Hands-on JTAG for Fun and Root Shells v2

Joe FitzPatrick (@securelyfitz)
Piotr Esden-Tempski (@esden)
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.

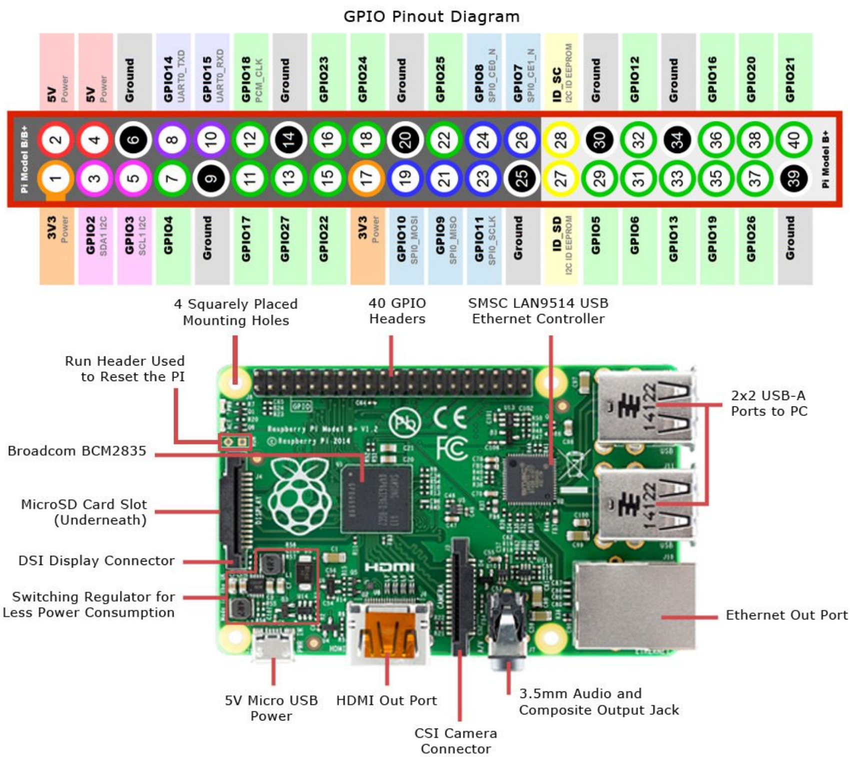
For this workshop, we'll target a Raspberry Pi 2. In order to interact with the system, we'll use a Black Magic Probe V2 JTAG adapter. We won't do any hardware modifications, but we will hook up wires in weird and wonderful ways to make the Raspberry Pi do things it otherwise shouldn't.

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 2. 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 2.

The techniques in this workshop are targeted at a Raspberry Pi 2 running Raspbian Linux, but are generally portable to other cpu architectures and different operating systems.

Hardware

Raspberry Pi 2: Our target system is a Raspberry Pi B+. The Raspberry Pi 2 is based on the Broadcom BCM2836 with a 900MHz quad-core ARM CPU, dedicated graphics core, 40-pin GPIO header, 4 USB ports, and built-in ethernet.

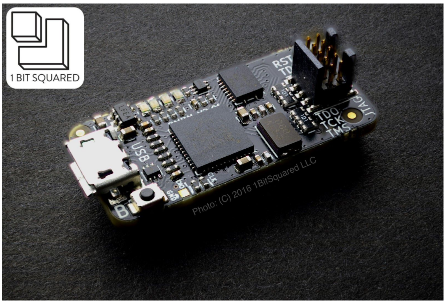


http://www.jameco.com/Jameco/workshop/circuitnotes/raspberry_pi_circuit_note.html

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 2016, based on Debian Jessie, with a couple extra files added to make our job easier.

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

Black Magic Probe Mini v2 (BMPM2): This tool implements the JTAG protocol and interfaces to the debug port of the Raspberry PI 2 Broadcom chip. It implements all the protocol layers of the ARM Debug Interface specification V5 used by many ARM Cortex



series of cores. It presents the JTAG interface as a GNU Debugger server over serial to the PC. Additionally it also comes with a USB to Serial adapter functionality that makes the access to the serial console of the RPi very convenient.
http://1bitsquared.com/products/black-magic-probe

USB Micro Cable: These are to connect the BMPM to your computer and to connect power to the Raspberry Pi 2.

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/ttyACM1 115200
```

To close and quit screen:

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

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 the Black Magic Probe to debug via JTAG.
Since we're debugging an ARM target on an X86 system, we'll use `arm-none-eabi-gdb` that is part of the GCC ARM Embedded toolchain from launchpad:
https://launchpad.net/gcc-arm-embedded
Once we've started, we can connect to the Black Magic Probe's GDB server and ask the BMPM about it's firmware version:

```
target extended-remote /dev/ttyACM0
monitor version
```


Getting Started with UART

Now it's time to start setting things up!

1. We need to connect Power, Ground, TX and RX to our Raspberry Pi. Keep in mind that TX on BMPM connects to the RX on the RPi2 and vice versa. Using labels and the diagram on page 3, complete the table below:

USB-Serial	Cable Color	Raspberry Pi 2
3.3V	RED	
TX	GREEN	
RX	PURPLE	
GND	BLACK	

2. Connect the serial wires from the Black Magic Probe to the correct pins on the Raspberry Pi 2, double check your connections, and plug the BMPM into your computer. Green & orange LEDs turn on & dim after 15 seconds.
3. Start screen to connect to your serial console:
   > **`screen /dev/ttyACM1 115200`**
4. Turn on the RPi2 by plugging in the micro USB power cable. A red LED will light up and a green one will start flashing.
5. The serial console will printing Linux Kernel boot messages. If you get a black screen, hit **`<enter>`** a few times. Still nothing? Try swapping the TX and RX pins.
6. You should see a login prompt. The default username is "pi" and the password is "raspberry"

Congratulations! You're now connected to the Raspberry Pi 2! Take some time to poke around. Browse the filesystem, 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 BCM2836 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/

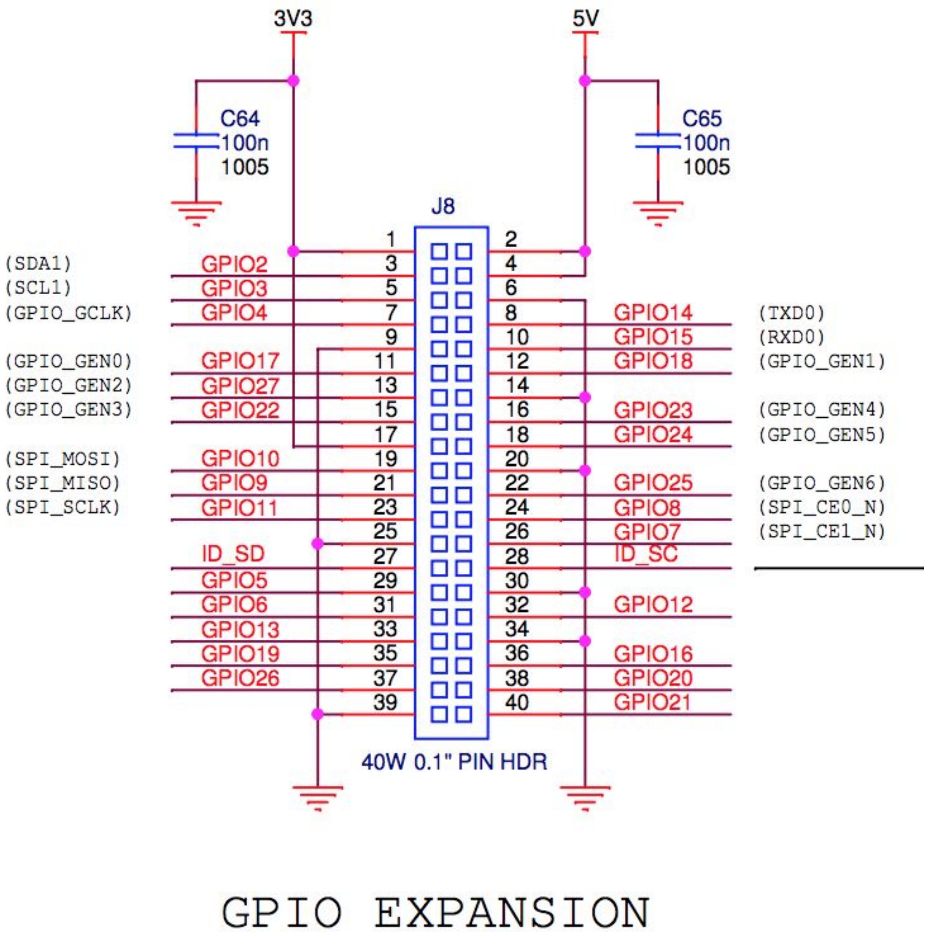
1. We need to map out what pins are what! We'll fill out the following table over the next few steps:

JTAG Pin	**ALT4 GPIO**	**ALT4 Pin**	**ALT5 GPIO**	**ALT5 Pin**	**ALT4/ ALT5?**
GND					
TDI	GPIO26	37			
TDO					
TMS					
TCK					
TRST					

2. First, let's figure out what pins can be JTAG. Based on the BCM2836 Datasheet, let's fill in the 'ALT4' and 'ALT5' columns with GPIO## numbers. For example, find TDI in the ALT4 Column. Look at the GPIO## on that line and enter that in the table.

ALT3	ALT4	ALT5	
			GPIO0
			GPIO1
			GPIO2
			GPIO3
		ARM_TDI	GPIO4
		ARM_TDO	GPIO5
		ARM_RTCK	GPIO6
			GPIO7
			GPIO8
			GPIO9
			GPIO10
			GPIO11
		ARM_TMS	GPIO12
		ARM_TCK	GPIO13
		TXD1	GPIO14
		RXD1	GPIO15
CTS0	SPI1_CE2_N	CTS1	GPIO16
RTS0	SPI1_CE1_N	RTS1	GPIO17
SCSL SDA / MOSI	SPI1_CE0_N	PWM0	GPIO18
SCSL SCL / SCLK	SPI1_MISO	PWM1	GPIO19
BSCSL / MISO	SPI1_MOSI	GPCLK0	GPIO20
BSCSL / CE_N	SPI1_SCLK	GPCLK1	GPIO21
SD1_CLK	ARM_TRST		GPIO22
SD1_CMD	ARM_RTCK		GPIO23
SD1_DAT0	ARM_TDO		GPIO24
SD1_DAT1	ARM_TCK		GPIO25
SD1_DAT2	ARM_TDI		GPIO26
SD1_DAT3	ARM_TMS		GPIO27
<reserved>			GPIO28

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 pin column with the appropriate pin numbers. For example, find GPIO26 in the schematic. It’s connected to header pin 37, so enter that in the table.



4. Now, we need to decide which mode to set each pin to. In the table, enter ALT4 or ALT5 in the last 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 2. 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 the Black Magic Probe and target system properly:

JTAG Pin	Black Magic Probe	Wire Color	RPi P1 pin
GND	16	Black	9
TCK	9	White	22
TDI	5	Grey	7
TDO	13	Purple	18
TMS	7	Blue	13
TRST	15	Green	15
VCC	1	Red	1

2. After double checking your wiring, connect the Black Magic Probe board to your laptop with the USB Micro cable.
3. The Black Magic Probe provides a GDB server so that you can use GDB to debug your JTAG target. Let's get that configured properly:
   a. In a new window, run: **`arm-none-eabi-gdb`**
   b. Connect to the Black Magic Probe:
      `target extended-remote /dev/ttyACM0`
4. Black Magic Probe firmware provides a few custom GDB commands that are accessible with the **`monitor`** command. Test your connection to the BMP with **`monitor version`** to see firmware version information
5. You can list all the commands that BMP implements with **`monitor help`**

6. The most important command for us is **`monitor jtag_scan`** Which will scan for target devices connected to JTAG, in our case RPi2 Broadcom CPU. Do you see a list of 4 available targets?
7. These are the 4 CPU cores found in the Broadcom BCM2836 SOC. We can attach to one of these cores the same way we would attach to a process thread. Attach to and halt the first core with **`attach 1`** so that we can inspect registers and memory on that device. Keep in mind the other 3 cores will continue running.
8. If you're familiar with GDB, have fun and play! If you need some guidance, you can try:
    a. Type **`i r`** to get **i**nformation about **r**egisters
    b. Type **`print $pc`** to **print** the **p**rogram **c**ounter
    c. Type **`x/10w $pc`** to display(**x**) **10 w**ords at the address in the **p**rogram **c**ounter
    d. Type **`x/10i $pc`** to display 10 **i**nstructions at the address in the program counter
    e. Type **`stepi`** to execute one instruction and halt
    f. Type **`nexti`** to execute one instruction but step over subroutine calls.
    g. Type **`disp/10i $pc`** to automatically **disp**lay 10 instructions every time you **`stepi`** or **`nexti`**
    h. Type **`hb *0x<address>`** to set a **h**ardware **b**reakpoint at that address
    i. Type **`cont`** to **cont**inue execution.
9. The last command we would like to highlight is **`detach`**. This command will let you “disconnect” from the current CPU you are attached to and allow you to connect to a different one without having to restart GDB
10. Continue to play around with the commands and get familiar with them.

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, then start GDB.
4. Let's look at memory at the address of generic_permission:
   x/60i <generic_permission>
5. The first thing a function does is preserve registers on the stack - a very long push instruction. The last thing it does is a long **ldm** (load multiple) instruction to restore them. Do you see any long **ldm** instructions?
6. We're looking for a non-zero return value in register r0. Which of the two returns is likely the failing case?
7. Let's put a breakpoint on the address of the failing case ldm instruction: **hb *0x<address>**
   then continue: **cont**
8. On your RPi, try to access /etc/shadow repeatedly:
   **while true; do cat /etc/shadow; **
   if [$? == 0] ; then break; fi; done
   Your breakpoint will hit when **cat** is run on our core.
8. In GDB, let's clear our bad retval from r0: **set $r0=0**
   remove the breakpoints: **d br**
   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 so login doesn't ask for a password.
3. Examine the source code for **`getty_patch.gdb`**
   a. This GDB script connects to BMP and starts reading the same offset at each page through memory
   b. If the signature matches, it patches '**`--`**' to "**`-f`**" by writing 0x66 or '**`f`**' in the right spot
4. If you're not already set up, connect your BMP, power on your RPi.
5. getty is probably somewhere higher in memory - how can we tell the script to start searching at a later point in memory? Try it and run: **`./getty_patch.gdb`**
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)

Notes:

If you are trying to reproduce this class at home, you can find all the class resources at: http://github.com/esden/jtagsploitation if you have any questions or suggestions join our Gitter channel at http://gitter.im/esden/jtagsploitation If you have specific questions regarding Black Magic Probe you can find us in http://gitter.im/blacksphere/blackmagic Gitter channel.

