IRON-HID: Create Your Own Bad USB

Seunghun Han
National Security Research Institute
hanseunghun@nsr.re.kr

Wook Shin
National Security Research Institute
wshin@nsr.re.kr

Junghwan Kang
National Security Research Institute
ultract@nsr.re.kr

Jun-Hyeok Park
National Security Research Institute
parkparkqw@nsr.re.kr

HyoungChun Kim
National Security Research Institute
khche@nsr.re.kr

Eungki Park
National Security Research Institute
ekpark@nsr.re.kr

Jae-Cheol Ryou
Chungnam National University
jcryou@cnu.ac.kr

Abstract

With the development of semiconductor processes and the emergence of open-source hardware, there has been a rise in low-cost, high-performance embedded hardware of wide variety. Consequently, the maker culture, in which users create the tools that they require and share them with others, is spreading. In addition, security researchers are creating custom security inspection tools by using these embedded hardware. We have analyzed some of the existing custom security inspection tools, and developed IRON-HID, a tool with superior and more accurate functions than the existing tools. IRON-HID is designed to be attached to the existing hardware, and is composed of a custom device, firmware that operates the hardware, a test agent program that are installed on the host, and a commander program that are installed on user's smartphones. IRON-HID offers keyboard input transmission and monitoring, test agent installation, command execution, screen capturing, and file transmission functions so that it can be used in various penetration tests. The security inspector can control functions of IRON-HID using a smartphone, and can utilize penetration tests in various environments. Further, the security inspector gains the ability to inspect security weak points in a smartphone, a Point of Sale (POS) system, and a PC by attaching IRON-HID with a keyboard, a portable charger, or a card reader.

1. Introduction

With the development of technology in the area of semiconductor processes, there has been a rise in low-cost, high-performance embedded hardware. Moreover, with the emergence of open source hardware, which enables sharing of hardware information without restriction, there are now many different types of embedded hardware. The recent embedded hardware, such as the Arduino [1] and Raspberry PI [15] has exclusive development tools and user guides so that people can use them with ease and convenience. Consequently, maker cultures, in which tools that one requires are created and shared, are becoming popular and are spreading through online and offline communities, such as Make Faire [12], Hack a Day [8], and Raspberry PI community.

This phenomenon has triggered the emergence of a new type of security inspection tools. Security researchers can now use low-cost, high-performance embedded hardware to create security inspection tools with ease. Lau et al. [11] and Dunning [5] attached a custom device, based on embedded hardware, onto a legacy device, and showed how security threats on the iPhone and POS systems can be inspected. The custom device based security inspection tool is small in size and its inspection software functions are operated using the hardware. Therefore, it offers the advantage of being portable, and an inspection can still be performed when software cannot be installed, or in restricted environments.

In this paper, we have analyzed the functions of the existing custom security inspection tools, and presented IRON-HID with superior and more accurate functions than the existing tools. IRON-HID is a framework comprising a custom device, firmware that operates this custom device, a test agent program that is installed on the host, and a commander program that is installed on user's smartphone. IRON-HID offers keyboard emulation functions that monitor and transmit keyboard entries, and a function that simulates mass-storage for installation on the test agent program host (PC, POS, and smartphone). It also offers the ability for the security inspector to control these functions using a smartphone. We used several small embedded hardware equipped with

Name	Environment	Power Consumption	I/O Pin	Description
Arduino	Open-source Hardware and Software	Low	- 22 Pin (Nano) - 60 Pin (Mega)	- Various sizes (Nano to Mega) and low performance. - Most successful embedded hardware with Raspberry PI. - Uses various processors from ATmega168 to SAM3X8E. - Uses Arduino Sketch for easy programming.
Glitch [7]	Open-source Hardware and Software	Low	- 10 Pin	- Small size and low performance. - Uses AT90USB1287 processor. - Uses Arduino Sketch for easy programming.
Teensy [14]	Open-source Software	Low	- 46 Pin	- Small size and low performance. - Uses AT90USB1286 processor. - Uses Arduino Sketch for easy programming.
BeagleBone Black [3]	Open-source Hardware and Software	High	- 72 Pin	- Large size and high performance. - Uses AM335x processor. - Can run Android and Linux.
USB Armory [10]	Open-source Hardware and Software	High	- 5 Pin	- Small size and high performance. - Uses i.MX53 processor. - Can run Android and Linux.
Raspberry PI	Open-source Software	High	- 40 Pin	- Large size and high performance. - Most successful embedded hardware with Arduino. - Uses BCM3847 processor. - Can run Android and Linux.

Table 1. Types of Embedded Hardware and their Characteristics

several I/O pins to create a custom device so that IRON-HID can be attached to the legacy device. The design was also made so that the functions of IRON-HID can be expanded through the commander program installed on the smartphone and the test agent installed on the host. IRON-HID, with an expandable design and structure, can be used for penetration tests in various environments. This paper is structured as follows. Section 2 explains the background of the study while Section 3 explains the design and creation of IRON-HID. Section 4 and Section 5 evaluate and discuss IRON-HID, and Section 6 concludes this paper.

2. Background

2.1 Hardware-based Security Inspection Tools

There are two ways to create a hardware-based security inspection tool. First is a method that modifies the firmware inside the device, and the second is a method that attaches the custom device onto the device. The device is composed of hardware and firmware. Firmware is software that drives the hardware. The manufacturer usually offers firmware updates to add functions to the device or improve its functions, or fix security weak points in the firmware. Security researchers can add security inspection functions by using firmware updates that are offered by the manufacturer or by overwriting the flash memory in the firmware with new firmware. When the security inspector connects these devices to the user PC for inspection, they can inspect the weak points of the user PC, check whether permissions can be raised higher [9], or check if the user's confidential information can be leaked [2].

The method of adjusting the firmware is less expensive than attaching a custom device, and is advantageous because there is no limit to the device's size. However, we must first assess the structure of the hardware via reverse engineering, and the inspection tool's functions are limited because it only offers functions that can be offered by the parts on the legacy device.

If an inspection tool is created using a custom device, we can overcome the aforementioned hardware function limitation. Moreover, if the inspection tool is designed so that custom device for inspection can be attached to legacy hardware, its range of movement increases because movement is possible through the inspection tool itself [5]. However, it is expensive to create and attach a custom device, and there may be devices that cannot be attached because of the size of the custom device.

2.2 Embedded Hardware for Custom Device Creation

Embedded hardware that is used to create custom devices is becoming smaller and more diverse, and can be easily purchased through online stores. There is a variety of embedded hardware that are being sold in the current market, from low-power, low-performance hardware that use 8-bit microprocessors all the way to high-power, high-performance hardware that use 32-bit microprocessors. These different types of embedded hardware are listed in Table 1.

Functions and performance vary in embedded hardware, and we must make decisions about creating custom device

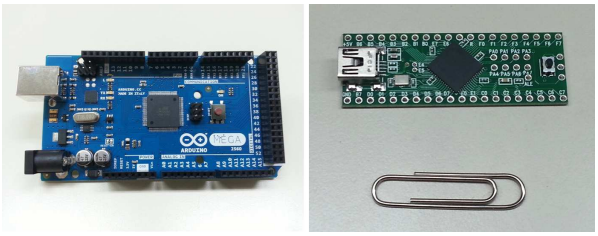

Figure 1. Arduino Mega (left) and Teensy (right)

after checking the power required for operation and whether all the required functions are included.

3. Design and Implementation

3.1 Requirements of IRON-HID

Hardware Requirements IRON-HID is a framework that is attached to the existing USB devices, expands the functions of legacy devices, and efficiently and automatically inspects their weak points. IRON-HID can be created by customizing embedded hardware, and the standards for selecting an appropriate embedded hardware include power consumption, I/O pin count, and USB communication support. Thus, IRON-HID's hardware requirements are as follows:

- Less Power Consumption

  - The maximum current that can be used for USB 2.0 is 500 mA.

  - Power consumption should be less since it is attached to a legacy device.

- Many I/O Pins

  - Having more I/O pins is advantageous in order to attach onto various types of devices (for example, keyboard, mouse, portable chargers, and card readers.)

- Internal USB Communication Functions

  - It is difficult for low-performance embedded hardware to process USBs that communicate at high-speed with software.

  - If USB communication functions are equipped at the hardware's level, communication efficiency increases and the firmware's code size decreases.

Comparing the above requirements with the embedded hardware in Table 1, Arduino and Teensy were selected. Arduino Mega has a relatively large flash memory with many I/O pins. Due to its large size, it is most appropriate for attaching on large devices with complex functions. Teensy has a relatively small flash memory, but due to its small size, it can be attached to devices of various sizes. The detailed specifications for Arduino Mega and Teensy are as follows.

- Arduino MEGA

  - Low power consumption, 60 I/O pin, 256KB flash memory, 4 inch x 2.1 inch.

  - Uses ATmega2560 and ATmega16U2 with USB communication functions.

  - Used when the device to be attached on is large, and when many I/O pins are required.

- Teensy

  - Low power consumption, 46 I/O pin, 128KB flash memory, 2 inch x 0.7 inch.

  - Uses AT90USB1286 with USB communication functions.

  - Can be used on devices of various sizes.

Functional Requirements IRON-HID is a framework that facilitates penetration tests by connecting to POS systems, PCs, and smartphones. The device for inspection may not offer network, keyboard, storage devices, and other surrounding devices, depending on the installation environment. Therefore, IRON-HID offers the following functions so that a penetration test can be conducted in various environments.

- Small form factor and communicates with a smartphone program.
- Emulates a CD-ROM, so that it can install a test agent program without a network connection.
- Hooks onto a user's keyboard event and sends keystrokes to the pen-tester.
- Performs screen captures of the target device.
- Receives input from a penetration tester via the smartphone program and sends it to the target machine.

3.2 Overall Architecture and Protocol

3.2.1 Architecture

IRON-HID is composed of four parts: a custom device with embedded hardware and communication modules, firmware that operates the custom device, a test agent program that are installed on the host, and a commander program for user's smartphone. Figure 2 is a schematic diagram of IRON-HID.

The custom device is composed of embedded hardware and wireless communication modules. It is attached onto the proxy device and IRON-HID firmware is installed on the custom device. When IRON-HID firmware is connected to hosts, such as a POS, a PC, or a smartphone, it becomes a key part that takes charge of USB device functions. IRON-HID firmware contains test agent programs, and when connected to a host, the keyboard and CD-ROM functions are activated and the test agent program is installed on the host. The commander program that is installed on the security inspector's smartphone is connected to IRON-HID's wireless communication module. The commander program sends keyboard events to IRON-HID firmware or sends a request to the test agent program to execute the command, and it receives the results of commands from the test agent program. A shell command is sent from the commander program pro-

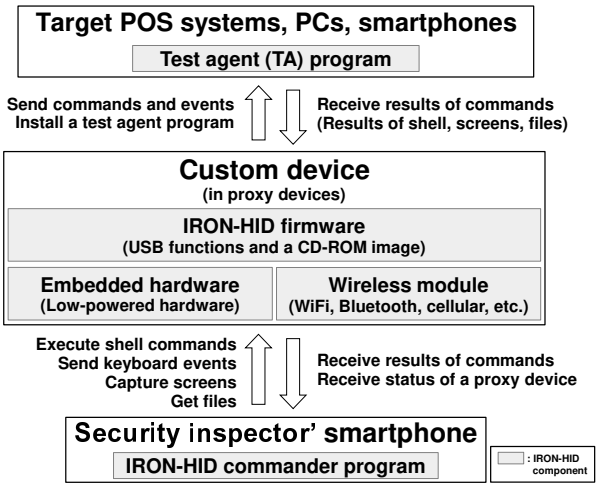

Figure 2. IRON-HID Architecture

cessed from the command-line interpreter, and a screen capture command and a host file transmission command are processed internally from the test agent program.

3.2.2 Communication Protocol

The test agent program, firmware, and commander program are connected to each other in IRON-HID, and simple communication protocols are defined in order to communicate. Table 2 shows the communication protocols that are defined by IRON-HID.

The firmware acts as a path that connects the test agent program and the commander program, and bypasses the delivered packets from both the sides as it is. The work that is processed by the firmware uses binary magic strings to send commands.

The test agent program and the commander program use the identifiers in Table 2 to compose data, and process the received data through each of the internal parsers. If new functions are required, it can be done by simply adding a protocol to the test agent program and the commander program.

3.3 Hardware Part

The hardware part directly connects with the POS, PC, or mobile device through a USB, and is composed of the embedded device that was selected in advance (Arduino Mega, Teensy), the wireless communication module, and the USB On-The-Go (OTG) cable. The wireless communication module can be selected from Bluetooth, WiFi, or cellular networks. We used the Bluetooth serial communication module (RN-42 silver) in this paper. The Bluetooth serial module is a module that uses a serial protocol to send and receive data, and is often used to expand the embedded hardware's serial communication functions to wireless communication. Bluetooth serial communication modules differ with each manufacturer, but the pin used for communication

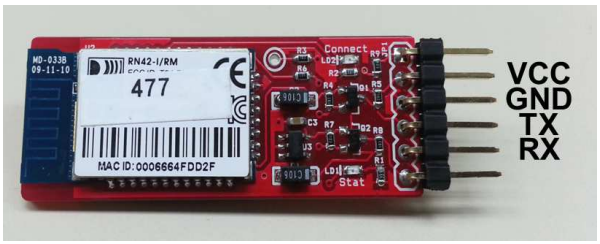

Figure 3. Pin Composition of the Bluetooth Serial Communication Module (RN-42 Silver)

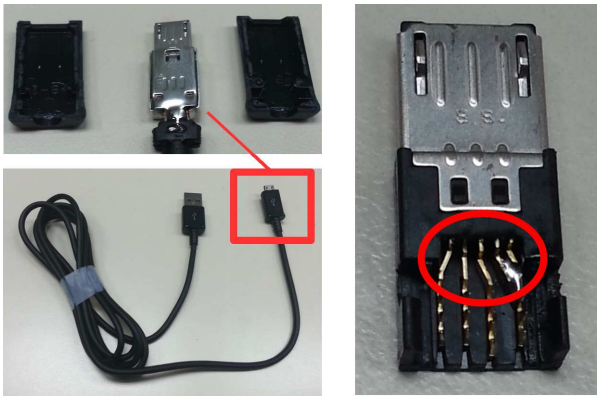

Figure 4. Smartphone Cable Disassembly Process (left) and OTG Cable Creation Method (right)

is identical. Figure 3 shows the pin composition of the Bluetooth serial communication module. The VCC is a pin that permits power, and the GND is a ground connection pin. The TX pin is for sending data to the embedded device, and the RX pin is for receiving data from the embedded device.

Arduino Mega and Teensy contains hardware modules for serial communication. In Arduino Mega, the TX pin is connected to the 18 pin and the RX pin is connected to the 19 pin for serial communication. In Teensy, the TX pin is connected to the D3 pin, and the TX pin is connected to the D2 pin. For making the custom device, the TX pin and RX pin in the embedded hardware are replaced with the RX pin and TX pin of the Bluetooth communication module. The USB OTG cable is a cable that activates USB host functions of the smartphone, and the user use the OTG cable to connect the keyboard, mouse, or storage device to the smartphone. Micro USBs that are used in smartphones are composed of 5 pins, and can easily create an OTG cable if the ID pins (4 pins) and GND pins (5 pins) are connected. Figure 4 shows the process of creating an OTG cable. OTG cables that are created as such can connect IRON-HID to smartphones to inspect the security weak points.

3.4 Firmware Part

Firmware that controls the custom device and acts as a USB device are composed of keyboard emulation functions,

Direction	Protocol Type	Description
Commander program -> Firmware	Magic String 1 + <Command>	- The commander program sends the Magic String 1 to the firmware, and switches the firmware's mode to the command transfer mode. - When switched to the command transfer mode, the firmware temporarily stores data that is received from the commander program in its buffer, and then sends it to the test agent program. - Before sending the command from the commander program to the test agent program, the firmware mode must be switched to the command transfer mode.
Commander program -> Firmware	Magic String 2 + <Keyboard Event>	- The commander program sends the Magic String 2 to the firmware, and switches the firmware's mode to the key event transfer mode. - When switched to the keyboard event transfer mode, the firmware changes the keyboard input event that is received from the commander program into an HID report and sends it to the host. - Before sending the key input event from the commander program to the test agent program, the firmware mode must be switched to the command transfer mode.
Commander program -> Firmware	Magic String 3	- The commander program sends the Magic String 3 to the firmware, and activates mass-storage (CD-ROM) functions of the firmware.
Commander program -> Test agent program	C;<Command>;	- The commander program sends a command to the test agent program, and requests execution from the command-line interpreter of the host.
Commander program -> Test agent program	G;<File Name>;	- The commander program requests file transmission from the test agent program.
Commander program -> Test agent program	S;;	- The commander program requests screen capture from the test agent program.
Test agent program -> Commander program	F;;<64byte Data>;	- The test agent program sends the processed results (command execution results, file data, and capture data) to the commander program.
Firmware -> Commander program	M;;<Keyboard Event>;	- The firmware logs the user's keyboard event and sends it to the commander program. - Sends if IRON-HID is attached to the keyboard.
Firmware -> Commander program	D;;<Debug Message>;	- The firmware sends a debug message to the commander program.

Table 2. Protocols of IRON-HID

mass-storage (CD-ROM) emulation functions, and vendor command communication functions. When the USB device is connected to the host, the USB descriptor that is stored with interface information and endpoint information is sent to the host, and makes known the functions provided by the device. The host interprets the received USB descriptor and creates a communication channel with the USB device. It sends and receives keyboard data, mouse data, mass-storage data, and vendor command data with the USB device through the independent channel. Table 3 shows the interface and the endpoint that are created by IRON-HID. In the endpoint and transfer type categories, the interrupt type is a method in which the host periodically checks if data exists from the device. Because it guarantees latency, it uses the same input device as the keyboard or mouse that sends data periodically. The control type is a method that is used to send and receive important and urgent data. It performs control functions related to USB connections or is used when sending or receiving random data that is defined from the vendor. The bulk type is used to transfer a large amount of data, and employed for storage devices as mass-storage.

Interface	Endpoint	Type	Description
0	1	Interrupt	- Used for keyboard data communication.
1	2	Control	- Used for vendor command data communication.
2	3	Bulk	- Used for mass-storage (CD-ROM) data communication. - Sends data from the host to the device.
2	4	Bulk	- Used for mass-storage (CD-ROM) data communication. - Sends data from the device to the host.

Table 3. Interface and Endpoint of IRON-HID

IRON-HID firmware is designed to be equipped with the ISO images for the installation of the test agent programs and two queues for internal keyboard transmission

and command transmission. The USB device may only send data when the host requests data. Therefore, when the commander program sends keyboard events and commands, the firmware stores the data in a buffer temporarily and waits for the request of the host. The ISO image inside the firmware is composed of the test agent program file and the autorun.inf file, and the host runs it automatically. It changes into a binary form and is built with the firmware source code. The ISO image inside the firmware is sent when bulk I/O is requested from the host, and is mounted as a CD-ROM in the host. Storage space in the custom device is hundreds of kilobytes, therefore, ISO image should be smaller than flash memory size of the custom device and includes only the necessary functions.

IRON-HID firmware is based on lightweight USB framework for AVRs (LUFA) [6], and the CD-ROM emulation function uses the USB CD emulation project [4]. LUFA provides an API that abstracts USB communication functions that are supported in the hardware as a USB framework for Atmel's microprocessors inside Arduino and Teensy. When LUFA is used, the handshaking process for the first connection of the USB can be simplified, and the USB data can be processed through the event-driven method, which makes it more efficient.

3.5 Test Agent Program and Smartphone Commander Program Part

The test agent program is installed in the POS and PC. It executes the commands sent by the firmware and returns the results. When the test agent program is executed, it lists the USB devices that are connected to the host, and searches IRON-HID firmware's Vendor ID (VID) and Product ID (PID). If a USB device with the same VID and PID is found, a vendor-command channel between the relevant device and the test agent program is created. The USB devices work passively and they wait a request from the host, therefore, the test agent program periodically checks if there are requests from the firmware and returns the execution results.

The commands that are executed from the test agent are divided into commands that are processed internally by the test agent program and commands that are processed through the command-line interpreter. Commands that are processed internally by the test agent program include the host's screen capture command and commands that sends the host's specific files. The test agent program uses the API provided from the OS for capturing screens and reading files, and the results are sent through the vendor command communication channel. Commands that are processed through the command-line interpreter uses the OS's API and transmits using the same method.

The test agent program was developed for Microsoft's Windows OS, but communication channel was created based on HIDAPI [16], which supports multiple platforms. Therefore, if only the parts related to the commands that are processed internally by the test agent program are ported, the test agent program can be operated on various OSes.

The commander program that is installed in the security inspector's smartphone creates a wireless communication channel with IRON-HID firmware and sends and receives the command and processed results. The commander program activates the wireless communication functions in the smartphone, and searches and connects wireless communication modules of IRON-HID. The commander program acts as the interface that connects the security inspector, IRON-HID firmware and the test agent program. The commander program is composed of the control tab, command execution tab and keyboard event tab. The control tab is in charge of connecting and ending IRON-HID. The command execution tab is in charge of the security inspector's command execution and display of results. The keyboard event tab is in charge of the security inspector's keyboard input transmission and user's keyboard event logs.

The commander program has been developed for Android smartphones. However, if data is sent and received by adhering to IRON-HID protocols, it can act as a commander program for the PC, laptop, and other mobile devices.

4. Evaluation

4.1 Applicability

IRON-HID can be attached onto various USB devices, and we attached it to portable chargers, card readers, and keyboards to see the applicable functionality of IRON-HID. Figure 5 shows IRON-HID attached onto each device.

We applied IRON-HID on 1 type of portable charger, 1 type of card reader, and 3 types of keyboard, and were successful in operating IRON-HID while maintaining the USB device's own functions.

4.2 Usability

IRON-HID can be used to inspect various security weak points. To examine IRON-HID's usability, we inspected the open weak points of smartphones, POS, and PCs, and successfully completed an inspection. IRON-HID offers various functions for inspecting weak points, and when these functions are combined, new weak points can be found and examined.

4.2.1 Verification of a Backup PIN Vulnerability

IRON-HID can be used to check whether the backup PIN vulnerability exists on lock screen of the smartphone or application. IRON-HID is activated by connecting it to the smartphone using an OTG cable, and when IRON-HID's keyboard input transmission functions are used, a well-known PIN can be sent to the smartphone. Figure 6 shows the verification of a PIN vulnerability using IRON-HID. If the smartphone receives PIN input unlimitedly or a weak PIN was being used on the lock screen or application, we could use IRON-HID to inspect this.

Figure 5. USB Device attached with IRON-HID - Portable Charger (top), Card Reader (center), Keyboard (bottom)

Figure 6. Verification of a Backup PIN Vulnerability on a Lock Screen

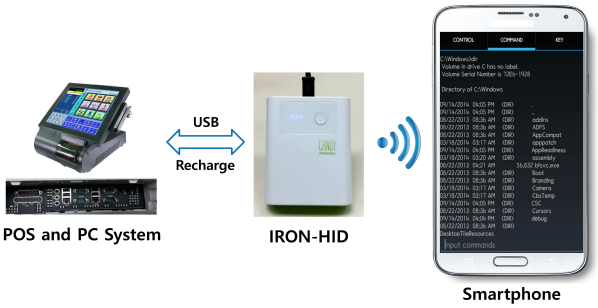

Figure 7. Verification of a Automatic Program Execution Vulnerability on a POS and a PC

4.2.2 Verification of an Automatic Program Execution Vulnerability

If a USB can be connected to the POS and PC, IRON-HID can be used to check if the program of CD-ROM is automatically executed. If automatic execution functions are activated, the test agent program is automatically installed when IRON-HID is connected. When the test agent program is installed, it notifies the installation completion through the commander program of smartphone. Figure 7 is a scene from checking automatic program execution in POS and PCs by using IRON-HID. We can check whether an automatic program execution option is turned on by connecting IRON-HID to POS and PCs.

5. Discussion

5.1 Limitations

Because IRON-HID currently includes CD-ROM images in the firmware, the size of the flash memory in the microcontroller becomes the maximum size of the test agent program. This limitation becomes a limiting factor for the expandability of the test agent program. When the host requests bulk I/O from IRON-HID, we will apply the method of sending CD-ROM images to the firmware from the commander program to overcome this limitation.

IRON-HID currently only supports functions for downloading files from the host. The penetration test is conducted in various environments, and the required files for inspection may not exist in the host. Therefore, the security inspector needs to upload files for inspection, and we plan to add file upload functions to IRON-HID.

5.2 Related Work

Among the existing works on custom security inspection tools, there were those that modified the device's firmware and those that added custom devices.

Studies that changed the firmware include studies of Balmas and Lior [2], Hudson et al. [9], and Nohl and Lell [13]. Balmas and Lior analyzed the firmware of the KVM device that divides the keyboard, mouse, and monitor, and changed it with a new firmware, showing that target PCs can be controlled. Hudson et al. analyzed Apple's thunderbolt adaptor firmware and modified it to show that Apple's PCs can be controlled. Nohl and Lell analyzed USB storage firmware and modified it, and added a new device without the user's permission and used it to show that target PCs can be controlled.

Studies that add custom devices include works of Dunning [5] and Lau et al. [11] Dunning used a glitch that is compatible with Arduino to create a tool that can be attached to various devices. Dunning inserted this device into a mouse, keyboard, and PC to show that PCs can be controlled. Lau developed a tool based on the BeagleBone Black board, which is an embedded hardware, and inserted it into a shared USB charging port that is installed inside the building. Lau used this tool to show that applications can be installed on an iPhone that is being charged in the USB charging port without the user's permission.

6. Conclusions

With the development of semiconductor processes there is a rise in low-cost, high-performance embedded hardware, and there is increased variety in embedded hardware with the emergence of open source hardware. And with the appearance of Arduino and Raspberry PI, which is targeted for common users, there is increased use of embedded hardware for hobbies and in the education field. As a result of these changes, maker cultures, in which users create the tools that they require and share them with others, are spreading, and security researchers are using custom devices based on embedded hardware for inspecting security threats.

We have analyzed existing studies that used custom devices, and developed IRON-HID, a tool with better and more accurate functions than the existing tools. IRON-HID is a framework composed of a custom device, firmware, a test agent program, and a commander program. IRON-HID offers features, such as keyboard input transmission and monitoring, test agent program installation, command execution, screen capture, and file transmission functions so that it can be used in various penetration tests. All the functions of IRON-HID can be controlled with the security inspector's smartphone.

7. Acknowledgements

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.R0236-15-1006, Open Source Software Promotion)

References

[1] Arduino. Arduino Board. `https://www.arduino.cc/`.

[2] Y. Balmas and L. Oppenheim. How to turn your KVM into a raging key-logging monster. *DEF CON*, 2015.

[3] BeagleBoard.org Foundation. BeagleBone Board. `https://beagleboard.org/black`.

[4] Curtis Reno and JRSmile. USB CD Emulation Project. `https://sourceforge.net/projects/usbcdemulation`.

[5] J. P. Dunning. Building Trojan Hardware at Home. *Black Hat Asia*, 2014.

[6] Four Walled Cubicle. Lightweight USB Framework for AVR (LUFA). `http://www.fourwalledcubicle.com/LUFA.php`.

[7] Glitch Operations LLC. Glitch Board. `http://theglitch.sourceforge.net`.

[8] Hack a Day. HackaDay. `http://hackaday.com/`.

[9] T. Hudson, C. Kallenberg, and X. Kovah. ThunderStrike2. *Black Hat USA*, 2015.

[10] Inverse Path. USB Armory. `https://inversepath.com/usbarmory`.

[11] B. Lau, Y. Jang, C. Song, and T. Wang. Mactans: Injecting malware into iOS devices via malicious chargers. *Black Hat USA*, 2013.

[12] Make Media, Inc. Make Faire. `http://makerfaire.com`.

[13] K. Nohl, S. Kribler, and J. Lell. BadUSB - On accessories that turn evil. *Black Hat USA*, 2014.

[14] PJRC. Teensy Board. `https://www.pjrc.com/teensy`.

[15] Raspberry PI Foundation. Raspberry PI Board. `https://www.raspberrypi.org`.

[16] Signal 11 Software LLC. HIDAPI Project. `https://github.com/signal11/hidapi`.