Compromising Windows: Mitigating Defender and Automating Payload Delivery with Arduino

By

Daniel Contreras-Gallardo, Tamia Hunter, Shaniya Russell, Joseph Vinyard

College of Business and Computing, Department of Computer Science

Georgia Southwestern State University

CSCI 4940: Capstone

Dr. Linqiang Ge

20200513

(This page intentionally left blank)

Compromising Windows: Mitigating Defender and Automating Payload Delivery with Arduino

By

Daniel Contreras-Gallardo, Tamia Hunter, Shaniya Russell, Joseph Vinyard

Abstract: The main goal of this project is to demonstrate and educate the average end user in regard to how easily a computer can be compromised if physical access is achieved. Specifically, the project focuses on utilizing a seemingly innocuous USB device to disable security settings of a computer using trusted applications.

The scope of this project includes defining what a "BadUSB" is, how this specific threat vector *abuses* the inherent trust that computers have with USB devices, an overview in setting up the scripting and testing environments and finally, discussing pragmatic solutions and steps that the end user can take in order to mitigate these types of attacks.

We made use of a few readily available and open-source resources which included Duckyscript, a Duckyscript to Arduino converter service, the Arduino IDE (our scripting environment), VirtualBox (to establish our testing environment), and the Metasploit-Framework, an open source penetration framework used to test and validate security vulnerabilities. In addition, this attack makes use of work demonstrated by Dan Tentler and incorporates an AMSI (Antimalware Scripting Interface) bypass originally credited to Matt Graeber.

Research findings have shown that for *this* specific attack, the first line of defense is to *prevent* physical access. In the event that physical access *cannot* be mitigated, we've found that hardening default privilege escalation (Priv-Esc) prompts from the default settings *can* prevent this type of attack from being successful. There are multiple ways to compromise a machine if physical access is achieved, we are showing one way of compromise and one way to defend against such attacks.

(This page intentionally left blank)

Table of Contents

(This page intentionally left blank)

Introduction

What *is* a BadUSB Device?

Computers inherently trust USB devices. BadUSB attacks take advantage of that inherent trust and abuse it. The first known example of BadUSB is attributed to security researchers Karsten Nohl and Jakob Lell. Nohl and Lell presented their research on how to create a BadUSB at Black Hat 2014, a popular hacking and security conference held annually in Las Vegas, Nevada. Their research found that if a device's controller chip can be reprogrammed, the device can then be programmed to behave maliciously.

This is accomplished by reprogramming the device's firmware to alter its product identification and vendor notification numbers (PID, VID), which uniquely identify whether a device is say, a USB thumb drive typically used for portable storage, a USB keyboard, or any other USB compatible peripheral device. Once the device's firmware has been reprogrammed the device can then be programmed to behave a certain way. The researchers state that this type of vulnerability isn't detected by traditional antivirus (AV) as AV doesn't scan firmware. In addition, the attack isn't limited to just USB thumb drives, as any peripheral USB device is susceptible so long as that USB device's firmware is programmable (Greenberg 2017).

Initially we wanted to try and recreate the attack with a USB thumb drive. Unfortunately, we found that there was a specific type of microcontroller that was needed in order to reprogram the firmware, and as the initial research conducted by Nohl and Lell was already six years old, many USB thumb drive manufacturers had stopped producing this specific chip. This led us to

our next avenue of approach, which was to take a preexisting piece of hardware off the shelf and see if it the attack could be recreated this way. Enter the Arduino Pro Micro.

The Arduino Pro Micro

We opted for the Arduino Pro Micro programmable microcontroller for a couple of reasons. First, the Arduino Pro Micro is relatively inexpensive compared to similar products such as the USB "Rubber Ducky". Second, the Arduino Pro Micro also incorporates the Arduino IDE, a cross platform application used to compile and upload code using functions written in either the C or C++ languages to Arduino compatible boards. We chose this specific software as the learning curve isn't very steep, there's plenty of documentation and supporting resources for end users to familiarize themselves with when setting up their scripting environment.

Methods – Setting up our environments

The Arduino IDE

The boards were purchased from Amazon and considered "generic" Arduino boards. As such, they didn't include instructions so we had to reference the Arduino documentation, forums and other publications in order to set the Arduino IDE up in such a way that we'd be able to have the IDE compile the code, and more importantly, communicate with and upload the code to the Arduino boards. After extensive research we were finally able to track down a well written "how-to" guide by user "Jimblom" in tutorial titled "*Pro Micro & Fio V3 Hookup Guide*" (Jimblom 2013). The first step after installing the Arduino IDE is to install the correct boards using the Board Manager feature of the IDE. We found this out the hard way and ended up bricking one of the first boards. To ensure accurate board installation, we would need to select

Preferences>Additional Board Manager URLs and then add the following URL into the textbox: https://raw.githubusercontent.com/sparkfun/Arduino_Boards/master/IDE_Board_Manager/package_sparkfun_index.json (Jimblom 2013).

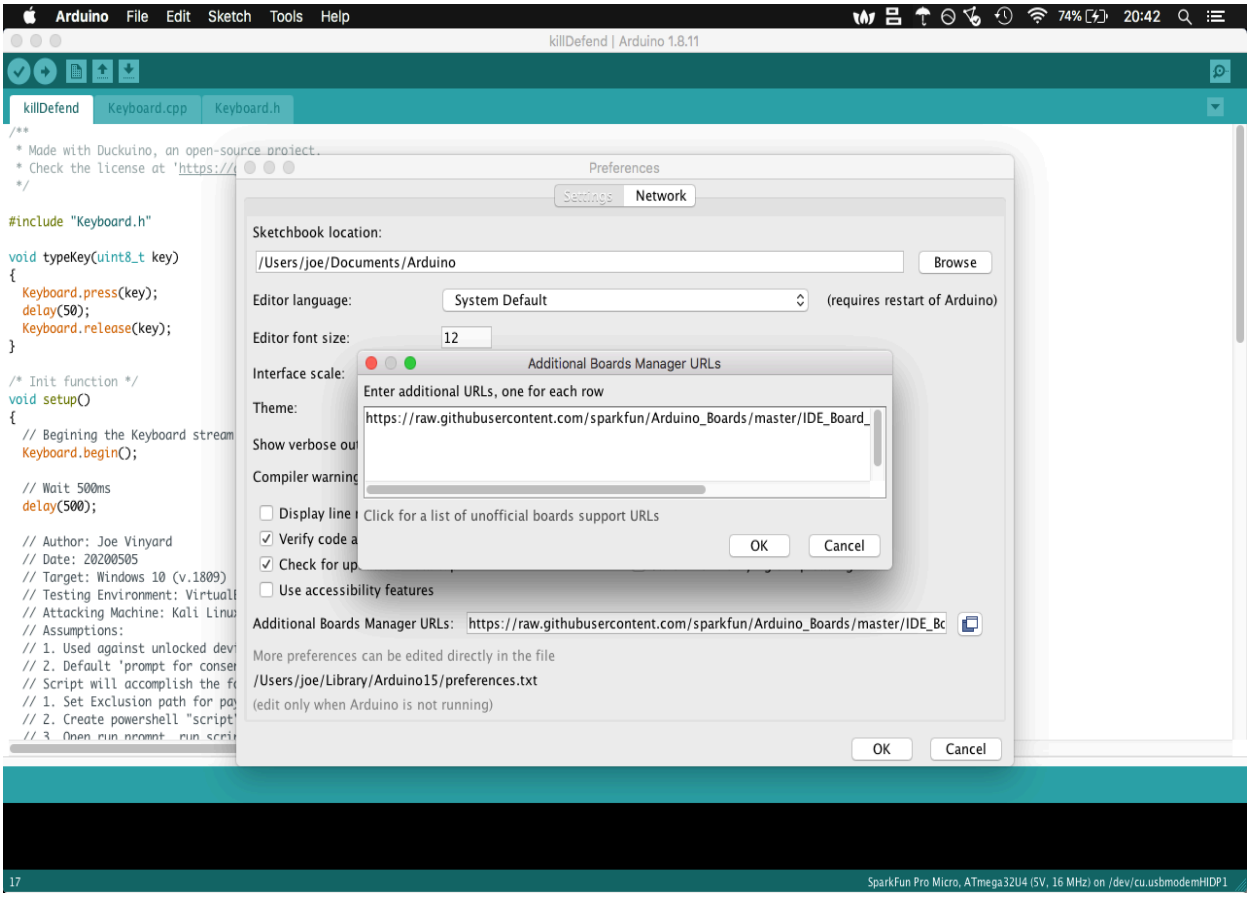


Figure 1: Pointing the Arduino IDE to a custom URL (all images pertaining to Arduino IDE set up courtesy of Joe Vinyard)

After having pointed the IDE to a custom URL, we then need to select Tools>Board Manager

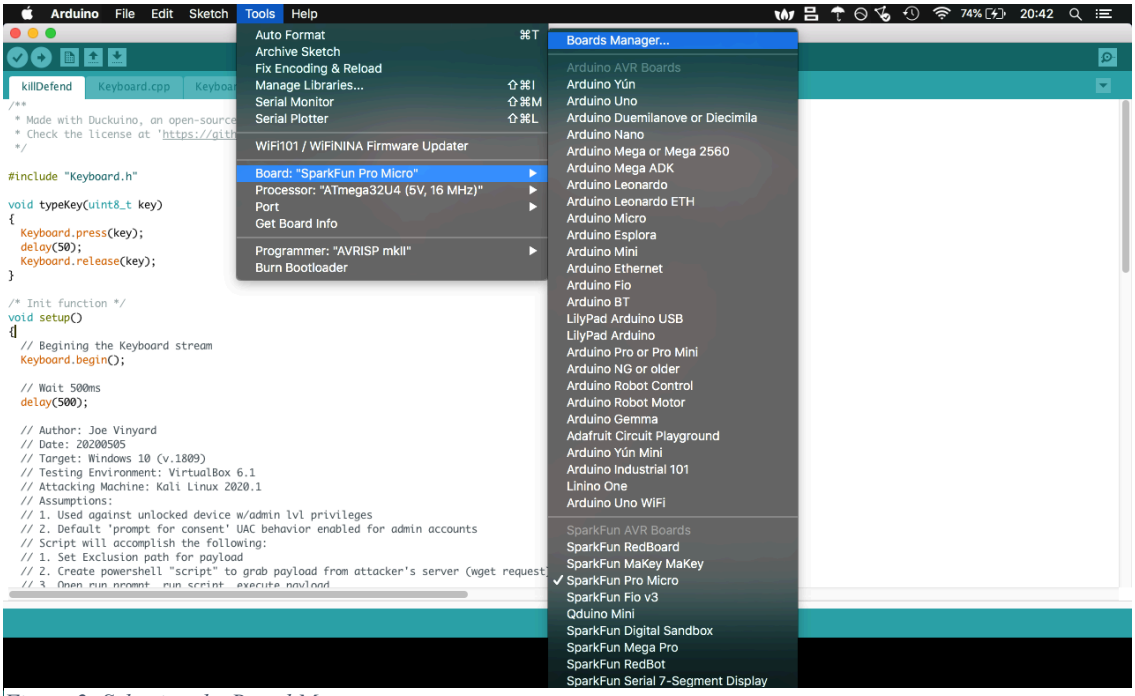


Figure 2: Selecting the Board Manager

and then search for “sparkfun” in the following text box. From there, find the “Sparkfun AVR Boards” and select “Install”.

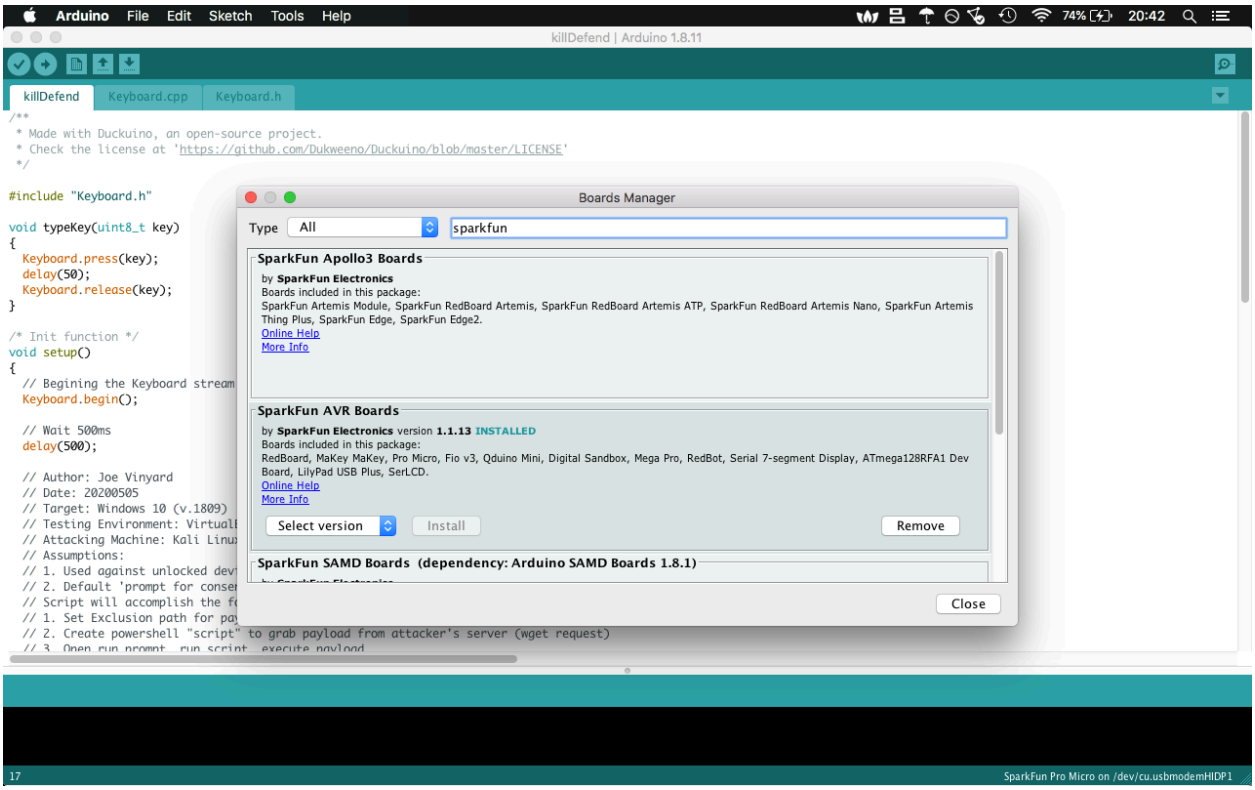


Figure 3: Installing the SparkFun AVR Boards

After the boards are installed, the last thing we need to do is ensure we’ve selected the appropriate board under Tools>Board. The board we need to select is the “SparkFun Pro Micro”, then under “Processor” we would need to select “ATmega32U4 (5V, 16 MHz)” and the appropriate port under “Port” as referenced by Figures 4, 5 and 6. In my case, I’m on an Apple MacBook Pro, so my port was “/dev/cu.usbmodemHIDP1”. Depending on which operating system you’re running, your port will be different.

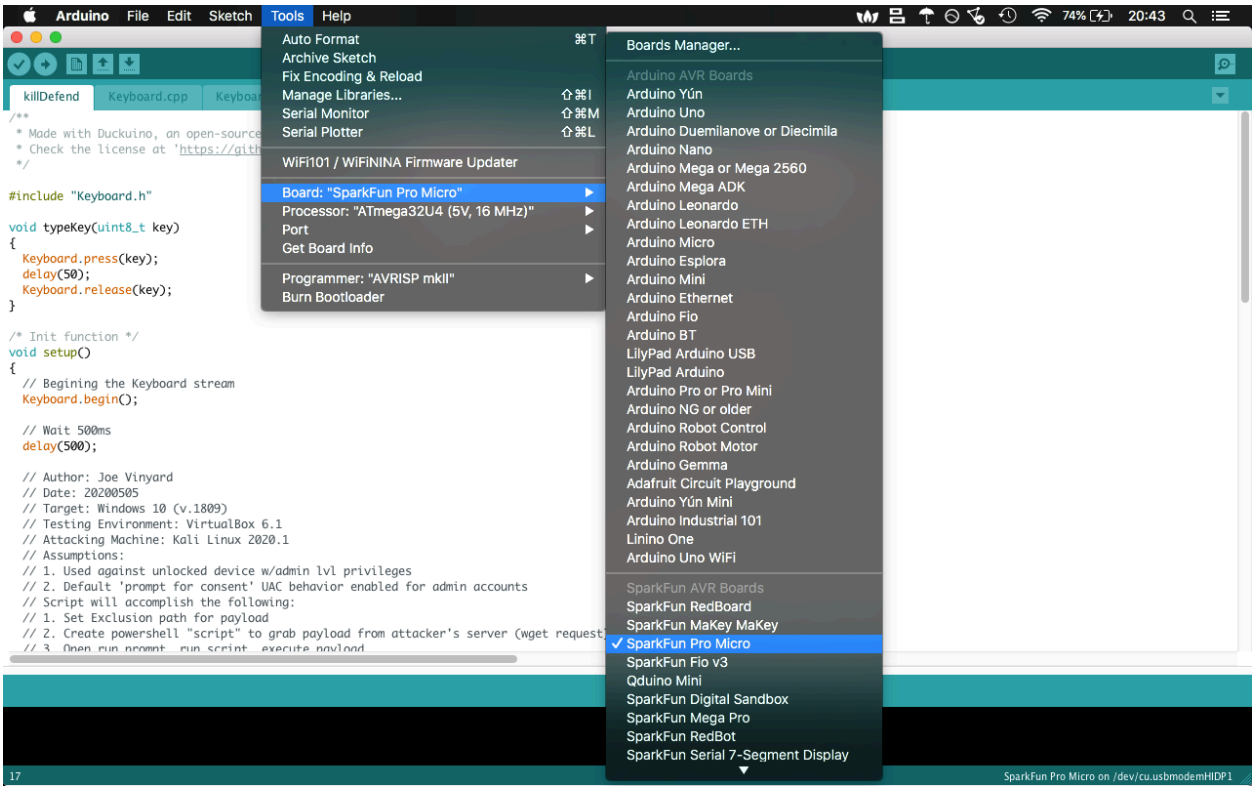


Figure 4: Selecting SparkFun Pro Micro Under Board Management

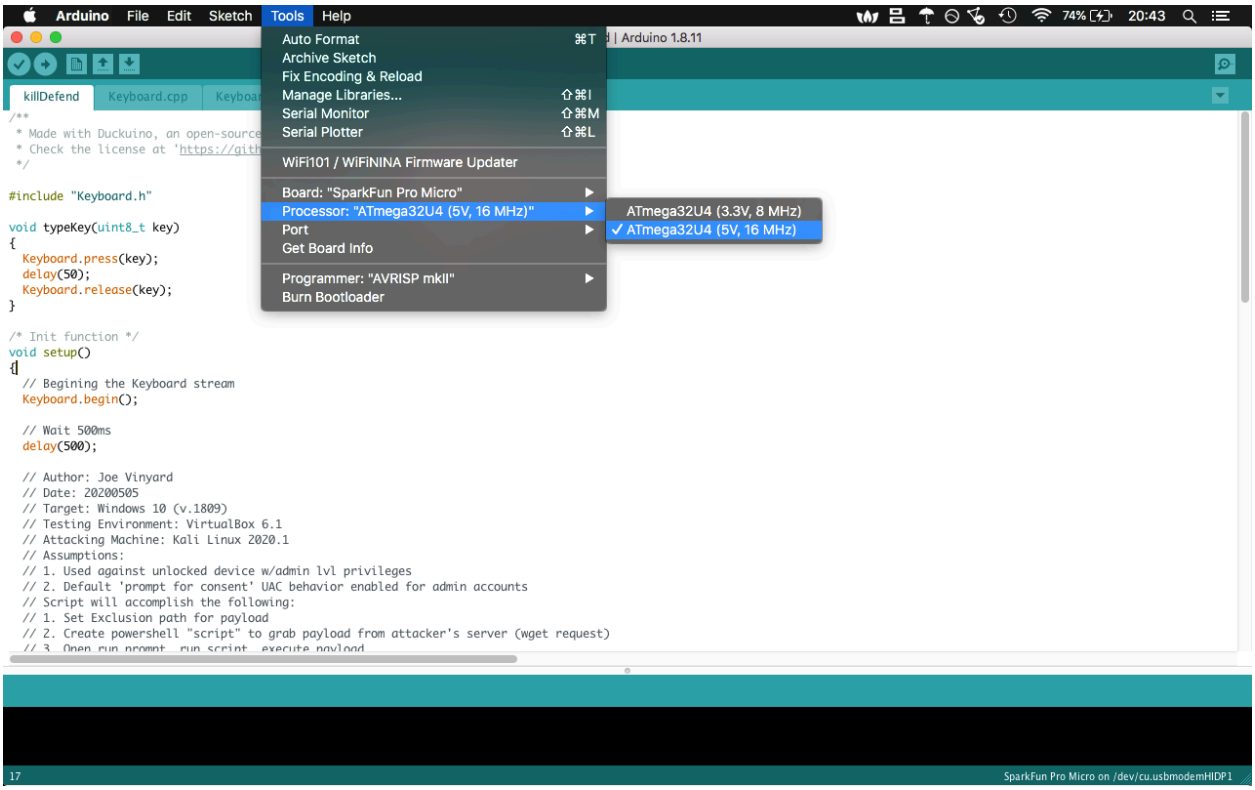


Figure 5: Selecting Appropriate Processor

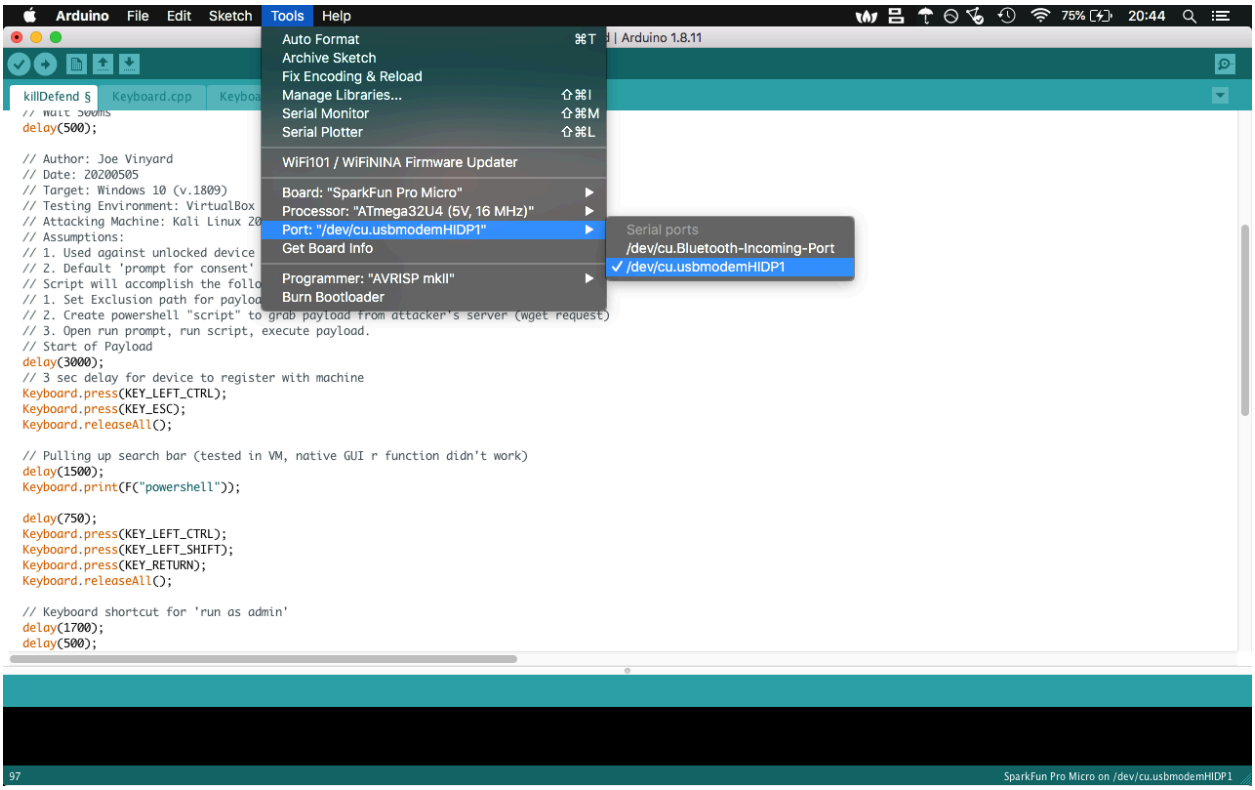


Figure 6: Selecting Appropriate Port

Now that all of the board specific settings are applied, we can write, edit, compile and most importantly, upload the code to our programmable micro controller. Next, we needed an environment to test and debug our code. Enter VirtualBox.

VirtualBox and Installing Kali Linux

VirtualBox was used as a way for us to test and debug our planned scripts without having to worry about any permanent damage to the host machine. We would test our scripts against Windows 10 (version 1809) and utilize Kali Linux as our command and control server (C^2) to not only host the payloads, but also be used to interact with the victim machine once we were able to establish a remote connection with the aid of the Arduino script I crafted.

Kali Linux is a popular penetration testing Linux distribution which is built on top of the Debian distribution of the Linux Kernel. Previously built on the Knoppix Linux distribution under the name "BackTrack", it was re-written to include a "rolling" distribution with near daily rolling updates to ensure the most up to date tools, libraries, databases, security updates, etc. are available to end users. Officially released as Kali Linux in 2013 by Offensive Security, it has gone through many changes throughout the years with the most recent version as of the time of this writing being Kali Linux 2020.1a (Kali.org 2020).

We chose to incorporate Kali Linux because it is purpose-built for penetration testers, ethical hackers and anyone interested in learning more about cybersecurity. Originally, I wanted to craft a program from scratch to create a reverse shell that could be deployed on the victim machine to call back to the attacking computer. However, due to the fact I'm more comfortable scripting in UNIX/Linux than I am in Windows, there would be a bit of a learning curve involved for me that might end up taking more time than we had. This is an important time constraint consideration since we were attempting to research a few things:

- Can we automate the creation *and* deployment of a reverse shell?
- Assuming we *can*:
    - Will we be able to accomplish the following?
        - Take screenshots of victim's desktop
        - Stream the victim's webcam
        - Transfer files between victim and attacking machine
        - Take pictures of the user with the webcam
        - Record the victim's microphone to eavesdrop on conversations

Assuming we would be able to develop a script to create and deploy a reverse shell from the Arduino, we would then need to create programs or scripts to run in the background to accomplish *each* of the subsequent tasks previously mentioned. Not one to attempt to reinvent the wheel and only having five weeks to deliver our findings, my research on these topics pointed me towards a purpose-built tool that was conveniently already baked into the Kali Linux operating system. This tool was none other than the Metasploit-Framework.

The Metasploit-Framework

Metasploit-Framework (hereafter referred to as *Metasploit*) is a very popular penetration testing framework used by security researchers, penetration testers and members of the "red" and "blue" teams, to name a few. Metasploit includes up-to-date information about security vulnerabilities, auxiliary scanners, payloads and countless other features that "enables you to find, exploit and validate vulnerabilities." (Rapid7). For the sake of brevity, we chose Metasploit since it would facilitate our ability to research the likelihood of reaching our objectives as they pertained to interacting with the victim machine once the remote connection was established. Now all that was left was to craft the initial payload to mitigate Windows Defender before downloading our malicious file from our C^2 server. The malicious file we're creating is a reverse shell with the aid of msfvenom, a module inside the Metasploit-Framework used to create payloads. When it comes to shells on a system there are two types; bind and reverse. With a bind shell, a new service is started on the target machine and the attacking machine must connect to that service in order to gain an interactive session with the target machine. A reverse shell on the other hand, accomplishes the interactive session in reverse order (hence the name). In a reverse shell, the *target machine calls back to the attacking computer* which has previously

set up a listener (to listen for incoming connections), and once that connection is established, the attacker can then interact with the remote target machine (rapid7). The following graphic depicts a reverse shell.

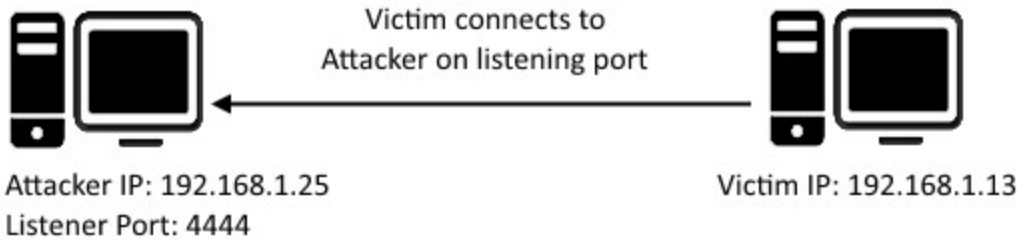


Figure 7: Reverse shell (source: https://irichmore.wordpress.com/2015/06/04/bind-shell-vs-reverse-shell/)

Matt Graeber and the AMSI Bypass

Attempting to mitigate the built-in, multi-layered defense that is Windows Defender is no small feat and a seemingly Herculean task for those who know little else about Windows Defender other than it helps to protect your computer from viruses and other malware. Having spent a little time behind the terminal troubleshooting, I at least knew where to search for information as it pertained to Windows Defender. My search lead me to an article written by Adam Chester, a security researcher and the Technical Director for MDSec, a UK based security consulting company. In his article, "*Exploring PowerShell AMSI and Logging Evasion"*, Chester breaks apart how AMSI, the Antimalware Scripting Interface, is used to run checks on PowerShell scripts that if left unchecked, might otherwise behave maliciously and how AMSI is designed to prevent that. He then goes into great detail explaining how a common bypass can be used and *how* this bypass tricks AMSI into thinking whatever file, script or command is being

run in PowerShell that would *normally* be detected, is bypassed (Admin, Chester 2018). The first known reference to the AMSI bypass was found via twitter from user @Mattifestation, also known as Matt Graeber, another security researcher at SpecterOps. Graeber's AMSI bypass is shown below:

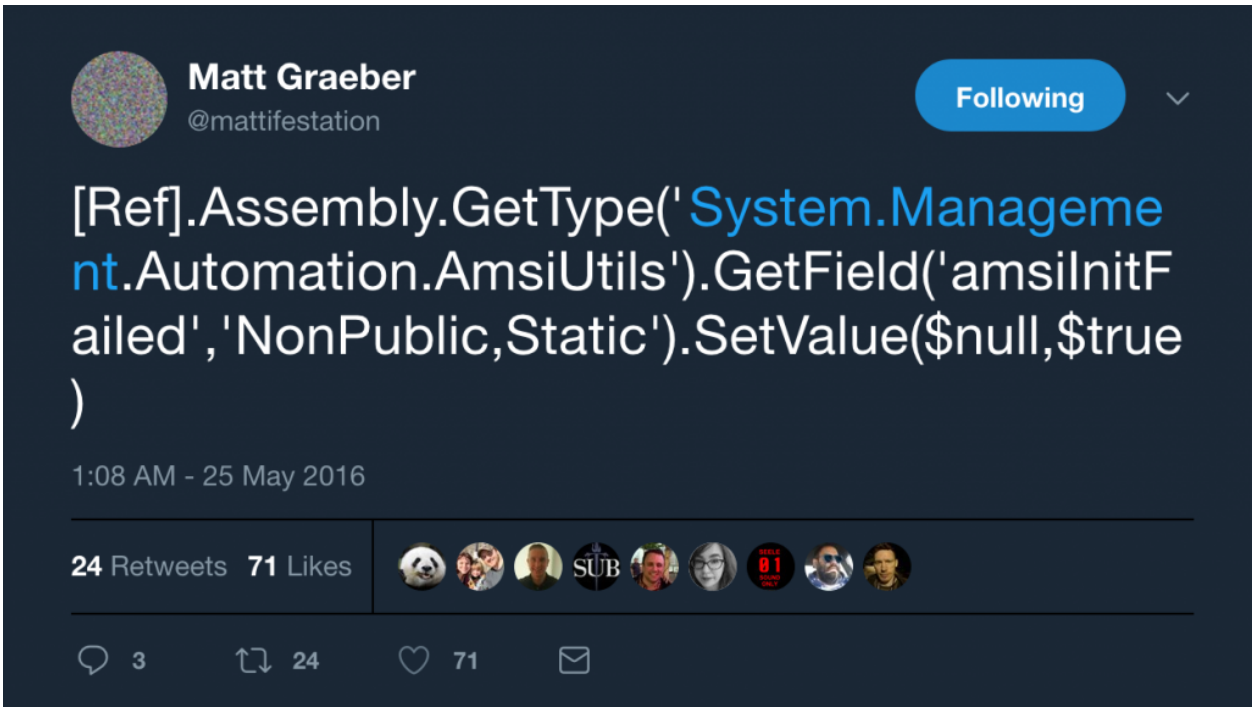


Figure 8: AMSI bypass (source Matt Graeber twitter.com/mattifestation/status/735261176745988096)

Disabling Defender for the LOLz

A popular technique that some ethical hackers take when conducting a penetration test or security audit is to incorporate a "living of the land" (LOL) approach. An adversary would use commands, features, applications, etc. that are already present on the device, networks, assets, etc. that they are attempting to audit or compromise. This reduces the footprint that a malicious actor would leave on a system as most of these applications have already been whitelisted which is to say that they're trusted applications and devices on the system. In our case, we made use of

both the cmd.exe and PowerShell applications to live off the land. Dan Tentler, the executive founder of Phobos Group, talks at length and demonstrates the efficacy of this approach when showing how an adversary with credentials to a virtual private network (VPN) client could be used to sniff around a domain and perform various attacks one they've established a connection on the private network (Kitchen, Tentler 2018).

With that methodology in mind, a series of commands are entered to roll back security definitions to their factory settings, exclusion paths are created and finally Matt Graeber's AMSI bypass is utilized. All of this is done to set the stage so that the malicious executable that will be created locally doesn't get flagged by Defender. Unfortunately Tentler wasn't able to get it work with Metasploit and opted for Unicorn, another penetration testing framework that behaves similarly to Metasploit, with the exception that it takes Metasploit arguments, obfuscates the payload by encoding it into a base64 string and, in the demonstration, saved it as a PowerShell file before running it on the victim machine.

I wanted to understand why Metasploit wouldn't work, as I'd already done quite a bit of research in getting myself familiarized with the framework and couldn't afford to dedicate any additional time to familiarize myself with a different framework as we had a deadline to adhere to. Since our attack vector utilizes a BadUSB device, and I wanted to restrict as much of the interaction with the victim as possible to the command prompt, I had to figure out which commands I would need to run from the terminal as initial testing kept getting flagged by AV despite having replicated the commands that Tentler had showcased to disable Defender.

This led to researching the possibility of completely removing and deleting Windows Defender from the victim machine, but this was problematic for two main reasons. First, and foremost, you can't with Windows 10. Even with a third-party vendor's AV software such as

Avast, Kaspersky or others, all that would happen is Windows Defender would disable itself so that the third-party AV would run. Second, running "sc stop windefend", which uses 'sc' a command line program used to interact with the Service Control Manager throws an access denied error message, despite running the command from administrator level privileges.

Fortunately, I was able to gain success in my efforts through PowerShell thanks to an article written by Philmore Minjentidz. In the article, Minjentidz goes through how to disable Defender using the command line, settings inside Windows and through PowerShell. While the article makes use of the "sc stop windefend" command to disable Defender, it does explicitly state that "if the service is unstoppable you will receive the [SC] OpenService FAILED 5: Access is denied. Error." (Minjentidz 2019), which we did.

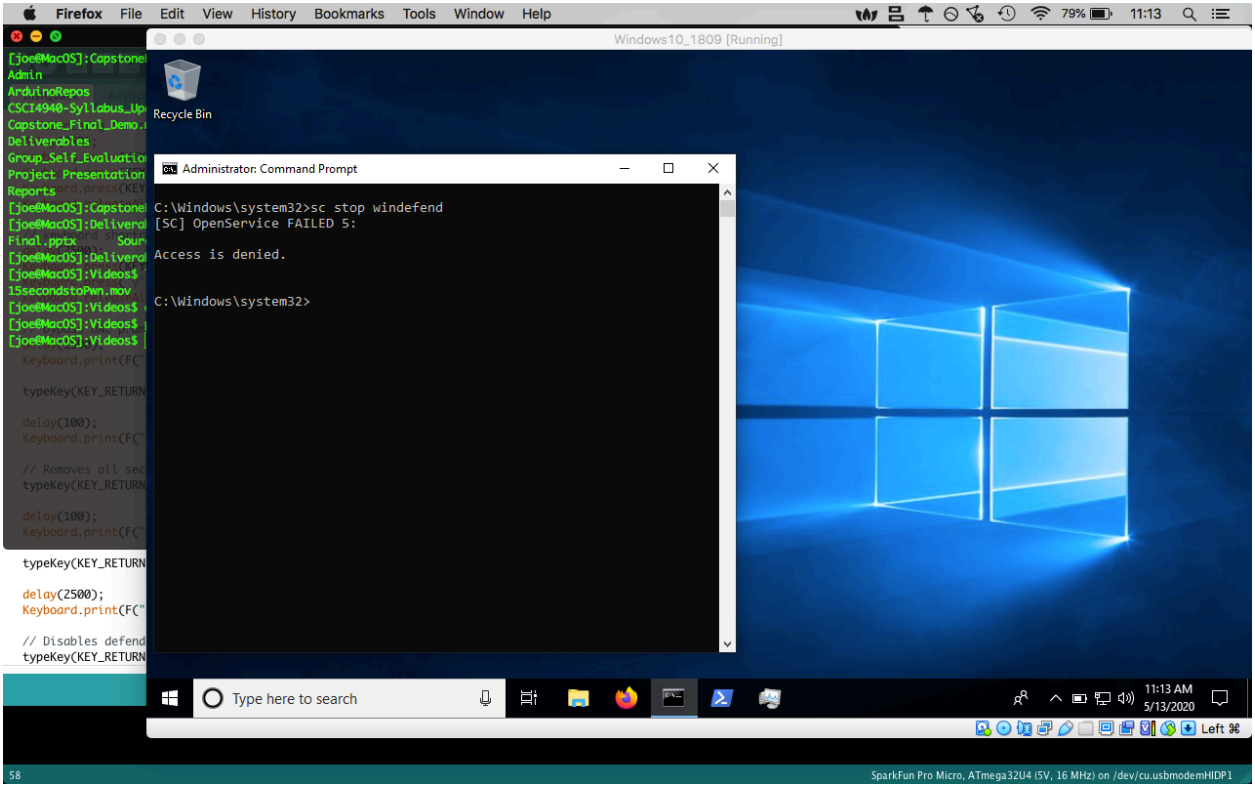


Figure 9: Attempting to disable Defender with 'sc stop windefend'

So, we've found out that we are unable to disable Defender, but Minjentidz presents us with a viable solution; disabling real-time monitoring protection.

Real-time monitoring protection is used by Defender to protect your computer against malware and other threats by incorporating signature detection and heuristics (in real-time, hence the name) against any sort of suspicious activity (Denisebmsft 2019). Which reinforces the importance of having removed the security definitions from Defender as the initial step in compromising Defender. To disable real-time monitoring protection, normally you'd navigate to the Windows Security dashboard, pull up the virus and threat protection settings and toggle the "Real-time protection" switch to the "off" position.

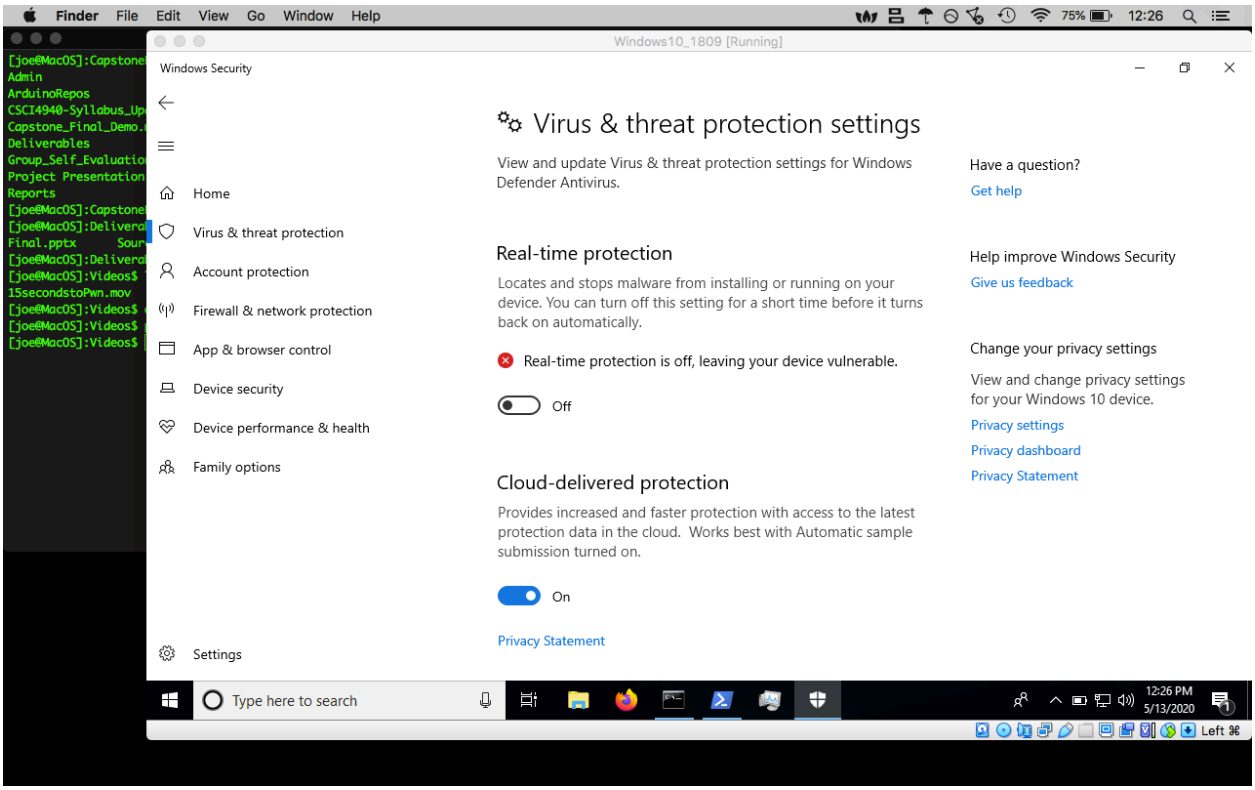
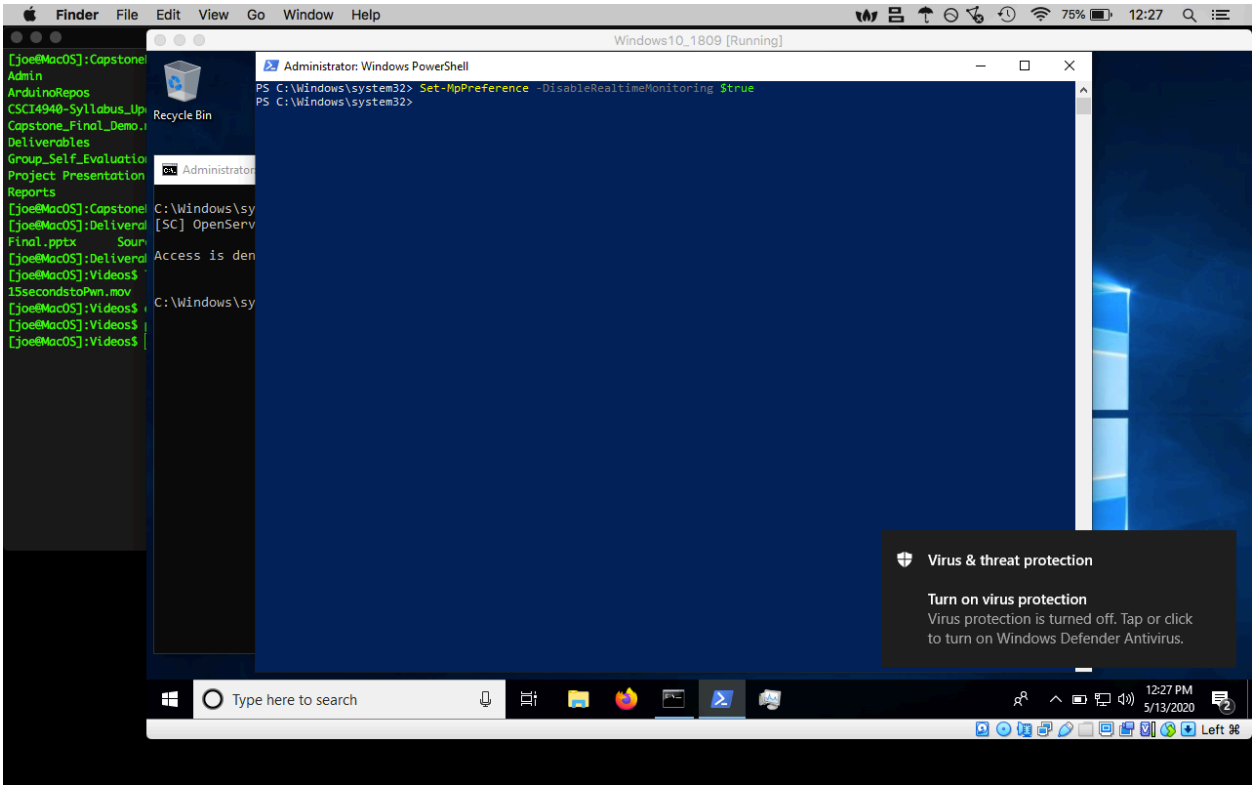


Figure 10: Locating the off switch for Real-time Protection

Again, we're trying to do as much as we can from the command prompt and PowerShell as possible. So, if we follow test the instructions to accomplish the same thing in PowerShell, we type "Set-MpPreference -DisableRealtimeMonitoring $true", the cmd-let 'Set-MpPreference' is used to set malware protection preferences in PowerShell, the preference we're changing is

‘DisableRealtimeMonitoring’ which takes a Boolean value of either true or false, the default of which is false. After running the command



we see that the system tray has alerted us that virus protection has been turned off. We confirm this by navigating to the same security settings and verifying that it is in the off position.

Next, we need to create an exclusion path telling Defender “stay out of here”, which is accomplished again with the Set-MpPreference cmd-let. This time we’re specifying an exclusion path and then declaring the path to be excluded from Defender scans and real-time monitoring. After declaring an exclusion path, the final steps are to utilize Graeber’s AMSI bypass, download the staged payload from the attacking machine’s webserver and run the file so that we call a reverse shell from the victim to the attacker. Now we need to take these

commands and program them into Arduino so we can test whether this entire process can be automated and delivered over USB.

Automating Delivery via Arduino

Personally, I (Joe) chose to write the payload for Arduino in Duckyscript, a simple scripting language created by Darren Kitchen and used to program commands to the USB Rubber Ducky (Kitchen 2017). I chose this language for its ease of use and then used a Duckyscript to Arduino converter service written by someone who goes by the username "Dukweeno" from Github.

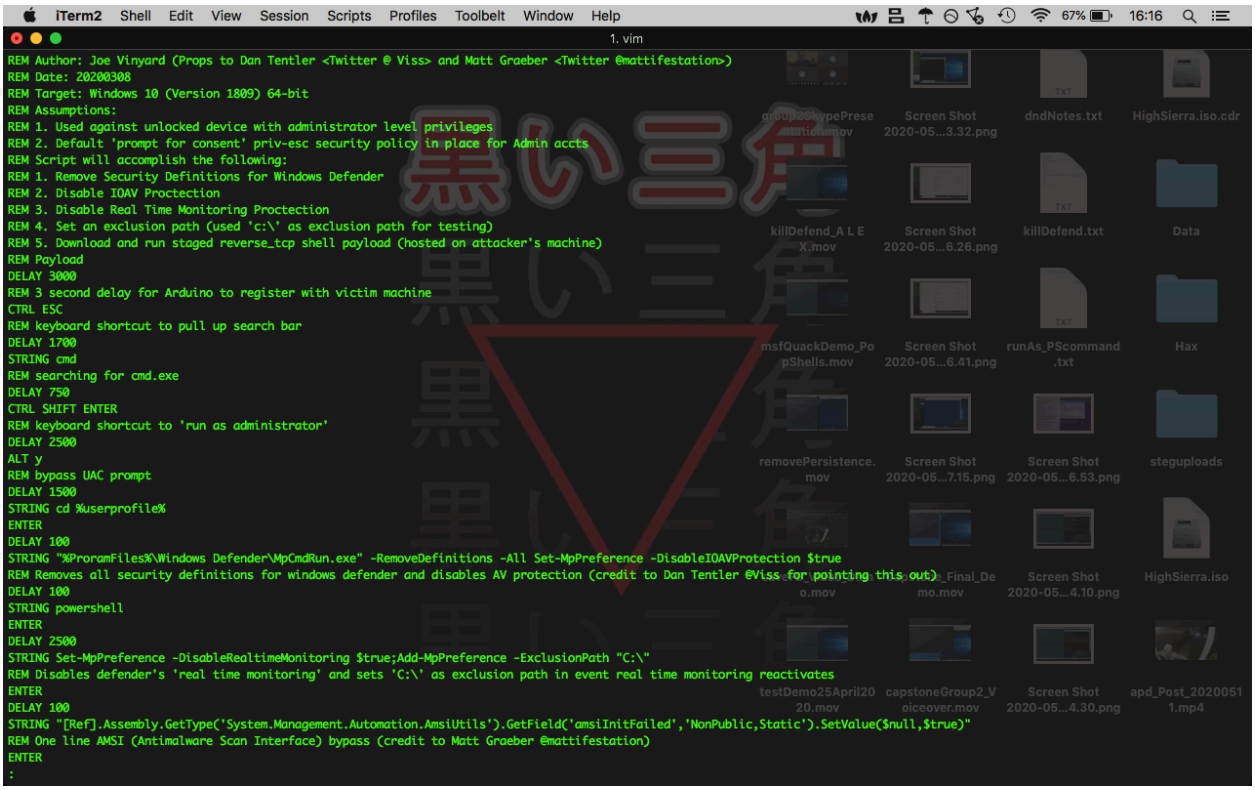


Figure 11: msfQuack.txt originally written in 'Duckyscript'

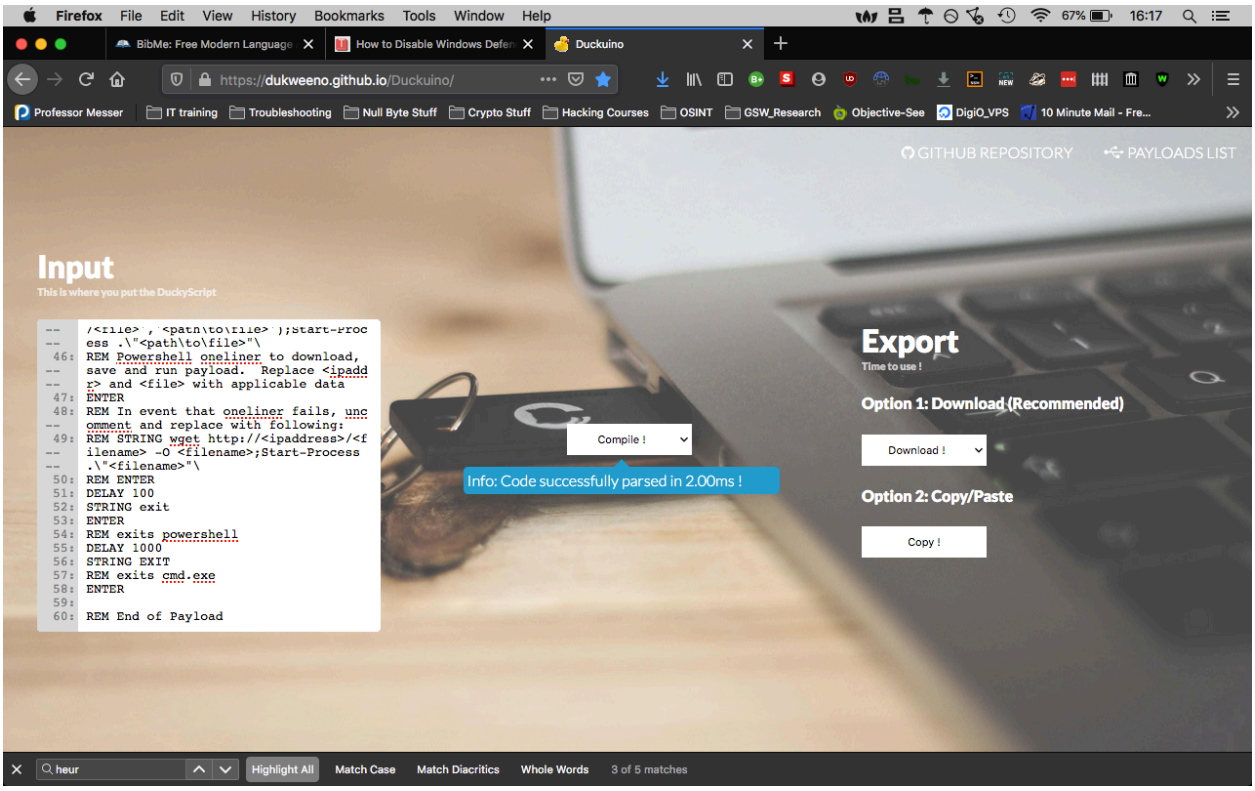


Figure 12: Duckyscript to Arduino C converter courtesy of username Dukweeno. (Source: https://dukweeno.github.io/Duckuino/)

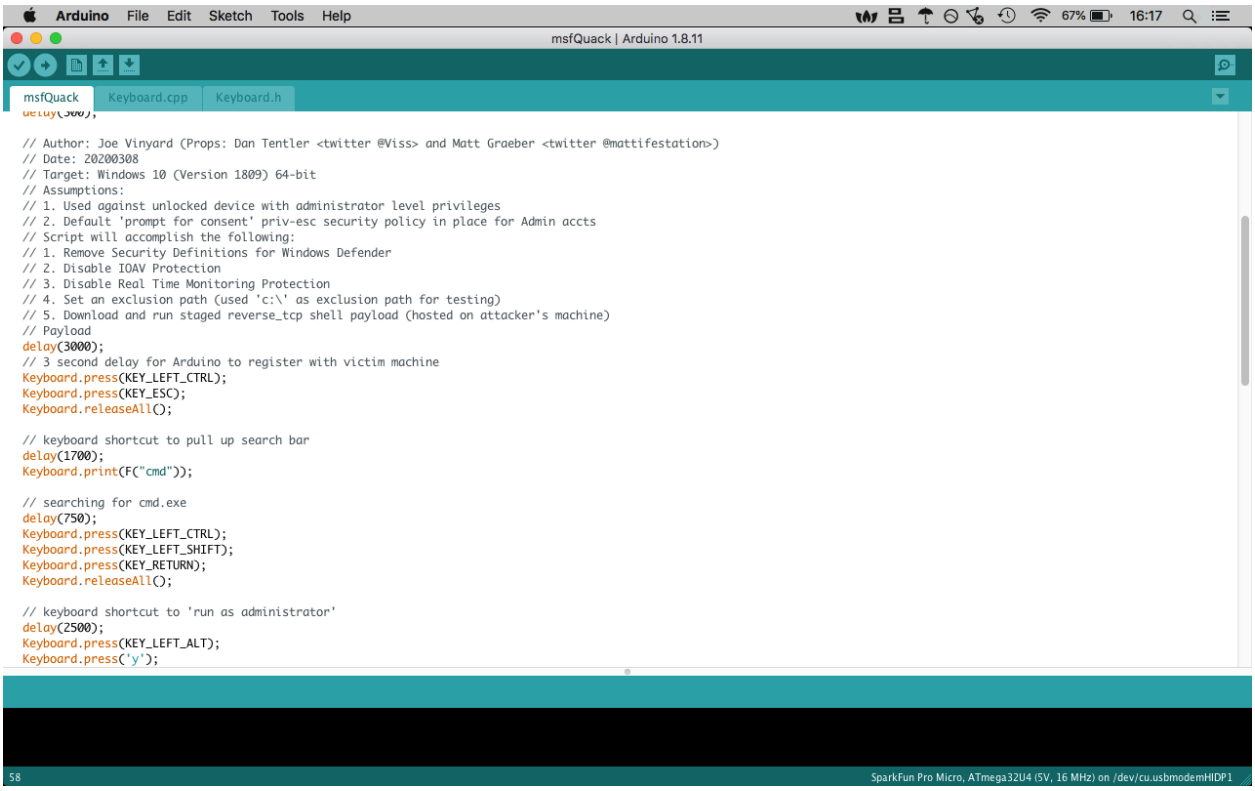


Figure 13: msfQuack.txt after being converted to Arduino

Results

A Ghost in the Shell

After much trial and error related to tweaking the delay times (the programmed delays in milliseconds between commands being typed in) I was finally able to run the msfQuack payload and create a reverse shell connection to our attacking machine, which can be seen in Figures 14 and 15 below. I've also uploaded a video demonstration of this payload being delivered to YouTube, the video can be found at the following link: https://youtu.be/RO6N1sgkqbM. The presentation and live demonstration in its entirety can be found at the following link: https://youtu.be/7zuhWVDqpdg.

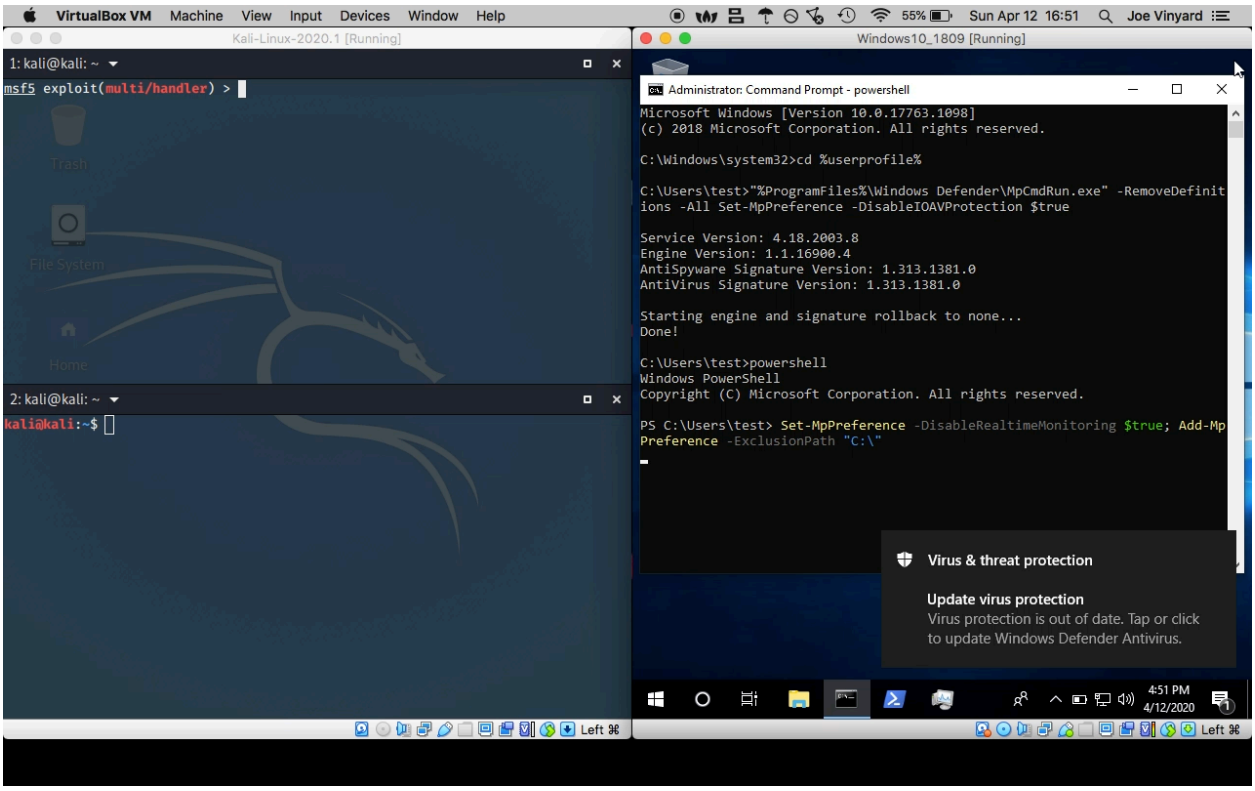


Figure 14: msfQuack payload being delivered over USB (source: Joe Vinyard)

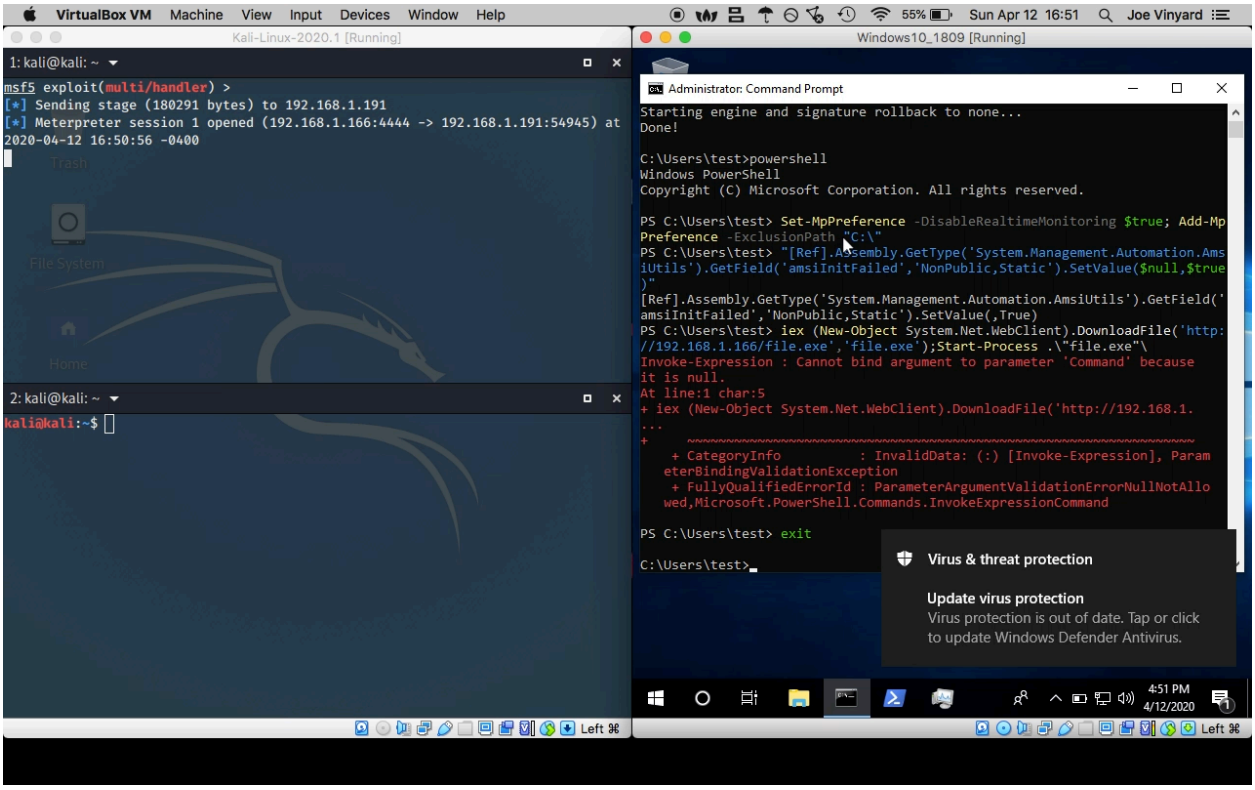


Figure 15: reverse shell connection established (source: Joe Vinyard)

I See You…

Now that we've confirmed that we can automate this attack with Arduino we need to showcase just how easy it is to move around the filesystem and accomplish our previously defined objectives. As a refresher, we wanted to see what we could do on the system if we were able to gain remote access. Now that we've established remote access, we'll show the results of attempting to

- Take screenshots of victim's desktop
- Stream the victim's webcam
- Transfer files between victim and attacking machine
- Take pictures of the user with the webcam
- Record the victim's microphone to eavesdrop on conversations

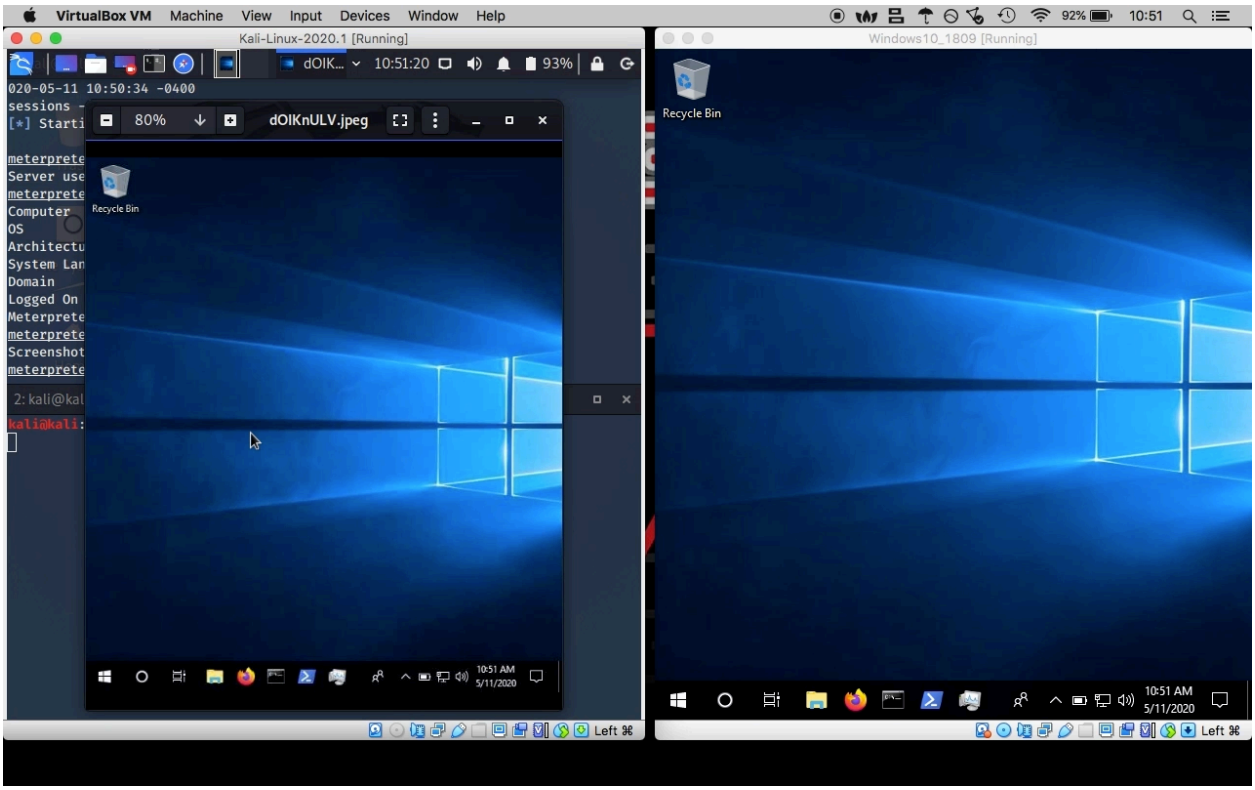


Figure 16: Screenshot (L) of victim's Desktop (R)

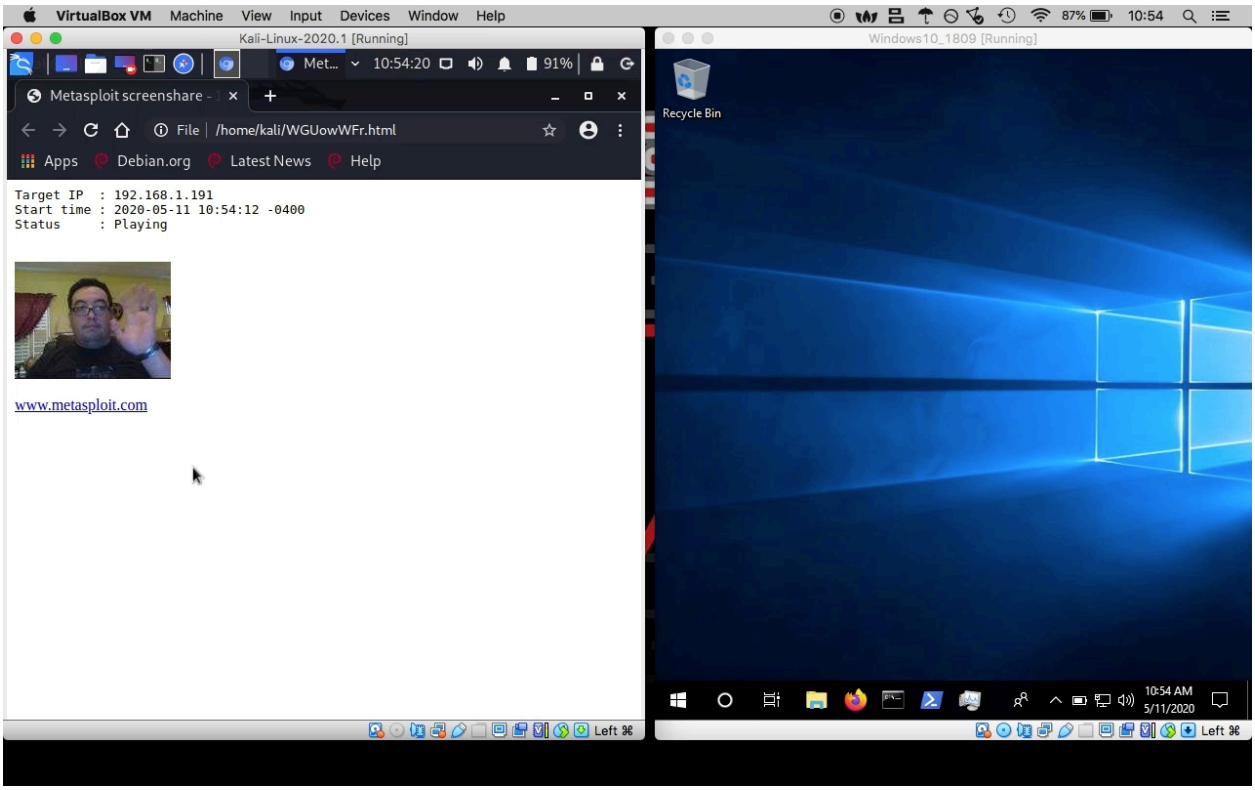


Figure 17: Joe waving "Hello" as he hacks his own webcam.

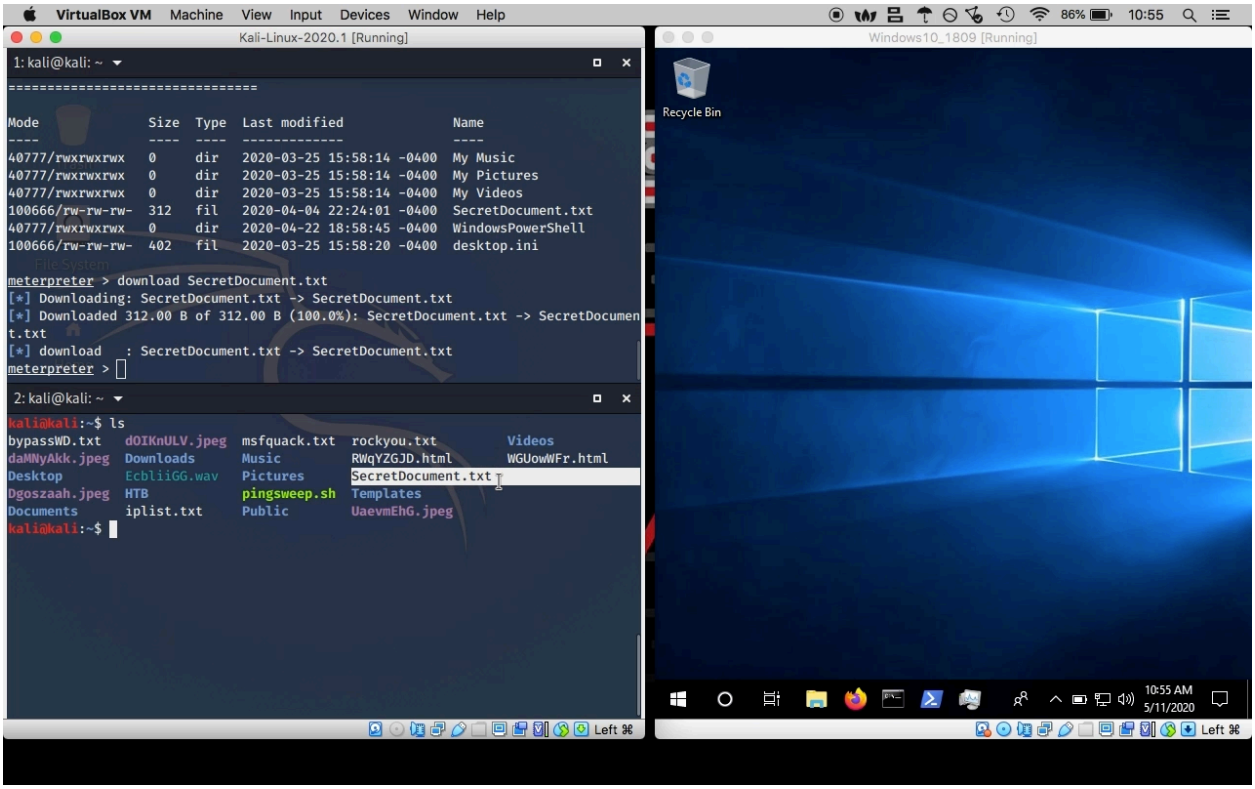


Figure 18: Downloading 'SecretDocument.txt' from victim (Upper left) and showing location on hacker's machine of downloaded file (Lower left, highlighted)

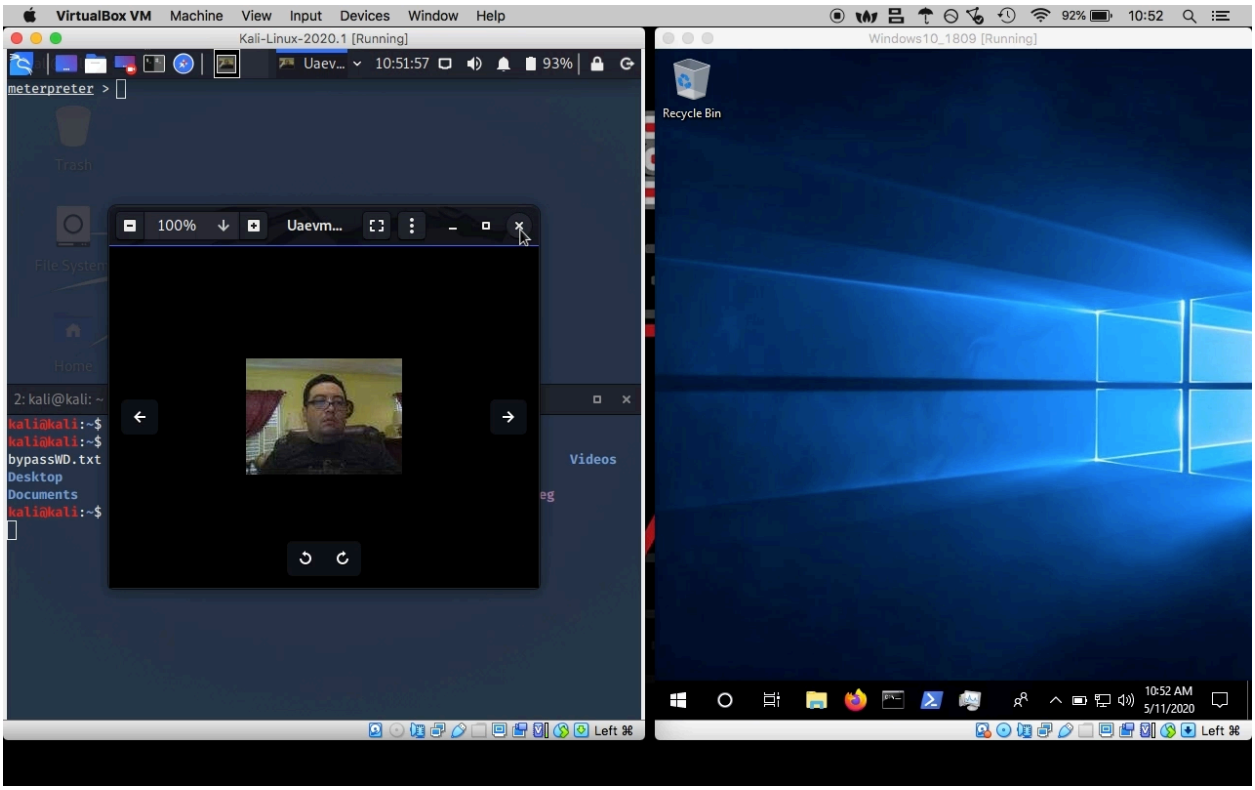


Figure 19: Taking picture with victim's webcam. At least my Mom thinks I'm handsome...

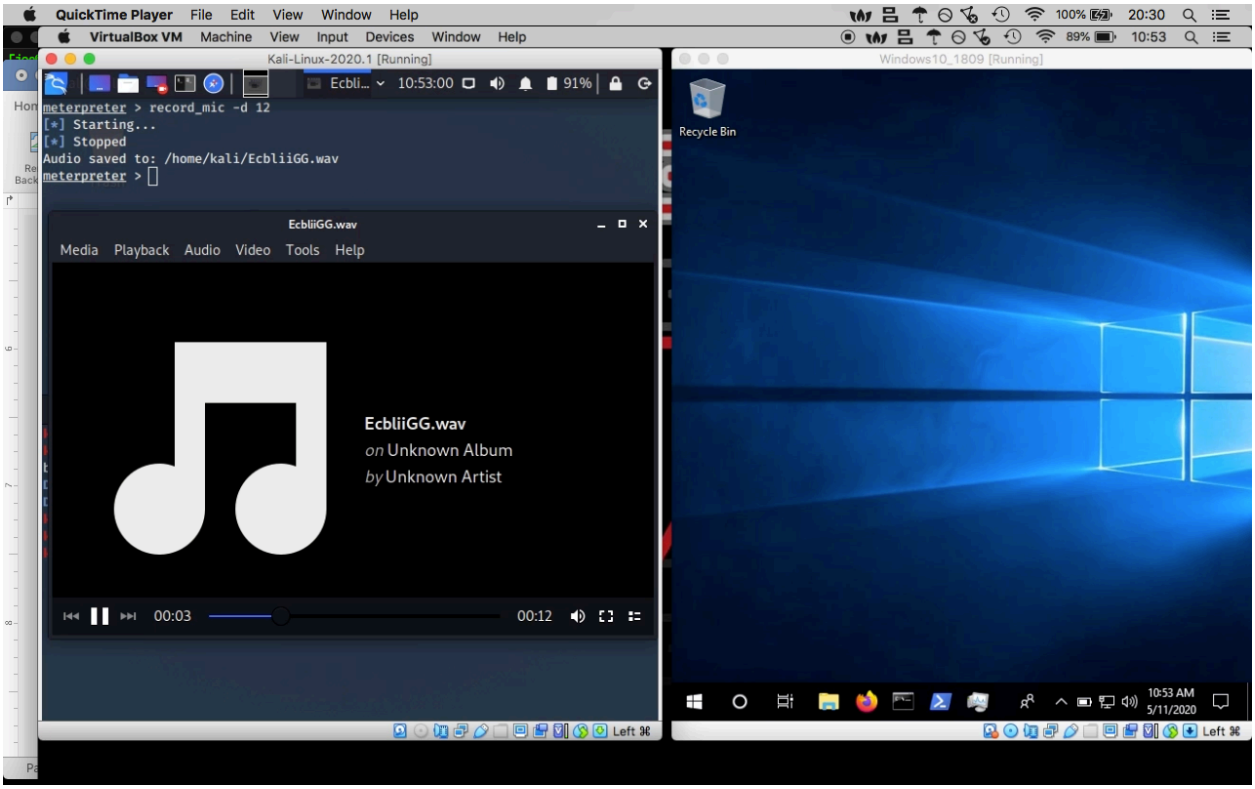


Figure 20: Playing back audio recording from victim's microphone

A Game of 'Cat and Mouse'

We (see Daniel Contreras) attempted to try and defend against this type of attack. Initially we wanted to try and create either a PowerShell or batch script to run in the background on a loop checking for whether certain elements of Defender were being disabled, and if they were, automatically re-enable those settings. However, we failed to take into consideration the resource consumption that this approach might create and weren't sure of the unintended consequences of trying to implement this as a tentative solution. Daniel attempted to write the following "windefend.ps1" script which would try and determine if Windows Defender was running by specifically checking the output of the "Get-MpComputerStatus" cmd-let in PowerShell. If the output was null or empty, the program would then run the appropriate cmd-lets to re-enable AV and real-time protection in the background. After running through the script, it would display the results of whether the defined variables were enabled.

Unfortunately, we would still get false positives as they related to real-time protection. I would manually disable the settings I wanted to test against the script (e.g. disable real-time protection), let the script run in the background and the output would state that real-time protection was *not* enabled, but when we would go into the Windows Security settings to verify, we found that the script had in fact turned it back on. From there I tried adding to the script by declaring some variables to update the security definitions for Defender as well as to select the appropriate properties from the Get-MpComputerStatus cmd-let and display their values when the script finished running and display the results. The properties I was interested in were "-DisableIOAVProtection", "-DisableRealtimeMonitoring" and "ExclusionPath". I wanted to not

only check for the existence of an exclusion path, but *if* one did exist, I wanted to know *where* on the system the exclusion path existed.

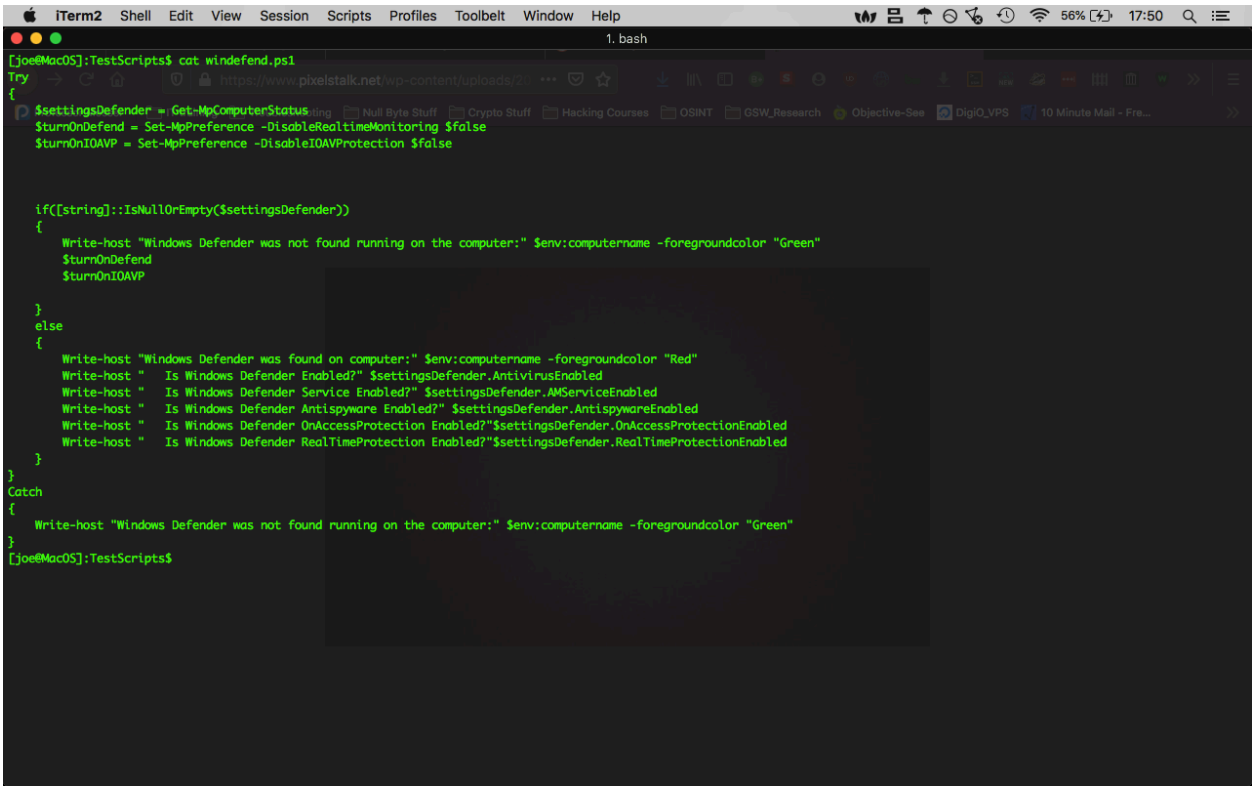
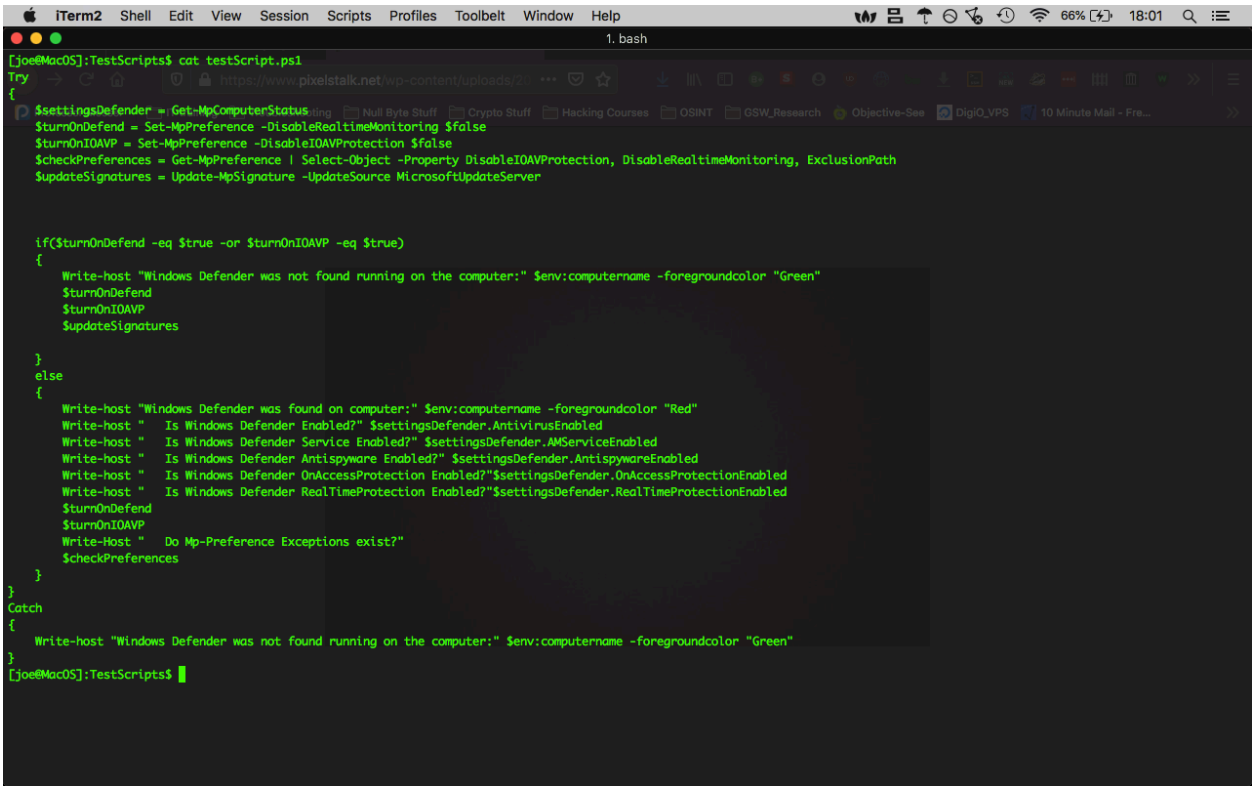


Figure 21: windefend.ps1 above (Source: Daniel Contreras) and testScript.ps1 below (Source: Joe Vinyard)


iTerm2 Shell Edit View Session Scripts Profiles Toolbelt Window Help
66% 18:01
1. bash
[joe@MacOS]:TestScripts$ cat testScript.ps1
Try
{
$settingsDefender = Get-MpComputerStatus
$turnOnDefend = Set-MpPreference -DisableRealtimeMonitoring $false
$turnOnIOAVP = Set-MpPreference -DisableIOAVProtection $false
$checkPreferences = Get-MpPreference | Select-Object -Property DisableIOAVProtection, DisableRealtimeMonitoring, ExclusionPath
$updateSignatures = Update-MpSignature -UpdateSource MicrosoftUpdateServer
if($turnOnDefend -eq $true -or $turnOnIOAVP -eq $true)
{
Write-host "Windows Defender was not found running on the computer:" $env:computername -foregroundcolor "Green"
$turnOnDefend
$turnOnIOAVP
$updateSignatures
}
else
{
Write-host "Windows Defender was found on computer:" $env:computername -foregroundcolor "Red"
Write-host " Is Windows Defender Enabled?" $settingsDefender.AntivirusEnabled
Write-host " Is Windows Defender Service Enabled?" $settingsDefender.AMServiceEnabled
Write-host " Is Windows Defender Antispyware Enabled?" $settingsDefender.AntispywareEnabled
Write-host " Is Windows Defender OnAccessProtection Enabled?"$settingsDefender.OnAccessProtectionEnabled
Write-host " Is Windows Defender RealTimeProtection Enabled?"$settingsDefender.RealTimeProtectionEnabled
$turnOnDefend
$turnOnIOAVP
Write-Host " Do Mp-Preference Exceptions exist?"
$checkPreferences
}
}
Catch
{
Write-host "Windows Defender was not found running on the computer:" $env:computername -foregroundcolor "Green"
}
[joe@MacOS]:TestScripts$

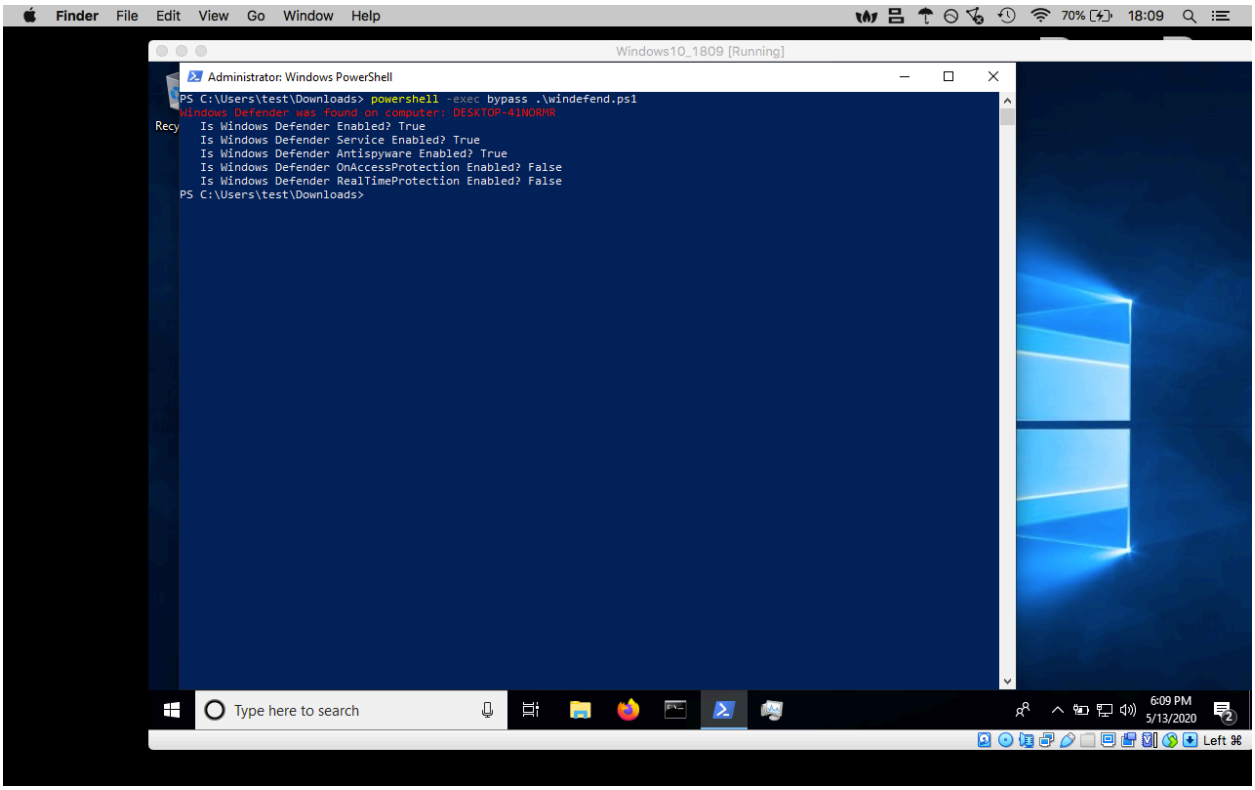
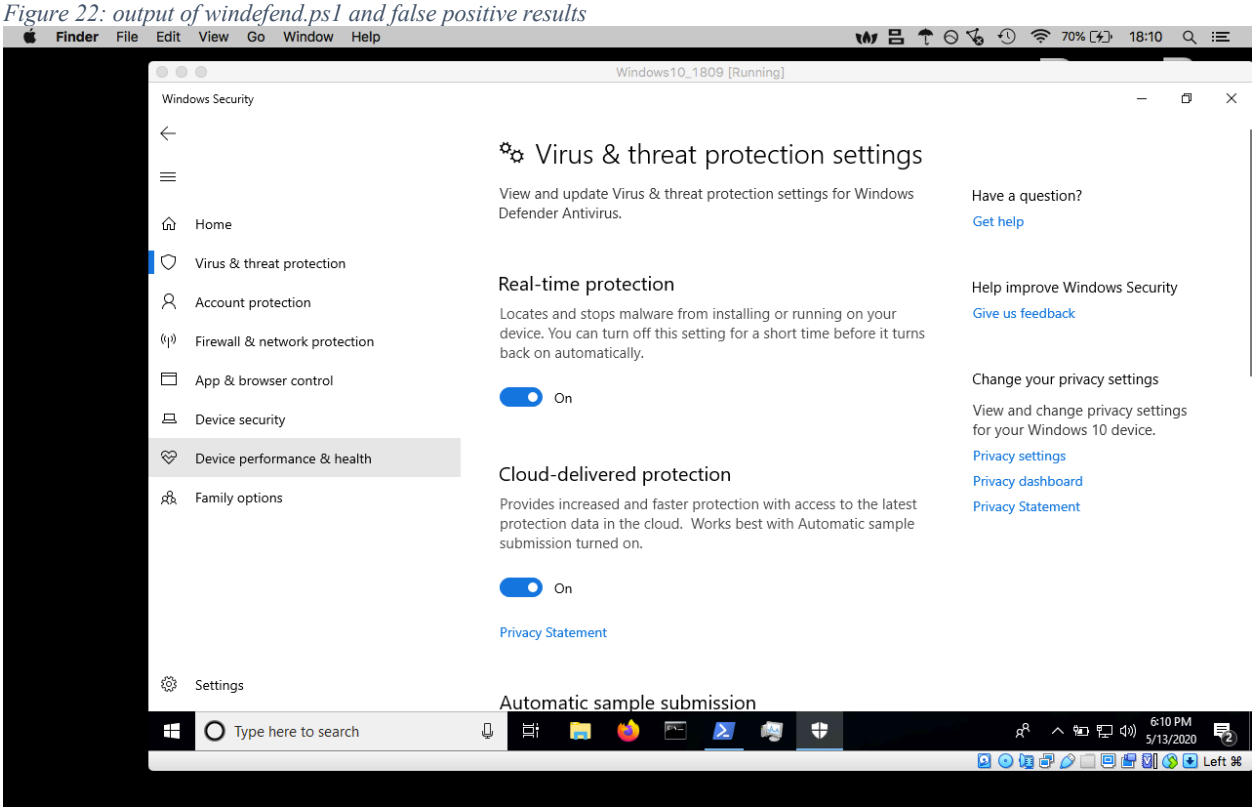


Figure 22: output of windefend.ps1 and false positive results



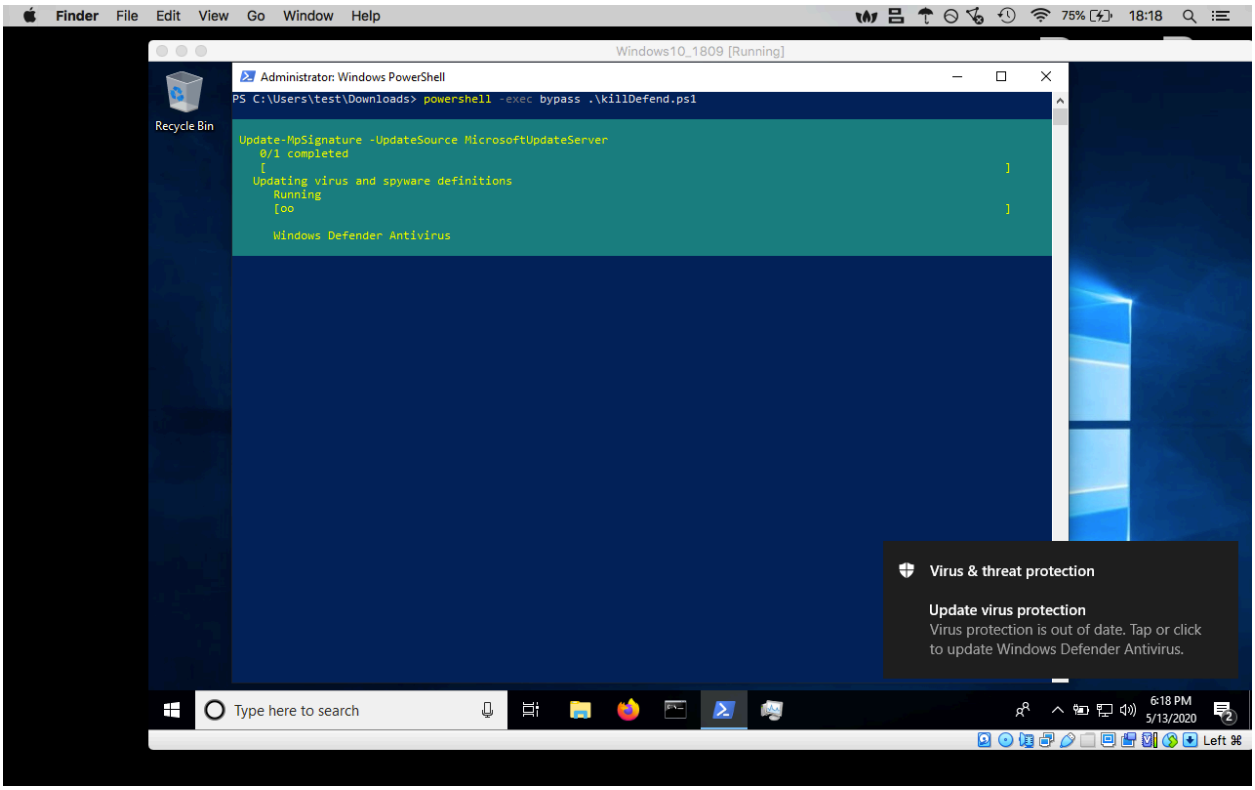
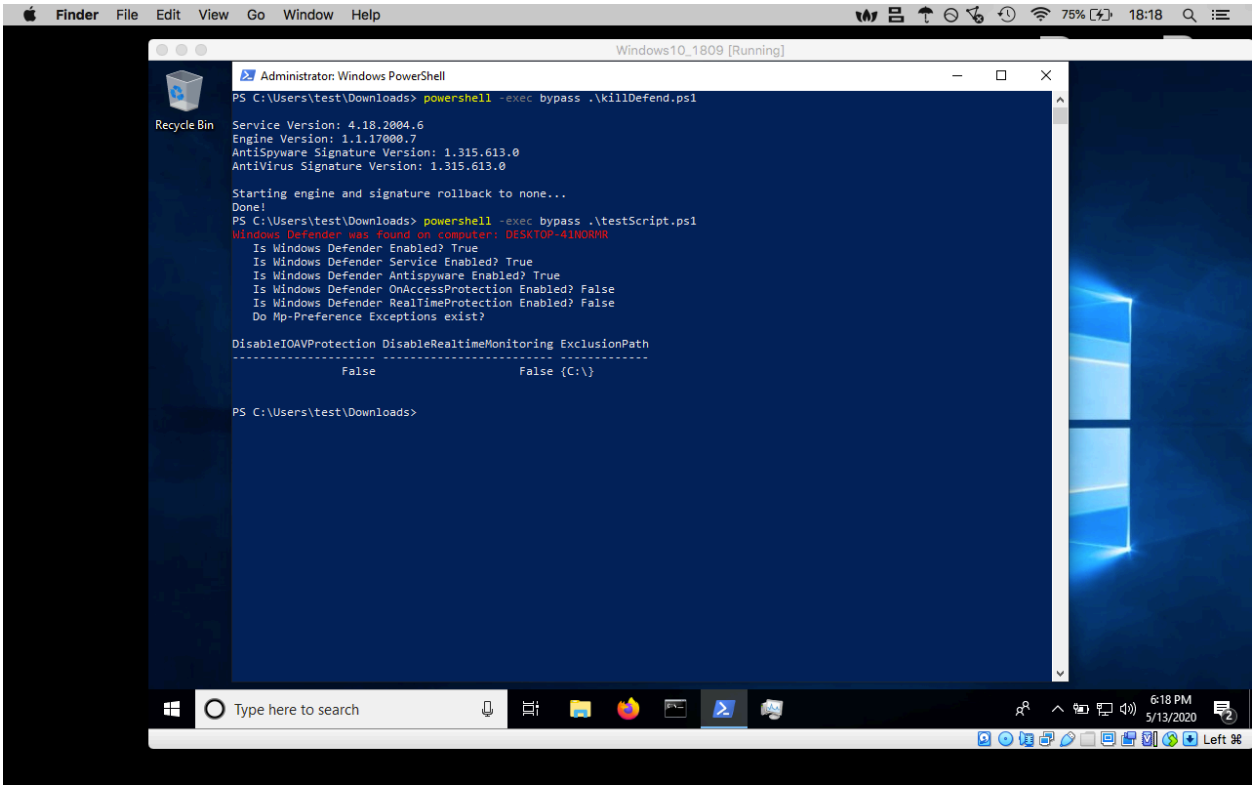


Figure 23: running testScript.ps1



In the preceding screenshots depicted by Figure 18, we see the results of the output for testScript.ps1. We still end up with a false positive result for “Is Windows Defender OnAccessProtection Enabled? False” and “Is Windows Defender RealtimeProtection Enabled? False”. However, with the added variable to grab and display the properties “DisableIOAVProtection, DisableRealtimeMonitoring” and “ExclusionPath”, we can see that the former is in fact not disabled and we now know the location of any exclusion paths on the system. Unfortunately, we would run out of time before needed to present our findings, and as such we weren’t able to develop a functioning solution.

Conclusion

Keep it simple, stupid

We’d previously changed the default behavior as it pertains to privilege escalation (Priv-Esc). By default, the initial local account that gets created when you first install Windows is a local administrator. This is important because the default User Account Control (UAC) behavior, which is the prompt you see to allow an application or program to make changes to

your computer, is to simply “prompt for consent” as we see here.

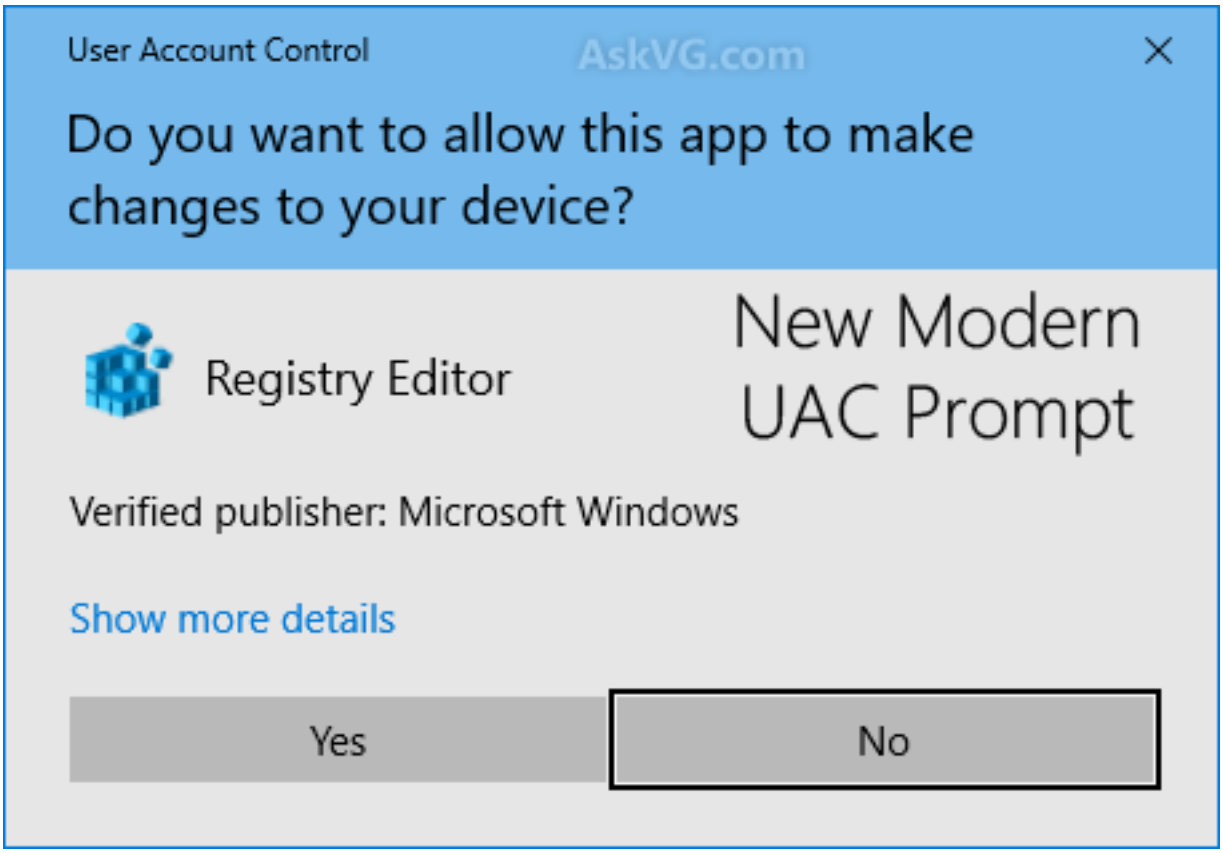


Figure 24: UAC prompt (source: https://www.askvg.com/tip-disable-modern-uac-prompt-and-credential-ui-in-windows-10/)

This is an important feature to take into consideration when it comes to the subject of trying to harden a system. The default prompt for administrator accounts is to simply prompt for consent whereas the default UAC behavior for standard users (non-administrator accounts) is to “prompt

for credentials” as shown below.

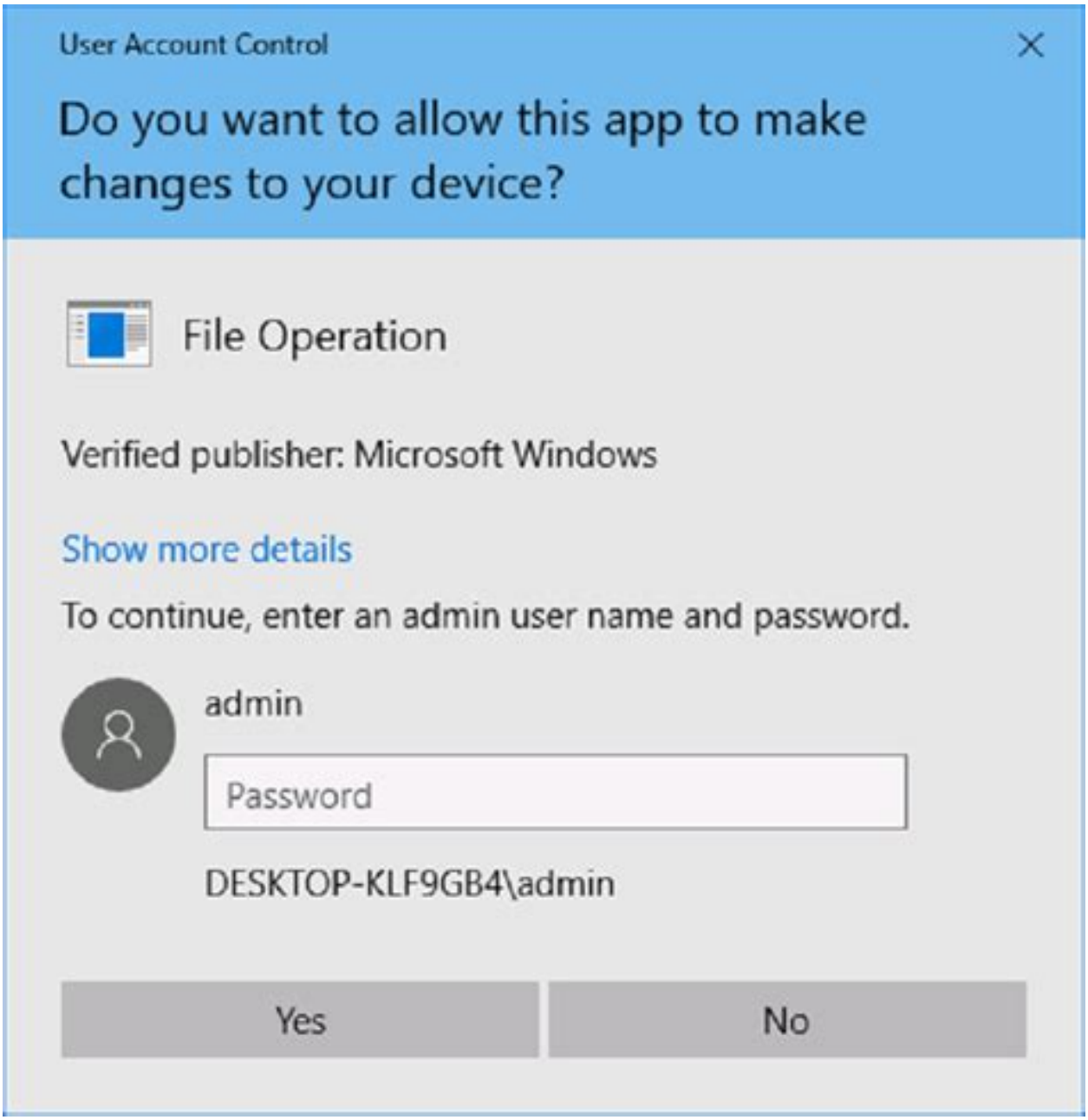


Figure 25: UAC prompt for credentials (source: https://windowsreport.com/anniversary-update-revamped-uac-prompt/)

We’ve found that changing the default behavior for administrator accounts from “prompt for consent” to “prompt for credentials” and changing the default behavior for non-administrator accounts from “prompt for credentials” to “Automatically deny elevation requests” is the best

method for defense as it pertains to *this* specific attack since the attack makes use of bypassing the UAC "prompt for consent" by inserting the keystrokes ALT+y to select the "yes" box on the UAC prompt before continuing with the script.

If, for an administrator's account, we implement the changes, the Arduino script will not be aware that the default behavior has been changed and type out the rest of the commands to disable Defender in the textbox where an administrator's credentials are to be entered prior to completing Priv-Esc attempt. For a standard account, the Priv-Esc attempt will be denied automatically, and the keystroke commands will be entered in the abyss.

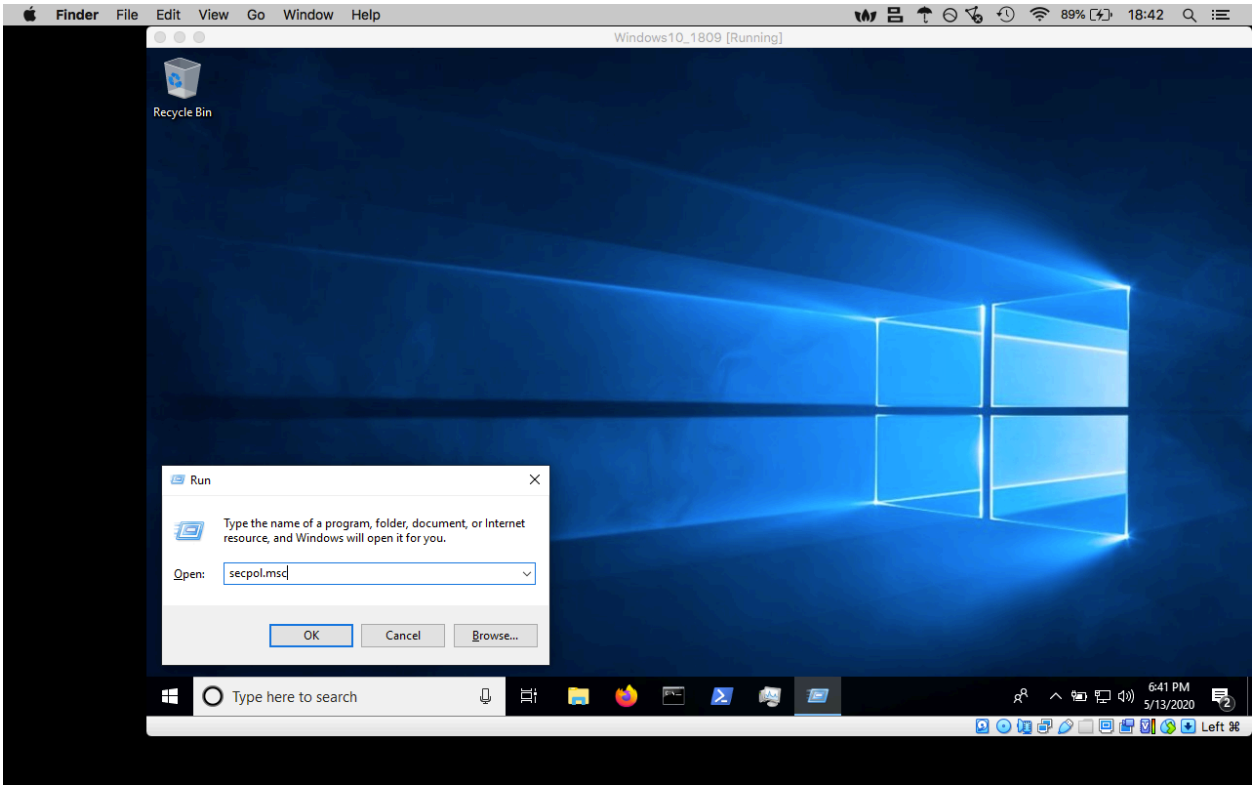
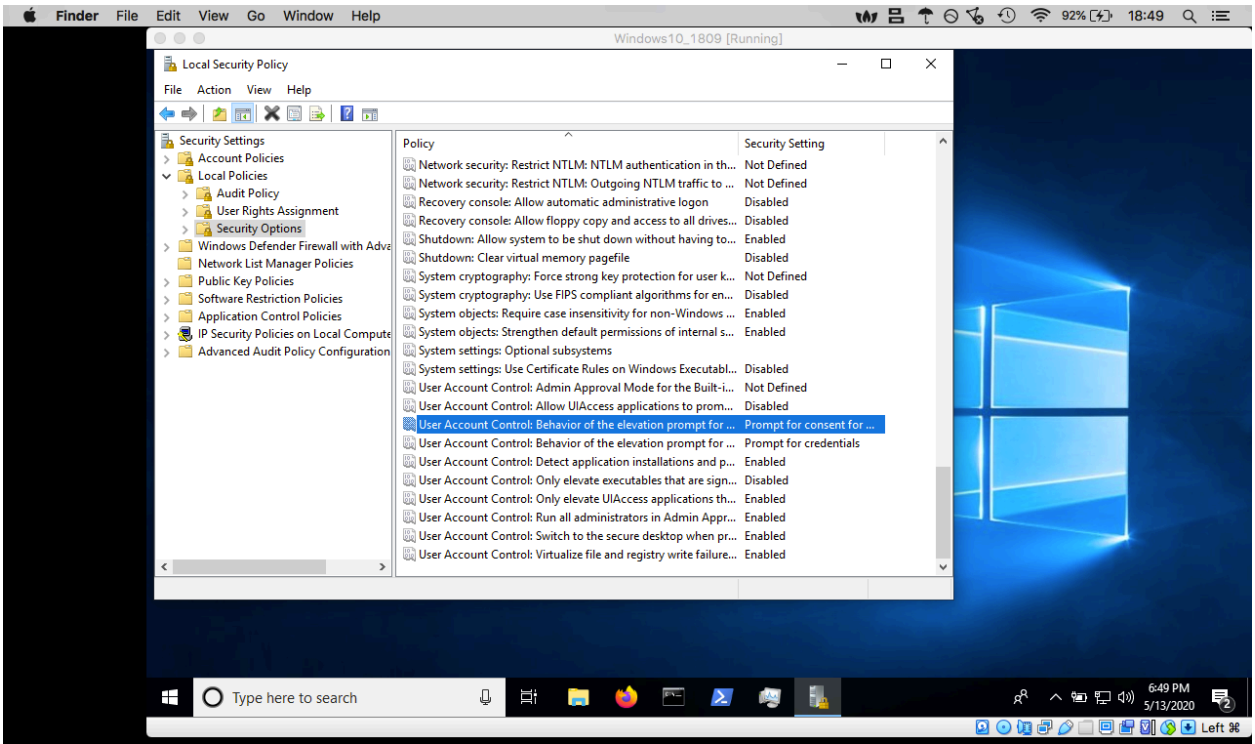
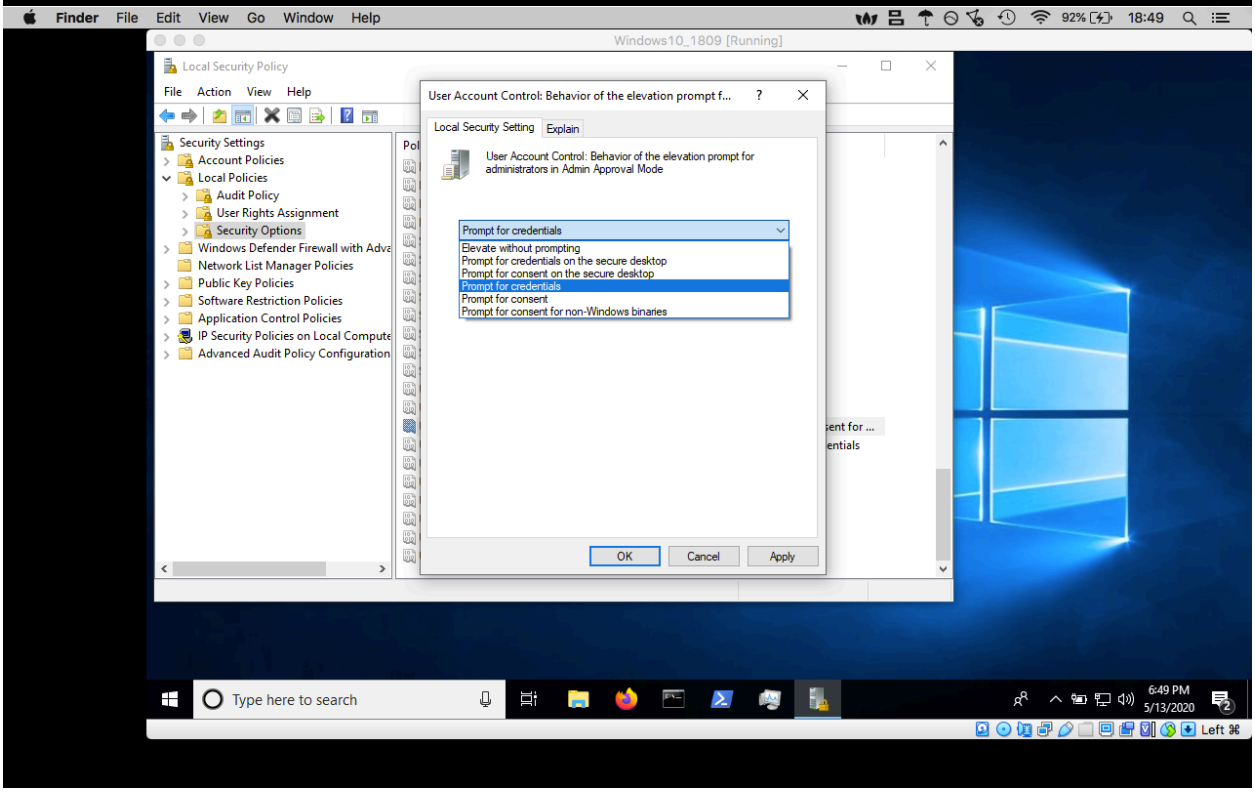
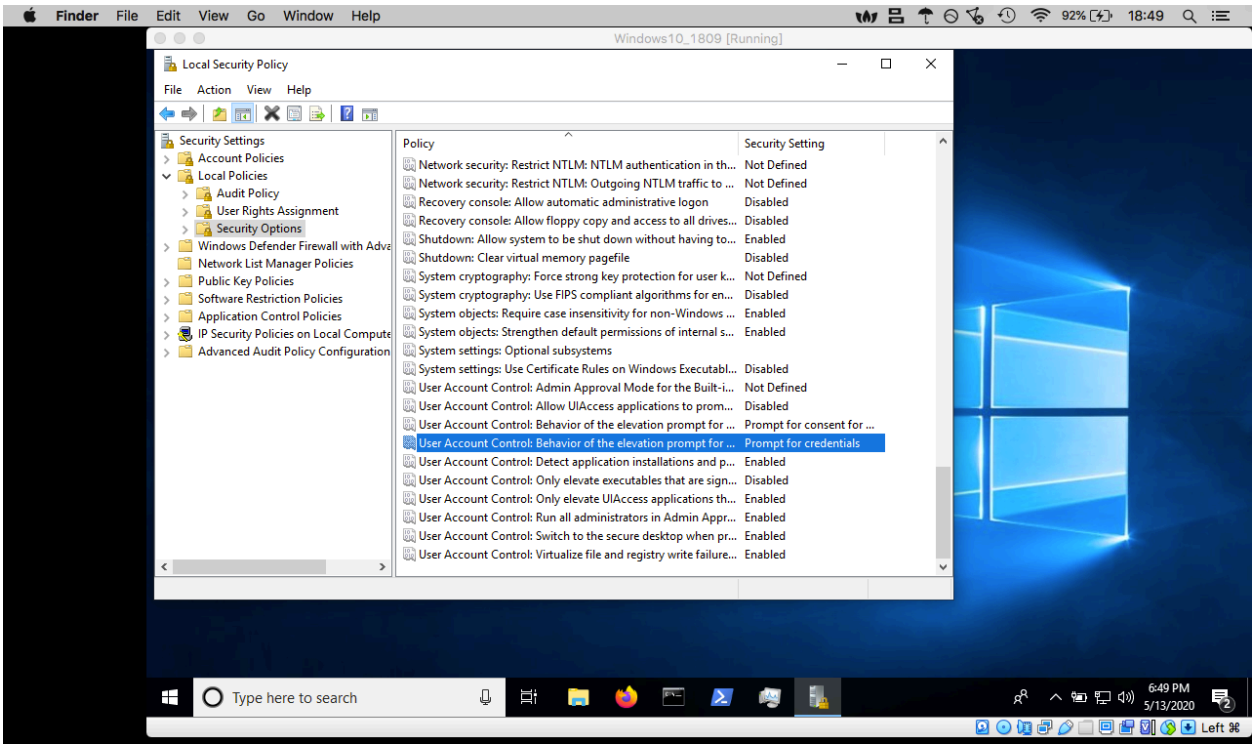


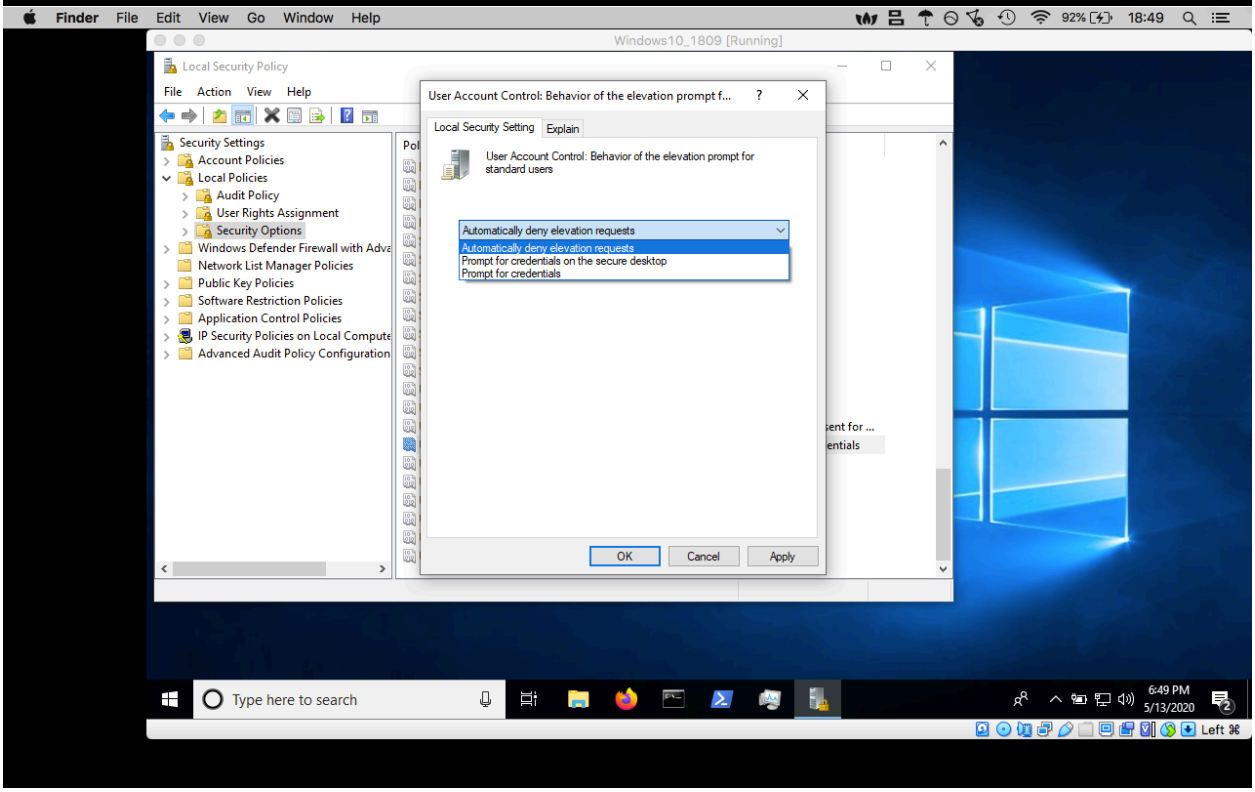
Figure 26: Using the 'Run' prompt to navigate to the security policy settings

Figure 21 and the following screen shots show how to navigate to the security policy settings (Enterprise, Pro and Education versions of Windows only) to harden the default UAC behavior for administrators and standard accounts.


Finder File Edit View Go Window Help
92% 18:49
Windows10_1809 [Running]
Local Security Policy
File Action View Help
Security Settings
Account Policies
Local Policies
Audit Policy
User Rights Assignment
Security Options
Windows Defender Firewall with Adva
Network List Manager Policies
Public Key Policies
Software Restriction Policies
Application Control Policies
IP Security Policies on Local Compute
Advanced Audit Policy Configuration
Policy
Security Setting
Network security: Restrict NTLM: NTLM authentication in th... Not Defined
Network security: Restrict NTLM: Outgoing NTLM traffic to ... Not Defined
Recovery console: Allow automatic administrative logon Disabled
Recovery console: Allow floppy copy and access to all drives... Disabled
Shutdown: Allow system to be shut down without having to... Enabled
Shutdown: Clear virtual memory pagefile Disabled
System cryptography: Force strong key protection for user k... Not Defined
System cryptography: Use FIPS compliant algorithms for en... Disabled
System objects: Require case insensitivity for non-Windows ... Enabled
System objects: Strengthen default permissions of internal s... Enabled
System settings: Optional subsystems
System settings: Use Certificate Rules on Windows Executabl... Disabled
User Account Control: Admin Approval Mode for the Built-i... Not Defined
User Account Control: Allow UIAccess applications to prom... Disabled
User Account Control: Behavior of the elevation prompt for ... Prompt for consent for ...
User Account Control: Behavior of the elevation prompt for ... Prompt for credentials
User Account Control: Detect application installations and p... Enabled
User Account Control: Only elevate executables that are sign... Disabled
User Account Control: Only elevate UIAccess applications th... Enabled
User Account Control: Run all administrators in Admin Appr... Enabled
User Account Control: Switch to the secure desktop when pr... Enabled
User Account Control: Virtualize file and registry write failure... Enabled
Type here to search
6:49 PM
5/13/2020
Left ⌘


Finder File Edit View Go Window Help
92% 18:49
Windows10_1809 [Running]
Local Security Policy
File Action View Help
User Account Control: Behavior of the elevation prompt f...
Local Security Setting
Explain
User Account Control: Behavior of the elevation prompt for administrators in Admin Approval Mode
Prompt for credentials
Elevate without prompting
Prompt for credentials on the secure desktop
Prompt for consent on the secure desktop
Prompt for credentials
Prompt for consent
Prompt for consent for non-Windows binaries
OK
Cancel
Apply
Type here to search
6:49 PM
5/13/2020
Left ⌘


Finder File Edit View Go Window Help
92% 18:49
Windows10_1809 [Running]
Local Security Policy
File Action View Help
Security Settings
Account Policies
Local Policies
Audit Policy
User Rights Assignment
Security Options
Windows Defender Firewall with Adva
Network List Manager Policies
Public Key Policies
Software Restriction Policies
Application Control Policies
IP Security Policies on Local Compute
Advanced Audit Policy Configuration
Policy
Security Setting
Network security: Restrict NTLM: NTLM authentication in th... Not Defined
Network security: Restrict NTLM: Outgoing NTLM traffic to ... Not Defined
Recovery console: Allow automatic administrative logon Disabled
Recovery console: Allow floppy copy and access to all drives... Disabled
Shutdown: Allow system to be shut down without having to... Enabled
Shutdown: Clear virtual memory pagefile Disabled
System cryptography: Force strong key protection for user k... Not Defined
System cryptography: Use FIPS compliant algorithms for en... Disabled
System objects: Require case insensitivity for non-Windows ... Enabled
System objects: Strengthen default permissions of internal s... Enabled
System settings: Optional subsystems
System settings: Use Certificate Rules on Windows Executabl... Disabled
User Account Control: Admin Approval Mode for the Built-i... Not Defined
User Account Control: Allow UIAccess applications to prom... Disabled
User Account Control: Behavior of the elevation prompt for ... Prompt for consent for ...
User Account Control: Behavior of the elevation prompt for ... Prompt for credentials
User Account Control: Detect application installations and p... Enabled
User Account Control: Only elevate executables that are sign... Disabled
User Account Control: Only elevate UIAccess applications th... Enabled
User Account Control: Run all administrators in Admin Appr... Enabled
User Account Control: Switch to the secure desktop when pr... Enabled
User Account Control: Virtualize file and registry write failure... Enabled
Type here to search
6:49 PM
5/13/2020
Left ⌘


Finder File Edit View Go Window Help
92% 18:49
Windows10_1809 [Running]
Local Security Policy
File Action View Help
User Account Control: Behavior of the elevation prompt f...
Local Security Setting
Explain
User Account Control: Behavior of the elevation prompt for standard users
Automatically deny elevation requests
Automatically deny elevation requests
Prompt for credentials on the secure desktop
Prompt for credentials
OK
Cancel
Apply
Type here to search
6:49 PM
5/13/2020
Left ⌘

Parting thoughts and future work

While the BadUSB attack has been around for some time, there is no fix for it. However, if we can incorporate some pragmatic solutions to make an attempt to at least harden our systems, we may be able to defend against PowerShell attacks implementing some of the advice given by Lee Holmes. Holmes, a security researcher and developer for PowerShell, recommends incorporating a slew of things from enabling and collecting PowerShell logs to implementing "just enough" administration (Holmes 2017) in order to defend against PowerShell attacks. Other viable solutions exist as well. Unfortunately, we weren't able to incorporate these recommendations as we ran out of time, but future work will incorporate our proposed solution, Holmes' advice and recommendations such as incorporating "constrained language mode" which blocks malicious running scripts in PowerShell by limiting the *types* of commands that can be executed (Chettitar 2019).

Github repository (for Arduino sketches)

For source code on this project and instructions on how to set up your testing environment with Arduino, please see the following link: https://github.com/jvin607/Capstone_Spring2020

References

Admin, and Adam Chester. “Exploring PowerShell AMSI and Logging Evasion.” *MDSec*, 18 June 2018, www.mdsec.co.uk/2018/06/exploring-powershell-amsi-and-logging-evasion/.

Chettiar, Rohit. “How to Prevent and Detect Malicious PowerShell Attacks.” *Rapid7 Blog*, Rapid7 Blog, 8 Aug. 2019, blog.rapid7.com/2019/08/08/the-importance-of-preventing-and-detecting-malicious-powershell-attacks/.

Denisebmsft. “Enable and Configure Windows Defender Antivirus Protection Capabilities - Windows Security.” *Enable and Configure Windows Defender Antivirus Protection Capabilities - Windows Security | Microsoft Docs*, 16 Dec. 2019, docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-antivirus/configure-real-time-protection-windows-defender-antivirus.

Dukweeno. “Dukweeno/Duckuino.” *GitHub*, 27 Oct. 2019, github.com/Dukweeno/Duckuino.

Graeber, Matt. “AMSI Bypass in a Single Tweet. :).” *Twitter*, Twitter, 25 May 2016, twitter.com/mattifestation/status/735261176745988096.

Greenberg, Andy. “Why the Security of USB Is Fundamentally Broken.” *Wired*, Conde Nast, 3 June 2017, www.wired.com/2014/07/usb-security/.

Holmes, Lee. “Defending Against PowerShell Attacks.” *Microsoft | Powershell*, 23 Oct. 2017, devblogs.microsoft.com/powershell/defending-against-powershell-attacks/.

Jimblom. “Pro Micro & Fio V3 Hookup Guide.” *Sparkfun.com*, Sparkfun Electronics, 8 Nov. 2013, learn.sparkfun.com/tutorials/pro-micro--fio-v3-hookup-guide/installing-windows.

Jimblom. “Pro Micro & Fio V3 Hookup Guide.” *Sparkfun.com*, Sparkfun Electronics, 8 Nov. 2013, learn.sparkfun.com/tutorials/pro-micro--fio-v3-hookup-guide/installing-mac--linux.

Kitchen, Darren. “hak5darren/USB-Rubber-Ducky.” *GitHub*, 1 Oct. 2017, github.com/hak5darren/USB-Rubber-Ducky/wiki/Duckyscript.

Kitchen, Darren and Tentler, Dan “Disabling Defender with Viss Episode 1 - Hak5 2416.” Hak5, 27 June 2018, https://youtu.be/dKUS26BlKlc.

Kali.org. “Kali Linux 2020.1a Release.” *Kali Linux*, 3 Mar. 2020, www.kali.org/releases/kali-linux-2020-1a-release/.

Minjentidz, Philmore. “How to Disable Windows Defender Using PowerShell, Command Line?” *Techno Result*, 10 Dec. 2019, technoresult.com/how-to-disable-windows-defender-using-powershell-command-line/.

Phobos Group. “Results Matter.” *Phobos Group*, phobos.io/index.html#about.

Rapid7. “Metasploit Framework-Package Description.” *Kali Tools*, tools.kali.org/exploitation-tools/metasploit-framework.

rapid7. “How to Use a Reverse Shell in Metasploit.” Edited by James Lee, *GitHub*, 24 Feb. 2017, github.com/rapid7/metasploit-framework/wiki/How-to-use-a-reverse-shell-in-Metasploit.

rapid7. “rapid7/Metasploit-Framework.” *GitHub*, 11 May 2020, github.com/rapid7/metasploit-framework.

Richiewijaya. “Bind Shell vs Reverse Shell.” *Irichmore*, 4 June 2015, irichmore.wordpress.com/2015/06/04/bind-shell-vs-reverse-shell/.