# Kindle Touch (5.0) Jailbreak/Root and SSH

**Kindle Touch (5.0) Jailbreak/Root and SSH** - @yifanlu, Yifan Lu.


- Published: 2011-12-10
- Original: <http://yifan.lu/2011/12/10/kindle-touch-5-0-jailbreakroot-and-ssh/>
- Preserved from: http://yifan.lu/2011/12/10/kindle-touch-5-0-jailbreakroot-and-ssh/ (browser) on 2026-08-09
- Licence: unknown




## Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

**Update** Kindle 5.0.3 has fixed the hole to allow for jailbreak. Upgrading an already jailbroken Kindle Touch is fine as the update does not remove the custom key to allow custom packages. If you on 5.0.3 and have not already installed the key, there is a new jailbreak.

So long story short, we can run custom code on the Kindle Touch now but because the operating system has changed so much from Kindle 3, most Kindle modifications will not run without changes. I hope developers will jump to this device now that it's unlocked. See the bottom of the post for download links. The directions for using are in the readme. Keep reading for technical details on how this came about.

**Obtaining the root image** Before we can look for vulnerabilities in the system that would allow us to break in, we need to break into the system and obtain the files that might contain vulnerabilities. Yes, this is a chicken-and-egg problem, but fortunately Amazon is nice enough to help us with this. On every Kindle device is a TTL serial port. I found this port on the bottom of the device when the cover is opened. Fortunately, I did not even have to mess with it, as **hondamarlboro** and **ramirami **both managed to get the dump before me. Once we have the root image, it was only a matter of painstakingly looking through all the files to see possible injection vectors.

**Looking for the needle**

At first, I was digging deep into the system, disassembling and maping out various native libraries, looking for stack overflows (I found a couple but none could be accessed efficiently). I found the bo otloader was unlocked but it would be a pain and danger for users (and even developers) to flash custom kernels and such. I also found that the Java code (the Kindle's entire GUI is written in Java) is NOT obfuscated (which means it would be easier to reverse and later modify) and Amazon has left in many places to place plugins. For example, once someone has the time to figure things out, it would be very possible to write a EPUB extension to read EPUBs from the native reader. There

are some other hidden secrets in the device too. The Kindle Touch has an accelerometer and proximity sensor (and a mic, but we know that) but they aren't used in the software (yet). The more I looked into the system, I was aware that because it was such a huge rewrite, I had misjudged when I assumed that it would be harder to break as Amazon had years to fix the holes now. In fact, I would say that the Kindle 4 is more secure until I found out that Amazon left in SSH in diagnostics mode. Anyways, as I searched up the complexity chain from the bootloader to the kernel to the libraries to the Java interface, I found something very curious. Much of the operating system is no longer written in Java, but are now in HTML5 and Javascript. In fact, many of the interfaces on the Touch are actually web pages in disguise. For example: the password entry screen, the search bar, the browser (is just an HTML page with a frame), the Wifi selection screen, and even the music player. Obviously, these can't all run natively in HTML and JS, or the device will be even slower (and it is pretty damn slow). What Amazon did is write a couple of Javascript hooks that are implemented by native libraries and events are read by these libraries and they perform actions accordantly. In short, Javascript will run native code. This is a goldmine, there could be many possible ways of using this to our advantage. There could be buffer overflows, heap overflows, string formatting bugs, etc. However, I didn't have to look though much before I found a curious function: nativeBridge.dbgCmd();. It seems too good to be true. This function takes any shell command, and runs it (as root). Yup. The web browser will run as root, any command given to it. Don't go looking for remote code execution yet (although it is highly possible), as the native bridge seems to be disabled when in web browser mode (it may be able to be bypassed, but I haven't looked into it).

**Calling the debug function**

So the normal browser (as the one you can enter URLs into) can't make use of this native bridge. However, as I've mentioned, a large part of the GUI in the Kindle Touch is HTML and JavaScript. All we need to do is inject some HTML into one of these and we would be all set. We need something that takes input and displays it to the user. The first thing I thought of was the media player. The Kindle displays the song title, artist, and album name in the music player, so what if we put some HTML into the ID3 tag? Yup, it works. How about some javascript? Running. Let's try to call the debug function. It works. Well, that was a freebie.

**Having some fun**

That was a bit too easy and I was disappointed that I didn't get to talk about how I whipped out IDA Pro and did some master debugging. So, let's make things harder. We can use a MP3 with custom ID3 tags to execute any command, but how can we make this into a cool one-click solution? First of all, we should limit ourselves to one file to copy. Why make the user keep track of MP3s and shell scripts and where to put them? I took the shell script payload (which installs a developer key into the device so custom packages can be installed) and placed it into the comments section of the ID3 tag in the MP3. Then I used "dd" to extract the script, chmod it, and execute it. Now, another problem in terms of user friendliness is how to let the user know that the process was successful? I quickly whipped up an awesome looking "splash screen" and planned on displaying it while the magic is taking place. At first I tried to encode it into a variable in the shell script payload and extract it, but it was too slow and memory intensive. Instead, I took the image, raw, and appended it into the end of the MP3 (after all, the file was a bit too small). You can see the result in the video attached.

**What's next?**

Just because the device is jailbroken does not mean it can now magically do anything you want. What needs to happen first is that developers need to take the device and write some code for it. This first jailbreak is really for these developers. For regular users, the only use is to preemptively unlock your device now in case the method is patched in an update or something. **No mods for older Kindles will work as-is on the Touch.** I've included a VERY basic usbnetwork package that will allow you to have SSH access to the device. I think that's as good of a starting point as anything. From there, developers should be able to rip the root filesystem, test modifications, and write useful tweaks. (And in case of a brick, read my previous post on the bootloader access). Some things I would have to see or do is GUI plugins in the device's operating system. The Java code is easy to decompile and read as the variable names have not been stripped out (like previous models). Hopefully people can write some reader plugins (like X-Ray) or even format plugins for other ebook formats. Being a touch screen device, one could also write games or useful apps (although the speed and eink are limiting). I need to finish writing the update creation tool so developers can package their modifications.

**Download**

Download the jailbreak here

Simple custom screensaver mod

Simple usbnet update (supports wifi ssh and resetting root password)

GUI menu launcher and screen rotation hack

**Demonstration **

---