Hands-on JTAG for Fun and Root Shells

Joe FitzPatrick (@securelyfitz)
Matt King (@syncsrc)

SECURINGHARDWARE.COM

Introduction

JTAG may be almost 30 years old with little change, but that doesn't mean most people really understand what it does and how. This workshop will start with a brief introduction to what JTAG really is, then quickly dive into some hands-on practice with finding, wiring, and finally exploiting a system via JTAG.

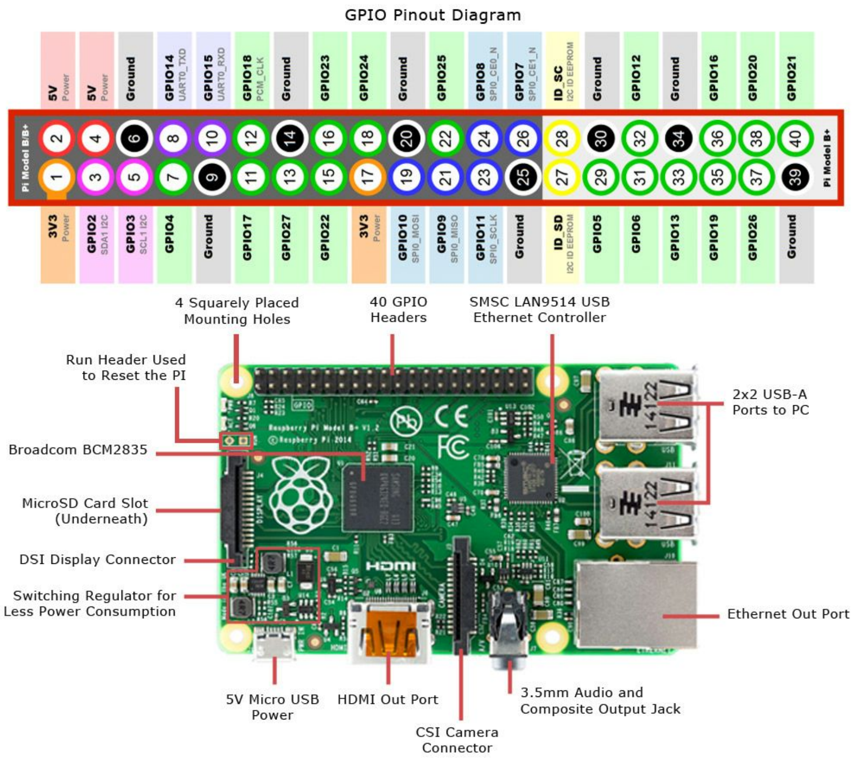
For this UK-themed workshop, we'll target a Raspberry Pi (Cambridge) with an ARM (also Cambridge) microprocessor. In order to interact with the system, we'll use a JTAG interface cable from FTDI (Glasgow). We won't do any hardware modifications, but we will hook up wires in weird and wonderful ways to make the Raspberry Pi do things it otherwise shouldn't

First, we’ll review all the hardware and software we’ll be using. Then, we’ll use a serial cable to connect to our Raspberry Pi. Once connected, we’ll run some code that will enable the JTAG pins. Next, we’ll configure some tools to access that JTAG port, and finally, we’ll use those tools to enable us to escalate privilege on the Raspberry Pi.

The techniques in this workshop are targeted at a Raspberry Pi running Raspbian Linux, but are generally portable to other cpu architectures and different operating systems.

Hardware

Raspberry Pi: Our target system is a UK-designed and UK-manufactured Raspberry Pi B+. The Raspberry Pi is based on the Broadcom BCM2835 with a 700MHz ARM cpu and dedicated graphics core. The B+ has a 40-pin GPIO header, 4 USB ports, and built-in ethernet.

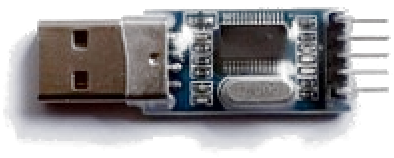


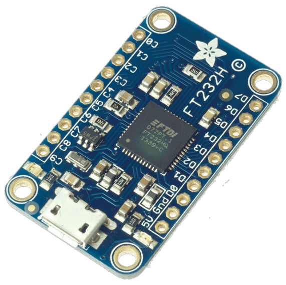
http://www.jameco.com/Jameco/workshop/circuitnotes/raspberry_pi_circuit_note.html

MicroSD card: The RPi doesn't have it's own storage but has a MicroSD slot. These cards contain the latest version of Raspbian from 5 MAY 2015, based on Debian Wheezy, with a couple extra files added to make our job easier.

https://www.raspberrypi.org/downloads/raspbian/

USB Serial Cable: While the RPi supports displays and USB keyboards, we'll interface with it through a serial console. The USB Serial Cable provides both 3.3 and 5V supplies and UART, TX, and RX signals. This is sufficient for us to power and interface with the RPi.





FT232H Board: This is an all-purpose breakout board for the FTDI 232H chip, the bigger brother to the 232R found in many usb-serial adapters. This one has the ability to communicate in a wide array of different serial protocols, including UART, I2C, SPI, and, of course JTAG which is what we will use.

USB Micro Cable: This is to connect the FT232H board to your computer. We don't need to connect anything to the RPi USB connector since it will be powered by our serial cable.

Jumper Wires: These female-female jumper wires are used to connect power, ground, and data lines between the RPI's UART and JTAG connectors. F-F jumper wires are convenient because it's nearly impossible to short things with stray wires.

Software

Screen: Screen is a terminal multiplexer, but is also handy for connecting to serial consoles on *nix systems. Other options include Minicom, Hyperterminal, and Putty.
To connect to a serial port:

```
screen /dev/ttyUSB0 115200
```

To close and quit screen:

```
<ctrl-a> <\> <y>
```

OpenOCD: This **Open**-source software is used to enable **O**n **C**hip **D**ebugging on a huge variety of CPUs, microcontrollers, and SOCs. It is highly configurable and can use a wide number of interfaces (we'll use the Adafruit FT232H board) and targets (the Broadcom chipset in the RPi). Each has a configuration file that can define details of your setup at runtime.
OpenOCD has a telnet command-line interface, a scripting interface, and also provides a GDB interface to your JTAG target system.
For this workshop, we will use OpenOCD 0.9.0. To invoke it:

```
openocd -f <configfile1> -f <configfile2>
```

GDB: Gnu Debugger is the standard debugger for GNU but has been ported to a huge range of architectures. There are plenty of forks and GUI versions, but we will use the standard command-line version.
Instead of connecting GDB to a process like normal software debugging, we will connect GDB to OpenOCD to debug via JTAG. Since we're debugging an ARM target on an X86 system, we'll use **`gdb-multiarch`**.
Once we've started, we can connect to OpenOCD's GDB server:

```
target remote localhost:3333
```

Getting Started with UART

Now it's time to start setting things up!

1. Insert the MicroSD card into your Raspberry Pi
2. Find the labels on your USB-Serial Adapter. We need to connect Power, Ground, TX and RX to our Raspberry Pi.
3. Based on the labels and the diagram on page 2, complete the table below:

USB-Serial	Cable Color	Raspberry Pi
3.3V	---	NO CONNECT
5V		
TX		
RX		
GND		

4. Connect all the color ribbon cables to the correct pins
5. Once you have double checked your connections, plug the USB-Serial adapter into your computer. The Raspberry Pi. A red light should turn on, and a green light will start to flash.
6. Start screen to connect to your serial console:

   ```
   > screen /dev/ttyUSB0 115200
   ```

   If you get a black screen, hit <enter> a few times. Still nothing? Perhaps you got TX and RX mixed up. Ask an instructor if you're having trouble.
7. You probably see a login prompt. The default username is "pi" and the password is "raspberry"

Congratulations! You're now connected to the Raspberry Pi! Take some time to poke around. Browse the file system, see what architecture, CPU and memory you've got and what devices are attached, etc. If you aren't familiar with linux, ask an instructor or a neighbor for some help.

Enabling JTAG

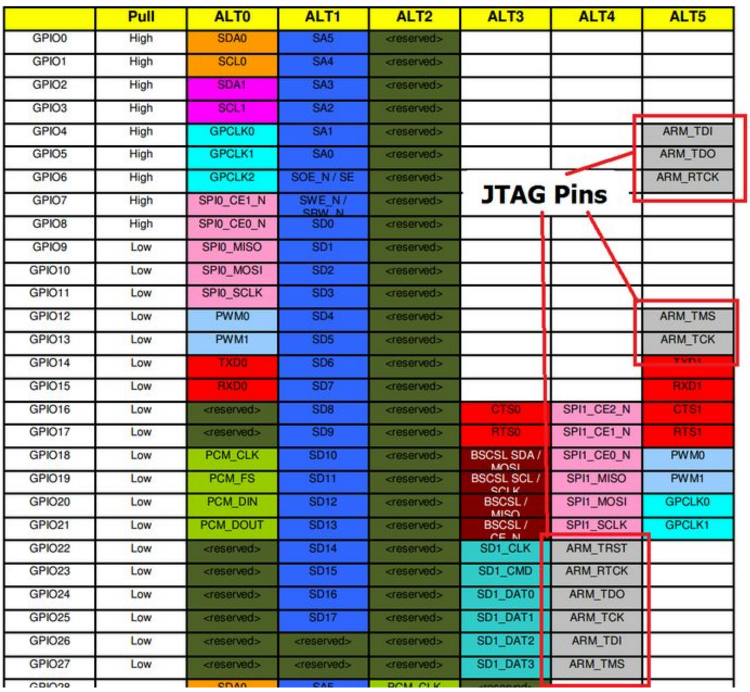
Now that we're logged into our RPi, we need to do some configuration in order to use JTAG. The BCM2835 has lots of multi-purpose IO pins, we need to tell it to use some of them for JTAG. Each GPIO pin has several different alternate modes, labelled ALT0 to ALT5. We need to set the right GPIOs to the right ALT mode so that we will connect the necessary JTAG signals - TDI, TDO, TMS, TCK, and TRST - to the right GPIO pins and the right headers on the board.

Many of these steps are based on information from http://sysprogs.com/VisualKernel/tutorials/raspberry/jtagsetup/

1. We need to map out what pins are what! We'll fill out the following table over the next few steps:

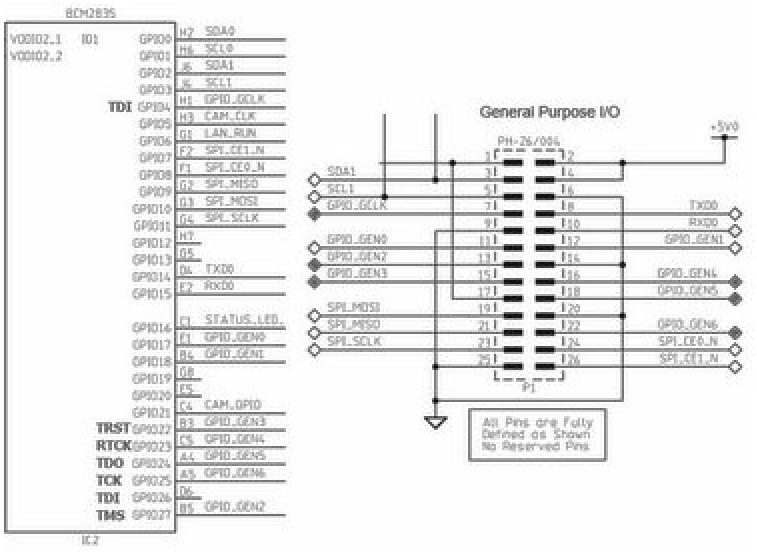
JTAG Pin	**ALT4 GPIO**	**ALT5 GPIO**	**P1 Pin**	**Mode**
GND				
TDI				
TDO				
TMS				
TCK				
TRST				

2. First, let's figure out what pins can be JTAG. Based on the BCM2835 Datasheet, let's fill in the 'ALT4' and 'ALT5' columns with GPIO## numbers



	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO0	High	SDA0	SA5	<reserved>			
GPIO1	High	SCL0	SA4	<reserved>			
GPIO2	High	SDA1	SA3	<reserved>			
GPIO3	High	SCL1	SA2	<reserved>			
GPIO4	High	GPCLK0	SA1	<reserved>			ARM_TDI
GPIO5	High	GPCLK1	SA0	<reserved>			ARM_TDO
GPIO6	High	GPCLK2	SOE_N / SE	<reserved>			ARM_RTCK
GPIO| GPIO7	High	SPI0_CE1_N	SWE_N / SRW_N	<reserved>			
GPIO8	High	SPI0_CE0_N	SD0	<reserved>			
GPIO9	Low	SPI0_MISO	SD1	<reserved>			
GPIO10	Low	SPI0_MOSI	SD2	<reserved>			
GPIO11	Low	SPI0_SCLK	SD3	<reserved>			
GPIO12	Low	PWM0	SD4	<reserved>			ARM_TMS
GPIO13	Low	PWM1	SD5	<reserved>			ARM_TCK
GPIO14	Low	TXD0	SD6	<reserved>			TXD1
GPIO15	Low	RXD0	SD7	<reserved>			RXD1
GPIO16	Low	<reserved>	SD8	<reserved>	CTS0	SPI1_CE2_N	CTS1
GPIO17	Low	<reserved>	SD9	<reserved>	RTS0	SPI1_CE1_N	RTS1
GPIO18	Low	PCM_CLK	SD10	<reserved>	BSCSL SDA / MOSI	SPI1_CE0_N	PWM0
GPIO19	Low	PCM_FS	SD11	<reserved>	BSCSL SCL / SCLK	SPI1_MISO	PWM1
GPIO20	Low	PCM_DIN	SD12	<reserved>	BSCSL / MISO	SPI1_MOSI	GPCLK0
GPIO21	Low	PCM_DOUT	SD13	<reserved>	BSCSL / CE_N	SPI1_SCLK	GPCLK1
GPIO22	Low	<reserved>	SD14	<reserved>	SD1_CLK	ARM_TRST	
GPIO23	Low	<reserved>	SD15	<reserved>	SD1_CMD	ARM_RTCK	
GPIO24	Low	<reserved>	SD16	<reserved>	SD1_DAT0	ARM_TDO	
GPIO25	Low	<reserved>	SD17	<reserved>	SD1_DAT1	ARM_TCK	
GPIO26	Low	<reserved>	<reserved>	<reserved>	SD1_DAT2	ARM_TDI	
GPIO27	Low	<reserved>	<reserved>	<reserved>	SD1_DAT3	ARM_TMS	

3. Next, let's map those GPIO pins to the pins on the RPi's 40-pin header using the RPi schematic. Fill in the 'P1 pin' column with the appropriate pin numbers:



4. Now, we need to decide which mode to set each pin to. In the table, enter ALT4 or ALT5 in the ‘mode’ column, depending on which pin is connected to a GPIO.
5. We’ve figured out how we want to set up our GPIOs, now we need to write some code to set the correct ALT modes for different pins.
   In the user directory on the RPi, you should find a JtagEnabler.cpp file. Open it, and scroll down to the main() function.
   For each of the pins we want to use, we need to call SetGPIOFunction with the pin number and the alt mode.
   For example, to set GPIO22 to ALT4, we need:
   ```
   selector.SetGPIOFunction(22,
   GPIO_ALT_FUNCTION_4);
   ```
6. You should have 5 lines total. When done, save your file, and compile it:
   ```
   g++ -o JtagEnabler JtagEnabler.cpp
   ```
7. If you were successful, run your program:
   ```
   sudo ./JtagEnabler
   ```

We should now have functioning JTAG on our RPi. Let’s move on to the next steps to see if we can connect to it to debug our target. Note that the way we’ve done it, our GPIO settings will not persist a reboot - we have to re-run JtagEnabler to re-enable the pins. If we wanted to, we could set this in an initialization script, or we could modify the kernel to do the same thing automatically on boot.

Using JTAG

We took several steps to enable JTAG on our target. Some systems will require this process, others will have a hardware setting, and many will simply have dedicated JTAG pins that are always available. Now we will move on to configuring our hardware and software to enable JTAG debugging.

1. The first step is to connect the JTAG wires between our JTAG adapter and target system properly:

JTAG Pin	Adafruit FT232H	Wire Color	RPi P1 pin
GND	GND	Black	9
TCK	D0	White	22
TDI	D1	Grey	7
TDO	D2	Purple	18
TMS	D3	Blue	13
TRST	D4	Green	15

2. After double checking your wiring, connect the FT232 board to your laptop with the USB Micro cable.
3. Next, we need to start OpenOCD. We need to pass it two configuration files:
    a. ft232.cfg is the configuration file for the JTAG adapter. If you look inside you'll see details about how OpenOCD uses different pins on the board
    b. raspi.cfg is the configuration file for the Raspberry Pi. If you look inside, you'll see details about how OpenOCD identifies the part, recognizes the architecture details, and handles some quirks about the chip
    c. Start OpenOCD with both config files:
    ```
    openocd -f ft232h.cfg -f raspi.cfg
    ```
    d. OpenOCD is often cryptic with it's error messages. If it was successful, It will tell you that it found and ARM cpu with multiple breakpoint controllers.

   e. If you were successful - quit with **`<ctrl-c>`**, and re-run your OpenOCD command line. There's a bug that makes it necessary to disconnect and reconnect to the raspberry pi like this before it works properly.
4. OpenOCD is a server - once it's started we need a client to interface with it. We'll start with telnet to use the command line interface.
   a. Connect with telnet to your server:
   `telnet localhost 4444`
   b. Try halting your CPU with the **`halt`** command. Test that it works by trying to interact with your RPi over the serial cable. Then use the **`resume`** command to pick up where you left off.
   c. Halt your CPU again, and then try to examine your registers by using the **`reg`** command. What register contains the address of the instruction that will be executed next?
   d. With your CPU still halted, let's examine memory with the **`mdw`** command, which stands for 'memory display word'. Give it an address and optionally a number, and it will display that many words of memory starting at that address.
5. The OpenOCD command line is helpful for many basic tasks, but a fully-featured debugger is much easier for more advanced tasks. OpenOCD acts as a GDB server so that you can use GDB to debug your JTAG target.
   a. In a new window, run: **`gdb-multiarch`**
   b. Set the architecture to ARM:
   `set arch arm`
   c. Connect to OpenOCD:
   `target remote localhost:3333`

   If you're familiar with GDB, have fun and play!

JTAG Exploit 1

Using tools the way you're supposed to is all well and good, but let's use the same tools to do some cool privilege escalation!

1. On your RPi, try accessing the /etc/shadow file:
   `cat /etc/shadow`
   Denied! Sure you could just sudo, but why bother when you can use JTAG instead?
2. Before we scour the entire address space, let's ask nicely to find something vulnerable in memory:
   `cat /proc/kallsyms |grep generic_permission`
   generic_permission is what's called to check if we're allowed to read that shadow file. What if it always returned a value that granted access?
3. If you're not already set up, power on your RPi, enable JTAG, start OpenOCD (then quit, and restart it, remember?)
4. Telnet to OpenOCD and **`halt,`**then start GDB.
5. Let's look at memory at the address of generic_permission:
   `x/50i <generic_permission>`
6. The first thing this function does is preserve registers on the stack - a very long push instruction. That means, before it returns, it will pop all those values. Where are there long ldm (load multiple) instructions?
7. We're looking for a non-zero return value in register r0. Which of the two returns is likely the failing case?
8. Let's put a breakpoint on the address of the failing case ldm instruction: **`b *0x<address>`**
   then continue: **`cont`**
9. On your RPi, try to access /etc/shadow:
   `cat /etc/shadow`
   Your breakpoint should hit!
8. In GDB, let's clear our bad retval from r0: **`set $r0=0`**
   remove the breakponts: **`d b`**
   and continue: **`cont`**
   Did it work?

JTAG Exploit 2

We might have seen after our last exploit that root had no password set, so there's no way to log in as root - unless we use JTAG to mess with the way login happens!

1. On your RPi, logout: **`exit`**
2. Try to login as root. You should be prompted for a password, which doesn't exist. getty is the userspace program that handles this - what if we modify it? from the source:

```
/* Let the login program take care of password
validation. */
(void) execl(options.login, options.login, "--",
logname, (char *) 0);
```

   That' "--" is the key - if we change it to "-f" then it forces authentication.
3. Examine ocd_rpc_getty.py
    a. After some header information, the OpenOcd class defines how to interact with OpenOCD
    b. The main function parses arguments, then hunts through memory, reading the same offset at each page
    c. If the signature matches, it patches - by writing 0x66 or 'f' in the write spot
4. If you're not already set up, power on your RPi, enable JTAG, start OpenOCD (then quit, and restart it, remember?)
5. Run ocd_rpc_getty.py. We're targeting raspian, and getty is probably somewhere higher in memory:
   ```
   ./ocd_rpc_getty.py -s 0xd80000000 -t
   raspbian
   ```
6. When the script completes, it has patched an instance of getty. Try logging in as root to see if it was the right one. If not, run the script again (update the start address to speed things up)