**Using Captcha Style Challenges to Defend Against BadUSB Attacks**

Alexander Bladow, Jensen Miller, Mitchell White

# **ABSTRACT**

In modern computing there are plenty of security measures that are in place. For data, there is encryption, for your network there are firewalls, and for your hardware there are options such as TPM (Trusted Platform Module). These all are handled in the background for most users so they have the peace of mind that they will remain safe if they take a risky click or wind up on the wrong website. However, one long time security flaw is with devices that we plug in to our computer.

This flaw is that HID devices are innately trusted devices by a computer. Leveraging this innate trust, attackers were able to craft devices that pretend to be HID devices and deliver pre-programmed payloads, often by autonomous typing. To address this security flaw, we have developed a tool that automatically detects device changes and actively attempts to defeat common USB security threats.

The solution we have crafted named BUCS (BadUSB Challenge System) is a software challenge that requires a live user to type in a random collection of dictionary words to prove that the device being plugged in is a legitimate HID device. To support our program, we have also created a set of test BadUSBs to test against.

# **1. INTRODUCTION**

Human interface devices (HID) such as mice and keyboards have been standardized to allow plug and play usage across most if not all computer platforms. This allows for any keyboard or mouse to be plugged into any system that has a corresponding port and be used with the system. HID devices are innately trusted input due to the fact that it is the way that a human user can interact with their system. Enter the category of device known as “BadUSB,” a small device, typically with a USB interface, that can look like any number of non-malicious items. These small items could be left in public places for victims to pick up and unknowingly insert into their devices in an attempt to return them to their owners or claim for themselves. They are a type of attack that preys on human curiosity or goodwill to attempt to deliver a malware payload. Flash drives, power banks, or wireless adapters just to name a few potential forms, all of them could instead be loaded with malicious hardware that masquerades as an HID device to inject keystrokes or other data into the system.

This injection can be very dangerous since it can take place as fast as the HID device and the system can communicate. The time it takes for a user to react to windows popping up and identifying the threat can be far too slow to rip the malicious device out before the system is contaminated. Some form of software protection would be very helpful to slow, or even defeat BadUSB attacks. However, the protection must be low profile and unobtrusive in order to prevent user frustration, but at the same time a small amount of user interaction must be present to ensure that the user was intending to plug in an input device instead of a disguised BadUSB.

BadUSB attacks have evolved since their rather basic inception and are capable of any number of attack styles. They can open prompts and pull packages from the internet, establish connections back to an attacker PC, or schedule a number of enduring malicious tasks for the system to undertake daily. The complexity of the BadUSB devices has increased to the point that some BadUSB devices are capable of functioning as an effective remote desktop connection to the victim PC with mouse and keyboard input and even transmit video output (Lu et al. 2021).

Currently the main ways to combat BadUSB style attacks are a mixture of hardware-based methods and speed detection software. The most effective way of combating BadUSB attacks and ensuring that they can never happen is to hardware disable the USB ports outside the one or two required to use the computer. But this is unrealistic in the best cases and a foolish implementation the rest of the time. Strong BadUSB protections using a hardware middleman were developed by Griscioli et al. but posed the problem of additional monetary costs as well as potential hardware incompatibilities when applied across a diverse spectrum of machines and headers (2016). Currently software-based protection methods fall into a few different categories, type speed, whitelisting, and lockout. Typing speed is a slightly weaker approach due to BadUSB attacks being able to adjust how fast input is being made but is still functional if an attacker does not take this into account. Whitelisting allowed devices and querying to ensure the attached item is a valid device is effective but poses risks of mislabeling valid devices. Lockout simply locks the screen once a new HID device is attached but requires accepting the risk that the attacker may know a valid password or credentials.

Enter the BadUSB Challenge System or BUCS. BUCS is a small script designed to run when a user plugs in a new HID device that will serve a CAPTCHA style verification the user will complete proving it was an intended action. If instead the device was malicious BUCS appearing mid BadUSB script can break typical BadUSB attacks since the system makes an unexpected action most BadUSBs are not able to work around. BUCS appearing and potentially throwing a warning when a user did not plug in an input device also serve as a clear signal to less technical users that something unusual is afoot and some form of reaction is most likely required. In this paper we will describe the USB system and scope of BadUSB attacks, the steps we took to design and test our challenge system, our testing scripts and methodology, the other related work around BadUSB protections, and where BUCS or a similarly designed system should move from here.

# **2. RELATED WORK**

The two papers that served as the initial foundation for the idea that became BUCS were an untitled 2019 Availability REliability and Security (ARES) conference article as well as a study on captchas by Hussein & Kaya discussing the efficacy of various types of captchas against automated attacks (2016). The ARES conference paper provided the strongest base for a BadUSB protection system since it leveraged a rather naive approach of blocking and locking where the reaction to a new device was to lock the screen and block input from the device until the user unlocked the system and addressed a notification about the device and device type. It was a functional approach that succeeded against all types of USB attack that they tested it with but lacked refinement and didn’t take into consideration some additional types of automated attack that weren’t leveraging a BadUSB as a way to intrude on an organization or the attacker simply knowing the credentials to the victim computer already. The idea of this lock and block approach connected to a captcha check to resist automated attacks was the initial design idea for BUCS.

We did also take into consideration other types of BadUSB software protections beyond the ARES approach, but we found some flaws that lead us to believe that the simple lock and block methodology would perform better for blocking automated typing attacks than something more involved. El Bouanani’s whitelisting approach was the most interesting since they created their own fingerprinting method due to weaknesses in other identifying methods like serial number tracking that could lead to spoofing attacks (El Bouanani & ACM-DL, 2018). Chekole and Guo built a much more robust whitelisting utility of their own to protect V2X systems from BadUSB attacks. It incorporated over the air policy updates and upgrades and provided perfect USB security without degrading service when it was tested. Their work focused on BadUSB deployed malware instead of automated typing attacks, but it was an impressive undertaking that they implemented to much success with their DARUD toolkit.

The other type of BadUSB protection we saw implemented successfully was the hardware protections. Griscoli et al’s USBCheckIn device forced human device interaction when a HID device was plugged in making the user confirm they had plugged in an input device (2016). This approach worked and did not interfere with normal use after the device was confirmed. The downfall to the hardware approach to this problem is the potential to introduce complexity with hardware compatibility since it does require compatibility with the USB headers on all of the machines it would be utilized on. Additionally, there is the added cost to purchasing all of the hardware and rolling it out if a hardware approach was used in a large organization. This made the decision that BUCS should be a software approach to allow for quick rollout in a large organization without the added costs of hardware.

# **3. BACKGROUND**

The beginnings of this project stemmed from a problem discovered in 2014 that was put on display at Defcon called BadUSB.​​(What Is Badusb | How to Protect against Badusb Attacks - Manageengine Device Control Plus, n.d.) This problem was with Human Interface Devices (HIDs) and the way they interacted with computers. HIDs are more commonly known as devices such as keyboards and mice, but can also be things such as joysticks, gamepads, and other devices that take input from users. (mhopkins-msft, n.d.)The main flaw in these devices is not within the device itself, but in how they are handled by the computer. In all common operating systems, HIDs are innately trusted meaning that they can be plugged in and be used with minimal security steps between. This is a requirement for machines as they need to be able to accept input without interruption when in stages such as BIOS. However, there is minimal reason why a set up computer in an OS stage needs to innately trust HIDs. This is because of attacks known as BadUSBs. These will be described in greater detail in section 4, but they prey on the innate trust that computers put in HIDs.

That is where our solution was born. We decided to go with a software solution as opposed to a hardware solution for ease of use and to have no compatibility issues with hardware. As far as threat level protection, our software is protecting against lower-level threats such as a malicious device being shipped to a legitimate user as opposed to a higher-level threat which would be an intruder inserting a device in the PC themselves. Additionally, we focused specifically on the Windows OS because of its share in the end user market, and the attack surface that BadUSBs cover which is mostly Windows. Our software-based approach will be used to catch input from an HID until it can be authenticated by a living user. We will accomplish this by using the idea of a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart)

The CAPTCHA describes itself perfectly in its name as a test to determine if something is a computer or a human. We decided to use this as it would fulfill our purpose of thwarting a computer, while being simple enough of a task for a user to perform.

# **4. PROPOSED SOLUTION**

The Project we decided to create was Referred to as BUCS (BadUSB Challenge System), and it was designed to act as a CAPTCHA system with the insertion and initialization of USB devices by interceding and require any USB device plugged in to be verified by the user by using a CAPTCHA test. A CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a program that protects websites against bots by generating and grading tests that humans can pass but current computer programs cannot. Standard CAPTCHAs are effective against most standard bots, such as press the button or select the image from the set of images of a given type, but there do exist bots and methods that can bypass them. Some examples are highly trained image recognition Ais, and solving of the CAPTCHAs at farms with dozens or hundreds of people that solve CAPTCHAs for the bot. Our system is unlikely to encounter this due to the limits imposed on a USB hardware device making it so that they are unlikely to use advanced AI or farming out. Our utility that we have made is an event driven script that was supposed to lockdown the keyboard either to only be able to type in the CAPTCHA window or lock the keyboard to only be able to being able to type alphanumeric characters or at least disabling the windows key or any escape/terminal key, but at least for windows we were unable to find an easy method to do it. At which point the user will have to enter a randomly selected word or phrase from a dictionary into the CAPTCHA to validate the keyboard. The script achieves its event driven status via looking for a security log entry in windows event viewer that indicates when a PnP (PlugNPlay) device, primarily a USB, is plugged in at which point it triggers and runs the CAPTCHA tool. The tool is written in Python. We were unable to get the Linux version of the utility working beyond the detection phase. In a later section, we will discuss the results of when we actually implemented it, and some threats we had to avoid.

**5. BADUSB ATTACKS AGAINST WINDOWS**

The threat model that we designed BUCS to work against is what is known as hot plug attacks. These are attacks that work by simply plugging some device, in our case a BadUSB, into the computer and removing it after the task is executed. These attacks are particularly devastating as they can perform any task an attacker could while sitting there typing, but in seconds.

To test BUCS, we decided to create some BadUSBs using a DigiSpark USB development board, and also the commercially available product Rubber Ducky by Hak5. (Hak5, n.d.) We used attacks based on publicly available scripts, which we used to make some of our own. Upon researching this, we discovered that there were two main types of hot plug attacks, reconnaissance, and exploitation. Within the reconnaissance group there is a subgroup of sorts that is data exfiltration, but for our purposes, we combined these two.

These scripts can be found at https://github.com/abladow/BAD-USB/tree/main/BadUSB%20scripts

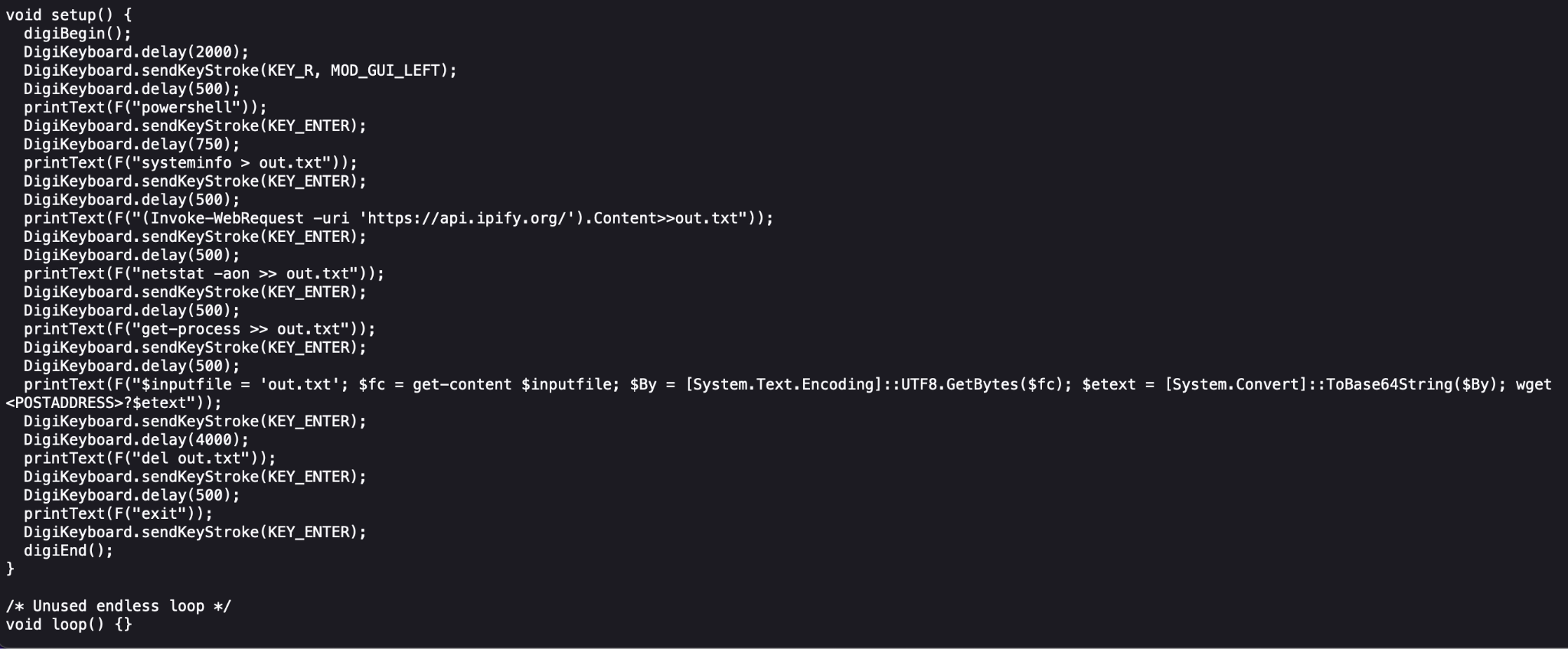
Our first attacks that we crafted were the reconnaissance attacks. These attacks would be used by an attacker to create reports about the target machine. This can be done by running various commands and using a few different exfiltration methods. These can be useful to the attacker to then go on and exploit a machine based on the information found. We created two scripts that we will call ReconA and ReconB.

**ReconA**

****

This script works by creating a file on the local machine, then filling it with information about the host PC using command line tools. It then sends that file using a hard coded email to the attacker. This is done by creating an email object in PowerShell and filling the info automatically.

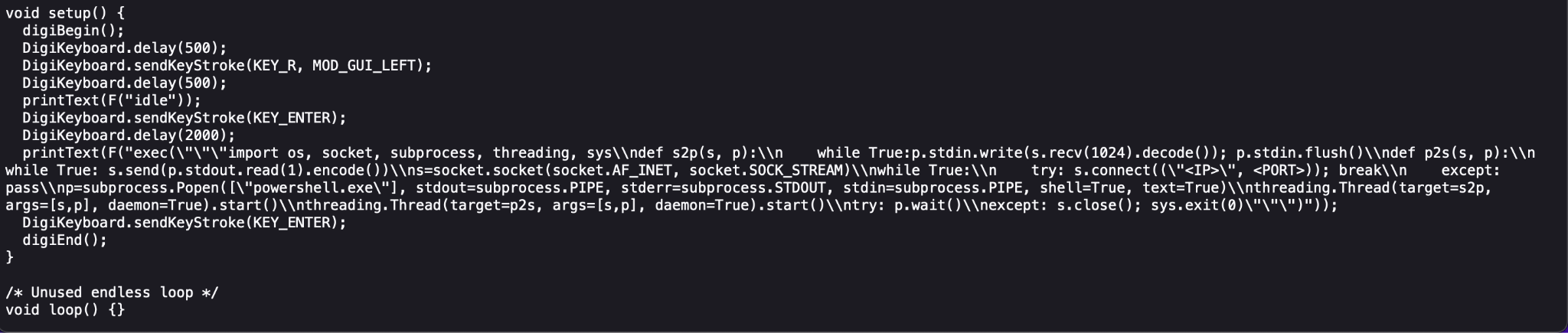
**ReconB**

****

This script works similarly to ReconA as it retrieves information the same way, creating a file and using command line tools to get system information, but the exfiltration is different. It converts the file into a base64 encoded string, then sends it to an address using a wget and appending the base64 string.

Our second attacks that we crafted were exploitation attacks. For these attacks an attacker will try to leverage access to the PC to create some form of exploitation for them to use. An easy method of doing this is using the command line to download a reverse shell, then running it to gain access to the machine. Another method of exploitation that is commonly used is generating a one-line command in python to open a reverse shell. WE created two scripts that we will call ExploitA and ExploitB.

**ExploitA**

****

This script works by opening Idle and running a python one liner that will connect to a hardcoded server using an IP and port, and opens a reverse shell.

**ExploitB**

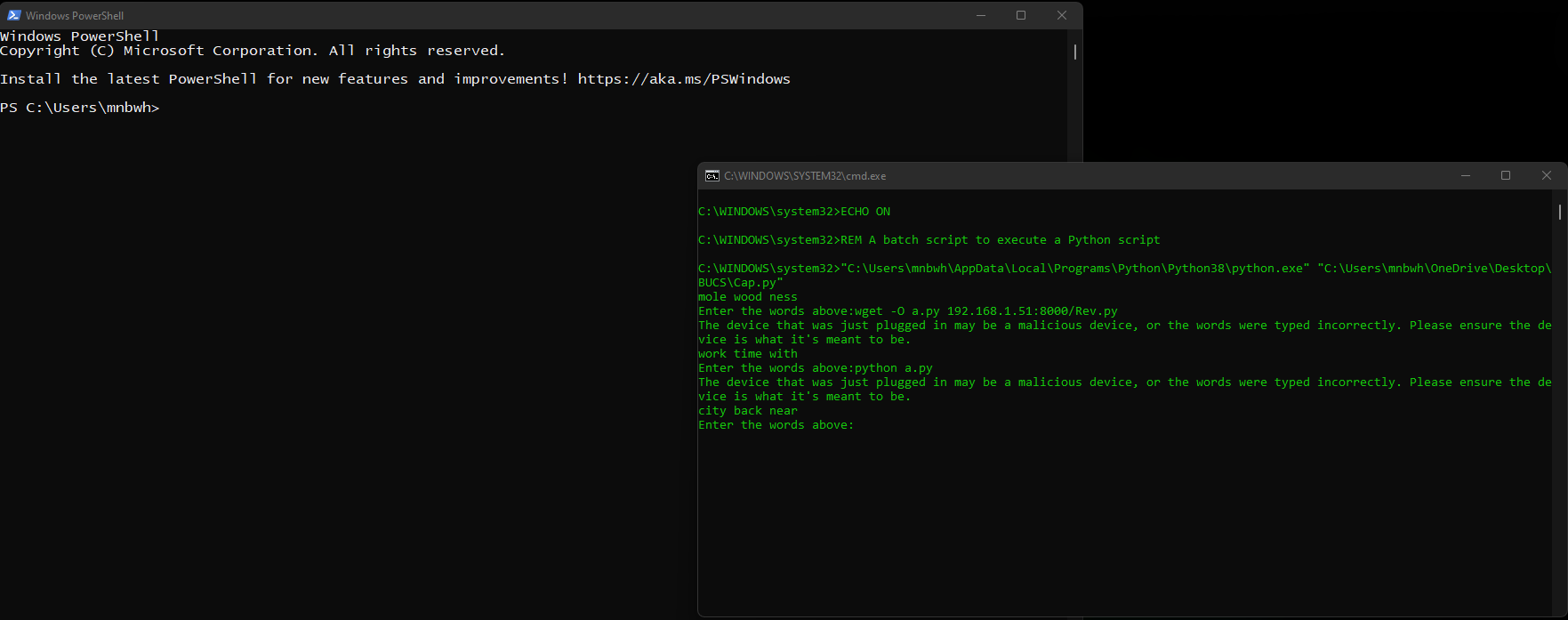
****

This script works by downloading onto the victim machine a custom Python file from the attacking machine, runs it, then attempts to hide the windows so that the victim would not be aware of what is happening.

These two classes of attacks would both need to be effectively stopped by BUCS. To test the program, we first ran the BadUSBs as they were intended to be run against a live Windows 10 PC. This gave us the baseline behavior that we needed to be able to stop.

**6. EVALUATION**

When we ran our tests against BUCS, the results overall were positive. We were able to successfully interject in the sequence of commands the BadUSB was delivering, thus making the attacks ineffective.



The image above, shows BUCS interfering with The ExploitB BadUSB payload. This is the same general idea for all of the payloads, so this is the only one we have included an image for. The payload begins by opening PowerShell, which it was able to do successfully, but when it began delivering the payload, BUCS interjected to capture it. Then all the additional payload was captured for the user to see, but it was not able to be effective at delivering the reverse shell. This was effective for all four payloads on both the DigiSpark and the Rubber Ducky devices.

However, we did run into issues with consistency of devices. Currently, we have a delay set into the task that will allow it to wait a few seconds before it launches BUCS which can be set as needed. This is to interject against the BadUSB while it is delivering its payload. This works effectively against all attacks programmed into a BadUSB, for both the DigiSpark and Rubber Ducky, but it suffers against different hardware. We found that the DigiSpark BadUSB takes approximately six seconds from plug in to begin its delivery. Therefore, a seven to eight second delay works effectively against it. Meanwhile, we found that the Hak5 Rubber Ducky only takes approximately four seconds to run, making it most effective to set a five or six second delay. Additionally, one large drawback to our current system is that if an attacker knows what the delay should be, they can have their script wait to begin until after that delay. The solution to this would be to have better control of the keyboard, but that will be discussed in a later section.

# **7. AN OPEN-ENDED PROBLEM AND DISCUSSION**

After all the testing was completed, we did find some aspects that could be improved upon. The first main improvement to the tools would be a completed Linux implementation that goes beyond the detection phase we were able to get it too. Another improvement would be creating a proper keyboard lockout system. We tried a number of potential solutions such as using registry to disable the windows key, forcing the windows focus on the CAPTCHA, or using the utility power toys to manage the keyboard, but these had problems such as requiring a restart to apply, being designed out of windows for security reasons, or just not have an automatable command line interface. There exist a few methods that we discovered that are capable of still defeating the tool we created, these are that attacks that use implanted devices such as KeyCroc can get past the tool, along with BadUSB devices that will remain plugged into the computer. Our utility has a few advantages and disadvantages when compared to similar solutions. The Advantages are that our utility does not require any form of hardware, which allows it to be very portable and quick to install as long as any device it is loaded on has python full use of USB ports and allows full use of the keyboard, and the utility is easy to use and potentially through text on the prompt self-explanatory. Some of the disadvantages that currently exist are that due to our inability to lock focus on the CAPTCHA and lock down the keyboard to only alphanumeric characters, it is currently possible to navigate around it, and the mere fact that it requires specific user input makes using the computer less convenient when a new USB device needs to be plugged in, and it’s current mitigation and detection methods do not detect all forms of BADUSB.

Additionally, we had a strong desire to implement a graphical interface for BUCS to further simplify ease of use for non-technical users since it aligns well with the goal of BUCS. We have a functioning prototype but at this time there is an issue that takes place when BUCS is triggered by a Windows event. The interface fails to display and instead pulls up a blank cli. With some bug squashing the gui could be rolled out and provide the same functionality as the CLI version of BUCS but we were unable to get it working by the 1.0 release date.

# **8. REFERENCES**

[1] Ducky script quick reference. (n.d.). Retrieved May 13, 2022, from https://docs.hak5.org/usb-rubber-ducky-1/the-ducky-script-language/ducky-script-quick-reference

[2] E. G. Chekole and H. Guo, "DARUD: Detecting and Arresting Rogue USB Devices in the V2X Ecosystem," 2021 IEEE International Conference on

[3]Service Operations and Logistics, and Informatics (SOLI), 2021, pp. 1-6, doi: 10.1109/SOLI54607.2021.9672338. https://ieeexplore-ieee-org.leo.lib.unomaha.edu/stamp/stamp.jsp?tp=&arnumber=9672338&isnumber=9672337

[4] F. Griscioli, M. Pizzonia and M. Sacchetti, "USBCheckIn: Preventing BadUSB attacks by forcing human-device interaction," 2016 14th Annual Conference on Privacy, Security and Trust (PST), 2016, pp. 493-496, doi: 10.1109/PST.2016.7907004. https://ieeexplore-ieee-org.leo.lib.unomaha.edu/stamp/stamp.jsp?tp=&arnumber=7907004&isnumber=7906913

[5] Hak5. (n.d.). Usb rubber ducky. Hak5. Retrieved May 13, 2022, from https://shop.hak5.org/products/usb-rubber-ducky-deluxe

[6] Hussein, Karmand & Kaya, Mehmet. (2016). AN EVALUATION OF DIFFERENT TYPES OF CAPTCHA: EFFECTIVENESS, USER- FRIENDLINESS, AND LIMITATIONS. 2. 12-19. https://www.researchgate.net/publication/340006275\_AN\_EVALUATION\_OF\_DIFFERENT\_TYPES\_OF\_CAPTCHA\_EFFECTIVENESS\_USER-\_FRIENDLINESS\_AND\_LIMITATIONS

[7] mhopkins-msft. (n.d.). Introduction to human interface devices (Hid)—Windows drivers. Retrieved May 13, 2022, from https://docs.microsoft.com/en-us/windows-hardware/drivers/hid/

[8] Proceedings of the 14th international conference on availability, reliability and security. (2019). https://dl.acm.org/action/showBook?doi=10.1145/3339252

[9] What is badusb | how to protect against badusb attacks—Manageengine device control plus. (n.d.). Retrieved May 13, 2022, from https://www.manageengine.com/device-control/badusb.html

**9. CONTRIBUTIONS**

Alex Bladow: Proposed Solution, Open Ended Problem and Discussion

Jensen Miller: Introduction, Related Work

Mitchell White: Sections: Abstract, Background, BadUSB Attacks Against Windows, Evaluation