Associate notes to piano keys. Chord dictionary and musical games. Play, view and print scales, chords and triads. Print sight reading test papers. Intervals, ear tests, key signatures, rhythm and more. Makes music lessons fun for children. Freeware Haxe Flash Tutorial 0. June 10, Haxe Flash Tutorial is a tutorial on Haxe with sample code ready to run out of the box. A library with reusable functions completes the tutorial. Freeware RRDtool tutorial 1. May 08, Very basic RRDtool tutorial. It is an example in that way. It can save you a couple of days of discovering on your own. Freeware Tk Tutorial 0. July 03, Tk Tutorial: Learn about references and anonymous arrays by using them. Use templates to build GUIs with menus, buttons, etc. Includes tips on tools, development. When launched for the first time, GalaXQL first generates the database. This may take some time, so please be patient. Note for mac users - apparently something has changed in the last few mac os updates, and the universal build for intel is broken. You can still use the 2. Sorry for the inconvenience. Emphasis on binary code analysis makes it particularly useful in cases where source is unavailable. OllyDbg is a shareware, but you can download and use it for free.

Chapter 7 : How to get the serial number of a program with OllyDbg ~ Hacking while you're asleep

This tutorial is specially written for those who want to master the art of software cracking using OllyDbg, providing all the necessary basic and associative knowledge. With these basics, you can read and understand other more advanced tutorials and theory.

So, since three four? This is a simple introduction to the program OllyDBG. Also known as Olly Debugger, and, as I like to call it; Olly. Added a quick video on solving "Crack Me"s 3, 4, and 5. Check the bottom of the post. First of all, a question many people ask is: The answer is simple, really. Olly is an x86, 32bit debugger originally intended for developers who had problematic errors in their applications. It allowed them to go through their application step-by-step, monitoring most every action that the application took. And by doing so, this allowed them to find where the error actually happened in real-time, and made it much easier for them to fix it. Now you may be wondering, what does Olly have to do with you, then? Well, it has quite a lot to do with you, actually. Aside from the basics of the debugger, it is more widely used for the purposes of reverse engineering. And at the same time, it allows us to go to things like conditional statements, and either change the condition, or change the whole statement, all in real-time without even having to recompile or restart the application. At the top portion of Olly, we have a long line of horizontal buttons that will save us having to even use the menus for the majority of the time. This is the Open button. This is the Restart button. Fairly obvious, it restarts our executable. This is the Close button. This is the Run button. This is the Run Thread button. It does the same as above, but only runs the current thread. This is the Pause button. It pauses out executable so we can look around or do other things. This is the Step Into button. It steps down into the next line, or enters the current function. This is the Step Over button. It does the same as above, but executes the function all at once, instead of going into it and stepping through each action. This is the Trace Into button. Same premise as the Step Into button, but works with our run trace. This is the Trace Over button. Same premise as the Step Over button, but works with our run trace. This is the Execute Until Return button. It will keep stepping into the application until it hits a return, either from a function, or the application itself. This is the Execute Until User Code button. It will keep stepping into the application until it hits code that is not part of the system functions. The following are windows. This is the Logger window. This is the Executable Modules window. This is the Memory Map window. We can use this to find something specific in the memory space of the application. This is a good way to find the un-packed data inside a packed application. This is the Window List. It usually shows us a list of window handles owned by our application. This is the Threads window. This allows us to see and select which thread we want to work with, amongst other things. This is the CPU window. This is where the core of the application is shown: This is usually shown in Assembly code, and this is where we will do most of our work. In this window we can do anything from monitor the actions the application takes, to changing what the application will do next in real-time. This is the Search Results window. This is the Run Trace window. This will be more useful later on, and is very helpful for tracing changed in certain things. This is the Breakpoints window. This gives us a list of the breakpoints we currently have set, so we can just double click then to jump straight to that location in the memory. This is the Memory Breakpoints window. This is the Hardware Breakpoints window. This is the Options window. We can change lots of things related to Olly in here, including colors. What I will do now is walk you through how to solve each of the three applications. Granted you downloaded the archive and unpacked it already, go ahead and open up crackme You should get something like this once it is done loading: Ignoring the fact my colors are different. Notice that in the bottom right corner, our status indicator says that the application is Paused? It will not run automatically, but rather pause and wait for input. First things first, use the Restart button to restart the application. Once it is done loading again, look at your CPU window. One of the first things we always want to do to make sure things are accurate, is make sure that the code has been re-analyzed. In older versions, it should be All referenced text strings. You should get a window that looks like this: Now, we can ignore those strings up top, because those are just part of the internals. Right, that means that these are the strings stored inside the application. Note that a string does not have to be stored in a variable for it to

show up here, so even if you had something like Code: Wow, looks like we solved it! More often than not, the information that we need to continue is right in front of our faces. It looks like this one is going to be slightly more difficult. Not only does it say it has two checks, but it also states that it will not work the same way. By taking a look at the console, we see that this time it is asking us for a username and not a password. So we know that if there really are two checks, it checks the username before even taking us to the password, which is slightly more difficult. But not only that, there was mention that it did not work in the same way as the first one. As the window comes up, we have a little more data than last time. So, what do we do now? This will take us to the address that it is referenced at. Well now, look at that. Right above our failure message we have a jump. What a jump does is it jumps to a certain address in the memory, going right over and ignoring any code in-between the two points. First, we look at the JE, what does that mean? Basically, it translates to Jump if Equal. The opposite being JNE: Jump if Not Equal So, if the condition in the line before above the jump is met, we will take the jump to the address of 0xC, and if not, we will go to the line after below the jump. Doing so will cause the application to pause and notify us when it gets to that point. We should now be at this point after entering the username. We can actually do many things: NOP the test line; change it to something like CMP BL,1; change the address of the jump to the address of the success message or the password check; or we can simply change the value of BL. We see the register list, which looks like this. Right now we only need to pay attention to our main registers: AL and AH are both 4 bit registers two bytes. We should now see this. Now, go ahead and click OK, and then Run the application.

Chapter 8 : Cracking WinRar with OllyDBG/x64dbg -

Cracking Tutorial Compilation Vol Most cracking tutorials say stuff like, this is only for educational purposes and to an OllyDBG kicks in here and needs.

This works because assembly allows you to speak directly to the processor and force a skip over the registration process. Requirements Windows for examples only, debuggers exist across platforms A debugger installed: IDA , ollydbg , etc. This is to verify that we can come up with the keys. Step 2 Run the Program in a Debugger Run ollydbg. Open up the program you wish to bypass with ollydbg. Click the play button to run the program with the debugger attached. Search for high interest DLLs. By stepping into the function with the debugger, we can examine the registration specifically. Test to see which one works to break out of the activation loop by right clicking the DLL call and setting a breakpoint for all instances of that call. Resume the program and enter any software key you feel like. EAX is the return of a value, which means that a check is being performed here. Upon examination, we can see that the EAX is checking for a number that is not equal to a null value. This means that if it is replaced with anything other than null, it will run. Right-click the EAX and change it in hex value to 1, instead of 0. Resume the program again, and you will have successfully activated the program. And for proof it was registered to me: This works because you are making the process jump from one register and skip the one that verifies the key entered. To exploit the key registration algorithm, keep an eye out for part two of this tutorial on making the key generator. Post to the forums.

Chapter 9 : OllyDbg - Wikipedia

OllyDbg is a bit assembler level analysing debugger for Microsoft Â® Windows Â®. Emphasis on binary code analysis makes it particularly useful in cases where source is unavailable. Intuitive user interface, no cryptical commands.

Section 2 - Getting Started Ok, so you should have downloaded the crackme and have Ollydebug installed. First thing to do is close this tutorial and have a play around. See what you can find and get a feel for the program. The very least this will do is teach you how to use basic Ollydebug functions. No cheating now ;- Done? Maybe you found nothing and reckon you just wasted 30 minutes? Okay, so run the crackme and lets have a look around. Enter a user name into the box and a random username. So we know what we need to do; we need to find the serial - at this point we dont know if its a hard coded number or if its generated from the username but thats part of the fun! Okay, so open Olly and select Crackme1. We therefore need Olly to intercept any calls this crackme makes where it could be reading what we typed from the username and serial boxes. So, what we need to do is make sure that if the Crackme makes this call, Olly intercepts it and breaks for us so that we can follow what is being done with the information. Press F9 and Olly will run the crackme, presenting you with its user interface. Go to the registration box and enter a name and any serial. Press the register button and Olly should break here: This is important as when this says Kernel or User32, we know we can keeping stepping as it has nothing to do with our serial - we are only interested in the Crackme. Press F8 to step over the program and try to get a feel for what is going on. Pressing just twice will bring you into User32 and after 15 step overs we are back with the crackme. In future you will use F10 and F12 to step, F8 just shows you more of whats involved. If we continue this process we go through a long session in User32 and eventually land back here: Then select the line and press F2 to put a new breakpoint here. What this means is that you can now come back here whenever you run the program without stepping through all the previous steps we have taken. You dont want to search for this again if you press a wrong button somewhere! So, we probably know how we could get the congrats message - a flick of the Z bit at or simple patch of the JE at should do it. Our job is to trace the calls at D and to find out exactly what is going on here. Press F8 until you highlight the following row: The difference between F7 and F8 is that F8 steps over calls and F7 steps into them. In other words, if a call is of no interest to you, you can press F8 to step over it and carry on. If you think that it might contain some vital information, press F7 to step into it and you can look at it in detail. You should now be here: The first character of our username F in my case is then moved into AL before being tested to see if it is 0. Then the interesting stuff starts - at the F is compared with A strange comparison you might think? Open up a browser window and go to www. The computer deals with character values in hex i. What the line at is doing then, is its taking the first letter of our username and comparing it with A. The result of this comparison effects what happens at the jump on the next line B as if the first letter of our name is less than A see the ASCII table it jumps elsewhere. My F is above A though so we continue to D. Here a similar operation is performed. A quick look at our ASCII values shows us that our character is now being compared with Z - this time a jump is taken if the value is above Z. Obviously, my F is fine and we continue. At ESI is incremented before a jump is taken back to If you remember, our username is stored in ESI so this has essentially just moved us to the next letter of our username and gone back to the beginning of this routine. When we get to the comparison with Z though, it fails and the jump is taken at F to This is because, as the table shows, a 61 is greater than Z 5A. So we land here: Our character is in AL and gets 20 subtracted from it. It then jumps back to the routine, increments ESI to the next letter and continues. Thats all this bit is doing. A couple of points to note though are that if you only used uppercase letters anyway, this routine is redundant and you wont even see the SUB AL,20 part. Then comes this line: Setting a breakpoint here may be useful too! Then a similar thing happens to what happened in the above routine - the only difference being that the first letter of our capitalised username is move to BL rather than AL. Its then tested incase its 0 before landing at CC. We then increment to the next letter of our username and the process is repeated although notice that the loop does not include the XOR functions each time. This basically has the effect of adding all the values of our username together and storing it in EDI. For my username I get this: We know that EDI contains our

summed username so in my case, this equation is: We then jump back to the initial code we looked at in section 2. Now see if you can follow the same procedure for the second call below! Trace into it with F7 and see what you can find We can then use F7 to trace our second call.