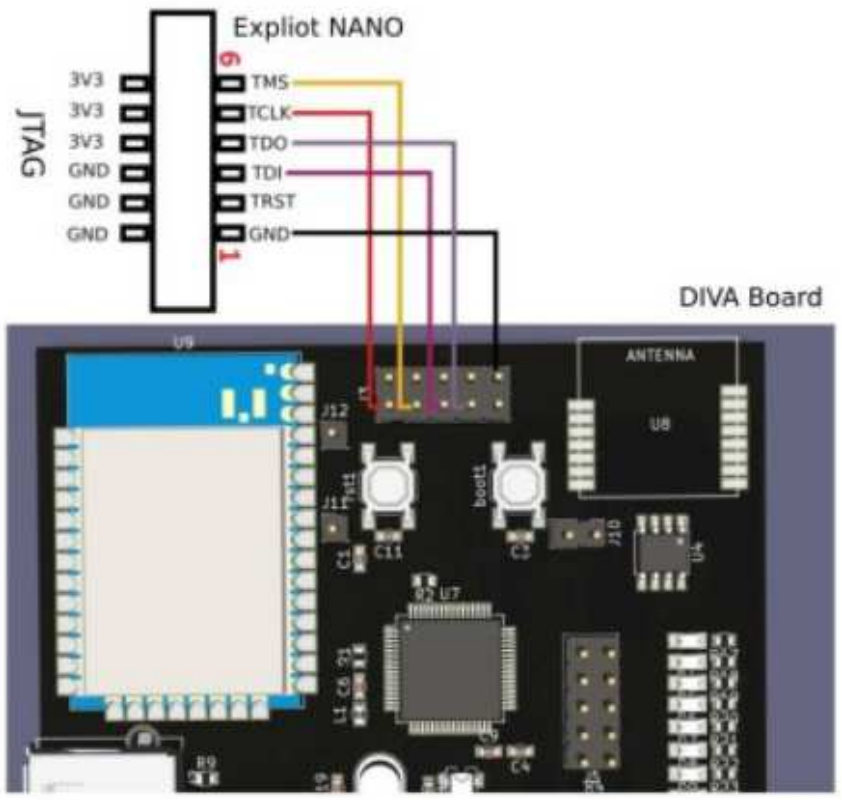
IoT Security - Part 18 (101 - Hardware Attack Surface: JTAG, SWD)

Asmita-Jha

14-October-2020



This blog is part of the IoT Security series, where we discuss the basic concepts about the IoT/IIoT eco-system and its security. If you have not gone through the previous blogs in the series, I will urge you to go through those first. In case you are only interested in hardware JTAG, SWD, feel free to continue.

IoT Security - Part 1 (101 - IoT Introduction And Architecture)

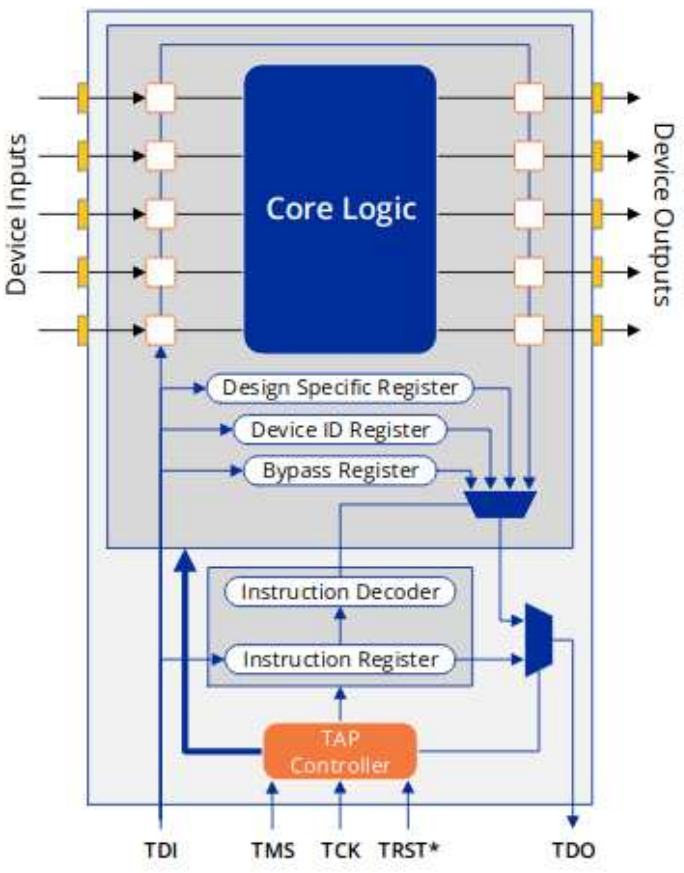
IoT Security - 17 (101 - Hardware Attack Surface: UART)

If you are a beginner at hardware hacking and looking for some points to start with, this blog series can help you. We discussed JTAG and SWD as a hardware attack surface in the IoT attack surface blog here. In this blog, we will discuss them in detail. Along with their basic introduction, we will discuss the possible attack scenarios, tools, and methods that you can use to attack the device if you get access to the JTAG/SWD interface on the hardware. So, if you are new to it and want to get started, stay tuned :).

Introduction

JTAG

JTAG (Joint Test Action Group), an industry-standard, was developed by the association, the Joint (European) Test Access Group, in 1985. It was originally developed for verifying designs and testing of printed circuit boards (PCB). Later in 1990, it was standardized as an IEEE standard 1149.1-1990 as **Standard Test Access Port and Boundary-Scan Architecture**. Initially, with the invention of integrated circuits, increasing complexity, higher density, and smaller components, it was not easy to test the physical interconnections on PCB. JTAG boundary scan came out as a solution to perform the testing and debugging of chips' physical interconnection by limiting physical access to just a few signals. Today, JTAG is used for many other applications, including in-circuit debugging, giving access to directly communicate with the memory/registers within the chip without direct external access to the system address or the data bus, and for programming devices. The image below shows the architecture of the JTAG chip (Source - here).



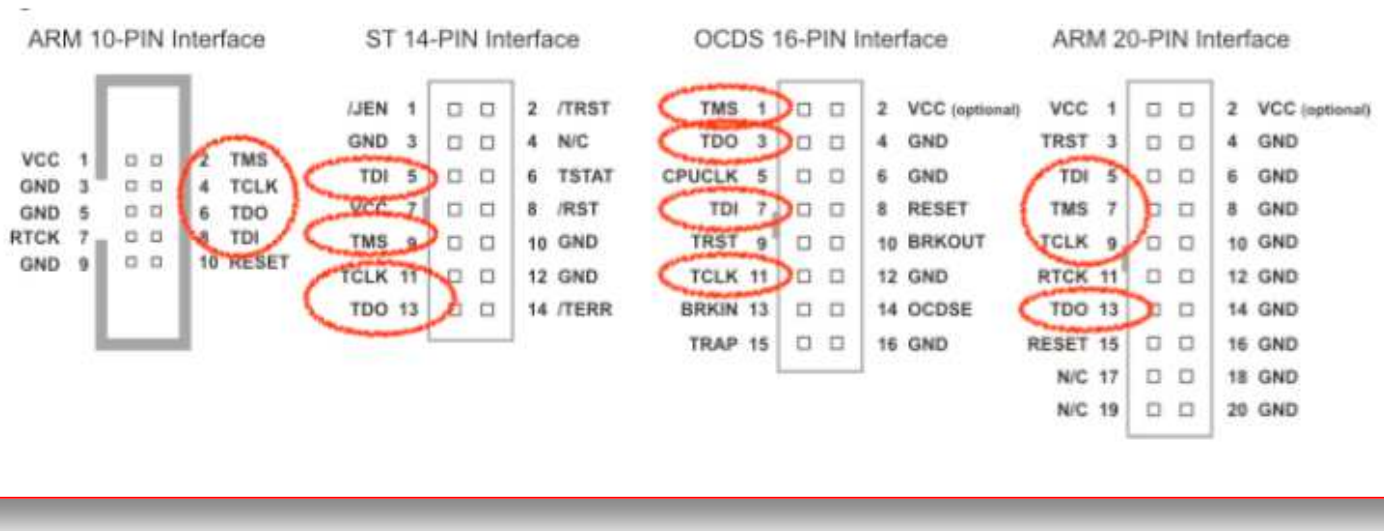
The JTAG chip consists of logic cells or boundary-scan cells that connect the chip to the PCB. These can capture data from pin or core logic signals as well as send data onto pins. These logic cells are accessed through a serial test data input (TDI) and test data output (TDO) interface. The test controller's primary interface that provides access to the logic is the Test Access Port (TAP), consisting of 4 required signals and an optional reset signal.

- TCK (Test Clk) - Test clock synchronizes the operations of the internal state machine. The JTAG standard does not specify the actual clock speed.

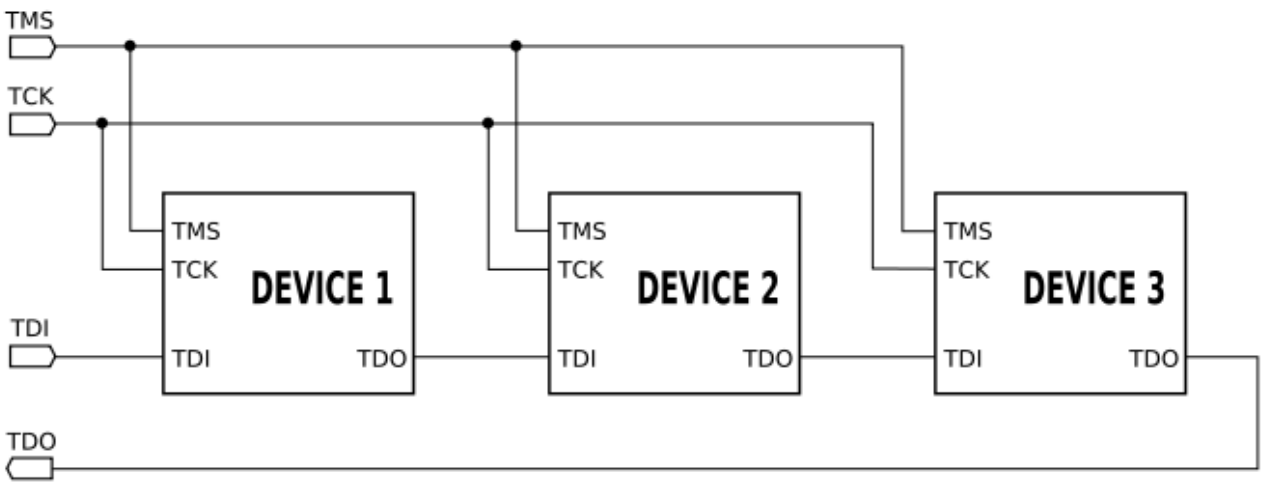
- TMS (Test Mode Select) - It controls the JTAG state machine. It is sampled at the rising edge of the TCK to determine the next state of the state machine.
- TDI (Test Data In) - Sends data into the chip. When the internal state machine is in the correct state, it is sampled at the TCK's rising edge.
- TDO (Test Data Out) - Data coming out of the chip. When the internal state machine is in the correct state, it is valid on TCK's falling edge.
- TRST (Test Reset : Optional) - Resets the TAP controller state machine.

As defined by the IEEE-1149.1 standard, the TAP controller uses a 16-state finite state machine controlled by TCK and TMS signals. The state of TMS on the rising edge of TCK determines the transition to the next state. Each JTAG TAP has an Instruction Register (IR) and a Data Register (DR). The size of these registers is variable. The state machine selects the operations/ instructions via IR and passes the parameters or data update via DR. The detailed working of the state machine can be read from the reference [1].

JTAG specification does not have the defined protocol for the connector design. JTAG connectors can be found varying from 6, 10, 14, 16, 20, etc. numbers of pin interfaces. These connectors can have extra signals apart from the 4 JTAG signals. as shown in the image below (Source - Reference [1])



For communicating with the device via the JTAG interface, we would have to identify the 4 JTAG signals TCK, TMS, TDI, and TDO that communicate with the JTAG chip state machine via TAP. It is also possible to communicate with more than one JTAG chips by connecting them in daisy chain as shown in the image below (Source - Wikipedia)



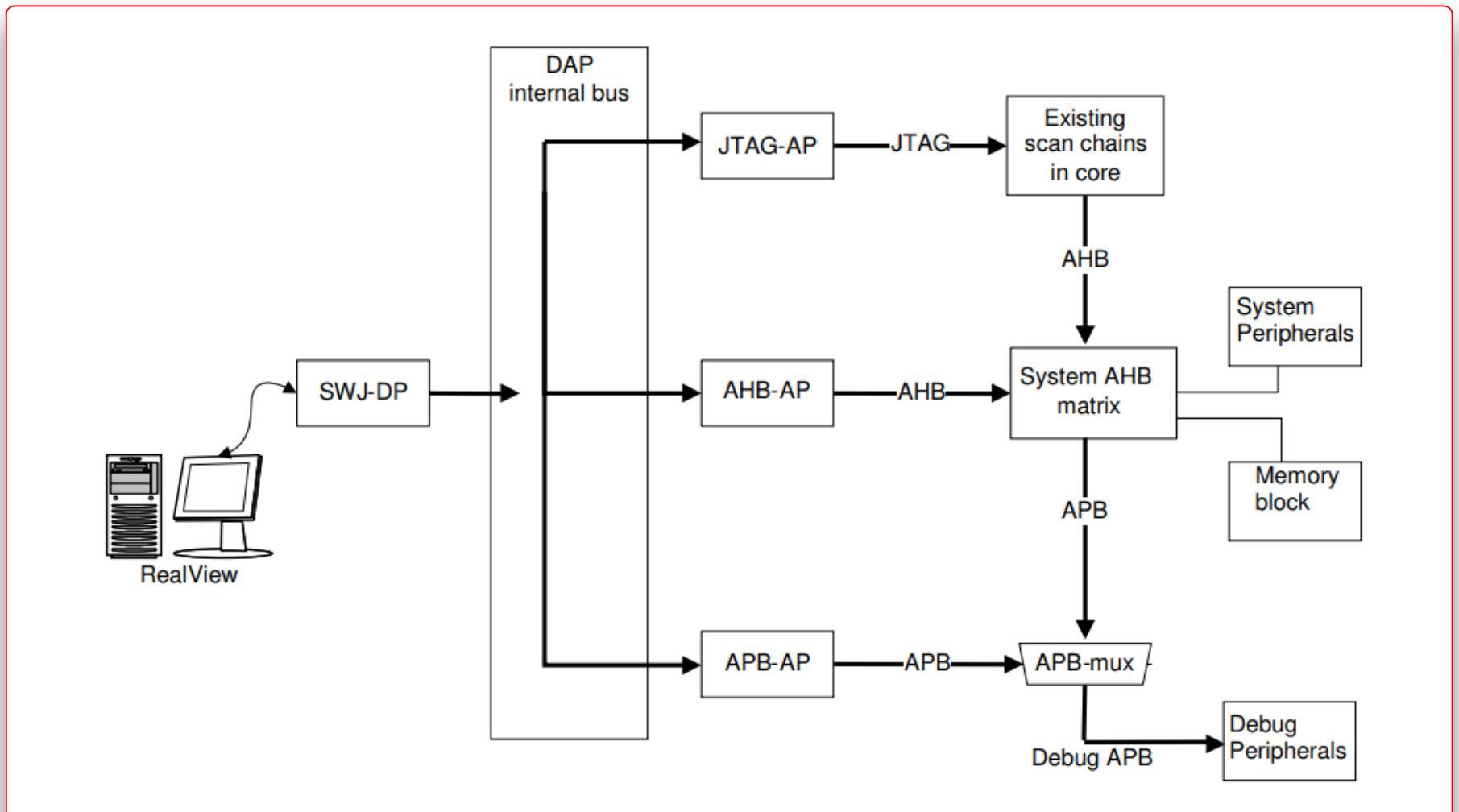
There is a reduced pin count JTAG called **compact JTAG (cJTAG)** that only has two pins, TMSC (Test Serial Data) and TCKC (Test Clock). It is defined as part of the IEEE 1149.7 standard. JTAG specification defines some mandatory boundary scan related instructions and some optional instructions. However, the TAPs that are used for debugging applications instead of boundary scan, generally provide minimal or no support for these mandatory instructions related to boundary scan. The two important instructions are :

- BYPASS - TDI and TDO are connected to a single-bit data register (also called BYPASS). This instruction allows the device to get bypassed while allowing the serial data to go through the next devices in the chain/scan path.
- IDCODE - This is an optional instruction but used widely (not universally). It is related to a 32-bit device ID register (IDCODE). It includes the chip ID i.e., the manufacturer code in the standardized format.

After the reset state, IR is preloaded with BYPASS or IDCODE instruction. Such identification can also help in identifying the kind of processor/microcontroller used in the chip. Other instructions include EXTEST, SAMPLE, PRELOAD, HIGHZ, INTEST, CLAMP, RUNBIST, and USERCODE. The devices provided by the manufacturer may define more instructions. JTAG supports different architectures, including ARM, Atmel AVR, TI MSP430, FPGAs, MIPS, CPLDs, etc.

SWD

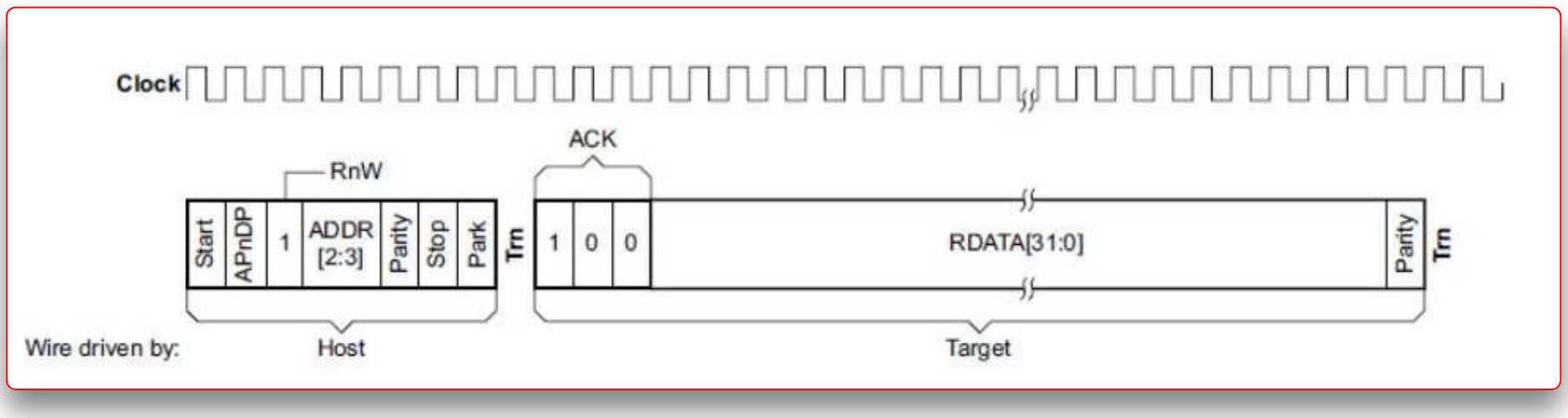
SWD (Serial Wire Debug) provides the debug port by reducing the pin count to just two, the bidirectional data signal (SWDIO), and a clock signal (SWCLK) sent by the host. It provides all the normal JTAG debug and test functionality (it does not provide the boundary scan feature as in JTAG). It uses an Arm standard bi-directional wire protocol, defined in the Arm debug interface v5. It is a standard interface for ARM processor-based devices. It is useful where limiting pin count is crucial. SWD uses a bus called Debug Access Port (DAP). DAP has one master (Debug Port - DP) and one or multiple slaves (Access Ports - APs). The image below shows the SWD architecture (Source - Reference 7).



SWD architecture

The external debugger connects to DAP using the debug port (DP). There are three debug port interfaces to access the DAP. * JTAG Debug Port (JTAG-DP) - It uses standard JTAG interface and protocol * Serial Wire Debug Port (SW-DP) - It uses SWD protocol * Serial Wire / JTAG Debug Port (SWJ-DP) - It can use either JTAG or SWD to access the DAP. In this, TCK and TMS JTAG signals are reused as SWCLK and SWDIO signals, respectively. For switching from one interface to the other, a specific sequence has to be sent.

SWD transaction has three phases. * 8-bit Request phase sent from the host * 3-bit ACK (Acknowledgement) phase sent from the target. * Data phase where up to 32 bits are transferred from/to the host with a parity bit. When the data direction has to be changed, TRN (Turnaround) cycle is sent. The image below shows the SWD read transfer (Source - Reference 7).



More conceptual details about the JTAG and SWD interface can be read from the references mentioned at the end of this blog. In the next sections, we will discuss the attack scenarios and methods.

Possible Attack Scenarios

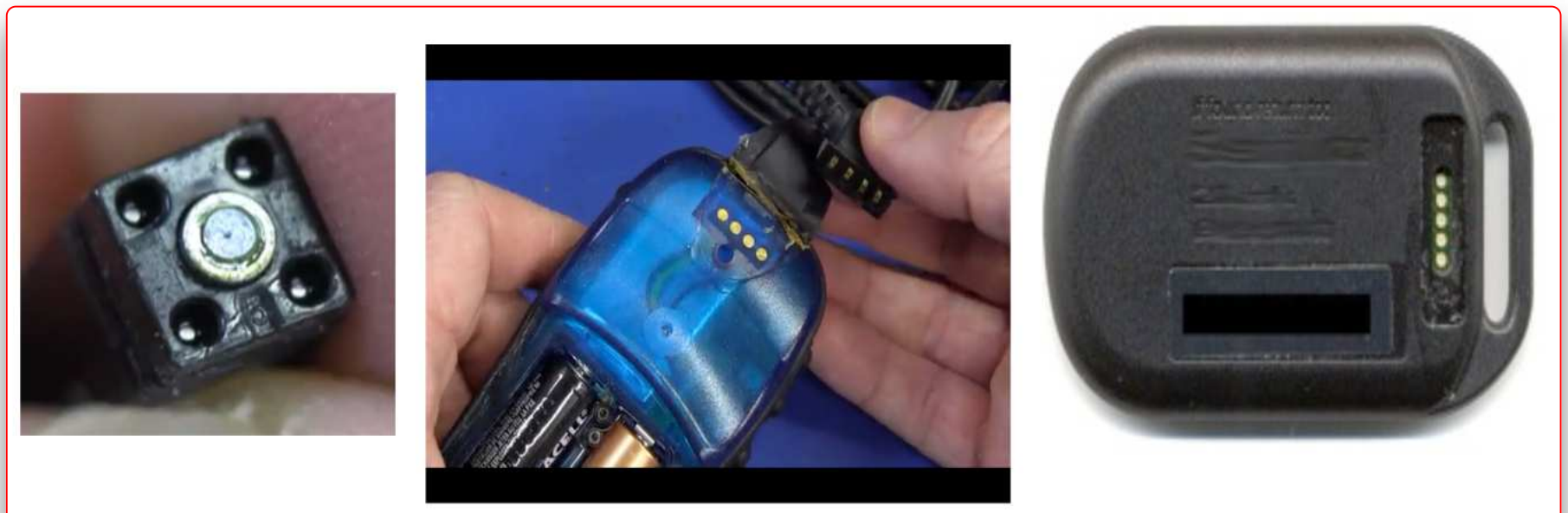
Getting access to the JTAG/SWD interface on the hardware opens many possibilities for an attacker to break into the device. The following are the possible attack scenarios :

- Attackers get access to the controller's internal memory leading to the manipulation of the register values. Manipulating the internal registers can have varying effects. For example, in some cases, even if a controller read-protection is implemented, the adversary can manipulate the register value, and there is a possibility of bypassing the protection implementation. Similarly, the attacker could also change settings and bypass protection related to the bootloader and other critical registers that the developer might have locked for security aspects.
- Having access to the hardware with the JTAG/SWD interface gives the attacker the possibility of debugging the system. It may help him/her to unearth more vulnerabilities and perform attacks on the device. For better understanding, you can refer to our blog, **"Hardware Attack - Stack Smashing And Protection"**. Here you would get a glimpse of how the access to the JTAG debug port on the hardware helped us identify a buffer overflow vulnerability leading to RCE (remote code execution)
- Another most significant advantage from the attackers' perspective is that the access to JTAG/SWD debug interface opens up the possibilities to extract the firmware from the device, patch the firmware, and re-flash the modified vulnerable/malicious firmware back into the

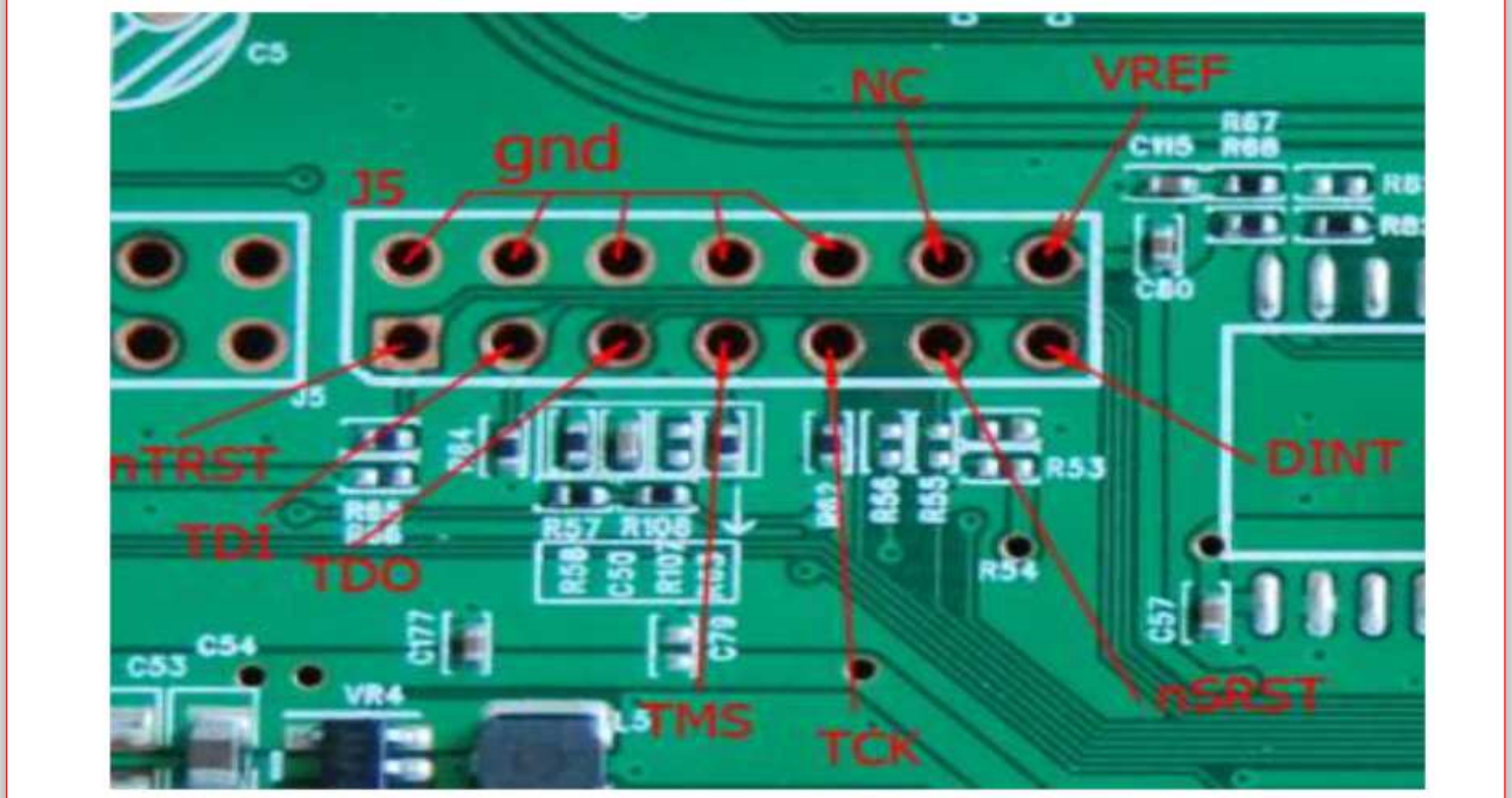
device. Getting access to the firmware opens up vast possibilities for an attacker to catch and exploit other vulnerabilities.

Performing the Attack

We first need to identify the respective port pins on the hardware to interact with the device via JTAG/SWD port. Below are few images showing how most JTAG port test points/pinouts on the hardware may look like.



(Source - here)



(Source - here)

Recon

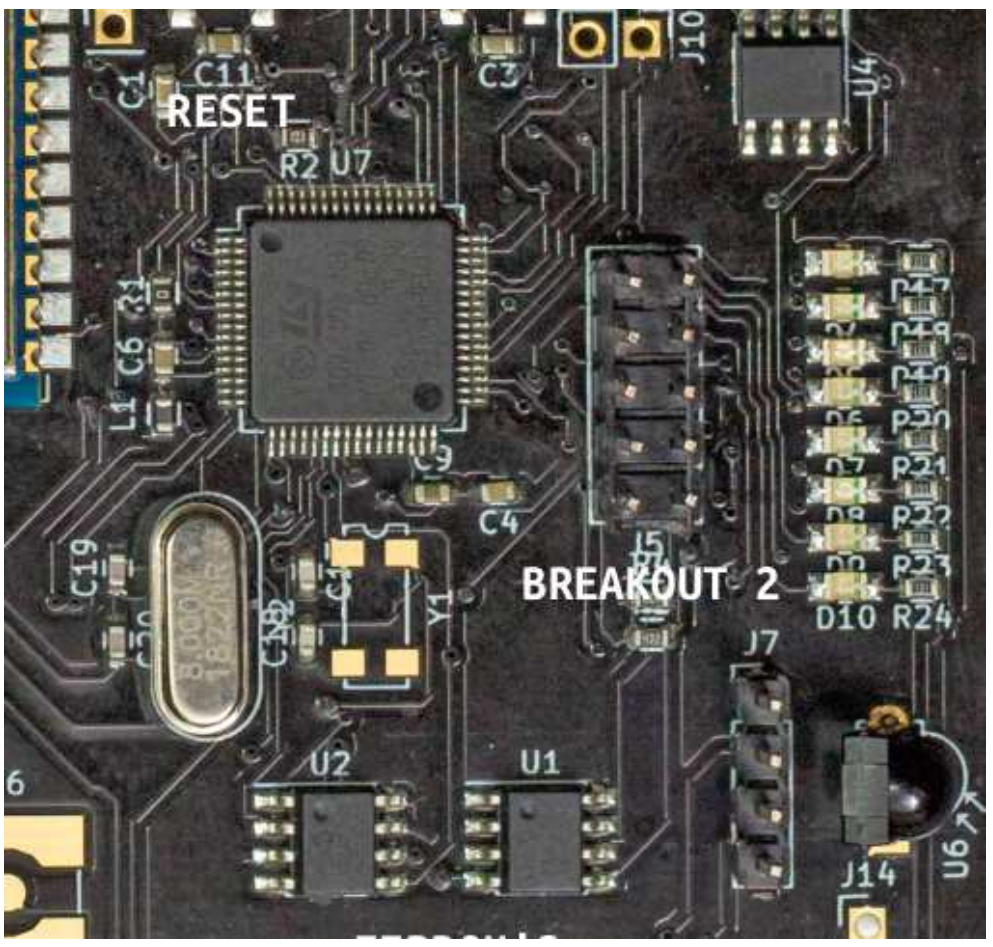
You can refer to our blog, "IoT Security-Part 13 (Introduction to Hardware Recon)", for a better understanding of the hardware recon.

Case 1: You do not have the actual hardware, but you know the FCCID number of the device. Go to the FCCID website, search for the FCCID number of the device. If correct, you will find all the internal images and detailed internal, external information about the device. You may get a hint for devices with JTAG/SWD interfaces on the hardware seeing the internal image. Yayy!! Once you know that you have the attack surface, you can get/purchase that device and perform further required steps to attack the device.

Case 2: You got access to the hardware. Once you have the physical hardware in your possession, the first step should be to perform reconnaissance. Inspect each test point and chips present on the printed circuit board (PCB) to look if you can get a JTAG/SWD interface.

As discussed above, for JTAG, we need to identify four signal pins (TCK, TDI, TDO, TMS) and Vcc, GND pins. For SWD, we need two signal pins (SWDIO, SWCLK) and Vcc, GND pins. Few ways in which you can identify these interface pins on the device are mentioned below :

1. Manual Identification - Identify the microcontroller used in the device, take out its datasheet, and identify the microcontroller's JTAG/SWD pins. For example, we will take the example of **EXPLIoT DIVA board**.



The microcontroller used in the diva board marked as U7 on the PCB is **STM32F411x**, we can search its datasheet for the pins having JTAG/SWD interface connection. The image below shows that section of the datasheet.

Table 8. STM32F411xC/xE pin definitions (continued)

Pin number					Pin name (function after reset)[1]	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	LQFP64	WLCSP49	LQFP100	UFBGA100						
10	14	F6	23	L2	PA0-WKUP	I/O	TC	(5)	TIM2_CH1/TIM2_ET, TIM5_CH1, USART2_CTS, EVENTOUT	ADC1_0, WKUP1
11	15	G7	24	M2	PA1	I/O	FT	-	TIM2_CH2, TIM5_CH2, SPI4_MOSI/I2S4_SD, USART2_RTS, EVENTOUT	ADC1_1
12	16	E5	25	K3	PA2	I/O	FT	-	TIM2_CH3, TIM5_CH3, TIM9_CH1, I2S2_CKIN, USART2_TX, EVENTOUT	ADC1_2
13	17	E4	26	L3	PA3	I/O	FT	-	TIM2_CH4, TIM5_CH4, TIM9_CH2, I2S2_MCK, USART2_RX, EVENTOUT	ADC1_3
-	18	-	27	-	VSS	S	-	-	-	-
-	-	-	-	E3	BYPASS_REG	S	-	-	-	-
-	19	-	28	-	VDD	I	FT	-	EVENTOUT	-
14	20	G6	29	M3	PA4	I/O	FT	-	SPI1_NSS/I2S1_WS, SPI3_NSS/I2S3_WS, USART2_CK, EVENTOUT	ADC1_4
15	21	F5	30	K4	PA5	I/O	FT	-	TIM2_CH1/TIM2_ET, SPI1_SCK/I2S1_CK,	ADC1_5

Depending on the IC package that is used in the device (here its LQFP64), we identify the JTAG port pins on the controller. Then, we set the multimeter in continuity mode as in the image below :

Then, we put one probe on the JTAG port pins on the controller and the other probe on the test points/ headers pins on the PCB suspected to be JTAG port pins. This test is repeated until pins are identified. The procedure for identifying SWD pins is the same. Manual identification is

undoubtedly going to be hectic, but no worries; we have automation tools with us to solve this problem :)

1. Automated Identification- Many available tools can help us scan and identify the JTAG/SWD pins on the hardware.

    - **EXPLIoT Bus Auditor** - It supports JTAG, SWD, I2C, UART. It has an adjustable target voltage. Its demo example in the case of JTAG and SWD interface can be understood from our blog, **"IoT Security-Part 14 (Introduction to and Identification of Hardware Debug Ports)"**.

    - **JTAGEnum** - It's the oprn source project based on Arduino. It does not support UART, and also it does not have the adjustable voltage. It is cheaper to use. Before connecting pins to Arduino, make sure to adjust the voltage levels. You can refer to the blog **here** to understand how to identify JTAG pins using JATGEnum.

    - **JTAGulator** - It supports JTAG, UART, SWD. It has an adjustable target voltage. You can refer to the blog **here** to understand how to identify JTAG pins using JTAGulator.

    ** NB** : *Before connecting any external device to the target board, make sure to adjust the voltage levels and make the Ground common for both the attacking/scanning and the target devices.*

Yayy!! Once the JTAG/SWD interface pins are identified on the hardware, it is time to break into the device.

Sniff the communication

Tools like Logic Analyzer can be used to sniff the communication between the device having the JTAG/SWD interface and the controller. The logic analyzer would show the TCK, TDI, TDO, TMS signal lines of the JTAG port or SWCLK , SWDIO lines of SWD port. Softwares like Saleae Logic Analyzer, PulseView have the feature to detect these signals that directly shows you the decoded data w.r.t the respective JTAG/SWD interface. Sniffing communication can help to decode the IDCODE and other sensitive information.

Interfacing

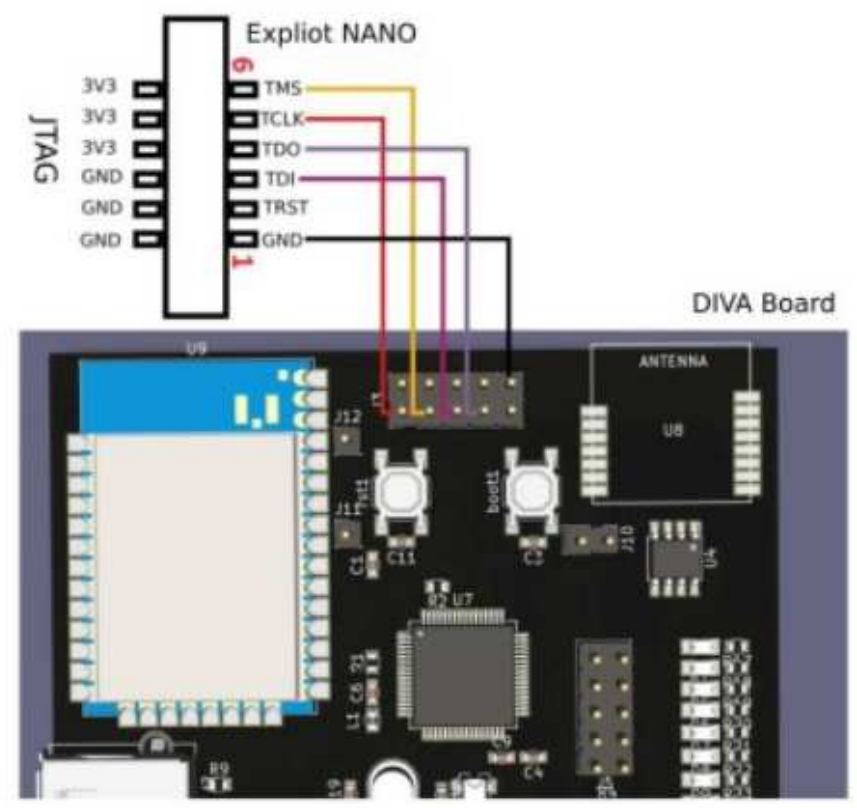
To communicate with the JTAG/SWD interface on the device we need a protocol adapter and a software that can communicate over JTAG/SWD via the adapter with the device. Keep note of an important point again, *" Before connecting any external device/ adapter to the target board, make sure to adjust the voltage levels and make the Ground common for both the attacking/scanning and the target devices."* Various adapters are available for connecting with JTAG/SWD interface. A few of them are mentioned below.

- **EXPLIoT Nano**
- **Bus Pirate**
- **Shikra**

For communicating with JTAG/SWD interface with the host machine via these protocol adapters, we need something that can take instructions from the user via the host machine and send the corresponding low-level instructions to the respective JTAG/SWD interface via the protocol adapter and vice-versa. **Openocd: Open on-Chip Debugger** is a debugging software

that provides on-chip programming and debugging support with a layered architecture of JTAG/SWD interface and TAP support. It supports Debug target (e.g., ARM, MIPS): single-stepping, breakpoints/watchpoints, etc. It has support for a variety of chips, interfaces, and targets. It also opens GDB and telnet server. You can perform GDB debugging on the hardware or communicate over telnet using openocd commands. To communicate with the respective chip and the interface, we need to provide the correct configuration files w.r.t the chip and the interface we are using. Below, we will show JTAG interfacing using EXPLIoT Nano on the target EXPLIoT DIVA board.

After identifying the JTAG pins by any of the scanning tools/ methods discussed above, connect the JTAG pins TCK, TMS, TDI, TDO with the corresponding JTAG pins on the adapter (here,EXPLIoT Nano as per its datasheet/user manual). The connection image is shown below.



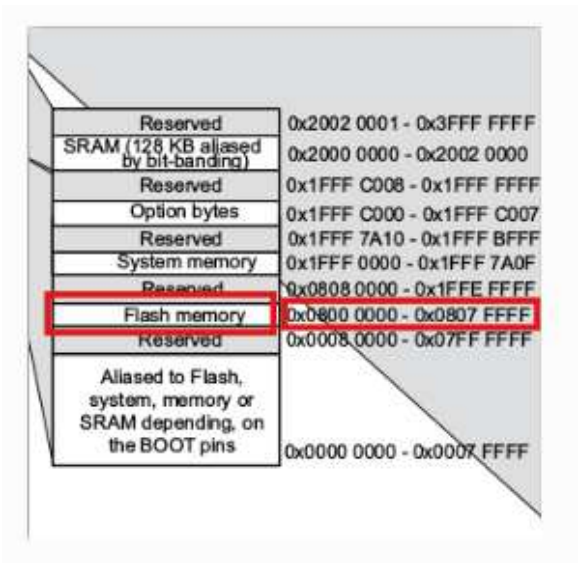
Now, to communicate with the JTAG interface, we need to launch the **Openocd**, `$ openocd -f expliot_nano_jtag.cfg`.

** NB** - *We pass configuration files to openocd to make it aware of the target microcontroller and the debug adapter used. For a different microcontroller, adapter configuration, you might need to choose the corresponding configuration files. More on openocd configuration files can be read **here*** The communication with the interface starts as in the image below.

```
GNU ARM Eclipse 64-bits Open On-Chip Debugger 0.10.0-00114-g8419536 (2017-04-1
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
swd
adapter speed: 2000 kHz
adapter_nsrst_delay: 100
none separate
cortex_m reset_config sysresetreq
Info : No device selected, using first device.
Info : J-Link V10 compiled Apr 28 2016 11:49:37
Info : Hardware version: 10.10
Info : VTarget = 3.219 V
Info : clock speed 2000 kHz
Info : SWD DPIDR 0x2ba01477
Info : stm32f4x.cpu: hardware has 6 breakpoints, 4 watchpoints
```

Now, in another terminal, we will open telnet and perform different operations over the JTAG interface. The link here describes in detail the commands for performing different operations including, memory read, write, dump, etc. So, for example to extract the firmware the command is , dump_image flash_dump.bin <address> <flash memory length>

We first need to identify the address and length of the flash memory of the chip that we are using. As discussed above, EXPLIoT DIVA board has **STM32F411x** microcontroller. We check its datasheet , we identified the flash memory address at **0x8000000** and length, **0x807FFFF – 0X8000000 = 7FFFF** as shown in image below.



And via the telnet we send the following commands, * halt * dump_image flash_dump.bin 0x8000000 0x7FFFF * resume

It successfully extracts the firmware from the device, as shown below.

```
>halt
target state: halted
target halted due to debug-request, current mode: Thread
xPSR:  0x61000000 pc:  0x080011f6 msp:  0x2001ffd0
>dump_image /tmp/flash_dump1.bin 0x8000000 0x7FFFF
dumped 524287 bytes in 4.230912s (101.804 KiB/s)
>resume
```

Thus, various openocd commands can be used to perform various operations, including manipulating the register value, patching the firmware, etc. A similar approach would be for the SWD interface, we just need to connect with the respective SWD pin, and corresponding SWD transport should be mentioned in the configuration file. Thus, once we are successful in interfacing and communicating with the JTAG/SWD ports, it opens the possibilities for various attack scenarios, as discussed above. In our future blogs, we will perform these attacks on some IoT devices, so stay tuned.

Conclusion

In this blog, we learned about the JTAG, SWD interface, the possibilities of attack scenarios, and the methods to attack. We hope you enjoyed and got some valuable information out of it. These tools and attack methods would give you a basic understanding of what to do when you find a JTAG/SWD interface on the hardware and play around with it to attack some devices.

References

[1] https://blog.senr.io/blog/jtag-explained

[2] https://embeddedbits.org/2020-02-20-extracting-firmware-from-devices-using-jtag/

[3] https://en.wikipedia.org/wiki/JTAG

[4] https://www.corelis.com/educationdownload/JTAG-Tutorial.pdf

[5] https://developer.arm.com/architectures/cpu-architecture/debug-visibility-and-trace/coresight-architecture/serial-wire-debug

[6] https://www.cnblogs.com/shangdawei/p/4748751.html

[7] https://research.kudelskisecurity.com/2019/05/16/swd-arms-alternative-to-jtag/

About Payatu

Payatu is a boutique security testing and services organization specialized in Products, Application, and Infrastructure security assessments and deep technical security training. We offer a full IoT ecosystem security assessment, including Hardware, Cloud, Web, and Mobile interface. If you are looking for security testing services then let's talk, share your requirements: https://payatu.com/#getstarted

Payatu is at the front line of IoT security research, with a great team, and in house tools like expliot.io. In the last 8+ years, Payatu has performed, security assessment of 100+ IoT product ecosystems and we understand the IoT ecosystem inside out.

Get in touch with us. Click on the get started button below.