INTO THE CORE: IN-DEPTH EXPLORATION OF WINDOWS 10 IOT CORE

Paul Sabanal

IBM Security X-Force Advanced Research

sabanapm[at]ph[dot]ibm[dot]com

@polsab

Abstract

The Internet of Things is becoming a reality, and more and more devices are being introduced into the market every day. With this, the demand for technology that would ease device management, improve device security, and facilitate data analytics increases as well.

One such technology is Windows 10 IoT Core, Microsoft's operating system aimed at small footprint, low cost devices. It offers device servicing and manageability, enterprise grade security, and - combined with Microsoft's Azure platform - data analytics in the cloud. Given these features, Microsoft Windows 10 IoT Core will likely play a significant role in the future of IoT. As such, understanding how this operating system works on a deep level is becoming important. Methods and techniques that would aid in assessing its security are also becoming essential.

In this talk I will first discuss the internals of the OS, including the security features and mitigations that it shares with the desktop edition. I will then enumerate the attack surface of a device running Windows 10 IoT Core as well as its potential susceptibility to malware. I will also talk about methods to assess the security of devices running Windows 10 IoT Core such as static/dynamic reverse engineering and fuzzing. I will end the talk with some recommendations on how to secure a Windows 10 IoT Core device.

1 INTRODUCTION

1.1 Background

As the Internet of Things are becoming more and more prevalent, the need for technologies that would make managing and securing these devices better are becoming more important. One of the things that would facilitate this is the operating system running on the device. While there are currently operating systems that are more than capable of handling the requirements of an IoT device, its simply not enough. IoT is not just about the device, it's also about the service ecosystem that provides most of the value and functionality to the users. That's why operating systems developed from the ground up with IoT in mind are going to be valuable.

A couple of these IoT-focused operating systems were announced last year - Microsoft's Windows 10 IoT, and Google's Brillo. While at the time of writing these operating systems are not yet fully released, they look promising and are poised to become more significant in the future.

This also means they are potentially interesting targets for security minded folks, attackers and defenders alike.

For a security researcher, investigating a new technology is a significant part of the job. Understanding the inner workings of a complex technology such as a new operating system, especially in an exploding field like IoT, is very exciting. It also goes without saying that assessing the security of these devices will become an important part of a security researchers job in the future.

When assessing these devices, we need to think about their attack surface. Typically this would include but will not be limited to, network communications between the devices and its service ecosystem, network services running on the device, and the applications running on the device. We have to know if it communicates securely with the cloud. We have to know what services are running on the devices, or if they even need to be running at all. In the event that an attacker has gained access to a device, we also need to know the extent of damage they can do. To do all this, we need to be able to know the techniques and methods of analyzing a device. Only after understanding and doing all this can we make effective recommendations to the manufacturers and users alike on how to secure these devices.

1.2 Overview

There are three editions of Windows 10 IoT.

Edition	Description	Target Devices
Windows 10 IoT Enterprise	UWP apps, Win32 apps, desktop shell, x86, advanced lockdown	Kiosk, POS, ATM, Medical devices
Windows 10 IoT Mobile	UWP apps, multiuser support, lockdown features	Mobile POS, Industry hand held terminals
Windows 10 IoT Core	For low-cost, low-power devices. UWP apps only. ARM and x86	Smart home devices, IoT gateway, digital signage

Windows 10 IoT Core was first released by Microsoft last August 2015. The last public release was last December 2015. Since then several Windows Insider Preview builds were released with a lot of improvements, including support for the Raspberry Pi 3. There is little prior research on Windows 10 IoT Core security, which is understandable since it is still in its infancy. The only research that we are aware of was done by FFRI[1] and was presented at Code Blue 2015. A lot has changed since then, and this paper will reflect those changes.

Windows 10 IoT Core currently supports four suggested development boards:

Developer Board	Architecture	Details
Raspberry Pi 2	ARM	4xUSB 2.0, Ethernet
Raspberry Pi 3	ARM	4xUSB 2.0, Ethernet, Onboard Wi-fi and Bluetooth
Minnowboard Max	x86	1xUSB 2.0, 1xUSB 3.0, Ethernet
Dragonboard 410c	ARM	2xUSB 2.0, Onboard Wi-fi and Bluetooth

In addition to these suggested devices, Windows 10 IoT Core may also support other devices that is built on the same SoC as the above devices. Unless otherwise stated, the OS version documented here is Windows 10 IoT Core Insider Preview build 14393. The devices used are Raspberry Pis 2 and 3.

2 INTERNALS

2.1 Fast Flash Update Image Format

Windows 10 IoT Core images use the Fast Flash Update (FFU) image format. The FFU format is documented here [2]. Windows 10 IoT Core uses the V2 version of the format. You can retrieve its contents

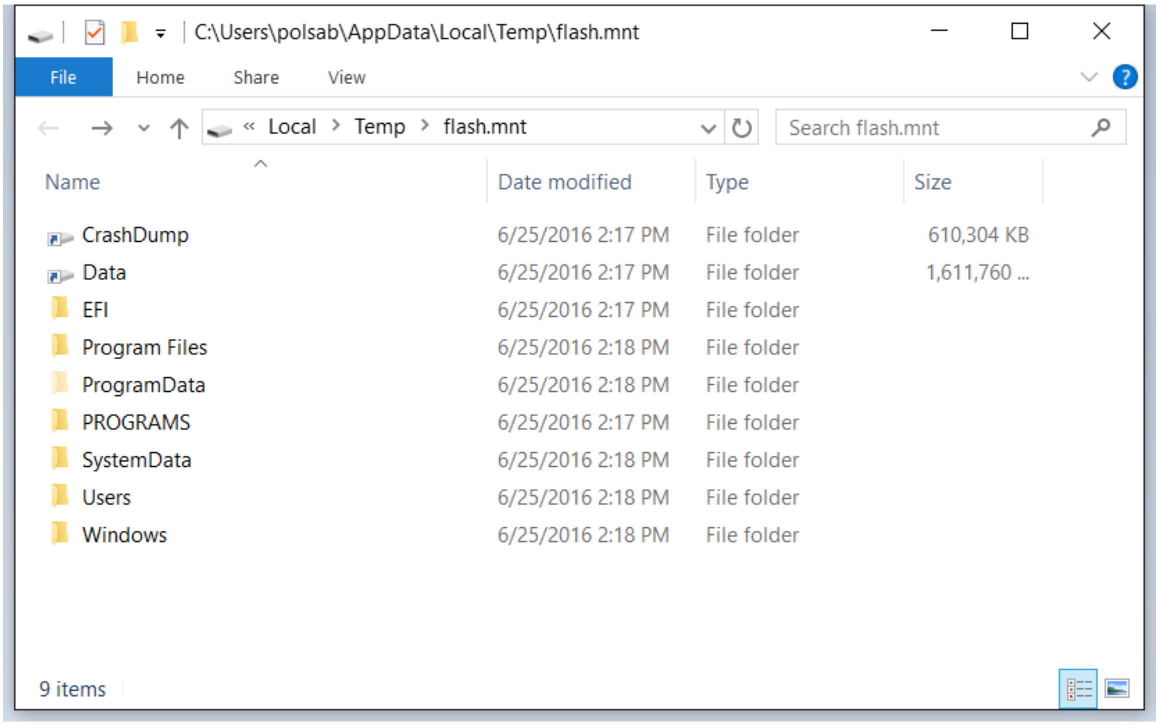
[1] "Threat Analysis on Windows 10 IoT Core and Recommended Security Measures" http://www.ffri.jp/assets/files/research/research_papers/Threat_Analysis_on_Win10_IoT_Core_en.pdf

[2] "FFU Image Format" https://msdn.microsoft.com/windows/hardware/commercialize/manufacture/mobile/ffu-image-format

by using the ImgMount[3] tool, which will convert the FFU file into a Virtual Hard Drive(VHD) image and mount it.

```
C:\>ImgMount.exe "c:\Program Files (x86)\Microsoft IoT\FFU\MinnowBoardMax\flash.ffu"

WP8 ROM Image Tools v.1.0.204
htc ROM Image Editor (⌐) 2007-2012 AnDim & XDA-Developers
ImgMount Tool v.1.0.15

(htcRIE) Mounting the image file : 'c:\Program Files (x86)\Microsoft IoT\FFU\MinnowBoar
dMax\flash.ffu'
Loading .FFU image ... ok
Creating virtual disk ... ok
Mounting MainOS partition as : '\\flash.mnt\' ... ok
(htcRIE) Successfully mounted an image file.
```

If the command was successful, the resulting VHD image will be mounted.


C:\Users\polsab\AppData\Local\Temp\flash.mnt
File
Home
Share
View
« Local > Temp > flash.mnt
Search flash.mnt
9 items

Name	Date modified	Type	Size
CrashDump	6/25/2016 2:17 PM	File folder	610,304 KB
Data	6/25/2016 2:17 PM	File folder	1,611,760 ...
EFI	6/25/2016 2:17 PM	File folder	
Program Files	6/25/2016 2:18 PM	File folder	
ProgramData	6/25/2016 2:18 PM	File folder	
PROGRAMS	6/25/2016 2:17 PM	File folder	
SystemData	6/25/2016 2:18 PM	File folder	
Users	6/25/2016 2:18 PM	File folder	
Windows	6/25/2016 2:18 PM	File folder	

Figure 1. Windows 10 IoT Core filesystem mounted by ImgMount

If you're not using Windows, there are some alternative tools to do this. ffu2img[4] and ffu2dd[5] will both convert the FFU image into a raw image that you can then mount using the *dd* tool. I haven't use these as much though so your mileage may vary.

[3] ImgMount Tool v.1.0.15 http://forum.xda-developers.com/showthread.php?t=2066903

[4] FFU2IMG https://github.com/t0x0/random

2.2 Partition Layout

A Windows 10 IoT Core image contains 4 partitions.

Partition	File System	Mount Point	Contents
EFI System Partition	FAT	C:\EFIESP	Boot manager, boot configurations, UEFI applications
Crash dump partition	FAT32	D:	Crash dump data
Main OS	NTFS	C:	OS, registry hives, OEM applications
Data partition	NTFS	U:	Applications, application data, user data

The EFI system partition contains the Windows Boot Manager (bootmgfw.efi) and the boot configuration database (BCD). The crash dump partition will contain crash dumps when a crashed occur that caused the device to restart. The Main OS partition contains all the components of the OS. The Data partition, which is linked to C:\Data, contains user data, installed apps, and app data.

2.3 Boot process

The typical boot process for Windows 10 IoT Core looks like this:

1. The device powers on and runs the SoC firmware bootloader.
2. The bootloader launches the UEFI environment and UEFI applications.
3. The UEFI environment launches the Boot Manager, which can be found in C:.\EFIESP\EFI\Microsoft\boot\bootmgfw.efi.
4. The Boot Manager launches the Windows Boot Loader, which can be found in C:\Windows\System32\Boot\winload.efi.
5. The Windows Boot Loader launches the main OS.

2.4 Apps

Windows 10 IoT Core supports different types of applications. First there are Universal Windows Platform (UWP) apps. UWP is the common app platform used in all Windows 10 editions. It allows the developer to theoretically develop an app that can run on any Windows 10 versions he may choose to support, with minimal changes in code. In Windows 10 IoT Core only one app can run in the foreground and is called the default app. You can install several apps on your device, but only one can be set as the default app, and it is launch when the system starts.

Background applications are apps that have no UI and runs on the background. They are launched at device startup and will continue to do so indefinitely, and will be respawned when they crash.

Windows 10 IoT Core also supports non-UWP apps such as console applications. In this case you can only use C++ and Win32 GUI APIs won't be available.

---

[5] FFU2DD **https://github.com/alxbse/ffu2dd**

Windows 10 IoT Core can also be configured to run on either headed mode or headless mode. In headed mode the default app displays a UI and is fully interactive. For devices that don't require any user interaction, headless mode is more appropriate. You can set your device to either mode by following the instructions here[6]

2.5 Security

In this section we will discuss the security features implemented in Windows 10 IoT Core. Windows 10 added new security features that offer significant improvements over earlier versions. Unfortunately, Windows 10 IoT Core does not support all of them.

2.5.1 What's not in Windows 10 IoT Core?

It may be possible that some of these features may be added in the future, but at the time of writing these are not supported:

- Security features that are built on top of Virtualization Based Security (VBS) such as Credential Guard, Device Guard, and Hypervisor Code Integrity (HVCI)
- Windows Defender
- Microsoft Passport

2.5.2 ASLR, DEP, and Control Flow Guard

Current IoT devices do not usually implement or enable modern exploit mitigations, and the fact the Windows 10 IoT Core implements these gives it an advantage over other operating systems. Executables included by default are compiled with ASLR and DEP enabled. Windows 10 IoT Core currently only supports 32-bit boards, so the ASLR implementation will inherently have lower entropy compared to the the 64-bit implementation. Control Flow Guard[7] is also enabled on the installed binaries, and can be be enabled by the developer on their app by setting the /guard:cf switch in the build configuration.

2.5.3 Trusted Platform Module (TPM)

The Trusted Platform Module[8] (TPM) is a secure crypto-processor that provides cryptographic key creation and storage. Other security features implemented in Windows 10 IoT Core such as Secure Boot and BitLocker will only work when TPM is installed.

Type	Description
Firmware TPM	TPM implemented in the SoC
Discrete TPM	Chip module that can be attached to a board
Software TPM	Software emulated TPM used in development

[6] "Headed and Headless mode" https://developer.microsoft.com/en-us/windows/iot/win10/headlessmode

[7] "Control Flow Guard" https://msdn.microsoft.com/en-us/library/windows/desktop/mt637065(v=vs.85).aspx

[8] "TPM on Windows IoT Core" https://developer.microsoft.com/en-us/windows/iot/win10/tpm

There are three types of TPMs. Firmware TPM is enabled in the Dragonboard 410c and Minnowboard Max(firmware version 0.8 or higher), but it's not available on Raspberry Pis. On devices that do not support firmware TPM, you can use Discrete TPMs, which can be attached on your chosen board. Software TPM only provides the software interface for your app and does not actually provide any security. It allows you to develop your application on a device without TPM (like the Raspberry Pi), but then deploy it later on a device with TPM without having to change your code.

The instructions to setup TPM on Windows 10 IoT Core devices can be found here[9]. You can also configure TPM on the Windows Device Portal's "TPM configuration" tab.

2.5.4 Secure Boot

Secure Boot is a feature that prevents a device from being tampered with during boot time. It stops the system for running binaries that are not digitally signed by the specified authority. It is designed to protect the system from rootkits, bootkits, and other low-level malware. Secure Boot on Windows 10 IoT Core requires TPM to be installed. Instructions to enable Secure Boot on Windows 10 IoT Core can be found here[10].

2.5.5 BitLocker

Windows 10 IoT Core implements a lightweight version of BitLocker[11]. Bitlocker allows automatic encryption of the user and system files on the OS drive. Bitlocker on Windows 10 IoT Core requires TPM to be installed. Instructions to enable BitLocker on Windows 10 IoT Core can be found here[12].

2.5.6 Windows Update

One of the most pressing problems in IoT security is the device firmware update problem. Vendors usually do not implement automatic update functionality and updates have to be done manually. Traditionally, device firmware update is not considered an simple process, often involving several steps such as downloading the firmware update from the vendor's website, connect to the device's web management interface, upload the firmware update, restart the device, etc. In some cases it may even involve pressing some button combination or some sort of unusual procedure just to put the device in firmware update mode. For most users, this is just too much effort and they will tend to put off applying updates. This leaves the device in a known insecure state until the update is applied.

Another issue is how to manage the updates of a lot of devices. A home of the future can potentially have dozens, maybe hundreds of IoT devices installed and monitoring which devices need updates and doing the update itself would be impossible to manage.

---

[9] "Setup TPM on Supported Platforms" **https://developer.microsoft.com/en-us/windows/iot/win10/SetupTPM.htm**

[10] "Enabling Secure Boot and BitLocker Device Encryption on Windows 10 IoT Core" **https://developer.microsoft.com/en-us/windows/iot/win10/sb_bl**

[11] "BitLocker Overview" **https://technet.microsoft.com/en-us/itpro/windows/keep-secure/bitlocker-overview**

[12] "Enabling Secure Boot and BitLocker Device Encryption on Windows 10 IoT Core" **https://developer.microsoft.com/en-us/windows/iot/win10/sb_bl**

Windows Update solves this problem for devices running Windows 10 IoT Core, as it was added in an Insider Preview Build earlier this year. Updates occur automatically, and it can't be disabled easily. If you want to disable, or want to have scheduled updates, you have to avail of the Pro edition of Windows 10 IoT Core. You can check for updates using the Windows Device Portal's "Windows Update" tab.

3 ATTACK SURFACE

In this section, we will enumerate the different potential entry points an attacker can leverage to gain access to a Windows 10 IoT Core device. Note that in this paper we will only talk about the attack surface exposed by the device or OS itself. The attack surface will be bigger if you factor in the device's connectivity with IoT platforms and services such as Microsoft Azure. However, it is outside of the scope of this paper, and would warrant a whole paper to itself.

3.1 Network Services

Using Nmap, we can see the open services on a freshly installed Windows 10 IoT Core device.

```
Starting Nmap 7.12 ( https://nmap.org ) at 2016-07-13 01:33 Malay Peninsula Standard Ti
me
Nmap scan report for 10.0.1.108
Host is up (0.020s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
135/tcp  open  msrpc
445/tcp  open  microsoft-ds
8080/tcp open  http-proxy
MAC Address: B8:27:EB:B5:A9:E0 (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 3.24 seconds
```

3.1.1 Windows Device Portal

Every edition of Windows 10 provides a web interface that you can use to manage and configure your device remotely called Windows Device Portal. It's enabled by default in Windows 10 IoT Core and runs upon device startup. You can access it by connecting to http://<device ip>:8080. The files for the Windows Device Portal can be found in C:\Windows\WebManagement\www on the device.

Here's a summary of the tabs currently available.

Utility	Function
Home	Device information, change device name/password, timezone settings
Apps	Install/uninstall of apps
App File Explorer	File explorer for installed apps locations
Processes	Running processes list, process memory usage, and process termination
Performance	Real time graphical display of CPU and I/O usage
Debugging	Starting VS remote debugger, downloading of live kernel and process dumps

ETW	Event tracing
Perf Tracing	Trace logging of CPU, disk, and memory usage
Devices	Device manager for peripherals attached to the device
Bluetooth	Bluetooth device search
Audio	Device speaker and microphone volume adjustments
Networking	WiFi configuration
Windows Update	Last update timestamp, check for updates
IoT Onboarding	Internet Connection Sharing settings, SoftAP settings, AllJoyn onboarding settings
TPM Configuration	TPM installation, configuration, and provisioning
Remote	Enable Windows IoT Remote Server

Let's talk about some of the more interesting ones. The Apps tab allows you to install/uninstall an app on the device. It also shows a list of the currently installed apps and their status. You can also use this tab to set an app as the default app.

The Processes tab shows a list of the running processes on the device. It also shows the process owner, session id, CPU usage, and memory usage. You can also terminate a process from this tab. There is also a box where you can enter a command and have it run on the device.

The Debugging tab contains operations related to debugging and crash dumping. Here you can click a button to start the Visual Studio Remote Debugger when you want to debug your app running on the device from Visual Studio. There are also buttons that allow you to download live kernel and processes dumps. We look more into crash dumps in the Dynamic Analysis section.

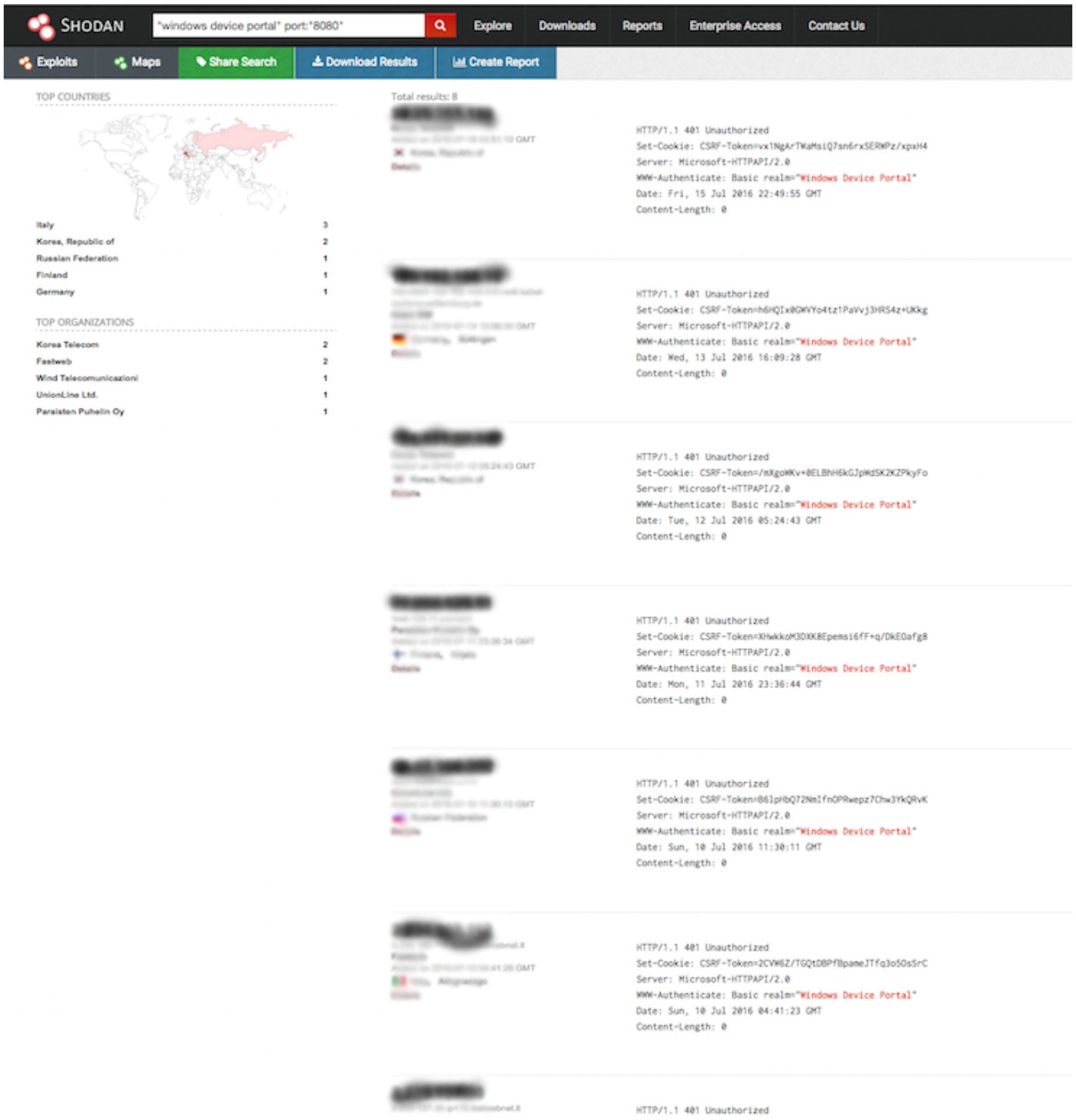
The Windows Update tab shows the latest update information and a button to check if an update for the device's OS is available.

In the TPM Configuration tab you can select which type of TPM you want to enable. Depending on the device, you can select from firmware TPMs, various discrete TPMs, and software TPM.

The Remote tab allows you to enable the Windows IoT Remote Server. We will discuss more about this feature below.

While the Windows Device Portal is useful for device management, it can be a security liability if not configured properly. In Windows 10 IoT Core, the default Administrator password is hardcoded on the device. You can login to the Windows Device Portal using the default Administrator credentials (User name: Administrator, Password: p@ssw0rd). If you did not bother to change the default password, your device is susceptible to unauthorized access. For example, an attacker can use Shodan[13] to search for devices running an HTTP server on port 8080 that returns a banner containing the string "Windows Device Portal". That Shodan search will yield a result similar to this:

[13] Shodan **https://www.shodan.io/**



TOP COUNTRIES

Italy	3
Korea, Republic of	2
Russian Federation	1
Finland	1
Germany	1

TOP ORGANIZATIONS

Korea Telecom	2
Fastweb	2
Wind Telecomunicazioni	1
UnionLine Ltd.	1
Paraisten Puhelin Oy	1

Total results: 8

[redacted]

```
HTTP/1.1 401 Unauthorized
Set-Cookie: CSRF-Token=vx1NgArTWaMsiQ7sn6rxSERWPz/xpxH4
Server: Microsoft-HTTPAPI/2.0
WWW-Authenticate: Basic realm="Windows Device Portal"
Date: Fri, 15 Jul 2016 22:49:55 GMT
Content-Length: 0
```

[redacted]

```
HTTP/1.1 401 Unauthorized
Set-Cookie: CSRF-Token=h6HQIx0GWVYo4tz1PaVvj3HRS4z+UKkg
Server: Microsoft-HTTPAPI/2.0
WWW-Authenticate: Basic realm="Windows Device Portal"
Date: Wed, 13 Jul 2016 16:09:28 GMT
Content-Length: 0
```

[redacted]

```
HTTP/1.1 401 Unauthorized
Set-Cookie: CSRF-Token=/mXgoWKv+0ELBhH6kGJpWdSK2KZPkyFo
Server: Microsoft-HTTPAPI/2.0
WWW-Authenticate: Basic realm="Windows Device Portal"
Date: Tue, 12 Jul 2016 05:24:43 GMT
Content-Length: 0
```

[redacted]

```
HTTP/1.1 401 Unauthorized
Set-Cookie: CSRF-Token=XHwkkoM3DXK8Epemsi6fF+q/DkEOafg8
Server: Microsoft-HTTPAPI/2.0
WWW-Authenticate: Basic realm="Windows Device Portal"
Date: Mon, 11 Jul 2016 23:36:44 GMT
Content-Length: 0
```

[redacted]

```
HTTP/1.1 401 Unauthorized
Set-Cookie: CSRF-Token=B61pHbQ72NmIfnOPRwepz7Chw3YkQRvK
Server: Microsoft-HTTPAPI/2.0
WWW-Authenticate: Basic realm="Windows Device Portal"
Date: Sun, 10 Jul 2016 11:30:11 GMT
Content-Length: 0
```

[redacted]

```
HTTP/1.1 401 Unauthorized
Set-Cookie: CSRF-Token=2CVW6Z/TGQtDBPfBpameJTfq3o5OsSrC
Server: Microsoft-HTTPAPI/2.0
WWW-Authenticate: Basic realm="Windows Device Portal"
Date: Sun, 10 Jul 2016 04:41:23 GMT
Content-Length: 0
```

[redacted]

```
HTTP/1.1 401 Unauthorized
```

Figure 2. Shodan results when searching for the Windows Device Portal banner

Now the attacker can connect to the device and attempt to login using the default Administrator credentials.

The Windows Device Portal also uses Basic Authentication by default, so anyone who is sniffing on the network can easily steal the credentials. We can fix by using HTTPS for the Windows Device Portal instead of HTTP. To do so, connect to the device through remote PowerShell or SSH and run the following commands:

```
# Enable HTTPS
Reg add HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\IoT\webmanagement
/v UseHttps /t REG_DWORD /d 1 /f
# Set HTTPS port
Reg add HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\IoT\webmanagement
```

```
/v HttpsPort /t REG_DWORD /d <PORT> /f
# Restart service
net stop webmanagement & net start webmanagement
```

The functionality of the Windows Device Portal is built on a set of REST APIs that you can use to control and configure your devices programmatically. As far as we know, there are currently no tools available to control or configure multiple devices at once, so this API is especially useful when writing your own tools to do so. There's some documentations available here[14], but a more updated version can be found by going to <device ip>:8080/restdocumentation.htm, or better yet by reading the JavaScript source code in C:\Windows\WebManagement\www\iot\js folder on the device.

3.1.2 SSH

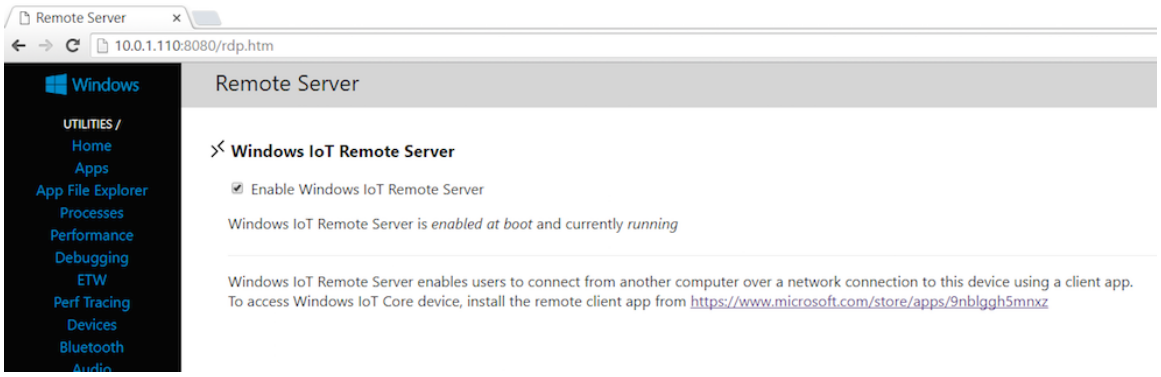
Windows 10 IoT Core allows remote administration and configuration through SSH, and it is enabled by default. SSH login also uses the default Administrator credentials, so if the user neglected to change this, an attacker can easily gain access. It can also be susceptible to password guessing and brute-force attacks.

3.1.3 Windows File Sharing

Windows File Sharing starts at boot time, and you only need the IP address of the device and user credentials to access it. It is also susceptible to the aforementioned default login credentials attack.

3.1.4 Windows IoT Remote Server

Windows IoT Remote Server is a feature that allows the UI of the UWP application running on the device to be viewed remotely through a client application running on a tablet mobile phone, or PC. You can enable Windows IoT Remote Server by checking the enable box in the Remote tab of the Windows Device Portal. Once enabled, the file NanoRDPServer.exe will be executed on the device and will start listening on port 8000 for incoming connections.



Windows IoT Remote Server

☑ Enable Windows IoT Remote Server

Windows IoT Remote Server is *enabled at boot* and currently *running*

Windows IoT Remote Server enables users to connect from another computer over a network connection to this device using a client app. To access Windows IoT Core device, install the remote client app from https://www.microsoft.com/store/apps/9nblggh5mnxz

Figure 3. Remote Server tab in the Windows Device Portal

[14] "Device Portal core API reference" https://msdn.microsoft.com/en-us/windows/uwp/debug-test-perf/device-portal-api-core

To use this remote display feature, you need to install the Windows IoT Remote Client app that can be downloaded from the Windows Store. However, this feature does not use any authentication, so anyone who knows the IP address of the device can connect to it using the remote client and remotely control your device.

3.2 Device Drivers Vulnerabilities

Device driver vulnerabilities are another potential attack vector on Windows 10 IoT Core devices. IoT devices obviously need connectivity with other devices in order to be useful. To facilitate this devices would need to use built-in or external peripherals, and these peripherals require device drivers to operate. These device drivers may contain vulnerabilities that could give an attacker remote access to the device if successfully exploited.

Drivers for wireless connectivity, such as for Wifi, Bluetooth, Zwave, Zigbee etc, are viable targets. One advantage of targeting drivers is that successfully exploiting them will often result in kernel level privilege.

3.3 Malware Susceptibility

Malware threats against Windows 10 IoT Core devices is indeed possible. As we have shown above, login credentials that are hardcoded and are not changed after install makes your devices susceptible to attacks. This is how current IoT malware typically infect an IoT device and we think that it will still be one of the most common infection method used by malware in the future.

Another possible infection method is the exploitation of vulnerabilities on the network services running on the device. This may not be as common as the login credentials method but there are malware that does this against embedded Linux devices.

Another possible way that a Windows 10 IoT Core device may be compromised by a malware is through lateral infection coming from an already infected machine. There are several ways in which a malware can gain access to a Windows 10 IoT Core device on the same network.

One scenario is for the malware to sniff on the network and look for traffic to the Windows Device Portal. Since by default it uses Basic HTTP Authentication, the credentials

For example, a malware has infected a machine that was used to login to a PowerShell session on a Windows 10 IoT Core device. In this case the attacker doesn't need to know the login credentials for the device beforehand. Using a tool like mimikatz[15] will yield the following results:

```
C:\>mimikatz.exe

  .#####.   mimikatz 2.1 (x64) built on Jul 11 2016 00:32:57
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                  with 20 modules * * */
```

[15] "Mimikatz: A little tool to play with Windows security" https://github.com/gentilkiwi/mimikatz

```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::ssp

Authentication Id : 0 ; 247557 (00000000:0003c705)
Session           : Interactive from 1
User Name         : polsab
Domain            : DESKTOP-39HUL88
Logon Server      : (null)
Logon Time        : 7/20/2016 6:15:59 PM
SID               : S-1-5-21-4294890806-594742593-2658599142-1001
        ssp :
         [00000000]
         * Username : Administrator
         * Domain   : 10.0.1.108
         * Password : diwata
```

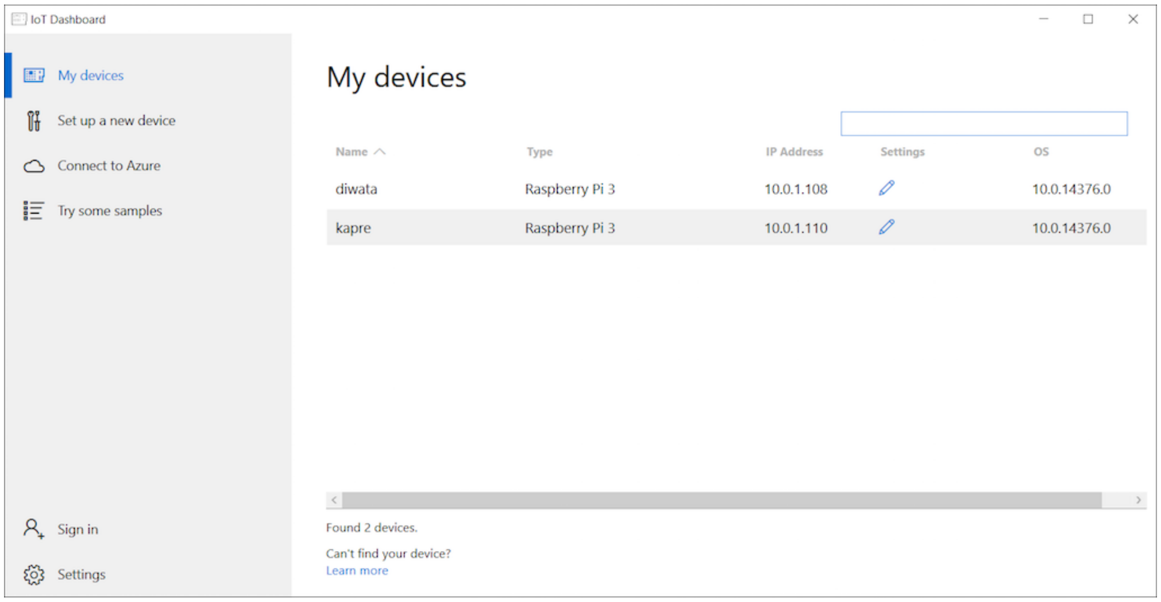
In this instance, we've logged in to the Administrator account on the device with IP address 10.0.1.108 using the password "diwata", and mimikatz can get this info from the infected machine's RAM.

4 Hacking Windows 10 IoT Core

In this section, we will discuss the various techniques that we can use when trying to assess the security of a Windows 10 IoT Core device.

4.1 Passive Device Discovery

Windows 10 IoT Core devices advertise their presence in the network by sending out multicast UDP packets. This is how the IoT Dashboard is able to list the running devices in the local network.



My devices

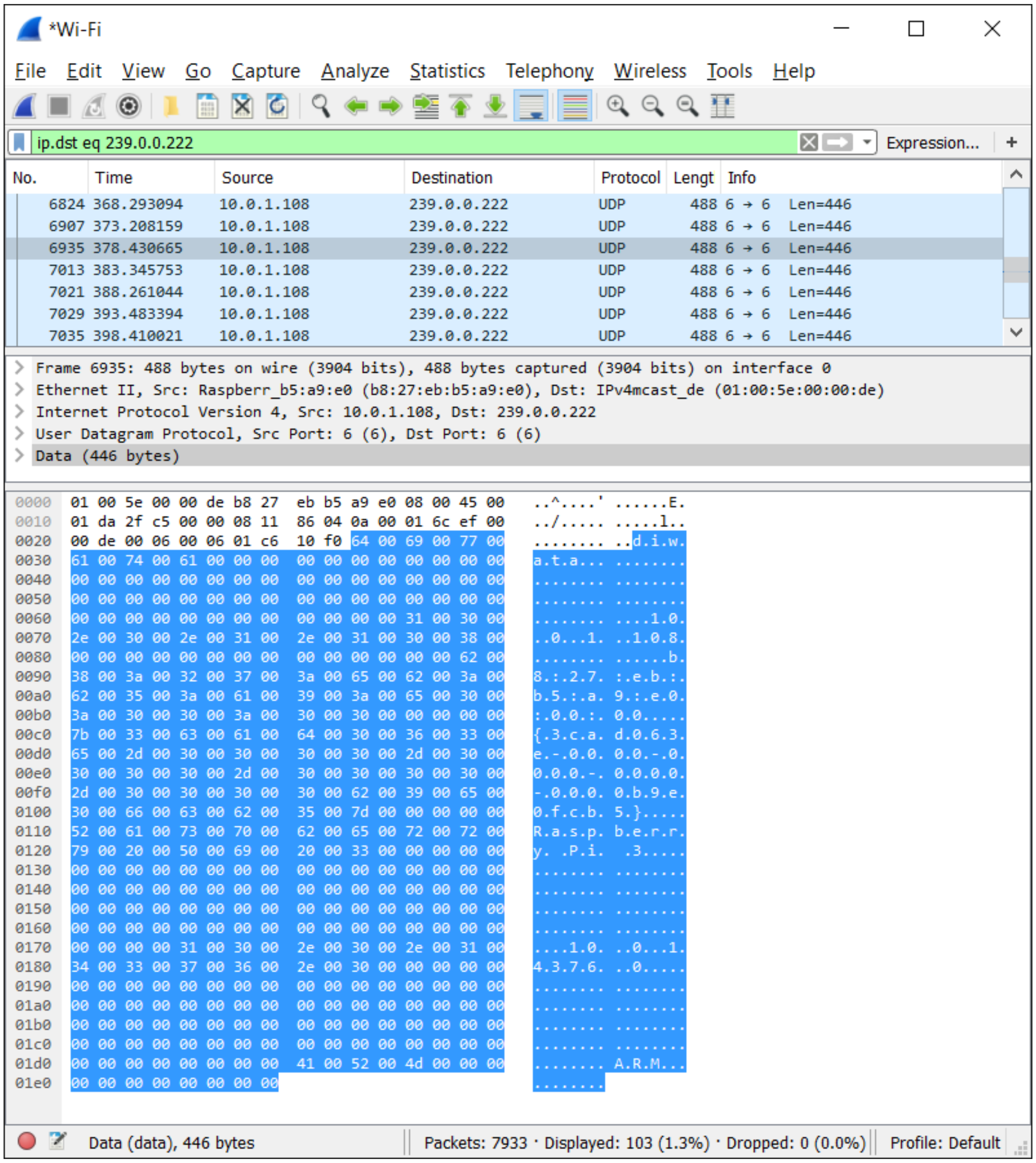
Name	Type	IP Address	Settings	OS
diwata	Raspberry Pi 3	10.0.1.108		10.0.14376.0
kapre	Raspberry Pi 3	10.0.1.110		10.0.14376.0

Found 2 devices.

Can't find your device?
Learn more

Figure 4. IoT Dashboard displaying the discovered devices

We can also do this by listening for multicast datagrams sent by the devices and parsing the data payload. The datagrams are sent to the multicast group 239.0.0.22 and multicast port 6 and contains the device name, IP address, OS version, MAC address, BIOS serial, device type, and device architecture.


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

```
ip.dst eq 239.0.0.222
```

```
No.   Time        Source      Destination  Protocol Lengt Info
 6824 368.293094  10.0.1.108  239.0.0.222  UDP       488 6 → 6  Len=446
 6907 373.208159  10.0.1.108  239.0.0.222  UDP       488 6 → 6  Len=446
 6935 378.430665  10.0.1.108  239.0.0.222  UDP       488 6 → 6  Len=446
 7013 383.345753  10.0.1.108  239.0.0.222  UDP       488 6 → 6  Len=446
 7021 388.261044  10.0.1.108  239.0.0.222  UDP       488 6 → 6  Len=446
 7029 393.483394  10.0.1.108  239.0.0.222  UDP       488 6 → 6  Len=446
 7035 398.410021  10.0.1.108  239.0.0.222  UDP       488 6 → 6  Len=446
```

```
> Frame 6935: 488 bytes on wire (3904 bits), 488 bytes captured (3904 bits) on interface 0
> Ethernet II, Src: Raspberr_b5:a9:e0 (b8:27:eb:b5:a9:e0), Dst: IPv4mcast_de (01:00:5e:00:00:de)
> Internet Protocol Version 4, Src: 10.0.1.108, Dst: 239.0.0.222
> User Datagram Protocol, Src Port: 6 (6), Dst Port: 6 (6)
> Data (446 bytes)
```

```
0000  01 00 5e 00 00 de b8 27  eb b5 a9 e0 08 00 45 00   ..^....' ......E.
0010  01 da 2f c5 00 00 08 11  86 04 0a 00 01 6c ef 00   ../..... .....l..
0020  00 de 00 06 00 06 01 c6  10 f0 64 00 69 00 77 00   ........ ..d.i.w.
0030  61 00 74 00 61 00 00 00  00 00 00 00 00 00 00 00   a.t.a... ........
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 31 00 30 00   ........ ....1.0.
0070  2e 00 30 00 2e 00 31 00  2e 00 31 00 30 00 38 00   ..0...1. ..1.0.8.
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 62 00   ........ ......b.
0090  38 00 3a 00 32 00 37 00  3a 00 65 00 62 00 3a 00   8.:.2.7. :.e.b.:.
00a0  62 00 35 00 3a 00 61 00  39 00 3a 00 65 00 30 00   b.5.:.a. 9.:.e.0.
00b0  3a 00 30 00 30 00 3a 00  30 00 30 00 00 00 00 00   :.0.0.:. 0.0.....
00c0  7b 00 33 00 63 00 61 00  64 00 30 00 36 00 33 00   {.3.c.a. d.0.6.3.
00d0  65 00 2d 00 30 00 30 00  30 00 30 00 2d 00 30 00   e.-.0.0. 0.0.-.0.
00e0  30 00 30 00 30 00 2d 00  30 00 30 00 30 00 30 00   0.0.0.-. 0.0.0.0.
00f0  2d 00 30 00 30 00 30 00  30 00 62 00 39 00 65 00   -.0.0.0. 0.b.9.e.
0100  30 00 66 00 63 00 62 00  35 00 7d 00 00 00 00 00   0.f.c.b. 5.}.....
0110  52 00 61 00 73 00 70 00  62 00 65 00 72 00 72 00   R.a.s.p. b.e.r.r.
0120  79 00 20 00 50 00 69 00  20 00 33 00 00 00 00 00   y. .P.i.  .3.....
0130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0170  00 00 00 00 31 00 30 00  2e 00 30 00 2e 00 31 00   ....1.0. ..0...1.
0180  34 00 33 00 37 00 36 00  2e 00 30 00 00 00 00 00   4.3.7.6. ..0.....
0190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
01a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
01b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
01c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
01d0  00 00 00 00 00 00 00 00  41 00 52 00 4d 00 00 00   ........ A.R.M...
01e0  00 00 00 00 00 00 00 00                            ........
```

```
Data (data), 446 bytes | Packets: 7933 · Displayed: 103 (1.3%) · Dropped: 0 (0.0%) | Profile: Default
```

Figure 5. Multicast datagrams

The fixed length data contains the device information. All strings are in Unicode.

Offset	Description
0	Device name
0x42	IP address
0x64	MAC address
0x96	BIOS serial number
0xe6	Device Type

0x14a	OS version
0x1ae	Device architecture

4.2 PowerShell

One of most useful built-in features in Windows 10 IoT Core is remote device administration and configuration using PowerShell. However, PowerShell is not just useful for system administration. It is also a powerful tool to use in security assessments. There are a lot of existing PowerShell modules - both built-in and from third-party developers - that could assist in reversing and penetration testing. Not all of them will work in Windows 10 IoT Core, but some of those that we have used are CimSweep[16] for remotely gathering device information, AutoRuns[17] to list autorun entries.

Here's an example using CimSweep to list Autostart entries:

```
PS C:\WINDOWS\system32> $CimSessionPi2 = New-CimSession -ComputerName 10.0.1.110 -Crede
ntial Administrator
PS C:\WINDOWS\system32> Get-CSRegistryAutoStart -CimSession $CimSessionPi2

Path           : HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
AutoRunEntry   : Shell
ImagePath      : IotShell.exe
Category       : Logon
PSComputerName : 10.0.1.110

Path           : HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
AutoRunEntry   : Userinit
ImagePath      : userinit.exe
Category       : Logon
PSComputerName : 10.0.1.110

Path           : HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
AutoRunEntry   : VMApplet
ImagePath      : SystemPropertiesPerformance.exe /pagefile
Category       : Logon
PSComputerName : 10.0.1.110

Path           : HKLM\SYSTEM\CurrentControlSet\Control\Session Manager
AutoRunEntry   : BootExecute
ImagePath      : autocheck autochk *
Category       : BootExecute
PSComputerName : 10.0.1.110
<snip>
```

Alternatively, you can also use AutoRuns which can show you more autoruns entries. However, you can only run it on-device.

---

[16] CimSweep https://github.com/PowerShellMafia/CimSweep

[17] AutoRuns Powershell Module https://github.com/p0w3rsh3ll/AutoRuns

To enable remote PowerShell sessions, follow the steps outlined here[18]

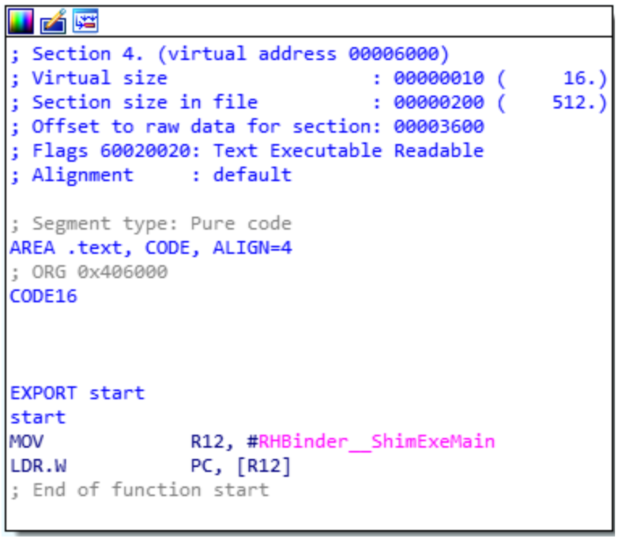
4.3 Static analysis

There is going to be little difference in reversing a console application compiled for Windows 10 IoT Core versus one compiled for the desktop, but there are some things to take note when reversing Windows Apps aka Universal Windwos Platform (UWP)[19] apps. Installed Windows apps can be found in the Data partition (U:\, also linked with C:\Data), specifically in the U:\Programs\WindowsApps folder. App installation folders will contain at least the following:

Filename	Description
<app_name>.exe	App startup stub
<app_name>.dll	App code
AppManifest.xml	UWP app package manifest
AppBlockMap.xml	Cryptographic block hashes for files in package
AppxSignature.p7x	App package digital signature file

In addition to the above files, other DLLs and XBF(binary XAML) files used by the application may be found in the app folder. There's also an assets folder that contains resources like images and fonts that the app uses. The <app_name>.exe file is simply a stub that calls the main exported function in the file <app_name>.dll. This DLL contains the application, .NET Framework, and third-party library codes.

Here's how <app_name>.exe looks like:



```
; Section 4. (virtual address 00006000)
; Virtual size                  : 00000010 (     16.)
; Section size in file          : 00000200 (    512.)
; Offset to raw data for section: 00003600
; Flags 60020020: Text Executable Readable
; Alignment     : default

; Segment type: Pure code
AREA .text, CODE, ALIGN=4
; ORG 0x406000
CODE16


EXPORT start
start
MOV             R12, #RHBinder__ShimExeMain
LDR.W           PC, [R12]
; End of function start
```

Figure 6. App startup stub

[18] "Using PowerShell to connect and configure a device running Windows 10 IoT Core" https://developer.microsoft.com/en-us/windows/iot/win10/samples/powershell

[19] "Guide to Universal Windows Platform (UWP) apps" https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide

All UWP binaries are compiled to native code using .NET Native[20] so all reversing will be done against x86 or ARM code depending on the target device. The code will be compiled from the same source so one advantage of this is that you can choose which architecture you're more comfortable reversing, and then choose the version of the binary compiled for that architecture by installing it on a device that runs on that architecture. Another thing to consider is that while we can deal with code written in C++ like we've always done, if the code was written originally in C# or Visual Basic, it would be different. Binaries written in .NET languages are compiled into IL (intermediate language) code and we can decompile them using .NET decompilers like ILSpy[21] or .NET Reflector [22]. With UWP apps they are now compiled into native code so we can't use those decompilers anymore so we have to deal with the idiosyncrasies in the resulting native code due to the conversion done.

4.4 Dynamic analysis

Now we'll take look at how to dynamically analyze a Windows 10 Core binary using a debugger.

4.4.1 Kernel Debugging using WinDbg

To dynamically reverse kernel-level code such as device drivers, we need to do kernel debugging using WinDbg. The general instructions to do this can be found here[23].

Let's use the Raspberry Pi 3 as an example. We will be using a Shikra[24] as our USB-to-UART adapter but feel free to use any of other ones like the Bus Pirate[25]. You can get the pin mappings for the Shikra here[26] and for the Raspberry Pi 2 & 3 here[27].

First, connect the Shikra's TX pin to the Raspberry Pi's RX pin, and the Shikra's RX pin to the Raspberry Pi's TX pin. Connect the ground pins for both as well.

---

[20] "Compiling Apps with .NET Native" **https://msdn.microsoft.com/en-us/library/dn584397(v=vs.110).aspx**

[21] ILSpy .NET Decompiler **http://ilspy.net/**

[22] .NET Reflector **http://www.red-gate.com/products/dotnet-development/reflector/**

[23] "Debugging Windows 10 IoT Core Devices Using WinDbg" **https://developer.microsoft.com/en-us/windows/iot/win10/windbg**

[24] "Using Shikra To Attack Embedded Systems" **http://www.xipiter.com/musings/using-the-shikra-to-attack-embedded-systems-getting-started**

[25] Bus Pirate **http://dangerousprototypes.com/docs/Bus_Pirate**

[26] "Shikra pinouts" **http://www.xipiter.com/uploads/2/4/4/8/24485815/shikra_documentation.pdf**

[27] "Raspberry Pi 2 & 3 Pin Mappings" **https://developer.microsoft.com/en-us/windows/iot/win10/samples/pinmappingsrpi2**

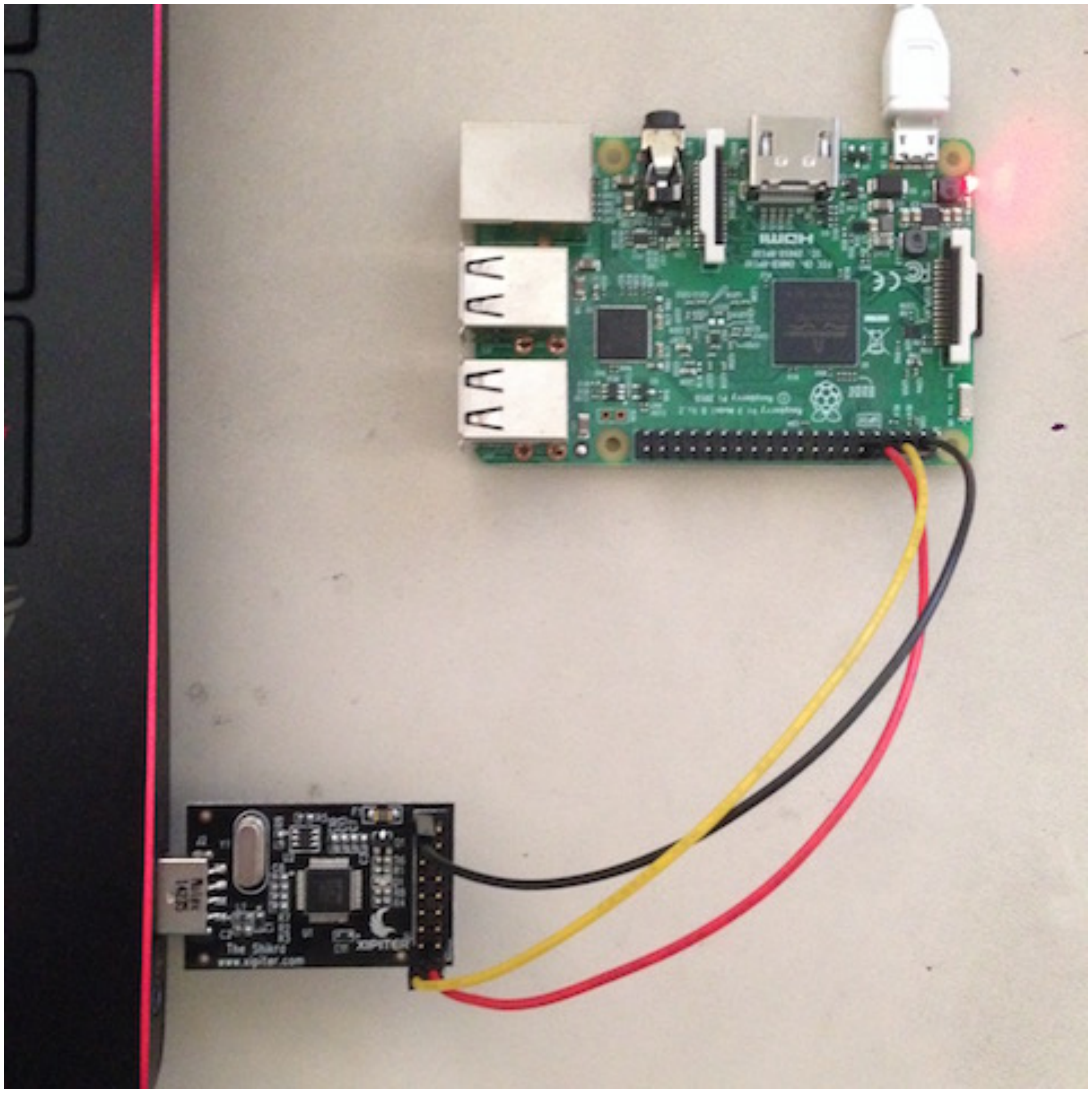


Figure 7. Connecting the Shikra to a Raspberry Pi 3's UART pins

Next, connect to your device using remote PowerShell or SSH. You will then need to enable serial debugging and turn on turn on debugging with the following commands:

```
# Enable serial debugging
bcdedit -dbgsettings serial
# Turn on debugging
bcdedit -debug on
```

You can find out the COM port used by your USB-to-serial adapter by using the Device Manager, or by running the following PowerShell command:

```
Get-WMIObject Win32_pnpentity | ? Name -like "*Serial*COM*"
```

Here's a sample output:

```
__GENUS                      : 2
__CLASS                      : Win32_PnPEntity
__SUPERCLASS                 : CIM_LogicalDevice
__DYNASTY                    : CIM_ManagedSystemElement
__RELPATH                    : Win32_PnPEntity.DeviceID="FTDIBUS\\VID_0403+PID_6014+5&32
78CBC5&0&3\\0000"
__PROPERTY_COUNT             : 26
__DERIVATION                 : {CIM_LogicalDevice, CIM_LogicalElement, CIM_ManagedSystem
Element}
__SERVER                     : DESKTOP-39HUL88
__NAMESPACE                  : root\cimv2
__PATH                       : \\DESKTOP-39HUL88\root\cimv2:Win32_PnPEntity.DeviceID="FT
DIBUS\\VID_0403+PID_6014+5&3278CBC5&0&3\\0000"
Availability                 :
Caption                      : USB Serial Port (COM3)
ClassGuid                    : {4d36e978-e325-11ce-bfc1-08002be10318}
CompatibleID                 :
ConfigManagerErrorCode       : 0
ConfigManagerUserConfig      : False
CreationClassName            : Win32_PnPEntity
Description                  : USB Serial Port
DeviceID                     : FTDIBUS\VID_0403+PID_6014+5&3278CBC5&0&3\0000
ErrorCleared                 :
ErrorDescription             :
HardwareID                   : {FTDIBUS\COMPORT&VID_0403&PID_6014}
InstallDate                  :
LastErrorCode                :
Manufacturer                 : FTDI
Name                         : USB Serial Port (COM3)
PNPClass                     : Ports
PNPDeviceID                  : FTDIBUS\VID_0403+PID_6014+5&3278CBC5&0&3\0000
PowerManagementCapabilities :
PowerManagementSupported     :
Present                      : True
Service                      : FTSER2K
Status                       : OK
StatusInfo                   :
SystemCreationClassName      : Win32_ComputerSystem
SystemName                   : DESKTOP-39HUL88
PSComputerName               : DESKTOP-39HUL88
```

In the above example, the USB-to-serial adapter uses COM3. You can now remotely debug the device from your machine by running the following command (Make sure you are using the x86 version of WinDbg):

```
# PORT is the COM port number used by your USB-to-serial adapter
windbg.exe -k com:port=<PORT>,baud=921600
```

If all goes well you will see WinDbg spawned like this:

```
Microsoft (R) Windows Debugger Version 10.0.10586.567 X86
Copyright (c) Microsoft Corporation. All rights reserved.
```

```
Opened \\.\com3
Waiting to reconnect...
```

Restart the Raspberry Pi and you will see this:

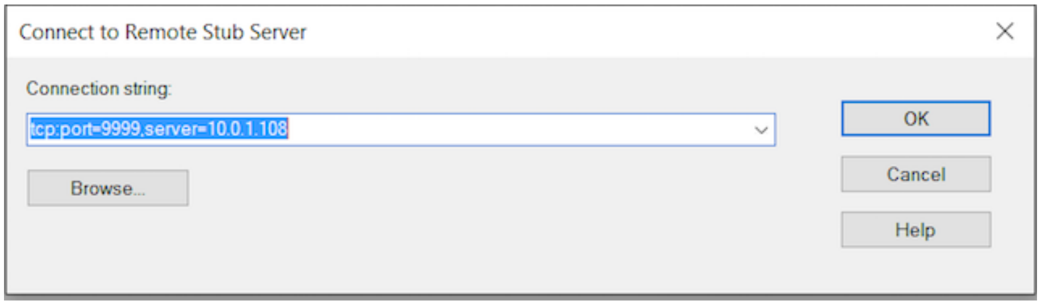
```
Connected to Windows 10 14393 ARM (NT) Thumb-2 target at (Sun Jul 24 19:32:43.111 2016
(UTC + 8:00)), ptr64 FALSE
Kernel Debugger connection established.
Symbol search path is: srv*
Executable search path is:
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntkrnlmp.ex
e -

Windows 10 Kernel Version 14393 MP (1 procs) Free ARM (NT) Thumb-2
Built by: 14393.0.armfre.rs1_release.160715-1616
Machine Name:
Kernel base = 0x80c1b000 PsLoadedModuleList = 0x80e07c78
System Uptime: 0 days 0:00:00.000
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
*                                                                             *
*   You are seeing this message because you pressed either                   *
*       CTRL+C (if you run console kernel debugger) or,                       *
*       CTRL+BREAK (if you run GUI kernel debugger),                          *
*   on your debugger machine's keyboard.                                      *
*                                                                             *
*                   THIS IS NOT A BUG OR A SYSTEM CRASH                       *
*                                                                             *
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntkrnlmp.ex
e -
nt!DbgBreakPointWithStatus:
80c40d90 defe     __debugbreak
```

4.4.2 Debugging user mode processes using WinDbg

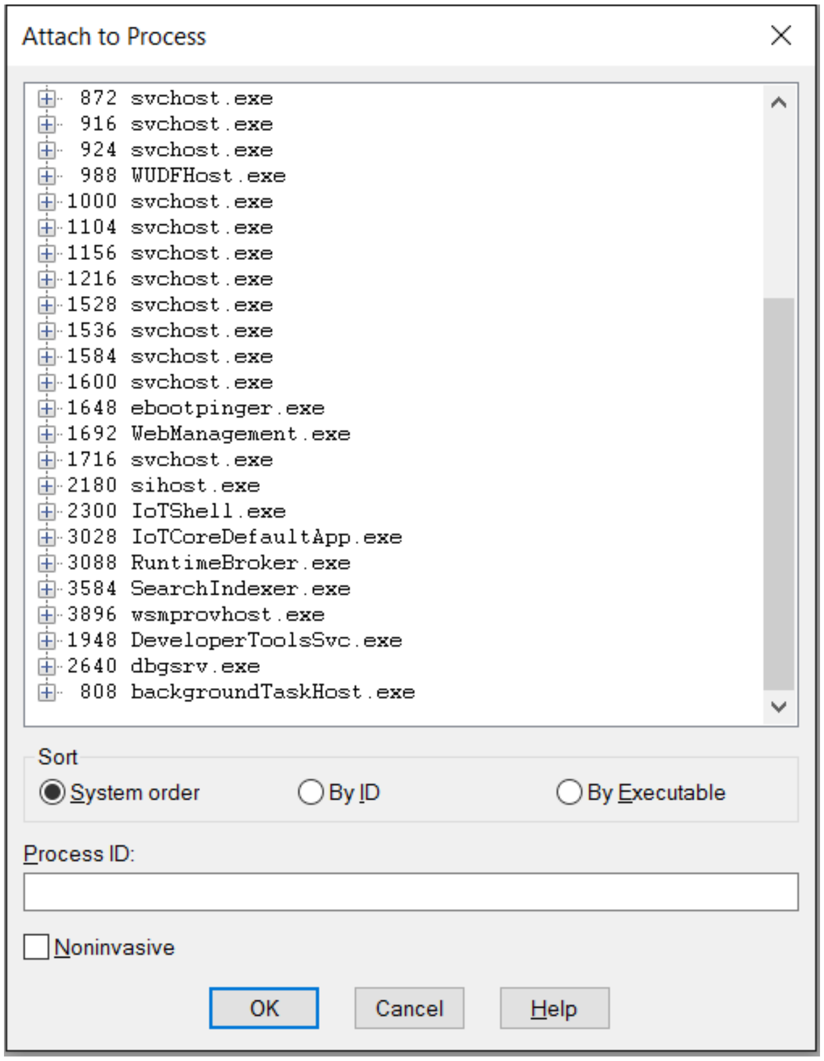
Debugging user mode processes is a bit easier than kernel-mode debugging. You only need a network connection to your device. We are going to use dbgsrv.exe (which can be found on the device's C:\Windows\System32\Debuggers folder) on the device and Windbg on the debugging host machine. First we need to make dbgsrv.exe listen on a port on the device so we can connect to it. On the device, run the following command using PowerShell or SSH:

```
# PORT is the local port you want dbgsrv to listen on
dbgsrv.exe -t tcp:port=<PORT>
```

In Windbg running on the debugging host, go to File > Connect to Remote Stub and enter the IP address of you device and your chosen port in the format shown, then click OK:


Connect to Remote Stub Server
Connection string:
tcp:port=9999,server=10.0.1.108
Browse...
OK
Cancel
Help

Go to File > Attach to a Process and select the process you want to attach to:

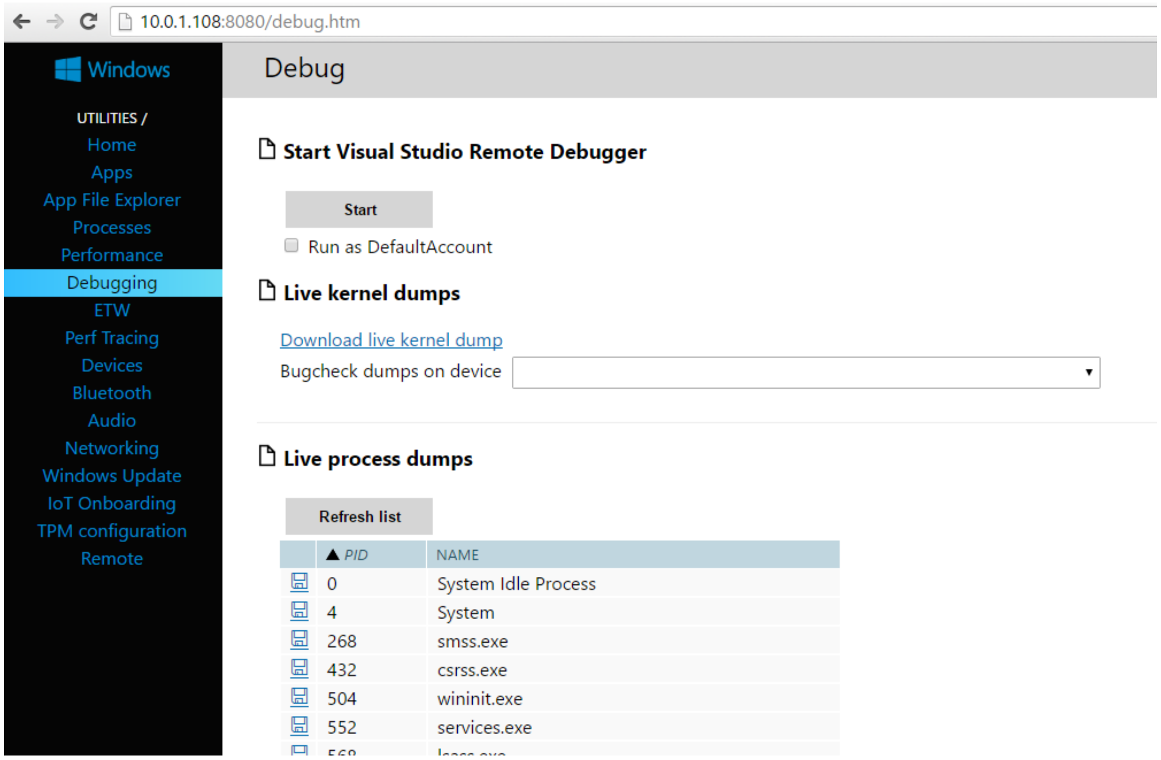


```
Microsoft (R) Windows Debugger Version 10.0.10586.567 X86
Copyright (c) Microsoft Corporation. All rights reserved.

*** wait with pending attach
Symbol search path is: srv*
Executable search path is:
ModLoad: 01110000 011db000   C:\windows\system32\WebManagement.exe
ModLoad: 77400000 77565000   C:\windows\SYSTEM32\ntdll.dll
```

```
ModLoad: 77270000 773fe000   C:\windows\System32\KERNELBASE.dll
ModLoad: 76fc0000 771cb000   C:\windows\System32\combase.dll
ModLoad: 76e60000 76f0e000   C:\windows\System32\ucrtbase.dll
ModLoad: 76cb0000 76d2c000   C:\windows\system32\msvcrt.dll
ModLoad: 76f10000 76fbe000   C:\windows\System32\RPCRT4.dll
ModLoad: 76e20000 76e58000   C:\windows\System32\kernel32legacy.dll
ModLoad: 76dd0000 76e1a000   C:\windows\System32\bcryptPrimitives.dll
ModLoad: 77230000 7726c000   C:\windows\System32\sechost.dll
ModLoad: 76460000 76489000   C:\windows\system32\IPHLPAPI.DLL
ModLoad: 76490000 764ea000   C:\windows\system32\WS2_32.dll
<snip...>
(69c.280): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
77422740 defe     __debugbreak
0:005> !peb
PEB at 00928000
    InheritedAddressSpace:    No
    ReadImageFileExecOptions: No
    BeingDebugged:            Yes
    ImageBaseAddress:         01110000
    Ldr                       774eb9e0
    Ldr.Initialized:          Yes
    Ldr.InInitializationOrderModuleList: 00c41738 . 00c4fcd0
    Ldr.InLoadOrderModuleList:           00c41810 . 00c4fcc0
    Ldr.InMemoryOrderModuleList:         00c41818 . 00c4fcc8
            Base TimeStamp                     Module
         1110000 57898ebe Jul 16 09:32:46 2016 C:\windows\system32\WebManagement.exe
        77400000 57898ba5 Jul 16 09:19:33 2016 C:\windows\SYSTEM32\ntdll.dll
        77270000 57898c4c Jul 16 09:22:20 2016 C:\windows\System32\KERNELBASE.dll
        76fc0000 57898c6d Jul 16 09:22:53 2016 C:\windows\System32\combase.dll
        76e60000 57898b83 Jul 16 09:18:59 2016 C:\windows\System32\ucrtbase.dll
        76cb0000 57898fad Jul 16 09:36:45 2016 C:\windows\system32\msvcrt.dll
        76f10000 57898cd8 Jul 16 09:24:40 2016 C:\windows\System32\RPCRT4.dll
        76e20000 57898eb7 Jul 16 09:32:39 2016 C:\windows\System32\kernel32legacy.dll
        76dd0000 57898f49 Jul 16 09:35:05 2016 C:\windows\System32\bcryptPrimitives.dll
        77230000 57898f0a Jul 16 09:34:02 2016 C:\windows\System32\sechost.dll
        76460000 57898c40 Jul 16 09:22:08 2016 C:\windows\system32\IPHLPAPI.DLL
        76490000 57898eaf Jul 16 09:32:31 2016 C:\windows\system32\WS2_32.dll
<snip...>
0:005> u $exentry
*** ERROR: Module load completed but symbols could not be loaded for WebManagement.exe
WebManagement+0xa6631:
011b6630 e92d4800 push        {r11,lr}
011b6634 46eb     mov         r11,sp
011b6636 f000fb65 bl          WebManagement+0xa6d04 (011b6d04)
011b663a e8bd4800 pop         {r11,lr}
011b663e f7ffbf25 b.w         WebManagement+0xa648c (011b648c)
011b6642 0000     movs        r0,r0
011b6644 f24c6c64 mov         r12,#0xC664
011b6648 f2c01c1c movt        r12,#0x11C
```

4.4.3 Crash dump analysis

Crash dumps can be found in the C: \CrashDump folder on the device, but you can also generate live dumps for the kernel or any user-mode process by using the Windows Device Portal's Debugging tab.



PID	NAME
0	System Idle Process
4	System
268	smss.exe
432	csrss.exe
504	wininit.exe
552	services.exe

Figure 8. Debugging tab of the Windows Device Portal

As an example, let's download a live process of dump the user mode process WebManagement.exe. Click on the icon on the left side of the process name to download the dump to your browser's Download folder. From WinDbg go to File > Open Crash Dump, and you're good to go.

```
Microsoft (R) Windows Debugger Version 10.0.10586.567 X86
Copyright (c) Microsoft Corporation. All rights reserved.


Loading Dump File [d:\winiot\WebManagement.exe-LiveUM-2016-07-24-12-36-09.dmp]
User Mini Dump File: Only registers, stack and portions of memory are available

Symbol search path is: srv*
Executable search path is:
Windows 10 Version 14376 MP (4 procs) Free ARM (NT) Thumb-2
Product: WinNt, suite: SingleUserTS
Built by: 10.0.14376.0 (rs1_release.160624-1700)
Machine Name:
Debug session time: Mon Jul 25 03:36:09.000 2016 (UTC + 8:00)
System Uptime: not available
Process Uptime: 1 days 4:48:37.000
................................................................
........
```

```
Loading unloaded module list
.
Cannot read PEB32 from WOW64 TEB32 ffffffff - Win32 error 0n30
Unable to load image C:\Windows\System32\ntdll.dll, Win32 error 0n2
*** WARNING: Unable to verify timestamp for ntdll.dll
ntdll!NtWaitForSingleObject+0x6:
*** WARNING: Unable to verify timestamp for KERNELBASE.dll
77320ab6 4770     bx          lr {KERNELBASE!WaitForSingleObjectEx+0xc0 (76fedf30)}
0:000> |
.  0    id: 698 examine name: C:\Windows\System32\WebManagement.exe
0:000> !peb
PEB at 032f8000
    InheritedAddressSpace:    No
    ReadImageFileExecOptions: No
    BeingDebugged:            No
    ImageBaseAddress:         00a00000
    Ldr                       773eb9e0
    Ldr.Initialized:          Yes
    Ldr.InInitializationOrderModuleList: 034a1730 . 034ae758
    Ldr.InLoadOrderModuleList:           034a1808 . 034ae748
    Ldr.InMemoryOrderModuleList:         034a1810 . 034ae750
            Base TimeStamp                     Module
          a00000 576dee48 Jun 25 10:36:56 2016 C:\windows\system32\WebManagement.exe
        77300000 576deb18 Jun 25 10:23:20 2016 C:\windows\SYSTEM32\ntdll.dll
        76f20000 576debe7 Jun 25 10:26:47 2016 C:\windows\System32\KERNELBASE.dll
        770b0000 576debda Jun 25 10:26:34 2016 C:\windows\System32\combase.dll
        76ce0000 576deb16 Jun 25 10:23:18 2016 C:\windows\System32\ucrtbase.dll
        76e30000 576ded32 Jun 25 10:32:18 2016 C:\windows\System32\RPCRT4.dll
        76de0000 576dee1b Jun 25 10:36:11 2016 C:\windows\System32\kernel32legacy.dll
        76d90000 576deeaa Jun 25 10:38:34 2016 C:\windows\System32\bcryptPrimitives.dll
```

4.5 Fuzzing approaches

Fuzzing is one of the most effective ways in finding vulnerabilities in software. It's a no brainer to attempt to this on Windows 10 IoT Core as well. Unfortunately due to the lack of existing tools for this OS the approach we have been doing is far from efficient - spawning processes using a debugger, no coverage measurement, etc. Remote control (device restart, process start/stop, crash dump collection) is done through the Windows Device Portal's REST APIs. It's basically fuzzing like its 2007. Also the device's low CPU power severely limits the rate of fuzzing iterations we can do.

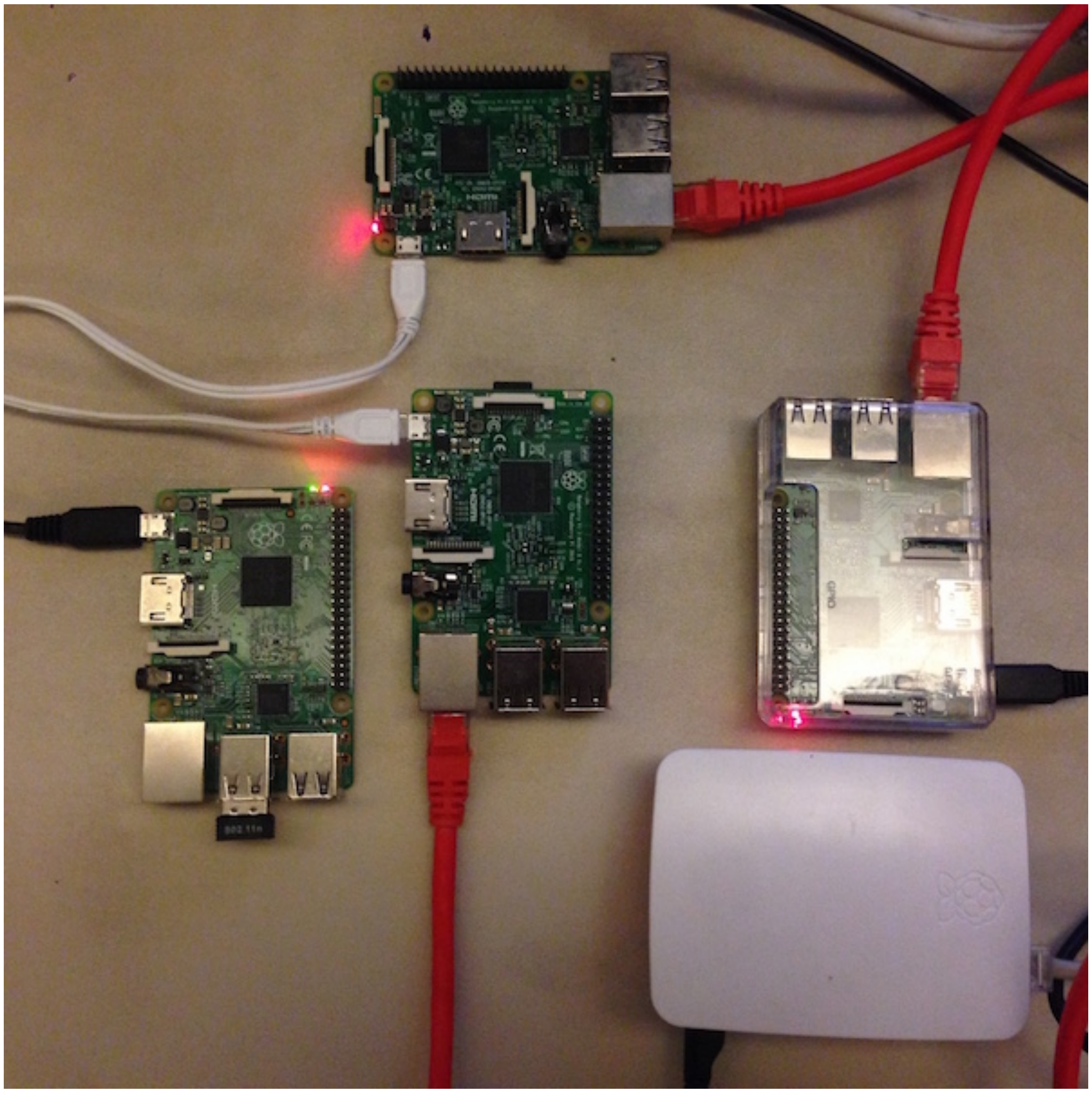


Figure 9. The author's lackluster fuzz cluster

One interesting approach would be corpus driven fuzzing[28]. We believe this will be effective especially when fuzzing a UWP app which it is possible to get hold hold of a build for the Windows 10 desktop e.g. from the Windows Store. Basically we don't fuzz on the device. We fuzz the app using the desktop version using whatever means of instrumentation to measure code coverage and we collect the corpora (samples) that resulted in wider code coverage as measured by the instrumentation. After collecting the best ones, we can then apply these corpora on the app running on the device without having to instrument it. All we have to then is collect the crashes and analyze them for exploitability. This way we can at least ease the load on the CPU and won't need as much tooling.

[28] "The Art of Fuzzing Without Fuzzing" **https://github.com/bnagy/slides/blob/master/fuzzing_without_pub.pdf**

Of course, this approach won't be applicable if you want to fuzz drivers for peripherals, or fuzz apps that interact with hardware. In those cases you have to do on-device fuzzing. However, there are some promising developments recently that may make the situation better. One of those is the release of WinAFL[29]. WinAFL is a Windows fork of the very popular fuzzer AFL[30]. While AFL uses compile-time instrumentation, WinAFL uses DynamoRIO for dynamic instrumentation to measure coverage. The challenge right now is to make DynamoRIO work on an ARM device like the Raspberry Pi. This is currently an ongoing project that we have embarked and hopefully we will have something to show for the effort in the near future.

5 RECOMMENDATIONS

Let's summarize the various ways we can minimize the risks against Windows 10 IoT Core devices.

5.1 Segment your network s

Segregating your IoT devices from your traditional computing devices such as laptops and servers is highly recommended. This is especially effective in cases where one of your machines have been compromised, and the attacker is looking to laterally move through network. This will also effective in isolating incidents and conducting cleanups.

5.2 Disable unnecessary network services

Network services that are not used in production should be disabled. Services that are enabled by default in Windows IoT Core include SSH and Windows File Sharing. To disable file sharing on startup, run the following command using SSH or PowerShell:

```
reg add HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\lanmanserver /v Start /t R
EG_DWORD /d 0x3 /f
```

5.3 Change Default Administrator Password

The Administrator password is hardcoded in the Windows 10 IoT Core image. Default login credentials that the user failed to change is still the most common way in which a malware infects an IoT devices. Changing the default password immediately after install will go a long way in avoiding unauthorized access. You can change the default Administrator password by using the following command using SSH or PowerShell:

```
net user Administrator <new password>
```

[29] "A fork of AFL for fuzzing Windows binaries" https://github.com/ivanfratric/winafl

[30] American Fuzzy Lop (AFL) http://lcamtuf.coredump.cx/afl/

5.4 Use a device that supports TPM

Using Raspberry Pis for hobby projects is fine, but if you are going to build a device that is going to be used in more sensitive situations e.g. home security, you should be using boards that support TPM. Your choices for now should be between a Minnowboard Max and a Dragonboard 410c, or use a discrete TPM with a Raspberry Pi.

5.5 Take advantage of available security features

So now you're using a board with TPM. Make sure you enable and setup security features such as Secure Boot and BitLocker.

6 CONCLUSION

In this paper, we laid out the various attack surfaces that may be taken advantage of by attackers to gain access to a Windows 10 IoT Core device. We also enumerated techniques to get you started in analyzing Windows 10 IoT Core devices. Built-in features like PowerShell and the many security-related tools written in it helps a great deal in assessing the security of a device. We also learned that leaving device configurations at its default settings, among other things, are a sure fire way to leave your device susceptible to attacks, and we gave some recommendations on how to avoid this.

Windows 10 IoT Core is still in its early stage, but we believe that once its matures it will become a more viable alternative to the other IoT focused OSes that currently exist. Aside from the many features this OS offers - including the security features we discussed earlier in this paper - what makes this OS attractive is that there are a lot of enterprises and developers already invested in Microsoft technologies, and they can leverage the knowledge and expertise they already have in developing the IoT devices of the future. As such, we expect Windows 10 IoT Core to become one of the major IoT OSes in the future. This would also mean that this OS will be a more attractive target for attackers, and it is our hope that to counter this, more people will engage in security research targeting this OS, and that this paper has somehow help encouraged it.

Finally, any corrections, questions, or comments regarding this paper are very much appreciated. The author can be reached at sabanapm[at]ph[dot]ibm[dot]com.